ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΙΓΑΙΟΥ

ΔΙΔΑΚΤΟΡΙΚΗ ΔΙΑΤΡΙΒΗ

# Το Πρωτόκολλο DNS ως Πολυλειτουργικός Φορέας Επίθεσης

*Συγγραφέας:*
Μάριος Αναγνωστόπουλος
*Επιβλέπων:*
Αναπληρωτής Καθ. Γεώργιος Καμπουράκης

*Διατριβή*
*για την απόκτηση Διδακτορικού Διπλώματος του*

Εργαστηρίου Ασφάλειας Πληροφοριακών και Επικοινωνιακών Συστημάτων
Τμήματος Μηχανικών Πληροφοριακών και Επικοινωνιακών Συστημάτων

Σεπτέμβριος, 2016

University of the Aegean

Doctoral Thesis

# DNS as a multipurpose attack vector

*Author:*
Marios Anagnostopoulos
*Supervisor:*
Associate Prof. Georgios Kambourakis

*A thesis submitted in fulfilment of the requirements*
*for the degree of Doctor of Philosophy*

*at the*

Laboratory of Information and Communication Systems Security
Department of Information and Communication Systems Engineering

September, 2016

# Declaration of Authorship

I, Marios Anagnostopoulos, declare that this thesis entitled, "DNS as a multipurpose attack vector" and the work presented in it are my own. I confirm that:

- This work was done wholly while in candidature for a research degree at this University.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date: September 28, 2016

# Advising Committee of this Doctoral Thesis:

---

Georgios Kambourakis, Supervisor
Department of Information and Communication
Systems Engineering

---

Panagiotis Rizomiliotis, Advisor
Department of Information and Communication
Systems Engineering

---

Elisavet Konstantinou, Advisor
Department of Information and Communication
Systems Engineering

---

University of the Aegean, Greece
2016

# Approved by the Examining Committee:

---

Stefanos Gritzalis
Professor, University of the Aegean, Greece

---

Vasilios Katos
Professor, Bournemouth University, UK

---

Georgios Kambourakis
Associate Professor, University of the Aegean, Greece

---

Panagiotis Rizomiliotis
Assistant Professor, University of the Aegean, Greece

---

Elisavet Konstantinou
Assistant Professor, University of the Aegean, Greece

---

Konstantinos Kolias
Assistant Professor, George Mason University, USA

---

Dimitrios Geneitakis
Researcher, Joint Research Centre - European Commission, Italy

---

University of the Aegean, Greece
2016

“Οἱ καιροὶ οὐ μενετοί.’’
(Θουκυδίδης 455-399 π.Χ.)

# *Abstract*

Few will argue that DNS security is a complex, multi-faceted, and of growing concern research topic. This is simply because virtually any protocol or service in the Internet depends its operation on the DNS service. Putting it another way, attacks on DNS can paralyze the network infrastructure of entire countries or even continents. Even worse, the original DNS design was concentrated on availability not security, and thus up to 1997 (and practically prior to 2008) the protocol did not afford any ilk of security protection, not even origin authentication of the offered DNS data. In this context, the primary aim of this PhD thesis is to alert the community by the investigation of novel ways of DNS exploitation, and thus our work can be mainly classified under the umbrella of offensive security. Specifically, we show that DNS can be exploited as a multipurpose attack vector that may severely threaten the integrity, authenticity, confidentiality, and availability of the offered resources in the cyberspace.

Nowadays, DNS security inefficiencies have been addressed in practice by DNSSEC, and at least up to now, only on a limited scale by DNSCurve. Both these mechanisms utilize public key cryptography with the aim of extending the core DNS protocol. Therefore, one of the contribution of this thesis is to provide a comprehensive and constructive side-by-side comparison among the aforementioned security mechanisms. This is anticipated to greatly aid the defenders to decide which mechanism best suits to each particular deployment.

Furthermore, there are overwhelming evidences that DNS is frequently abused by cyber crooks in Denial of Service (DoS) type of attacks. This is because that - even the typical - DNS records can greatly amplify the attack effect, meaning vastly augment the volume of network traffic reflected and destined toward victims. This amplification effect is foreseen to be far more devastating in the case of DNSSEC records, which normally are considerably bigger. In this mindset, an additional contribution of the thesis at hand is the investigation and assessment (in terms of attack amplification factor) of novel types of DNSSEC-powered DoS kind of attacks. The role of DNS forwarders as reflectors in such attack incidents is studied as well. Moreover, we examine some novel options about which publicly available resources (in terms of DNS servers) could be particular fruitful for the attacker to include them to their arsenal. In this direction, the potential of entangling the infrastructure of upper DNS hierarchy as both amplifiers and reflectors is thoroughly investigated. Regarding this point, the main advantage of our research compared with the standard type of DNS amplification attack is that we demonstrate that even a naive attacker is capable of executing a fruitful attack by simply exploiting the great amount of DNS machines existing out there.

This thesis also deals with the exploitation of DNS protocol by bot herders with the purpose of building hidden Communication and Control (C&C) channels for their botnets. In this respect, we delve into the so called DNS- and IP-fluxing techniques, and propose and evaluate three novel botnet architectures which solely rely on DNS to deliver the botnet's C&C infrastructure. Given the mushrooming of smart mobile devices, the proposed architectures utilize not only mixed structures consisting of both mobile and desktop bot agents, but more importantly, structures that are purely mobile. This aspect of our work also includes the evaluation of the robustness of the proposed botnet formations.

Finally, besides the contributions devoted to legacy DNS attacks, we investigate the potential of DNS as an attack vector to evade user's privacy by means of harvesting private sensitive information from, say, smartphone owners. To this end, we design and implement a privacy-invasive mobile application (spyware) able to manipulate the DNS service running on devices based on the Apple's iOS platform. The spyware is capable of acting as a man-in-the-middle to the tethering and intelligent personal assistant (such as Siri and Goggle now) services present in virtually every modern mobile platform. In this case, the aim of the spyware coder is that of redirecting all users connected via the device to a malicious website in order to phish user's credentials, harvest sensitive personal information, and so forth.

# *Greek Abstract*

Αδιαμφισβήτητα, τα ζητήματα ασφαλείας για την υπηρεσία ονοματοδοσίας χώρου του Διαδικτύου (DNS) αποτελούν ένα πολύπλοκο, πολυδιάστατο και ιδιαίτερης βαρύτητας πεδίο έρευνας. Ουσιαστικά, κάθε πρωτόκολλο ή υπηρεσία που παρέχεται μέσω του Διαδικτύου βασίζει την απρόσκοπτη λειτουργία του στην υπηρεσία DNS. Συνεπώς, κάθε τύπος επίθεσης που στοχεύει το DNS ενδέχεται να επιφέρει σοβαρές επιπτώσεις στη δικτυακή υποδομή οργανισμών ή ακόμα και ολόκληρων κρατών. Δυστυχώς, οι αρχικές εκδόσεις του συγκεκριμένου πρωτοκόλλου εστίαζαν στη διαθεσιμότητα της υπηρεσίας παρά στην προστασία της από πιθανές απειλές. Έτσι, σχεδόν μέχρι το 1997, αλλά πρακτικά όχι νωρίτερα από το 2008, το πρωτόκολλο DNS δεν υποστήριζε κάποιο μηχανισμό προστασίας, ούτε καν αυτόν της διασφάλισης της αυθεντικότητας προέλευσης των παρεχόμενων δεδομένων. Σε αυτό το πλαίσιο, η παρούσα διδακτορική διατριβή έχει ως βασικό στόχο να ενημερώσει και να θέσει σε επιφυλακή την επιστημονική κοινότητα μέσω της διερεύνησης και επισήμανσης νέων μεθόδων κακόβουλης εκμετάλλευσης του DNS. Έτσι, μπορούμε να πούμε ότι η εν λόγω διατριβή επικεντρώνεται στην επιθετική ασφάλεια, δηλ., εστιάζει στον επιτιθέμενο πάρα στον αμυνόμενο. Με άλλα λόγια, η διατριβή φιλοδοξεί να καταδείξει ότι το πρωτόκολλο DNS δύναται να χρησιμοποιηθεί ως ένας πολύπλευρος φορέας επίθεσης (attack vector) με σκοπό την παραβίαση της ακεραιότητας, αυθεντικότητας, εμπιστευτικότητας και διαθεσιμότητας των προσφερόμενων πόρων στον κυβερνοχώρο.

Σήμερα, οι ανεπάρκειες του DNS θεραπεύονται από δύο κρυπτογραφικούς μηχανισμούς· στην πράξη με την επέκταση ασφαλείας του DNSSEC και (μέχρι στιγμής σε μικρή κλίμακα) το DNSCurve. Αμφότεροι οι μηχανισμοί αυτοί χρησιμοποιούν κρυπτογραφία δημοσίου κλειδιού με σκοπό την προστασία των μηνυμάτων του βασικού πρωτοκόλλου DNS. Ως εκ τούτου, μία από τις συνεισφορές της παρούσας διατριβής είναι η διεξαγωγή μιας συγκριτικής και εποικοδομητικής παράθεσης των δύο προαναφερθέντων μηχανισμών ασφαλείας. Μια τέτοια αντιπαραβολή είναι βέβαιο ότι βοηθάει σημαντικά τους αμυνόμενους στο να επιλέξουν ποιος μηχανισμός είναι καταλληλότερος για κάθε περίπτωση.

Επιπλέον, υπάρχουν σημαντικές αποδείξεις ότι το πρωτόκολλο DNS χρησιμοποιείται κακόβουλα από τους κυβερνο-εγκληματίες προκειμένου να διεξάγουν επιθέσεις άρνησης υπηρεσίας (DoS). Συγκεκριμένα, τα περιστατικά αυτά ανήκουν σε μια ειδική κατηγορία επιθέσεων που ονομάζονται επιθέσεις ενίσχυσης μέσω DNS (DNS amplification attack). Κάτι τέτοιο είναι εφικτό, διότι ακόμα και μια τυπική DNS απόκριση (response) ενδέχεται να ενισχύσει σημαντικά την δικτυακή κίνηση που καταφτάνει στην πλευρά του θύματος. Ουσιαστικά, το πρωτόκολλο ενισχύει (amplify) σε μεγάλο βαθμό τον όγκο της δικτυακής κίνησης που ανακλάται (reflect) και κατευθύνεται μέσω της υποδομής DNS εναντίον του στόχου. Το εν λόγω φαινόμενο της ενίσχυσης αναμένεται να είναι ακόμα ισχυρότερο και συνεπώς να επιφέρει καταστροφικότερες επιπτώσεις στην περίπτωση χρήσης εγγραφών

(records) DNSSEC, οι οποίες είναι μεγαλύτερες σε μέγεθος από αυτές του βασικού πρωτοκόλλου. Σε αυτήν την κατεύθυνση, η παρούσα διατριβή διεξάγει μια λεπτομερή μελέτη νέων τύπων DoS επιθέσεων που βασίζονται αποκλειστικά σε δεδομένα DNSSEC. Η αποτίμηση της επίπτωσης των προτεινόμενων επιθέσεων επιτυγχάνεται με τον υπολογισμό του συντελεστή ενίσχυσης (amplification factor) της επίθεσης. Συμπληρωματικά, διερευνούμε τις διαφορετικές μεθόδους εκμετάλλευσης των προωθητών DNS (DNS forwarders) με σκοπό την αποτελεσματικότερη ανάκλαση της επιτιθέμενης δικτυακής κίνησης. Επιπλέον, η διατριβή απαντάει στο ερώτημα σχετικά με το ποιοι δημόσια διαθέσιμοι εξυπηρετητές DNS θα μπορούσαν να είναι προσοδοφόροι για τους σκοπούς του επιτιθέμενου έτσι ώστε να τους συμπεριλάβει στο οπλοστάσιο του. Για το σκοπό αυτό, εξετάζεται εκτενώς το αποτέλεσμα της εμπλοκής των εξυπηρετητών της ανώτερης ιεραρχίας της υποδομής DNS σε περιστατικά επιθέσεων ενίσχυσης. Συνεπώς, το κύριο πλεονέκτημα της ερευνητικής μας προσπάθειας σε σύγκριση με τις τυπικές επιθέσεις ενίσχυσης μέσω DNS έτσι όπως αναφέρονται στην σχετική βιβλιογραφία, είναι ότι καταδεικνύουμε ότι ακόμα και ένας ανεπαρκής επιτιθέμενος με ελάχιστα υπολογιστικά μέσα στην διάθεση του, είναι ικανός να διεξάγει μια επιτυχημένη επίθεση DoS εκμεταλλευόμενος τις υπάρχουσες (δημόσιες) υποδομές.

Επιπροσθέτως, η έρευνά μας εστιάζει στην εκμετάλλευση του πρωτοκόλλου DNS από τους διαχειριστές (bot herders) δικτύων υπολογιστών-ρομπότ (botnets). Συγκεκριμένα, με σκοπό τη μετάδοση των εντολών του διαχειριστή στα μέλη του δικτύου (bots), είναι απαραίτητη η υλοποίηση συγκεκαλυμμένων καναλιών επικοινωνίας (Command & Control-C&C channels). Τέτοια κανάλια επικοινωνίας κατά κόρον αξιοποιούν τις τεχνικές DNS- και IP-Flux. Έτσι, στο πλαίσιο της παρούσας διατριβής, σχεδιάζουμε και αξιολογούμε τρείς νέες βοτνετ αρχιτεκτονικές, οι οποίες βασίζονται στο πρωτόκολλο DNS για τη δημιουργία της υποδομής C&C. Δεδομένης της ευρείας διάδοσης των έξυπνων κινητών συσκευών, οι προτεινόμενες αρχιτεκτονικές αξιοποιούν τόσο μικτή δομή (αποτελούμενη από κινητά και σταθερά bots) όσο και πιο εξειδικευμένη, αποτελούμενη αποκλειστικά από κινητές συσκευές. Εκτός των άλλων, η συγκεκριμένη ερευνητική συνιστώσα της διατριβής περιλαμβάνει μια αναλυτική αποτίμηση της ευρωστίας των προαναφερθέντων αρχιτεκτονικών botnet.

Εκτός των τυπικών μεθόδων εκμετάλλευσης του πρωτόκολλου DNS από τους επιτιθέμενους, η παρούσα διατριβή εστιάζει στη διερεύνηση της δυνατότητας αξιοποίησης του πρωτοκόλλου για την παραβίαση της ιδιωτικότητας των χρηστών. Πιο συγκεκριμένα, εξετάζουμε την περίπτωση υποκλοπής ευαίσθητων προσωπικών δεδομένων από τους χρήστες έξυπνων κινητών συσκευών. Για το σκοπό αυτό, σχεδιάζουμε και υλοποιούμε μια εφαρμογή κατασκοπίας (spyware) για κινητές συσκευές που έχει ως στόχο τη χειραγώγηση της υπηρεσίας DNS που παρέχεται από κινητές πλατφόρμες και συγκεκριμένα την iOS της εταιρίας Apple. Η εφαρμογή δρα ως ενδιάμεσος (man-in-the-middle) στην υπηρεσία διαμοιρασμού του ασύρματου δικτύου (tethering) ή/και σε αυτή του έξυπνου προσωπικού βοηθού (Siri). Αφού μολύνει τη συσκευή, η εφαρμογή ανακατευθύνει τον ανυποψίαστο χρήστη σε κακόβουλες

ιστοσελίδες ώστε να συλλέξει συνθηματικά, στοιχεία λογαριασμών, προσωπικά δεδομένα, κ.ά.

# *Acknowledgements*

Φτάνοντας στο τέρμα αυτής της πολύχρονης πορείας και κατ' επέκταση στην ολοκλήρωση των φοιτητικών μου χρόνων, θεωρώ καθήκον και υποχρέωση μου να ευχαριστήσω όλους αυτούς που συνέβαλαν με τον έναν ή τον άλλον τρόπο στην επιτυχή περάτωση αυτής της διατριβής.

Πρώτα και κύρια θα ήθελα να εκφράσω τις βαθύτερες ευχαριστίες μου στον επιβλέποντα Αναπληρωτή Καθηγητή κ. Γεώργιο Καμπουράκη, που δεν υπήρξε απλά ένας επιβλέπων αλλά και ένας καλός φίλος και συνεργάτης στην όλη προσπάθειά μου. Επιπλέον θέλω να ευχαριστήσω και τα υπόλοιπα μέλη της τριμελούς και επταμελούς επιτροπής για τον χρόνο που αφιέρωσαν για την ολοκλήρωση αυτής της εργασίας. Φυσικά δεν πρέπει να παραλείψω να ευχαριστήσω τον Καθηγητή κ. Στέφανο Γκρίτζαλη για την αμέριστη αρωγή του στα όποια εμπόδια και δυσκολίες παρουσιάστηκαν όλα αυτά τα χρόνια. Σε κάθε περίσταση επενέβαινε για να βοηθήσει στην αντιμετώπιση τους και συνέβαλε στην απρόσκοπτη συνέχιση της ερευνητικής μου πορείας.

Επιπλέον θέλω να ευχαριστήσω τους Παναγιώτη Κόπανο και Γεώργιο Λουλουδάκη, που ως προπτυχιακοί και αργότερα ως μεταπτυχιακοί φοιτητές με βοήθησαν να αποκτήσω αυτοπεποίθηση στην εκτέλεση και την υλοποίηση πειραμάτων. Η συνεργασία μας, μου έδειξε ότι καμιά ιδέα δεν είναι αδύνατον να υλοποιηθεί και κάθε πείραμα καλά σχεδιασμένο και καθορισμένο θα φέρει τα επιθυμητά αποτελέσματα. Καλή επιτυχία στην ζωή σας, παιδιά!

Ακόμα θέλω να ευχαριστήσω τους καλούς φίλους και συνεργάτες, Ζήση Τσιάτσικα και Δημήτριο Παπαμαρτζιβάνο, για την εποικοδομητική ανταλλαγή ιδεών και απόψεων οι οποίες συνέβαλαν στην διευκόλυνση της προσπάθειας μας. Με το καλό να ολοκληρώσετε το διδακτορικό σας και να επιτύχετε τους στόχους σας!

Φυσικά θέλω να εκφράσω τις θερμότερες ευχαριστίες και ευχές μου στους αδελφικούς φίλους Γιάννη Αχιλλιά, Δημήτρη Παπαγιαννούλη και Παύλο Χατζηδημητρίου, που από τα πρώτα χρόνια της φοιτητικής μας ζωής έως τώρα στα χρόνια της ωρίμανσης και υλοποίησης των ονείρων μας, αποτελούν τον βασικό πυλώνα στήριξης και συμπαράστασης και ως φίλοι συνεχίζουμε όλα αυτά τα χρόνια να συμβουλεύουμε αλλήλους και να στηριζόμαστε στις δύσκολες αλλά και να μοιραζόμαστε τις ευχάριστες στιγμές. Γιάννη, ήσουν και παράμεινες το πρότυπο και η ψυχή της παρέας μας! Είναι ιδιαίτερη τιμή που σε γνώρισα και απολαμβάνω την φιλία σου...

Ιδιαίτερες ευχαριστίες θέλω να εκφράσω στην συνοδοιπόρο και επ-Αναστάτρια της ζωής μου, που όλα αυτά τα χρόνια μου συμπαραστάθηκε και με βοήθησε να μη καταβάλλομαι από τις απογοητεύσεις και τα εμπόδια, αλλά με θάρρος και χαμόγελο να συνεχίζω προς τον επιδιωκόμενο σκοπό.

Δεν πρέπει να παραβλέψουμε και όλες αυτές τις αθόρυβες ψυχούλες, που σαν αόρατοι άγγελοι στέκονται πάνω από τον ώμο μας, και με τον δικό τους μοναδικό τρόπο φωτίζουν τα βήματα μας και δείχνουν τον δρόμο προς την ψυχική ανάταση και την ολοκλήρωση μας ως ανθρώπινη οντότητα.

Τελευταία μα πιο σημαντικά θέλω να ευχαριστήσω τους γονείς μου, για την ψυχική και ηθική συμπαράσταση τους όλα αυτά τα χρόνια. Τους ΕΥΧΑΡΙΣΤΩ που μου παρείχαν όλα τα απαραίτητα υλικά εφόδια για την επιβίωση και επένδυσαν στην μόρφωση μου. Κυρίως όμως τους ΕΥΓΝΩΜΟΝΩ που σαν γνήσιοι Εκπαιδευτικοί μου παρείχαν όλα τα ηθικά εφόδια και την αγάπη για την γνώση και την μόρφωση, προκειμένου να γίνω ένας σωστός και χρήσιμος άνθρωπος. Ευχαριστώ και τον αδελφό μου τον Στάθη, που με το πείσμα του, την επιμονή του, το θάρρος και τον –ας μου επιτραπεί η έκφραση- ‘‘τσαμπουκά’’ του, μου έδειξε ότι δεν πρέπει να απογοητευόμαστε στις πρώτες δυσκολίες, αλλά πρέπει να προσπαθούμε και να στοχεύουμε στο καλύτερο, καμία φορά δε το καλύτερο δεν είναι αυτό που εμείς επιδιώκουμε αλλά αυτό που κάποιος ‘‘Άλλος’’ μας επιφυλάσσει.

♫ *Mes chers parents je pars*
*Je vous aime mais je pars*
*Vous n'aurez plus d'enfants*
*Ce soir*
*Je ne m'enfuis pas je vole*
*Comprenez bien je vole*
*Sans fumée sans alcool*
*Je vole, je vole* ♫
*(Louane-Je vole)*

*To those who enlighten the path of our life,*
*To my parents*

*Σε αυτούς που φωτίζουν το μονοπάτι της ζωής μου,*
*Στους γονείς μου*

# Contents

**Declaration of Authorship** i

**Advising Committee of this Doctoral Thesis** ii

**Approved by the Examining Committee** iii

**Abstract** v

**Greek Abstract** vii

**Acknowledgements** x

**List of Figures** xvii

**List of Tables** xix

**Abbreviations** xx

**1 Introduction** 1
- 1.1 Motivation and Objectives . . . 4
- 1.2 Contributions . . . 6
- 1.3 Thesis Structure . . . 9

**2 Background** 13
- 2.1 DNS Service . . . 13
  - 2.1.1 Introduction . . . 13
  - 2.1.2 DNS Domains . . . 14
  - 2.1.3 DNS zones . . . 15
  - 2.1.4 DNS Reverse Mapping . . . 15
  - 2.1.5 DNS Operation . . . 16
  - 2.1.6 DNS Queries . . . 17
  - 2.1.7 Entities in a DNS transaction . . . 17
    - 2.1.7.1 Resolver . . . 18
    - 2.1.7.2 Name Server . . . 18
      - Primary Authoritative Name Server . . . 18

Secondary Authoritative Name Server . . . 18
Authoritative-only Name Server . . . 19
Recursive Name Server . . . 19
Forwarding Name Server . . . 20
2.1.8 Format of Zone File . . . 21
2.1.9 Resource Record Types . . . 22
2.1.9.1 SOA Record . . . 22
2.1.9.2 NS Record . . . 24
2.1.9.3 A and AAAA Record . . . 25
2.1.9.4 CNAME Record . . . 25
2.1.9.5 TXT Record . . . 25
2.1.9.6 MX Record . . . 26
2.1.9.7 PTR Record . . . 26
2.1.9.8 CSYNC Record . . . 27
2.1.10 DNS Message Format . . . 27
2.1.11 Zone Update . . . 29
2.1.11.1 Full Zone Transfer . . . 29
2.1.11.2 Incremental Zone Transfer . . . 30
2.1.11.3 Notify . . . 30
2.1.11.4 Dynamic Update . . . 30
2.1.11.5 Wildcards . . . 31
2.1.12 Resolution of a domain name . . . 31
2.2 Legacy DNS Attacks . . . 33
2.2.1 Attack Surface specific to Integrity of DNS transactions . . . 33
2.2.2 Exploiting DNS infrastructure for DDoS . . . 35

**3 DNS Cache Poisoning 36**
3.1 Introduction . . . 36
3.1.1 Target of Cache Poisoning . . . 37
3.1.2 Steps for a Cache Poisoning Attack . . . 38
3.1.3 Kaminsky-Style Poisoning Attack . . . 42
3.2 Poisoning Antidotes . . . 44
3.2.1 Interim Solutions . . . 45
3.2.1.1 Transaction ID Randomization . . . 46
3.2.1.2 Source Port Randomization . . . 46
3.2.1.3 0x20-Bit Encoding . . . 47
3.2.1.4 WSEC DNS . . . 49
3.2.1.5 Multiple Queries . . . 51
3.2.2 Cryptographic Solutions . . . 51
3.2.3 DNSSEC . . . 52
3.2.3.1 Security Operations . . . 53
3.2.3.2 DNSSEC-related RR types . . . 55
DNSKEY Record . . . 55
RRSIG Record . . . 56
NSEC/NSEC3 Record . . . 58
DS Record . . . 62
Cryptographic Algorithms for DNSSEC . . . 64

3.2.3.3 DNSSEC-related header flags . . . 65
3.2.3.4 DNSSEC in action . . . 67
3.2.3.5 Trust Anchor . . . 69
3.2.3.6 DNSSEC Lookaside Validation . . . 69
3.2.3.7 Challenges of DNSSEC Deployment . . . 71
3.2.4 DNSCurve . . . 72
3.2.4.1 Security Operations . . . 73
3.2.4.2 Publication of Public Key . . . 74
3.2.4.3 DNSCurve message format . . . 75
3.2.4.4 DNSCurve operation . . . 75
3.2.4.5 Trust anchor . . . 77
3.2.4.6 Elliptic Curve Cryptography . . . 78
3.2.5 DNSSEC vs. DNSCurve: A side-by-side Comparison . . . 78
3.2.5.1 Cryptography . . . 78
3.2.5.2 Integrity and Origin Authentication . . . 79
3.2.5.3 Confidentiality . . . 81
3.2.5.4 Authenticated Denial of Existence . . . 83
3.2.5.5 Amplification Attacks . . . 84
3.2.5.6 Modification of DNS Infrastructure . . . 85
3.2.5.7 Zone Administration . . . 85
3.2.5.8 Key Management . . . 86
3.2.5.9 Performance . . . 88
3.2.5.10 Conclusion . . . 93
3.2.6 DNS over DTLS . . . 94
3.2.7 DNS over TLS . . . 95

**4 Novel DNS amplification attack vectors 103**
4.1 DNS Amplification . . . 103
4.1.1 Amplifiers and Reflectors . . . 105
4.1.2 Victims of DNS amplification attack . . . 109
4.2 Going one step further: Obfuscating DNS amplification . . . 110
4.2.1 Attack Scenario . . . 111
4.2.2 Results . . . 113
4.2.3 Discussion . . . 119
4.3 Authoritative TLD nameserver-powered DNS amplification . . . 121
4.3.1 Methodology and Results . . . 124
4.3.1.1 Types of Queries . . . 125
4.3.1.2 Examining the response size for a single query . . . 125
4.3.1.3 Examing RRL mechanism for positive responses . . . 127
4.3.1.4 Examing RRL mechanism for negative responses . . . 130
4.3.2 Discussion . . . 131
4.4 Countermeasures . . . 133
4.4.1 Proactive Measures . . . 134
4.4.1.1 Lowering the Amplification Factor . . . 134
4.4.1.2 Eliminate Reflection Capabilities . . . 135
4.4.2 Reactive methods . . . 136

**5 DNS-driven botnet C&C architectures** **139**
5.1 Introduction . . . 139
5.2 Botnet Architectures . . . 141
5.3 Life Cycle of a Bot . . . 142
5.4 DNS Fluxing . . . 143
5.5 C&C channels . . . 145
5.6 DNS as C&C Channel . . . 145
5.7 Mobile botnets . . . 147
5.7.1 Benefits and Limitations of Mobile Botnet . . . 149
5.8 New facets of mobile botnets . . . 150
5.8.1 Preliminaries and attack planning . . . 150
5.8.2 Architecture I: A purely mobile botnet . . . 152
5.8.2.1 Architecture II: Mobile Botnet with PC-based proxies . . . 156
5.8.2.2 Architecture III: Exploiting DNS as covert C&C channel 157
5.8.2.3 Other considerations . . . 159
5.8.3 Comparison of architectures and Results . . . 159
5.9 Countermeasures . . . 162
5.9.1 DNS-based Botnet detection . . . 163
5.9.1.1 Detection of DNS fluxing . . . 163
5.9.2 Botnet shutdown operation . . . 166
5.9.2.1 Botnet Sinkholing . . . 166
5.9.2.2 Botnet Infiltration . . . 168

**6 DNS as an attack vector in Mobile Platforms** **183**
6.1 Introduction . . . 183
6.2 Preliminaries . . . 184
6.2.1 mDNS . . . 185
6.2.2 The Tethering and Siri services . . . 186
6.3 Implementation . . . 187
6.3.1 The DNS poisoning malware . . . 187
6.4 Attack Scenarios . . . 191
6.4.1 Scenario I: DNS Hijacking . . . 192
6.4.2 Scenario II: Privacy leak over Siri . . . 192
6.4.2.1 Exposing User's Geographical Location . . . 195
6.4.2.2 Obtaining sensitive information via SMS . . . 195
6.4.2.3 Acquiring user's password . . . 197
6.5 Related Work . . . 197

**7 Conclusions and Future Directions** **201**
7.1 Thesis Contributions . . . 202
7.2 Future Directions . . . 204

**Bibliography** **207**

# List of Figures


1.1 Contribution of PhD thesis with the relative security services . . . . . 12

2.1 DNS tree structure . . . . . 14
2.2 DNS Reverse Tree . . . . . 16
2.3 DNS transaction . . . . . 32
2.4 Attack Surface specific to Integrity of DNS transactions . . . . . 33

3.1 DNS poisoning attack . . . . . 39
3.2 Time window for DNS cache poisoning attack . . . . . 96
3.3 Algorithm of DNS-0x20bit encoding . . . . . 97
3.4 WSEC DNS query process . . . . . 98
3.5 Typical example of DNSSEC chain of trust . . . . . 99
3.6 Example of a DNSSEC transaction . . . . . 100
3.7 Example of a DNSCurve transaction . . . . . 101
3.8 Message Flow for Full DTLS Handshake . . . . . 102

4.1 High-level architecture of a typical DNS amplification attack . . . . . 105
4.2 DNS forwarders discovery process . . . . . 112
4.3 High-level architecture of the attack introduced in this work . . . . . 114
4.4 Progress of Resource Consumption at the victim-side . . . . . 117
4.5 Exploiting ANSs administrating TLD zones . . . . . 123

5.1 Botnet Structure . . . . . 140
5.2 The life cycle of a Bot . . . . . 143
5.3 TCP Flooding Attack . . . . . 169
5.4 Architecture I: Initialization Phase . . . . . 170
5.5 Architecture I: Migration Phase . . . . . 171
5.6 Architecture I: Recovery Phase . . . . . 172
5.7 Architecture I: Amplification Attack . . . . . 173
5.8 Architecture I: TCP Flooding Attack . . . . . 174
5.9 Architecture II: Initialization Phase . . . . . 175
5.10 Architecture II: Migration Phase . . . . . 176
5.11 Architecture II: Recovery Phase . . . . . 177
5.12 Architecture II: Amplification Attack . . . . . 178
5.13 Architecture II: TCP Flooding Attack . . . . . 179
5.14 Architecture III: Initialization Phase . . . . . 179
5.15 Architecture III: Amplification Attack . . . . . 180
5.16 Architecture III: TCP Flooding Attack . . . . . 180
5.17 Inbound traffic in MBps for both attack variations of Architecture I . . . 181

5.18 Architecture III: Network traffic generated due to botnet coordination . . 182

6.1 Malware module . . . . . 188
6.2 The /etc/hosts file after poisoning . . . . . 189
6.3 Source code snippet for disabling/enabling mDNSResponder . . . . . 190
6.4 Network architecture utilized for the attack scenarios . . . . . 191
6.5 Siri protocol flow . . . . . 193
6.6 Basic source code example of a custom plugin . . . . . 195
6.7 Snippet of the plugin responsible to retrieve user's location . . . . . 195
6.8 Log file created when sending an SMS . . . . . 196
6.9 Message flow for acquiring user's password . . . . . 198

# List of Tables


2.1 Common DNS Resource Record Types . . . . . 28
2.2 DNS Message Format . . . . . 28

3.1 Definition of symbols for DNSSEC Resolution Algorithm . . . . . 67
3.2 DNSSEC vs. DNSCurve . . . . . 92

4.1 Percentages of open forwarders per country in regards to the size of response they return . . . . . 115
4.2 Effects on target proportional to the power and number of attacking nodes per scenario . . . . . 118
4.3 Demographics of the TLD ANSs . . . . . 126
4.4 Amplification factor for a single query with EDNS0 buffer size 8,192 . . . 127
4.5 Percentage of authoritative NSs for positive responses (ver. 1) . . . . . 129
4.6 Percentage of authoritative NSs for positive responses (ver. 2) . . . . . 129
4.7 Percentage of truncated positive responses . . . . . 130
4.8 Percentage of authoritative NSs of negative responses (ver. 1) . . . . . 131
4.9 Percentage of authoritative NSs of negative responses (ver. 2) . . . . . 131
4.10 Percentage of truncated negative responses . . . . . 131

5.1 Inbound traffic in MBps proportional to the number of attacking bots per architecture . . . . . 162

7.1 Overall PhD Thesis Contribution . . . . . 204

# Abbreviations

| | |
|---|---|
| **2-TLD** | Second **T**op-**L**evel **D**omain |
| **ANS** | **A**uthoritative **N**ame **S**erver |
| **AP** | **A**ccess **P**oint |
| **AS** | **A**utonomous **S**ystem |
| **ASN** | **A**utonomous **S**ystem **N**umber |
| **BCP** | **B**est **C**urrent **P**ractice |
| **CA** | **C**ertification **A**uthority |
| **ccTLD** | **c**ountry **c**ode **T**op-**L**evel **D**omain |
| **CDN** | **C**ontent **D**elivery **N**etwork |
| **CVE** | **C**ommon **V**ulnerabilities & **E**xposures |
| **DDNS** | **D**ynamic **D**omain **N**ame **S**ystem |
| **DDoS** | **D**istributed **DoS** |
| **DGA** | **D**omain **G**eneration **A**lgorithm |
| **DHCP** | **D**ynamic **H**ost **C**onfiguration **P**rotocol |
| **DNS** | **D**omain **N**ame **S**ystem |
| **DNSSEC** | **DNS SE**curity Extentions |
| **DoS** | **D**enial of **S**ervice attack |
| **DTLS** | **D**atagram **T**ransport **L**ayer **S**ecurity |
| **EC** | **E**lliptic **C**urve |
| **ECC** | **EC C**ryptography |
| **ESSID** | **E**xtended **S**ervice **S**et **ID** |
| **FQDN** | **F**ully **Q**ualified **D**omain **N**ame |
| **GPS** | **G**lobal **P**ositioning **S**ystem |
| **gTLD** | **g**eneric **T**op-**L**evel **D**omain |
| **HTTP** | **H**ypertext **T**ransfer **P**rotocol |

| | |
|---|---|
| **HTTPS** | **HTTP**ypertext **S**ecure |
| **ICANN** | **I**nternet **C**orporation for **A**ssigned **N**ames & **N**umbers |
| **IDS** | **I**ntrusion **D**etection **S**ystem |
| **IP** | **I**nternet **P**rotocol |
| **IPv4** | **IP v**ersion **4** |
| **IPv6** | **IP v**ersion **6** |
| **ISO** | **I**nternational **O**rganization for **S**tandardization |
| **ISP** | **I**nternet **S**ervice **P**rovider |
| **KSK** | **K**ey **S**inging **K**ey |
| **MAC** | **M**essage **A**uthentication **C**ode |
| **mDNS** | **m**ulticast **DNS** |
| **MitM** | **M**an-**i**n-**t**he-**M**iddle attack |
| **NAT** | **N**etwork **A**dress **T**ranslation |
| **NN** | **N**eural **N**etwork |
| **NONCE** | **N**umber used **Once** |
| **NS** | **N**ame **S**erver |
| **NXDOMAIN** | **N**on-e**x**istent **Domain** |
| **ORDNS** | **O**pen **R**ecursive **NS** |
| **OS** | **O**perating **S**ystem |
| **PAT** | **P**ort **A**dress **T**ranslation |
| **PC** | **P**ersonal **C**omputer |
| **PH** | **Pe**rsonal **H**otspot |
| **PKC** | **P**ublic **K**ey **C**ryptography |
| **PKI** | **P**ublic **K**ey **I**nfrastructure |
| **PSK** | **P**re-**S**hared **K**ey |
| **RDNS** | **R**ecursive **NS** |
| **RFC** | **R**equest **F**or **C**omments |
| **RIR** | **R**egional **I**nternet **R**egistry |
| **RR** | **R**esource **R**ecord |
| **RRL** | **R**esponse **R**ate **L**imiting |
| **RTT** | **R**ound **T**rip **T**ime |
| **SMS** | **S**hort **M**essage **S**ervice |
| **SOA** | **S**tart **o**f **A**uthority |

| | |
|---|---|
| **SPR** | **S**ource **P**ort **R**andomization |
| **TCP** | **T**ransmission **C**ontrol **P**rotocol |
| **TLD** | **c**ode **T**op-**L**evel **D**omain |
| **TLS** | **T**ransport **L**ayer **S**ecurity |
| **TTL** | **T**ime **t**o **L**ive |
| **TXID** | **T**ransaction **ID** |
| **UDP** | **U**ser **D**atagram **P**rotocol |
| **ZCN** | **Z**ero **C**onfiguration **N**etworking |
| **ZSK** | **Z**one **S**igning **K**ey |

# Chapter 1

# Introduction

Undoubtedly, Domain Name System (DNS) [1, 2] is one of the most decisive elements of Internet infrastructure. In fact, excluding IP protocol, DNS is the most utilized network protocol. DNS Name Servers (NS) provide the mapping of a domain name to its corresponding IP address. This simple process constitutes the cornerstone of Internet due to the fact that it comes before any other transaction takes place. It is well-known that resources in the Internet or an internal network are assigned with a domain name which corresponds to a unique IP address. This way, whenever a users wishes to access a network resource of any type, they find first the IP address of the server by resolving the appropriate host name. The usage of domain names instead of numerical addresses is more convenient for the users, since they need to remember the domain name rather than the 32-bit number of IP address. In any case, DNS permits the smooth and reliable operation of Internet, since otherwise, the end-users would have to memorize and provide the numerical IP address of any network resource that they desire to access.

DNS is based on the client-server architecture. The server side of the service forms a distributed database that utilizes a hierarchical multi-tiered tree structure to organize the domain name space into zones. For each zone, there is an Authoritative NS (ANS) responsible to provide answers to incoming requests regarding the resources of the zone under its administration. For this reason, the ANS contains a zone file with the appropriate answers in the form of Resource Records (RR). Each RR maps the resources of the zone to its corresponding domain name. Whenever an application running on a given host needs the IP address of a domain name (or other related DNS RR), it issues a

query to a predefined Recursive DNS NS (RDNS). In turn, RDNS undertakes to traverse DNS hierarchy and locate the appropriate answer. Moreover, for performance reasons, RDNS maintains a cache memory for storing the received RR with the aim of fulfilling subsequent similar requests stemming from the same or another client of the internal network.

Security has become a critical issue for modern computer information systems due, among others, to the rapid and fast growth of computer networks during the past two decades. This growth has exposed computer networks and related protocols to an increasing number of security threats, thus increasing their attack surface. As expected, DNS is not exempt from this rule. However, it can be safely argued that the effects of violating DNS are more devastating compared to any other protocol. Originally, DNS protocol design was not focused on the protection of data's authenticity, but rather on the scalability of the service. Consequently, the initial implementations did not include cryptographic mechanisms that could ensure the validity of the provided data. As a result, the inherent absence of protection and authentication mechanisms have lured potential attackers to manipulate DNS for providing forged responses. Today, however, attacks on DNS have become more sophisticated; the attackers' target is not solely DNS protocol and infrastructure per se, but aim to exploit the protocol for accomplishing various objectives, including assaults against other valuable targets or the construction of advanced and stealth Communication & Control (C&C) channels for coordinating Botnets.

Since the introduction of DNS and until recently, the protocol has proved itself vulnerable to of critical flows. For instance, Bellovin in 1990 [3] discovered a flaw that allow an attacker to breach a system. More recently, Kaminsky in 2008 [4] found a weakness on DNS protocol that permits the violation of DNS data integrity. Specifically, the effect of the so called Kaminsky-style cache poisoning attack is so devastating that its announcement was initially considered as the end of DNS infrastructure as we know it. Also, DNS infrastructure is frequently exploited by DNS amplification attacks, which is a perilous type of DDoS attack. For instance, in a recent DDoS incident, which was characterized as the biggest cyber-attack of this kind in Internet's history, the network infrastructure of Spamhaus was targeted [5]. Actually, Spamhaus suffered for at least a week period a network flood of DNS data that reached 300 Gbps at its peak. As already pointed out, DNS protocol messages can be used for the coordination of a

botnet, i.e., a network of compromised machines that act at the will of their botmaster. In such a scenario, DNS provides a way for the bots to locate their botmaster and communicate with him. In fact, as discussed in chapter 5, this is commonly achieved with Domain Fluxing scheme [6]. It is not to be forgotten that DNS is also utilized for attack reconnaissance, namely an attacker is able to identify their target and/or defensive mechanisms inside an internal network by interacting with DNS service [7]. From the above discussion, it becomes evident that DNS represents a multipurpose vector for a potential aggressor to achieve their malicious objectives.

On the other hand, as DNS is abused for malicious purposes, the aggressors leave noticeable traces of their actions in DNS traffic. Thus, passive and/or active monitoring of DNS network traffic could significantly contribute to the objectives of the defenders. In this way for example, one may be able to detect botnet through group activities in DNS queries [8]. Similarly, spamming services based on worms can be disclosed due to DNS queries related to Mail Exchange (MX) servers [9]. Moreover, detection of malware scanning and zero-day worm outbreaks can be identified with the correlation of DNS and IP traffic [10]. Finally, DNS reputation systems could assist to the identification of malicious domain names [11].

At some point in DNS evolution, DNS Security Extensions (DNSSEC) was proposed. Its aim was the protection of DNS data integrity and verification of their origin. More specifically, the initial proposal was published in 1997 in the form of RFC 2065 [12], but attempts to implement RFC 2065 led to a revised version in 1999 as RFC 2535 [13]. However, RFC 2535 specification had significant problems for scaling to the Internet, so it soon became evident that was unusable for large networks. Finally, a new specification which is fundamentally modified compared to the previous one, was introduced. The current version, known also as DNSSEC-bis is mainly specified in RFCs 4033 [14], 4034 [15] and 4035 [16]. Nevertheless, the DNSSEC deployment in large-scale networks still remains challenging. For the end-user to benefit from the protection offered by DNSSEC, the majority of the DNS infrastructure should adopt DNSSEC. For the root and TLD (upper DNS hierarchy) this is mandatory, but eventually, DNSSEC was first fully deployed at the root level on July of 2010.

Although, DNSSEC intend to add a protection layer to DNS service, as it is discussed in the context of this thesis it can also facilitate the launching of DNS amplification

attacks. In fact, the introduced modification to DNS protocol augment the size of DNS RR, so a potential aggressor favor DNSSEC-related data in order to raise the effects of their actions. Although in such an attack incident the target of the attack is not necessarily the DNS infrastructure itself, the involuntary involvement of DNS entities in DDoS incidents hamper DNS availability to the legitimate users.

## 1.1 Motivation and Objectives

From the findings of DNS security pioneer Steve Bellovin almost 30 years ago until the present days, it is obvious that the battle amongst aggressors and defenders concerning DNS is continuous and intensive. Although in the literature exist a great amount of publications concerning DNS security, constantly new security issues emerge that require the urgent attention of the research community. Novel threats appear almost every day whose implications affect the normal operation of information systems and organizations. Since, DNS precedes virtually any other online transaction, a disruption to its normal operation is reflected to the functionality of the remaining network protocols. Nonetheless, the inconsistent operation of DNS influence substantially the way an end-user experiences the Internet.

Indeed, DNS security is a challenging topic of research. As already mentioned, the protocol is abused in many ways and for various malicious objectives. On the one hand, the violation of DNS data integrity is notably profitable for the attacker, while it may simultaneously put the end-user at great risk. On the other hand, DNS can facilitate evildoers to fulfill their actions by, say, exploiting it to build covert communication channels. The openness of DNS protocol and the absence of full-fledge security mechanisms (with the exception of DNSSEC) allows anyone without restriction and unconditionally to interact with DNS infrastructure.

As already pointed out, DNSSEC was only until recently widely deployed in the upper DNS hierarchy. Therefore, only recently the power of DNSSEC protection and the associated deployment challenges can be scrutinized by the security community in realistic network conditions. Nevertheless, alternative proposals, such as DNSCurve, aspire to take the lead on DNS safeguarding. So, a comprehensive side-by-side comparison that will aid the community to take the necessary decisions for shielding the service is missing.

At the same time, the contribution of DNSSEC to facilitate DNS amplification attacks is neglected. Namely, although there exist scattered concerns about the hefty size of DNSSEC-related RR, so far none of them have studied DNSSEC-powered amplification attacks in much detail.

Given the above mentioned observations, the motivation of the PhD thesis at hand is at the offensive side rather on the defensive one. That is, our intention is to demonstrate and alert the community that DNS service can be exploited as a multipurpose instrument that threatens to violate the integrity, authenticity, confidentiality and availability of the offered resources in Internet. Also, we elaborate on new ways of misuse that render even more challenging for the monitoring mechanisms to detect and repel the corresponding attacks. In a nutshell, the objectives (and simultaneously the research pillars) of the thesis at hand are as follows:

**Objective 1:** We aim to shed light on the various ways attackers are able to forge the integrity of DNS data. This is critical because if this kind of attack is successful, then the aggressors are in position to provide unauthenticated DNS data to the end-users, and put them at risk. This goal also mandates the exhaustive review of the literature regarding the interim countermeasures and cryptographic mechanisms that aspire to defend against forging attempts and theoretically assess their performance.

**Objective 2:** This objective delves into DNS amplification attacks, which as already pointed out, takes advantage of the open nature of DNS protocol and infrastructure to hamper the availability of specific targets on the Internet. That is, DNS amplification attack does solely aim at DNS infrastructure but it can be triggered against any target ranging from personal computer to corporate server. In this context, we indent to explore, propose and evaluate new flavors of DNS amplification that will ensure the qualities of anonymity and reduced resource requirements at the aggressor side.

**Objective 3:** Another significant and emergent misuse of DNS pertains to botnet coordination. More specifically, our goal is to research the contemporary trends of DNS fluxing and DNS-based C&C covert channels used by botmasters to hide their armies. By combining the latter goal with the results of *obj. 2*, we aspire to design and evaluate novel botnet topologies with particular focus to mobile ones. Actually, nowadays, mobile botnets are still in their nascent stages of development but it is for sure to dominate

the future botnet architectures. This is because such botnets greatly benefit from the mobility of the bots and consequently their detection becomes much harder.

## 1.2 Contributions

As already pointed out in the previous subsection, the main goal of this PhD research work is to elaborate on the different ways that DNS service can be exploited as a multi-purpose vector for fulfilling the malicious desires of malevolent entities in the Internet. Namely, one can abuse DNS for threatening the integrity, authenticity, confidentiality and availability of the offered resources in Internet. The realization of this intention is in line with the objective described in the previous section. Overall, the contributions of the current thesis are in accordance with the aforementioned objectives, but also consist novel proposals which considerably add to the literature of DNS security.

Specifically, to deal with obj. 1, we study the current trends on DNS cache poisoning attacks and the possible countermeasures. Moreover, we conduct a side-by-side comparison of the so far prevailing cryptographic mechanisms, namely DNSSEC and DNSCurve, that aim to protect, among others, the integrity and authenticity of DNS data [17]. This comparison will aid the defenders to infer which defensive mechanism best fits to their needs. Chapter 3 details on the aforementioned contributions.

Considering obj. 2, we deal with DNS amplification attacks which target the availability of Internet services. In regards to obj. 1 outcomes, which highlight on the fact that DNSSEC-related RR are considerably sizeable, the facilitation of advanced types of DNSSEC-powered amplification attacks is examined. Precisely, we study the ways a competent and smart attacker can orchestrate and launch wide-scale DDoS attacks which only utilize DNSSEC-related RR [18]. A detailed assessment of the impact of these attacks is also offered. As a side contribution, the performance of (open) DNS forwarders as reflectors is investigated. Generally, the main advantage of our research compared with the standard type of the attack as described in the literature up to now is that it does not disclose any illegal or suspicious activity during its execution. This quality make the attack especially attractive from an aggressor's standpoint. Essentially, we demonstrate that the attacker is able to trivially take advantage of the available resources existing in the Internet without the need to acquire some of their own. To the best of

our knowledge, our research on this particular issue constitutes the first comprehensive study of DNS amplification attack involving DNSSEC-related RRs. Further details of this work is provided in chapter 4.

Additionally, we investigate the potential of taking advantage the ANSs of TLD as both amplifiers and reflectors [19]. From a contribution point of view, this is also the first work in the literature which assess the involvement of the upper DNS hierarchy ANSs in DNS amplification incidents. More precisely, we measure for all ANSs (including root NSs) the size of their response for DNSSEC-related RR, that is, we evaluate the amplification factor these servers may produce. Moreover, we assess the degree of adoption of RRL mechanism which constitutes the primary defensive barrier against involvement in amplification attacks at ANS side. The outcomes of our research exhibit that a worryingly large number of ANSs could be entangled involuntarily to the actions of DDoS attackers. This contribution is also detailed in chapter 4.

As already highlighted in the previous sections, an important part of this doctoral thesis is dedicated to botnets. Since, botmasters consistently misuse DNS for concealing their actions, we concentrate our effort on conducting a comprehensive survey of mechanisms that detect botnets based on their DNS activities, that is, mechanisms that analyze passively and/or actively DNS network traces stemming from bot-infected devices. Based on the results, we propose and evaluate novel botnet architectures, which either rely solely on mobile bots or mixed ones, i.e., include both mobile and desktop agent [20]. The proposed architectures exploit DNS as C&C channel for coordinating the botnet and disseminating botmaster's commands. Chapter 5 further analyzes these contributions.

In addition to legacy DNS attacks, we offer the first to our knowledge work in the literature aimed at examining the potential of DNS as an attack vector to harm mobile users. Specifically, we implement a privacy-invasive mobile app able to manipulate the DNS service provided by an iOS device [21, 22]. This app manages to interfere (acting as man-in-the-middle) to the tethering service present in Apple's mobile devices with the aim to redirect all users connected via it. In this way hijacks the DNS service and enables the attacker to phish user's credentials while they are trying to access legitimate websites. Additionally, by targeting on the popular Siri facility [23], the spyware exposes sensitive user information including its geographical location, account credentials, telephone numbers, sent/received messages, etc. Actually, by using this case study, we

emphasize on the ways DNS forging attempts can by especially profitable to attackers targeting the continuously growing population of mobile users. This contribution fulfills both obj. 1 and 4. Chapter 6 elaborates on this kind of attack.

To sum up, the contribution of the current PhD thesis with reference to publications in scientific journals and conference proceedings is as follows:

- A side-by-side comparison[1] of the prevailing DNS cryptographic mechanisms, namely DNSSEC and DNSCurve, that aim to protect, among others, the integrity and authenticity of DNS data. The aforementioned security extensions aspire to safeguard DNS protocol against DNS cache poisoning attacks.
- Design and evaluation of a new kind of DNS amplification attack[2] that:
  - utilizes solely DNSSEC-related RR as attack amplifiers.
  - takes advantage of the vast number of (open) DNS forwarders existing out there as reflectors.
- A detailed assessment of the potential of taking advantage the TLD's ANSs as both attack amplifiers and reflectors[3]. The great advantage of this attack variation is that it does not require zone walking of the exploited domain zones, but rather utilizes publicly available information. Precisely, this aspect of our research is realized via the following steps:
  - evaluation of the amplification factor these servers may contribute to the attacker.
  - estimation of the degree of RRL adoption by these servers.
- Design and evaluation of novel architectures of mobile botnets[4] with emphasis to mobile ones. We demonstrate that the proposed architectures are:

[1] Anagnostopoulos M., Kambourakis G., Konstantinou E., Gritzalis S., DNSSEC vs. DNSCurve: A side-by-side comparison, chapter in Situational Awareness in Computer Network Defense: Principles, Methods and Applications, Onwubiko C., Owens T., (eds), pp. 201-220, 2012, Hershey, USA, IGI Global. doi:10.4018/978-1-4666-0104-8.ch012

[2] Anagnostopoulos M. , Kambourakis G., Kopanos P., Louloudakis G., Gritzalis S., DNS Amplification Attack Revisited, Computers & Security, Vol. 39, pp. 475-485, 2013, Elsevier. doi:10.1016/j.cose.2013.10.001

[3] Anagnostopoulos, M., Kambourakis, G., Gritzalis, S., Never say never: Authoritative TLD nameserver-powered DNS amplification, Under Preparation, 2016.

[4] Anagnostopoulos M. , Kambourakis G., Gritzalis S., New facets of Mobile Botnet: Architecture and Evaluation, International Journal of Information Security, 2016, Springer. doi:10.1007/s10207-015-0310-0

  - robust, stealth, and flexible mainly due to the use of mobile bots and DNS as a covert C&C channel.
  - particularly effective in launching DNS amplification attacks.

  It is to be noted that in the context of our research we also investigated[5] the Session Description Protocol (SDP)[24] part of Session Initiation Protocol (SIP) [25] request messages as a different way of creating covert C&C channels [26]. However, as this direction of our research is not directly associated to DNS but to botnets we opt to exclude it from the current thesis.

- Desing and implementation of an iOS based spyware app[6,7], which hijacks and forges the DNS service offered by operating system of the mobile platform with the mission of:
  - poisoning the device's tethering service, thus redirecting the user to bogus websites.
  - leveraging on the specific to iOS devices Siri facility to intercept sensitive user information, including their geographical location, account credentials, address book, etc.

Furthermore, Fig. 1.1 illustrates these contributions with reference to the security services they aim to address.

## 1.3 Thesis Structure

The next chapter starts by providing the basic background of DNS service. This includes the entities that participate in a typical DNS transaction, the type of DNS data, and the possible configurations. This way, the reader acquires the essential concepts to

[5]Tsiatsikas Z., Anagnostopoulos M., Kambourakis G., Lambrou S., Geneiatakis D., Hidden in plain sight. SDP-based covert channel for Botnet communication, 12th International Conference on Trust, Privacy & Security in Digital Business (TrustBus), September 2015, Valencia, Spain, Springer. doi:10.1007/978-3-319-22906-5_4

[6]Damopoulos D., Kambourakis G., Anagnostopoulos M., Gritzalis S., Park J.H., User-privacy and modern smartphones: A siri(ous) dilemma, FTRA AIM 2012 International Conference on Advanced IT, Engineering and Management, S. Rho, N. Chilamkurti, W.-E. Chen, S.-O. Park, (eds), February 2012, Seoul, FTRA.

[7]Damopoulos D., Kambourakis G., Anagnostopoulos M., Gritzalis S., Park J., User privacy and modern mobile services: Are they on the same path?, Personal and Ubiquitous Computing, Vol. 17, No. 7, pp. 1437-1448, 2013, Springer. doi:10.1007/s00779-012-0579-1

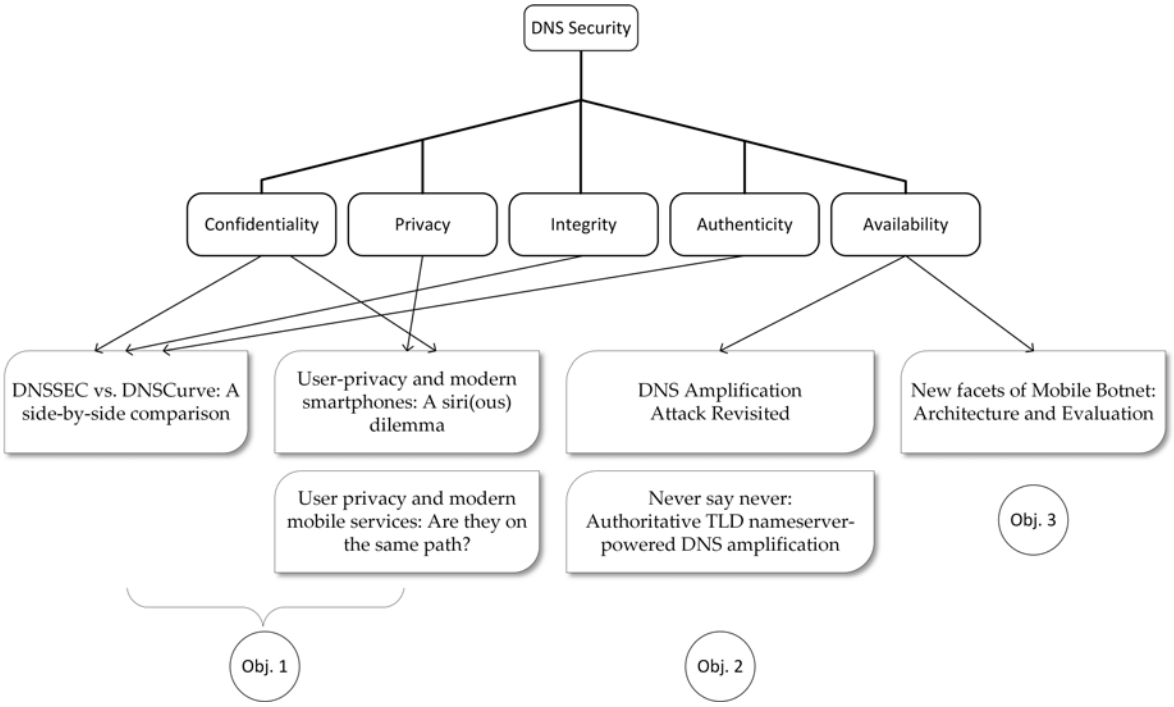



FIGURE 1.1: Contribution of PhD thesis with the relative security services

comprehend the remaining doctoral thesis. The chapter concludes by briefly presenting all legacy attacks against DNS. Precisely, we highlight on DNS attack vectors and pave the way towards chapter 3 which aims at addressing more advanced attacks on the protocol.

Chapter 3 details on DNS cache poisoning attack that threatens the integrity of DNS data. Specifically, this chapter details on the the aforementioned type of attacks, examines the attacker's strategy, and highlights on its consequences for an end-user's viewpoint. Furthermore, possible countermeasures are given. First off, we discuss interim measures that intend to increase the entropy of a DNS query, and therefore to obstruct poisoning attacks. The same chapter succinctly overviews the prevailing cryptographic mechanisms, namely DNSSEC and DNSCurve, that extend DNS protocol and aim to safeguard against forging attempts. A detailed comparison between the two aforementioned DNS security mechanisms is also presented. Lastly, the until now most recent Internet drafts related to DNS security, namely DNS-over-TLS and DNS-over-DTLS DNS over DTLS and TLS Internet drafts are briefly described. At a glance, these Internet drafts incorporate the well-known and reliable DTLS and TLS protocols to DNS.

The DNS amplification attack problem is addressed in chapter 4. Based on the literature,

the chapter elaborates on the ways a typical DNS amplification attack unfolds and what entities are exploited as amplifiers and reflectors. Next, based on our research, two novel attack strategies are presented. The first one takes advantage of the hefty size of DNSSEC-related RR and the available (open) DNS forwarders that operate on Internet. The latter investigates the potential of entangling ANSs of the upper DNS hierarchy to be unknowingly engaged by attackers in DNS amplification attacks. We specifically assess the amplification factor that these entities may provide when replying to both individual and multiple DNS queries and measure the adoptation degree of RRL mechanism at ANS side. The chapter ends by suggesting possible countermeasures against this kind of attacks.

Chapter 5 is devoted to exploring DNS as a vehicle for controlling botnets. Actually, this research aspect constitutes the second pillar of our research. Precisely, we detail on the ways DNS protocol can be exploited by botmasters for the coordination of their infrastructure. We specifically focus on mobile botnets and examine different architectures - both purely mobile and mixed - that may be used by bot herders to build a robust and potentially untraceable botnet. The construction of effective C&C based on DNS to control the botnet is also demonstrated and assessed under various attack scenarios. The chapter concludes by presenting a comprehensive survey of countermeasures, both passive and active, that are based on DNS trace analysis.

Another aspect of our research this time pertaining to mobile applications is given in chapter 6. Namely, we detail on the development of an iOS based privacy-invasive malware application. Interestingly, the malware interferes with the DNS service to redirect the user to bogus websites and hijack Siri's personal assistant session aiming at stealing sensitive user information.

The last chapter concludes the thesis by summarising and commenting on the results of the conducted research. Best practices regarding the protection of DNS service and directions for future work are provided alongside.

# Chapter 2

# Background

In this chapter, we elaborate on the basic concepts of DNS protocol. Initially, we explain the purpose of this service and analyze the process of domain name resolution. In this respect, the chapter is also includes a list of the various DNS data types. Finally, the two legacy DNS attacks, namely DNS cache poisoning and DNS amplification, are briefly described. The goal of this chapter is to provide the necessary information for understanding more advanced DNS attack vectors included in the following chapters.

## 2.1 DNS Service

### 2.1.1 Introduction

The Domain Name System (DNS) is the service that translates domain names, e.g., www.example.com, into the numerical IP address of a host in a network. Any network or the Internet operates by allocating a locally or globally unique IP address to every endpoint (host, server, switch, router, etc). This address, which an end-user has to utilize in order to access a resource available on the network, is formed by a 32-bit number in case of the IPv4 or a 128-bit number in the case of IPv6. Consequently, the absence of the DNS service would imply that whenever a user desires to access a resource they need to memorize its physical IP address. For that reason, every public node in the network is assigned with a descriptive domain name, as it is more convenient for the users to remember its name rather than a numerical address.

DNS constitutes a distributed database organized into a hierarchical tree structure (similar to Fig. 2.1) to form the domain namespace. This tree structure is comprised of labeled nodes, each one corresponding to a domain. The Fully Qualified Domain Name (FQDN) of a node is composed of the bottom-up concatenation of the nodes, i.e., labels, with each label separated by a period.

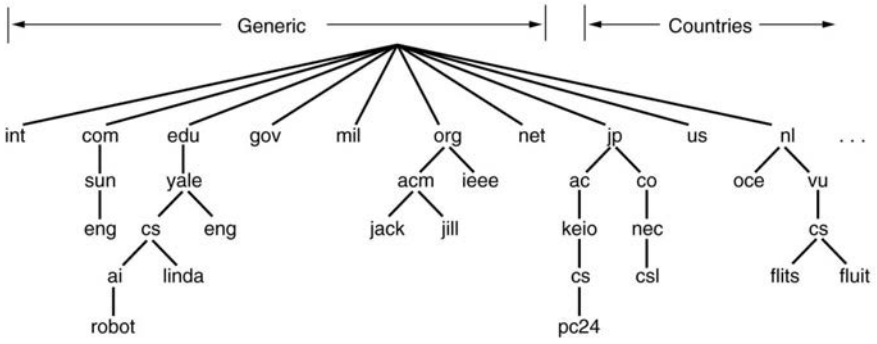



FIGURE 2.1: DNS tree structure

### 2.1.2 DNS Domains

The namespace of Internet is divided in domains, which are groups of names with a logical relationship amongst them. For example, a domain can specify names that belong to a particular country, organization or university, etc. Further, any domain can be subdivided into sub-domains aiming at better organization of the contained domain names. As previously mentioned, DNS utilizes a hierarchical tree structure that is similar to the structure of the Unix filesystem. The whole DNS database is pictured as an inverted tree. At the top of the tree is located the root node [27], which is represented with a dot ".". Next, are the Top-Level Domains (TLDs), then the Second-Top-Level Domains (2-TLDs) followed by any other number of lower levels. Each one of the labels of the domain name separated by a dot corresponds to a level in the tree structure. The root level has an empty label, while the most right label represents the most general domain, i.e., TLD. Accordingly, the second most right fragment corresponds to the 2-TLD (more specific domain) and so forth. The left fragment of the domain name is the distinct host within this domain.

TLDs are split into two main categories [28], namely generic Top-Level Domains (gTLD) and country code Top-Level Domains (ccTLD). The domains under gTLD are the ones

that operate directly under policies established by Internet Corporation for Assigned Names and Numbers (ICANN) processes for the global Internet community. These gTLD domains are used mainly for services located in the USA. The core group of generic top-level domains consists of the .com, .info, .net, and .org domains. On the other hand, ccTLD are used for the organization of domain names within individual countries. These are two-letter domains and correspond to ISO 3166 [29] country codes as described in RFC 1591 [28]. Examples of ccTLD are the .gr domain for Hellas, .es domain for Spain and .cn for China to name a few.

### 2.1.3 DNS zones

A DNS zone is a group of nodes with a common parent, which forms a subtree structure. Specifically, a zone comprises of a collection of domain names with common upper level domain. The parent of a zone is called the Start of Authority (SOA) and is stated as such in the corresponding database. Authoritative Name Servers (ANS) have the responsibility to administer the nodes inside their zones, i.e., to provide answers for the contained domain names. Hence, either they delegate such authority for their subzones downwards to others authoritative servers (delegation points) or they have this authority for their leaf nodes [1]. In other words, an ANS either provides referrals for sub-zones whose authority the ANS has delegated or supplies mappings for its leaf nodes [1]. Essentially, the separation to zones materializes the distributed administration of the whole namespace.

### 2.1.4 DNS Reverse Mapping

Besides providing mappings from memorable domain names to corresponding IP address, DNS also offers mappings from IP addresses to domain names. This process, called reverse mapping or reverse lookup, determines the domain name of a host having a particular IP address. This type of operation could be useful for security reasons, for instance can be used by a mailing software to resolve IP address to domains in order to authenticate the sender. For performing reverse mapping using DNS infrastructure, a special tree structure is defined and specialized domains are reserved. The domain names formulated by IP addresses are called Reverse Domains and are within the in-addr.arpa zone for the case of IPv4 and the ip6.arpa for the case of IPv6 [30]. As it is observed

from Fig. 2.2 [31] under the inaddr-arpa zone there are domains, whose name is the first number of the IP address. This in turn means there are 256 distinct domains with names from 0 to 255. Similarly, under each of them there are 256 domains representing the second number of the address etc. The domain name of reverse domains is composed by the IP address in reverse order and the in-addr.arpa label at the end. For example, the IP address 15.16.192.152 corresponds to the 152.192.16.15.in-addr.arpa domain name.

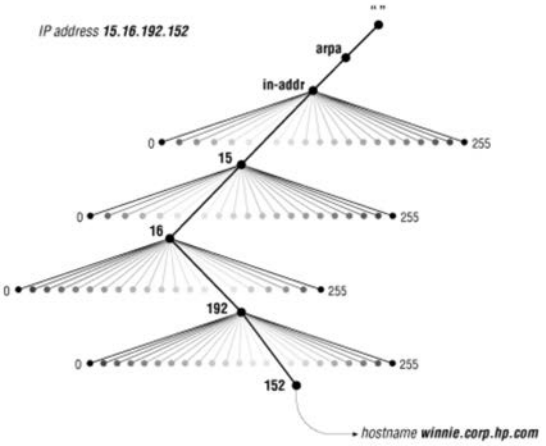



FIGURE 2.2: DNS Reverse Tree

As it is well-known, the 127.0.0.1 IP address is a special address which is reserved for loopback, i.e., for communication of the host with itself. Hence, this address is utilized by every host. Therefore, each server is also authoritative for the domain 0.0.127.in-addr.arpa and it should have the proper configuration to support queries for that domain.

### 2.1.5 DNS Operation

The basic entities that are involved in a DNS transaction are (a) the end-user, (b) Recursive Name Server and (c) Authoritative Name Server.

Basically, a DNS transaction takes place every time a user desires to access a resource available in the Internet by utilizing its corresponding domain name. The user's machine and in particular its stub-resolver library is responsible for resolving, namely fully translate, the remote machine's domain name. This means that the stub-resolver has to find the IP address and other relevant information of that machine. The stub-resolver depends on a Recursive NS (RDNS) for obtaining the required information about the remote machine. Usually, the RDNS is operated by the ISP of the network that the

machine is connected to. In turn, the RDNS undertakes to traverse the DNS hierarchy for locating the ANS for the domain name in question. Then, it directly contacts this ANS and acquires the response in the form of resource records (RRs). The ANS responds to incoming requests according to the zone file stored in its file system. Finally, the recursive caches the RRs contained in the response for a given Time to Live (TTL) and delivers them back to the initiator stub-resolver [32].

### 2.1.6 DNS Queries

There are two types of queries that a NS can respond to (a) recursive and (b) iterative (non-recursive) query:

A *recursive* query is a query that the receiving NS endeavors all the necessary tasks to locate the responsible ANS in order to find and return the complete answer (or an error). Resolving a query recursively may involve the NS in multiple transactions with a number of other NSs. It is optional for a NS to support recursive queries [1]. On the other hand, in the case of an *iterative* query, the receiving NS provides a complete answer for the queried domain name if the NS is authoritative for that domain or it provides a partial result (or an error). This means that if the queried domain name is contained in the NSs zone file, it replies with the appropriate RRs. While if the NS does not contain the answer, it provides referrals to other NS that do so. The default requirement is that NSs must support iterative queries [1]. By default NSs listen on port 53 for DNS transactions. For performance reasons this transaction is based mainly on the User Datagram Protocol (UDP) Protocol with packet size limited to 512 bytes in the initial versions of protocol. The limitation of response's size intends to primarily hamper the entanglement into denial of service (DoS) incidents.

### 2.1.7 Entities in a DNS transaction

This section elaborates on the basic entities of DNS infrastructure, that participate in the resolution of a domain name [33].

#### 2.1.7.1 Resolver

The DNS resolver represents the client-side of DNS architecture. A resolver, software or library, is installed on every host (for example it may be contained in a web browser) and provides the appropriate functionality for translating a user's request into queries to a RDNS. Also, it is responsible for interpreting the received RRs from the RDNS into a proper answer for the requesting program. The RDNSs utilized by a resolver depend on the network that the resolver is connected to and usually are defined by the user's ISP. Basically, the resolver creates recursive queries and accredit the full resolution process to the responsible recursive server of its connected network. As already pointed out in section 2.1.5 a resolver is also known as stub-resolver because of its simplistic implementation and absence of cache memory. The lack of cache memory constrains the resolver to solely depend on a RDNS for each request for resolution of a domain name.

#### 2.1.7.2 Name Server

A Name Server (NS) accepts DNS queries from a resolver or another NS that acts on behalf of a resolver and aims to fulfil this request. Depending on its configuration the NS provides full or partial answer to a DNS query. Its configuration also determines the mode that the NS operates and consequently the way it stores or locates the requested data. The operation mode is controlled by its configuration file, which in the case of BIND is called named.conf [32]. Next, we analyze the variant types of NS.

**Primary Authoritative Name Server**

A Primary ANS contains one or more zone files stored in its local filesystem. Recall that as explained in 2.1.3, a zone file consists of domain names for which the server is authoritative. The main operation of a primary ANS is to respond to DNS queries for those names. Putting it another way, a primary ANS provides answers only for domain names of its zone file. The zone file is created and updated by the administrator of the zone. Also, the primary ANS is responsible for transferring the contents of the zone files to one or more Secondary ANS whenever the file is modified [34].

**Secondary Authoritative Name Server**

A Secondary ANS obtains the data of the zone that is authoritative, via a zone transfer

operation from the Primary ANS [34]. Hence, it polls periodically the Primary ANS for changes at a time interval that is defined by the SOA record. The Secondary ANS also provides authoritative answers for the specific zone.

#### Authoritative-only Name Server

An Authoritative-only NS is a type of NS that provides authoritative answers, namely it is primary or secondary NS for one or more domains, and also does not support recursive queries and cache capabilities. The root NS and authoritative NSs for TLD zones or for domain names with high traffic are operating in this mode as they have excessive performance requirements.

A Root NS is a special authoritative-only name server. It is authoritative for the root zone, and thus it provides referrals to the corresponding ANS of the TLD zones. The currently 13 root NS have assigned with names from A.root-servers.net to M.root-servers.net [35]. Actually, their number is much larger as for each root NS multiple instances in various location on the globe exist, which still have the same IP address. Actually, the operation of multiple NS with the same IP address but different physical location is feasible with anycast routing [35]. Nevertheless, the choice of utilizing only 13 root NS was made in order to comply with the limitation of 512 bytes of a UDP message packet in DNS protocol.

#### Recursive Name Server

A RDNS or commonly known as recursive resolver is not authoritative NS for any zone. Instead, its role is to accept recursive queries and provide back to requestors complete answers. This is the reason why recursive servers are considered as part of the client-side of the infrastructure. Whenever a RDNS accepts a query from a stub-resolver, it firstly examines if it has the desired answer in its cache memory. If not, it traverses the tree hierarchy from the ANS of root zone downwards to ANS of the specific domain issuing iterative queries for the required domain name. Afterwards, it caches the acquired RRs locally in its cache memory and returns the response to the requesting resolver. Upon subsequent requests for the same data, the recursive will respond with its locally stored data from the cache rather than search again for and query the responsible ANS. These data remain in the cache memory until their TTL value expires and so they are discarded. Following requests for the specific RRs will force the recursive to lookup and contact

again the respective ANS. Every recursive bootstraps with the list of the 13 root ANS preconfigured. This way, it knows from which node of the DNS hierarchy should start the traversal. For performance, reasons the recursive remembers the response time of each root server and selects to consult the one that is closer to it.

Whenever a RDNS obtains the required RRs directly from an authoritative, it qualifies the response as authoritative. Otherwise, in the case the records are picked from his cache, the answer is marked as non-authoritative. The use of the cache memory is for performance reasons, as it prevents the redundant communication between the recursive and the ANS. Due to cache capabilities the recursive does not overwhelm the root or TLDs ANS with queries for identical or similar RRs. Furthermore, cache memory facilitates the rapid resolution of frequently accessed resources by distinct end-users connected to the same internal network. In fact, the recursive obtains an authoritative copy of the frequently requested RRs and keeps them available for TTL value with no additional overhead.

Usually, such type of NSs are deployed by ISPs to provide naming services to their customers. In most cases, the access to a RDNS is limited to the internal users only. However, recently, there have been deployed recursives that allow recursive queries from anyone on the Internet. Such servers are called Open Recursive NS (ORDNS) [36].

#### Forwarding Name Server

A Forwarding NS, or simply forwarder, is a special type of NS which is utilized for performance reason on slow networks. As implied by its name, a forwarder just forwards all receiving DNS queries to another NS, which is capable to handle recursive queries, and then it caches the acquired RRs. Such formation is useful when the access to an external network is slow and expensive. This is because the resulted response will take a single DNS transaction, while multiple transactions would occur in the case the local NS recursively resolves the DNS queries and not simply forwards them. Also, the forwarder is capable to supply rapidly responses for popular resources, and thus eliminates unnecessary external traffic by caching the RR.

Summarizing, it is worth noticing that the existing DNS software permits the hybrid usage of the variant modes. So, a contemporary NS may act as a primary for a given domain, meaning that administers its zone files, as a secondary for others zones, and

at the same time will accommodate recursive or forwarding functions for internal users [32].

### 2.1.8 Format of Zone File

As explained previously in section 2.1.5, a *zone file* is a text file that is stored in the Primary NS's filesystem and represents the available resources of the specific zone. Whenever, the NS receives a query for a domain that is authoritative, it consults the zone file in order to respond. The translation of a domain name into the characteristics, properties or entities contained within the domain is defined by the *Resource Record* (RR). A RR is spread on a single line with the exception that entries enclosed in parenthesis can expand across multiple lines. The formation of the file is standardized by RFC 1035 [2]. The syntax of each line, which represents a single RR, is as follows [37]:

*[Name] [TTL] [Class] Type Type-specific-Data*

The fields inside brackets are optional and when they are omitted the corresponding field takes the value of the previous line record.

- **Name**: This field contains the name of the domain, either in FQDN (that name ends with a dot) or in relative format. Also, this field could have the @ value in the case of a SOA type RR, which means that the field has the domain name of the zone as its value as specified in the configuration file.
- **TTL**: The TTL field determines the time in seconds for which the particular record is valid. In other words, when the TTL expires, this RR has to be discarded from the cache of the NS.
- **Class**: The Class field can take the values IN (Internet), CH (Chaos) and HS (Hesiod). Each one of these classes forms a completely independent tree. Usually, the class is set to IN as the records refer to Internet hostnames, services or addresses, while the other types of classes are considered as stale.
- **Type**: This field determines the form of the RR and its intended use. The possible values of this field are presented in section 2.1.9.
- **Type-specific-Data**: This last field takes values depending on the type of the RR.

Additionally, a zone file may contain directives, which are utilized to control the file processing. Directives start with the symbol '$'. Mostly, the following three directives are employed:

- **$ORIGIN**: The $ORIGIN directive determines the domain name of the zone. So, if a domain name is in relative format, namely does not terminate with a dot '.' symbol, then it is concatenated with the value of this directive [2].
- **$INCLUDE**: The $INCLUDE directive dictates to insert the defined file into the zone file [2].
- **$TTL**: The $TTL directive defines the default TTL value for the RRs of this zone. Thus, if the TTL value in a RR is omitted, then its TTL is equal to the directive's value [38].

### 2.1.9 Resource Record Types

In the following, we detail on the prevailing RR types that are employed in a typical zone file. We decided to exclude the DNSSEC-related RR types, from the analysis and instead present them in section 3.2.3.2. Table 2.1 summarizes the common RR type of DNS protocol.

#### 2.1.9.1 SOA Record

The *Start of Authority (SOA)* RR type specifies the Primary ANS of the zone. There is only one SOA record per zone file and is placed at the beginning of the file. As an example, the following listing contains the SOA RR of the root.zone file, which is the zone file administer by root ANS [39]:

```
. 86400 IN SOA a.root-servers.net. nstld.verisign-grs.com. (
              2016062900 ; serial number
              1800 ; Refresh every 30 minutes
              900 ; Retry after 15 minutes
```

```
        604800 ; Expire after 7 days
        86400 ; Negative Cache TTL for 24 hours
       )
```

We can easily notice that the previous example of the SOA RR consists of the following fields [1]:

- **Name**: In the name field we observe the '.' value, which is the domain name of the root zone.
- **TTL**: As TTL value, the RR has 86400 seconds that is equivalent to one day.
- **Class**: The class is IN, so this record refers to an Internet domain.
- **Type-specific-Data**: The specific data provided by SOA RR are as follows:
  - **Primary ANS**: The Primary ANS of the zone is a.root-servers.net., the ANS's domain name is specified in FQDN format.
  - **Email Address**: The email address of the administrator of this zone is nstld@verisign-grs.com. The first dot in the email address field represents the symbol @, as this symbol has a different meaning in zone file's syntax.
  - **Serial Number**: The serial number defines the current version of the zone file and it is used to determine when a zone transfer is needed. Any modification of the file should increase the serial number. Usually, that number is based on the date when the modification happens. For instance, the specific zone file has been modified on 06/29/2016.
  - Subsequent values are expressed in seconds and designate the frequency of specific administration processes:
    - **Refresh**: The refresh interval defines how often the Secondary ANS polls the Primary to infer whether the data of the zone is up to date. This is done by comparing the serial numbers of the zone files. If a modification has occurred, then the Secondary ANS should initiate a zone transfer, namely it should request the file in order to update its zone.

- **Retry**: The retry value designates the time period to pause before the Secondary ANS retries to reconnect the Primary ANS after a failed poll.
- **Expire**: The expire value determines the time after which the data stored in the Secondary ANS will become invalid in the case it fails to contact with the Primary one. After that time, the Secondary ANS will cease to act as authoritative of the zone as the RRs are considered obsolete.
- **Negative Caching TTL**: The Negative Caching TTL value applies to negative responses by all the ANS of the specific zone. It signifies for how long a response for a non-existent domain (NXDOMAIN) of the zone will remain in the cache memory of the requesting recursive.

#### 2.1.9.2 NS Record

The *NS* RR type specifies the ANSs of a zone, both Primary and Secondary. If the ANS provided is within that zone, then the file should also contain its IP address with a corresponding A RR. The next listing depicts a fraction of the NS RR contained in the root.zone with the name field in FQDN format [39]

```
. 518400 IN NS a.root-servers.net.
gr. 172800 IN NS gr-m.ics.forth.gr.
gr. 172800 IN NS estia.ics.forth.gr.
estia.ics.forth.gr. 172800 IN A 139.91.191.3
gr-m.ics.forth.gr. 172800 IN A 194.0.4.10
```

The right part of RR lists the ANS of the domain comprized of the name field. As shown in the listing, a NS RR also is used to indicate the delegation of a subzone. For instance, the authority of the gr. subzone has been delegated to gr-m.ics.forth.gr. and estia.ics.forth.gr. ANS. Additionally, the file should specify the IP address for these ANS. This way, RDNS are able to reach the subzone's ANS. Such a RR, namely RR that maps IP address of ANS, is called *glue record*.

#### 2.1.9.3 A and AAAA Record

The *A* and *AAAA* RR types are the most frequently requested RRs. This kind of RR provides mappings between hostnames and IP addresses. An A RR maps to IPv4 addresses while an AAAA maps to IPv6 addresses.

```
a.root-servers.net. 518400 IN A 198.41.0.4
a.root-servers.net. 518400 IN AAAA 2001:503:ba3e:0:0:0:2:30
```

As the above example implies, the host of A Root ANS is in 198.41.0.4 (IPv4) and 2001:503:ba3e:0:0:0:2:30 (IPv6) address. Both records refer to the same domain name, but provide IP addresses of different versions.

#### 2.1.9.4 CNAME Record

The *CNAME* RR type supplies alias of hostnames, which allows one host to be defined as the alias name for another host.

```
www1 IN CNAME www.example.com.
www IN A 93.184.216.34
```

The above statements for example result to that www1.example.com. and www.example.com. domain names match to the same host with 93.184.216.34 IP address. CNAME record is applicable for hosts that run different services.

#### 2.1.9.5 TXT Record

The *TXT* RR type contains a textual description for the specific domain name. Usually, this RR type is capitalized for declaring that the zone supports specific mechanisms like Sender Policy Framework (SPF) [40], which is a validation mechanism for detecting email spoofing. SPF utilizes a specially formatted TXT RR that defines the list of authorized email servers for the particular domain.

```
example.com. 59 IN TXT "v=spf1 -all"
example.com. 59 IN TXT "$Id: example.com 4415 2015-08-24 20:12:23Z davids $"
```

As observed in the previous example, a given domain zone can posses more than one TXT records. Whenever a domain supports the SPF mechanism, then it encloses a RR similar with the first one.

#### 2.1.9.6 MX Record

The *Mail Exchanger (MX)* RR type determines the liable email servers of a domain. Therefore, whenever a user desires to send an email, the email software inquires these type of RR to detect the responsible mail exchanger that accepts email messages for the specified recipient. Additionally, a MX RR indicates a priority number that specifies the order that email exchanger servers will be utilized in case of connection failures. For instance the following records:

```
gmail.com. 3600 IN MX 5 gmail-smtp-in.l.google.com.
gmail.com. 3600 IN MX 10 alt1.gmail-smtp-in.l.google.com.
gmail.com. 3600 IN MX 20 alt2.gmail-smtp-in.l.google.com.
gmail.com. 3600 IN MX 30 alt3.gmail-smtp-in.l.google.com.
gmail.com. 3600 IN MX 40 alt4.gmail-smtp-in.l.google.com.
```

determine that the gmail.com zone has five mail servers. Thus, one desires to send a message to an @gmail.com address, the message is firstly sent to gmail-smtp-in.l.google.com server, if that connection fails then it is sent to alt1.gmail-smtp-in.l.google.com, and so on.

#### 2.1.9.7 PTR Record

As described in section 2.1.4, the *PTR* RR is used to implement reverse DNS mapping. Opposite to A RR type, PTR maps IP Address to hostnames. In the following example, we can see the host that responds to the IP 195.251.124.222 address.

```
222.134.251.195.in-addr.arpa. 900 IN PTR hra.aegean.gr.
```

#### 2.1.9.8 CSYNC Record

Most recently, a new RR type called *CSYNC* is proposed for informing a parent zone about which of the child's RRs are renewed and should be updated [41]. Specifically, there exist some RRs associated with the delegation from the parental zone to the child which have to be synchronized whenever modified, otherwise the child zone would be inaccessible. Prior to the proposition of RFC 7477, that notification was manual, so CSYNC aims to automate this process. An example of a CSYNC record is given below [41]:

```
example.com. 3600 IN CSYNC 66 3 A NS AAAA
```

As it is easily observed, the CSYNC contains in its RDATA part the serial number of the SOA RR, a set of flags, and a list of RR types. The SOA's serial number determines the version of the child's zone file. The flags define the time when the synchronization should take place, where the two possible values are 1 (immediate) and 2 (soaminimum) or combination of them. Finally, the set of RR types indicates which of the RR types in the child zone should be queried by the parental in order to revise the delegation RRs within its zone. From the paradigm we can deduce that for the example.com child zone the A, NS, and AAAA types should be queried by the parental ANS and the latter entity should extract the corresponding RRs only if that zone has no less than 66 as SOA serial number. Whenever the administrator of the child zone reform the delegation RR, they have to publish the corresponding CSYNC RR at the zone for informing about the necessary update.

### 2.1.10 DNS Message Format

The format of a DNS transaction message is similar for both queries and responses [2]. As we can see in table 2.2, the message contains five sections some of which can be empty depending on the type of message or requested RR type.

| RR | RFC | Description |
|---|---|---|
| A | RFC 1035 | 32-bit IPv4 Address record |
| AAAA | RFC 3596 | 128-bit IPv6 Address record |
| CNAME | RFC 1035 | Canonical Name. Alias of the domain name |
| HINFO | RFC 1035 | Host Information. Description of hardware and software of the server |
| MX | RFC 1035 | Mail Exchanger domain name |
| NS | RFC 1035 | Name Server. The authoritative name server for the particular zone |
| PTR | RFC 1035 | Pointer record. Reverse mapping from IP address to domain name |
| SOA | RFC 1035 | Start of Authority. Determines the Primary ANS, email of administrator and serial number of zone file |
| SRV | RFC 2782 | Defines Service available in the zone |
| TXT | RFC 1035 | Textual description |
| URI | RFC 7553 | Defines the full URI of a service |

Table 2.1: Common DNS Resource Record Types

| | |
|---|---|
| Header | |
| Question | Question to the NS |
| Answer | RRs answering the question |
| Authority | RRs pointing toward an authority |
| Additional | RRs holding additional information |

Table 2.2: DNS Message Format

The *Header* section is always present and is comprised of control fields. The fields declare which of the remaining sections are present, whether the message is query or response, etc. The most important control field is the transaction ID (TXID). This field is a 16 bit message identifier randomly generated by the software that initiates the DNS query. This number is copied unchanged in the corresponding response and its usage is to distinguish a transaction. Therefore, the requestor is able to match up responses with pending queries.

The *Question* section carries the queries which are sent to a NS, regardless it is RDNS or ANS. It contains at least one entry with the according parameters which specify what the client asks to find out. These parameters should include the domain name, the type and the class of the resource in question. Subsequently, the question section is copied to the response without modification so the client can check the genuineness of the response.

The remaining three sections compose the response from a NS and they all have the same format, as they carry a set of RRs. Each section consists of zero or more records

and the only difference is the semantics that the client assigns to the contained RRs. The *Answer* section contains RRs that answer the initial question. The *Authority* section carries RRs (of type NS) which refer to the ANS responsible to resolve the domain in question. Finally, the *Additional* section conveys records that provide additional information relevant to the query, but not directly resolving the query. Usually, the Additional part is utilized to supply address mappings for the ANS contained in the authority. These mappings are also called glue records, as they are companying NS type RRS and are employed for traversing the DNS hierarchy.

### 2.1.11 Zone Update

In order to simplify the operation of multiple ANS, it will be useful whenever the administrator of a zone needs to update the zone to just modify the zone file of a single ANS and propagate the changes to the rest of the ANS. This is accomplished through a zone transfer process which deploys features of DNS protocol. Thus, the zone transfer is a DNS transaction that allows the replication and synchronization of DNS data (zone files) amongst the primary and one or more secondary ANS of that zone. This transaction can only be initiated by the secondary ANS, although the primary ANS could inform its secondary ANSs that an update has occurred. Albeit DNS communication is mainly based on UDP protocol, zone transfer operates over Transmission Control Protocol (TCP). Recall from section 2.1.6 that this is because the that initial version of DNS protocol limits UDP packet size on 512 bytes for security reasons. Usually, however, zone transfer needs more than 512 bytes, since the transferred data contain far more RR than a typical DNS response. In the rare case that the transferred data have size less than 512 bytes, UDP protocol can be used by the primary ANS.

#### 2.1.11.1 Full Zone Transfer

At regular intervals the secondary ANS should poll the primary ANS about updates of its zone. As analyzed in section 2.1.9.1, the time interval between pollings is determined by the refresh value of SOA record. The secondary ANS firstly request the SOA record and checks its serial number. If the serial number is greater than the serial number maintained by the secondary, then is asks for a Full Zone Authoritative Transfer (AXFR).

This means that the secondary ANS asks the primary server to send all RRs contained in its zone file [34].

#### 2.1.11.2 Incremental Zone Transfer

The latter mode of zone transfer generates redundant network traffic since it transmits even RRs that have not been modified. Therefore, an Incremental zone Transfer (IXFR) would be desirable, videlicet a transfer mode that would send only the RRs that have changed (added, updated or removed) since the last zone transfer. This way, the primary ANS preserves network bandwidth by avoiding unnecessary transmissions. As a result, whenever the secondary ANS perceives through the regular polling process that it has an out-of-date zone file informs the primary one about the current version of its SOA. A primary ANS that supports IXFR should keep track of the recent version of its zone and the differences amongst the various versions. Thus, it is able to send only those RRs that have been modified since the last transfer to the secondary ANS [34, 42].

#### 2.1.11.3 Notify

The choice of refresh time interval is challenging. Long time interval could cause slow propagation of modified RRs and incoherence of zone files amongst the ANS. On the other hand short intervals would increase the load on the side of the primary ANS. The DNS Notify message type introduced in [43] comes to rescue. The primary ANS of a zone is able to send a NOTIFY message to the secondary ANS, which are defined in NS RRs in the zone file, whenever the zone is loaded or updated. Following, the secondary ANS will request the SOA record from the primary one. If the SOA serial number is greater since the last time the zone was fetched, the secondary ANS will request for a zone transfer (full or incremental). This process can considerably reduce the propagation time of zone changes to secondary ANS.

#### 2.1.11.4 Dynamic Update

Initially, DNS was designed to support queries on statically configured zone files. So, the typical practice to update the zone was to manually edit the zone file and then to restart the name service. Thus, as already pointed out, the primary ANS reads the file and

propagates the changes to the secondary ANS. This approach imposes the ANS to be out of operation every time there is a modification in the zone. A process called Dynamic DNS (DDNS) [44] allows modifications on the zone records from external sources while the NS stay still operational. This method permits all types of RR except SOA to dynamically added, deleted or modified within an existing zone. However, DDNS does not enable the addition or deletion of new domain or zone. Whenever the administrator updates the zone file on-the-fly, it is essential to update only the primary ANS. The remaining of the secondary ANS will update their database via zone transfer (AXFR or IXFR). Moreover, in the configuration file of the DNS software the administrator is able to determine the sources from which dynamic updates for the zone are allowed.

#### 2.1.11.5 Wildcards

The wildcard character '*' (asterisk) can be used as the most left label of the domain name in a RR [1]. RRs containing wildcard will match requests for non-existent domain names within the zone. To exemplify this situation the following RR will force DNS queries for subdomains of example.com zone that they have not been explicitly specified with other RR to have a MX RR resolved from that one. For instance, if there is no record for random.example.com. then a query for that name of type MX will return the specified record.

```
*.example.com. 3600 IN MX 10 mail.example.com.
```

### 2.1.12 Resolution of a domain name

After explaining the basic terminology around DNS, in this section we analyze the resolution process for a domain name. In the following analysis derived by [45] (fig. 2.3), we aim to resolve the IP address of www.example.com domain name. In the figure, the query is denoted as www.example.com A?. Note that, the process is akin for any DNS transaction irrespective of the requested RR type or domain name. We assume that this is the first time that the specific RR is inquired, and thus this record or other intermediate results are not already stored in the cache memory of the recursive.

1. The process is initiated by the stub-resolver issuing the query. The stub-resolver creates a recursive type query and relies on the RDNS for the full resolution.
2. Upon reception, the recursive sends an iterative type query towards the root ANS.
3. The latter entity provides a referral along with the IP address of the ANS for the next responsible domain, i.e., the .com domain.
4. Following, the RDNS sends the same query to the ANS of .com zone.
5. This one replies with the referral of example.com ANS.
6. Once again, the RDNS dispatches the request to the acquired ANS.
7. Finally, as this server is authoritative for the zone containing the requested domain www.example.com, it supplies the RR in question, namely the mapping IP address.
8. Hereafter, the RDNS is capable to provide the full answer to the end-user and
9. cache the response including all intermediate results for TTL value. Thereby, the RDNS is able to fulfill subsequent similar requests without the obligation to contact again the root or the TLD's ANS.
10. However, when the TTL value expires, the specific RR is discarded from the cache memory and the recursive should repeat the same procedure upon request.

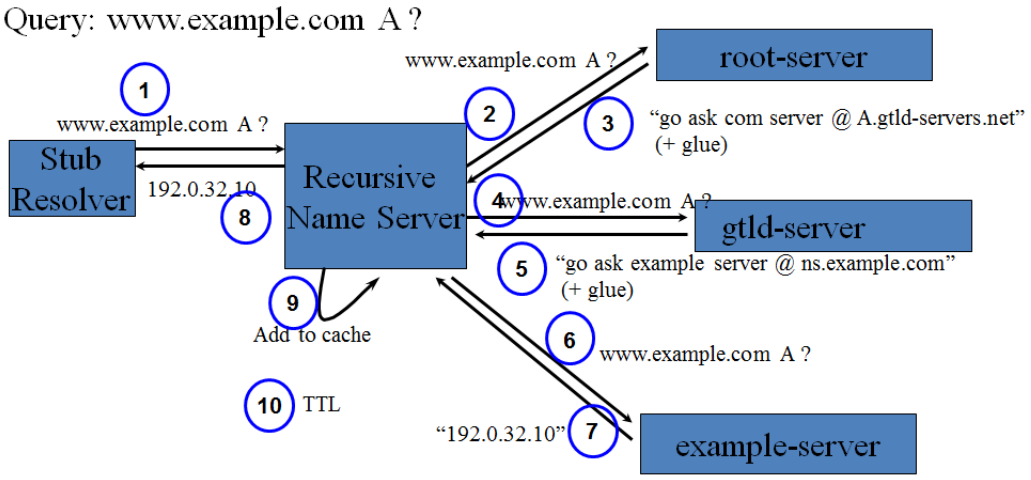



FIGURE 2.3: DNS transaction

## 2.2 Legacy DNS Attacks

It can be safely argued that DNS suffers from the same inherent problems as it did at the inception of its deployment [46, 3]. Two are the major attacks that involve DNS protocol and infrastructure. The first one threatens the integrity of the provided RR, while the second exploits DNS to unleash DDoS attacks against targets on the Internet.

DNS plays an important and crucial role in Internet communication. Normally, before any other connection, all network protocols, such as email, web, ssh, fpt, etc, before any other connection they employ DNS to acquire the necessary information for connecting to the remote hosts. Particularly, DNS resolution precedes every other communication transaction. Therefore, DNS's potential vulnerabilities could set at risk the secure operation of Internet applications and influence the perspective of end-users about the Internet [47].

### 2.2.1 Attack Surface specific to Integrity of DNS transactions

As discussed previously in section 2.1.12, the full resolution of a domain name, could divided into two distinct phases. In the first phase, the stub-resolver inquires recursively the RDNS. Then in turn, the RDNS traverses iteratively the DNS hierarchy aiming to locate the suitable ANS. For accomplishing the full resolution of a domain name however, various data flows in the DNS infrastructure take place. These internal data flows are shown in Fig. 2.4 in terms of numbered steps.

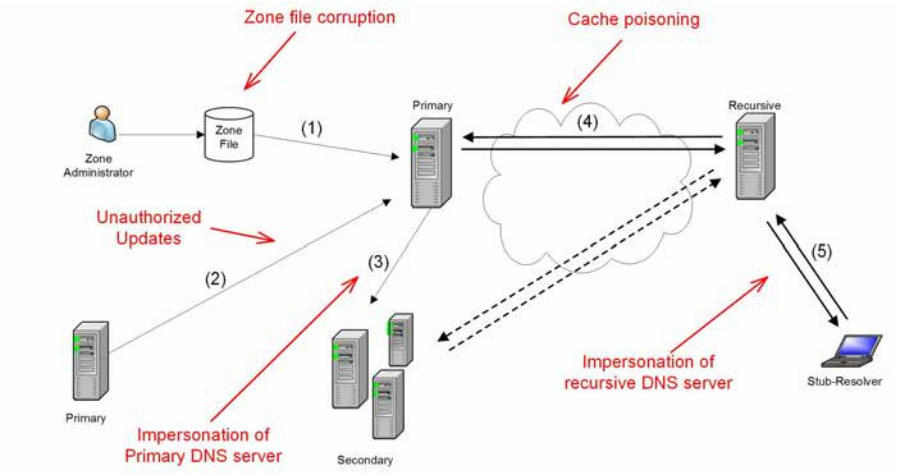



FIGURE 2.4: Attack Surface specific to Integrity of DNS transactions

The information about a zone is contained in the corresponding zone file. This file is kept in the primary ANS of the zone, and modified by the administrator whenever needed (1). At the same time, data are added dynamically from various external sources (2). The contents of the zone file are transferred to the secondary ANSs, during a zone transfer. Thus, these servers can act as ANS of the zone (3). On the other hand, the DNS recursive resolver requests information about the resources of the zone on behalf of the end-user. Note that this server cannot differentiate whether it contacts the primary or the secondary ANS (4). Finally, it provides the requested information to the stub-resolver (end-user side) who was responsible for the initiation of the resolution process (5).

Several threats are identified that aim to exploit each above mentioned process of data transfer and inject bogus DNS records. First off, an attacker could corrupt the zone file if they succeed in compromising the ANS (1). Then, the attacker could trick the primary ANS and maliciously perform a dynamic update of false records (2). Also, the aggressor may impersonate the primary ANS to any secondary and force them to initiate a zone transfer (3). Another effort to pass bogus data could target the connection in (5), where an attacker could forge the reply provided by the recursive resolver, or impersonate the recursive to the stub resolver. However, the most common target of forging attempts is the connection between the RDNS and the ANSs (4), as they reside in the open Internet and are therefore exposed to any kind of external attack.

In section 3 we are dealing with the latter case (4), where the attacker strives to delude the DNS recursive and enforce it to accept and store in its cache memory forged RRs. This type of attack takes place more often, because the communication happens through external networks, and thus it is more trivial for the aggressor to manufacture the proper DNS packet and exploit the recursive. Furthermore, the impact of the attack is more profitable since it affects all the clients served by the targeted recursive. On the downside, for the remaining attacks they are taking place inside internal networks, that are more controlled, and thus limited. The countermeasures aiming to block and deter such attacks include specific for DNS, cryptography mechanisms based on TCP protocol, such as TSIG or SIG(0) [48]. Other DNS independent protection measures could involve IPsec, TLS/SSL or SSH, which provide mutual authentication of both parts of the communication.

### 2.2.2 Exploiting DNS infrastructure for DDoS

Due to the nature of the DNS protocol, an attacker is able to exploit DNS for launching DDoS attacks. DNS amplification attack is a type of DDoS attack that combines amplification and reflection characteristics. Namely, it multiplies the attacking network traffic and simultaneously reflects the traffic by third-party servers towards the target. The amplification attribute of the attack augments the impact on the targeted victim, while the reflection obfuscates the forensic signal of the attack. In section 4, we explain the standard type of DNS amplification attack as long as we present how an attacker can enhance the effects of their attack by taking advantage of DNSSEC security extension.

# Chapter 3

# DNS Cache Poisoning

Despite the fact that DNS constitutes the cornerstone of Internet, until very recently, it lacks a mechanism to provide origin authentication of its data. This way, DNS is vulnerable to poisoning attacks. This security gap is addressed by two public key cryptographic mechanisms, namely DNSSEC and DNSCurve. In the meantime, only short-term and custom-tailored solutions were deployed for the sake of suppressing poisoning attacks.

In this chapter, we detail on DNS cache poisoning attack and explain its impact. A discussion on the cardinal interim solutions for dealing with this type of attacks also falls under the scope of this chapter. Finally, for better grasping the benefits of each proposal, the chapter includes a detailed side-by-side comparison of DNSSEC and DNSCurve based on nine distinct criteria for comprehending the benefits of each proposal.

## 3.1 Introduction

Nowadays, the majority of popular services, such as web, mail, ftp, and business critical computational resources are accessible over the Internet with the usage of DNS protocol. Hence, the potential vulnerabilities of DNS could set at risk the secure use of any Internet application that depends on the proper operation of the DNS service. Therefore, the smooth operation of DNS service constitutes a fundamental consideration for accomplishing the availability of Internet services and the connectivity to valuable resources.

Despite its crucial significance, DNS still suffers from the same vulnerabilities that it had at the beginning of its deployment. Potential attackers aim to exploit the lack of protection mechanisms and corrupt or undermine the integrity of the authoritative responses. Although the motivation for DNS-oriented attacks varies, the ultimate goal of any aggressor is to provide misleading or bogus data to the end-user. Usually, however, the attackers aim to monetary profit, e.g., by identity theft, or phishing. In such an attack incident, called *DNS cache poisoning*, the attacker tries to deceive the DNS recursives resolvers into accepting and storing forged DNS data in their cache.

Generally, DNS cache poisoning attack aims at maliciously injecting non-authoritative RRs to the cache memory of a RDNS. When a recursive resolver receives and caches such non-authentic data, it is considered as poisoned. As a result, it supplies bogus data to its clients, and thus the end-users based on these resolvers are redirected to malicious sites instead of what they have requested [47]. Currently, DNS cache poisoning attacks constitute a constant, ongoing threat to DNS infrastructure since they aim to tamper the integrity of DNS data and provide to the end-users fabricated records. As a result, successful cache poisoning could change the way the end-users experience the Internet and expose them to a variety of serious threats. For instance, consider the case where a user is redirected to a malicious site that mimics the original one. Then the attacker would be able to delude the user and collect its personal information.

### 3.1.1 Target of Cache Poisoning

As already pointed out, the target of cache poisoners are the cache memory of a RDNS. This type of attack is extremely profitable as it affects all the internal users that depend on the targeted recursive for DNS service. In the case an attacker achieves to introduce forged RR about a frequently accessed or valuable site, then all the users served by the poisoned recursive are redirected to the malevolent site instead of the original. Moreover, the attackers tend to target open recursive as it is more feasible to launch a poisoning attack against them. As in the case of a non-open recursive, the attacker needs to be located in the internal network or to control the behavior of an insider stub-resolver in order to trigger the attack.

### 3.1.2 Steps for a Cache Poisoning Attack

Normally, a cache poisoning attack unfolds in two phases [49]. Firstly, the attacker persuades the recursive to perform a DNS lookup. Next, whilst the recursive awaits for the authoritative response by the ANS, the attacker crafts a number of spoofed DNS answers that seem to originate from the ANS and immediately sends them to the recursive.

Fig. 3.1 [49] depicts a simplified view of the three DNS entities that are involved in a DNS poisoning attack incident. In the typical case, the stub-resolver queries the recursive resolver about the IP of a domain name (1). This query is notated as www.google.com A?. It is assumed that the recursive does not posses the specific RRs in its cache memory, so it needs to contact the ANS of the domain in question (2). In the case that the requested information exist in the cache, the recursive will respond directly and the attacker will not have the opportunity to launch the attack. The ANS in turn replies with the appropriate data (3). In the figure, the corresponding answer is denoted as www.google.com IN A: 64.233.167.99. Finally, the recursive sends the answer to the resolver (4) and caches the data for future use.

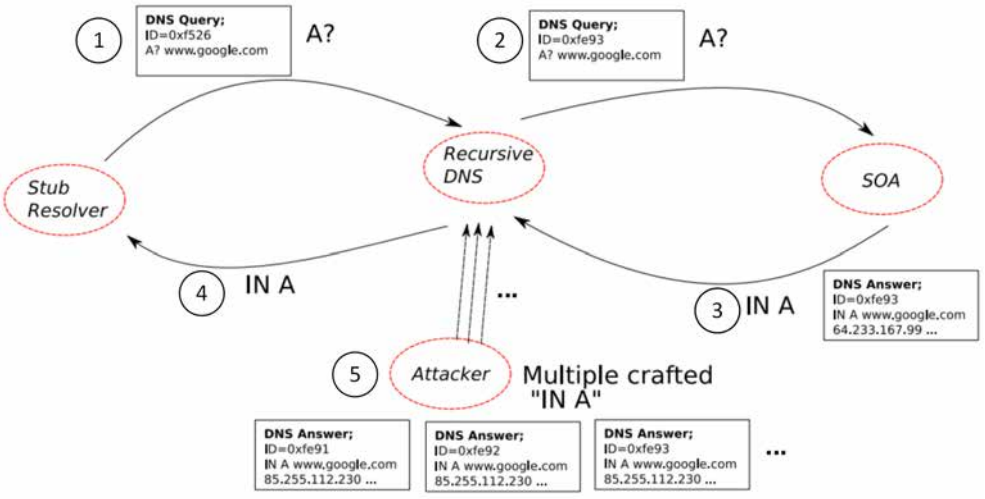



FIGURE 3.1: DNS poisoning attack

As already mentioned, the proper time for executing the poisoning attack is when the targeted domain name is not stored in the cache memory of the recursive. For this reason, the poisoner would query the recursive about the particular name in order to deduce whether it is in the cache and for how long it will reside there. After that,

the aggressor initiates the attack the moment the TTL value expires and therefore the recursive should consult the ANS again. Let us assume the hypothesis that the poisoner desires to forge the IP address of www.google.com. Initially, they begin by forcing a stub-resolver under their control to query the targeted recursive about the specific address. In this way, they initiate a DNS resolution like the aforementioned one. The time period the recursive interacts with the ANS of google.com domain, it is the perfect time for the attacker to unleash the second phase of their attack.

In the second phase, the poisoner dispatches numerous packets that seem to originate from the ANS (IP source spoofing) and hopes that eventually one of them will be accepted and stored in the recursive's cache as legitimate. In the figure, the crafted packets are illustrated as answers in the form www.google.com IN A 85.255.112.230 (5). Where 85.255.112.230 is the IP that the attacker desires to redirect all the users that request the IP of the domain name www.google.com. From the above discussion it becomes clear that it is more trivial for a potential attacker to poison an open recursive, because they can directly trigger the resolution of a domain name. Nevertheless, as already pointed out in section 3.1.1, in the case of a restricted RDNS, the poisoner has to be located in the internal network served by the recursive or control a stub-resolver inside that network.

At this point, we have to mention that a recursive accepts only responses that are correlated with pending queries and all other unexpected packets are rejected. Thus, the recursive applies the following criteria to incoming responses in order to distinguish the legitimate from the forged ones [50].

- The Question Section of the response should be identical with that of the query, due to the fact that the responding ANS copies the question section from the query to the reply unmodified.
- The TXID should be equivalent for both the request and reply packet. Therefore, the recursive checks the 16-bit TXID contained in the incoming DNS message header whether it matches with any unresolved query.
- The response packet originates from the same IP addresses to which the query was sent.

- The response packet arrives on the same IP address and port from which the query was sent.

If all of these conditions are met, the recursive will accept the message as valid and it will cache the included RRs. In any other case that the received packet is not associated with any pending query, it is discarded.

A successful poisoning attack requires the attacker to craft a DNS message that appears to come from the ANS. This will force the recursive to accept the fake message as authentic. Since normally the transaction takes place over UDP protocol, it is straightforward for the aggressor to manufacture such a message. Firstly, the fabricated reply should has the IP of the ANS as its source IP address. In the case the targeted zone maintains more than one ANS, the aggressor needs to predict which of the ANS was queried. Additionally, they need to guess correctly the transaction elements used in the recursive's query packet. That is the unique 16-bit TXID of the transaction and the port that the recursive send the query and expects the response. The domain name used in the question section is known to the attacker because they initiated the resolution. Finally, the message will contain spoofed RR. Namely, this record will have the domain name of the target, but the mapping IP address will be fraudulent.

Consequently, the attacker sends forged answers to the recursive and hopes that one of them will match with the anticipated. The more packets they manage to send the more probabilities they have to succeed. Thus, if one of these packets looks authentic and arrives prior to ANS's answer, then the recursive accepts and caches the forged RRs. The subsequent received answers about the same domain are ignored, including that from the authoritative. One can liken a DNS poisoning attack to a packet race [49]. The recursive server accepts whichever answer arrives first, as long as it matches with the query. As it is well-known, due to the nature of UDP protocol, it is common for these packets to be lost or delayed during the transmission. Hence, the pairing between outgoing queries and received responses is not always one to one. So, it is typical for a DNS server to receive unsolicited DNS messages that eventually are ignored.

Figure 3.2 [49] shows the time window for a DNS resolution to complete. We can deduce that the time interval between the moment the recursive resolver dispatches the query and the moment it receives the response by the ANS constitutes the vulnerability window

for a DNS poisoning attack. During this interval, the attacker is able to send numerous packets to the recursive hoping that one of them will match with the query. Dagon et al., [49] estimate that the time of the transaction amongst recursive and authoritative server is 100ms on average. They calculate this time window by measuring the Round-Trip Time (RTT) of sequential queries. Namely, they probe continuously open recursives for the same domain name. The first time the ORDNS contacts the ANS of that domain name, while the subsequent times it responds from its cache. Thus, the difference among these RTT values represents the time needed for a recursive to contact with the responsible ANS. This time value is crucial for the accomplishment of a poisoning attack, because the greater the more forged packets the attacker has the ability to send.

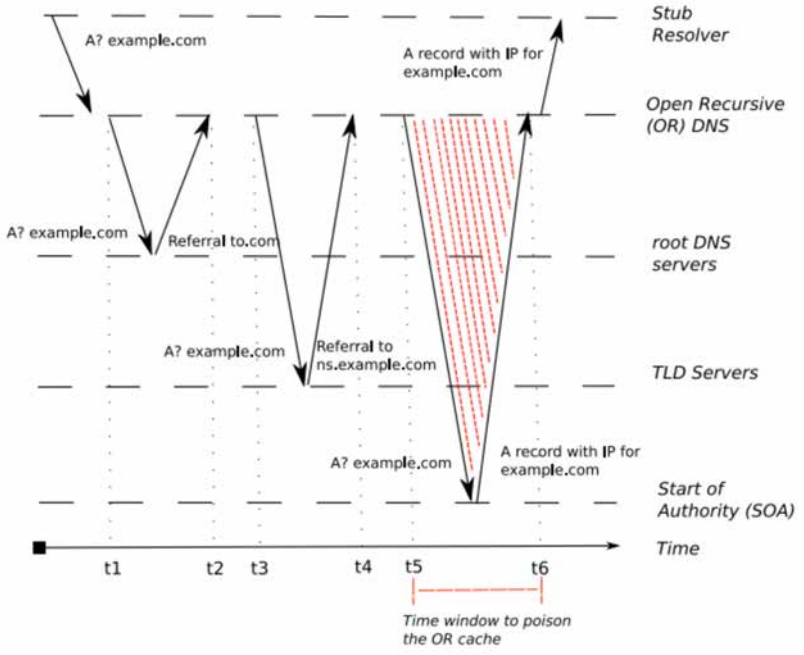



FIGURE 3.2: Time window for DNS cache poisoning attack

Equation (3.1) [50] aims to calculate the probability of a successful cache poisoning attack. Essentially, this probability is equal to the volume of the forged packets the attacker is able to send during the vulnerable time window divided by the size of the problem space, that is, the number of possible TXID and UDP port combinations [50]. The success probability $P\_s$ of a cache poisoning after $F$ spoofed DNS responses within the time window is equal to:

$$\mathrm{P\_s} = \frac{\mathrm{D} * \mathrm{F}}{\mathrm{N} * \mathrm{P} * \mathrm{I}} \tag{3.1}$$

where $N$ is the number of ANS serving the targeted domain name. The value of N is estimated around 2.5 in average [50]. The $P$ parameter is the number of the available UDP ports, which at maximum is around 60,000. This is because ports under 1024 are reserved. Also, $I$ is the number of TXID with a maximum of $2^{16}$. When the recursive has $D$ multiple outstanding queries for identical responses, each spoofed packet has a proportionally higher chance of matching any of these queries. In equation (3.1), we assume that the values TXID and UDP port are randomly selected, and thus their values are non-deterministic.

Once the matching response arrives, it is accepted and the corresponding records are cached for as long as defined in the TTL field. In the case the attack fails, the poisoner must wait until the TTL of the RR expires and the data are discarded from the cache. At this moment they have again the opportunity to attempt poisoning the same recursive for the same domain name. Therefore, in the case of unsuccessful attempt, the next retry will be possible after a time period depending on TTL, whose typical value is one or more days. Otherwise, if the attack is successful, the recursive has stored a poisoned RR with a long TTL. The choice of TTL is induced by the attacker, so it is expected that they select the highest possible value. The poisoned record resolves the targeted domain name to an IP address under the control of the attacker. Therefore, all clients of the recursive in subsequent requests for the same domain name will be redirected to the malicious site. This certainly will expose the end-users to imminent risks, such as information theft, infection by malware, etc.

In this type of attack, it is assumed that the attacker is not capable of monitoring the network traffic between the recursive and the authoritative. Alternatively, they are in position to intercept the transferred packets, they could be able to capture the query and observe the contained fields. In such case, the poisoner needs only one crafted response to poison the cache. The creation of the forged packet could be fast enough, so the fake response could arrive before the legitimate from the ANS. This kind of attack is only possible when the attacker is in position to capture the traffic amongst the two transaction entities [51]. For instance, as discussed in [52], they could compromise a router on the transit network and accomplish a Man in the Middle (MitM) Attack.

### 3.1.3 Kaminsky-Style Poisoning Attack

As already pointed out, by performig a conventional type of poisoning attack the attacker is able to modify the corresponding IP address for a domain name, and hence they would redirect the users to a site under their control. The aforementioned method requires several days or weeks to succeed in a properly configured recursive resolver. This is because the attacker should wait lengthy TTL periods after unsuccessful attempts. In August of 2008, Dan Kaminsky, a security researcher, announced a serious flaw in DNS [4], that allows a successful cache poisoning attack in significantly limited time compared to the traditional attack. By exploiting this flaw the aggressor could populate the cache with a forged A RR within approximately 10 seconds, as it bypass the TTL constrains. The deployment of the proposed type of attack is so straightforward and the effects are so devastating that its discovery was initially considered as the end of the DNS infrastructure.

Expanding the previous example let us assume the scenario where the attacker desires to poison the IP address of www.google.com domain name. They put a stub-resolver under they control to query the targeted recursive about the address of a non-existent domain name within the zone of google.com, say RANDOM1.google.com. Since this domain name is meaningless and probably does not exist, it is expected that it is not stored in the cache memory of the recursive. Therefore, the recursive is forced to transact with the ANS of google.com zone and request the corresponding RRs. Simultaneously, the attacker sends numerous spoofed packets towards the recursive that seem to originate from the ANS and they try to guess the appropriate transaction elements akin to the traditional type of attack. Normally, the ANS responds with NXDOMAIN in such a query. However, the attacker's forged responses provide a referral for the authoritative of the random name. Videlicet, the DNS messages defines into the authority section of the response the www.google.com as the ANS for the random domain (NS type RR) and into the additional section the mapping IP address (glue record) [4]. Obviously, the contained A RR points to a location under the control of the attacker. Unlike the typical poisoning attack which provides malicious data in the answer section, in this type of attack the poisoner takes advantage of the authority and additional section. In other words, the aggressor supplies a misleading A record for the forged NS record. Because

glue records are deemed more trustworthy [53, 54], it is sure to replace any existing A RR for the www.google.com name stored in the cache.

Assuming the foreseen transaction elements are valid, the recursive will store in its cache the malicious IP address for www.google.com. But if the first attempt of poisoning fails and the NXDOMAIN response from the ANS arrives first, then the attacker can immediately retry by requesting another random name in the zone, such as RANDOM2.google.com. The evildoer has the ability to continuously query fictitious random names within the targeted zone until they succeed to poison the cache. From the previous analysis, one can conclude that the considerable improvement of Kaminsky's style attack is that after a failed attempt the attacker does not have to wait until the TTL of the cached RR expires. But rather they can retry immediately. This fact undoubtedly decreases the success time from weeks to few seconds and reduces the effort of the attacker [49]. The particular flaw derives from the inherent design of DNS protocol and is not limited to any software implementation. The majority of the software products so far suffer from the described vulnerability and only few implement from scratch simple countermeasures that contribute to the prevention of the attack. For this reason, as further discussed in section 3.2.1.2, the vendors respond by releasing a patch that increments the entropy of the DNS message by introducing Source Port Randomization (SPR) [55]. This way, it is more challenging for the attacker to predict correctly the parameters of a DNS response.

Alexiou et al. [56] utilized the PRISM probabilistic model checker to formally model and analyze the Kaminsky-style poisoning attack. The authors aim to validate the existence of Kaminsky-style cache-poisoning attack and to evaluate the effectiveness of the proposed remediation, namely SPR. Indeed, the model confirms the possibility of the attack even in the presence of a poisoner with essentially no knowledge of the victim RDNS's actions. In addition, they demonstrate that the attack probability is inversely proportional to the number of the available UDP ports utilized to issue the DNS query.

## 3.2 Poisoning Antidotes

Overall, two types of antidotes are deployed to protect DNS from poisoning attacks [47]. The first one includes long-term solutions that utilize secure cryptographic mechanisms.

Such mechanisms are able to eliminate the forgery of DNS messages as they enable origin authentication. That is, with the utilization of digital signature, responses originating from non-legitimate sources will be discriminated. However, the major drawback of these solutions is that they require the partial modification or total substitution of the current infrastructure. Namely, the RDNSs and ANSs including root servers should change their functionality and be enhanced with operations of Public Key Infrastructure (PKI). It is obvious however that this process is complex and time consuming to complete. In this direction, two main solutions exist; DNSSEC, which is the most prevalent security extension of DNS, and its competitor DNSCurve. Both these solutions are presented and compared to each other in section 3.2.2. Also, in the same section, the most recent DNS over DTLS and DNS over TLS are presented. These proposals do not introduce cryptographic security to DNS protocol per se, but rather enable it to the underlying protocols, UDP and TCP respectively.

Until the global-scale adoption of DNSSEC, which will safeguard the end-users against cache poisoning attacks, interim short-term solutions have been also recommended and deployed. These methods contribute to the smooth transition to more secure architectures. Usually, they affect only the configuration of the recursive, and therefore it is much easier to implement. Mostly, interim solutions intend to increase the entropy of a DNS message. To wit, the proposed mechanisms aim to magnify the possible values a DNS message containing specific RR can take, and thus decrease the possibilities of the poisoner to achieve the attack. The concept of the specific methods is analyzed in section 3.2.1.

### 3.2.1 Interim Solutions

This section presents simple countermeasures that intend to preclude cache poisoning attacks. This type of measures are effective until the moment DNS cryptographic mechanisms will be widely adopted. As already mentioned in section 3.2 with the extensive compliance of DNSSEC, it will be impractical for the poisoner to spoof DNS messages that seem to originate from ANS, because the integrity and authenticity of the contained RRs will be protected via digital signatures. Therefore, the basic characteristic of the interim antidotes is that they endeavor to increase the entropy of DNS queries to the point that the poisoning attacks would become practically infeasible [47]. Namely these

methods propose means to augment the uncertainty or the possible values that a DNS query packet (for the same RR in question) could have. This way, an effective fabrication of a valid response would be much more laborious to be accomplished.

The basic advantage of the proposed techniques is that they are based on the standard DNS protocol and thus do not involve any extension. This implies that they do not require massive modifications to the existing DNS infrastructure and adjustments on the functionality of NSs and stub-resolvers. Generally, it is desirable for the interim mechanisms to only introduce changes to the operations of recursive resolvers. Nevertheless, whenever it is necessary to alter the functionality of a NS, it should be introduced with backward compatibility. This is because any mechanism should be optional and not disturb the normal operation of the NS [49].

#### 3.2.1.1 Transaction ID Randomization

In the initial versions of DNS software products, the TXID field was incremented deterministically. This allows the attackers to predict TXID effectively and therefore to poison the cache more easily. This is because they only need few interactions with the server in order to estimate the next TXID number. As a result, Theo de Raadt, OpenBSD developer, suggested [49] that TXID should be randomized. As TXID field is 16-bit long, it offers 16 bit of entropy or 65,536 distinct TXID values. Furthermore, the randomization should be based on cryptographically secure methods instead of techniques that that are based on weak random number generators.

#### 3.2.1.2 Source Port Randomization

An additional source of entropy could be the UDP source port from which the DNS query is issued. D. J. Bernstein firstly introduced *Source Port Randomization* (SPR) on his DNS software product, djbdns [57]. A SPR-enabled recursive dispatches his queries using at random one of the possible 65,536 ports. Thereafter, the recursive expects the matching response on the same UDP port. Certainly, not all of the available $2^{16}$ ports could be utilized since the first 1024 typically are bound by protocols (well-known ports) [58]. In a nutshell, this measure contributes approximately 14 to 16 bits of entropy.

Unfortunately, SPR may not be usable in all information systems. This applies to the case where the recursive resides behind network devices, such as routers, firewalls, proxies or other gateways that implement network and port translation (NAT/PAT). This kind of devices transparently modifies the IP address and port number in the outgoing IP packets. Thus, the PAT devices could revoke the randomness of UDP source port generated by the recursive, and this way become straightforward for the attacker to predict the source port [55]. In any case, the widely employment of SPR was the response of DNS community to the Kaminsky-style poisoning attack, as recommended in the vulnerability note of US-CERT [55].

#### 3.2.1.3 0x20-Bit Encoding

Dagon et al., [49] presented a simple and easily deployed method that could increase notably the entropy of a DNS query. This method is based on the observation that while the case sensitivity of a domain name in the question section is preserved during a DNS transaction, this is not taken into consideration during the resolution process [59]. The authoritative just copies the question section of the query to the response packet. So, the authors propose to use mixed upper and lower cases of the queried domain name as the ANS usually preserves the case pattern in the answer. Therefore, the attacker has to also predict correctly the mixed-case of the questioned name besides the remaining parameters involved in poisoning attempts. This way the proposed method will increase the difficulty of the attack. The method was named 0x20-bit encoding due to the fact that any upper and lower letter differ by the 0x20 bit in ASCII representation. Hence, the modification of a letter's case only requires the alteration of the 0x20 bit.

Generally, domain names are case insensitive, and for this reason queries for the same domain name and different case pattern will be resolved the same way. Also, for performance reasons, the majority of DNS software copy bit by bit the question section from query to response packet. In other words, a responding NS instead of creating from scratch the response, it rewrites the packet as soon as it is received, then adds the answer section and only changes the communication parameters. This way, the case pattern is preserved in the response packet.

This fact offers an opportunity for taking advantage the question section as a covert channel that could encode transactional information. The recursive could send a query

with some randomly chosen letters of the name written in capitals. The case pattern of the queried name, which is unique to every transaction among recursive and authoritative, could be used to identify the communication. All the combination of case pattern are treated as equivalent by the receiving ANS, but only one would be valid for the recursive. The number of different combinations depends on the number of letters contained in a given domain name. Obviously, only letter characters could be encoded with 0x20 mechanism, while the remaining (such as digits) should remain unaffected. Whenever the recursive receives a response for the required domain name, it inspects the encoding to verify if the message originates from the ANS or it is forged.

Figure 3.3 [49] depicts the proposed algorithm for encoding the queries according to DNS-0x20.

1. As *input* the algorithm takes the queried domain name. This could be the question section of the query by the stub-resolver or of the response by the ANS.
2. The letters of the query string are all converted to lowercases.
3. The encryption scheme is used to encipher the string and calculate a cipher block. Possibly, the keys of the scheme should alter frequently in order to avoid cryptanalytic attacks.
4. Each bit of the generated block is used to determine whether the corresponding letter would appear on the query with upper or lower case. Since the size of the block is longer than the number of "0x20 capable" characters, only the first bits from the block are read, one by one.
    - If the bit is 1, make the character upper case (i.e., calculate char |=0x20)
    - If the bit is 0, make the character lower case (i.e., calculate char &=0x20) where char represents the ASCII numerical value of the letter.

This procedure generates the 0x20-encoded domain name. In the case the packet comes from a stub-resolver is sent to the ANS. On the other hand, if it is a response is used to verify the authenticity of the answer.

The improvement of DNS forgery resistance with this method depends solely on the number of letters in the domain name. For example, for the domain name xe.gr which has

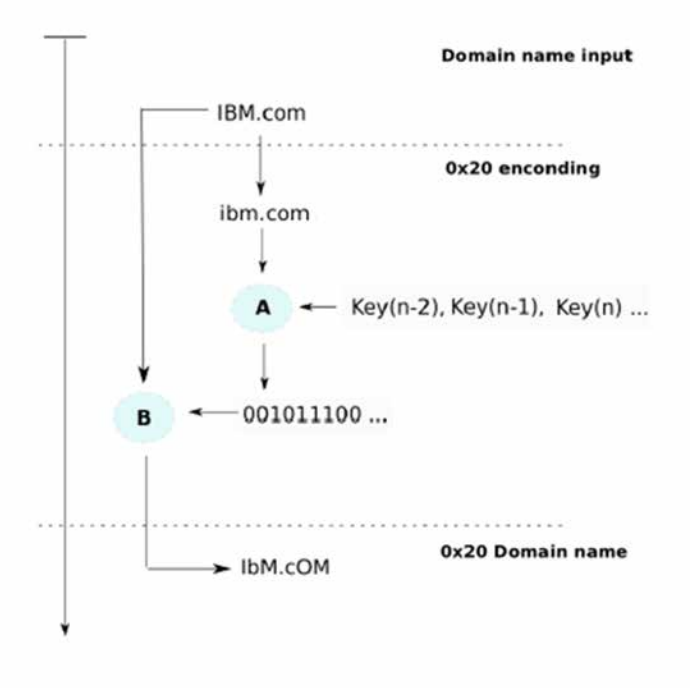



FIGURE 3.3: Algorithm of DNS-0x20bit encoding

small size it only provides 4 bits of entropy, but for the domain name www.icsd.aegean.gr adds 15 bits. Unfortunately, there are many sites that contain only few letters and some additional digits. For example, popular sites ranked to the top 500 sites on the web [60], such as 163.com, t.co or qq.com have maximum $2^5$ different combinations. The proposed mechanism makes the poisoning of shortened domain names more difficult to succeed, but certainly not impossible. Consequently, the contribution of the particular method is minimal for a significant fraction of frequently accessed domain names.

However, 0x20 encoding does not disrupt the operation of DNS protocol neither requires modification to the DNS infrastructure. It only introduces additional functionality on the side of the recursive. Thus, it provides backward compatibility for the vendors that do not choose to support.

#### 3.2.1.4 WSEC DNS

Another proposal for augmenting the entropy of DNS messages is Wild-card SECure DNS (WSEC DNS) by Perdisci et al. [61]. WSEC utilizes an one-time random number prefixed to each queried domain name. Such queries are resolved with the usage of wildcard records of CNAME RR type. As mentioned in section 2.1.11.5, wildcard domain

names are defined as domains that have on their left-most label the '*' (asterisk) symbol. This label is interpreted as any valid combination of characters. The aggressor who wants to poison the cache of a WSEC-enabled recursive, needs to guess correctly the random nonce of the query besides all the other parameters required in typical poisoning attacks.

WSEC is based on characteristics of standard DNS protocol and infrastructure, and thus it does not require changes on the existing implementation. However, it introduces modifications on the zone files of the ANS that desire to support WSEC. Namely, for each RR of any type, it inserts two RRs of CNAME type. These RRs are utilized to resolve the randomly prefixed domain name to its actual data. Whenever a user queries for a RR, a process similar to that depicted in Fig. 3.4 [62] will take place.

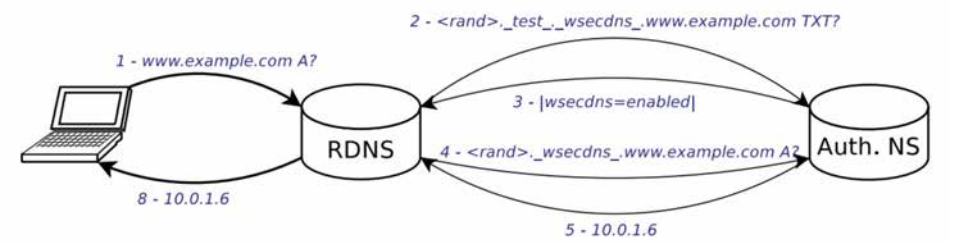



FIGURE 3.4: WSEC DNS query process

Say for instance that a stub-resolver wishes to query for the IP address of *www.example.com* (www.example.com IN A?) (1). Then, the recursive generates a random alphanumeric string <*rand*> of length N, which is distinct for each query, and queries for the TXT record of <*rand*>.*_test_._wsecdns_.www.example.com* (2). In the case the example.com zone has enabled the WSEC method and its zone file has been modified accordingly, this question will be resolved with the record

**._test_.wsecdns_.www IN CNAME _test_.wsecdns_*

Namely, the queried name is an alias of *_test_.wsecdns_example.com* and the corresponding response is the string |*wsecdns=enabled*|. On the other hand, if the zone does not support WSEC queries, then NXDOMAIN response is returned. Therefore, the recursive should continue by issuing the standard query (www.example.com IN A?). Steps (2) and (3) constitute a form of handshake between the recursive and the ANS of the zone in order to examine whether the ANS is WSEC enabled. Since the zone of our example is WSEC compatible (3), the recursive will query for the RR of type A of the domain

name <*rand*>.*_wsecdns_._www.example.com.*, which is the initial queried name with the random prefix (4). The specified name will be resolved with the wildcard CNAME RR

**._wsecdns_.www IN CNAME www*

This RR states that the random requested name is an alias for *www.example.com*. Consequently, the ANS will respond with a message that contains the random prefixed queried name, the CNAME resource record which states that the <*rand*>.*_wsecdns_._www.example.com* is an alias for *www.example.com*, and the RR of type A that provides the IP address which is initially requested by the user (5).

In order to validate the authenticity of the response, the recursive will examine if the domain name in the question section matches with the queried name, i.e., the initial name with the random prefix. If the random prefixes are not equal, then the response will be rejected as forged. Otherwise, the response originates from the ANS, and thus the recursive accepts it as legitimate. Finally, the RDNS will normalize the response by stripping out the records related with WSEC, that is, the wildcard CNAME RR, will cache the normalized record which was initially requested, and finally send back that record to the requestor. This query process is entirely transparent to the end-user. The reason for this is the normalization algorithm that clears the responses from WSEC-related data. The user will receive the correct message independent of the requested type of record.

The one-time random string, prepended to each queried domain name, consists of lowercase letters and digits. Therefore, assuming that this prefix has a length of 5 characters, the proposed method magnifies the DNS messages space by $36^5$. Consequently, the potential attacker has to make a considerable effort to guess correctly the prefixed string in WSEC transactions, besides the other parameters protecting the DNS messages. This improvement is independent of the length of the domain name or the effectiveness of any other introduced method.

From the previous analysis, it is obvious that the WSEC mechanism has the side effect of duplicating the traffic amongst the recursive and the ANS, since every query requires a preceding handshake. As a result, it consumes network bandwidth and causes delay of a query resolution. Also, it requires revisions on the configuration file of any name

server willing to support this method and needs certain additional functionality on the part of the recursive.

#### 3.2.1.5 Multiple Queries

A different approach for evading forged DNS responses is by issuing multiple DNS queries. For instance, Trostle et al. [63] proposed a DNS proxy implemented within BIND RDNS, that sends multiple DNS queries when cache poisoning attempts are detected. Specifically, the proxy listens on a number of additional random UDP ports apart from those opened from the RDNS. Whenever a forged DNS response, i.e., an unsolicited response, is received on one of the monitoring ports, then RDNS is probably suffering cache poisoning attack. During the duration of the attack for every outstanding DNS query, the proxy issues one additional query for the same domain name and RR type but with different parameters, namely TXID, source port, etc. This way, it is expected that the effort of the attacker will be exponentially harder to successfully fabricate two subsequent DNS responses. In this mindset, the RDNS caches the (non-empty) intersection of the corresponding responses.

### 3.2.2 Cryptographic Solutions

The original design of DNS protocol [64, 65] was focused on the availability of the provided information not on the protection of data authenticity. Consequently, as it is discussed in the previous sections, the initial implementations did not include cryptographic mechanisms that could ensure the validity of their data. In fact, as explained, the absence of authentication mechanisms have lured cache poisoners for subverting the legitimacy of DNS records by poisoning the cache of DNS recursives. The fact that the validity of a record lies only on transaction parameters, that could trivially forged, and not on cryptographic operations facilitate the attainment of the attack.

Nowadays, this security gap is attempted to be addressed by two cryptographic mechanisms: DNSSEC and DNSCurve. They both utilize PKI and aim to extend the core DNS protocol. The first mechanism was initially proposed in 1997 with RFC 2065 [12]. Since then, several modifications have been introduced that lead to the current version of the security extension, known also as DNSSEC-bis. This new version is mainly described

in RFCs 4033 [14], 4034 [15] and 4035 [16]. On the other hand, DNSCurve was lately proposed [66, 67] by Daniel J. Bernstein and is based on Elliptic Curve Cryptography (ECC). Essentially, it constitutes part of an effort to deploy public key cryptography (and specifically ECC) in every Internet protocol. This effort's purpose is to secure the confidentiality and integrity of all transactions on Internet based on ECC. Unfortunately, so far DNSCurve has not been documented in detailed, it is only implemented by a handful of vendors [68] and introduced in a limited scale. Therefore, its effectiveness cannot be examined in practice but only theoretically. Until now, only one internet draft has been published that provides a general overview of DNSCurve [69].

The following analysis aims to provide a comprehensive and constructive comparison between the aforementioned security mechanisms (extensions). Towards this direction, we theoretically cross-evaluate and assess the benefits and drawbacks of each particular mechanism based on several distinct criteria. This is prerequisite in order to decide which mechanism is the best fit for each particular deployment.

### 3.2.3 DNSSEC

The Domain Name System Security Extensions (DNSSEC) constitutes a set of specifications aiming to protect certain aspects of DNS protocol. The introduced cryptographic mechanisms satisfy the following security requirements on behalf of the DNS end-users:

- Data origin authentication (of DNS messages).
- Data integrity (of DNS messages).
- Authenticated denial of existence (of DNS RRs).

For accomplishing the aforementioned requisites certain modifications to the existing DNS specifications and infrastructure are introduced. Moreover, DNSSEC requires the proposition of four new RR types, these are the DNS public key (DNSKEY) [15], the resource record digital signature (RRSIG) [15], the delegation signer (DS) [15] and the next secure (NSEC[15]/NSEC3[70]). Furthermore, DNSSEC requires the EDNS0 extension mechanism [71] in order to support hefty DNS packets resulting from the use of the DNSSEC-related RR. Finally, it needs the utilization of new flags of DNS packet header

that are used to indicate the support of DNSSEC and the mode of verification for the DNSSEC responses.

DNSSEC is mainly described in RFCs 4033 [14], 4034 [15] and 4035 [16]. RFC 4033 declares the DNS security requirements [14]. RFC 4034 presents the required new RR types [15]. While in RFC 4035 are clarified the necessary modification for the DNSSEC protocol [16]. Furthermore, RFC 5155 defines an alternative RR type, the NSEC3, which intends to replace NSEC RR type [70]. NSEC RR type allows zone enumeration and for this reason is considered as a threat to the confidentiality of a zone's contents, as we will see further down in section 3.2.3.2.

#### 3.2.3.1 Security Operations

Every DNSSEC-enabled zone possesses a pair of public and private key. With the private key of the zone a digital signature is generated for each authoritative RRset contained in the zone. These signatures are placed within the zone file in the form of RRSIG record type. As a result, whenever a properly configured resolver, namely *security-aware resolver*, requests a RR, it receives the corresponding signature along with the desired RR. This way, the resolver has the ability to verify the authenticity and the integrity of the response and to perceive whether it is fabricated or not. Certainly, in order to prove the validity of the records, the resolver should hold the legitimate public key of the zone. The public key of a zone is available to the clients in the form of DNSKEY record type. On the opposite, a security oblivious client, a client that cannot support DNSSEC, has not the capability to verify the DNSSEC-related RR. Therefore, such a client is unable to discriminate between authentic answers and forged ones.

A security-aware resolver holds the public key of a zone built-in in its configuration, as *trust anchor*, or obtains it through a DNS transaction. In the first case, the client trusts unconditionally the configured public key, while in the latter it should verify its authenticity. In such a case, the resolver consults the parental zone of the zone that provided the public key. Eventually, the resolver tries to construct an *authentication chain* from a recently learned public key to a formerly verified public key. For this reason, any client should be configured with at least one trust anchor, which is the public key of the root zone.

Finally, DNSSEC supports the *authenticated denial of existence* with the use of NSEC record type. NSEC allows the authentication of negative answers, which normally return a NXDOMAIN or NODATA response in typical DNS protocol. Without authenticated denial of existence, an attacker could persuade a client to accept a forged response with referral for a domain which actually does not exist. Moreover, the attacker will not include a DS RR, so the victim will apprehend that the domain does exist and is unsecured. A NSEC record permits a security-aware resolver to validate the negative answer for a non existing name or type. The NSEC RRs form a chain, as each record points to the next existing name sorted in a canonical order. So, this chain shows the gaps, i.e., the missing and non-existent domain names, and denotes the types of records belonging to an existing name. Every NSEC RR is accompanied with a RRSIG RR, and thus it is validated the same way as any other RR.

Summarizing the operations of a security-aware resolver, we can denote that this entity is able to:

- Authenticate that a DNS RRset is actually contained in a specific domain zone.
- Verify the integrity of a message, so as to ensure that it is not modified during the transmission.
- Verify that in the case of a negative response, then indeed the queried RR does not exist.

Finally, the specification of DNSSEC mechanism determines the subsequent four states that a validating resolver can categorize the received data [14]:

- Secure - the resolver has a trust anchor and a chain of trust, so it can verify all the signatures in the response packet.
- Insecure - the resolver has a trust anchor and a chain of trust, but at some delegation point there is an authenticated denial of existence of a DS record. This fact designates that subsequent child zones are unsigned and therefore insecure.
- Bogus - the resolver has a trust anchor and indications that the data are signed, but the answer fails to authenticate for some reason.
- Indeterminate - there is no trust anchor for the domain.

#### 3.2.3.2 DNSSEC-related RR types

In order to accomplish the security operation described in the previous subsection, DNSSEC mechanism needs the introduction of four new RR types. Subsequently, these new types of RR are described in RFC 4034 [15]. All the following paradigms of DNSSEC-related RR types have been generated with dnssec-keygen [72] and dnssec-signzone tool [73] both part of BIND DNS software.

**DNSKEY Record**

The DNSKEY RR type advertises the public key that would be used for the verification of signed data in the zone file.

| **Name** | **TTL** | **Class** | **RR** | **flags** | **protocol** | **algorithm** | **key-data** |
|---|---|---|---|---|---|---|---|
| example.com. | 3600 | IN | DNSKEY | 256 | 3 | 5 | ( |
| | | | | AwEAAeujRvs5RhgUDXwf0m/pIHtlTX8IhHB6 | | | |
| | | | | soYpBBP7TAg18StP27rTZKTG4sbK3MqwlQb1 | | | |
| | | | | VAZJgeaCJleK16LBfKBkUIw6dRXVI6G5rUlJ | | | |
| | | | | 2GAg7ydk0OPQCRRUqvP7tN1PEapeCQxGv/6o | | | |
| | | | | ia9eQJn9w7vK1s+IqnhUfEK1/y4jGG2d | | | |
| | | | | ) ; key id = 64677 | | | |

The public key of the paradigm belongs to *example.com* zone. The private part of this key, used to create the appropriate digital signature, should be stored in a safe place with limited access. The *flags* field a 16-bit length. Bit 7 of this field determines the zone key flag. This implies that if it is on, then the RR contains the DNS zone key and the name field must have the zone name. In the case this bit is off, the RR carries some other type of DNS public key that should not be used to verify signatures. Bit 15 is the Secure Entry Point (SEP) flag. If it has the value 1 (in this case also the bit 7 must be 1) the RR encloses a key for use as a secure entry point. This also means that the corresponding public key should be advertised in the parental zone (in order to delegate the chain of trust) or will be used as the beginning of the trust [74]. All other bits (0-6, 8-14) of the same field are reserved and must be zeroed. Thereupon, the possible values of the field in decimal representation are 0, 256 and 257. A zero value indicates that this

is not a DNS zone key and therefore should not be used to verify signed zone's data. The value 256 implies that the key is a *Zone Signing Key (ZSK)*, while the value 257 indicates that the key is a *Key Singing Key (KSK)*.

The private pair of ZSK is used for the digital signing of the RRs contained in the zone, including DNSKEY RR itself. On the other hand, the private pair of KSK is used to create only the digital signatures of the zone's keys (both ZSK and KSK). Also, this kind of key is either used for the generation of the DS type record, which will be published in the parental zone as part of the trust chain, or is configured in security-aware resolvers as trust anchor of the zone. Although, the same key could be used as ZSK and KSK, the deployment of separate keys is recommended [70]. As one can observe, KSK is used for the authentication of the DNSKEY records and therefore its role is more crucial than that of ZSK. For this reason, it is suggested to have larger size in order to be more resistant and require renewal less often (longer key effectivity period).

The next field, *protocol* field, takes only value 3. All other values are invalid and the record should not be used for verification in the case it has other value than 3. The *algorithm* field specifies the algorithm of the public-key cryptographic mechanisms utilized in the particular zone. There is a number of cryptographic algorithms available to be used in DNSSEC. The complete list along with the corresponding values are detailed in section 3.2.3.2. The last value of DNSKEY record type, *key-data* field, includes the base64 representation of the public key of the zone.

#### RRSIG Record

The Resource Record Signature (RRSIG) type contains the digital signature of a unique RRset. The latter consists of all RRs with the same name, class and RR type. In other words, an RRSIG RR authenticates all the records that have the same name and type. In the subsequent example both the RRset (in this case constituted by only one RR) that is protected and the corresponding RRSIG record are represented.

| **Name** | **TTL** | **Class** | **RR** | **type-covered** | **algorithm** | **labels** | **ottl** | **expire** |
|---|---|---|---|---|---|---|---|---|
| | | | | inception | key-tag | signer-name | signature | |
| www.example.com. | 3600 | IN | A | 192.168.254.7 | | | | |

```
www.example.com. 3600 IN RRSIG A 5 3 3600 20160210145637 (
                                20160111145637 64677 example.com.
                                Ig2ED4UOxdXJEmTJ7AqBF7MiSBmVWeO0d1Pp
                                9pheb8QlcGJoTnkYxG/x7/4lhKHdnFkXoLWm
                                ZEDd/pAsbHx70IjPMmSs1xwgtclmnZo75QV6
                                5KTZxiJxLDGuCOm4lbKN8XDOxrlN64C5yKEa
                                xk/kRrlgvNDe1fj1mNkM+61/wLE= )
```

The *type-covered* field identifies the type of the RRset that is protected by the signature of this RRSIG. In this example, the RRset is of type A (IPv4 address). There should be separate RRSIG RR for each different type with the same domain name.

The next field, *algorithm*, determines the cryptographic algorithm which is used to produce (and should be used to verify) the signature. This field has identical behavior as that of the respective field of DNSKEY type. As it is explained in section 3.2.3.2, the value 5 indicates that the signature is created with the RSA-SHA1 algorithm.

The *labels* field provides the number of the labels that the initial domain name in FQDN form of the RRset consists of. This number is used during the validation process in order to specify whether the response was synthesized from a wildcard RR, and therefore to determine the name of the record used in the creation of the signature.

Following, the *original TTL* field defines the TTL of the protected RRset's as stated in the original zone file prior to the signing process. This value is vital, because the TTL value of the cached records stored in the recursive is constantly modified. For this reason, it is infeasible to calculate the valid signature of the RR without the ottl value field.

The two subsequent fields define the *signature expiration* and *inception* respectively. This way is determined a validity period for the contained signature. As a result the corresponding RR should not be used for authentication prior to the interception and afterwards the expiration time. In their textual form these fields have the format YYYYMMDDHHMMSS, where YYYY represents the year, MM the month, DD the day, HH the hour, MM the minute, and SS the second. As we can observe, the specific RRSIG

record becomes valid the 11th of January of 2016 at 14:56:37 and it expires on the 11th of February of 2016 at 14:56:37. Usually, the inception time is the time when the signature is generated, and the default period of validity is thirty days. After the expiration time the zone file should be signed again in order to be verifiable.

The *key-tag* field stores the key-tag of the DNSKEY that should be used to validate the signature. Since multiple DNSKEY RR may be present in a single zone file, e.g., ZSK and KSK, this tag is used to determine the correct one. The tag is created by the dnssec-keygen tool during the creation of the public and private key and corresponds to the DNSKEY example given in section 3.2.3.2.

The *signer-name* field declares the owner's name of the DNSKEY which its private key was utilized to produce the contained signature, in our example the owner of the public key is example.com.

Finally, the last and most important field is the digital *signature* itself, which is represented in base64 format. The signature is generated by firstly calculating a hash of the RDATA of the RRSIG record, except off course the signature, and the covered RRset. Then the hash value is encrypted with the signer's private key. The digital signatures for the zone's records are created with the execution of the dnssec-signzone tool during the signing process of the zone.

#### NSEC/NSEC3 Record

The Next Secure (NSEC/NSEC3) RR type is used to provide authenticated denial of existence for records. Namely, this kind of RR is returned when a non-existing domain name or RR's type (for existing domain name) is requested. Typically, an ANS will return a NXDOMAIN or NODATA response respectively in standard DNS protocol [38]. Upon reception, the security aware resolver utilizes NSEC response in conjunction with the corresponding RRSIG to prove the non-existence of the requested information.

| **Name** | **TTL** | **Class** | **RR** | **RR-list** |
|---|---|---|---|---|
| example.com. | 3600 | IN | NSEC | ftp.example.com. NS SOA MX TXT RRSIG NSEC |

As we observe, the NSEC RR firstly points to the next existing name in the canonical ordering of the zone file, that contains either authoritative data or a delegation point, i.e., RRs of NS type. The canonical order is defined as the sorting of the domain names based on their alphabetical order of the most significant label. Firstly, the domain names are sorted according to their most right label, then to the second most right label, and so forth. Finally, the record lists the types of RR that the domain name of the record also holds.

Every domain name has a respective NSEC record that points to the next authoritative name in the zone. The last one record of the zone file points back to the owner of the SOA RR, thus creating a circle. Consequently, a chain of valid domain names is formed and any other domain name not contained to this chain does not exist in the zone. Similarly, any other type not listed in the record list does not exist for this name.

From the preceding example, one can perceive that for the example.com domain name the zone file includes the NS, SOA, MX, TXT, RRSIG and NSEC RR types, listed in *RR-list* field. Also, the subsequent authoritative record has the name ftp.example.com. Therefore, for this example zone, it is certain that firstly there is no other record with a name between example.com and ftp.example.com in canonical ordering (for instance with domain name aaa.example.com), and secondly there is no record with other type for example.com (for instance of type A). The NSEC RR are created automatically during the signing process of the zone with the help of the dnssec-signzone tool.

However, the proposition of NSEC type produced a new security vulnerability, the *zone enumeration*, also known as *zone walking*. Generally, DNS deployment does not concern about the confidentiality of a zone's data since its typical functionality is as directory of the contained hosts. Yet a client should know the name of the host in order to request the relative RR. So, a potential attacker has to exhaustively investigate the possible domain names, aiming to find out the host within a zone and their corresponding types of RR. Although with the adaptation of DNSSEC and the usage of NSEC type such discovery is almost straightforward. A NSEC RR links two consecutively names (in the canonical sorting) in order to prove that intermediate domain names do not exist. Thereupon, for retrieving the zone's data, the attacker only has to follow the chain created with this type of records. This way, the attacker is able to acquire valuable information about the structure of a targeted network. The specific enumeration is not an attack on the DNS

protocol itself, but rather is a threat to the privacy of the zone's data [14]. The main issue of NSEC RRs type is that the records are calculated and signed beforehand and each negative answer is independent of the request. For instance, if it is applicable to generate the negative responses on the fly, it will prevent the zone walking. RFC 4470 [75] suggests a scheme that discloses the minimum possible information about a DNSSECf-enabled zone. Namely, a new RR of NSEC type is constructed for each negative answer, which indicates a gap between two random names that falls lexicographically before (but after any existing) and after (but before any existing) the queried domain name [76]. This RR is signed on-the-fly and returned to the requestor. Obviously, the scheme expects the online storage of the private key.

The response to the above mentioned threat is the introduction of a new extension called DNS Security (DNSSEC) Hashed Authenticated Denial of Existence, specified in RFC 5155 [70]. With this mechanism, the DNSSEC-enabled name servers should respond with an NSEC3 record instead of a NSEC, whenever a requested RR does not exist. The NSEC3 type provides also authenticated denial of existence for DNS RRsets, but additionally does not permits zone enumeration.

Next, we can see an example of NSEC3 RR that corresponds to the domain name example., as described in [70]:

| **Name** | **[TTL]** | **[Class]** | **RR HashAlg Flags Iteration Salt NextHashed RR-list** |
|---|---|---|---|
| 0p9mhaveqvm6t7vbl5lop2u3t2rp3tom.example. | | | NSEC3 1 1 12 aabbccdd ( 2t7b4g4vsa5smi47k61mv5bv1a22bojr MX DNSKEY NS SOA NSEC3PARAM RRSIG ) |

As observed, the NSEC3 RR has as domain name the cryptographic hash value of the original owner name concatenated with the zone name. Furthermore, it points to the next hashed owner name sorted in hash order. This sorting is defined as the order in which hashed owner names are arranged according to their numerical value. So, all domain names with hashes between the owner name and this value do not exist. Additionally, the RR indicates the records' types with the same original owner name. Finally, it defines which hash function is used to generate the hash, the salt value

concatenated, and how many iterations of the hash function are performed over the original owner name.

After the RR's type follows the *Hash Algorithm* field, which identifies the cryptographic hash algorithm that was used to calculate the hash value. The possible values for this field are

- 0 : Reserved.
- 1 : SHA-1.
- 2-255 : Unassigned.

Following is the *Flag* field that contains 8 one-bit flags. These flags can be used to indicate different processing. Currently, the only flag defined is the Opt-Out one, which notifies if the NSEC3 record covers unsigned delegations. Putting it another way, the gaps indicated by NSEC3 records could contain existing but unsecured subdomains. The usage of the specific flag facilitates the modification of unsecured delegations points without re-adjustment of the gaps.

Next is the *Iteration* field, which declares the number of times that the hash function is applied over the owner name. The multiple iterations aim to increase the strength of the generated hash value against dictionary attacks. Definitely, the additional calculations of the hash function raise the computational load for both the server and the security-aware client.

The *Salt* field provides the salt value that is concatenated to the original owner name before the calculation of the hash. This field is represented as a sequence of hexadecimal digits. As with password security, the usage of the salt value intends to protect against pre-computed dictionary attacks. So, is suggested to rotate frequently.

The *Next Hashed* owner name contains the next (hashed) existing domain name in hash order. This field holds the hash of the owner name that immediately follows the owner name of the NSEC3 record. Obviously, the last NSEC3 RR points back to the first hash name in the zone. Finally, the last field carries all the types with the same original domain name as this NSEC3 RR.

The NSEC3 RR is accompanied with the corresponding signature (RRSIG) and thereby it provides proof of non-existence. But instead of containing the domain name directly, which enables zone enumeration, NSEC3 record employs a cryptographically hashed value of its name. This hash value is computed with multiple iterations and the use of a salt. As already pointed out, the salt increases the resiliency against attackers who utilize pre-computed dictionary attack (rainbow table), while additional iterations complicate the execution of dictionary attacks. Nevertheless, the additional calculations raise the computational cost, both at server and client side.

Also, RFC 5155 [70] determines the NSEC3PARAM RR type. This type contains the NSEC3 parameters, such as hash algorithm, flags, iterations, and salt, that is used by authoritative NS to create the hashed owner name. The presence of a NSEC3PARAM record indicates that the contained parameters may be used to determine the proper NSEC3 RRset for negative responses. But by any means this type of RR should not be used by security aware resolvers. The corresponding NSEC3PARAM RR for the previous example is [70]:

| **Name** | **TTL** | **Class** | **RR** | **HashAlg** | **Flags** | **Iteration** | **Salt** |
|---|---|---|---|---|---|---|---|
| example. | 3600 | IN | NSEC3PARAM | 1 | 0 | 12 | aabbccdd |

### DS Record

A record of Delegation Signer (DS) type is placed in the parental zone file in order to create a *chain of trust* from the parental toward the child zone. DS RR holds the hash, also called digest, of the (KSK) DNSKEY RR of the child zone and is used for the authentication of its zone's public key. Actually, this record could be considered as a fingerprint of the child's KSK key and accompanied with the related RRSIG RR (which is signed with the parent's private key) constitute a form of certification of the child zone's public key. Both records' types, DS and DNSKEY, have the same name but are located in different places. The DS RR is authoritative data of the parental zone's file, while the corresponding DNSKEY is published in the child zone's file. So, along with the NS RR of the child zone the parental zone also holds its DS RR with the aim to form a trusted point of delegation.

**Name [TTL] [Class] RR key-tag algorithm digest-type digest**

; RRs in parental's zone file

secure.example.com. NS ns.secure.example.com.

secure.example.com. DS 35761 5 2 (
06E284B0532C6D1AF3587A12AF4B8BF47B94
99C7822108C8E393EEFDA59E4D16)

---

; the corresponding RR in child's zone file

secure.example.com. DNSKEY 257 3 5 (
AwEAAdzyF96pzF8bxiIqjmKc15kYoStbcsmB
O5Pf7iTclAEYe7HomXR2K4vQaWb9fyeIbjAZ
tglNTrDQqIL2cxLcVWjl6itsWy5vBHoF7R1l
GmIQ3dGW1DdGcqenOugaDY0Ok1RJ7P2pVP36
4H/c6lyWIqEi+phKqnaKaz+qKpka/ScN
) ; key id = 35761

As noticed in the paradigm above, the DS RR specifies on the respective *key-tag* field the tag of child zone's public key. This is necessary in order to distinguish the appropriate key, as more than one DNSKEY RR could exist in the child zone. Furthermore, the RR lists the algorithm of the cryptographic function (*algorithm* field) of the public key and the algorithm of the hash function (*digest-type* field) used to calculate the digest. The values for the algorithm field of this record are identical with that of the DNSKEY and RRSIG records (see section 3.2.3.2. On the other hand, the possible values for the hash algorithm are listed below [77]. Both these hash functions are mandatory for the authoritatives to implement.

- 0 : Reserved.
- 1 : SHA-1.
- 2 : SHA-256.
- 3-255 : Unassigned.

Finally, DS supplies the *digest* of the DNSKEY RR that is referred to. The digest is represented as a sequence of hexadecimal digits, while its size depends on the digest algorithm. Consequently, from the example, we notice that the child zone secure.example.com has a public key of the cryptographic algorithm RSA-SHA1 with tag 35761, which is advertised with the relevant DNSKEY RR in the child's zone file. Also, the digest of the DS record was created using the SHA-256 hash function.

A DS RR could be optionally created during the signing process of the child's zone with the use of the dnssec-signzone tool. Then, the resulting record should be transferred and inserted to the parental's zone file which in turn should be resigned.

### Cryptographic Algorithms for DNSSEC

So far, there have been a number of cryptographic algorithms that have been adopted for usage in DNSSEC. Their detailed list including the corresponding field value is provided thereunder:

- 0 : Reserved [15].
- 1 : RSA-MD5 is specified in RFC 2537 [78] and deprecated with RFC 3110 [79, 80] due to security weaknesses of MD5 hash algorithm.
- 2 : Diffie-Hellman is specified in RFC 2539 [81]. Diffie-Hellman keys are not used for signing purposes, but for facilitating other security operations.
- 3 : DSA-SHA1 is specified in RFC 2536 [82].
- 4 : Initially, the value 4 was reserved to indicate the usage of ECC, which was not implemented at that time (placeholder). However, RFC 6725 determines this value as reserved [83].
- 5 : RSA-SHA1 is specified in RFC 3110 [79] and is mandatory for the ANS and the security aware recursives to [80].
- 6 : DSA-NSEC3-SHA1 is specified in RFC 5155 [70] and is an alias for algorithm 3, with only updates the support of NSEC3 record type.
- 7 : RSA-NSEC3-SHA1, also specified in RFC 5155 [70] and similarly to 6 is an alias for algorithm 5, with only updates the support of NSEC3 record type.

- 8 : RSA-SHA256 is specified in RFC 5702 [84] and incorporates the employment of SHA256 hash function.
- 9 : RFC 6725 determines value 9 as reserved [83].
- 10 : RSA-SHA512 is specified in RFC 5702 [84] and incorporates the employment of SHA512 hash function.
- 11 : RFC 6725 determines this value also as reserved [83].
- 12 : ECC-GOST. RFC 5933 describes how to utilize the GOST R 34.10-2001 and GOST R 34.11-94 algorithms in DNSSEC [85]. GOST algorithm are based on ECC.
- 13 : ECDSA-P256-SHA256. RFC 6605 describes how to utilize the Elliptic Curve Digital Signature Algorithm (ECDSA) with curve P-256 in DNSSEC [86]. As implied from its name, ECDSA algorithm is based on ECC. ECDSA key of 256 bit size is equivalent to RSA key of 3072 bit.
- 14 : ECDSA-P384-SHA384. RFC 6605 describes how to utilize the ECDSA with curve P-384 in DNSSEC [86].
- 252 : Indirect. Reserved for indirect keys, i.e., keys that are stored in different places [15, 13].
- 253 : Private (testing purposes) [15].
- 254 : Private (testing purposes) [15].
- 255 : Reserved [15].

As concerns the remaining values for algorithm field, 15 to 122 are available for assignments by IETF standards Action [83], while 123 to 251 are determined as reserved [87].

#### 3.2.3.3 DNSSEC-related header flags

From the previous paradigms, we can deduce that the size of a DNSSEC packet containing these new introduced records is considerably increased. Since the standard

UDP-based DNS payload is limited to 512 bytes (excluding IP and UDP headers) [2], it is necessary for DNSSEC protocol to support EDNS0 extension. Otherwise, the messages most likely will be received truncated, and the client should retry the request using TCP. The Extension Mechanism for DNS (EDNS0), firstly defined in RFC 2671 [71] and redefined in RFC 6891 [88], allows the usage of DNS UDP packets with larger payload than 512 and up to 4096 bytes [88]. Consequently, messages containing DNSSEC-related RR, such as RRSIG and DNSKEY, which are sizeable enough could be received intact by the security aware clients.

Furthermore, DNSSEC employs DNSSEC OK (DO) header bit of EDNS0 extension, as specified in RFC 3225 [89] in order to advertise the ability of a client to handle DNSSEC-related RR. Setting the DO bit in a query declares to the receiving NS that the client desires to accept such records. Hence, a NS supporting DNSSEC is forced to include DNSSEC-related RR in the response. This way, the DNSSEC-enabled NS avoids to send meaningless data to clients not supporting the security extension, and hence they increase the performance of DNSSEC usage [16, 89].

Finally, DNSSEC adds two new DNS header flags, namely Authenticated Data (AD) and Checking Disabled (CD) [14, 90]. Both these flags are part of the standard DNS header and they are allocated from previously unused space. Their purpose is to be used in the transactions between a resolver and a recursive server, that both support DNSSEC [16]. The AD flag is controlled by NSs. A NS sets the AD bit in the responses when it successfully verifies the authenticity of all the contained RRsets, including NSEC records. A NS failing to verify the returning RRs should set this flag to zero. The usage of this flag is for the clients that desire to query a DNSSEC supporting NS, but they are unable to validate the authenticity of its response. As a result, they authorize the corresponding recursive to authenticate the data for their behalf. On the other hand, the CD bit is controlled by the resolvers. When a resolver sets this flag in a query, then it forces the security aware recursive not to validate the RRSIG records of the answer. The meaning of that action is that the requestor wants to perform the validation by itself according to its local security policy. It therefore undertakes the responsibility for the validation process and the NS does not need to verify the authentication of the RRs.

#### 3.2.3.4 DNSSEC in action

Now that we have seen the basic components of DNSSEC, we can analyse the procedures of the security aware ANS and RDNS. The following description is based on the most simplistic example of a DNSSEC transaction between a client and a server. Algorithm 1 illustrates a pseudocode of the verification procedure based on [91], while table 3.1 clarifies the symbol utilized in the algorithm. The exact same procedure takes place when more than one ANSs are involved during the resolution.

| Symbol | Description |
|---|---|
| r | A RR |
| k | A DNSKEY RR |
| $s^k$ | A RRSIG verifiable by k |
| R | A RRset defined as ($r_0$,...,$r_i$,$s_0$,...,$s_j$ |
| $R^s$ | A secure R |
| K | A DNSKEY RRset |
| $T^a$ | A k trust anchor |
| z | A secure zone containing only secure R and corresponding K |
| $N_i$ | The set of NS for zone $z_i$ |
| $Z^s$ | The set of all secure zones |

TABLE 3.1: Definition of symbols for DNSSEC Resolution Algorithm

**Algorithm 1** : Resolution Algorithm for DNSSEC

**Input:** Query (Q) "www.example.com IN A?" for zone example.com ($z_i$)

```
if get K_i from N_i then
    if able to create chain of trust from known T^a then
        send Q to N_i
        if ∃ R^s ∈ z_i such that Q ∈ R^s AND verify signatures in R^s against K_i then
            Data is verified
        else
            Unable to verify data
        end if
    else
        K_i can not be verified as to z_i
    end if
else
    No data for z_i can be verified
end if
```

Let's say that the RDNS desires the A RR's type of *www.example.com* domain name. We assume that it has already located the ANS of this zone and that all intermediate ANS are DNSSEC-enabled. Furthermore, all the following queries have the DO flag equal to one, so the receiving NS can perceive that the requestor supports DNSSEC, and hence it includes the RRSIG RR along with the requested RR types. This record represents the digital signature of the answer's RRset. Firstly, the recursive queries

for the DNSKEY of the zone. If the answer does not contain any DNSKEY, then the records of the zone cannot be verified. Afterwards, the resolver tries to create a chain of trust from a known (and trusted) DNSKEY towards the public key of the example.com zone. That is achieved through the DS RR (fingerprint of the public key) pointing to the DNSKEY record of example.com. This RR is stored in the zone file of the parental zone, i.e., .com zone. Hence, the recursive requests from the ANS of .com zone the DS record with domain name example.com. The answer from this server contains the DS record along with the relevant RRSIG RR, which is signed with the private key of the .com zone. Once again, for the recursive to confirm the validity of the signature of the DS RR it needs to possess the authenticated public key of its parental zone. That is why subsequently it queries for the DNSKEY RR of .com. This DNSKEY record is certified by a DS RR in the root zone file, which its authenticity could be proved with the root's public key. Certainly, the root's public key is known and should be hardcoded on every recursive. Of course, in the case the recursive has already in its cache a verified copy of the required RRs, for instance, the trusted DNSKEY of the parental zone, it utilizes the cached copy instead of traversing the DNS hierarchy again.

Since the recursive accomplishes to verify the authenticity of the zone's public key, it can now make the initial query. Following, it has to verify the accompanying signature with the authenticated public key of the zone. To do so, it matches the key-tags (of DS and DNSKEY RR) in order to locate the proper key, as example.com could have multiple DNSKEY RRs. In the case the RRSIG is unable to be verified, this is an indication that the data have been modified during the transmission.

The overall concept is that the DNSKEY RR of the root zone would be distributed in a secure manner beforehand and stored at the security aware resolvers [92]. So, as we notice in Fig. 3.5 every client possesses the public key of the root as a trust anchor and it therefore is able to authenticate any zone's key downwards the DNSSEC hierarchy. The root, which is the highest authority and everyone trusts it, certifies that the DNSKEY record listed in .com file actually belongs to the organization that owns the zone. Similarly, the .com zone certifies the authenticity of the example.com, and so on. Thus, a chain of trust is created from the root of the DNS infrastructure towards to the leaves.

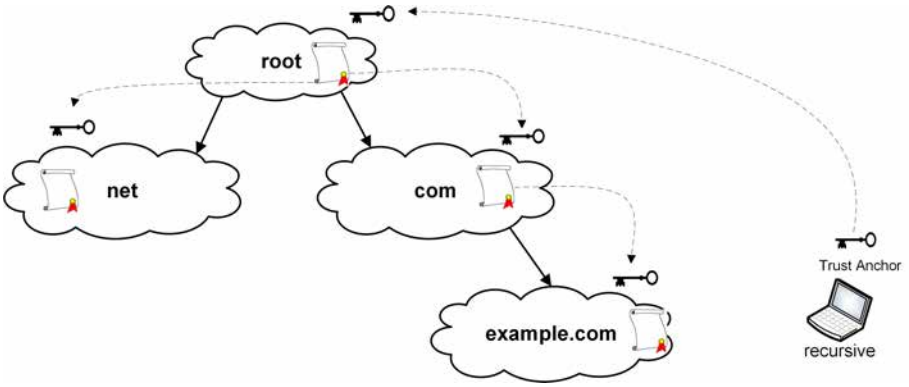



FIGURE 3.5: Typical example of DNSSEC chain of trust

#### 3.2.3.5 Trust Anchor

As pointed out in section 3.2.3.1, a trust anchor is the public key of a zone in the form of DNSKEY RR's type, that is trusted by all clients and therefore its authenticity has not to be proven. Generally, the trust anchor is utilized by the recursive resolvers as starting point for building the chain of trust. Certainly, the ultimately trust anchor is the public key of the root zone. The KSK of root zone should be statically configured in security aware resolvers in order to facilitate the validation of DNS data and in turn they will be utilized to create the authentication chain towards the required domain zone. The administration of the root zone's KSK is provided by ICANN [93]. Also, the security aware resolver could contain trust anchors for islands of security, i.e., DNSSEC-enabled zones with insecure parental zone. These trust anchor are utilized to prove the authenticity of the records contained in the islands or to verify any of the child zones, considering the child's public key is pointed by a DS record. For simplifying the authentication of trust anchor an automatic update mechanism is proposed [94, 95].

Additionally, a recursive could be configured with a *negative trust anchor* (NTA) [96] for temporarily disabling the DNSSEC validation for a specific domain. So, the resolver treats the domain in NTA as insecure and does not set the AD bit in its response. In some cases, there are misconfigurations in the DNSSEC-related RR of a zone that will lead to the verification failure of the zone's data. Hence, that zone will be invisible and the end-users will experience difficulties to access the desired resources. For this cases, the NTA will restore access to the domain until the misconfiguration errors are corrected by the domain's administrator. It is suggested that the inclusion of a zone as

NTA should take place only after the confirmation of failure due to operational errors and is limited to a specific time period.

#### 3.2.3.6 DNSSEC Lookaside Validation

The *DNSSEC Lookaside Validation* (DLV) constitutes an alternative method for creating a DNS chain of trust. Typically, a chain of trust is generated along the delegation from a trust anchor, which ideally is the root zone. However, at the time period that a handful of root or gTLD/ccTLD zones were signed the security aware resolver should maintain a substantial amount of preconfigured trust anchors' keys for each of the islands in order to securely communicate with them. In contrary, the DLV mechanism permits the advertisement of trust anchors located outside the DNS infrastructure and this way eliminates the demand of storing a trust anchor for every isolated island. This is accomplished with the proposition of a new RR type called DLV. This new RR is defined in RFC 4431 [97], while RFC 5074 [98] describes the mechanism of validation. The predominant DLV repository is maintained by Internet Systems Consortium (ISC DLV Registry).

The DLV RR is identical to DS record in terms of functionality and syntax. As we can see in the example below, a DLV RR looks like DS. Specifically, it has the same fields with same possible values except of course the type field. But unlike DS, which is stored to the parent zone file, a DLV RR is inserted in a special signed zone called the lookaside zone. Such zones should contain only DLV and DNSSEC-related RR for validating the DLV RRs. The advantage with the lookaside zone is that it is located somewhere else in the DNS tree structure and not necessarily in the same branch with the domain name. In the following paradigm, we assume that the zone of example.com is secure but its parent is not and consequently could not be used as trust anchor for the certification of its DNSKEY. Rather, the lookaside domain dlv.isc.org is used as a trust anchor. The dlv.isc.org domain is a repository that provides the publication of DLV RR pointing to secure zones. Initially, DLV RR is generated by the administrator of example.com zone using the dnssec-signzone tool. Then, this record is added to the zone file of the lookaside zone and next the zone file is resigned for generating the corresponding RRSIG records.

```
Name                          TTL Class RR key-tag algorithm digest-type digest
; Record in lookaside's zone file
example.com.dlv.isc.org. 3600 IN  DLV 51316     5         2           (
                                  3B822636C70A46CFBBA2ECC3925E72F154
                                  C2DB6B994CD9E98577CF9D F58D02B5)
```

In the case a security aware client supports DLV mechanism, it is forced to browse the lookaside repository as to whether the repository could certify the key of the domain name in question. This method is used whenever any other validation method is inapplicable. Apparently, the client should known that the lookaside zone is actually such a zone and also possesses its authentic public key. For this reason, modification should be applied to its configuration file. Namely, the appropriate lookaside zone name should be specified with the dnssec-lookaside statement and the trusted anchor for the lookaside zone should be included with a trusted-keys statement. Afterwards, the resolver is qualified to query for the DLV RR of the domain name example.com.dlv.isc.org. In the case the client receives a proper RR it uses it to validate the authenticity of the example.com DNSKEY. On the other hand, if the DLV record does not exist for the queried domain name, then this zone could be regarded as insecure. The procedure of the DLV RRs verification is identical as in the occasion of the DS RR.

#### 3.2.3.7 Challenges of DNSSEC Deployment

Although the latest specifications of DNSSEC were announced in 2005, the complete deployment on the 13 root name server was finalized on 2010. This means that for a significant period the adoption of DNSSEC mechanism was limited to certain segments of DNS infrastructure, which are called *islands of security*. The main reason for the delay of the immediate deployment of DNSSEC was the fact that DNS is consisted of millions of autonomous administrative domains collaborating in a decentralized infrastructure. Substantially, DNSSEC constitutes an effort to apply PKI by leveraging the existing DNS delegation hierarchy [99]. For this reason, it requires coordination amongst zones which in the most cases belong to different administration domains with probably diverse

policies and priorities. In other words, it is meaningful for a zone to implement DNSSEC only in the case its parental also does.

Essentially, the islands of security are domain zones that have enabled DNSSEC but their parental zones have not, thus there is not a secure delegation towards them. So in order a security aware resolver to initiate a chain of trust, it should configure the DNSKEY of the islands as trust anchor or obtain an authentic public key from the ISC DLV Repository with the use of DLV RR type as described in section 3.2.3.6. The ISC DLV Repository is operating since 2006, however due to DNSSEC's wider adaptation during the recent years, the administrators are planning the termination of its operation in 2017 [100]. Currently, from the total of 1232 TLDs contained in the root zone 1069 are signed, of which 1062 have published a DS RR in the root zone and the remaining have published a trust anchor in the ISC DLV Repository [101]. However, only a negligible portion of 2LD zones (approx. 3%) are signed [102].

Another issue that hampers the utilization of DNSSEC is the increased complexity of DNSSEC enabled zones' management, which in turn could impose administrative errors. Mainly, DNSSEC misconfigurations are related with missing DNSKEY and DS RR or improperly signed RRsets that violate the chain of trust [103, 104], thus render the secure zones unreachable to secure-aware resolvers. Finally, it is estimated that a significant portion of recursive resolvers do not support DNSSEC [105] or even worst they query for DNSKEY and DS RR, but neglect to validate the RRSIG records [106]. Similarly, the majority of the stub-resolvers do not verify the accompanying DNSSEC related RR [107]. Eventually, in overall, the end users do not benefit from the protection of DNSSEC wider deployment.

### 3.2.4 DNSCurve

DNSCurve constitutes an alternative method to introduce public key cryptography to DNS protocol. This method is based on Elliptic Curve Cryptography (ECC), having appealing characteristics like the short key-length and the high speed of the encryption and decryption process. According to its creator, DNSCurve is aiming to offer the following security requirements to the entities that implement DNS protocol:

- Confidentiality of exchanged DNS packet.

- Integrity of DNS responses.
- Availability of DNS service.

Essentially, DNSCurve creates a secure communication link amongst a ANS and a RDNS. So, new types of DNS messages are necessary. Furthermore, there is the need for DNSCurve-enabled NS and clients to implement the suitable functionalities in order to benefit from the usage of ECC. Finally, the ANSs require a way to publish their public key, so the clients could obtain it.

Unfortunately, so far, DNSCurve has neither been documented in detail nor deployed in a large scale. Therefore, the accuracy and effectiveness of its operations can not be tested in practice but solely theoretically. As already pointed out in section 3.2.2, until now, only one internet draft has been published that provides the general overview of the DNSCurve [69].

#### 3.2.4.1 Security Operations

Every DNSCurve-compliant NS or client maintains a pair of EC public and private key. Whenever the client desires to query a NS or the NS wants to return the response to a query, then the sender enciphers the message with its private key for authentication and with the recipient's public key for confidentiality. Moreover, every pair of query/response is characterized by a distinct nonce, that is, a random number used only once. The usage of this unique number distinguishes each transaction.

Currently, the contents of a DNS packet are not encrypted during its transmission, which stands also true for DNSSEC. Therefore, the data of the message are not protected and are visible to any eavesdropper who is sniffing the network. On the other hand, DNSCurve encrypts DNS messages [108] with the aim to protect their confidentiality, so an attacker is not able to perceive the contained information, either in a query or response message. However, it does not cover the packet's information, such as source/destination IP address and length. In addition, it protects against replay attacks with the usage of nonce, so an eavesdropper is unable to record DNS traffic and transmit it later. Since, every packet is calculated with the sender's private key, the receiver's public key and a nonce, this packet cannot be recorded and later sent to another receiver, because the

attacker does not hold the proper private key to decrypt the message. Even this specific packet cannot be sent to the same receiver at a later time, because of the nonce value. The receiver will comprehend that the packet is related to an earlier transaction and he will discard it.

Concerning the protection against forged RRs, DNSCurve authenticates DNS messages. Therefore upon the reception of a response, the client verifies the message with the sender's public key, in order to detect bogus packets. The involved parties utilize a 255-bit key length ECC. Due to ECC, the operation of encryption/decryption are significantly faster than the corresponding cryptographic operations based on RSA algorithm. The same approach is applied for the case of the authentication of negative responses. Instead of pre-computing signatures, which indicate the gaps between existing domain names, DNSCurve encrypts on the fly the negative answers, in the same way it does with positives. The enhanced performance of ECC allows the NS to handle all the necessary cryptographic operations online. Finally, DNSCurve does not expand the size of DNS packets, neither introduce large RR types. As a result, the amplification ratio for DNSCurve is at the same level as that of typical DNS protocol. Consequently as detailed in section 3.2.5.5, it does not provide to attackers the means to launch amplification attacks [108].

#### 3.2.4.2 Publication of Public Key

In order to send a query to a ANS, the client needs to locate its public key. This key is published in domain's parental zone file as part of the domain name of the delegating NS RR. The same NS RR is also included in the child's zone file. That is the only modification introduced to DNS RR. Following, we present an example of the requisite modification [66], as added to both parental's and child's zone files. Assuming that the IP of ANS of the nytimes.com domain name is 199.239.137.201, the corresponding NS record with its relative glue record is:

```
nytimes.com. IN NS
uz5xgm1kx1zj8xsh51zp315k0rw7dcsgyxqh2sl7g8tjg25ltcvhyw.nytimes.com.
uz5xgm1kx1zj8xsh51zp315k0rw7dcsgyxqh2sl7g8tjg25ltcvhyw.nytimes.com.    IN  A
```

| 199.239.137.201 |
|---|

As we notice, the first label of the domain name of NS type RR is a lengthy string of 54 alphanumeric characters that constitutes the EC public key. The first 3 characters (uz5) is an indication that the remaining characters are a valid EC public key. These characters are the base-32 representation of the 255-bit public key. Hence, for a client to distinguish the public key inside a domain name it should examine each label of the name whether it contains precisely 54 base-32 characters, from which the first three are the string "uz5".

#### 3.2.4.3 DNSCurve message format

The clients have the ability to send a DNS query in two different formats. These are "streamlined" and "TXT" expanded DNS packet format. The first one is smaller and exposes lesser information about the transaction, as it carries a cryptographic box containing the original DNS query and response. The latter has actually the same format as a typical DNS query and a reply of TXT type. So,"TXT" packets are expected not be blocked by firewalls with rigid format constrains on DNS messages. The NS should send the response in the same format as was the received message of the client's query.

A streamlined query packet carries a cryptographic box containing the encrypted initial DNS query packet. In addition, the box incorporates the public key of the client and a nonce selected by him. Similarly, the matching response packet has a cryptographic box containing encrypted the initial DNS response packet, plus the nonce of the client and a nonce chosen by NS. As the response does not contain the queried domain name or the client's public key, the client's nonce is used to match the answer with the pending query.

On the other hand, a TXT packet has actually the same format as a DNS TXT message of typical DNS protocol. It only differs in the semantics of the queried domain name and the RDATA of the response that conveys the encoded message. The domain name in the query section, that should be in compatible domain name format, is constituted by several labels, while the requested RR is of TXT type. From the concatenation

of the labels one can extract the base-32 string of the cryptographic box that holds the encipherment of the original query, accompanied with the client's nonce and public key. The endmost labels carry the name of the zone. In the response packet, the string of the RDATA is the encrypted response also transformed to base-32 format. Namely, the string contained in the answer's TXT RR provides the server's nonce and the cryptographic box that encodes the initial DNS response.

#### 3.2.4.4 DNSCurve operation

Initially, every DNSCurve-compliant server and client possesses a 32-byte private key ($s$ and $c$ respectively). The corresponding public keys are produced with the use of the Curve25519 function, namely *Curve25519(s)* and *Curve25519(c)* [109]. This key pair can be generated during the installation of the system or the reboot process. The public key of each ANS is published as part of its domain name in order to be accessible by various clients. This NS RR type is located in the parental's zone file. Therefore, whenever a recursive resolver desires to query a server, firstly it examines if the authoritative's domain name contains a valid DNSCurve public key. Then, it can produce a shared secret key $k$ from its private key and the server's public key. The server is able to produce the same shared key from its own secret key and the client's public key. This key is calculated by *Curve25519(sc)*. The same shared key could be used for subsequent transactions between these two entities and for performance reasons can be stored locally [66].

Afterwards, the client selects the format of the packet that it will send to NS. The possible options are "streamlined" and "TXT" expanded DNS query packet as explained in 3.2.4.3. The client expects that the server would respond back using the same format. Thereafter, the client selects a 12-byte nonce and assigns it to the packet. Then, it uses the shared key $k$ to expand the nonce into a long stream key with the usage of the Salsa20 stream cipher function [110]. The input nonce to the stream cipher is a 24-byte long string, which consists of the 12-bytes chosen from the client and 12 zeroed bytes. The resulting key is then used for the encryption and authentication of the packet [66]. At this point, the client uses the first part of the previously computed keystream to create the authenticator of the message. In this calculation, the client utilizes the Poly1305-AES message-authentication code function [111]. Then, it encrypts the initial

query by XORing it with the remaining part of the key stream. The final encipherment is comprised of the authenticator followed by the ciphertext itself. Now the client is ready to create and send the query packet to NS. This packet will contain in cleartext the client's DNSCurve public key and the packet specific 12-byte nonce, followed by the cryptographic box that encodes the original query packet [66]. The use of the cryptographic box provides origin authentication (due to the sender's private key) and confidentiality (due to the recipient's public key) for the original packet.

Upon reception, the NS extracts the sender's public key and the nonce for the specific transaction and generates the shared stream keys. Then, it calculates and validates the authentication code of the received packet so as to ensure that the sender truly possesses the private key corresponding to the public key located in the packet. Following, the NS decrypts the box and extracts the query. If any of the previous process fails, then the server should handle the packet as unprotected. Otherwise, it should create the corresponding response and send it encrypted to the client.

As previously noted, the steps for the construction of the answer are similar to the steps followed by the client. The only difference is that the input to the Salsa20 function is a 24-byte long string, of which the 12-bytes are the nonce chosen by the client and the remaining are the nonce chosen by the server. This 12-bytes string is attached in cleartext to the answer. Therefore, the response packet, which is in the same format as the query, includes the 12-byte nonce of the server in cleartext and the cryptographic box containing the original response. The box is created with the usage of the server's private key, the client's public key and the 24-byte nonce. The client verifies and decrypts the box to extract the desired response. If this process fails, it means that the packet is forged. Therefore, the client should silently reject it and continue to wait for the legitimate one. Provided that the opening of the box is successful, the client would find the initial response packet, which would be treated the same as a typical DNS response packet [66]. It is important to notice that the client and server should generate and use a different nonce for each DNSCurve packet encrypted with the same shared key. This requirement is crucial in terms of cryptography and applies during the whole period that the same shared key is utilized.

#### 3.2.4.5 Trust anchor

In the ideal case, where every DNS ANS would support DNSCurve, the security extension would protect all transactions amongst ANS, including root servers, and DNSCurve-aware resolvers during the resolution of a RR. Thereupon, a client should know the public keys of the root and trust them unconditionally. So, the clients would be configured with the list of the root as an trust anchor accompanied with their DNSCurve keys. Consequently, the communication with the root will be protected and the client will securely find the domain names of the TLD's ANS, and so forth. These names should conform to the requirements of section 3.2.4.2 in order to distribute their public keys to the resolvers. In any case, a resolver could be configured with trust anchors for some subtrees of the DNS hierarchy that form a DNSCurve protected island of security. The knowledge of keys for isolated zones would eliminate the requirement of their parental's zone to support DNSCurve as well.

#### 3.2.4.6 Elliptic Curve Cryptography

Recall from section 3.2.4 that DNSCurve deploys link-level public key protection of DNS transactions using ECC. ECC takes advantage of the algebraic structure of elliptic curves over finite fields. Its most prominent characteristic is the small key length compared with other public key cryptosystems. Also, so far, only few attacks have been developed against some specific type elliptic curves, which have special algebraic structure. Security requirements for the elliptic curves used in ECC were specified with IEEE P1363 project that developed standard specification for public key cryptography. In this context, as pointed out in section 3.2.4.4, DNSCurve use the Curve25519 algorithm proposed also by D. J. Bernstein [112]. This type of elliptic curve meets the security requirements imposed by IEEE P1363 standard [108, 113], while it accomplishes immunity to side-channel attacks.

### 3.2.5 DNSSEC vs. DNSCurve: A side-by-side Comparison

In the following sections, we compare DNSSEC and DNSCurve based on nine different criteria. The analysis is based on the observations of section 2.2.1, which present the attack surface specific to the integrity of DNS transactions, whilst Fig. 2.4 illustrates

this surface. Table 3.2 provides a pivotal comparison of the analyzed security extensions based on the selected criteria. In the rest of this section, these criteria are explained and the two solutions are compared with each other. By using this approach, a clear view of the advantages and disadvantages of each method is provided. At the end of the section a discussion of the findings is furnished.

#### 3.2.5.1 Cryptography

DNSSEC utilizes mainly RSA algorithm for Public Key Cryptography (PKC) and permits the operation of various key lengths. The usage of ECC is compatible but implemented recently [85, 86]. It is recommended that a zone should have two separate pairs of keys, ZSK and KSK. The private part of ZSK pair is used for signing the RRsets contained in the zone, including DNSKEY records themselves. On the other hand, the private key of KSK pair is used to create only the signatures of the zones' keys, for both ZSK and KSK. This key is also used for the creation of DS RR. It is endorsed that the KSK is used for the certification of DNSKEY records, and thus its role is more crucial than that of ZSK. For this reason, larger key sizes are suggested in order to be more resilient and require a rollover less often (i.e., longer key effectivity period). Overall, a 1024-bits key size is proposed as ZKS [114].

On the other hand, DNSCurve is based on Curve25519 algorithm [108, 113]. As already pointed out, ECC takes advantage of the algebraic structure of elliptic curves over finite fields and its unique characteristic is the small key length and high performance. However, the size of ECC public key is fixed to 255 bits. This is because each label of a domain name is not allowed to have more than 63 characters [2]. So, the major advantage of the DNSCurve is the usage of high speed ECC that provides a strong level of security with significantly smaller key length. According to NIST recommendations, the typical 1024-bit RSA algorithm provides equivalent security to 80 bits of symmetric keys, while 255-bit ECC is equivalent to 128 bits of symmetric keys [115]. In addition to Curve25519 algorithm, DNSCurve utilizes Salsa20 stream cipher and Poly1305 Message Authentication Code (MAC). Both these functions are proposals of Daniel J. Bernstein. Salsa20 is a stream cipher that maps a 256-bit key, a 64-bit nonce, and a 64-bit stream position to a 512-bit output [110]. It was selected as a member of the final portfolio of the eSTREAM project, part of European Union research directive [116]. On the other

hand, Poly1305 is a MAC that can compute a 128-bit authenticator of a variable-length message [111].

#### 3.2.5.2 Integrity and Origin Authentication

The designers of DNSSEC extension primarily aim to offer robust means in order for the clients, both resolvers and end-users, to be capable of validating the integrity and origin of DNS responses, either positives or negatives. This is achieved with the pre-computation of RRSIG for each authoritative RRset. The private key of the zone is used during the signing process, while the rest of the time is stored in a safe place offline. This way only whoever controls the private key can produce authenticated RR. However, there are some cases where DNSSEC is not able to protect the integrity of offered RRs. Firstly, a zone does not sign the delegating records, namely the NS RR of a child zone with the corresponding A RR (glue records). This happens because these records do not belong to the parental zone, and the zone ought only to authenticate its own authoritative data [99]. Therefore, whenever a resolver receives a DNSSEC response containing delegating records for a child zone is is unable to verify whether the contained RR have been forged [117]. For example, in the response, the address of the glue record can be forged, while the RR in the authoritative section will contain valid signatures. For this reason, the response packet seems to the client legitimate and unmodified. The resolver only through the DS RR is able to perceive that an forged attempt has occurred, since the DNSKEY RR of the child zone will not match to the DS of the parental zone. In the case the delegating zone does not support DNSSEC, then the resolver will fall victim of forged RR that are DNSSEC protected.

Another related flaw of DNSSEC is the case when some data are administratively modified and the zone's file is resigned. However, the signatures of the stale RRs could not yet expired [91, 118]. In such a situation, an attacker may store the preceding RRs with the accompanying valid signatures and exploit them until their expiration date as part of a replay attack, also referred to as freshness attack [52]. This also means that the aggressor is able to redirect the victim to a malicious site or just cause them DoS. Up to the time the signatures become obsolete, the victim cannot tell if received data are valid. Unfortunately, this expiration time could be lengthy as the RRSIG records are pre-computed and not created on-the-fly, so they are generated with a high validity

period (usually 30 days) for performance reasons. A possible countermeasure against this flaw is proposed in [119], which utilizes hash chains in an effort to prove that RRs are fresh and to limit down the replay window. However, this method is rather incompatible, requiring changes to both RRSIG type's format and the functionality of the servers and resolvers.

DoS could also be caused due to the manipulation of localized services' RRs. Many organizations install multiple copies of their resources in different network domains with the purpose to offer specific services to individual local area groups. By doing so, these groups access localized services, while the companies achieve load balancing. On the other hand, an eavesdropper could capture records with their appropriate signatures from a particular domain and replay it to the users of a different domain. Consequently, even though these users are able to verify the authenticity of the records, they are not able to access the content they request. So, by replaying valid RR, the attacker is able to cause DoS to the unsuspecting users [66].

Summarizing, there are some occasions when the end-user may be deceived by seemingly legitimate DNSSEC responses and redirected to a malicious site or experience DoS. Moreover, as stated in [91], cryptographic verification is not always equivalent to validation. In other words, there are some conditions that even verifiable RRs could still contain false data and not what the zone administrator intended to provide initially.

On the opposite, DNSCurve performs its cryptographic operations online (in realtime). Therefore, the private key of the zone is stored in the server and is used for the encryption and authentication of the original response upon the reception of the query. Each response packet is created independently from other responses and is unique. Consequently, authenticated answers, positives or negatives, can only be created by whoever controls the (primary or secondary) authoritative server. Finally, with the usage of nonce each response is bind to a specific transaction and cannot be used later to a replay incident.

#### 3.2.5.3 Confidentiality

Generally, DNS protocol does not concern about the confidentiality of DNS messages or the zone's contents. This is because the zone operates as a public directory of the

contained hosts. Therefore, DNSSEC does not propose any mechanism to protect the confidentiality of the packets in transit. Although DNS data are considered as public information, the transaction between an end-user and a NS should not. Let us suppose the case where the user utilizes an Hypertext Transfer Protocol Secure (HTTPS) connection to access a website. Prior the HTTPS they deploy an unencrypted DNS transaction which reveals significant information about the forthcoming request. Thus, DNS protocol may become the weakest link amongst the network protocols regarding privacy protection [120].

Moreover, the introduction of NSEC type [15] deployed for the authenticated denial of existence creates a new security vulnerability, namely zone enumeration as described in section 3.2.3.2. Using the standard DNS protocol the user must know the domain name of the host in order to request the respective RRset. Therefore, an attacker has to exhaustively scan possible names, aiming to find out the host names within a zone and the corresponding types of RRs. However, with the deployment of DNSSEC and the utilization of NSEC type such discovery is almost straightforward. A NSEC record links two consecutive names of the zone (in canonical ordering) with the aim to prove that intermediate domain names do not exist. Consequently, the aggressor only has to follow the chain created with this type of RRs to eventually retrieve the zone's data. This way, a potential attacker is able to acquire valuable information about the structure of a targeted network, referred to as *reconnaissance attack*. In other words, the evil-doer is able to discover possible sensitive information about the internal configuration of the network infrastructure or details about the offered services. This enumeration is not considered as an attack against the DNS protocol itself, but rather as a threat against the privacy of the zone's data [15].

The response against this flaw is the proposition of a new extension called DNSSEC Hashed Authenticated Denial of Existence given in RFC 5155 [70]. With this extension, whenever a requested RR does not exist, DNSSEC-enabled NSs respond with a NSEC3 record's type, instead of a NSEC. This RR type also provides authenticated denial of existence for DNS RRsets, but does not permit zone enumeration. The main difference of NSEC3 is that it links two consecutive hash values of domain names in order to prove that domain names with intermediate hash values do not exist. Nevertheless, even with the usage of NSEC3 record zone enumeration is still possible and can be argued that it is more effective than in the case of original DNS protocol [108]. When an attacker

wishes to determine the hosts and their respective attributes inside a zone implementing DNS protocol, the only thing they have to do is to exhaustively query for any possible combination of domain names. However, this action requires sending a vast amount of packets towards the authoritative servers of the zone. Therefore, such an attack does not remain unnoticed by the administrator of the server and could be easily blocked during its execution.

Nevertheless, in the case of DNSSEC, the attacker could silently obtain the NSEC3 records through limited interactions with the ANS, and then offline test the hash values in order to figure out the initial domain names. This can be achieved by performing a dictionary attack, i.e., by calculating the hash value of likely possible domain names. Such an attack is noiseless and the administrator of the server will not suspect anything [121]. In fact, recently, a GPU-based attack accomplished to extract 64% of all domain names in the .com domain zone. Namely, the researchers collected the majority of NSEC3 RR in the .com zone and with cryptanalysis they managed to reveal a significant portion of contained domain names. Specifically, they extracted nearly 62% of all the contained domain names within 14 hours with dictionary attack and the remaining portion within 4.5 days by applying a Markov chain attack [122]. Goldberg et al., [123] propose a RSA-based keyed hashing scheme instead of an unkeyed hash function for offering authenticated denial of existence without the possibility of zone walking. Specifically, they present NSEC5 RR type which likewise NSEC3 RR contains the hash value of domain name singed with a private key. The RSA key pair is different from that of the zone and is stored online for performing signing operations for every negative answer at the time of the request.

On the other hand, DNSCurve calculates a different keystream for each DNS packet for its encryption. By doing so, DNSCurve protects the contents of every DNS packet (request or response) of the transaction from a passive Man-in-the-Middle attack. Certainly, the eavesdropper is able to determine the inquired zone, but they are not capable of discovering the exact domain name and resource type requested. Also, concerning the confidentiality of the zone, DNSCurve behaves as the original DNS protocol. Therefore, an attacker should exhaustively query the DNSCurve-enabled NS in order to find out the contents of the zone.

#### 3.2.5.4 Authenticated Denial of Existence

The typical DNS protocol provides NXDOMAIN or NODATA type of response, whenever a domain name or a RR are not contained in the zone. However, NXDOMAIN response can trivially fabricated, and thus lead the victim to DoS. Let us consider that a client issues a query for a RR that truly exist, but in the meantime the aggressor injects successfully a forged NXDOMAIN response with a similar technique as that of section 3.1.2. The client would accept the NXDOMAIN response as genuine and it would be unable to connect to the resource that it desires. Besides their query section NXDOMAIN responses do not differentiate to any other field of the answer section. For this reason, a way to provide authenticated denial of existence is requisite. In DNSSEC this is done with the NSEC/NSEC3 RR type.

Due to performance reasons, DNSSEC executes cryptographic signing operations on the zone's data prior to their publication. Therefore, to provide proof of non-existence a ANS sends back a Next Secure (NSEC/NSEC3) RR rather than a simple NXDOMAIN response. Unfortunately, as discussed in section 3.2.5.3 this type of RR leaks sensitive information about the hosts within the zone.

DNSCurve operations are designed to operate on-the-fly and therefore the responses (either positives or negatives) are encrypted and authenticated upon the reception of the DNS query. By doing so, NXDOMAIN response is encrypted and authenticated with the combination of client's private key, server's public key, and the nonce of the session. Therefore, this negative response is unique for the specific transaction.

#### 3.2.5.5 Amplification Attacks

During a DNS amplification attack, the aggressor tries to force multiple authoritative NSs to produce large responses when processing corresponding queries. In these queries, the attacker advertises the usage of a large size UDP buffer. Additionally, the queries seem to originate from the victim system, as the attacker forges the source IP address of the query packet. Therefore, the ANSs unconsciously flood the targeted system with large UDP packets causing it DoS [124, 125].

Unfortunately, the size of DNS packets among transacting DNSSEC-enabled entities is increased significantly, as the response packets carry a signature for every contained

RRset. Even worse, the rest DNSSEC related RR have extensive size as well. Hence, these entities should support EDNS0 extension. This way, DNSSEC facilitates successful reflection attacks against targeted systems since it increases the size of the responses and thus augments the amplification ratio of the attack even more.

Contrariwise to DNSSEC, DNSCurve does not augment the size of the messages, as it restricts the size of the initial (prior to execution of any cryptographic function) DNS responses to a maximum of 512 bytes. Whenever the initial response exceeds this limit, the server sends the reply as a truncated packet, which is also protected by DNSCurve. Possibly, after encryption, a DNSCurve packet would exceed the 512 bytes threshold; these extra bytes are generated due to cryptographic operations. Moreover, DNSCurve lengthens the name of each NS by 54 bytes, causing extensive glue records compared to the original DNS. However, the overall amplification ratio is low and at the same level as that of standard DNS. Consequently, DNSCurve does not provide to attackers the means to launch an amplification attack [66].

#### 3.2.5.6 Modification of DNS Infrastructure

Certainly, both DNSSEC and DNSCurve require major modifications to DNS functionality and infrastructure, which hampers the wider adoption of cryptographic solutions to DNS protocol. Obviously, both DNS network entities, server and client, need to change their software in order to perform cryptographic operations for carrying out the necessary functions.

Furthermore, as already pointed out in section 3.2.3.2, DNSSEC utilizes four new RR types. Also, it uses the DNSSEC OK (DO) EDNS0 header bit [89] to advertise the ability of the client to handle DNSSEC RR. Finally, it adds two new DNS header flags, namely the Authenticated Data (AD) and Checking Disabled (CD) [16]. These flags are used in the transactions amongst a stub-resolver and a recursive NS, that both support DNSSEC. A recursive sets the AD bit in the responses when it successfully verifies the authenticity of all the contained RRsets, including the NSEC records. Also, a resolver sets the CD bit, when it wants to perform the validation of the RRSIG records by itself according to its local security policy. This way, it forces the recursive to not validate the signatures of the response, but rather include the DNSSEC-related RR in the response.

On the other hand, DNSCurve also requires means to carry out the encoded message between the client and the server. Towards this direction, two expanded DNS message formats, namely streamlined and TXT are introduced. The first is more compact and transfers securely the encoded message. But, it differentiates from the standard DNS packet, and thus could be blocked from firewalls imposing rigid format constraints on DNS packets. In such cases, the second type could be utilized, which transfers the encoded response as a string of TXT RR. However, the size of the latter is larger and leaks more information compared with the first format. Additionally, it modifies the domain name of ANS for announcing the public key of the zone.

#### 3.2.5.7 Zone Administration

Regarding DNSSEC, the zone file should be re-signed whenever a modification of the zone file occurs. This is because the signatures for the new or modified RRs and the NSEC/NSEC3 RRs have to be regenerated (re-adjustment of the gaps). This process is considered very computational intensive and time consuming. Nevertheless, a malicious user, who has gained dynamic update permissions, is capable of consuming the computational resources of a DNSSEC-enabled NS with cryptographical operations by forcing dynamic updates to its zone file [14]. Also, a re-signing process is necessary before the expiration of the signatures. Otherwise, if the RRSIG RR expires, then the zone data will not be validated by the clients and the domain will be invisible, i.e., the clients will not be able to verify the authenticity of the RRs, and hence they will discard them. Another disadvantage of DNSSEC is that it imposes time synchronization amongst the transacting entities [114, 46]. Since every signature is valid for a particular time-duration with specific time of inception and expiration, the resolver should have the same concept of absolute time as ANS so as to be able to figure out if the signatures are still valid or they have expired. Unfortunately, an attacker could fool a user into trusting expired signatures or rejecting still valid records by manipulating the client's perception of time. Similarly, an attacker could also drive the NS to generate signatures with validity period different than the current time.

DNSCurve on the other hand only changes the way the servers and clients communicate. Besides this, it does not imply modifications to the core DNS protocol. The only

difference is that they just send encrypted the initial message. Also, its operation is not affected by modification of the zone file.

#### 3.2.5.8 Key Management

Another important issue that applies to both these protection mechanisms is the storage of the zone's private key. Because DNSSEC pre-computes the signatures prior to their publication, the key should be preserved offline for safety. However, in the case the zone allows dynamic updates, the safe keeping is not possible. So, for such a case the private key should be maintained online in the primary ANS in order to create the RRSIG for the updated RR and re-adjust the gaps indicated by NSEC/NSEC3 RRs in real-time. Concerning the key management, DNSSEC requires the regular replacement of the asymmetric key and particular the ZSK. As this key is used often for the signing of the records, an attacker could collect enough plaintext and ciphertext samples to launch a known-plaintext attack. Also, for performance reasons, the ZSK length is smaller than the rarely-used KSK. So, it is essential not to be used for long periods of time. However, this rollover process is complex, lengthy and needs to be scheduled somehow [126].

Regarding ZSK, two ways have been proposed to maintain verifiability during key rollover [114]. The first one, known as *key pre-publication*, designates to publish the new ZSK in the zone file but without generating RRSIG with that key for a time period. This way, the key will be diffused in the DNS infrastructure and reside in the cache of the recursive before the definitive removal of the stale ZSK. The second method known as *double signatures* determines the existence of RRSIG created by both ZSKs, the old and the new one. On the other hand, as concerns the KSK a similar approach to double-signatures scheme is followed; however the parental zone is also involved in the procedure as dictates the reception of the new key and following the modification and re-signing of the matching DS RR. In order to avoid the manual notification of the parental zone about the change of KSK, RFC 7344 [127] proposes an automatic procedure. Namely, two new RR types are presented called CDS (Child DS) and CDNSKEY (Child DNSKEY). Similar to DS and DNSKEY RR, these two RR contain the KSK's fingerprint and public key respectively. They should be placed in the child zone after a rollover process take places. So, the parent have to periodically poll for the specific RR in order to get informed about the new delegations of the chain of trust. The precise

procedure of key rollover are outlined in RFC 6781 [114] and RFC 7583 [128]. In case the rollover is not successful, the DNSSEC clients are not able to obtain the proper public key in time, and therefore cannot validate the signatures of the records they receive.

Additionally, a plan for emergency rollover of compromised key is essential since otherwise the attacker possessing the private key of a zone would be able to create valid signatures for malicious data as long as the trust chain exists. However, a trade-off should be taken into consideration between the preservation of valid chain of trust and the quick revocation of the compromised key. Maintaining the chain of trust will allow the attacker to offered fabricated data as legitimate as long as the compromised key resides in the cache of the resolvers, while the urgent replacement of the DNSKEY RR will render the zone invisible to security-aware resolver. Also, for the case of compromised KSK, the DS should be replaced as well.

On the opposite, DNSCurve-enabled ANSs store the private key online, as they use it in every DNSCurve transaction. Naturally, this raises the risk that in case the server is compromised the intruder would be in position to entirely control the DNS transactions for this server. Moreover, the key rollover for DNSCurve should not necessarily take place so often, since ECC is considered more resilient against this type of attacks. In addition, when the process of the replacement takes place, it only affects the domain name of the ANS, which should be added in the parental zone's file. The propagation of the new public key to the clients would happen within TTL value of the NS RR. However, it is not clear which of the two public keys the server should use during that time and how it should handle queries produced with the revoked key. Finally, in case a malicious user discovers the private key of the zone, they would be able to masquerade as the ANS for the duration of TTL of the corresponding NS RR.

#### 3.2.5.9 Performance

For estimating the actual performance of the security mechanisms, one has to analyze the actions taken by each of the DNSSEC and DNSCurve-enabled servers, when they accept a query. Here, we assume that the client has located the ANS of the parent zone for the inquired domain name. In the case of DNSSEC, we also assume that the client has created a chain of trust towards the parent zone, and thus has authenticated the DNSKEY of this zone. This is depicted in Fig. 3.5, where starting from the root zone

the client trusts its public key as trust anchor, which in turn is used to authenticate the public key of the .com zone and so forth. Also, we have to mention that the actual behavior of the security-aware server and client may vary from that of the following analysis, as it depends solely on the implementation. However, in this section we present the common case.

Firstly, the client queries the parent zone for the NS RR of the child. This step is illustrated as (1) in Fig. 3.6. The answer (2) indicates the ANS of the child zone and the appropriate glue records. In the reply of the delegating RR is also included the DS RR containing the fingerprint of the child's public key. The inclusion of the DS RR augments the size of the reply by 36 bytes [129]. The latter RR is accompanied by its corresponding RRSIG generated by the private key of the parent zone. Each RRSIG RR adds an overhead of 46 bytes plus the key size and the length of the zone name [129]. Afterwards, the client requests from the child's ANS the desired RR (3). The response contains this record, with its RRSIG calculated by using the private key of the inquired zone (4). In addition, this packet could carry more RRSIG RR in the authority and additional sections for each authoritative RRset that the server chooses to include in the response [129]. For verifying the authenticity of the response, the client needs as well an authenticated copy of the DNSKEY RR of the zone. It is possible however that the DNSKEY RR does not fit into the previous reply [14]. This is due to the extensive volume this RR's type might have. So, the client would again ask the authoritative explicitly about its DNSKEY RR (5). This final reply (6) is increased by 18 bytes in addition to the size of the key for each DNSKEY RR plus the size of the corresponding RRSIGs [129]. It is recommended that medium-value zones should have a KSK of 1300 bit size and a ZSK of 1024 bit size [130]. There are two accompanying RRSIGs, one generated with the KSK's private key and one with the ZSK's. Upon the reception of this final response, the client is able to authenticate the public key of the zone and then the desired records.

In the case the desired RR does not exist, the DNSSEC-enabled server should respond with a NSEC/NSEC3 RRset to provide authenticated denial of existence along with the corresponding signatures. In most cases, the response is comprised of up to three NSEC/NSEC3 RR, one that authenticates the non-existence of the exact name and one or two that authenticates the non-existence of an applicable wildcard RR that can expand to the desired domain name [131]. The size overhead for each NSEC record is

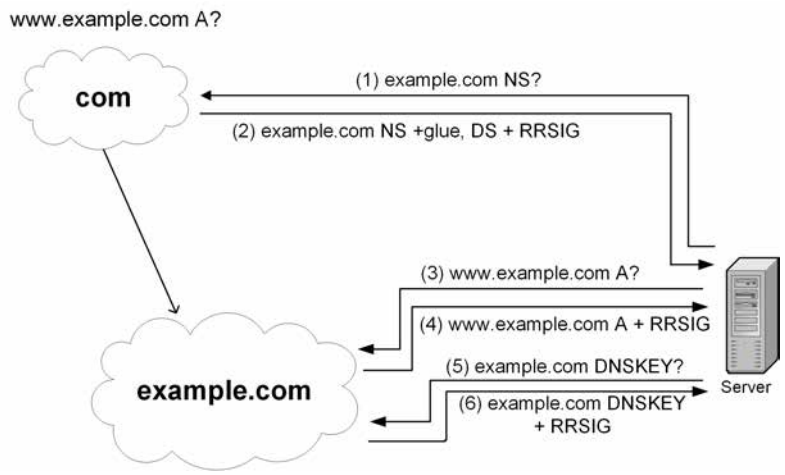



FIGURE 3.6: Example of a DNSSEC transaction

23 bytes plus the length of the domain name and the length of the label [129]. If the server also supports NSEC3 RR, it must first calculate the hash value of the requested domain name before sending the response. This would introduce a computational overhead that depends on the number of the iterations of the hash function. According to recommendations, the number of iterations should not exceed 150 for small length ZSK [70]. As long as the client possesses an authenticated copy of the public key of a zone, the following interactions between this zone server and the client require only the final query-response (3),(4). All steps of the complete DNSSEC transaction are depicted in Fig. 3.6. Concluding the analysis of DNSSEC, we have to mention that for the ANS the overhead for the computational load arises from the larger data that the server has to send, as it does not have to perform any cryptographic operation online [129]. Therefore, the computational load is almost infeasible to be estimated theoretically, but only in practice.

Regarding DNSCurve, the client requests the NS RR of the ANS (step (1) of Fig. 3.7). The domain name contains the public key of the zone, and therefore the reply (2) is 54 bytes greater than in the simple case (due to 54 alphanumeric characters representing the 255-bit public key in base-32 format). Next, the client calculates the shared key from its private key and the server's public key with the usage of Curve25519 algorithm. The process of creating the shared key is estimated not to take more than 957,904 cycles on a Pentium 4 processor [112]. Then, the server uses this key to generate the keystream by expanding the nonce with the Salsa20 stream cipher. If we consider that the size of the input nonce is 24 bytes, the overall procedure of the stream cipher will require

24*4.3=103.2 cycles on an Intel Core 2 1.83 GHz processor [132]. Afterwards, the client utilizes the first part of the keystream to generate the authenticator of the message by using the Poly1305 authentication code function. In addition to the encrypted query, the packet contains the client's public key (255 bits), the 96-bit nonce and the 128-bit authenticator. When the NS receives the DNS message (3), it calculates similarly the shared key from its private key and the client's public key and generates the keystream. Then, it verifies the authenticator of the message and decrypts the query. Next, it calculates the answer with a new nonce and keystream. The response (4) is comprised of the requested RR encrypted, the 96-bit nonce and the 128-bit authenticator. Finally, upon the reception of the response, the client generates the new keystream in order to authenticate and verify it. Any subsequent interaction between the same client and server does not request the calculation of the shared key. The above procedure is the same for positive and negatives responses. An example of the way the DNSCurve client interacts with the authoritative DNSCurve server is illustrated in Fig. 3.7.

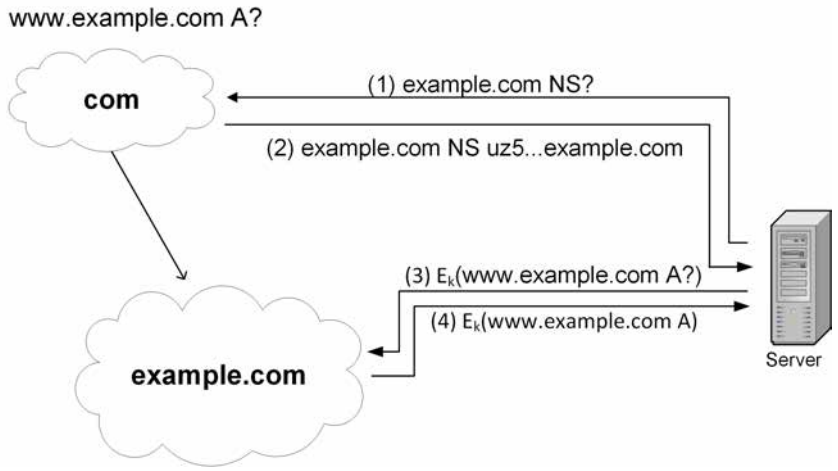



FIGURE 3.7: Example of a DNSCurve transaction

Taking into account the performance of the proposed mechanisms, it seems that DNSCurve is supposed to behave better than DNSSEC. First of all, DNSSEC uses primarily RSA algorithm [15] for cryptographic procedures. RSA algorithm requires larger key length and thus exhibits low performance. For this reason, and in order to reduce its computational load to minimum possible, a DNSSEC-enabled server pre-computes the signatures for every RRset of the zone. Consequently, the computational load is not affected by the volume of the requests. However, a DNSSEC server should perform hash calculations in the case it utilizes NSEC3 RR, whenever a non-existent domain name

or type is inquired. If we consider that for such a response the server should iterate repeatedly the hash function (the upper limit is 150 iterations for s small ZSK [70]) and the significant percentage of queries for non-existent RR toward the authoritative servers, the computational load of the server is considerably increased. Overall, the computational overhead because of the cryptographic operations peaks during the signing process, while in normal operation it depends on the utilization of NSEC3. Secondly, due to the new types of RR, the size of a signed zone is considerably augmented (by a factor of 5 to 7 times) [52]. This occurs because for every RRset two extensive RRs are inserted, e.g., RRSIG and NSEC/NSEC3. As a result, the computational load on the server due to the larger size of the database is heavily aggravated.

Moreover, the size of DNS packets among the transacting DNSSEC-enabled entities is increased notably, since the packet in this case carries a signature for every contained RRset. The remaining DNSSEC-related RR types are as well sizeable enough. Thus, the limit of 512 bytes per DNS packet is no longer applicable and thereby the deployment of EDNS0 mechanism is compulsory. The increased size of UDP DNS packets augments the network traffic between server and client and consumes valuable bandwidth. Overall, the total amount of bandwidth is estimated to increase by a factor of 2 or 3 [133]. Nonetheless, the increased size of DNSSEC-related DNS responses facilitates the launch of DNS amplification attacks with notable amplification factor. This kind of attacks is extensively discussed in chapter 4. Another disadvantage of DNSSEC is that it augments the amount of messages that a resolver sends towards a DNSSEC-enabled ANS [129]. In order to locate and verify the authenticity of a zone's public key, the resolver must first consult the zone about its DS and DNSKEY records. This way, the client approximately sends two new queries for every new zone in question. Accordingly, the total network bandwidth allocated for DNSSEC traffic is increased. Furthermore, the overhead of the ANS is also augmented. This is because now the ANS possesses a larger size zone and serves more queries. Also, the signature validation process increments the workload of the client [52]. All these factors affect substantially the overall time of a transaction to complete. Ultimately, these overheads are reflected in the way that the end-users experience their access to the Internet and other related network resources and services.

On the downside, DNSCurve is based on ECC, which is characterized by short key lengths and high speed of cryptographic operations. The computational load because of the cryptographic operations depends on the queries as the server encrypts/decrypts

each response/request. However, the average overhead is expected to be tolerable, because of the ECC characteristics. Furthermore, DNSCurve does not imply extensive modifications to the zone file of a domain. It only requires the modification of the NS's domain name of the zone in question. This RR provides the public key of the zone, so a client is able to obtain the public key by requesting the specific RR. The extra size for each NS RR/public key is 54 bytes, which is negligible compared to the size of a zone file. Additionally, it does not increase notably the size of the packet. It always keeps the original DNS response smaller than 512 bytes. The extra bytes exceeding this 512 bytes threshold are generated because of the cryptographic operations and the carried nonces. In such a case, the DNSCurve server is allowed to send the expanded response packet through UDP. Also, as already pointed out, DNSCurve lengthens the name of each NS by 54 bytes, causing extensive glue records compared to the original DNS [66]. Finally, since the public key is embedded in the domain name of the server, the client does not need additional queries to acquire it, as in the case of DNSSEC. Therefore, the volume of DNS messages exchanged among DNS entities remains the same.

#### 3.2.5.10 Conclusion

DNS constitutes one of the most critical services of Internet infrastructure, since many other protocols base their undisturbed operation on it. Therefore, any DNS security breach could cause severe problems to affected network domains and in the worst case to Internet as a whole. Potential attackers could take advantage of the design and implementation flaws of the DNS service and provide the end-users with forged data. By acting this way, they may be able to redirect end-users to network locations controlled by them instead of what was requested in the first place. The impact of such attack could be identity theft, commercial and financial loss, malware infection to mention just a few. Ultimately, successful cache poisoning attacks change the way the users experience Internet. To cope with this kind of attack, DNSSEC and DNSCurve aspire to be the secure replacement of DNS protocol. It is true that so far DNSSEC leads the way having several of its features standardized. Both protocols utilize public key cryptography and aim to protect the DNS transaction among ANS and clients. Based on the cryptographic operations the proposed mechanisms are able to protect the integrity of DNS messages and ensure that cache poisoning attacks are very difficult to achieve, if not infeasible.

| Criterion | Sub-criterion | DNS | DNSSEC | DNSCurve |
|---|---|---|---|---|
| **Impact** | Documentation | ✓ | ✓ | Limited |
| | Implementation | ✓ | ✓ | Limited |
| **Cryptography** | Performance | - | Low speed (PKC) | High speed (ECC) |
| | Security | - | 80-bit | 128-bit |
| | Key length | - | 1024-bit (variable) | 1255-bit (fixed) |
| **Security** | Protection | - | End-to-end | Link-level |
| | Confidentiality of Transaction | - | ✗ | ✓ |
| | Confidentiality of zone (Zone walking) | ✓ | ✗(NSEC/NSEC3) | ✓ |
| | Integrity & Origin Authentication | - | Yes, in the general case. Not provided for delegating, stale but not expired, localized records | ✓ |
| | Authenticated Denial of existence | - | ✓(NSEC/NSEC3) | ✓ |
| **Attacks** | Amplification | - | ✓ | ✗ |
| **Performance** | Size of Zone File | - | x 5-7 times | Same |
| | Packet size | 512 bytes limit | Support of EDNS0 | The original packet size is limited to 512 bytes, the domain name of the NS is increased by 54 bytes |
| | DNS traffic | - | Increased (queries for DNSKEY and DS) | Same |
| | Total query-response time | - | Increased | Same |
| **Modification** | Change of SW | - | ✓ | ✓ |
| | Message format | - | Same | Different (TXT & streamlined packet format) |
| | New RR types | - | 4 | ✗ |
| **Administration** | Modification of the zone | - | Requires singing and re-adjustment of the gaps | None |
| | Absolute time synchronization | ✗ | ✓ | ✗ |
| **Key management** | Private Key storage | - | Offline | Online |
| | Key Rollover | - | Often, complex, lengthy, scheduled process | Seldom |

TABLE 3.2: DNSSEC vs. DNSCurve

Based on the side by side comparison of the two solutions given in section 3.2.5, we can conclude that DNSCurve offers stronger security against active and passive man-in-the-middle attacks, while it keeps the computational load at the ANS side minimal. However, DNSCurve only protects the communication between ANS and recursive resolver by creating an encrypted link amongst them. Yet, it fails to offer end-to-end security. A DNSCurve-enabled recursive resolver does not have the means to notify the stub-resolver about the validity of the response, and thus the stub-resolver should trust it blindly. In addition, DNSCurve differentiates itself from the core DNS protocol and in an effort to offer enhanced security obscures some of the basic DNS protocol characteristics,

making it less "open". For instance, a firewall may block a DNSCurve packet because it is impossible to process it and identify that it comprises part of a DNS transaction. Certainly, we have to wait until an implementation of DNSCurve is announced in order to estimate its behavior in actual scenarios.

On the other hand, DNSSEC covers all DNS transactions and hence end-users can fully benefit from its deployment. It aims to protect the integrity of authoritative data against active attacks. However, there are some cases where it fails to accomplish this purpose. Another disadvantage of DNSSEC is that it augments the workload of NSs and imposes a significant penalization in DNS traffic. As a result, DNSSEC increases the overall time for the involved entities to respond and loads the network with additional traffic. Nevertheless, DNSSEC appears to comply better with traditional DNS protocol and be the cryptographic descendant of it.

Taking all the above into account a mechanism that combines the advantages of both methods, would be highly appreciated. Such a new mechanism should utilize ECC cryptography, which is characterized by the high speed of encryption and decryption process. Also, it should guarantee both the integrity and confidentiality of transmitted messages. On the other hand, it should protect all the aspects of a DNS transaction, not only the communication link between the NS and the client. This way, the end-users would obtain secure, reliable and smooth DNS services. Unfortunately, such a mechanism is impractical for the moment. The main reason is that it would impose significant changes to the current DNS infrastructure and severe modifications to DNS protocol messages format. It is also obvious that such changes would not be backward compatible. Another issue is that the usage of ECC is not widely adopted for use in DNSSEC protocol.

### 3.2.6 DNS over DTLS

As outlined previously in section 3.2.3, even with the presence of DNSSEC, the confidentiality of the transmitted DNS data is not guaranteed. This absence of protection may rise privacy concerns, since DNS data often contain sensitive private information [120]. To remedy this inefficiency, the specifications for employing *DNS over the Datagram Transport Layer Security (DNS-over-DTLS)* were very recently introduced [134]. These specifications are published in the form of and experimental Internet Draft. As it

is well known, the Datagram Transport Layer Security (DTLS) specified in RFC 6347 [135] constitutes the equivalent of Transport Layer Security (TLS) for the case of UDP protocol. With the help of DTLS, a secure channel between the DNS transacting entities is built, and thus the data are protected from eavesdropping and active injections attempts. The DNS-over-DTLS mechanism mainly benefits the stub-resolvers, as they can safely communicate with their corresponding RDNS. The proposed mechanism listens for connections on 853 UDP port.

Particularly, once a DNS client desires to make a DNS request, it should first determine whether the DNS server supports DNS-over-DTLS. To do so, it sends a *DTLS ClientHello* message to port 853 of the server. This situation is illustrated in Fig. 3.8 [136] as message *Flight 1*. In the case the DNS server does not support the mechanism, it will not respond to the Hello message. Otherwise, the *DTLS handshake* procedure is initiated, during which the cryptographic parameters are negotiated for establishing a secure session between the two parts. In order to eliminate the creation of half-open sessions (from the same client) and for the verification of the requostor, the exchange of *Cookie* in the initial phase of the handshake is introduced (flight 2 & 3). Subsequent transactions are cryptographically protected, namely the DNS queries and responses are encrypted.

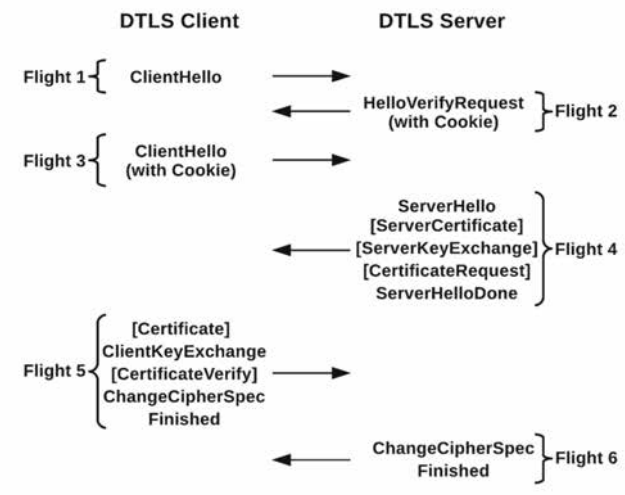



FIGURE 3.8: Message Flow for Full DTLS Handshake

### 3.2.7 DNS over TLS

Similarly to DNS-over-DTLS mentioned above in section 3.2.7, DNS over Transport Layer Security (DNS-over-TLS) [137] aims to provide protection from passive listeners and on-path tampering in the case of DNS transactions over TCP protocol. The TLS protocol specified in RFC 5246 [138] provides cryptographic protection of TCP connections. The establishment of a secure session is similar to that of the DNS-over-DTLS, however this time the communication channel takes place over TCP. Whenever a DNS client wishes to send a DNS query over TLS, they initiate a TLS handshake as described in [138, 139] for negotiating the session's cryptographic suite. Once the secure session is successfully set up, DNS queries and responses are transmitted encrypted. The proposed mechanism listens on 853 TCP port and applies for the case of connections among stub resolvers and recursive servers.

# Chapter 4

# Novel DNS amplification attack vectors

In this chapter, we detail on legacy DNS amplification attack and also propose and evaluate new flavors of the same attack. As already pointed out in chapter 2, DNS amplification is a type of DDoS attack that takes advantage of DNS infrastructure to amplify and reflect the attack traffic. In this context, we propose and assess DNSSEC-powered amplification attacks, which are based on DNSSEC-related RR. Alongside, we evaluate the entanglement of the upper DNS hierarchy to attack incidents. The main goal of this chapter is to demonstrate that even an unsophisticated aggressor is capable of launching devastating DDoS attack by exploiting publicly available resources of the Internet.

## 4.1 DNS Amplification

DNS amplification is a type of DDoS attack that combines amplification and reflection characteristics [140]. Namely, it multiplies the attacking network traffic and simultaneously reflects the traffic by third-party servers towards the target. Usually a possible victim of a DNS amplification attack could be anything connected to the Internet, from a user machine to a high-end server. The amplification attribute of the attack augments the impact on the targeted victim, while the reflection obfuscates the forensic signal of the attack.

In a nutshell, the effectiveness of DNS amplification attack lies on the fact that a small DNS request could trigger a much larger response. The efficiency of the attack is measured by the amplification factor; the greater the amplification factor, the quicker the bandwidth and resource consumption on the victim's side. Typically, in the literature, two equations are used to express the amplification factor:

$$\text{Amplification Factor} = \frac{\text{length(response)}}{\text{length(request)}} \tag{4.1}$$

$$\text{Amplification Factor} = \frac{\text{sum of length(responses)}}{\text{sum of length(requests)}} \tag{4.2}$$

Equation (4.1) [141] illustrates the ratio of the size of the response versus the size of the initiating request. This formula is applicable when one sends a single request towards the exploited DNS NS. Equation (4.2) [140] on the other hand, indicates the cumulative size of the responses divided by the size of all requests. Therefore, it is suitable in cases where a batch of requests is dispatched against a NS, but the size of the disparate responses varies, for instance because some of them are truncated. It is to be noted that from the aforementioned calculations usually the packet headers are excluded and are taken into consideration only the size of DNS packet (UDP payload).

For accomplishing reflection, the aggressor manipulates the network traffic to make it appear as it originates from the victim, for instance via IP source spoofing. Thus, the attack traffic is reflected by unconscious third servers towards the clueless victim. This way, neither the reflector nor the victim are aware of the true source of the attacking traffic. The reflection is trivially implemented when connectionless protocols are utilized [142]. Especially in the case of protocols that mostly rely on UDP, such as DNS, the attacker is in position to easily spoof the source IP address to that of the victim's. This faculty is extended by the fact that many ISPs do not check for falsified source addresses in outbound packets. So here, we are mostly interested for the case where DNS transactions take place over UDP. DNS amplification attack is not applicable for the case of DNS over TCP protocol.

As depicted in Fig. 4.1, a typical amplification attack begins by instructing the attacking nodes to continually issue DNS requests which normally correspond to sizeable responses

(1). These requests are directed to preferably multiple NSs with the capability of serving recursive queries (2), e.g., open recursive NS or recursive ones (see section 2.1.7.1 ) that belong to the ISP the nodes are connected to. Nevertheless, the request is crafted in such a way that its source IP address is that of the victim. As DNS protocol is based on UDP, such fabrication is straightforward. Therefore, after the recursive NSs resolve the received requests (3), they are deceived into directing their responses towards the sufferer (4).

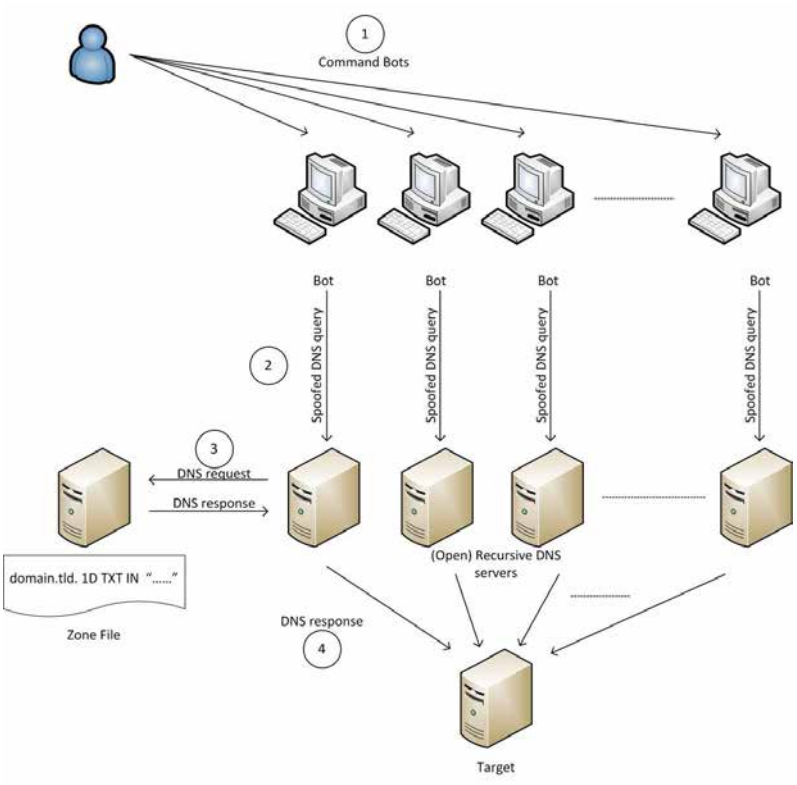



FIGURE 4.1: High-level architecture of a typical DNS amplification attack

### 4.1.1 Amplifiers and Reflectors

In the first documented study of DNS amplification attack, Vaughn and Evron [124] observed that the perpetrators tended to exploit large TXT RR as amplifiers and open recursive resolvers as reflectors. Namely, the attackers placed a sizeable TXT RR of about 4 KB in the DNS hierarchy and repeatedly inquired for the specific RR numerous open resolvers with the capability of serving recursive DNS queries. Moreover, each inquiry was crafted in such a way that it seemed to originate from the victim. The authors reported an amplification factor of 60. However, to achieve their goal the aggressors had

to control an ANS. This is possible by compromising an ANS and corrupting its zone file or by registering a domain of their own in order to insert a hefty TXT RR into its zone file.

Furthermore, the authors indicated that the analyzed attacks were able to create a flood of 2.8 Gbps in average, while in some cases the ingress traffic could reach an amount of 10 Gbps [124]. They estimated that the assailants were able to employ a maximum of about 140,000 open recursive NS as their reflector. Typically, to mitigate this threat, the network administrators are advised to disable open recursion [143]. Substantially, they should restrict the services of their DNS recursive only to the users of their internal network. However, as recorded by [144], to circumvent this restriction, the aggressors utilized a query for the NSs of root. The corresponding answer contains the list of the 13 root servers and their IP addresses. Its size is about 500 bytes, which provide an amplification factor of 8. By doing so, NS that support only non-recursive requests are forced to get engaged in the attack.

For augmenting the impact of their attack, the perpetrators aim to send a query which resolves to a response containing multiple RRs. Typically, the ANY query type is suitable for their objectives [145]. Since, this type of query essentially returns all the available RRs about the queried domain name. There is no guarantee about what types and how many records the response will contain. However, it is expected to be sizeable enough compared to the rest type of queries. To this direction the adaptation of EDNS0, which allows a NS to create a response with size by far larger than the basic limit of 512 bytes, contributes to the significant increase of the amplification factor. The reader can also refer to section 3.2.3.3 about EDNS0.

From the initial announcement of DNSSEC-bis there were concerns that its deployment would facilitate aspiring aggressors to mount improved DNS amplification attacks due to the increased record size. Ager et al., [146] calculated the packet overhead of a DNS response as a result of the new introduced DNSSEC-related record types. Likewise, Ariyapperuma and Mitchell [52] expressed worries about the size of a DNSSEC response. In the same direction, Cowperthwaite and Somayaji [147] evaluated the amplification factor of two DNSSEC-enabled NSs. They observed an amplification factor of 17.3, while they were expecting a much higher factor in the case the response fully complies with RFC 4033-4035 [14, 15, 16].

The first comprehensive study of DNS amplification attack involving DNSSEC-related RRs is given in [18]. As detailed in section 4.2, an attacker could exploit the increased size of DNSSEC-related RR for amplifying DDoS effects. Furthermore, they could take advantage of the vast number of open DNS forwarders existing out there as reflectors. Overall, it is shown that an attacker is able to utilize available resources (DNSSEC-signed zones and open DNS forwarder/recursive NSs) in the DNS infrastructure for accomplishing DDoS, without the need to register or install a NS of their own. This way, the forensic signal of the attack is minimized.

More recently, van Rijswijk-Deij et al., [141] highlighted that DNSSEC-related RR can be exploited to augment the amplification factor of a DNS amplification attack. Specifically, they calculated the provided amplification factor of almost 2.5 million DNSSEC-signed zones under 6 major TLSs. They concluded that ANY type query could provide an amplification factor of around 47 on average, while the worst case had a factor of almost 179. On the other hand, Rossow [148] evaluated the potential of exploiting 14 UDP-based network protocols for amplification attacks, including DNSSEC. From a sample of 1,404 ANSs crawled from IP space, they calculated that the top 10% of them provide an amplification factor of nearly 98 for ANY query type.

On the other hand, for reflecting their attacking traffic, the aggressors exploit open DNS resolvers. The work in Dagon et al., [36] in 2007 was the first to highlight on the open resolvers issue. The authors realized that almost 10.5M devices functioned globally as open resolver at each run of their probe. Utilizing a similar testbed environment, Takano et al., [149] detected in 2013 about 25M open resolvers. Primarily, those resolvers were located in APNIC and RIPE NCC Regional Internet Registry (RIR), while there was existing a portion that operated on obsolete version of naming software. Also, alarming was the fact that many open resolvers resided in networks suspicious for spamming campaigns.

In 2010, the initiative of Open Resolver Project begun its operation with the purpose to locate and shutdown open recursive resolvers. From the initial results, it was evident that nearly 30M devices functioned as open resolvers. Currently, after their effort to eliminate open recursion, the Open Resolver Project records about 17.5M open resolvers at each probe [150]. This fact is further verified by the DNS Factory Measurement [151].

Furthermore, Kührer et al., [152] examined the population of open resolver for a period of 13 months. During January 2014 and February 2015, the authors probed weekly the IPv4 space for locating devices that operate as open resolvers. The initial results indicated a number of 26.8M, while the final probe discovered about 17.8M distinct devices. The authors also identified that a number of 630K to 750K of devices functioned as open DNS forwarders. Further analysis of the Autonomous Systems (AS) that the devices were connected to, indicated that the majority of the resolvers were related with telecommunication or internet service providers worldwide. Moreover, software fingerprinting showed that from the identified DNS software a significant portion existed that operated vulnerable versions of DNS software. Regarding hardware fingerprinting, the same authors demonstrated that most devices were network equipment such as routers and broadband modems. This is an indication that devices supplied by network providers could be misconfigured as open resolvers without their users being aware of this fact (and the vulnerability it creates). There were also evidences that although the total number of resolvers remained similar through time, the corresponding devices migrate their IP address. The authors observed that only 4% of the initially discovered open resolvers were still alive after one year of experiments, namely maintain the same IP address. Essentially, 40% of the resolvers disappeared within one day, while half of them vanished in one week. Finally, the geographical distribution of resolvers' IP addresses illustrated that the Top 10 countries, including USA, China, Turkey, Mexico, Venezuela, and others hosted almost the half of their population.

In an effort to explain the existence of so many open resolvers Kaizer et al., [153] discovered that 17M out of 29M (almost 60%) identified open resolvers were ADSL modems made by well-known network equipment manufacturers. In fact, these home devices exhibited a strange behavior and responded to DNS request with different destination port than the source port of the query. This anomaly was considered as misconfiguration of Network Address Translation (NAT) implementation. Furthermore, their population was elevated to specific ASs, an indication that ISPs supplied their consumers with defective devices.

Despite the notable reduction of open resolver population, it is straightforward for an aggressor to acquire a large pool of them appropriate for their objectives. Rossow proved that someone needs only 92.5 sec for compiling a pool of 100K open resolvers [148].

At this point, we should note that ANSs are also frequently exploited as reflectors. The work in MacFarland et al., [154] showed that from a random sample of ANSs serving 2-TLDs belonging to 9 different g-TLDs, only 2.69% of the examined servers employed Response Rate Limiting (RRL) [155]. However, we have to acknowledge that at that time the RRL countermeasure was not implemented by popular NS vendors [145]. However, more recently, as it is detailed in section 4.3, a substantial portion of ANSs of TLDs are still poorly implementing or neglecting RRL mechanism. Hence, they constitute an alluring means of reflecting the attacking traffic of DNS amplification attacks. More details on RRL countermeasure mechanism are given in section 4.4.1.1.

Recently, the ANSs of root encounter an unusual type of DNS amplification attack. Specifically, several of the root NSs were flooded with a surge of queries. The queries were well-formed, valid DNS messages about a single domain name. However, their source IP address appear to be random and evenly distributed throughout the IPv4 address space. In total, the network traffic reached a peek of 5 million queries per second for each root NS involved in the attack [156]. It is believed that the utmost objective of the aggressors was not the end-clients receiving the responses but rather the root themselves which reflect the DNS traffic.

### 4.1.2 Victims of DNS amplification attack

A plethora of DNS amplification attack incidents have been reported over time against large corporations, banks, top ranked e-commerce sites, DNS infrastructure servers etc. A recent report by Symantec [157] confirms that DNS amplification represents a persistent threat and possess a prevalent place among the diverse types of DDoS attack. Specifically, the report indicates a remarkable raise on DNS amplification DDoS incidents during the period 2014-15.

The latter years, some notorious hacker groups have threatened to blackout the Internet by launching a DNS amplification attack against root servers [158]. Although for the moment such allegations are proven to be pretentious [159], no one can exclude the possibility this kind of attack to be used with the aim to tear down the operation of the NS of a critical domain name. For instance, in May 2007, US-CERT has received a report that Estonia was experiencing a national DDoS attack [160]. According to that source, the attacks consisted of DNS flooding of Estonia's root level servers. By that time 2,521

unique IPs have been identified as part of the attacking botnets. More recently, in a DNS amplification incident that is characterized as the biggest cyber-attack of this kind in Internet's history, the network infrastructure of Spamhaus was targeted [5]. Spamhaus, which is an organization responsible for blacklisting spam-related sources, sustained for at least a week period a network flood that reached 300 Gbps at its peak, originating from 30,000 unique DNS resolvers. The attack was considered as an act of retaliation on the part of blacklisted operators.

Finally, one should not overlook the fact that the third-party servers which are not the targeted victim per se, but are unwillingly entangled in DNS amplification incidents, are affected also from this type of attacks. Namely, DNS providers or ANSs of domain zones, whose infrastructure is exploited as reflector, are overwhelmed from the volume of ingress requests and high computation load due to DNS query resolution. Eventually, they get frequently incapable of serving their legitimate clients [161].

## 4.2 Going one step further: Obfuscating DNS amplification

As already explained in section 4.1 for a typical DNS amplification attack, the perpetrator needs to perform the following actions. First, they have to place in the DNS hierarchy at least one large RR, either by compromising an ANS or by registering a domain zone of their own. Usually, they prefer to exploit a RR of TXT type, since this type has variable length and is trivially constructed. Second, they need to recruit a botnet by distributing a malware. The final step is that of locating a pool of open recursive NS. However, it is conceivable that the first two steps are bound to leave traces that might put the attacker at the risk of being detected.

In the following, we present a new flavor of DNS amplification attack which grants the privilege of almost full anonymity to the attacker. Among others, the main advantage of our proposal compared with the standard type of the attack as described in the literature so far is that it does not disclose any illegal or suspicious activity during its execution. Namely, the network flow during the coordination of the attack seems to be perfectly legitimate, except of course the attacking traffic itself. Moreover, the attack is very hard to be traced back to the perpetrator who, as a result, enjoys the advantage of anonymity.

Specifically, the attack scenario is separated into two parts. First off, the aggressor needs to perform a degree of reconnaissance to identify the devices, in a specific geographical area or IPv4 address block, which operate as open DNS forwarders. Second, they need to repeatedly send spoofed queries of ANY query type requesting DNSSEC-related RR [14, 15, 16] to this pool of forwarders. The forwarders consider the victim as the originator of the queries because the (spoofed) source address of the query packet contains the victim's IP address. Overall, the outcomes we obtained indicate that with proper planning and a relatively fair amount of resources, an attacker is capable of creating a large torrent of bulky DNS responses towards its target. Certainly, as shown in the next subsections, the power of the attack increases proportionally to the number of attacking nodes.

### 4.2.1 Attack Scenario

As already pointed out, the attack scenario is divided into two independent phases. First off, a large pool of IP addresses belonging to network devices that operate as (open) DNS forwarders needs to be collected. Recall that a DNS forwarder accepts DNS queries, and after it consults a DNS recursive, it returns the appropriate answer to the initiator of the request. Bear in mind that usually DNS forwarders afford cache capabilities as well. Namely, as explained in section 2.1.5, they cache the received RRs with the intention to fulfill subsequent similar requests.

The discovery process of DNS forwarders is akin to that given in [36]. First, for the countries of interest, we acquire their block of IP addresses. A straightforward way to do so is to utilize data from www.countryipblocks.net. In our case, as explained later in this section, these countries are Greece, Ireland and Portugal (in alphabetical order). Next, a DNS query is dispatched for a given RR to each IP address in the country-list. Specifically, the requested RR belongs to a domain zone under the attacker's administration. The first *label* of the domain name in the question section (QNAME) contained in the request is an indication of the IP address of the device that the packet is headed to. Moreover, the request has the DO flag enabled. This flag designates if the machine being queried supports DNSSEC. Therefore, by doing so, we distinguish which forwarders are able to send back DNSSEC-related RR. On the other hand, with the help of a typical

packet sniffer we capture the DNS requests reaching our authoritative NS. These requests are trying to resolve the queried RR of the previous step, meaning that they are originated from devices that have the ability of resolving DNS queries recursively.

Comparing the IP address contained in the QNAME with that of the source IP address of the request, one can determine whether the device operates as recursive NS or as forwarder. That is, in the case of a request originated from a recursive NS both IPs are identical. Otherwise, if the IP addresses are different, the examined machine operates as forwarder. The resolution of RRs that differ in the first label of the QNAME is performed with the help of wildcards following the directions given in RFC 1034 [1]. This situation is exemplified in Fig. 4.2. Specifically, the client asks every IP (excluding reserved IPs like 127.x.x.x) found in the IP blocks of the country of interest, to resolve the corresponding queried domain name. Every device that operates as open recursive NS or open forwarder receiving the query will undertake to resolve it. But to do so this device will send the query to the attacker's ANS informing them that this IP truly belongs to a machine that acts as a recursive or forwarder.

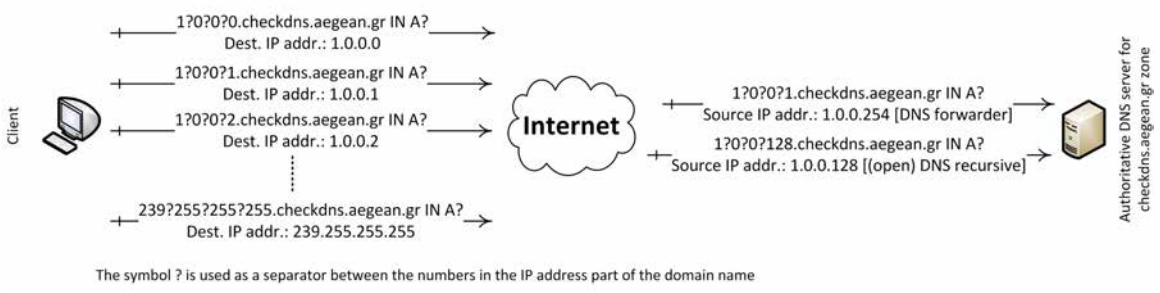



FIGURE 4.2: DNS forwarders discovery process

In the incoming queries, the EDNS0 section of the packet is examined. In fact, we select only those forwarders that are able to handle queries with DO flag enabled (i.e., they support DNSSEC) and simultaneously advertize a large EDNS0 buffer size (e.g. 4096 bytes or larger). As we have already discussed in section 3.2.5, DNSSEC-related RR (RRSIG, DNSKEY, DS, NSEC/NSEC3) are large in size [17]. As a result, our aim is to filter and keep only forwarders that support DNSSEC and are configured to respond with large payload size. These devices are appropriate for our objectives, and thus they will be used in the latter phase of the attack in order to augment its amplification factor.

After compiling the pool of preferable forwarders, one is ready to launch the actual attack. To do so, we utilize a network of attacking nodes that repeatedly send DNS

requests of ANY query type for DNSSEC-related records toward the forwarders contained in the final list. Though, the source IP address of the packet is being spoofed to the IP of the victim, so that all replies are eventually reflected toward the target. The requested domain names are related with TLD zones that have already adopted DNSSEC and their corresponding ANSs are verified to provide sizeable responses. The process of locating the desired zones and afterwards examine the size of their reply is straightforward, because the information of which zones have adopted DNSSEC is publicly available.

We execute two variations of the attack scenario depending on the destination port the attack flood is delivered. In the first one, this port is 53, while in the second is totally random. Bear in mind that typically all DNS queries are sent from an ephemeral source port (≥49,152) to destination port 53, while responses are sent from source port 53 to the same ephemeral, but this time, destination port. The tool used for coding the script that fabricates DNS packets is Scapy [162]. Algorithm 2 presents a pseudocode version of the script, while Fig. 4.3 depicts the actual way the attack unfolds.

**Algorithm 2** : Scapy pseudocode

```
1:  procedure SendSpoofedPacket(LIST)
2:      READ Forwarders IP Addresses from LIST
3:      while notEndOf(LIST) do
4:          CREATE UDP_PACKET
5:          UDP_PACKET.DestinationAddress ← IPAddress
6:          UDP_PACKET.DestinationPort ← 53
7:          UDP_PACKET.Protocol ← DNS
8:          UDP_PACKET.DNS.RD ← 1 /*Recursion is desired*/
9:          UDP_PACKET.DNS.QR ← QUERY
10:         UDP_PACKET.DNS.QNAME ← "DNSSEC − enabledTLD"
11:         UDP_PACKET.DNS.QTYPE ← ANY
12:         UDP_PACKET.DNS.QCLASS ← IN
13:         UDP_PACKET.EDNS_FLAG ← DO /*Enable DNSSEC*/
14:         UDP_PACKET.DNS.SourceAddress ← 195.251.161.155 /*IP address of target*/
15:         UDP_PACKET.DNS.SourcePort ← 53 /*version 1, port of DNS service*/
16:         //OR for the second variation of the attack
17:         UDP_PACKET.DNS.SourcePort ← Random /*version 2, Not a specific service*/
18:         SEND UDP_PACKET
19:     end while
20: end procedure
```

### 4.2.2 Results

For compiling the pool of forwarders, we considered to examine the network IP blocks of three European countries which have more or less the same allocation of IP addresses, but are expected to differ on the level of security awareness. In fact, this assumption

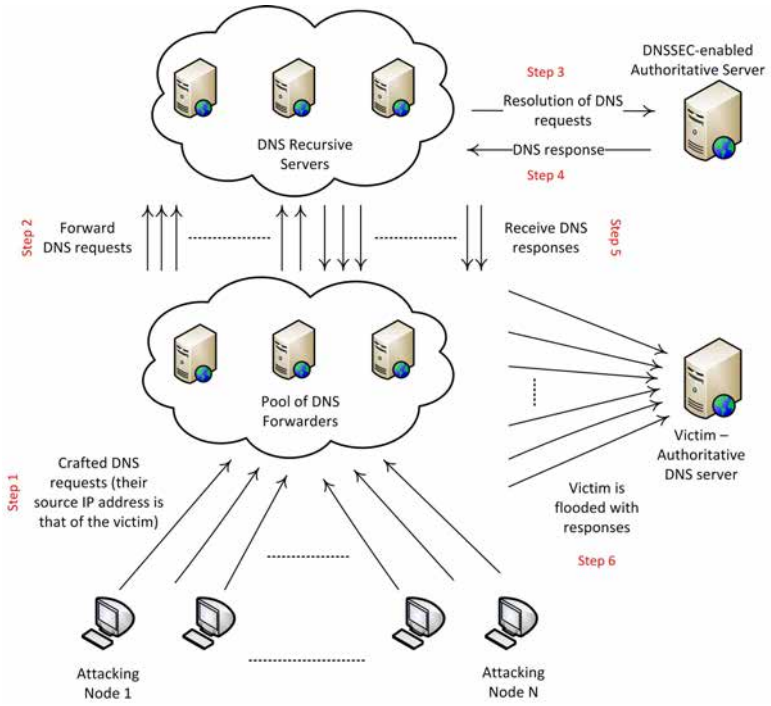



FIGURE 4.3: High-level architecture of the attack introduced in this work

is proved to stand true by the results presented in the following subsections. Those countries are Greece, Ireland and Portugal. We test the first phase of the attack (i.e., detection of open forwarders) several times in different days (working days, weekends and holidays) and time zones (working hours and nights), with the purpose to figure out whether their existence is something ephemeral or ordinary. This procedure have been performed twice in a time frame of six months. In the case of Greece, we detected about 60K open forwarders on average per execution. For Portugal the probing process returned about 35K, while for Ireland 10.5K forwarders on average. The exact numbers per country used for implementing the attack are given in the last line of Table 4.1.

To further investigate the contribution of each forwarder to the efficacy of the attack, we examine the size of the responses these devices return to the DNSSEC-related query. These results are also summarized in Table 4.1. As it can be observed from the table, a small but not negligible portion of the forwarders return an answer that exceeds 2,900 bytes. For instance, this number for Greece is 1,094, while for Ireland is much smaller, about 65. In any case, these forwarders are very important to be included in the arsenal of the described attack, as they present two significant benefits. Firstly, by exploiting them an attacker is able to accomplish an amplification factor of at least 40 (assuming an average size of 70 bytes per DNS request). Secondly, due to its large

size, the response is fragmented into three or more IP datagrams. This means that the reassembling (and perhaps reordering and fragment loss) process of packets also conduces to the consumption of resources at the victim side. An attacker is able to integrate this filtering of forwarders in the first phase of their attack as the case may be.

| Size of response in bytes | **Amplification Factor** | **Greece** | **Portugal** | **Ireland** |
|---|---|---|---|---|
| $< 1000$(or No response) | $< 14$ | 42,569 (69.95%) | 25,983 (74.17%) | 8,809 (82.35%) |
| $1000 - 1500$ | $14 - 21$ | 15,112 (24.83%) | 6,603 (18.85%) | 963 (9.00%) |
| $1500 - 2000$ | $21 - 28$ | 1,962 (3.22%) | 2,205 (6.29%) | 802 (7.50%) |
| $2000 - 2500$ | $28 - 35$ | 80 (0.13%) | 26 (0.07%) | 42 (0.39%) |
| $2500 - 2900$ | $35 - 41$ | 41 (0.07%) | 9 (0.03%) | 16 (0.15%) |
| $\geq 2900$ | $>= 41$ | 1,094 (1.80%) | 204 (0.58%) | 65 (0.61%) |
| | **Total forwarders:** | 60,858 (100%) | 35,030 (100%) | 10,697 (100%) |

TABLE 4.1: Percentages of open forwarders per country in regards to the size of response they return

As already mentioned, the amplification factor is the most crucial element for an attack to be effective. With the purpose to better estimate its magnitude in the context of the attack described in this section, we initially run only one instance of the script for both attack variations and counted how many responses arrived at the victim and what their size was. More specifically, the script dispatched a single DNS query towards the 1,363 forwarders given in the last but one line of Table 4.1. As already pointed out, each DNS query packet created by scapy has a size of approximately 70 bytes. Considering the first variation of the attack, the total number of packets arrived at the target machine reached 3,110 packets having a total size of 3,526,046 bytes. For the second variation, we recorded 3,539 packets with a total size of 4,187,901 bytes. Therefore, it can be safely argued that utilizing equation (4.2) the amplification factor for the first version of the attack is almost 37, whereas for the second is nearly 44.

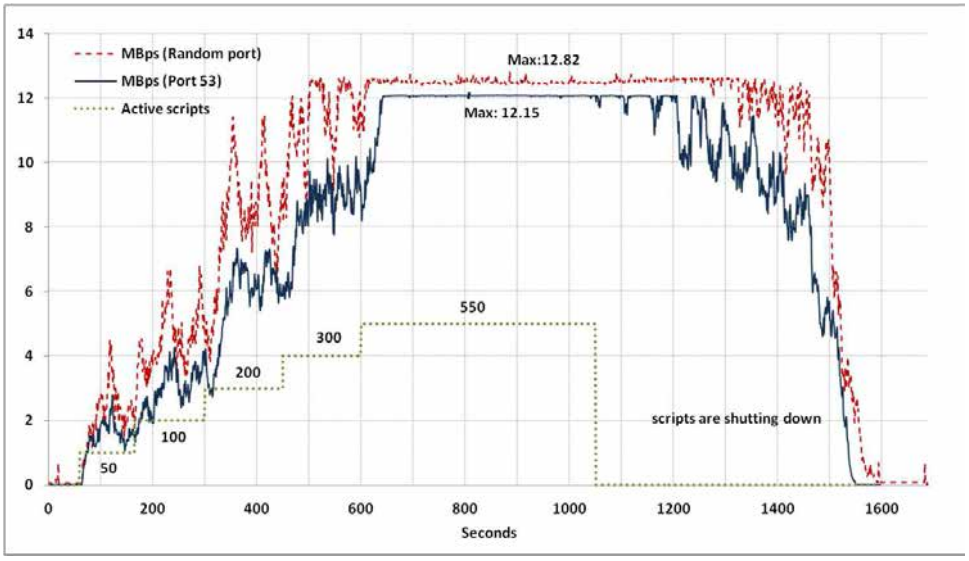
From the above result, one is able to observe that for every query the attacker dispatches, the target receives 3 packets that are reassembled in one DNS response sized about 3,100 bytes. Furthermore, the volume of the incoming packets is a little smaller in the first variation of the attack, which means in the case the destination port of the responses is not 53 but rather random, we are able to accomplish a slightly bigger amplification factor. This difference is anticipated as many firewalls are configured to block egress DNS queries originated from sources other than their internal network recursive NS.

For the needs of the actual attack, we utilized 22 typical Personal Computers (PC) each one connected to a 100 Mbps network interface. Our tests demonstrated that every attacking node is capable of sending nearly 880 DNS queries per second on average, or 61.6 KBps. This is the case when a device runs 25 instances of the attack script simultaneously. In the ideal case, this means that each attacking node is capable of flooding the target with 880 DNS responses per second. However, this data volume is magnified by the amplification factor, that is, 37 and 44 for each attack variation correspondingly. Consequently, a single attacking node unleashes on average a stream of 2.28 and 2.71 MBps respectively towards the victim. In order to investigate the accumulative impact of each node that joins the attacking group, we progressively triggered the scripts on each one of them.

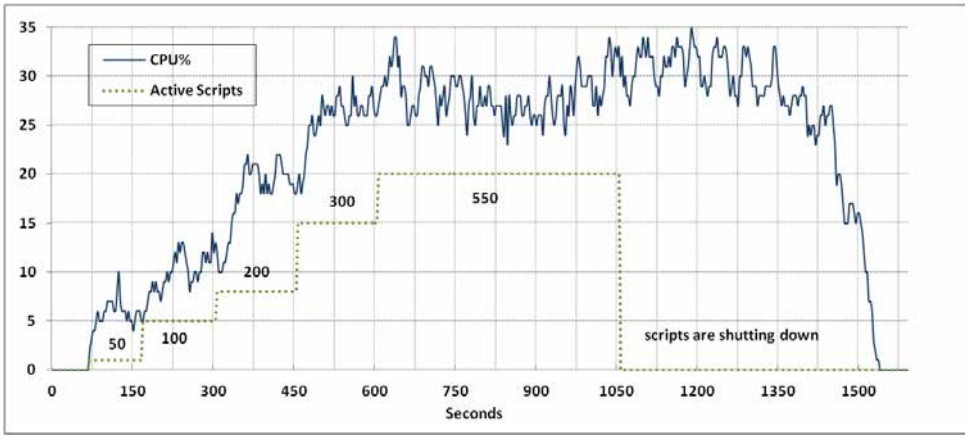
On the other side, the target, acting as DNS authoritative NS, was a desktop machine equipped with a Dual 2.8 GHz CPU and 4 GB RAM connected to 100 Mbps network interface. This NS had DNSSEC extensions enabled. Table 4.2 summarizes the progress of the attack for both of its variations. Regarding the first variation, besides the inbound traffic, we also recorded CPU overhead caused by *bind* process, i.e. how much the incoming unsolicited DNS packets affect the performance of the victim as ANS.

Figure 4.4 depicts the level of resource consumption at the victim-side during both attack variations. More precisely, Fig. 4.4(a) shows the incoming traffic in MBps for both variations, while Fig. 4.4(b) depicts the CPU consumption due to *bind* process, which runs as a service on port 53. Time 1050 signals the moment that the initial scripts are beginning to terminate. However, the effects linger to recede because, as explained further down, the use of more than 12 attacking nodes inflict the same impact on this particular target.

As we can easily deduce from Table 4.2, with a small number of attacking nodes we were able to exhaust the network bandwidth of the victim's machine. To put it another way, for both attack variations, it is apparent that only a dozen of nodes are capable to flood a 100 Mbps network. More importantly, as it can be observed from Fig. 4.4(a), the addition of new attacking nodes to the testbed does not achieve cumulative effects on the target. In fact, in the last stage of each scenario, the network was so overflowed, that a great amount of fragmented packets was dropped and the ratio of the crafted requests versus the ingress corresponding responses is fallen below 1:1. Actually, due




(a) Incoming traffic in MBps for both attack variations




(b) CPU consumption (port 53)

FIGURE 4.4: Progress of Resource Consumption at the victim-side

to fragmentation, this ratio is anticipated to be 1:3, which for the initial phases of the attack scenario stands true. Comparing the two attack variations, we can put forward that for the second one the volume of the flow has augmented by 45% on average with relation to the first variation. This is obvious when comparing the values contained in the third and sixth column of Table 4.2. As a matter of fact, this applies to the first three phases of the attack, because in the latter phases the network was so monopolized leading a great amount of packets to be discarded. However, for the first variation of the attack, in addition to the intense increase in the volume of incoming traffic, the impact on performance is also significant. This is because when a DNS response arrives at (standard) port 53 it is being processed by *bind*, thus consuming resources on the

victim's machine. In contrast to this, when a DNS packet arrives at a random port (second variation of the attack) it is simply dropped. Therefore, in the case the victim is an authoritative or recursive NS, the most productive way of performing the attack is to use the standard destination port.

Moreover, during attack escalation, we queried a group of 75 open recursive NS for unique RR contained in the victim's domain zone. Hence, these queries need to be resolved by the victim. This would give us a clear estimation on how the sufferer - being under continuous flooding - will behave when trying to serve legitimate DNS requests. Thus, we record the average time a request needs to be fulfilled, as well as the ratio of the abortive queries. That is how many queries are lost and return a "connection timed out, no servers could be reached" error. Clearly, the average query time increases proportional to the volume of flooding traffic the victim undergoes. It is remarkable to point out that at the latest stages of the attack, where 550 instances of the script were active, almost the half of the queries were lost.

| | **Scenario 1: port = 53** | | | | **Scenario 2: port = Random** | | |
|---|---|---|---|---|---|---|---|
| **Number of attacking nodes** | **CPU utilization (*bind*)** | **Inbound traffic** | **Average query time** | **Loss Packet Ratio** | **Inbound traffic** | **Average query time** | **Loss Packet Ratio** |
| None | 0% | 0.5 KBps | 119 msec | 0% | 1.2 KBps | 113 msec | 0% |
| 2 (50 scripts) | 5.65% | 1.48 MBps | 125 msec | 0% | 2.16 MBps | 118 msec | 0% |
| 4 (100 scripts) | 10.8% | 2.91 MBps | 128 msec | 0% | 4.44 MBps | 128 msec | 0% |
| 8 (200 scripts) | 20.1% | 5.67 MBps | 137 msec | 0% | 8.06 MBps | 149 msec | 0% |
| 12 (300 scripts) | 25.78% | 8.47 MBps | 156 msec | 0% | 11.28 MBps | 157 msec | 40% |
| 22 (550 scripts) | 30.94% | 12.05 MBps | 241 msec | 41% | 12.50 MBps | 243 msec | 42% |

TABLE 4.2: Effects on target proportional to the power and number of attacking nodes per scenario

Each individual stage of the amplification attack, corresponding to the gradual activation of new attacking nodes as described in Table 4.2, lasts for approximately 2 minutes, while altogether the various stages have a duration of about 27 minutes. During this time, the attacking nodes via the use of the scripts dispatched about 14,993,000 DNS requests. If we consider that on average our DNS request has a size of approximately 70 bytes, in overall the attacking nodes dispatched approximately 0.97 GB of network flow toward the 1,363 DNS forwarders contained in the joined list obtained for all three countries. However, regarding the first variation of the attack, the victim was flooded with 10,507,706 packets of roughly 12.8 GB volume, while in the latter the victim suffered 10,999,752 packets of nearly 14.2 GB. Amongst other reasons, it is evident that we have

a loss of roughly the half of the volume due to the excessive traffic. This is also verified from the fact that in the last step of the attack 42% of the DNS queries were timed out.

### 4.2.3 Discussion

The first phase of the attack described above, sadly exhibits (and verifies the claims of section 4.1.1 that a worryingly large number of machines operate in the open Internet as DNS forwarders and do serve DNS requests originating from sources outside their network. Further analysis of the hardware and Operating System (OS) of these devices reveals that their majority are the so called Small Office Home Office (SOHO) network devices, such as network printers, ADSL routers, NAT (Network Address Translation) devices etc. OS fingerprint of the forwarders with XPROBE2 tool indicated that 75% of the forwarders in Greece, 45% in Ireland and 55% in Portugal have HP JetDirect, Foundry Networks IronWare OS or Cisco IOS as their OS. In any case, these devices erroneously or due to misconfiguration operate as DNS proxies. Moreover, WHOIS analysis attests that most forwarders belong to AS of ISP networks. Actually, the possibility of malicious exploitation of a device in order to turn it into an open forwarder is highlighted in a very recent Common Vulnerabilities and Exposures (CVE) report issued by NIST [163]. This report notifies the corresponding vulnerability for DNS related library, but it is quite possible this security weakness to stand true for other software as well.

As we can observe from Table 4.2, at the time that a dozen attacking devices were running, the victim had to cope with an ingress traffic of nearly 10 MBps that overwhelmed the victim's network bandwidth capacity. Even though this rate cannot be considered as a successful DoS attack for a real target connected to a Gbps network interface, we believe that with proper scaling of the attacking network, a determined attacker is able to achieve a very large volume of DNS flow. For example, think of a case where a large group of ill-motivated persons scattered around the world start simultaneously running several instances of the attack script. If we consider that a simple PC, as those used in our experiments, is capable of dispatching a maximum of 880 DNS queries per sec on average, ideally this machine could contribute a 2,5 MBps stream toward the target. Naturally, this rate is the upper limit according to the testbed used, but with proper equipment and a larger pool of forwarders (more countries under consideration)

the perpetrator could easily exceed these limits and paralyze the victim. Besides, the amplification factor of the attack - which is independent of the number of attacking nodes, and as already mentioned, is in the order of 37 to 44 - is self-evident about its effectiveness.

As already emphasized, the advantage of our proposal compared to that of standard amplification attack given in section 4.1 is the elimination of all traces that can be used toward disclosing the attacker's actions. Essentially, there is no need to infect any machine with malware in order to turn it into a bot. For assembling a botnet, the aggressor only needs to recruit in a transparent to them manner the available forwarders existing out there. This is also strongly in favor of the attack as the usage of the forwarders conceals the attack's real source. Thinking of a large number of forwarders participating in the attack, the only reaction left to the victim is to block the inbound traffic arriving from numerous sources. This for example could be done by instructing the firewall to ban the traffic originating from certain but constantly changing IPs. Nevertheless, in practice this could be proved quite hard to achieve. Also, the attacker does not have to penetrate into an authoritative in order to place a large RR in the DNS hierarchy. Instead, to intensify the amplification factor of the attack, one simply has to exploit the existence of large DNSSEC-related RRs.

Moreover, the recursive NSs that provide the large responses to the forwarders do not possess any record of the attacker's actions. They only observe legitimate queries coming from devices residing in their internal network. Provided that some of the forwarders may have caching capabilities (this is the usual case), the forwarders do not even consult continuously the corresponding recursive NS, but only for the very first query. Finally, with the presented attack scenario one becomes able to circumvent the countermeasures against amplification attacks employed by the majority of recursive NS. This is because the attack does not entangle any recursive NS but only forwarders, meaning that any countermeasure deployed on the recursive-side is not applicable in this case. Since, all recursives are queried not directly, but through devices located in their inner network, we involve them in the attack like they operate as open recursive NS (see Fig. 4.3). To put it in other way, these recursive NSs usually do not function as open, but by following the aforementioned strategy we force them to act like they do. Moreover, our attack is also immuned to the recently proposed DNS DNS RRL [155], which normally is integrated in the functionality of ANSs.

One can argue that the first phase of the attack described in section 4.2.1, i.e., the discovery of forwarders, is very noisy (due to its large volume of packets it produces) and easily detectable. However, this is partially true. During the experiments of the first phase, we sent legitimate DNS queries to all IP address allocated for all of the countries in question. We executed this process twice during a six month period. However, as already explained in section 5.8.3, each time we carried out the discovery process for seven different occasions to include different days and time zones. This means, that we sent almost 160M DNS packets towards IP addresses of these countries. However, the detection of such legitimate but unusual traffic is entirely up to the administrator of the corresponding network domain. From our experience in the conducted experiments described above, only one network administrator noticed our bizarre queries and notified our abuse list. Also, this information gathering process may be quite more chronologically separated from the actual attack, meaning that the exact time the attack will be unleashed against its target is entirely up to the aggressor.

As previously mentioned, to the best of our knowledge, the only work in the literature that analyzes data stemming from real DNS amplification attacks is given in [124]. In this work the authors report a maximum amplification factor of nearly 60 but they exhibit a DNS query size of 60 bytes. It should be noted that for achieving the aggressors placed a large TXT RR in the DNS hierarchy that had a fixed size of 4000 bytes. On the other hand, according to the scenario at hand the response's size depends on the DNSSEC related resource records that the forwarders opt to include in the packet. That is, the attacker does not need to place a large RR, but rather to exploit a DNSSEC enabled zone.

## 4.3 Authoritative TLD nameserver-powered DNS amplification

Certainly, the authoritative NS of popular domain zones, and in particular the DNSSEC-enabled ones, do not elude the attention of DNS amplification attackers for entangling them in their actions. The ANS list of TLD are publicly accessible in the form of root.zone file, so even a casual attacker is able to acquire the list of TLD zones and their matching ANSs. In this context, in the following we examine the potential of

ANSs of TLD to be unknowingly engaged by attackers in DNS amplification attacks. In particular, using two distinct versions of the root.zone file, we assess the amplification factor that these entities may produce when replying to both individual and multiple queries.

While until now several works in the literature deal with DNS amplification and examine DNSSEC-enabled zones for providing data alluring to amplification attackers, to our knowledge none of them addresses the potential of taking advantage the authoritative NSs of TLD as both amplifiers and reflectors. From an attacker's viewpoint, the motivation is simple; the list of these ANSs are publicly known, so it can be obtained by anyone, anytime. Even a unsophisticated attacker can efficiently compile the pool of TLD ANSs by downloading the root.zone file [39] and then extracting the RRs of NS type along with their corresponding glue records.

Today, the majority of TLD zones have already adopted DNSSEC [101], and thereby by default these ANSs respond to DNSSEC-related queries. This gives more opportunities to prospective attackers to maximize the amplification effect of their attack. As a result, the aggressor does not have to extensively crawl DNS hierarchy for locating DNSSEC enabled zones, but rather simply utilize those contained in the corresponding root.zone file. As TLD zones constitute the pillars of Internet, they utilize rich computational and network resources and cutting edge techniques (such as clustering and anycast routing) for accommodating the high-load demands. This fact makes them even more tempting to potential perpetrators having the aim to entangle them in DDoS incidents.

Also, it should not be overlooked the fact that FQDN of TLDs is comprised of only one label with 2 or 3 characters for the majority of the domains. Consequently, the size of the query packet is limited compared with the commonly used domain names. Since this size is inversely proportional to the amplification factor, the lesser it is the larger the amplification factor. More importantly, since the authoritatives of TLDs integrate the foundation of DNS hierarchy, they are involved in every DNS resolution no matter what the inquired name is. This means that it is rather futile to blacklist the traffic originating from an authoritative exploited as reflector because it also provides crucial data to legitimate users. On the contrary, in the case that only open DNS resolvers/forwarders are utilized as reflectors, like these recorded by [150] and crawled from Internet [164], it is quite easy for the defenders to ban the incoming traffic originated by them.

In the following section, we assess the potential of exploiting the authoritatives of TLDs as both amplifiers and reflectors. First off, we measure the volume of the response packet to a single DNS request for ANY and DNSKEY RR types which are expected to convey large data. Following, we examine and evaluate the degree of adoption of RRL mechanism which constitutes a defensive barrier from amplification attacks on the side of ANSs. This is done by dispatching a stream of consecutive DNS requests during a limited time window and observing the number of complete, truncated and missed responses. For the latter case, we take into consideration both positive and negative answers, namely, we inquire for both existent and non-existent domain names.

Figure 4.5 presents a typical DNS amplification attack, which is nevertheless amended to illustrate the involvement of ANSs. The left side of the figure shows the way an ANS is exploited as reflector, while the right side exemplifies the ANS as amplifier.

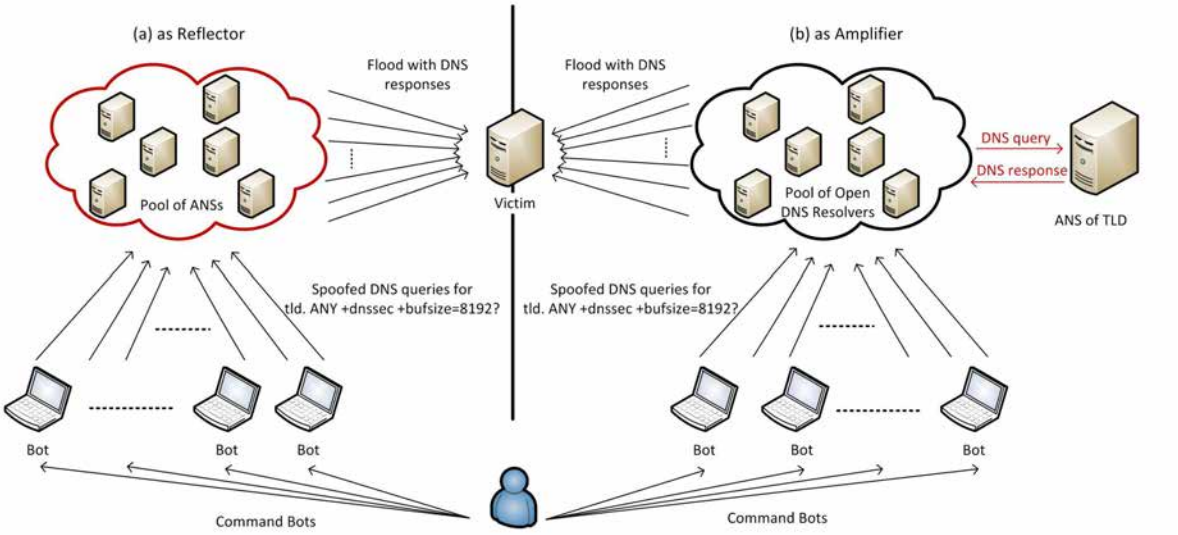



FIGURE 4.5: Exploiting ANSs administrating TLD zones

So far, two are the prevalent and simplest to implement countermeasures for preventing a NS to be abused as amplifier and/or reflector; truncated responses (TC) and DNS RRL [155]. For TC countermeasure, RFC 5966 [165] recommends that if a UDP response is larger than 512 bytes then the responsive NS should truncate its response and enable TC flag for signalling the requestor to retry in TCP mode. In this case, the NS responds with a minimal size packet and enforces the requestor to repeat its query in a manner that is infeasible to spoof its IP address, in the case he desires a sizeable answer. Capitalizing on this functionality, in section 4.3.1.2 we investigate which ANSs in our list enforce truncation, that is, TC flag in their response is on. This is measured when an individual

(unique) query is issued toward them, not a stream of it. As explained right after, this is because a stream of queries would trigger the RRL mechanism, which mandates the truncation of some of the replies.

On the other hand, DNS RRL limits the identical responses that can be returned by an ANS to the same requestor within a time interval. As explained, RRL is applicable to ANSs only, since the normal DNS flow towards an authoritative is expected to contain limited duplicate queries from the same recursive resolver. This is due to the caching facility every recursive affords, allowing it to locally store RRs for fulfilling subsequent requests.

### 4.3.1 Methodology and Results

Initially, our approach consists of an reconnaissance phase to compile the list of profitable ANSs. Specifically, we record which NSs are able to respond with a sizeable answer, and thus how noteworthy is the generated amplification factor. Next, we aim to deduce the adoption degree of RRL mechanism for the case of requests that return positive and negative answers. So, our goal is to examine if and up to what degree RRL is implemented by ANSs as a countermeasure.

Our experiments span over a period of six months and cover two versions of root.zone file [39]. The first one corresponds to 2015090700 serial number of SOA record type, while the second to 2016021300. We execute our experiments several times in different days and time zones (working hours and nights), with the purpose to figure out whether the investigated ANSs exhibit a constant behavior or their performance is affected by workload peaks.

The first version of the root zone file contains 5,103 unique records of NS type, which are accompanied with 2,431 glue records of A type. These RRs correspond to 1,056 unique zones (including root zone "." ) that are served by 2,063 servers having a distinct IPv4 address. The second and newer version of the root file consists of 5,962 NS RR with 3,150 glue A RRs. The total number of discrete zones are 1,227 served by 2,755 servers (distinct IPv4 address). From these numbers it becomes evident that some ANSs are hosted to the same IPv4 address or there exist ANSs that map to two different IPv4 addresses. The fact that plenty of the NSs act as authoritative for multiple zones does

not affect our results as RRL mechanism examines the similarity of the responses, which in this case vary in the queried domain name. In total, we extracted 5,079 and 5,938 distinct tuples of <domain name, IPv4 address> respectively. Note that the first and the second version of root.zone file correspondingly contain 3 and 2 authoritatives that only afford IPv6 address, and therefore were excluded from further investigation. Those ANSs are included in 35 and 34 records of NS type respectively.

#### 4.3.1.1 Types of Queries

The most alluring types of DNS queries for an amplification attacker are ANY and DNSKEY. The first one essentially returns all the available RRs about the queried domain name. There is no guarantee about what types and how many records the response will contain. The latter type retrieves the RRs that contain the public keys of the inquired zone. Obviously, the public keys are necessary for the validation of DNSSEC-related RRs. The response is expected to be sizeable, because RFC 6781 [114] suggests the usage of RSA public keys with key length of at least 1024 bits. Also, an aspiring aggressor could exploit the response for non-existent domain names. Normally, an authoritative returns a negative answer in the form of NXDOMAIN status, or in the case of DNSSEC, includes NSEC/NSEC3 RR type in its response. Typically, such a response is estimated to contain up to 2 NSEC or up to 3 NSEC3 [131] with the accompanied RRSIG records, and therefore is expected to be sizeable enough.

#### 4.3.1.2 Examining the response size for a single query

For each extracted pair of <domain name, IPv4 address> corresponding to an ANS, we issue directly to the IPv4 address a DNS query about the matching domain name for both types of records as detailed in section 4.3.1.1 with the DO bit enabled. That is, we indirectly force the ANS to include DNSSEC-related RR to its response. We repeat the execution three times, each of them advertising a different buffer size of EDNS0 extension mechanism, i.e., 1,024, 8,192 and 65,535.

As summarized in Table 4.3, the great majority of the available ANS in each version of zone file produces no error responses. However, a tiny portion of them demonstrate no interest for the context of our experiments. Specifically, several ANS returned FormErr,

ServFail or Refused response status, meaning that they are unable to process our request or do not tolerate ANY query type. Also, it seems that an approximately 10% of the authoritatives still do not support EDNS0 extension, namely they are unable to provide responses with size larger than 512 bytes. Additionally, as observed from Table 4.3 a constant portion of approximately 3% of ANSs timeout to respond. Essentially, plenty of the latter ANSs do not respond to any of our request during both phases. Therefore, an aspiring aggressor will apparently preclude those ANS from his arsenal since they exhibit limited amplification capabilities. Nevertheless, for the sake of completeness, we include them to the subsequent analysis. Note that if an authoritative fails to respond to even one of our queries, we flag it as 'timeout'. This means that in the case an ANS is not always reachable, this ANS is not suitable for the objectives of the attacker.

| | **Ver. 1** | **Ver. 2** |
|---|---|---|
| NoError | 1,939 (93.99%) | 2,656 (96.41%) |
| NoEDNS0 | 11 (0.53%) | 10 (0.36%) |
| FormError, ServFail or Refused | 36 (1.75%) | 14 (0.51%) |
| TimeOut | 77 (3.73%) | 75 (2.72%) |
| **Total** | 2,063 (100%) | 2,755 (100%) |

TABLE 4.3: Demographics of the TLD ANSs

As pointed out, as the size of the query packet is limited, the amplification factor will be elevated. Nevertheless, one has to consider that the usage of EDNS0, which is mandatory throughout our experiments as we seek for DNSSEC-related RR, adds a burden of 11 bytes. Clearly, the smallest packet size is that of the query packet for the root zone, which is 28 bytes. On the downside, the largest one is 53 bytes and corresponds to a zone which has 24 in section 4.1, to calculate the amplification factor provided by each ANS we apply formula (4.1). So, along with the size of the response, the size of the query packet is recorded. Typical DNS queries have a size of around 20-60 bytes [141]. However, commonly, a domain name consists of at least two labels from which the second-level label is comprised by a number of characters. In our case, as we deal with TLD zones, the inquired domain names contain solely one label with two or three characters in most cases. Hecharacters in its domain name. The average packet size for queries of distinct domain names is 34.3 bytes.

As expected, all ANSs comply with the limit of 1,024 bytes for both types of query, when the buffer size of EDNS0 is 1,024. However, for larger values of buffer size, it is observed a sizeable response packet. The majority of ANS responded similarly (same

size of packet) to both runs for the ANY query type, while only 12 tuples out of 5,079 displayed a packet size difference of at least 500 bytes. The corresponding variation for the DNSKEY query type is also negligible.

For both versions of zone file, the largest response packet size has a size of 7,728 bytes, containing in its Answer Section 32 answers and 37 additional RRs. Note that the corresponding answer consists of 6 IP fragmented packets. Therefore, one can expect that the authoritative can provide a hefty amplification factor of up to 249.3, as the triggered query packet has a size of only 31 bytes. The aforementioned response was received for a query of ANY type, no matter what was the defined buffer size (i.e. 8,192 or 65,535). The outcomes of the amplification factor imposed by the responses of ANSs of TLD zones are outlined in Table 4.4. We choose to present the results for the queries with buffer size of 8,192, since the difference amongst the two runs are insignificant and a potential aggressor will probably prefer to advertise minimum possible buffer size for avoiding suspicions. Also, a larger buffer size will probably create reachability problems due to fragmentation [166].

For the examination of the amplification factor in the case of negative answer, we select to query for a domain name that has a random string of 5 characters as second label. Thereby, we intend to increase the possibility that truly the random domain name is nonexistent. In any case, in total the size of the query packet increases by 6 bytes, that is 5 bytes for the random string and 1 byte for the extra label. Table 4.4 also presents the amplification factor of ANY query type for negative responses (labelled ANY NSEC) for both versions of root.zone file.

| **Amplif. Factor** | **Ver. 1** | | | **Ver. 2** | | |
|---|---|---|---|---|---|---|
| | ANY | DNSKEY | ANY NSEC | ANY | DNSKEY | ANY NSEC |
| <=20 or TimeOut | 1,362 (26.82%) | 1,270 (25.00%) | 2,605 (51.29%) | 1,173 (19.75%) | 1,044 (17.58%) | 2,976 (50.12%) |
| 20−40 | 280 (5.51%) | 3,247 (63.93%) | 2,419 (47.63%) | 367 (6.18%) | 4,254 (71.64%) | 2,797 (47.10%) |
| 40−60 | 2,101 (41.37%) | 537 (10.57%) | 55 (1.08%) | 2,489 (41.92%) | 618 (10.41%) | 137 (2.31%) |
| 60−80 | 948 (18.67%) | 22 (0.43%) | 0 (0.00%) | 1,265 (21.30%) | 18 (0.30%) | 28 (0.47%) |
| 80−100 | 181 (3.56%) | 2 (0.04%) | 0 (0.00%) | 395 (6.65%) | 2 (0.03%) | 0 (0.00%) |
| 100−150 | 199 (3.92%) | 1 (0.02%) | 0 (0.00%) | 241 (4.06%) | 2 (0.03%) | 0 (0.00%) |
| >150 | 8 (0.16%) | 0 (0.00%) | 0 (0.00%) | 8 (0.13%) | 0 (0.00%) | 0 (0.00%) |
| **Total** | 5,079 (100%) | | | 5,938 (100%) | | |

TABLE 4.4: Amplification factor for a single query with EDNS0 buffer size 8,192

#### 4.3.1.3 Examing RRL mechanism for positive responses

In the context of this work, we are mostly concerned about the following parameters of RRL mechanism: *window*, *responses-per-second*, *nxdomain-per-second* and *slip*

[155, 167]. The *window* parameter with a default value of 15 secs designates the time period over which rate limiting is calculated and during which excesses are stored. The *responses-per-second* parameter determines the maximum number of times that the same response will be given to the same requestor (default 5). The *nxdomain-per-second* defines the maximum number of times that the same negative response will be given to the same requestor independently of the queried domain name (default 5). Finally, *slip* specifies the rate that successive identical requests will be answered with a truncated response. The default value for this last parameter is 2, meaning that every second response is truncated.

For investigating the adoption of RRL mechanism [155], we dispatched a stream of 10,000 DNS queries from the same source IP address within a time window of 70-80 secs. Next, we calculated the success ratio of this effort expressed as a percentage of the returned answers versus the total submitted queries (in our case 10,000). We conducted this experiment twice in different days and time zones to deduce possible variations due to workload and congestion on the ANS's side. Special care has been taken to minimize the burden imposed by our experiments on the examined ANSs. Hence, in order to minimize the window of the request bursts to minimum possible, we do not query consecutively the same authoritative for a different domain name under its administration. For the same reason, we examine the root ANSs only once. Naturally, it is expected that their behavior to be similar between different runs. It is also to be noted that some responses bear erroneous response code (REFUSED or SERVFAIL). However, as we are solely interested in the volume of the response not its status, we include them in the final outcomes. In any case, these erroneous responses decrease the overall amplification factor, since their packet size does not exceed 50-60 bytes.

During the execution of this phase, for the first version of root file, we record that more than 64% of the ANSs display a negligible difference of no more than 1% on success ratio of positive responses (3,261 out of 5,079). Furthermore, most of the ANSs (91.8%) demonstrate a variation of no more than 10% (4,665 out of 5,079). This means that they show a reliable behavior as reflectors. Nevertheless, a small percentage (1.6% or 79 out of 5,079) has a deviation higher than 50% between the two runs. We can assume that the performance of the specific ANSs are highly influenced from workload or similar factors.

Regarding the second version of the root.zone file, we observe that almost 62% of the NSs, exhibit a steady performance with a difference not exceeding 1% on success ratio (3,679 out of 5,938). Moreover, the majority (90.1%) shows a variation of no more than 10% (5,347 out of 5,938). As with the first version of the root.zone file, there exists a small portion (2.2% or 128 out of 5,938) that has a great deviation amongst the two runs (50%-99%).

It should be remarked that the most beneficial reflector missed only one response out of 10,000 and its response had a size of 4,093 byte. In other words, the specific authoritative reflects the ingress malicious traffic multiplied by 132.

Tables 4.5 and 4.6 summarize the results for the authoritatives that have a variation less than 10%. In total, the tables include 91.8% and 90.1% of unique pairs <domain name, IPv4 address> extracted correspondingly from the first and second version of the root zone file. Apparently, a determined aggressor would consider only ANSs that reveal a consistent performance on their behavior. Each table displays the success ratio of the ANSs, meaning the number of the received responses versus the total queries, along with the corresponding amplification factor calculated only from the successful queries. This way, one is able to deduce the magnitude of the received response. To ease the reading of the tables, we circle the actual cumulative amplification factor (considering all the volume of the submitted queries) only for the most profitable ANSs.

| | | **Amplification Factor** | | | | **TOTAL** |
|---|---|---|---|---|---|---|
| | | <=20 | 20−40 | 40−60 | >60 | |
| **Success Ratio** | <=70% | 734 (15.77%) | 8 (0.17%) | 91 (1.95%) | 142 (3.05%) | 975(20.95%) |
| | 70%−80% | 3 (0.06%) | 0 (0.00%) | (41) 12 (0.26%) | (49) 22 (0.47%) | 37(0.79%) |
| | 80%−90% | 4 (0.09%) | 0 (0.00%) | (50) 1 (0.02%) | (58) 6 (0.13%) | 11(0.24%) |
| | 90%−100% | 962 (20.67%) | (37) 231 (4.96%) | (51) 1,798 (38.63%) | (73) 641(13.77%) | 3,632(78.02%) |
| | **TOTAL** | 1,703 (36.58%) | 239 (5.13%) | 1,902 (40.86%) | 811(17.42%) | 4,655 (100.00%) |

TABLE 4.5: Percentage of authoritative NSs for positive responses (ver. 1)

| | | **Amplification Factor** | | | | **TOTAL** |
|---|---|---|---|---|---|---|
| | | <=20 | 20−40 | 40−60 | >60 | |
| **Success Ratio** | <=70% | 676 (12.64%) | 11 (0.21%) | 91 (1.70%) | 151 (2.82%) | 929(17.37%) |
| | 70%−80% | 3 (0.06%) | 1 (0.02%) | (41) 14 (0.26%) | (53) 43 (0.80%) | 61(1.14%) |
| | 80%−90% | 3 (0.06%) | 0 (0.00%) | (50) 1 (0.02%) | (58) 6 (0.11%) | 10(0.19%) |
| | 90%−100% | 898 (16.79%) | (38) 348 (6.51%) | (50) 2,074 (38.79%) | (76) 1,027(19.21%) | 4,374(81.30%) |
| | **TOTAL** | 1,580 (29.55%) | 360 (6.73%) | 2,180 (40.77%) | 1,227(22.95%) | 5,347 (100.00%) |

TABLE 4.6: Percentage of authoritative NSs for positive responses (ver. 2)

Next, we examine how the endorsement of TC contributes to the reduction of the overall amplification factor. We record only those authoritatives that feature a considerable success ratio (higher than 60%), since the remaining by default demonstrate a moderate amplification factor. The corresponding outcomes are summarized in Table 4.7. As observed, the results are ordered by the percentage of truncation on the successful responses. The column labelled as "Number of zones" indicates the total of distinct domain zones that reply with the corresponding rate of TC responses.

| | Ver. 1 | | Ver. 2 | |
|---|---|---|---|---|
| **Truncated Responses** | **Number of zones** | **AVG Size (Bytes)** | **Number of zones** | **AVG Size (Bytes)** |
| 0 | 3,356 (81.85%) | 1,628.54 | 4,014 (80.68%) | 1,782.23 |
| 0−10 | 141 (3.44%) | 2,076.40 | 197 (3.96%) | 2,265.79 |
| 10−20 | 166 (4.05%) | 2,009.55 | 216 (4.34%) | 2,158.03 |
| 20−99 | 25 (0.61%) | 214.64 | 29 (0.58%) | 278.15 |
| 99−100 | 412 (10.05%) | 133.73 | 519 (10.43%) | 354.43 |
| **Total** | 4,100 (100.00%) | | 4,975 (100.00%) | |

TABLE 4.7: Percentage of truncated positive responses

#### 4.3.1.4 Examing RRL mechanism for negative responses

An insidious attacker may assume that they could be able to evade the RRL mechanism by inquiring continuously random domain names. Since the corresponding response will probably contain different records of NSEC/NSEC3, the threshold of RRL mechanism will not be triggered so often as in the case of positive responses, where the contents of the response are exactly the same for all the queries.

For inspecting the validity of such an assumption, similarly to the previous phase, we unleash a burst of 10,000 queries each time with a random string as the second label of the queried domain name each time. Then, we calculate again the success ratio of negative responses and the degree of compliance with TC mechanism.

As illustrated in Table 4.8, for the first version of root.zone file we observe that nearly 78% of ANSs demonstrate a difference of no more than 1% on success ratio of negative responses (3,960 out of 5,079). Additionally, the majority of them (94.8%) exhibit a variation that does not exceed 10% (4,817 out of 5,079), while only 0.4% (22 out of 5,079) present a high difference among the two runs.

Regarding the second version, we observe from Table 4.9 that about 76% of ANSs display a negligible difference (4,498 out of 5,938), while 96.3% has a rather expected variation

of 10% (5,718 out of 5,938). Merely 0.9% or 57 out of 5,938 show a significant difference that exceeds 50%.

Tables 4.8 and 4.9 summarize the results for the authoritatives that have a variation less than 10% during the two runs of the experiment. Likewise, 94.8% and 96.3% of unique pairs <domain name, IPv4 address> extracted from the two examined version of the root zone file are included respectively. Finally, Table 4.10 details on the degree of compliance with TC for the case of negative responses and only for those ANSs with success ratio greater than 60%.

| | | Amplification Factor | | | TOTAL |
|---|---|---|---|---|---|
| | | <=20 | 20−40 | 40−60 | |
| Success Ratio | <=70% | 868 (18.02%) | 192 (3.99%) | 3 (0.06%) | 1,063(22.07%) |
| | 70%−80% | 26 (0.54%) | 0 (0.00%) | 0 (0.00%) | 26(0.54%) |
| | 80%−90% | 446 (9.26%) | 47 (0.98%) | (34) 4 (0.08%) | 497(10.32%) |
| | 90%−100% | 1,378 (28.61%) | (27) 1,836 (38.12%) | (46) 17 (0.35%) | 3,231(67.07%) |
| | **TOTAL** | 2,718 (56.43%) | 2,075 (43.08%) | 24 (0.50%) | 4,817 (100.00%) |

TABLE 4.8: Percentage of authoritative NSs of negative responses (ver. 1)

| | | Amplification Factor | | | | TOTAL |
|---|---|---|---|---|---|---|
| | | <=20 | 20−40 | 40−60 | >60 | |
| Success Ratio | <=70% | 924 (16.16%) | 277 (4.84%) | 3 (0.05%) | 0 (0.00%) | 1,204 (21.06%) |
| | 70%−80% | 30 (0.52%) | 3 (0.05%) | 0 (0.00%) | 0 (0.00%) | 33 (0.58%) |
| | 80%−90% | 495 (8.66%) | 235 (4.11%) | (35) 100 (1.75%) | 0 (0.00%) | 830 (14.52%) |
| | 90%−100% | 1,486 (25.99%) | (27) 2,106 (36.83%) | (46) 37 (0.65%) | (69) 22 (0.38%) | 3,651 (63.85%) |
| | **TOTAL** | 2,935 (51.33%) | 2,621 (45.84%) | 140 (2.45%) | 22 (0.38%) | 5,717 (100.00%) |

TABLE 4.9: Percentage of authoritative NSs of negative responses (ver. 2)

| | Ver. 1 | | Ver. 2 | |
|---|---|---|---|---|
| **Truncated Responses** | **Number of zones** | **AVG Size (Bytes)** | **Number of zones** | **AVG Size (Bytes)** |
| 0 | 3,403 (80.83%) | 816.04 | 4034 (79.27%) | 911.90 |
| 0−10 | 10 (0.24%) | 534.70 | 22 (0.43%) | 1,022.50 |
| 10−20 | 317 (7.53%) | 607.39 | 434 (8.53%) | 794.30 |
| 20−30 | 88 (2.09%) | 462.92 | 129 (2.53%) | 516.65 |
| 30−40 | 24 (0.57%) | 321.27 | 42 (0.83%) | 761.78 |
| 40−50 | 0 (0.00%) | – | 20 (0.39%) | 1,425.00 |
| 50−70 | 0 (0.00%) | – | 14 (0.28%) | 1,419.29 |
| 70−99 | 27 (0.64%) | 128.65 | 41 (0.81%) | 478.92 |
| 99−100 | 341 (8.10%) | 40.15 | 353 (6.94%) | 40.15 |
| **Total** | 4,210 (100.00%) | | 5,089 (100.00%) | |

TABLE 4.10: Percentage of truncated negative responses

### 4.3.2 Discussion

Even from a cursory look at Table 4.4, we can assert that between the examined query types ANY is more profitable for a potential aggressor. Also, as given in table 4.4,

more than 25% of the ANSs provide an amplification factor of higher than 60, which is considered fatal. Furthermore, the second version of the root file exhibits worse percentages from a defender's viewpoint, meaning that the new introduced authoritatives provide hefty DNS responses. Also, an interesting discovery is that only 5 out of the 13 permanent root NSs, as well as 43 out of 2051 (from the 1st ver. of the root.zone file) and 58 out of 2,743 (from the 2nd ver.) remaining ANSs support truncated responses. That is, considering both versions of the root.zone file, the TC bit was enabled in only 387 of 5,079 and 464 of 5,938 responses respectively, when we issue a single request.

As expected, the accomplished amplification factor is preferable when requesting positive responses rather negative. Nevertheless, the success ratio due to RRL mechanism performs almost akin in both cases, which in turn implies that the relevant parameters (i.e., *responses-per-second* and *nxdomain-per-second*) have similar values.

Overall, the results strongly support the view that ANSs of TLDs are attractive to attackers for getting them involved in DNS amplification incidents. Depending on the intention of the perpetrator and the resources available on his arsenal, he could take advantage of the particular ANSs that best suit his purposes. In the case he already possesses a pool of reflectors, such as open resolvers or open forwarders [18], he may use only those authoritatives that provide high amplification factor. Since usually the resolvers and the forwarders afford cache capabilities as well, the responsive ANSs will not perceive that they are engaged in such an incident. To exemplify this, table 4.4 shows the existence of a considerable portion (over 25%) of DNS queries towards TLDs that contribute an amplification factor higher than 60. On the other hand, if the attacker lacks reflectors, he may employ those ANSs that exhibit a high success ratio. There is a substantial fraction of ANSs that respond to nearly all the queries and induce an amplification of medium to high magnitude. Specifically, we realized that 1,197 distinctive authoritatives out of the total 2,755 serving TLDs (43%) for the most recent version of root.zone file reflect the inbound network traffic by magnifying it by a factor that exceeds 50. In any case, those ANSs are alluring for launching Slow-rate attacks.

The deployment of TC countermeasure [165] significantly reduces the overall amplification factor. We can observe that the average message size decreases inversely with the percentage of TC. Essentially, it diminishes the amplification factor to nearly 10,

when almost every response is truncated. Hence, this mechanism prevents ANSs unconsciously participation in attacks, even though it preserves a high success response ratio. As a result, it is desirable to endorse TC rather to drop suspicious incoming queries. This is because the latter countermeasure may hamper legitimate clients from receiving an answer, and thus force them into increasing their DNS traffic.

Compared with [141, 148], our methodology does not deploy zone walking nor requires collaboration with the zone's administrator for extracting domain zones and their matching authoritatives. In contrary, we take advantage of publicly available data (root zone file) and concentrate our research on TLDs instead of 2-TLDs. Furthermore, besides their role as amplifier, we investigate the reflection capabilities of the examined ANSs.

It is also to be noted here that although there were some scarce and undocumented concerns that the gradual adoption of DNSSEC (due to its increased RR size [52]) would facilitate aspiring aggressors to mount improved DNS amplification attacks, to authors' best knowledge this is the first study that involves DNSSEC-related RR in a (D)DoS attack. So, although DNSSEC is used among others to drastically confine, if not eliminate, the well-known cache poisoning vulnerability [47, 168], in the course of the current research it will become evident that it may be used as a vehicle for launching large-scale DDoS assaults. The authors in [169] observed that the increase of ANY type DNS queries derived from the increase of DNS amplification incidents.

## 4.4 Countermeasures

Actually, even the most advanced DNS amplification attacks are not impossible to preclude. The efforts of the defenders must focus on diminishing the ammunition available in the arsenal of a DNS amplification attacker, and especially on eliminating misconfigured or unprotected infrastructure that facilitate the reflection of malicious network traffic. Note, that the adaptation of these type of remediations does not prevent the infrastructure itself to get targeted, but rather to get involved to an incident. Thus, it contributes to the global protection of Internet. Section 4.4.1 elaborates on proactive measures. Furthermore, mechanisms that aim to detect and suppress DNS amplification attacks at their very beginning (reactive methods), should be deployed on the side of alluring targets. Section 4.4.2 details on such proposals.

### 4.4.1 Proactive Measures

#### 4.4.1.1 Lowering the Amplification Factor

From the discussion in the previous sections it becomes clear that the efficiency of DNS amplification attack lies on the fact that a small request packet sent from the attacker can generate a large toward the victim. Thus, the defenders should endeavor to eliminate any potential source that can be exploited as amplifier by the aggressors.

- *ANY query*: From the available results in section 4.4, one can easily deduce that a DNS query of ANY query type for DNSSEC-related RR, represents a fruitful type of request for magnifying the effects of a DNS amplification attack. Consequently, it is strongly advised for the defenders to put effort to restrict or totally cease the support of this type of query by the NSs.

  Although there are two certain softwares that rely on ANY query (qmail and firefox) [170], its functionality can be substituted by other query types. In fact, there is a trend among the naming software vendors to deprecate ANY query and instead respond with NOTIMP (RCODE 4, Not Implemented) status in such queries [170]. So, a debate about the necessity of ANY query and the possible ways the remaining relying software could be patched to avoid its usage would be very useful. While there exist other types of records that provide hefty responses, their magnitude is notably smaller compared to those provided by ANY.

- *Truncated Responses*: Another remediation is the restriction of the response's size provided by NSs to an upper limit. According to RFC 5966 [165] and its substitute RFC 7766 [171], DNS responses containing multiple RRs and thus, having hefty packet size, should be truncated and retried in TCP mode. In this case, the (truncated) UDP response should have a minimal packet size and its TC flag should be on. Therefore, whenever a requestor desires a DNS answer with numerous RRs, it should submit its query in TCP mode in order to verify their IP address. From the results contained in Table 4.7 the reader can clearly notice that although the success ratio of the responses is considerable, the overall amplification factor remains low due to the high percentage of truncated responses. In other words, the truncation of sizable responses will hinder the exploitation of NSs as amplifier.

- *Response Rate Limiting*: DNS RRL is a mechanism that limits the identical responses that can be returned to the same requestor within a certain time interval by a NS. Essentially, RRL forces a NS to respond with truncated responses or drop them at all, whenever the ratio of the identical request from the same IP or network address exceeds a predefined threshold within a time window. RRL is applicable to ANSs only, as the normal DNS flow towards an authoritative is expected to contain limited duplicate queries from the same recursive resolver. As discussed in section 2.1.7.2, this is due to the caching facility every recursive affords, allowing it to locally store RRs for fulfilling subsequent requests. As it is clearly shown in Tables 4.5 and 4.6, whenever an ANS properly applies RRL, it discourages the potential attacker to entangle this machine into DNS amplification attack. Note that DNS RRL is already implemented in BIND 9.

#### 4.4.1.2 Eliminate Reflection Capabilities

Recall from section 4.1 that for achieving the reflection capabilities of DNS amplification attack, the aggressor performs source IP address spoofing and employs open recursives resolvers, i.e., DNS entities that respond to DNS recursive queries originating from open Internet. This way, they are able to involve unconscious third servers into their attack and to conceal the traces of their actions. Following, we firstly introduce mechanisms that aim to prevent IP spoofing, and then we present initiatives that address the issue of open recursives.

- *Source validation*: Since this type of attack requires the spoofing of the requestor's source IP address, any IP address validation would block malicious packets. Of course, it is not possible for every firewall or router to examine the source IP of all UDP packets passing through it. Though, the devices located at the borders of a network should inspect and allow a packet to pass through only if it has a source IP address assigned from an internal subnetwork. This guideline is explicitly outlined in RFC 2827 [172]. The specific measurement is also known as BCP 38. Considering that almost 25% of the ASs [173, 174] permit spoofing, the adoption of source validation from their side is compulsory.

  Another solution could be the usage of sessions for the case of UDP protocol [140]. Videlicet, session information should be included in UDP packet for each

request corresponding to a large response. Since UDP is the most vulnerable transport protocol for IP spoofing the aforementioned restriction will force the attacker to open multiple sessions to each open resolver, and thus lessen the surge of attacking requests. A survey of research publications about methods to prevent source address IP spoofing by utilizing session tokens is given in [140].

- *Disable open recursion:* Any DNS recursive should only accept DNS queries from clients residing inside its network. However, there exist a vast number of devices that operate as open resolvers and accommodate request originating from open Internet. As given in section 4.1.1 initiatives, including Open Resolver Project [150] and Measurement Factory [151], intend to discover such type of machines and with the collaboration of the network administrators to cease their operation. Yet, even in the case of restricted to internal users resolvers, as it is observed from the results included in section 4.2, an attacker may be able to evade such a restriction with the exploitation of open DNS forwarders. For this reason, the network administrators must also disable DNS forwarding to all network devices. Whenever, the installation of a forwarder is a requisite, its service should be restricted to solely trusted or internal users.

### 4.4.2 Reactive methods

Regardless of the proactive measures, the responsible network administrators should adopt solutions, which aim to detect and suppress DNS amplification attacks at their very beginning. In this section, we shortly present the most important ones of them.

Kambourakis et al., [175, 176] were the first to propose a mechanism that could be integrated into the functionality of a DNS NS. This solution capitalizes on the matching of DNS requests and corresponding responses of the NS. Therefore, any response reaching the server that does not correspond to a request, i.e., is not solicited by the server, is inevitably characterized as suspicious. When the ratio of the unsolicited responses exceeds a predefined threshold, then an alert is generated and banning rules are automatically set/updated in the firewall in order to block traffic stemming from the attacking nodes. Di Paola and Lombardo [177] extended the aforementioned works by incorporating bloom filters in an effort to speed up the process of detection. Further,

bloom filters have been also recruited by Sun et al., [178] to deal with DNS amplification. Nevertheless, this time the proposed solution was based on hardware, aiming for efficiency.

Other proposals apply machine learning techniques for identifying DNS amplification attacks. For instance, Rastegari et al., in [179, 180] proposed an Intrusion Detection System (IDS) capable of detecting DNS amplification with the help of Neural Networks (NN) and Support Vector Machines (SVM). Similarly, Ni et al., in [181] employed characteristics of attack traffic time series to produce SVM classifier for the detection of attacks.

Furthermore, Deshpande et al., [182] introduced a probabilistic model based on Continuous Time Markov Chain model to conduct a cost-benefit analysis for DNS amplification countermeasures. In their work the three countermeasures under consideration were: 1) filtering and blocking attack sources, 2) random drops of DNS (UDP) packets as described in [183] with the purpose to regulate incoming traffic, and 3) aggressive retries from the clients for increasing the legitimate traffic [184]. According to the authors, this probabilistic model was able to deduce significant reductions in DNS amplification attack probabilities when all the three aforementioned countermeasures are deployed. Also, their model indicated that the usage of DNSSEC is more vulnerable than that of DNS, and thus, DNSSEC gains noticeable fewer benefits from the proposed countermeasures.

A different approach has been followed by Fachkha et al., in [185, 169], where the authors were detecting DNS amplification attacks by monitoring DNS traffic heading to a darknet space. Specifically, the authors observed that amplification attackers trend to scan the Internet for open resolvers. So, they analyzed benign DNS requests arriving to unallocated IP addresses. This technique differs from the prevalent proposal [186] that depended on backscattered network traffic.

MacFarland et al.,[154] proposed to outsource the service of DNS ANSs and to forward the local DNS resolution requests to an off-site resolver during the time an organization is under a DNS amplification attack. Thereby, the organization would be capable of filtering the overall ingress DNS traffic. Even though such type of filtering would drop both malicious and benign traffic, the organization's normal operation would not be affected due to outsourcing. However, the link between the internal DNS forwarder and

the remote DNS resolver should be tunnelled, via IPSec or TLS, in order to evade the restrictions on the frontier firewall, i.e., to bypass typical DNS communication channels. The authors estimated that the overall latency introduced for establishing the tunnel, starting the remote resolver and reboot the local resolver to operate as forwarder does not exceed 1.36 secs in average.

In the nutshell, based on the results gathered from the experiments described in section 5.8.3, we can easily deduce that a significant number of the inspected network devices do not support any of the aforementioned countermeasures. In the aftermath of the results obtained, we can safely argue that poor practices and omissions from the side of network administrator and ISP companies may put the Internet at risk. Putting it another way, usually, for reason of cutting down cost and work time, network providers decide not to conform with security advisories which would greatly hinder the feasibility of amplification attacks.

# Chapter 5

# DNS-driven botnet C&C architectures

Recall that one of the purposes of this thesis is to highlight the manipulation of DNS by botnets. In this chapter, we elaborate on the botnet issue and provide evidences of their actions. In this direction, we propose three novel facets of mobile botnet that utilize DNS for building their C&C channel. At the end, we conduct a comprehensive review of DNS-based countermeasures against botnets.

## 5.1 Introduction

Undoubtedly, botnets pose a growing threat to the Internet, with DDoS attacks of any kind carried out by botnets to be on the rise. Nowadays, botmasters rely on advanced, hidden *Command & Control* (C&C) channels to achieve their goals and most importantly to remain undetected. A *botnet* [187] is considered as a network consisting of infected and compromised devices, called bots, zombies or slaves, which are remotely controlled by an attacker, known as *botmaster* or *botherder*. A basic architecture of a botnet is given in Fig. 5.1 [187]. As seen from the figure, a bot agent obeys every command received by its botmaster ordering it to initiate, alter the parameters or terminate an attack. Botnets pose a serious threat to Internet, since they are capable of disrupting the normal operation of services, networks and systems at will of their botmaster. For instance, botnets could be used for launching DDoS attacks [188], sending spam emails

on a massive scale [189], identity theft [190], distributing malware or even copyrighted material [191], just to name a few. Generally, botnets are considered as a relatively inexpensive and easy way to conduct illegal activities in the Internet, while their botherders gain monetary profit by leasing the botnet to potential perpetrators for accomplishing their criminal activities [192].

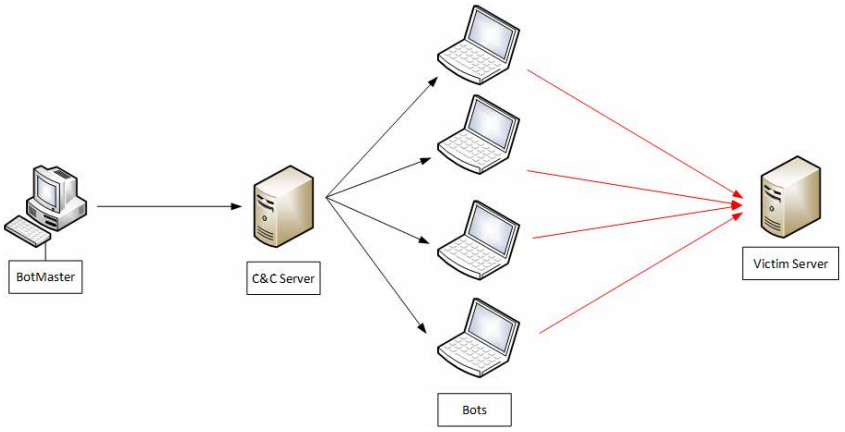



FIGURE 5.1: Botnet Structure

Usually, a device is turned into a bot client, by malware infection, for instance by a malicious software or by accessing an infected website [193]. After that, the device joins to a network of bots waiting for commands. The bot takes action only whenever the botherder says so, through a (covert) C&C channel, while the remaining time stays silent. In this respect, C&C enables a bot to acquire new instructions and malicious capabilities, as injected by a remote ill-motivated entity. In the literature, a variety of C&C topologies have been explored by botmasters with the dual aim to curtail network breakdowns and system failures, and to cope with deployed defences, hijacking attempts, and legal shutdowns.

In this chapter, we firstly describe the basic architectures of botnets and elaborate on the use of DNS to construct C&C channels. Next, based on the research done in the context of this thesis, we detail on more advanced architectures with particular focus on mobile botnets, and practically evaluate them in terms of different kind of attacks.

## 5.2 Botnet Architectures

Depending on how the bots are remotely controlled by their herder, i.e., how the C&C channel is structured, one is able to classify them into centralized, decentralized or hybrid architectures. Indeed, the C&C channel constitutes the most critical part of a botnet. For the attacker is the way they coordinate the bots and disseminate their commands, while for the defender represents the point which once it is detected the whole botnet can be eliminated, hijacked or neutralized.

The *centralized* architecture (as that of Fig. 5.1) is based on the client server model, where all bots are directly connected with one, or few, C&C servers. It is also known as star topology [194]. The C&C servers undertake to coordinate the bots and instruct them to take action. Although a centralized botnet exhibits optimum coordination and rapid dissemination of the commands, it also poses a single point of failure. From the moment the C&C server is detected and deactivated the entire botnet is turned off.

To overcome the aforementioned weaknesses and to evade detection, a *decentralized* architecture may be selected to carry out the C&C operation. In this approach, no central C&C server exists, but rather the various bots communicate with each other via P2P protocols [195]. In other words, the bots behave as C&C server and client at the same time. Therefore, if any of the bots is tracked down and deactivated, there are minimal implications for the robustness of the entire network [196].

The *hybrid* architecture on the other hand combines the advantages of both centralized and decentralized ones. That is, in this setting, the bot agents exhibit diverse functionalities. Some of them, temporarily undertake the C&C server role, with the aim to coordinate the botnet and disseminate the instructions, while the remaining connect to those C&C servers for receiving instructions before springing to action [197].

Moreover, the *hierarchical* architecture [194] allows bots to forward botmaster's instructions to their descendants, which they have been previously infected. On the downside, this topology suffers from network latency issues making it unsuitable for real-time activities. Also, any bot client is unaware of the C&C location.

The work in [198] presents a rather theoretical "random" botnet infrastructure in which the botherder, or any other member of the botnet possesses no information for the other

members of the same botnet. This model is completely contrary to the centralized model, where the botnet operator knows beforehand all the members of the botnet. In a *random topology*, whenever the botmaster wishes to command the bots, has to randomly scan the Internet to locate them. This model has the benefit that the tracking of a single bot will not reveal any information about the botnet or the botherder. Of course, the applicability of such a random structure is questionable, as it imposes limited coordination whenever an instruction needs to be delivered to the bots.

## 5.3 Life Cycle of a Bot

A device upon infection follows specific steps (phases) in order to turn into an active member of the botnet. In the literature, the various phases are described with different names, but in general they present similar functionality. In the current study, we will describe the botnet's life cycle as described in [187, 199] and depicted in Fig. 5.2 [187].

Phase one is the initial injection, where a host is infected into becoming a bot. As already pointed out, the infection can caused by typical malware infection, for example by malicious software or by accessing an infected website. The secondary infection phase updates the device with bot capabilities and the means to contact with botnet's C&C channel. So then, the bot can connect to C&C server to receive instructions and updates. This procedure is also called as *rallying*. Actually, the third phase is repeated regularly to ensure that the bot is an active member of the botnet. It is to noted that since in that phase the bots contact directly to C&C servers, they create distinguished traffic patterns which can be identified by an IDS. Afterwards, the bot is ready to execute malicious actions as instructed by its botmaster. During this phase, several messages may be transmitted amongst botmaster and bots over a short period of time. However, C&C traffic exhibits low volume and is hardly identified by defence mechanisms. The last phase is the maintenance and update of the bot client, so that the bot keeps up with new capabilities or ways to evade detection techniques.

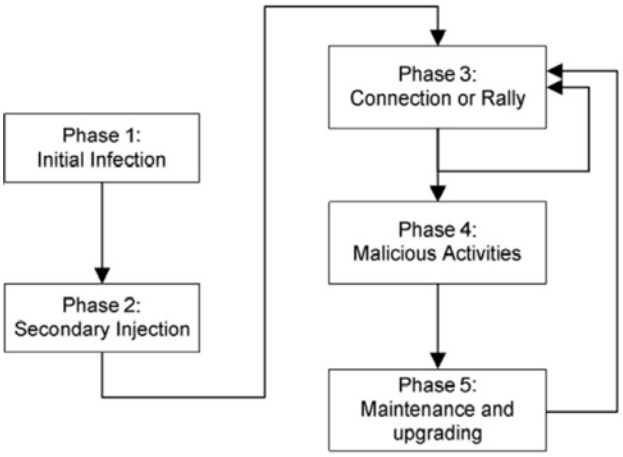



FIGURE 5.2: The life cycle of a Bot

## 5.4 DNS Fluxing

Perhaps the most vital demand for maintaining control of the entire botnet is the ability for a bot to constantly stay in touch with C&C infrastructure. This requirement is especially true for botnets that hinge on centralized C&C. That is, a bot will not be able to receive new instructions if the C&C cannot be located, and as a result, will continue to probe the vanished C&C in vain. Upon infection, the bot client should know a way to contact with its botmaster. Simply embedding in the binary of the bot's malware a list of static IP addresses corresponding to the C&C servers will jeopardize the botnet's robustness. Namely, a reverse engineering of the binary would reveal those IP addresses and the defenders could shutdown the botnet outright [187].

To cope with this issue, botmasters employ a number of advanced techniques to not only minimize the probability of bots losing contact with their C&C infrastructure, but also to render their botnet more agile to hijacking and stoppage attempts. *"Fluxing"* seems to be the preferred technology to deal with the aforementioned issues, i.e., uninterrupted provision of C&C location resolution and failover resilience. Currently, *IP Flux* and *Domain Flux* are the two dominant ways of "Fluxing" [200]. The first one involves the regular altering of IP address pertaining to a particular FQDN. This potential is particularly fruitful for botnet operators because it enables them to associate multiple IP addresses with a specific host name and change the linked addresses at a rapid pace (also well-known as "fast-flux")[6, 201]. Two types of fast-flux are utilized. In *Single-flux*, the botmaster correlates numerous, hundreds or thousands, IP addresses with one

domain name. This is accomplished with the exploitation of a RR of A type, which has minimal TTL value and is registered and de-registered in a round-robin fashion [194]. On the other hand, *Double-flux*, apart from fluxing the IP addresses of the domain name, it fluxes also the IP addresses of the corresponding ANS, i.e., the IP contained in the NS type RR [194].

Domain flux on the other hand is essentially the opposite of IP flux, enabling botmasters to continuously alter and associate multiple FQDNs to a single IP address or C&C infrastructure. An easy way to accomplish domain fluxing is by the utilization of wildcard DNS records [194]. Recall from section 2.1.11.5 that wildcard records have an asterisk ('*') as the leftmost label and resolve any domain name with the same remaining labels.

Lately, domain fluxing is achieved with the help of a technique known as Domain Generation Algorithm (DGA) [202]. Given a random seed, DGA produces a number of unique pseudorandom domain names based on cryptographical operations (one-way hash function). However, only a small fraction of the generated domain names may resolve to the IP address of the C&C server. On its initial implementations, the algorithm received as input the current epoch. Torpig [203] and Conficker [204] for instance followed that approach. The work in [205] proposes a more advanced method of automatic text generation that produces genuine-looking domain names. Furthermore, a DGA algorithm could possibly take as input a secret key which is obfuscated in the malware's binary, so as the bot client to be able to authenticate itself with the C&C server. In addition, the generated domain names are dynamically calculated in volume and are valid for a limited time period. Thus, the defenders' attempts focuses on the timely prediction of the random domain names in an effort to confront them. In the case the defenders accomplish a successful prediction in time, the bots will be unable to connect to the C&C infrastructure and receive orders.

So, the bot client could contain in their binary either a list of domain names for the case they adopt IP fluxing or the DGA accompanied with the seed for the case of DNS fluxing. In any case, the use of DNS is mandatory. Therefore, as detailed in section 5.9.1 the majority of the countering solutions focus on the analysis of DNS traffic for the sake of locating botnets and hijack the C&C channels.

## 5.5 C&C channels

The bots are connected to C&C channels waiting for the botmaster's instructions. As a rule of thumb, these communication channels are based on HTTP or IRC protocol. In the first case, the communication is disguised inside the normal Web network traffic, as the usage of Web is allowed in most networks, including corporate ones [206]. On the other hand, in IRC-based architecture, the bots join to IRC channels and await commands. Considering that IRC allows multicast communication to specific groups of clients or private unicast communication amongst two parties, the botherder can customize the coordination of their bots, i.e., they are able to choose specific bots for accomplishing a malicious action [187]. In any case, the messages on the IRC channel are in an obfuscated custom dialect, e.g., encrypted or hashed to avoid disclosure [207]. Mainly, IRC and HTTP based C&C channels are utilized in centralized architectures.

## 5.6 DNS as C&C Channel

Apart from HTTP and IRC, DNS has been also exploited as a carrier protocol for establishing a C&C channel. With the help of a technique called *DNS tunneling*, the botnet entities transfer data embedded within the RRs of a DNS packet, either query or response [208]. Usually, the first labels of the domain name in a query is utilized by bots for transmitting data to the botmaster, while TXT RRs in responses contain the botmaster's commands [209]. The key benefit for employing DNS is that it is one of the few protocols that is very rarely filtered out by firewalls [210].

Dietrich et al., [209] were the first to investigate botnets that take advantage of DNS tunneling for C&C purposes. The authors analysed by reverse engineering techniques Feederbot, a botnet malware, which exfiltrate data within sub-domain labels of DNS query and disseminated the attack parameters in DNS responses. Actually, Feederbot created an end-to-end communication channel among bots and botherder in the form of DNS messages. Furthermore, they discovered that DNS queries were headed directly to DNS recursive resolvers under the control of the botherder, thus circumventing the DNS hierarchy. Thereby, the botherder was in position to utilize any domain they desired, even unresolvable or unregistered. In some other cases, the bot's malware modifies the host's DNS settings, so all DNS traffic is directed to a malicious DNS recursive

[36]. This way, besides the covert channel, the botherder accomplishes to intercept and possibly manipulate all the requests to Internet resources. Additional, Binsalleeh et al., [211] classified the patterns of DNS traffic produced by bot malware inspection, and described how these traces can be utilized for optimum and subtle communication amongst the bots and botherder.

An extensive investigation about the practicability of DNS protocol as a C&C channel was conducted by Xu et al., [205]. The authors identified two modes of C&C communication depending on the messages exchanged between the bots and botherder. The first one, called *codeword communication*, permits one-way communication from botmaster to bot, i.e., it is appropriate for propagating commands for attacks or other similar activities. The client queries about the predefined domain name and the response contains the botmaster's commands. This type of mode only involves the creation of the proper DNS response, which could be easily implemented through a free dynamic DNS service.

The second mode called *tunneled communication* allows for the transmission of data in both directions, which facilitates the collection of stolen data. In fact, the client embeds, usually encoded, his message to the queried name. In turn, the botmaster extracts from the query the bot's message and places his instructions in the response. However, the latter needs the installation of ANS under the control of the botherder and the registration of a zone in the DNS hierarchy. All the aforementioned DNS responses have minimal TTL value for avoiding caching. Finally, the authors proposed two query strategies for minimising the forensic signal of such a DNS covert channel. That is, the attacker can follow an exponentially distributed query strategy, meaning that the bot can issue DNS queries in such a way that their intervals comply with an exponential distribution. Otherwise, the bot client can piggyback its queries, e.g., send a DNS query when legitimate DNS queries are made by the legitimate applications.

A recent real example of botnet with DNS C&C capabilities is *Morto worm*. As Symantec reported [212], this worm try to resolve a DNS TXT record. The textual description returned contains the encrypted IP address where the bot could find and download a binary executable.

## 5.7 Mobile botnets

Over the last decade, mobile devices quickly evolved from pure telecommunication devices to small and ubiquitous computing platforms. Nowadays, such devices are equipped with enough capabilities to even replace the usage of laptops. Therefore, it comes at no surprise that these devices attract the attention of resourceful attackers. Whereas the first appearance of malware that spread via mobile devices has been reported back in 2004 with the emergence of *Cabir* worm for Symbian OS, the initial materialization of malware with C&C capabilities happened some years later [22].

In 2009, *Symbian/Yxes* worm was detected to infect mobile devices with Sybian OS 9. This worm has the functionality to stealthy communicate via device's network interface with a remote server in order to send an HTTP request. Although this malware does not create a real C&C channel, it demonstrates that with few modifications to its functionalities it can easily construct one [213]. In fact, later that year (Nov. 2009), the first ever mobile botnet appeared, namely *iKee.B* bot client. This malware targeted jailbroken iPhones with the aim of turning them to bots. When the binary of iKee.B is installed on the infected iPhone, among other malicious functionalities, it periodically sends a HTTP GET request to the C&C server (based on a centralized architecture) with the aim of receiving new scripts to update its binary. The corresponding malware was released in Europe, while the C&C server was located in Lithuania [214]. For the case of Android platform, *Geinimi* malware emerged in Dec. of 2010, presented similar to iKee.B C&C capabilities based on HTTP protocol. This malware actually receives commands instructing it to steal private data from the mobile device [215]. A more advanced way for botnet coordination is exhibited by *AnserverBot*, where the bot agents acquire the commands from encrypted contents posted in blogs [216].

Excluding legacy network interfaces for connecting to the Internet (Wi-Fi, 4G), in a mobile botnet, the bots are able to communicate via Bluetooth, SMS or MMS messages with their master or with each other [217]. Singh et al. [218] evaluated the Bluetooth technology as a means for establishing a C&C channel. They proposed a scenario in which the herder sends the commands to devices with the "highest" Bluetooth connectivity, that is, to bots that are connected with many infected devices. Afterwards, these bots forward the commands to their own connections. The authors concluded that the

usage of Bluetooth will endorse the spread of the commands and reduce the detectability of the underlying botnet, as the volume of traffic which passes through observable channels (Wi-Fi or 4G data) will be minimum. Moreover, Hua and Sakurai [219] presented a botnet capitalizing on SMS messages to materialize its C&C channel. In this decentralized architecture, the various bots forward the commands to neighboring nodes via texting, thus forming a P2P infrastructure. Expanding this approach, Mulliner and Seifert [220] proposed a hybrid botnet that combines SMS and HTTP protocol as C&C for reducing the billing cost, and therefore avoid to draw the suspicion of the device's owner.

More recently, Xiang et al. [221] proposed *Andbot*, an advanced mobile botnet for Android mobile devices. Andbot uses a centralized C&C topology, that is, the bots connect to specific MicroBlog services. However, the herder utilizes different blog pages of the services, with the URL of the blogs generated by a predefined algorithm, similar to DGA as detailed in section 5.4. Therefore, if a blog is blacklisted, the bots will connect to the next blog page to retrieve their orders. The authors call this technique as "URL Flux" in correspondence with that of Fast Flux (see section 5.4. Similarly, *SoCellBot* proposed by Faghani and Nguyen [222] takes advantage of the Social Networks (SN). Specifically, this type of bot exploits the messaging system of a given SN to infect mobile devices and also builds a covert C&C for the coordination among each bot and the herder. In the same context, Zhao et al., [223] abuse cloud-based messaging services to distribute orders to infected devices. Finally, an interesting approach is presented by Hasan et al., [224] where the authors do not investigate the network capabilities of a mobile device but rather its embedded sensors including optical, audio, vibration and magnetic field ones as a means for creating a covert C&C. For example, a potential botherder could disseminate commands via acoustic or magnetic signals in nearby infected devices.

### 5.7.1 Benefits and Limitations of Mobile Botnet

Given the fact that mobile devices are equipped with powerful capabilities and features, it is corollary to get targeted by potential attackers. As already pointed out in section 5.7, nowadays, mobile devices have networking functionalities, i.e., they can connect to Internet via WiFi connections or via mobile broadband, and hence are able to utilize popular network protocols such HTTP, DNS, etc. Even more, their owners tend to

constantly keep the wireless or data connection turned on in order to stay tuned with their favorite SN or connected to an Instant Messaging (IM) or Voice over IP (VoIP) service. Another advantage of the mobile botnets is that they do not exhibit diurnal behavior as that of the equivalent PC-based botnets [225], since mobile devices rarely get turned off during the night period. Moreover, mobile devices are capable of acquiring new IP addresses very often [226]. Thereupon, traditional defence mechanisms that block requests stemming from blacklisted IP addresses are not normally applicable to this case. Furthermore, a resourceful botmaster could configure the bots to utilize only open Wi-Fi networks in an effort to eliminate the traces of the true perpetrator behind, say, a DDoS incident [227].

On the other hand, there are several issues that complicate the deployment of a mobile botnet and therefore it should carefully be considered by the botnet operator [221]. Firstly, the battery consumption of the mobile device is a critical factor. In the case the power consumption - as the bot agent drains the device's battery - is far more quicker than that of normal usage, then it will notify the end-user that something goes wrong with their device. This situation would probably alert them to deactivate the device and, at least temporarily, disconnect it from the botnet. Secondly, providing that the volume of the data traffic created by the C&C channel or the execution of the commands, exceeds a normal threshold, then the overloaded network connection (or even worst the increased billing cost) will certainly raise the suspicion of the device's owner. Finally, the allocation of internal IP addresses rather than external, it will hamper the creation of C&C channel similar to PC-based botnets.

## 5.8 New facets of mobile botnets

Based on our research, this section introduces new facets of mobile botnets. Our intention is to propose and scrutinize new botnet architectures based on mobile proxies for the sake of protecting the botmaster's true identity. That is, the botmaster's actions are concealed behind the mobile devices that undertake to temporarily play the role of the C&C server.

As already pointed out in section 5.5, the most valuable asset for a botnet operator is to retain its anonymity. So, the idea here is to employ one or multiple mobile proxies in

front of the botmaster. This means that all the bot agents communicate with a proxy rather than the botmaster directly. More importantly, both this proxy's FQDN and IP address change constantly (by exercising both domain and IP flux) with the intent of minimizing the chances of having the botnet detected. On the other hand, using a mobile device as a HTTP proxy to carry out C&C could significantly deplete its energy reserves. So, a betterment is offered, having a separate PC-based botnet as a sidekick to handle proxy operations. Although this may slightly complicate the deployment of the botnet, as the botherder needs to also infect typical PCs, it is estimated to increase the overall stability of the botnet. On top of that, a third powerful setup of the C&C infrastructure is put forward in which all botnet communications exploit DNS protocol as a covert channel. This further contributes in keeping the botnet operation obscured, as all C&C transactions appear to the security systems, say, a firewall or IDS perfectly legitimate in the form of DNS queries and responses. Moreover, this layout infuses simplicity given that no HTTP proxy or other intermediate server is required. All the above mentioned architectures are evaluated through the use of a legacy TCP flooding attack and a more advanced DNS amplification one as detailed in section 4.2.

### 5.8.1 Preliminaries and attack planning

The core idea behind the introduced C&C architectures is to design and evaluate a mobile botnet that will be able to launch DDoS attacks against alluring targets based on the DNS amplification attack scenario given in [18] and detailed in section 4.2. Bear in mind that the latter section presents a new flavor of DNS amplification attack, which is reported to achieve a 44 amplification factor depending on the scenario parameters. Its main advantage is that it does not disclose any illicit or dubious activity during its execution, and thus to our knowledge is the most advanced of its kind so far. Namely, the network traffic during the execution of the attack seems to be perfectly legitimate (excluding the flooding effect of course). Moreover, the attack is very hard to be traced back to the perpetrator who, as a result, enjoys the advantage of anonymity. The scenario exploits the existence of network devices which operate as (open) DNS forwarders and simultaneously utilizes large DNSSEC RR as payload to maximize the overall amplification factor. A very recent work [141] strongly supports this argument, namely DNSSEC-related RR can be exploited to augment the amplification factor of a DNS

amplification attack. On the other hand, as already mentioned in section 5.7, the mushrooming of mobile devices connected to the network and the absence of security measures predicts the spread of mobile botnets.

Furthermore, in this chapter, a second variation of DDoS attack, namely a TCP flooding one is examined taking advantage of the proposed C&C architectures. Its aim is to demonstrate that the proposed architectures can be employed also for any kind of DDoS attack and are not solely dedicated to DNS amplification. In this case, the bots unleash a hefty number of TCP packets towards network ports that are well-known to host popular services, for example WEB, FTP, SSH, etc. Figure 5.3 depicts how the second attack scenario unfolds.

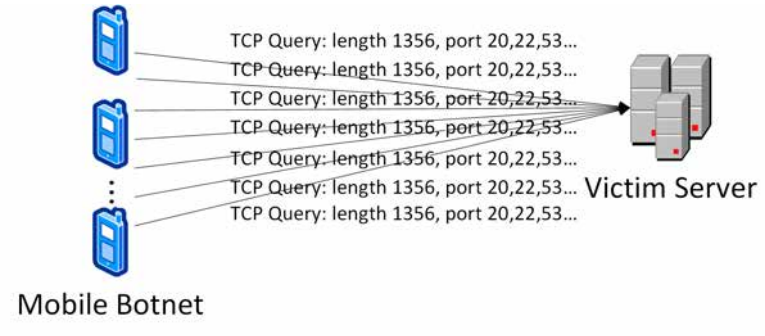



FIGURE 5.3: TCP Flooding Attack

As explained in section 4.1, for the implementation of a DNS amplification attack or any other type of reflection DoS attack, it is crucial for the bot to be capable of conducting IP packet spoofing and in particular spoofing the source IP address of UDP packets. As Google's Android OS does not permit apps to invoke such actions, we develop a dedicated malicious app that has embedded the necessary libraries for executing Scapy tool [162] and other apps and scripts. However, this app needs to gain administrative privileges (root) for being able to install itself on the infected mobile device. Therefore, our intention is to disguise the aforementioned app into a legitimate looking one, such as an anti-malware scanner. This way, the owner of the device will provide the required permissions without knowing the true purpose of the app (given of course that the device is rooted/jailbroken). However, it is to be noticed that the way the bots are infected lies out of scope of this PhD thesis.

In a nutshell, we design three scenarios for the creation of the C&C channel. The first two employ a C&C HTTP-based server, while the third uses DNS protocol as the covert

communication channel.

### 5.8.2 Architecture I: A purely mobile botnet

The first architecture is given in Figs. 5.4 to 5.8. As observed from the figures, it involves the following entities:

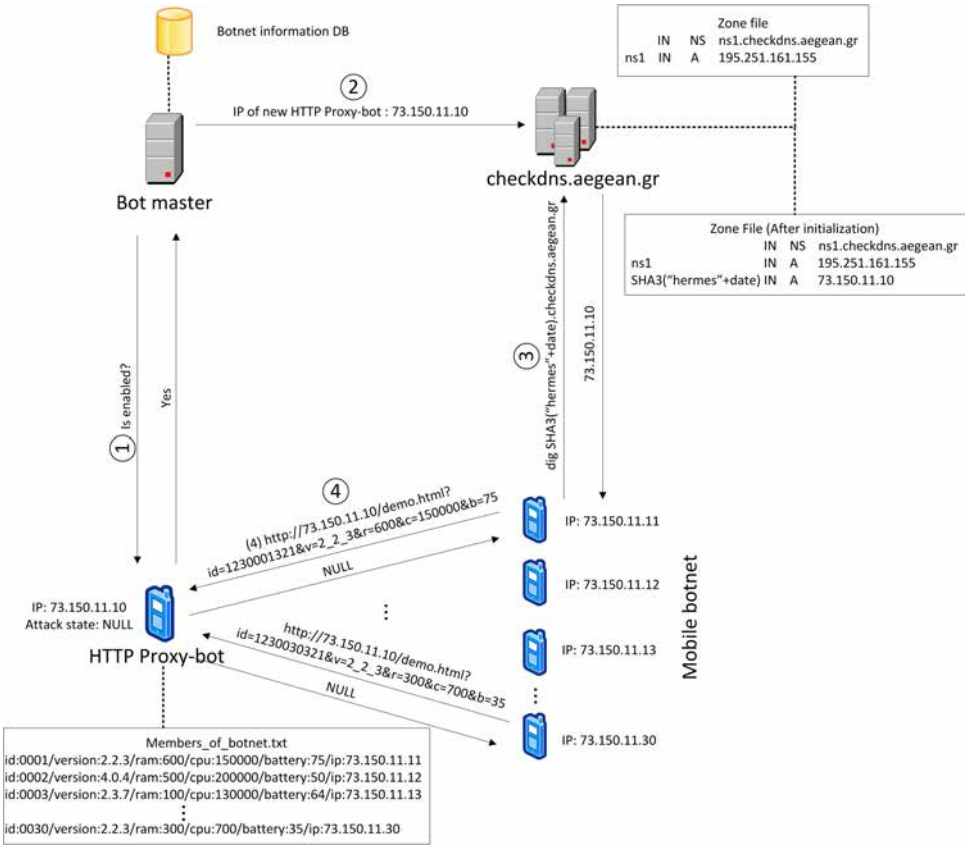



FIGURE 5.4: Architecture I: Initialization Phase

- A typical desktop computer that is responsible for the coordination of the bots, and therefore is controlled by the botnet operator.
- An HTTP server (hereinafter also referred to as "proxy"). As detailed in the following, the "proxy" role is interim; it is assigned to one of the mobile bots and frequently reassigned to another.
- A DNS ANS which is responsible for the resolution of the proxy's domain names.
- The bot agents, that is, infected mobile devices under the botmaster's implicit control.

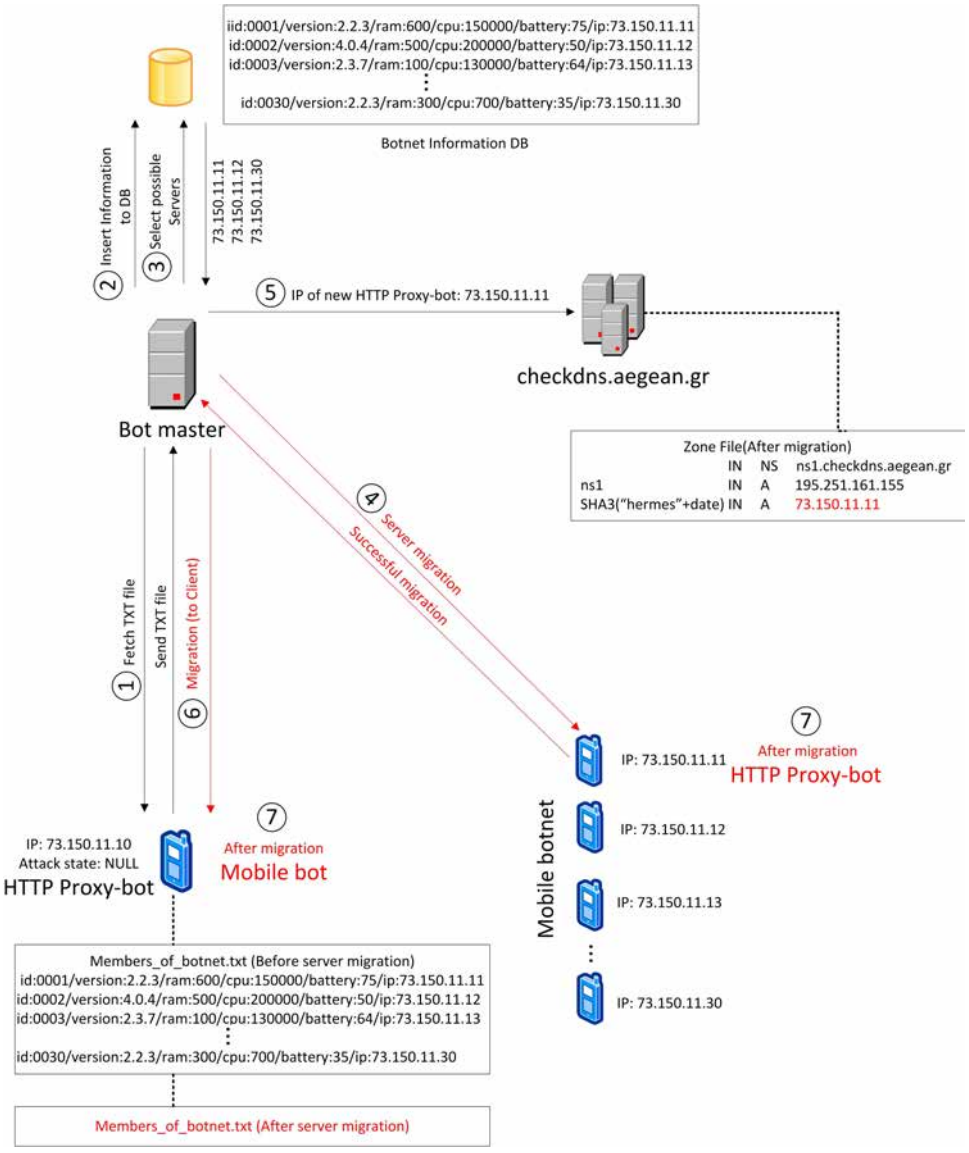



FIGURE 5.5: Architecture I: Migration Phase

In this architecture, the botmaster communicates directly only with the bot which is currently the proxy. Then, the proxy undertakes to disseminate the commands to the rest of the botnet's members. Following, the various bot agents send a DNS request to the DNS ANS with the aim to acquire the mapping IP address of the proxy. As a final step, they transmit an HTTP request to the proxy and receive the commands placed by the botmaster in the form of a HTTP response. It is therefore obvious that the botherder remains always hidden behind the transient proxy. The various phases of the botnet C&C establishment are analyzed below:

- *Initialization phase* - For the botnet to boot up, we consider that the very first proxy-bot exists on a specific IP address. This is a logical assumption as the

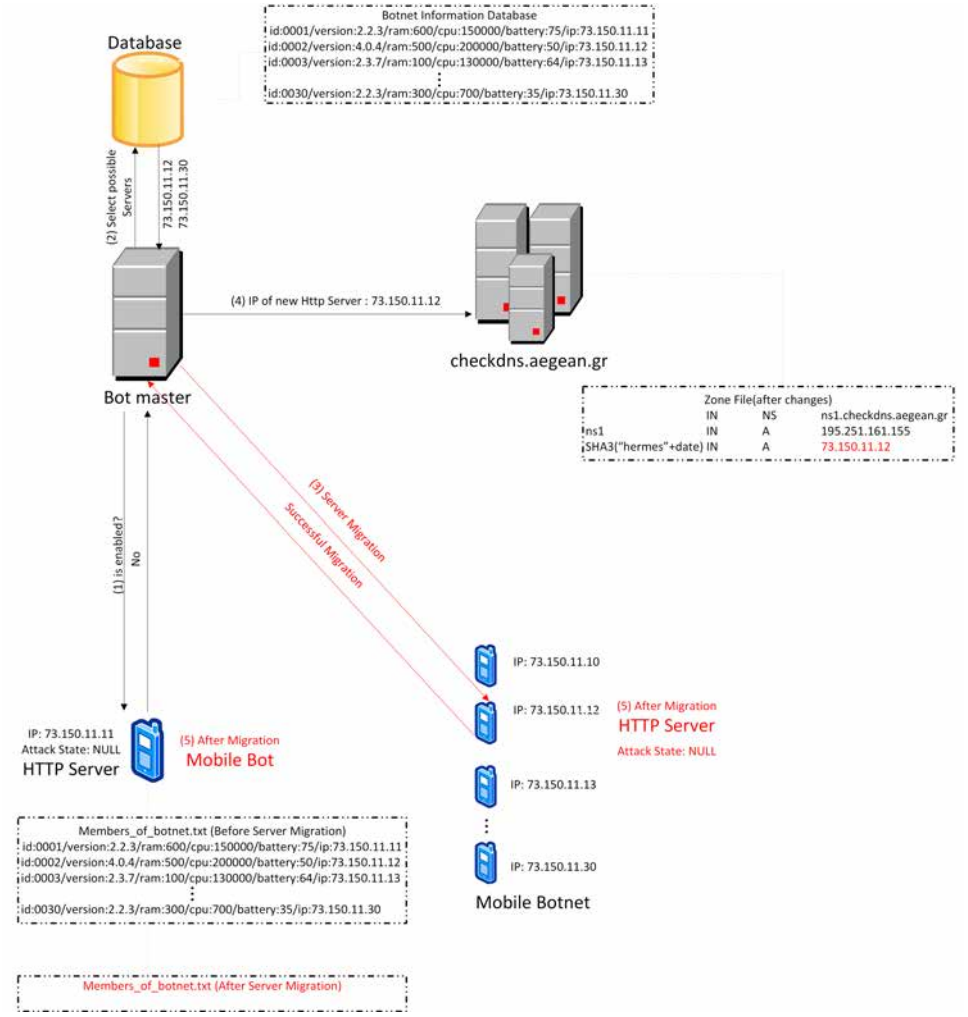



FIGURE 5.6: Architecture I: Recovery Phase

botnet operator can always employ a mobile device on their own. As the proxy will shortly shift, we assert that this would not jeopardise the camouflage of the botnet. First off, as it is depicted in Fig. 5.4, the botmaster queries whether the initiatory proxy is alive and listening (1). In the case of a affirmative response, they update the zone file of the DNS server with the RR that links the domain name of the proxy with its IP, that is, following a Fast-Flux strategy (see section 5.4). Naturally, this domain name is not a trivial one, but rather the output of the keyed-hash message authentication code (HMAC-SHA256) of the current global date with a secret code, which in our case is the string "hermes" (2). Note that with the use of the HMAC function, the domain name of the proxy will be different every day, and thus evade suspicion. Also, this rapid turnover makes it very hard to track down or block every possible domain name. Although in our case only the leftmost label of the domain label is generated, a potential attacker will easily

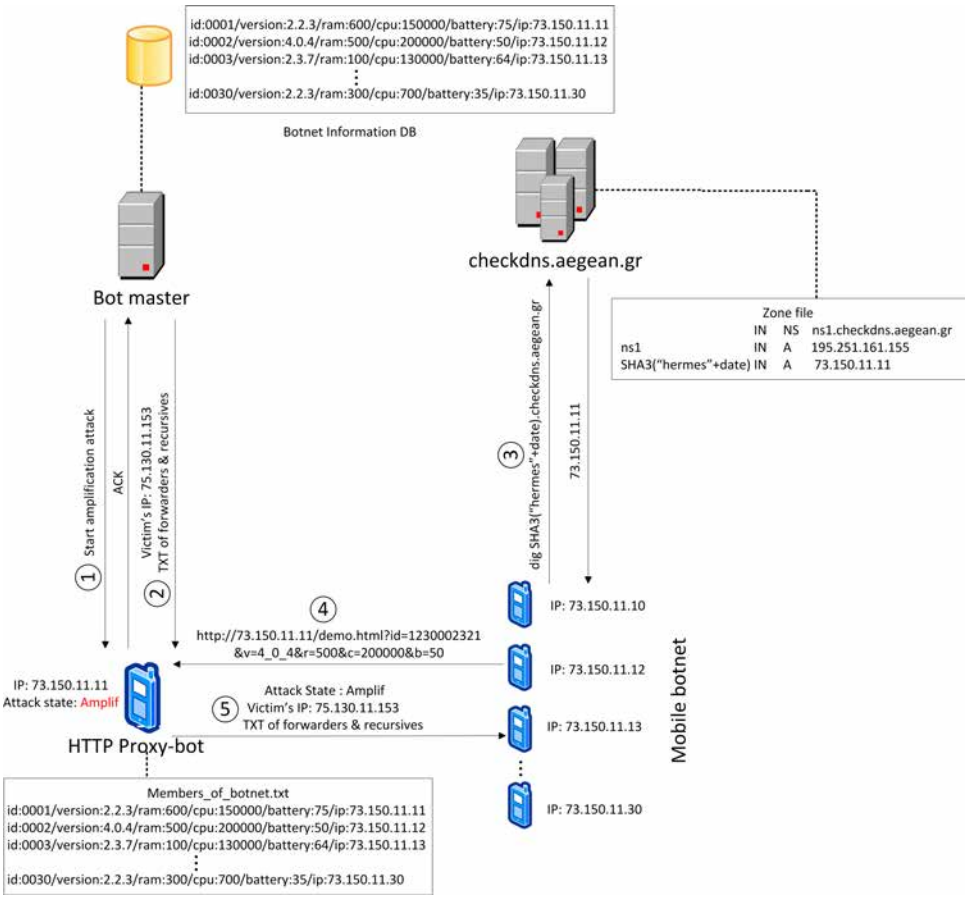



FIGURE 5.7: Architecture I: Amplification Attack

register FQDNs that will be produced by the aforementioned generation algorithm or even employ dynamic DNS providers. In such a case, the cybercrook is able to leak out even less traces of the botnet's infrastructure. Following, all the bots resolve the domain name of the proxy (3) and are able to connect with it via HTTP protocol. This query is repeatedly issued for example every couple of minutes in order the bots to become aware of any change regarding the proxy's IP address. After that, as the bots know their proxy, they create an HTTP GET request that includes information of their operational status and settings, namely RAM, CPU, level of battery power, device's ID (International Mobile Equipment Identity (IMEI) number) and OS version (4). In our scenario, these values are transmitted in cleartext. Nevertheless, a more careful botherder could encrypt them with the help of, say, a symmetric cipher. On the opposite, an encryption process will cause significant power consumption and require some sort of key management. In any case, we obfuscate the IMEI by padding meaningless numbers at the front and the end of the value. It can be safely argued that the remaining values do

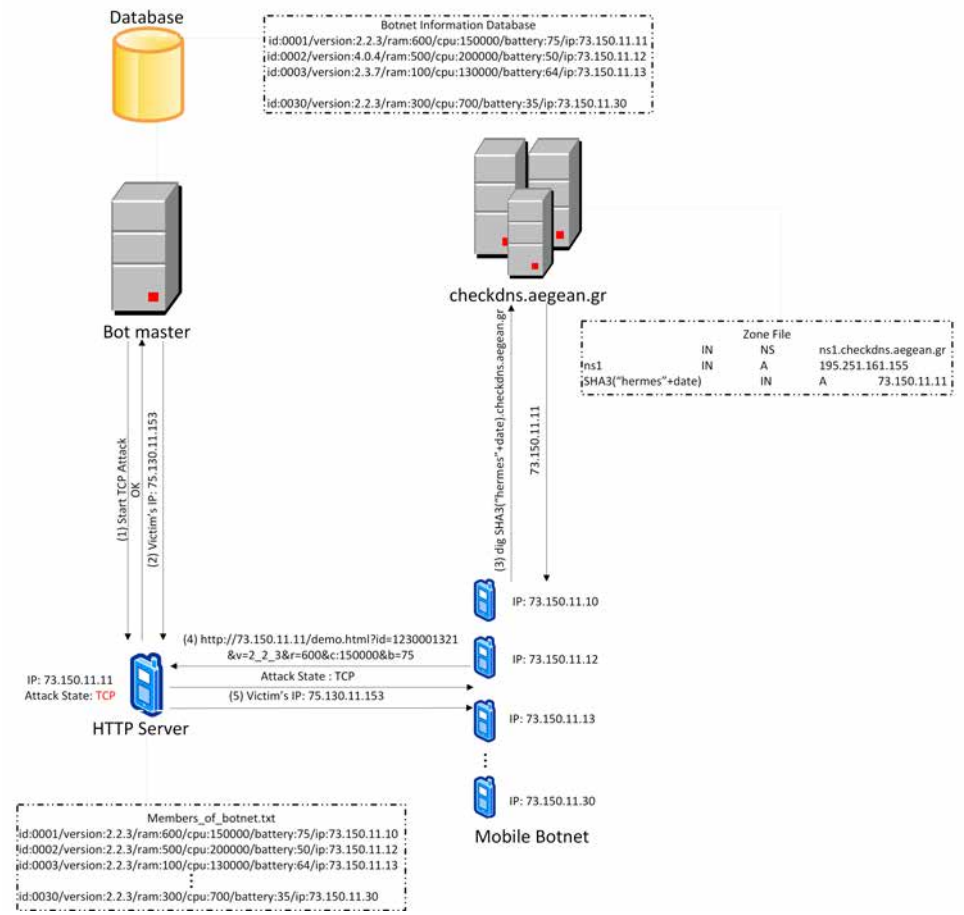



FIGURE 5.8: Architecture I: TCP Flooding Attack

not directly provide any useful info for the defender to trace the botnet. These provided information are used later on for proxy migration. In turn, the proxy collects all the information, adds to each record the bot's IP and responds with the status of the attack, indicating what action the various bots must take. For the moment, as the botnet is in its initialization phase, the bots should remain silent.

- *Proxy migration phase* - Every, say, five minutes the proxy should migrate to another bot. This procedure is presented in Fig. 5.5. As observed in the figure, before the migration begins, the botmaster requests the file that contains all the information of the bots (1). For the protection of the latter, this file is kept (and sent) encrypted with the public key of the botmaster. When the herder receives and decrypts the data with their private key, they update the corresponding database (DB) with the members of the bot. If a member is already registered in DB, they simply update its IP address and battery level, otherwise they create a new entry

based on the device ID (2). Next, the herder deduces which bots are capable of undertaking the role of proxy, based on their status, i.e., have enough computational power and acceptable battery level (3). From the possible candidates, they choose randomly one, and notify it to change its mode from simple bot to proxy (4). Also, they update the attack status in the new proxy to make it available to the bots to acquire. In case a migration problem occurs, the botmaster selects another candidate, otherwise they update the zone file (5) by modifying the relevant RR with the new IP address. Finally, the herder informs the previous proxy to change its mode back to bot and to erase any information related to the botnet, namely the encrypted text file that contains the information for all the members of the botnet (6).

- *Recovery phase* - One significant issue that needs to be considered is the case where the proxy fails. This, for example, may happen because the proxy-bot is deactivated by its owner or its battery reserves have been depleted. In the case a recovery process has not been foreseen, then the botnet will be uncoordinated for some time until the next proxy migration phase takes place. For this reason, as depicted in Fig. 5.6, the herder regularly sends an HTTP request (heartbeat) to the current proxy to examine whether it is alive. If not, the recovery phase initiates. Similarly to the mitigation phase, the botmaster chooses one of the candidate bots for the role of the proxy, notifies it to change its mode and the botnet attack status. At the same time the botmaster updates the zone file for announcing the new proxy to the bots. As the connection with the previous proxy was lost, in the case of restart, the app at the bot side is configured to operate as client and to also erase the information of the botnet, if any. This way, the previous proxy will begin to operate as a simple bot client, and therefore wipe out all the valuable traces of the botnet members.

- *Attack phase* - Whenever, the botnet operator decides to launch an attack, he triggers the attack phase. As depicted in Figs. 5.7 and 5.8, they notify the proxy for the status of the attack (1) and the corresponding parameters (2). As explained in section 5.8.1, in our case, we implement two DDoS attacks; the first one is based on [18] and targets mainly DNS authoritative servers (see Fig. 5.7), while the latter unleashes a typical TCP flooding attack targeting a multipurpose server (see Fig. 5.8). As illustrated in Fig. 5.7, for the amplification attack, the botmaster sends

the IP of the victim and a text file containing all the (open) DNS forwarders and RDNS that will be exploited during the assault. Note that these two messages are silently sent without an ACK, since the proxy affirms that it is alive in (1) message round-trip. Following, the proxy changes its attack status to "Amplif", meaning that a DNS Amplification attack must be unleashed. After that, as soon as the various bots connect to the proxy, they receive the attack status and the associated parameters (5). Soon after, they start to launch the DDoS attack. At the time the botmaster wishes to terminate the assault, they change the attack status to the string "NULL". The bots are informed for that change since as already pointed out in the initialization phase, the bots repetitively query the proxy to learn about its status. Bear in mind that during the attack phase the proxy may migrate to another bot. However, the attack continues uninterrupted because the new proxy is updated with the current attack status and the relative parameters. In a similar manner, for the case of a TCP connection flooding represented in Fig. 5.8, the botmaster changes the attack status to "TCP" (1) and informs about the victim's IP address (2). When the bots become aware of the target, they continuously issue large TCP packets to the network ports of the victim that are well-known to host popular services, e.g., WEB, FTP, SSH, DNS, etc.

It is stressed that depending on the size of the botnet, the botherder will enable two or more proxies for the dissemination of the commands. Thus, the zone file will contain one RR for each proxy and the various bots will randomly choose one of them to connect, say, in a round-robin fashion. This segmentation is also backed up by the fact that the mobile devices may not have the sufficient computational power to serve a large amount of HTTP requests.

#### 5.8.2.1 Architecture II: Mobile Botnet with PC-based proxies

The previous architecture could be considered as a pure mobile botnet, given that all bots are mobile devices. However, for diminishing the communication and processing costs, we came up with the idea of employing standard PC-based bots as proxies. Consequently, this second architecture constitutes a variant of the first one, with the difference that only desktop PC bots acquire the role of the proxy machine (HTTP server) and not

mobile ones. The advantage of doing so is that a PC has fewer chances to become non-operational due to power constraints etc. Also, it has far more computational capabilities to serve a larger number of bots. Putting it another way, in this architecture, the burden of the attack is undertaken by a mobile botnet, while the coordination of the mobile botnet is carried out by a PC-based botnet. To sum up, this scenario uses the same entities as the first one except the desktop PC bots. Similar with architecture I, we perceive four distinct phases.

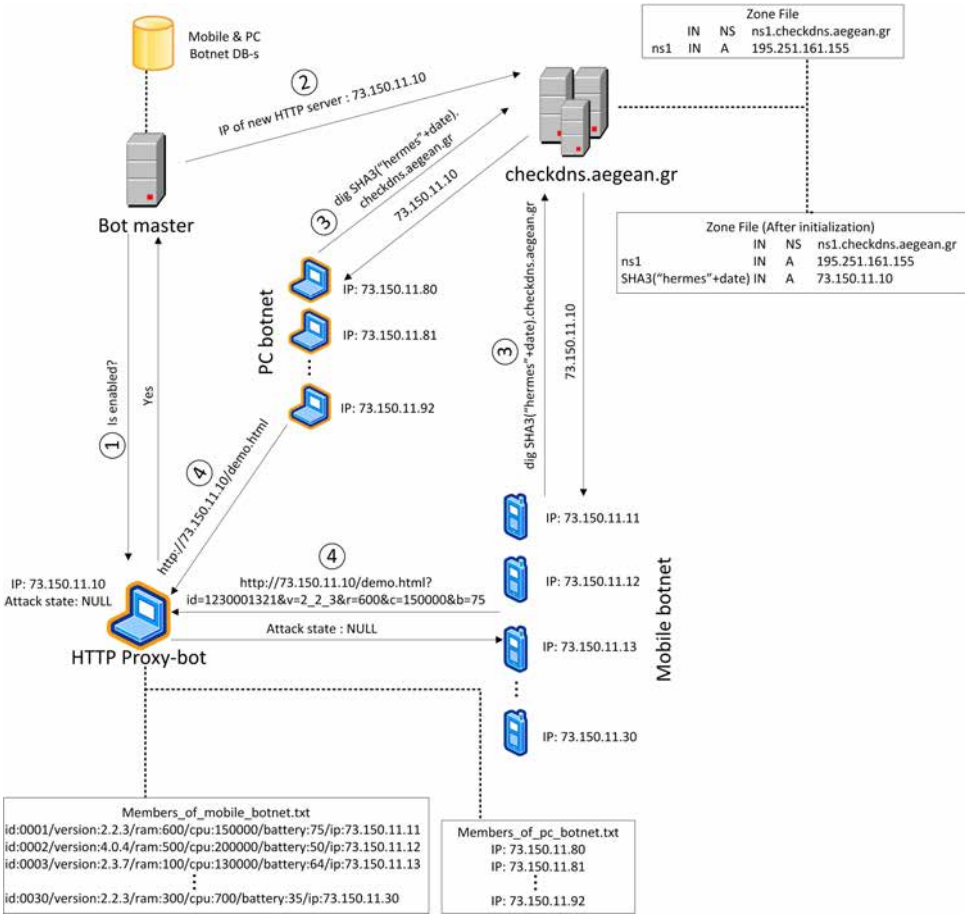



FIGURE 5.9: Architecture II: Initialization Phase

- *Initialization phase* - As depicted in Fig. 5.9, similarly to the first scenario, the botmaster queries whether the proxy is alive (1) and updates the zone file with its IP, if this is the case (2). The main difference is perceived in steps (3) & (4), where the members of the PC-based botnet are introduced. In (3) the PC bots resolve the domain name of the proxy to which following they send an HTTP request for

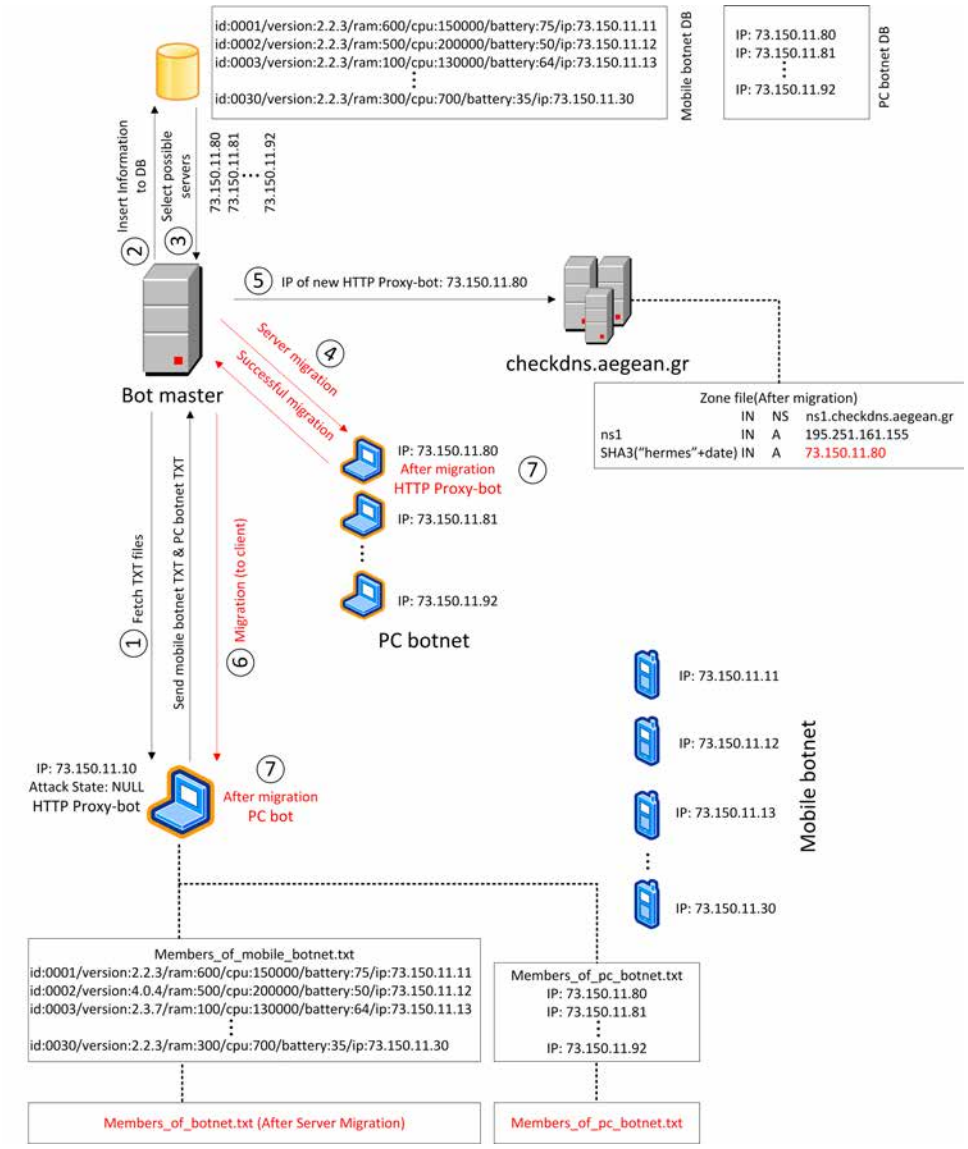



FIGURE 5.10: Architecture II: Migration Phase

registering (4). Additionally, the proxy manages a list with the details of the PC-based botnet, from which the botmaster chooses the candidates for determining the next proxy.

- *Proxy migration phase* - Like in the first scenario, the migration phase happens regularly for the transition of the proxy to be completed. As shown in Fig. 5.10, prior to migration, the botmaster requests the files that contain all the operational information of both the PC and mobile bots (1) and updates the corresponding DB (2). From the available PC bots, the botherder chooses randomly one (3) and notifies it to alter its mode from simple bot to that of a proxy (4).

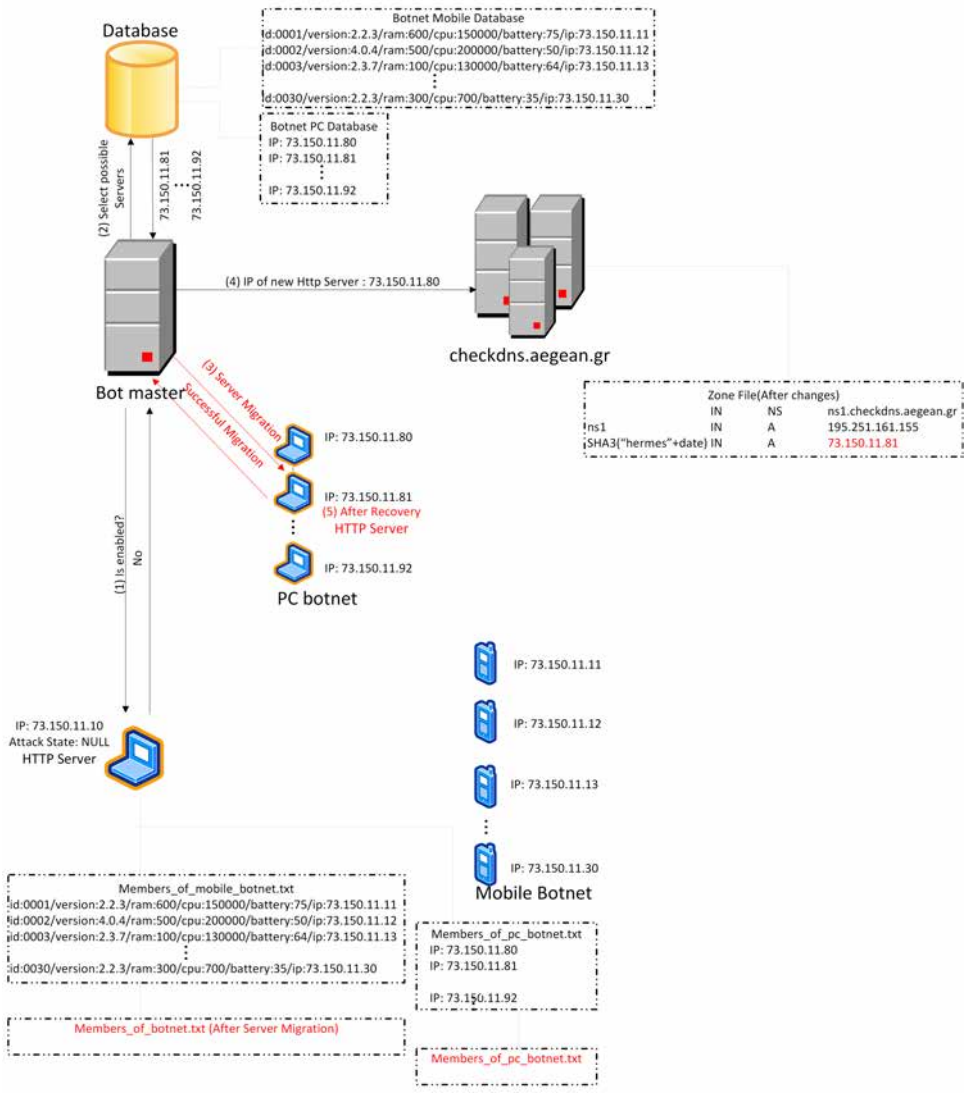



FIGURE 5.11: Architecture II: Recovery Phase

- *Recovery phase* - This phase is exactly the same as in architecture I,, where the candidates are picked up from the list of the infected PCs. This process is also detailed in 5.11. However, we expect that the probability of PC-based proxy to crash is much lesser than that of a mobile one.

- *Attack phase* - For the launch of the attack, the botherder changes the attack status in the proxy (1) and provides the required parameters (2), namely the IP address of the victim and the type of the attack (Fig. 5.12 and 5.13). Through the use of HTTP GET requests (5) the commands, either "Amplif" or "TCP" attack status, are being disseminated to the mobile bots (6) for them to start the DDoS assault. Note that the remaining PCs of the PC-based botnet do not participate in the actual attack as an extra precaution against detection since their role is

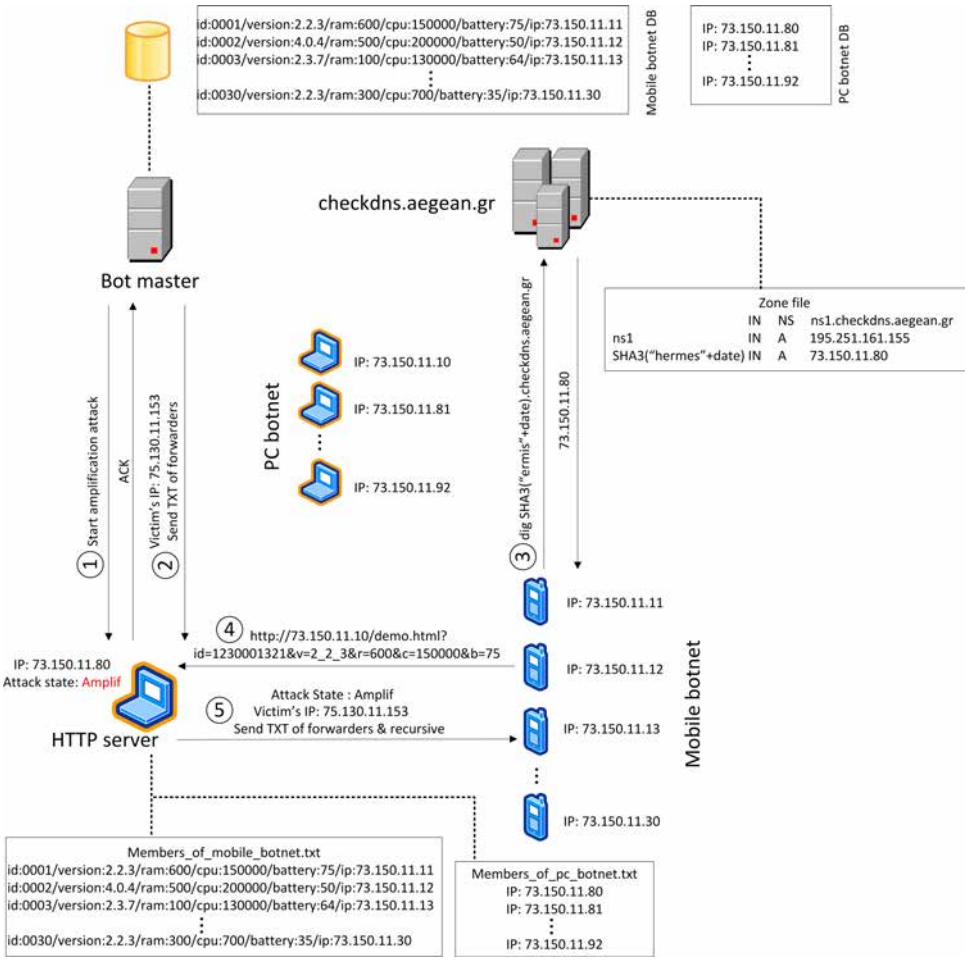



FIGURE 5.12: Architecture II: Amplification Attack

more valuable as proxies than attacking bots. Actually, this decision is up to the botherder and usually depends on several factors, including the number of the PC-based bots, their geographical dispersion, etc.

#### 5.8.2.2 Architecture III: Exploiting DNS as covert C&C channel

The main concept behind this third architecture is to use DNS protocol as a covert channel for coordinating the botnet. An overview of this architecture is depicted in Fig. 5.14. The core idea here is that the botmaster controls a DNS ANS and publish the relative parameters of the attack as RRs to that zone. Thus, the various mobile bots will neither contact the botmaster directly nor via a proxy, but instead they directly receive their instructions through the DNS ANS. Similar to previous scenarios, as observed from Fig. 5.14, the domain name of the RR is not a trivial name, but rather the output of the HMAC-SHA256 hash function taking as input the current global date and a secret

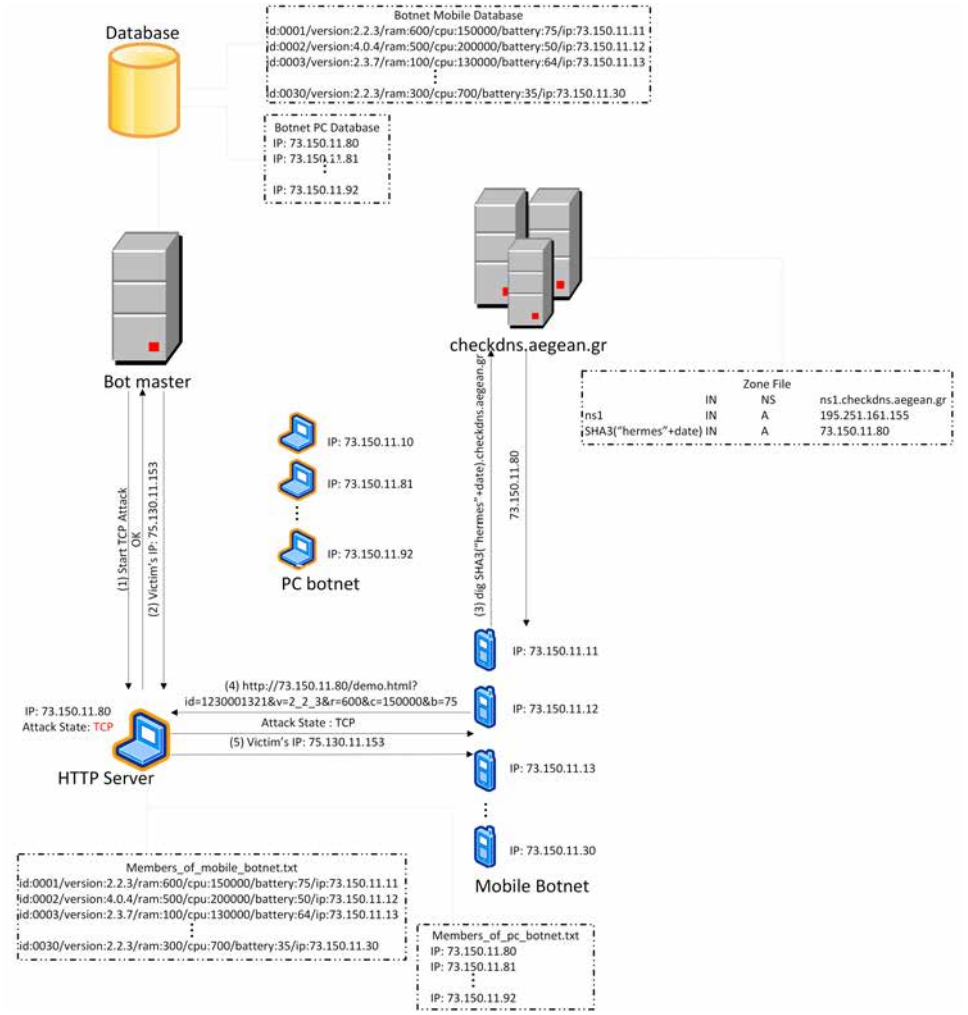



FIGURE 5.13: Architecture II: TCP Flooding Attack

alphanumeric sequence. Note that the use of a hash value ensures that the domain name will be disparate every day, and thus evade suspicion. Once more, although in our proposal only the leftmost label of the domain label is random, a potential attacker will easily register domain names that will be produced by the generation algorithm. In such a case, they are able to exploit various DNS ANS and leave minimum traces of the botnet's C&C activity. The two phases that complete this attack variation are given below.

- *Initialization phase* - Originally, the zone file of the DNS ANS contains only the RRs that are required for the operation of the zone, namely which nameserver is responsible for that zone (NS record) and in which IP address it is located (A record). For the coordination of the attack, the botmaster updates dynamically the zone file with two RR of type A (1). As explained in section 2.1.9, this type

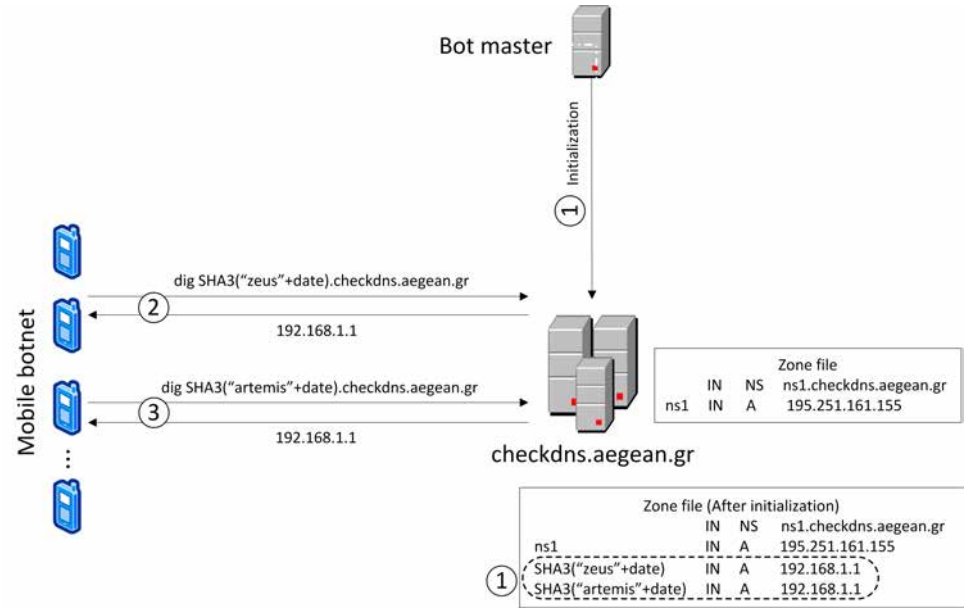



FIGURE 5.14: Architecture III: Initialization Phase

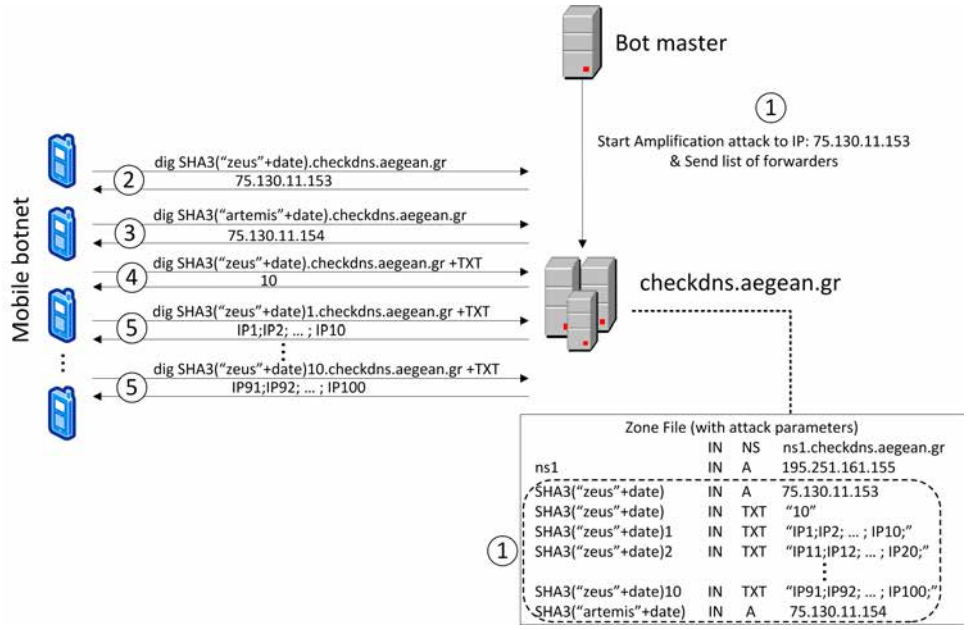



FIGURE 5.15: Architecture III: Amplification Attack

of RR maps a domain name with its corresponding IP address. The first A record corresponds to the amplification attack, while the second to the TCP flooding one. The purpose of these records is to signal the beginning of the attack and indicate which the target is. The domain names of these records are generated by the HMAC-SHA256 function, getting as input the current date and the alphanumeric "zeus" and "artemis" respectively. With the usage of the current global date, the name of the RR is different every day. In the beginning, the A RRs map to the

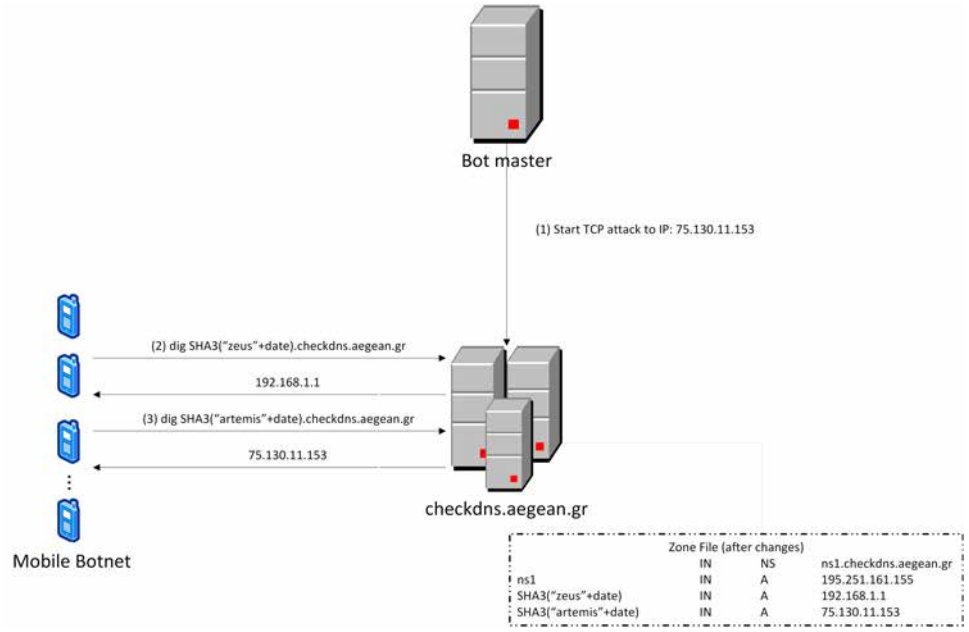



FIGURE 5.16: Architecture III: TCP Flooding Attack

private IP address 192.168.1.1, indicating that the bots must not take any action and stay silent. All the mobile bots periodically generate the random domain names and resolve their IP addresses (2) & (3). If the answer contains the IP address 192.168.1.1, then the bots stay silent. Otherwise, they obey the attack instructions as the case may be. It is not compulsory for the various bots to directly issue the DNS queries toward the ANS controlled by the botmaster, but rather they could consult the local recursive resolver. In such a case, RRs should have a zero TTL value.

- *Attack phase* - Whenever the botmaster wishes to unleash a DDoS assault, they dynamically modify the contents of the zone file with the parameters of the attack. Depending on which kind of attack they desire to execute, i.e., DNS amplification (Fig. 5.15) or TCP flooding (Fig. 5.16), or even both, they add to the corresponding A RR the IP address of the victim (1). As for the DNS amplification attack, the bots require the list of the exploited DNS forwarders, so the botmaster adds one or more TXT RRs that contain this list. Usually, the number of the DNS forwarders employed in [18] is large enough, thus the added TXT RRs are more than one. Hence, the TXT RR that matches to the domain name *HMAC-SHA256("zeus"+date)* indicates the number of the required TXT RR for the publication of the complete list of the forwarders, while the RRs of the domain

name *HMAC-SHA256("zeus"+date)1*, *HMAC-SHA256("zeus"+date)2*, etc., contain a fragment of that list. Consequently, at the time the bots observe the change of the A record, they know that they should commence the corresponding DDoS attack(s) (2). After that, they resolve the TXT record to get informed of the size of the list (4). Finally, the bot agents issue as many DNS queries as needed to acquire the complete list of the DNS forwarders (5). Then, they are ready to initiate the attack(s). Any time the attacker desires to terminate the assault, they modify the A RR to the private IP address 192.168.1.1 and removes the appropriate TXT RRs. So, the bots are notified of the change and they cease their part of the attack instantly. Similarly, for the TCP flooding attack case, the aggressor updates the RR corresponding to the domain name *HMAC-SHA256("artermis"+date)* with the IP address of the victim. Thereupon, the bots become aware that they have to start TCP connections to that IP address (3). From that point onward, the bots issue a surge of TCP queries to ports known to host popular network services until the botherder modifies the IP address of the RR back to 192.168.1.1.

#### 5.8.2.3 Other considerations

The decision of using a collateral PC-based botnet for dispatching proxy communications depends on the size of the mobile botnet and other parameters as the case may be. For example, it is natural to think that the greater the number of mobile bots the greater the need for PC-based proxies in order to easily control all of them and increase robustness.. However, if the botmaster already is in control of such a PC-based botnet it can always take advantage of its services. Furthermore, depending on the attack impact the botmaster wishes to accomplish, i.e., immediate collapse of the target or "low and slow" attack, they will form accordingly the size of the botnet. Segmentation of the whole space of the botnet is also possible. So overall, in the eyes of the botmaster it is basically a matter of how rich, populous and scattered the bot arsenal is.

### 5.8.3 Comparison of architectures and Results

The aforementioned architectures present a novel mechanism to coordinate a mobile botnet. The first two exploit one of the infected device to act as HTTP (proxy) server, while the third one uses a DNS ANS as the means for disseminating the commands.

The basic difference between the first two architectures resides on the fact that a mobile device has limited resources to handle a large number of bot clients. On the other hand, a PC-based proxy is more reliable and has fewer chances to crash or go offline. Additionally, the PC-based bots do not participate in the final DDoS attack(s), they do not reveal directly their location, and thus is more challenging to get detected. In any case, our results described in the remaining of this section indicate that if the attacker employs multiple proxies simultaneously, the impact of the attack will be the same in both cases (i.e., either with mobile or PC-based proxies). The most notable advantage of the third scenario is that the bots do not directly connect to the botmaster, but rather they issue legitimate DNS queries for frequently changing the RRs that correspond to the targeted machine. More importantly, since mobile devices are used for accessing websites or other network resources by their owners, the portion of DNS traffic originating from the botnet coordination is minimal compared to the whole traffic.

For creating the botnet and implementing the attacks, twelve Sony Xperia L Android Jelly Bean mobile devices were utilized in total. Each device has a dual core 1GHz CPU and 1GB RAM and was connected to a wireless hotspot. On the other side, the victim was a desktop machine equipped with a Dual 2.8 GHz CPU and 4 GB RAM connected to a 100 Mbps network interface. This machine acts as DNS ANS having the DNSSEC extension enabled. For each scenario and for each type of attack a botnet consisting of 1, 6 and 12 members has been created. In this way, we were able to infer the accumulative impact of the gradual activation of the mobile bots. Our tests demonstrate that every mobile bot is capable of executing 3 instances of the client attack script simultaneously without increasing the computational burden to a level that is perceptible by the owner of the mobile device.

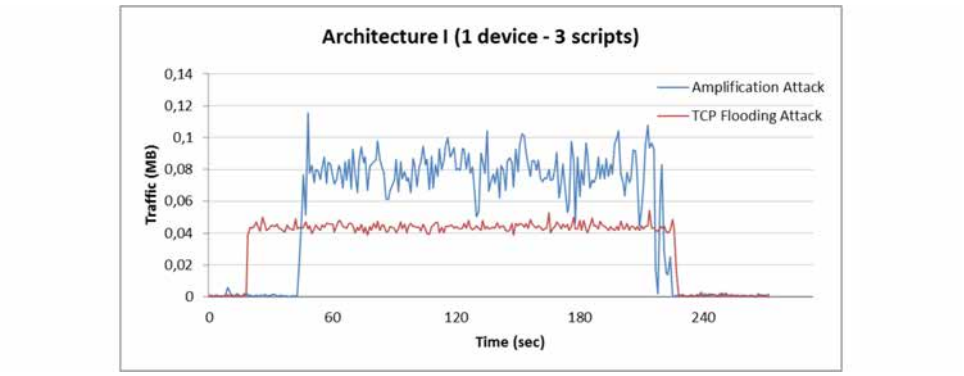
Naturally, the number of attack scripts a device can bear prior its user notices it depends on the underlying hardware and OS. Therefore, given that each bot reports its performance capacities to the C&C server, the latter may dynamically instruct the bot about the number of attack instances it should initiate. The concurrent execution of three instances of the attack script creates a stream of nearly 34 DNS queries per second on average or 2.33 KBps. Also, this implies that in the ideal case each attacking bot is capable of flooding the target with 34 DNS responses per second. However, this data volume is multiplied by the amplification factor, which as it is discussed further down varies from 32.7 to 34.1 and it is solely dependent on the behavior of the chosen DNS

forwarders. Consequently, a single attacking bot unleashes on average a stream of 76.9 KBps towards the victim. For compiling the pool of DNS forwarders to be used in the attack, we considered to examine the network IP blocks of Greece similar to [18] (details on this procedure are given in section 4.2). From the located open forwarders, we kept about a number of 1.1K because those return back large DNSSEC-related RRs.

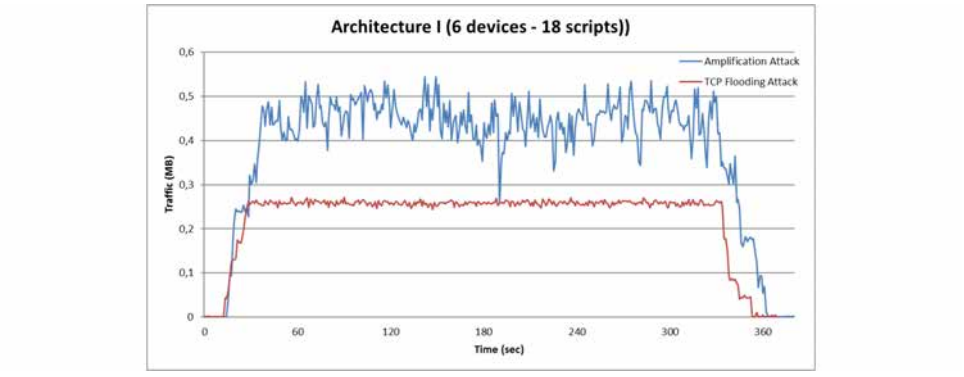
Regarding architecture I, Fig. 5.17 depicts the level of bandwidth consumption at the victim's side for both attack variations and for 1, 6 and 12 bots respectively. The flow of the inbound traffic remains similar for the rest of the architectures. Furthermore, table 5.1 details the average volume of the inbound network traffic for all three architectures. The case of the single bot is implemented with the intent to accurately calculate the amplification factor of the DNS amplification. During this experiment 6,000 DNS queries or 0.4 MB was dispatched by this single bot, given that in average our DNS request had a size of 70 bytes. On the other end, the victim received 13.64 MB of network traffic, which is translated to an amplification factor of 34.1. We can conclude that in total the DNS amplification attack creates larger volume of traffic towards the target because of the amplification nature of the attack. Also, a smoothness in the incoming traffic of the TCP flooding can be observed due to the direct connection amongst the bots and the victim. While for DNS amplification a variation with both upward and downward peaks is presented as the forwarders interpose in the communication.

As expected, the second architecture reveals pretty much the same results with the previous one. Similarly to the first scenario, a single bot was able to send about 6,000 DNS queries or 0.4 MB and the target was flooded with 13.1 MB, which is reflected to an amplification factor of 32.75. As the current configuration facilitates the recording of the proxy's network traffic, we analyze the required HTTP transactions for the coordination of the botnet. As such a procedure imposes the usage of a sniffer app on the proxy side, it is normal to affect its operation as a bot too. However, with high confidence, it is asserted that the following analysis is identical for both architectures, because the proxy operates exactly the same way.

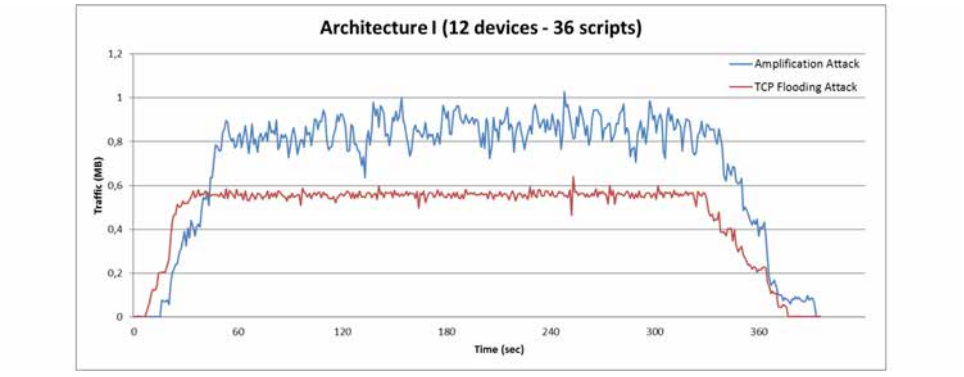
As observed from Fig. 5.18(a), between the botmaster and the current proxy, two types of transactions take place. The first one, depicted in red color, happens during the attack phase, where the botmaster sends the attack parameters. The three red upward pointing peaks observed in Fig. 5.18(a) occur during the time of the launch of the




(a) Botnet size : 1 bot




(b) Botnet size : 6 bots

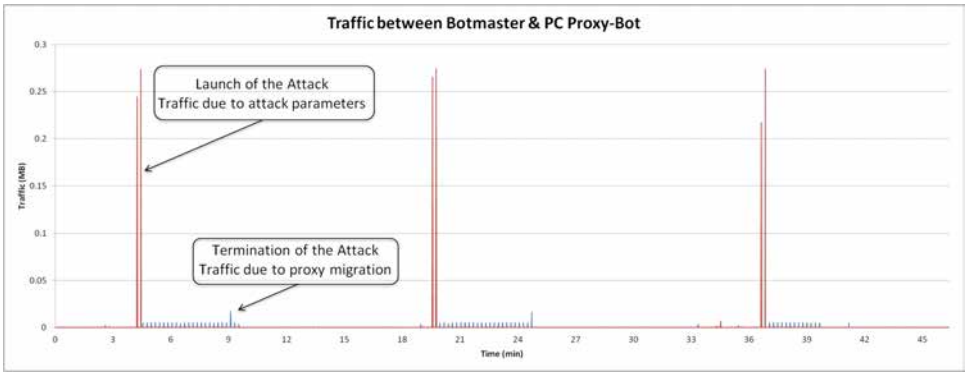



(c) Botnet size : 12 bots

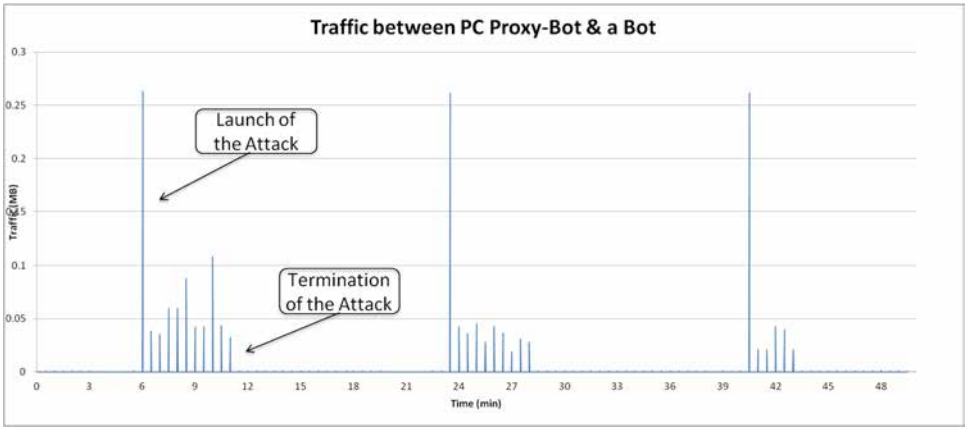
FIGURE 5.17: Inbound traffic in MBps for both attack variations of Architecture I

attack, that is, when the botmaster transfers the list with the DNS forwarders. The second type displayed in blue color refers to the phase of migration. This traffic is mostly evident during the attack, as the botmaster sends again the list of the forwarders to the new proxy. Moreover, between the HTTP proxy and a bot the only noticeable traffic occurs during the attack phase (Fig. 5.18(b)). At that moment, the proxy informs each connecting bot about the available list of DNS forwarders, thus creating a peak of 0.25 MBps. During the attack, there is a tiny amount of traffic as the proxy sends the

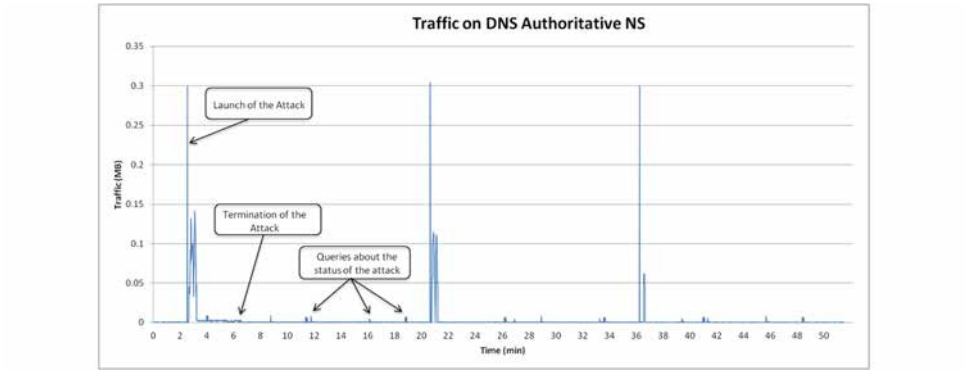
list regardless if the bot has already receive it or not. This happens because the proxy does not record if a bot is already in possession of the list required for the amplification attack.




(a) Network traffic between Botmaster and PC Proxy




(b) Network traffic between Proxy and a single Bot




(c) DNS traffic for the botnet coordination

FIGURE 5.18: Architecture III: Network traffic generated due to botnet coordination

Similarly to the previous ones, the third scenario reveals nearly the same outcome on the victim side. A single bot was capable of sending about 6,000 DNS queries or 0.4 MB and the target was flooded with 13.4 MB, which corresponds to an amplification

factor of 33.5. For this scenario is interesting to examine the burden imposed on the DNS ANS for the coordination of the botnet. Specifically, we capture the DNS network traffic using a sniffer app. As it is observed from Fig. 5.18(c), there are three upward pointing peaks in the network traffic which correspond to the launch of the amplification attack. Specifically, the initial bursts originate from the dynamic update of the zone with the attack parameters, while the subsequent spikes are due to the requests by the various bots about the necessary DNS records during the attack phase.

| | **Architecture I** | | **Architecture II** | | **Architecture III** | |
|---|---|---|---|---|---|---|
| **Number of attacking nodes** | **Amplif** | **TCP Flooding** | **Amplif** | **TCP Flooding** | **Amplif** | **TCP Flooding** |
| 1 (3 scripts) | 0.075 | 0.042 | 0.074 | 0.042 | 0.074 | 0.042 |
| 6 (18 scripts) | 0.45 | 0.27 | 0.43 | 0.25 | 0.37 | 0.22 |
| 12 (36 scripts) | 0.75 | 0.5 | 0.85 | 0.58 | 0.80 | 0.57 |

TABLE 5.1: Inbound traffic in MBps proportional to the number of attacking bots per architecture

## 5.9 Countermeasures

One way to cease the operation of a botnet is to block or completely interrupt the C&C channel. As already pointed out in section 5.4, when the bots are unable to contact with their botherder they cannot receive their commands. Following, we explain the various tactics followed by the defenders for tracking and shutting down the operation of a botnet. Initially, section 5.9.1 describes DNS-based solutions exhibiting possible ways to reveal the members of a botnet, either simple bots or C&C servers, while section 5.9.2 discusses DNS-based hijacking methods of C&C infrastructure.

### 5.9.1 DNS-based Botnet detection

In their majority, the measurements of this type are based on the analysis of DNS network traffic, either by active probing or passive investigation, and aim to discern between domain names that are part of a flux mechanism and benign ones. As soon as the possible rendezvous points of the C&C server are perceived, then as explained in section 5.9.2 further down stoppage attempts can be initiated.

#### 5.9.1.1 Detection of DNS fluxing

The first empirical study of DNS Fast-Flux was conducted by Holz et al. [228] (Feb 2008). The authors observed that fast-flux agents were located in disparate IP addresses belonging to various ASNs. Therefore, the number of distinct DNS records of A type might be large in a single lookup. Moreover, the number of NS type records might exhibit similar behavior since in the case of double flux also the ANS are part of the fast-flux mechanism. The authors designated a set of features which considered the number of divergent IP addresses in a single and multiple lookups, the number of ASs that these belong, and the number of DNS NS. They experimented with the aforementioned metrics against manually verified fast-flux and legitimate domains in order to calculate the significance weight for each metric. In overall, they recorded over 50,000 unique IP addresses during the measurement period of four weeks for the case of *Storm Worm*. According to the authors' evaluation, the proposed metrics could contribute to the automatic identification of fast-flux domains and to blacklist them. The novelty of the work in [228] is that it is one to designate specific characteristics of fast-flux networks that, combined together, could allow to accurately detect whether a suspicious IP address hosts a fast-flux service or not.

Passerini et al. (July 2008) [229] presented a system to detect and monitor fast-flux service networks, called *FluXOR*. This system applies a set of suitable features for distinguishing between malicious and legitimate domain names. The detection features used by FluXOR depend on specific domain names' characteristics, the degree of availability of the potential fast-flux network, and the heterogeneity of the corresponding flux agents. In total, the authors determined 9 distinct features based on the aforementioned 3 categories. For the first category, they concluded that domain names related with fast-flux services are registered at a rapid pace and hence are active for short time. In fact, the average age of malicious domains is less than five weeks, while for benign is much greater. Also, these domains are registered to specific (suspicious) registrants located to countries with lax legislation. As concerns the degree of availability, the number of involved IP addresses and the value of TTL are taken into consideration. Finally, for estimating the heterogeneity of flux agents, the system investigates the number of discrete networks, ASs, host (reverse lookup) and network names, and organizations that the IPs belong to. Following, by using the 9 features, they applied a naive Bayesian classifier

to a dataset of domain names collected from spam emails. They planned to also utilize web crawlers and honeypots as a source of suspicious domain names. The corresponding classifier was able to identify correctly flux domains with zero false positive. For a month deployment, FluXOR detected 387 fast-flux service networks, which totally composed by 31,998 unique bots. It is believed that the detected bots were associated with 16 different botnets. The contribution of the work in [229] is that expands the number of features to 9, considering other aspects of domain names not solely correlated with DNS RRs. Namely, it utilized also WHOIS queries and registration information.

At about the same time, Nazario et al. [6] (Oct 2008) proposed a system for identifying fast-flux domain names. Their system is designed to first collect suspicious names from spam mails, blacklists or malware analysis, and then to investigate the relative DNS RR for the collected domains. As in the previous works on this topic, their classification were based on features typical for fast-flux domain names, for instance, low TTL value, number of different IP addresses corresponding to the same domain name, network distance amongst these IP addresses, number of distinct AS that belong, and so forth. The idea behind selecting these criteria is that a botherder will choose to set corresponding DNS records (of A type) to have a short TTL value and be widely dispersed across the world with the purpose of providing maximum availability of their C&C infrastructure. Nevertheless, for the case of Content Delivery Networks (CDN), a whitelist is used to exclude legitimate domains, which may employ similar to fast-flux techniques for load balancing. Once a domain name is determined as fluxed, then consecutive DNS queries are issued to locate the members (in terms of IP addresses) of the botnet. In total, the authors were able to identify 928 distinct fast-flux domains and track down 15,080,044 distinct IP addresses during a period of 4 months. The contribution of this proposal lies on the consideration of the number and characteristics of the utilized features, for example short TTL value etc.

Nearly a year after, Hu et al. [230] (February, 2009) introduced a system for detection of fast-flux domains which combines analysis of network traces, derived from NetFlow, spam mails and DNS logs, and active probing. The network traces provide data about suspicious domains and their correlation with IPs. After that, further DNS probes for the suspected domains reveal information about the corresponding IP, the ANS, the ANS's IP, reverse lookup, and so on. Finally, a SVM classifier is applied to deduce whether a domain name is benign or fluxed. The authors evaluate their proposal for a period of two

months using NetFlow records from their university's edge router, spam emails, and DNS logs from local resolvers. The results demonstrate the capability for rapid identification with low false positive rate for the case of aggressive fast-flux domains. On the other hand, for stealthy fast-flux domains, namely domains exhibiting valid DNS behavior such as CDNs, a monitoring for longer period is necessary for accurate detection. The contribution of the work in [230] is that assembles possible fluxed domains from various sources not solely from spam emails. Although, the authors' system utilizes passive network traces, it highly depends on DNS active probing.

The first work coping with passive analysis of DNS traffic traces extracted from DNS resolvers is is that of Perdisci et al. [231] (Dec 2009). They proposed a passive DNS traffic analysis system for detecting and tracking malicious fast flux networks. More specifically, their system examines DNS traffic originating from clients' machines heading to their local DNS resolvers. Thereby, domain names related to various malicious actions and not solely collected from spam traps or blacklists are considered. For instance, the DNS traffic may contain domain relevant to blog spam, social websites spam, search engine spam, instant messaging spam, and so on. The system captures in front of the resolvers the DNS network traffic and stores it in a central DB. Initially, a clearing of definitely non-fluxing domains is requisite due to the excessive volume of DNS traffic. Then, a clustering of domains is applied based on their relation, for instance because they resolve to a common set of IPs. These clusters represent a candidate network, i.e., a probably fast flux network. Finally, a statistical classifier is applied with the aim of detecting if a group of domains is indeed relevant to a malicious flux or a non-flux network. The classifier is based on a supervised statistical algorithm trained from flux and non-flux samples. The examined features take into consideration the number of distinct IPs and domains, TTL values, IP diversity, IP addresses growth (i.e., new discovered IPs), and so forth. The latter characteristics illustrate the expansion of the cluster in time, namely the rapid change of IP addresses corresponding to the same set of domain names.

The novelty of the above mentioned system compared with the previous ones is that it does not consider single domains independently from each other, but rather it acknowledges the fact that many fast flux networks involve more than one domain name simultaneously. Besides, it utilizes DNS traces passively captured from live users' traffic. So, it collects data without interactions with the flux networks. On the other hand, active probing of fast-flux domains may create noise on the DNS ANS which are usually

controlled by the botherder, and thus possibly get detected [231]. In the case of an active probing detection, the attacker may stop responding to DNS queries coming from the probing system to prevent revealing further information. Moreover, the system raises serious privacy concerns as it monitors DNS traffic directly from the end-users. The authors evaluated their proposal for a period of 45 days collecting DNS traces from two large ISP networks. Their experimental results showed that the passive analysis of DNS traffic can contribute to the accurate detection of malicious flux service networks.

In a subsequent work by the same group of authors [232] (Sept 2012) a similar passive DNS traffic analysis system, called *FluxBuster*. In fact, this proposal follows a similar rationale to [231]. However, in this case, the proposed system captures DNS traffic traces "above" DNS resolvers located on various networks around the world. By doing so the privacy of the end-users is protected, as the DNS traces are aggregated for all the mass of the clients served by a DNS resolver. The authors evaluated their proposal for a period of five months. The results showed that FluxBuster was capable to correctly identify malicious flux domains in the wild with a low false positive rate. The contribution of this proposal compared to [231] is firstly the protection of client's privacy and secondly the execution of a large-scale analysis.

### 5.9.2 Botnet shutdown operation

#### 5.9.2.1 Botnet Sinkholing

A prevailing method to shut down a botnet is by seizing the control of the C&C channel. The work in [233] defines the specific steps followed by the defenders for taking over the C&C channel and eventually to halt the operation of the bots. The described procedure is similar to any other attempt of botnets' hijacking by controlling the C&C channel. However, the taking over of the C&C channel is applicable mostly to the case of a centralized botnet, not to those employing P2P communication channels.

The work of Stone-Gross et al. [233] encountered with *Torpig*, a bot malware that uses domain fluxing with DGA for locating the C&C server and HTTP protocol for C&C covert channel. In the case of Torpig, the DGA takes as seed the current date and a number. Furthermore, the botherders registered the generated domain name in .com and .net zones. The latter domain zone was utilized for reasons of redundancy. However,

the botherders failed to allocate in advance the generated domain for the subsequent weeks. Hence, the defenders reversed-engineered the bot malware and unveiled the DGA and its hardcoded parameters. So, they anticipated the domain name of DNS fluxing mechanism for the forthcoming weeks and managed to register the domain names on .com and .net registrant. The corresponding DNS records were resolving to an IP address which hosted a device under the control of the defenders playing the role of C&C server. This DNS redirection is also called botnet *sinkholing* [225]. Thereby, in the case the defenders decode the dialect of C&C channel, they can simply instruct the bots to stay dormant or uninstall the malware. Furthermore, the sinkholing facilitates the evaluation of the botnet's size, since the various bots connect directly to the fake C&C server, and thus are directly observable.

Nevertheless, the success of a botnet's sinkholing attempt relies mostly on economical factors, since the registration of a single domain name is costly. Specifically, the botmasters could render countermeasures against domain fluxing economically infeasible by forcing the DGA to produce an excessive number of domain names. In fact, *Conficker* malware is capable of producing 50,000 domains per day in a non-deterministic manner [204, 234]. In other words, the registration of all potential domain names in anticipation will cost no less than in current monetary values $100M [233]. Note that, for the botherder is inexpensive to generate a disproportionate number of random domain name, as it only requires a modification of the bot's binary. After all, they have only to register a negligible fraction of them. On the other hand, for the defenders it requires significant time and money [203]. Therefore, the cooperation of DNS registrants in the battle against botnets' controllers is crucial.

#### 5.9.2.2 Botnet Infiltration

A more active approach to investigate the behavior of a botnet, and thus to prevent their actions, is via *infiltration*, through which the defenders join in the botnet's C&C channel [188]. To do so, they utilize an actual malware sample or a client simulating a bot on a controlled environment. As a result, they are able to supervise a client that undertake the role of an active bot and interact with the C&C server, by actively monitoring the bot's network activity [235], including DNS activity. In this way, information about the DNS fluxing mechanism or the malicious instructions of the bothereder could be

revealed. Alongside, the identities of other members of the botnet may be identified with the purpose to enumerate the botnet's population [193]. Actually, it is trivial to acquire a malware sample through honeypots, honey clients, or spam traps. A real example of infiltration strategy is given in works [236, 237], which cope with *MegaD* a botnet with spamming activities.

# Chapter 6

# DNS as an attack vector in Mobile Platforms

In the current chapter we demonstrate how DNS can be used with the aim of undermining the privacy of mobile users. Specifically, in the context of this thesis, we implemented a privacy-invasive mobile app capable of manipulating the DNS service provided by an iOS device. Precisely, this spyware manages to interfere to the tethering service present in Apple's mobile devices with the aim to stealthily redirect all users connected via the the targeted device to a malicious web page. That is, the malware hijacks the DNS service running on the mobile platform and enables the attacker to phish user's credentials while they are trying to access legitimate websites.

## 6.1 Introduction

Perhaps the most important parameter for any mobile application or service is the way it is delivered and experienced by the end-users, who usually, in due course, decide to keep it on their software portfolio or not. Most would agree that security and privacy have both a crucial role to play towards this decision.

Recall from section 5.7, that over the last years smart mobile devices have evolved to small and ubiquitous computing platforms. This way, these devices are able to store a rich set of personal information and at the same time provide powerful services, including location services, Internet sharing via tethering, and intelligent voice assistants

to name just a few. As expected, this situation draws the attention of aggressors who aim at stealing or misusing private information, or even disrupting the information flow. Typical methods to achieve this goal are by gaining root permissions (known as Jailbreak [238] on iOS, or Root on Android platforms [239]), exposing new vulnerabilities [240], and developing some ilk of malware. In fact, every new facility or service offered for modern smartphones may be susceptible to attacks and/or privacy leaks. To this direction, we came with the idea that the abuse of the DNS service offered by all mobile OSs can subvert the owner's privacy.

Under this prism, this chapter sheds light on the relationship amongst DNS service and modern mobile paltforms focusing on iOS-based ones. Specifically, our primary aim is to elaborate on privacy risks that may come with the introduction of new mobile services. This means that services destined, say, to smartphones may expose private information without the user consent. In this direction, we concentrate on two very popular services (a) the Personal Hotspot (PH), which is a very common way of tethering an iPhone Internet connection with other WiFi devices, and (b) Siri, the new intelligent personal voice assistant available since iOS ver. 5.

To unveil user's privacy risks that may stem from the aforementioned services, we implement as a first step a DNS hijacking malware. On the one hand, this malware is capable of manipulate the iPhone tethering service, and hence redirecting all users connected via it to fake websites aiming to phish their credentials while they trying to access some Internet resource. On the other hand, by specifically targeting on the Siri facility, the malware manipulates the DNS service of the device in an effort to expose sensitive user information including its geographical location, account credentials, telephone numbers, etc. As far as we are aware of, this is the first work in the literature to discuss and analyze ways of exposing user privacy by leveraging on such popular mobile services.

## 6.2 Preliminaries

While it is expected a minimum level of familiarization with iOS programming and mobile services by the reader, this section is necessary for reasons of completeness. That is, we discuss mDNS as well as the basic components and functionality of both the tethering and Siri services as an essential prelude to the following sections.

### 6.2.1 mDNS

The goal of the proposed malware is to compromise the DNS service running on a mobile device, say, smartphone. In fact, this is a sine qua non for the attacks described further down to be successful. Toward this direction, one of the main technologies used in iOS for networking is *Bonjour*. Bonjour enables a device to allocate an IP address and advertise a service to other computers or devices plugged into the same TCP/IP network. In addition, Bonjour includes service discovery, address assignment, and name resolution. On top of that, Bonjour, being a Zero Configuration Networking (ZCN) facility, needs to be able to translate name-to-address even without the presence of a RDNS. To meet this requirement the Multicast DNS (mDNS) [241] protocol is used. This protocol uses the same packet format, domain name structure, and DNS RR types as unicast (standard) DNS. However, two main differences apply. The first one is that mDNS queries are sent to all local hosts using multicast routing opposed to DNS protocol, which queries are sent to a specific, preconfigured RDNS. The second is that mDNS listens on UDP port 5353, opposite to DNS which listens on standard UDP port 53. Also note that mDNS queries use the multicast address 224.0.0.251. In case a device triggers the Bonjour service, it listens to the multicast requests and if it knows the answer, it replies to this multicast address. mDNSResponder is the application which is responsible for handling Bonjour on Mac OS X and iOS devices and for listening for services out of the box.

Furthermore, iOS supports a *hosts file* configuration so as to be able to correlate already visited hostnames with IP addresses prior to consulting DNS. This temporary mapping per hostname is kept in the /etc/hosts, which is also manipulated by our malware as described further down in section 6.3.1. Last but not least, iOS holds in the *Network.identification.plist* the settings of all the wireless networks with which the mobile device has been associated sometime in the past. This happens as part of a new feature that allows the iOS device to remember the network settings and automatically connect to a network, using the same settings, without user intervention. Therefore, our malware needs to replace the DNS IP address of all networks logged in the Network.identification.plist with a forged one (where our RDNS resides) and to restart the mDNS service in order the new settings to take effect. This situation is discussed in detail in section 6.3.1.

### 6.2.2 The Tethering and Siri services

Recall from section 6.1 that the main purpose of the described attack is to compromise the privacy of the end-user by exploiting DNS service and capitalizing on two popular services; Tethering and Siri. Tethering is a network service which gives the device's owner the ability to share their mobile phone cellular data connection with other devices (users). This sharing can be offered over a WiFi, Bluetooth, or by a physical connection via a cable. Currently, Tethering incorporates a software functionality known as Personal Hotspot (PH). The PH service is accounted for transforming the device into a wireless Access Point (AP), so that iPhone users are able to share their 4G connection. Once the PH starts up, the device selects the first empty 802.11 wireless channel to emit the signal using the device name as the Extended Service Set ID (ESSID) name for the AP. From this point on, PH can support and share the Internet connection simultaneously with up to five devices. The PH service operates by default in the WPA2 Pre-shared key (PSK) mode.

Siri, on the other hand, is one of the highlights of iOS ver. 5 only provided for the iPhone 4S. It is a personal intelligent software assistant that uses a natural language interface to interact with the user and execute their requests expressed in voice commands (note that the corresponding service for the android platform is called Google now). Siri is able to carry out a variety of tasks (e.g. send SMS, e-mail, arrange meetings, make questions about the weather, points of interest, etc.). To accomplish such tasks, Siri communicates securely via https with a remote server residing at `https://guzzoni.apple.com:443`. This server is responsible to translate user's voice requests into text commands, and text commands into actions. To fulfill a task, Siri can exchange a variety of data with the Guzzoni server, such as raw audio data, plist files, confidence score of each word in a sentence, timestamps, location information, and more importantly, information derived directly from the device's local databases (calendar, contacts, etc.).

French mobile development company Applidium [242] has recently reverse engineered Siri protocol. They also provided the first evidence about its structure as well as the open source tools they used. For using Siri, the device must firstly authenticate the Siri server. This is done during the standard SSL handshake as the server certificate, namely `guzzoni.apple.com`, is preinstalled on every iPhone 4S device. Note that the authentication is unilateral, that is, the client (device) does not authenticate itself to

the service by means of a certificate. Upon successful authentication and under the protection of the SSL tunnel, Siri sends four keys to the Guzzoni server, namely {x-ace-host, assistantID, speechID, validationData}, where *x-ace-host* is a unique identifier generated by Siri on the device and updated every two weeks; *assistantID* is a string containing information about the user, which is generated by Siri on the device at every use; *speechID* is a speech identifier, generated by Siri on the device on-the-fly at every use; *validationData* is a string that gets generated every 24 hours on the device via FairPlayed. By using this quadruple of keys, the Guzzoni server is able to authenticate the device.

From the above discussion it becomes clear that attacking Siri service is not straightforward. Specifically, as already pointed out, Siri is a proprietary software designed to securely communicate (https) with the legitimate Siri server(s) controlled by Apple. Therefore, to fool the protocol, one has to somehow hijack the device-to-Siri legitimate server communication in an undetectable manner. In this direction, as analyzed in [243], a solution is to create a fake SSL Certification Authority (CA) and inject it into the device in order to replace the original one. This is necessary to create and sign a fake certificate for `guzzoni.apple.com`. After that, by using a VPN connection, Applidium managed to redirect all iPhone packets, through a custom DNS NS for further analysis. A few weeks later, P. Lamonica [244] created an open-source server, namely SiriProxy, having the ability to capture and manipulate Siri packets. Also, through the creation of customized plugins he was able to execute certain actions (for instance control a thermostat over Siri).

## 6.3 Implementation

In this section, we delve into the internal workings of the malware responsible for poisoning the DNS service running on the device. Keep in mind that, this is a first step towards executing the two attack scenarios described further down in section 6.4.

### 6.3.1 The DNS poisoning malware

To manipulate the mDNS service provided by iOS we implement a malware (or more precisely a spyware) which, as we show in what follows, acts as a rootkit. This malware

is written in Objective-C and compiled for iPhone ARM CPU using *Theos* [245]. It is tested to run on iOS ver. 5 and above. Also, it has been built using the unofficial ways for backgrounding (daemons and dylibs), the public and private frameworks for developing iOS applications, and the *MobileSubstrate* framework [246] with the *substrate.h* header that overrides iOS internal functions. This is why certain modules of our malware can be classified as rootkit and more specifically as a DNS poisoning one. The malware infects the mDNS protocol, thus making possible the execution of a man-in-the-middle attack at a later time depending on the attack scenario. Namely, to take over the control of the Siri service upon its activation by the user, or if tethering is in use, redirect any connected device to websites controlled by the attacker.

As depicted in Fig. 6.1, the core of the malware consists of a main daemon combined with a proper launch plist (activated at device boot time) and six subroutines written as Objective-C functions and dylibs. The daemon is responsible for managing all subroutines, namely SirInvervine, HUpdate, NIUpdate, mDNSReloader, NetDetector, and PHDetector, which in turn carry out the malware tasks. In the following, we elaborate on the functionality of each subroutine.

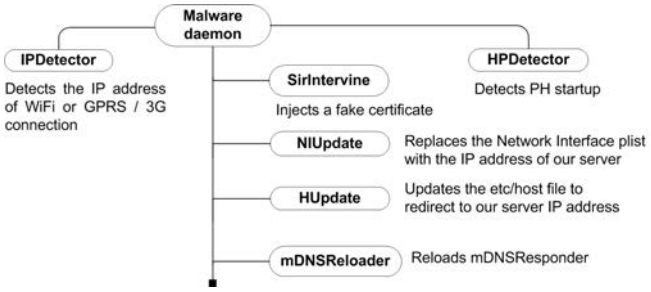



FIGURE 6.1: Malware module

Recall from section 6.2.2 that for utilizing Siri, the smartphone must first authenticate the Siri server. This is done in a unilateral fashion, i.e., the client (device) does not authenticate itself to the service. So, to act as man-in-the-middle and hijack the https session, one needs to replace the original Siri certificate stored in the device with a fake one. This is accomplished by *SirIntervine*. Upon execution, this routine installs a custom SSL CA into iOS and at the same time adds into the com.apple.assistant.plist file, which is a Siri setting file, the domain name of our man-in-the-middle server (in this case spe.samos.icsd.gr). This is needed to create and sign a bogus certificate for `guzzoni.apple.com`.

The *HUpdate* routine is responsible for poisoning the device's /etc/hosts file. Although the pieces of data stored in this file are mostly used when a RDNS is not available in the local network, as explained in section 6.2.1, this file is always queried upon Bonjour activation. Once we gain root permissions, the /etc/hosts file becomes vulnerable as it is stored in plaintext. Specifically, for our attack scenarios, HUpdate inserts two hostname records into the /etc/hosts file, which correspond to the IP-address of our man-in-the-middle server. The two hostnames that are poisoned are "`guzzoni.apple.com`" and "`facebook.com`". In this way, all packets corresponding to the aforementioned domain names will eventually be sent to man-in-the-middle entity controlled by us. Nevertheless, while the first hostname is essential for accomplishing the second attack involving Siri application, we can poison whatever domain name we desire. For our case, we choose the Facebook user login page as it is the most prevalent social network and in this way an aggressor can extract profitable sensitive information about their victim. HUpdate adds the poisoned hostnames in the /etc/hosts file using the public NSFileManager class (only if not poisoned already). Figure 6.2 depicts a snapshot of the /etc/hosts file after a successful poisoning attempt. Note that the two last entries correspond to our man-in-the-middle server.

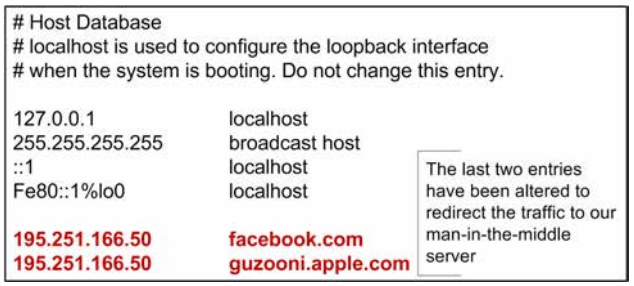



FIGURE 6.2: The /etc/hosts file after poisoning

*NIUpdate* is the subroutine responsible for poisoning the IP address of any RDNS found in the Network Identification file, with a malicious one under the attacker's control. Every time an iPhone device connects to a WiFi or a 3G/4G network, an entry is created in the Network.identification.plist file containing all settings specific to this network, namely, router's IP address, subnet mask, local RDNS IP address, MAC address, etc. Hence, every time the device attempts to connect to a known network, it will load the settings used during the previous session. Once NIUpdate is activated, it changes all the predefined RDNSs' IP addresses with the one of our man-in-the-middle server. Once

again, the Network.identification.plist file is stored in plaintext (plist), thus it can be easily falsified using the NSMutalbeDictionary class.

*mDNSReloader* is a dylib responsible for shutting down or restarting the mDNSResponder service (deamon) running on the device aiming to activate new network settings. Specifically, by disabling the mDNSResponder service one also terminates the unicast DNS resolution. By doing so, the mDNS service is stopped, meaning that instantly the device cannot resolve hostnames. Once the service gets restarted, the mDNSResponder will parse the /etc/hosts and Network.identification.plist files in an effort to use the default settings before obtaining new ones. Note that mDNSReloader enables or disables the service by simply modifying the "ProgramArguments" settings (in the com.apple.mDNSResonder.plist file), which is responsible for the activation of the service into "Yes" or "No". Figure 6.3 depicts a snapshot of the source-code responsible for this modification.

```
NSMutableDictionary *mDNSR = [NSMutableDictionary dictionaryWithContentsOfFile:
        @"/System/Library/LaunchDaemons/com.apple.mDNSResponder.plist"];
...
if (mDNSR) {
  ...
  [mDNSR setObject:args forKey:@"ProgramArguments"];
  [mDNSR writeToFile:@
        "/System/Library/LaunchDaemons/com.apple.mDNSResponder.plist" atomically:YES];
}
```

FIGURE 6.3: Source code snippet for disabling/enabling mDNSResponder

Both *NetDetector* and *PHDetector* are dylibs triggered directly from the iOS Notification Center (more specifically the CFNotificationCenterGetDarwinNotifyCenter) every time the device connects to any wireless or mobile network interface, e.g., WiFi, GPRS, 4G, or after PH activation. As soon as one of these dylibs is executed, it will re-run all the aforementioned subroutines to update the network settings for the device.

Lastly, our man-in-the-middle server incorporates three basic modules:
(a) A typical RDNS that provides fabricated answers for every domain name that is queried for. Specifically, for the first scenario, this is the domain name corresponding to the Siri legitimate server, while for the second, a bogus version of the Facebook user login page.
(b) The open source SiriProxy Ruby script [244] which allows us to manipulate Siri packets and create our own custom plugins to violate user's privacy though the Siri

technology.

(c) An http server used during the first attack scenario.

The server runs on a typical laptop machine which incorporates a 2.53 GHz Intel Core 2 Duo T7200 CPU and 4 GB of RAM. The OS of this machine is OS X Leopard Snow. The lightweight open source DNS Server named *Dnsmasq* has been used to provide DNS service. We also tinkered with the pre-alpha version of the SiriProxy that runs on our server to handle (decipher, encipher, modify) Siri packets.

Both Dnsmasq and SiriProxy server, which is the main software employed for realizing man-in-the-middle and handling Siri packets, are by design able to accommodate multiple users.

## 6.4 Attack Scenarios

In the current section, it is demonstrated how the aggressor is capable of collecting private information, while the users utilize Tethering or interact with Siri. We analyze these two attacks scenarios in detail and show that any private information the user provides for the benefit of both of these services (e.g., passwords, account numbers, telephone numbers, emails, user's location, etc.) is at stake. A high-level representation of the attack architecture is given in Fig. 6.4. It is stressed that all experiments had 100% accuracy in logging private and sensitive information without exposing any malicious behavior to the user of the device.

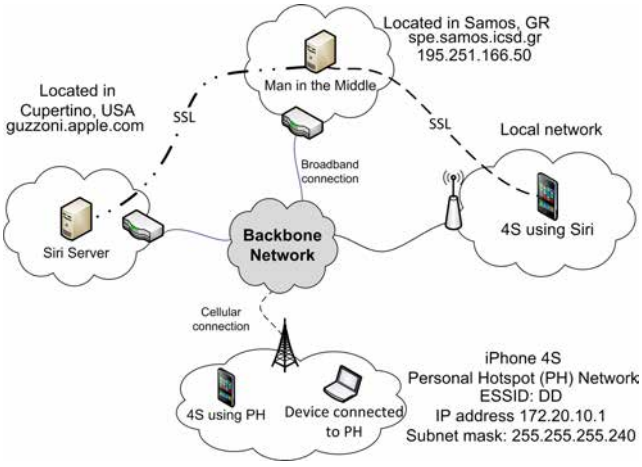



FIGURE 6.4: Network architecture utilized for the attack scenarios

### 6.4.1 Scenario I: DNS Hijacking

According to the first scenario, we utilize an already infected with our spyware iPhone 4S for tethering its data connection, and therefore enable it to act as an IEEE 802.11 hotspot. This situation is illustrated in the lower part (cloud) of Fig. 6.4. From this time forth, the device behaves as a Wi-Fi router meaning that any Wi-Fi capable device (for instance the laptop in Fig. 6.4) will be able to connect via the iPhone PH service to Internet. Once a device gets connected, it will allocate an IP address in the range of 172.20.10.2 to 172.20.10.14 using the Dynamic Host Configuration Protocol (DHCP). Afterwards, all network packets will be routed via the smartphone behaving as PH. One of the main iPhone tasks when acting as a PH is to translate any hostname into a valid IP address. To do so, firstly it lookups into the /etc/hosts file and if it does not find the answer, it will query the corresponding RDNS. However, the device is infected with the malware and both the /etc/hosts file and the local RDNS IP address have been falsified to contain the IP address of our man-in-the-middle server. This means that all traffic generated by the users connected via the PH will be redirected to a server under the attacker's control. To demonstrate the perilous effects of this attack, we have install on our server a webpage that appears exactly the same as that of Facebook's user login page. We chose Facebook as it is the most popular social network and many people check their profiles as soon as they connect to Internet. In fact, the only functionality of our fake webpage is to log into a MySQL database the credentials of the user in plaintext, once they try to login into the site. As soon as the credentials are stored, the fake website returns a message that the page is temporally unavailable due to heavy loads or other innocuous reason.

### 6.4.2 Scenario II: Privacy leak over Siri

The second attack scenario capitalizes on the Siri service. Once more, the malware compromises the mDNS protocol with the purpose to redirect all (or selected) Internet traffic to our man-in-the-middle server. In this way, we accomplish to place a malicious entity between the device and the legitimate Siri server controlled by Apple. After that, we are able to intercept user's private information transferred over Siri. At present, this is realized through the implementation of three custom plugins for SiriProxy [244]. To further exemplify this situation, Fig. 6.5 exhibits the basic message flow taking place

among the Siri service running on the mobile device and the legitimate server, but with our server placed in the middle.

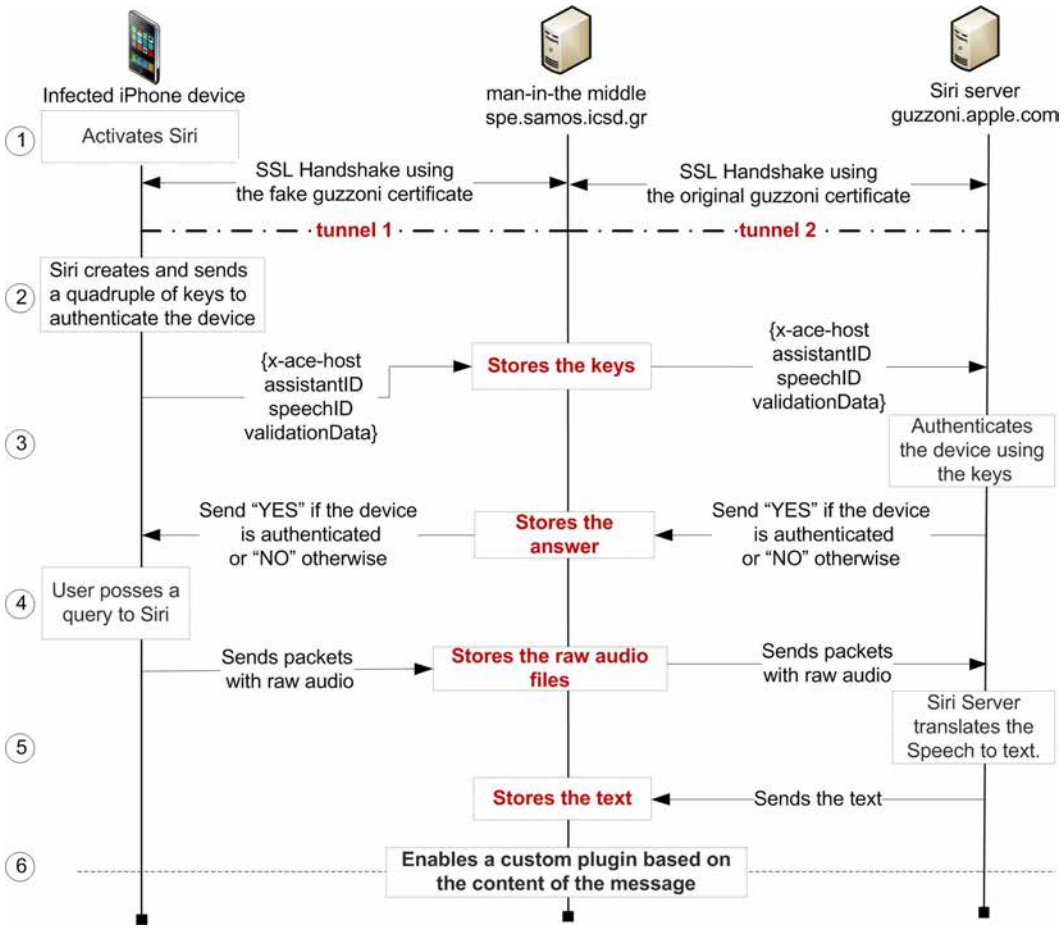



FIGURE 6.5: Siri protocol flow

Upon Siri activation (1), an SSL handshake between Siri and our man-in-the-middle server is performed, and at the same time a second handshake is conducted between the man-in-the-middle server and the Apple's legitimate Siri server. Recall, that SiriProxy runs on a server under attacker's control with the aim of handling (decipher, encipher, modify) Siri packets. Specifically, to initiate the handshake, Siri sends a "Hello" message which is redirected to man-in-the-middle server and forwarded to the original one. The Siri server replies and sends over its authentic server certificate containing its public key. The man-in-the-middle entity transmits to the device its bogus server certificate containing the corresponding (fake) public key. This certificate has been created from the same CA authority with the one been injected to the device when infected by the malware. Once Siri verifies the fake certificate and subsequently authenticates our fake server, it sends a premaster secret (premaster secret 1) to our server encrypted with

the corresponding fake public key. At this moment, both sides (Siri and man-in-the-middle) calculate a session key (session key 1) and establish an SSL session (tunnel 1). After that, our server acting as Siri client sends a second premaster secret to Siri server encrypted with the original server's public key. As a result, the man-in-the-middle entity and the Siri server calculate another session key (session key 2) and establish a second SSL session (tunnel 2).

Under the protection of tunnel 1, as explained in section 6.2.2, Siri generates and sends the quadruple of keys necessary to authenticate the device with the server (2). Our server receives the keys and forwards them to the legitimate Siri server but this time through tunnel 2. Upon reception, Siri server will check if the received keys correspond to a legitimate iPhone 4S and if true, it will answer with "YES" (else "NO") (3). Assuming a positive answer, Siri is ready to listen to user commands (4) & (5). Otherwise, it will respond with a "Siri server unavailable" message. From this point on, the user is able to make a request by talking to the service. Siri records the voice containing the user's command (or an answer to a question posed by Siri during a transaction), converts it into raw audio files and sends them to legitimate Siri server. The server translates the audio file to text and sends back the translated text which is eventually passed to the user by synthetic speech. It is therefore obvious that every personal information being transmitted by the user it becomes available to the man-in-the-middle entity as well.

To further analyze this situation, we implemented three custom SiriProxy plugins specially crafted to expose usual private information. This means, that once the man-in-the-middle server receives a Siri message from the device it will try to match its context with one of this plugins. Figure 6.6 depicts a characteristic example of such a plugin that is activated once the translated string coming from the user side is "iPhone privacy". Upon activation, our server will respond with the string "Siri is having some privacy leaks!" to the Siri service. Siri will complete the request by displaying the message on device's screen and at the same time by pronouncing it. The next three sections describe in detail how we were able to intercept valuable user private information through the employment of such plugins.

```
listen_for /iPhone privacy/i do                  # checks the string
   say "Siri is having some privacy leaks!" # sends a custom answer
   request_completed                             # completes the request
end
```

FIGURE 6.6: Basic source code example of a custom plugin

#### 6.4.2.1 Exposing User's Geographical Location

Deploying the first plugin, we are able to successfully retrieve user's location in the form of GPS coordinates. This happened after the user asked Siri about the weather, e.g., "How is the weather today?". Note that with minor modifications, the same plugin is able to acquire user's location for any posed question such as "How can I get to Ocean Park?", "Where is the nearest metro station and bus stop?", etc. It is stressed, that Siri obtains the geographic coordinates without directly asking the user about their location. This happens because Siri has access to the device's location services by default (assuming that the user has not changed the default settings; otherwise the user will be alerted to enable GPS). Figure 6.7 depicts a snippet of the plugin source code responsible to retrieve the geographical location of a user. Once the Siri server asks Siri about the location of the device, the plugin is activated and waits for the standard value (header) "SetRequestOrigin" to obtain the exact user's location.

```
filter "SetRequestOrigin", direction: :from_iphone do |object|  # filters the packet for location info
   puts "[Info - User Location]                                  # prints the location on terminal screen
         lat: #{object["properties"]["latitude"]},               # on the side of the man-in-the-middle
         long: #{object["properties"]["longitude"]}"
   return 1                                                      #forwards the object
end
```

FIGURE 6.7: Snippet of the plugin responsible to retrieve user's location

#### 6.4.2.2 Obtaining sensitive information via SMS

The second plugin capitalizes on Short Message Service (SMS). According to this scenario, the user sends an SMS by just speaking to Siri. The plugin intercepts the telephone number of the message's receiver, the SMS payload, and the final outcome, i.e., whether the end-user finally gave their consent to send the SMS or not. By this use case scenario, it is clear that a variety of private information sent to Apple's servers can be exposed to an intruder without the user be aware of it. Figure 6.8 illustrates the log file created

by the corresponding plugin on our man-in-the-middle entity under this scenario. In the same figure, we can easily identify the user's private information leaked out ((a), (b), (c) & (d)). Bear in mind that the entries starting with [*Info-iPhone*] correspond to messages sent from Siri, while those starting with [*Info-Guzzoni*] to messages deriving from the Siri legitimate server. Also, messages being transmitted from SiriProxy are marked with [*Info–Plugin Manager*]. For emphasis, each privacy leak is placed within a gray frame.

To exemplify this, once the user activates Siri and starts interacting with it, Siri sends user voice towards the server in many fragmented packets. After the user stops speaking, Siri transmits a flag message (1). Then, Siri translates the voice into text and sends it back to our server (2). Upon reception, SiriProxy tries to match the translated text with a custom plugin (3). The plugin is in charge to log the translated text when a user tries to send an SMS (4). Once the text is logged, the message is sent to Siri. As a final step, the Siri legitimate server sends a message to inform Siri to create a graphical view for presenting the translated text (5).

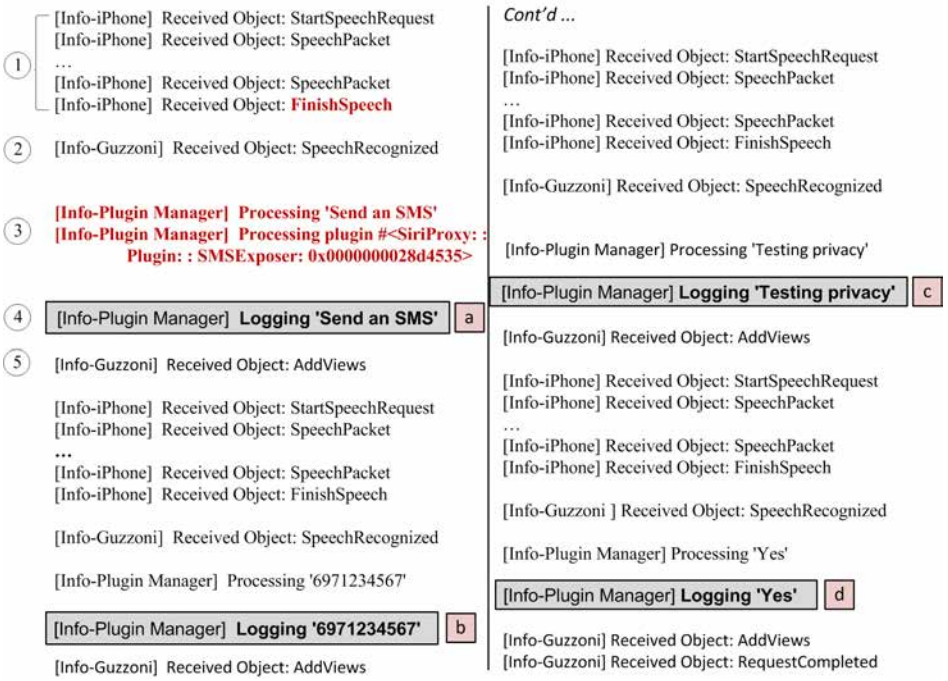



FIGURE 6.8: Log file created when sending an SMS

#### 6.4.2.3 Acquiring user's password

One of Siri highlights is that the user can be engaged in a form of conversational dialog with the assistant using any of the available input and output mechanisms, e.g., speech, graphical user interfaces, text entry, and so on. So, for the last use case, we developed a smarter plugin able not only to eavesdrop on private information, but also interact with the user and ask them custom questions. By doing so, it becomes very likely for our man-in-the-middle entity to intercept confidential information such as the user's e-mail address or even the password of their e-mail account(s). Due to the fact, that Siri uses artificial intelligent to interact with the user in order to accomplish a task, e.g., send out an email, the question about the password would not bear any evidence of malicious behavior.

Figure 6.9 illustrates the message flow, when the user attempts to send an e-mail using Siri. This results to the activation of the corresponding plugin residing on the man-in-the-middle entity (1). Once SiriProxy receives the translated text from the original Siri server - in this case "Send an email"- it will match it against the plugin settings (2). As a consequence, the plugin will temporally block the original text message from being transmitted towards the original Siri server, and instead, it will send back a custom question to Siri asking the user which sender's email address it should use. Since the e-mail address is generally considered public information the user is highly probable to reply providing its email address to Siri (3). As a next step, the plugin shall force Siri to pose a second question to the user. This time, Siri will ask for the password of the e-mail address provided in the previous step (5). Typically, a naive user will trust Siri and think that the password is required for the e-mail to be sent. Hence, they will respond with the password, thus enabling the plugin to log it in cleartext (6).

## 6.5 Related Work

Smart mobile devices are used in everyday life to store a variety of users' sensitive information. So, it should come as no surprise that they attract the attention of resourceful attackers. In this context, over the last few years, traditional malware also seem to evolve in an effort to catch up with the so called mobile era. Mainly, such malware affect the most popular OS, namely Android and iOS. It is also relevant to note that according

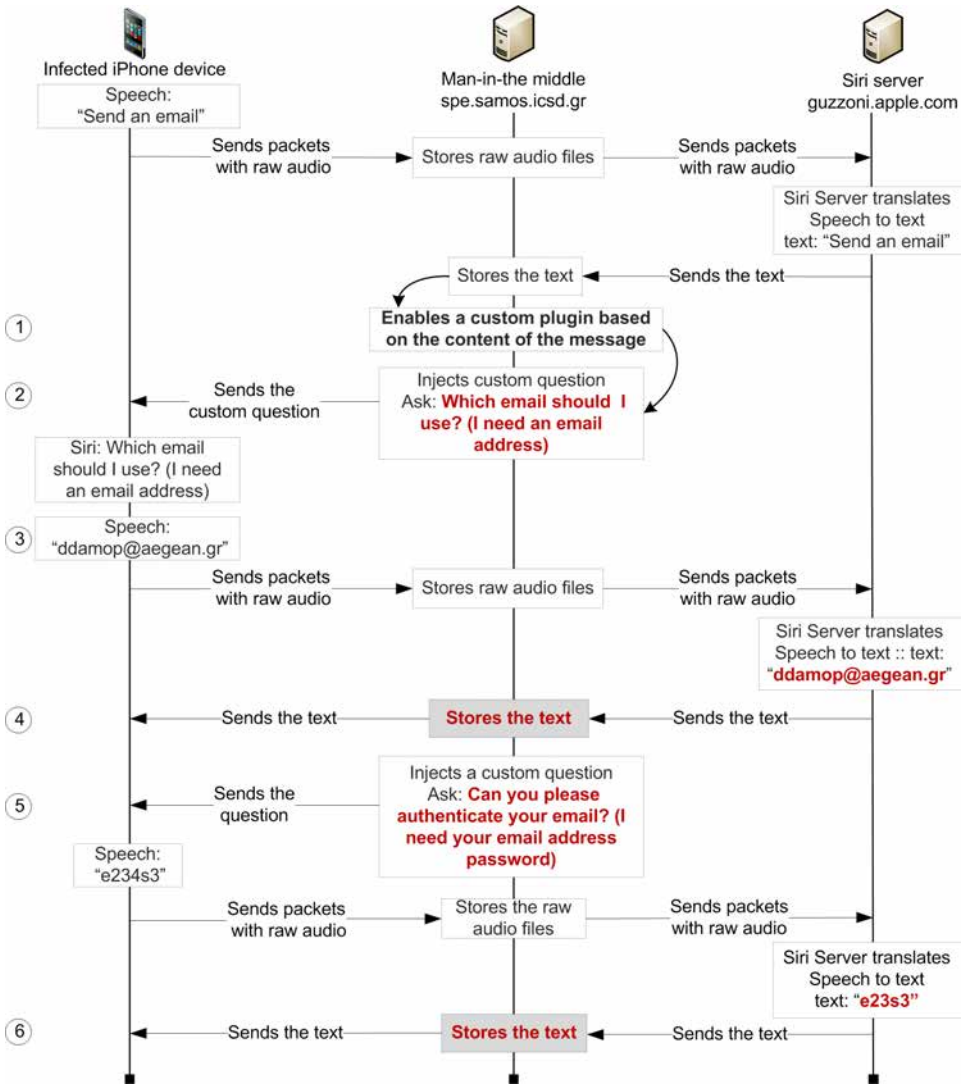



FIGURE 6.9: Message flow for acquiring user's password

to the work in[226] the most common goal for propagating malicious applications on smartphones is the collection of private information. Unfortunately, up to this moment the literature in regards to the privacy level of advanced mobile services, including Siri and Tethering, is scarce. Therefore, this section attempts to review the existing literature in chronological order and with respect to the impact of this particular threat. As a result, it focuses solely on mobile malware that aim to jeopardize the privacy of the device's owner. Recall that a literature review relevant to mobile malware having C&C capabilities and botnet involvement is given in section 5.7.

As discussed in section 5.7, the first appearance of mobile malware was that of *Cabir* worm destined to Symbian OS in June 2004. However, *Pbstealer* a trojan spy based on

Cabir was the first malware that threaten the privacy of mobile device owners. Specifically, once Pbstealer infects a device searches for its address book and sends the contained data to the first device discovered in range [217]. Following the introduction of Pbstealer, another trojan that violates user privacy has been detected. *StealWar* presents similar functionality to Pbstealer, but it has the ability to propagate itself via MMS as well [217]. *Flexispy* is another trojan that takes over control of smartphones running Symbian OS and sends call information and SMS data to the herder of the Trojan [217]. A different malware is described in [247] with the functionality of taking pictures from the phone's camera without the user's knowledge and send it to a predefined phone number via MMS. The creators of the particular malware aimed to demonstrate that it is quite easy to implement a privacy violation attack for mobile devices.

In 2008, two modern OS platforms for smartphones, Apple's iOS and Google's Android have been introduced. Since then, the research community focused its attention on the security level of these OS. For the case of Android, the beginning of privacy concerns arises with the first Android commercial smartphone, namely G1 which was shipped with the initial version of this OS. This version was accompanied with vulnerabilities in the web browser application. By exploiting these vulnerabilities, a determined attacker could gain access to any data that the browser stores, including cookies, text typed in form fields or even saved user passwords [217]. In the same year, serious privacy concerns appeared with several applications within the AppStore market administered by Apple [248].

Regarding iOS, *Privacy.A* was the first appeared worm, which was running in stealth mode and aimed to steal personal data from iPhone devices. In Nov. 2009, a highly perilous version of *Ikee*, namely iKee.B appeared [214]. In addition to the C&C functionality described in section 5.7, iKee.B had the ability to poison iPhone's DNS cache (/etc/hosts) in order to redirect specific domain name requests to a given (malicious) RDNS.

On the Android camp, *LeNa*, a descendant of Legacy Native virus family, was hiding in seemingly useful apps to gain the required root privilege and expose user private information [217]. Moreover, two well-known applications, namely *MobileSpy* and *MobileSteath* [249], exhibit a similar to previously approach with the aim of sending the captured

data (e.g., SMS, call history, GPS coordinates, contact details, video and pictures) to a remote server.

In 2010, Seriot [248] presented some interesting attack scenarios on how a malicious application can use official and public frameworks, provided by Apple, to extract private data, including contacts, email account setting, keyboard cache entries, Mobile Safari searches, and the most recent GPS location of the device. This can occur without the user's knowledge and without the malicious app being rejected by the AppStore review. Also, five Android malware families, namely *BaseBrigde*, *JiFake*, *DroidKungFu*, *Hongtoutou*, *Geinimi*, have been identified after trying to expose the victim's geographic location, International Mobile Subscriber Identity (IMSI), bookmarks, and/or place calls, send SMS to premium phone numbers, and so forth [217].

# Chapter 7

# Conclusions and Future Directions

Reaching to the end of this thesis, the reader should have perceived until now that although DNS protocol constitutes the pillar of the modern Internet infrastructure, it lacks a full-fledged protection mechanism, and thus it is frequently abused in several and diverse ways. Throughout our research, we provide evidence that DNS service can be exploited as a multipurpose instrument in order to violate the integrity, authenticity, confidentiality and availability of the offered resources in Internet. While some facets of the DNS attack surface are already well-known to the community, the thesis at hand sheds light to novel ones that are not so obvious but are equally dangerous.

As discussed in chapters 3 and 6, DNS itself suffers from certain vulnerabilities which threaten the integrity and authenticity of DNS data, and hence may be exploited with the aim of jeopardizing the privacy of end-users. Moreover, DNS can facilitate the efficient execution of DDoS attacks or can be abused for the coordination of botnets, as analyzed in chapters 4 and 5 respectively. Both of these situations endanger the availability of Internet services. Undoubtedly, the race between aggressors and defenders is continuous and intensive, but it seems in many cases the aggressors manage to stay one step ahead. So, we can safely argue that many more have to be done toward securing DNS service and with that ensure the smooth operation of Internet as a whole. This chapter summarizes the overall contribution of the current PhD thesis, while at the same time provides research directions for future work.

## 7.1 Thesis Contributions

In a nutshell, and in accordance to the objectives given in chapter 1, this PhD thesis elaborates on the different ways the DNS service can be exploited as a multipurpose vector for fulfilling the nefarious desires of malevolent entities in Internet. Table 7.1 offers a global map of the PhD accomplishments with respect to contributions in literature for quick reference.

Initially, we examined the various types of DNS cache poisoning attack as well as the possible countermeasures. To this direction, we conduct a side-by-side comparison between DNSSEC and DNSCurve, the two prevailing cryptographic mechanisms, that aim to protect, among others, the integrity and authenticity of DNS data. This way, the defenders are in position to infer which defensive mechanism best suits to their needs. Chapter 3 and [17] presents this side-by-side comparison.

Based on the outcomes of the aforementioned comparison, we further investigated the role of DNSSEC, this time not concerning the cache poisoning attack but rather its involvement in DNS amplification attacks. Specifically, by capitalizing on the fact that DNSSEC-related RR are considerably sizeable, we underpinned that the facilitation of advanced types of DNSSEC-powered amplification attacks is absolutely feasible. Precisely, we studied the ways a competent and smart attacker can orchestrate and launch wide-scale DDoS attacks, which only utilize DNSSEC-related RR. As a side contribution, the performance of (open) DNS forwarders as reflectors was investigated. To do so, we conducted a detailed evaluation of the impact of this type of attack by planning and executing a large-scale DDoS attack involving a great number of (open) DNS forwarders located in three European countries. The main advantage of our research compared with the standard type of the attack as described in the literature thus far, is that we demonstrated that even a naive attacker is capable of accomplishing the attack by exploiting publicly available resources existing in the Internet without the need to acquire some of their own. To the best of our knowledge, our research on this particular issue constitutes the first comprehensive study of DNS amplification attack involving DNSSEC-related RRs. Further details of this work is provided in section 4.2 and [18].

Moreover, this thesis answers the question of what already available resources in terms of Internet entities can be particular fruitful for a potential attacker to incorporate in

their arsenal. Namely, we scrutinized the potential of taking advantage the ANSs of TLD as both amplifiers and reflectors. Therefore, from a contribution point of view, this is also the first work in the literature to assess the involvement of the upper DNS hierarchy ANSs in DNS amplification incidents. More precisely, we measured for all ANSs (including root NSs) the size of their response for DNSSEC-related RR, that is, we evaluated the amplification factor these servers may produce. Furthermore, we estimated the current degree of adoption of RRL mechanism by ANSs, which constitutes the primary defensive barrier against their involvement in amplification attacks. The outcomes of our research exhibit that a worryingly large number of ANSs out there can be entangled involuntarily by DDoS attackers. This important contribution is given in chapter 4 and [19].

Following, we dealt with botnet issue. Throughout our research, we witnessed concrete evidences that DNS is constantly abused by botherders for the sake of concealing their actions. In our effort to draw the attention of the research community on this topic, we proposed and evaluated novel botnet architectures. More specifically, the proposed architectures are either solely rely on mobile bots or a mixed infrastructure, namely include both mobile and desktop agents. In any case however, all the proposed architectures exploit DNS as C&C channel for coordinating the botnet and disseminating botmaster's commands. Chapter 5 and [20] elaborate on these contributions.

Finally, in addition to legacy DNS attacks, we considered the potential of DNS as an attack vector to invade user's privacy, and specifically to harvest private sensitive information from smartphone owners. Especially, we designed and implemented a privacy-invasive mobile app able to manipulate the DNS service provided by an iOS device. This app is capable of acting as a man-in-the-middle to the tethering service present in Apple's mobile devices with the aim to redirect all users connected via the targeted device to, say, malicious sites according to attacker's goal. In this way, the implemented malware hijacks the DNS service and enables the attacker to phish user's credentials while they are trying to access legitimate websites. Additionally, by targeting on the popular Siri personal assistant, the spyware exposes sensitive user information including their geographical location, account credentials, telephone numbers, sent/received messages, etc. Actually, by using this case study, we emphasize on the ways DNS forging attempts can by especially beneficial to attackers targeting the continuously growing population of mobile users. This way, we offered the first to our knowledge work in the literature

aimed at examining DNS as an attack vector against popular mobile platforms. This contribution is further analyzed in chapter 6 and [21, 22].

| Objective | Chapter | Contribution | Publication |
|---|---|---|---|
| Obj. 1 | 3 | A side-by-side comparison between DNSSEC and DNSCurve | [17] |
| Obj. 1 | 6 | Mobile spyware that manipulates the DNS service in order to steal private information | [22, 21] |
| Obj. 2 | 4 | DNSSEC-powered amplification attack | [18] |
| Obj. 2 | 4 | Assessment of upper DNS hierarchy infrastructure as amplifier and reflector | [19] |
| Obj. 3 | 5 | Novel mobile botnet architectures that exploit DNS for the coordination of the botnet | [20] |

TABLE 7.1: Overall PhD Thesis Contribution

## 7.2 Future Directions

The PhD thesis at hand generally resides on the offensive side of cyber-security. It particularly contributes to the topic of DNS security and its main goal is to demonstrate that indeed DNS can be used as a multipurpose attack vector against offered resources in Internet. This way, our intention is to offer concrete proofs that many more need to be done toward to safeguarding this important Internet service. Since a number of DNS's aspects have still left unexplored, it becomes obvious that further research should be conducted. In the following, we elaborate on these possible future research directions and categorize them according to the security service they provide.

- *Origin Authentication:* Currently, even with the presence of DNSSEC, the end-user is not directly benefited from its protection mechanism. At least, up to now, this is due to clients' inability of performing strong cryptographic operations, because of computational constraints. This way, the clients are unable to validate the authenticity of the signed DNS RRs. Consequently, a DNS stub-resolver library that will be capable of evaluating the trustworthiness of DNS answers without solely depending on a single RDNS will be highly appreciated.

- *Integrity:* Although DNSSEC is been introduced almost a decade ago, it is still in its infancy as concern its adaptation level. In fact, albeit the majority of TLDs are DNSSEC-signed, only a small fraction (nearly 3%) of 2-TLDs are signed. Furthermore, no more than 15% of the users around the world request DNSSEC-related RR so as to benefit from the protection of DNSSEC mechanism [102]. As a result,

further investigations pertaining to the operational status and the reasons that hamper the wider adoption of DNSSEC are required.

- *Confidentiality:* Alongside with the preceding direction, a mechanism that will also protect the confidentiality of the transmitted DNS messages, besides integrity and authenticity, is deemed essential. Recently, concerns about this aspect of DNS security have been expressed [250], however without proposing any mitigation mechanism. Nonetheless, such a proposal should utilize ECC cryptography, which is characterized by the high speed of the encryption and decryption process, and hence could speed up the resolution process. Furthermore, such mechanism should secure all the diverse aspects of a DNS transaction, including the communication link between the RDNS and the end-user. This way, the end-users would obtain secure, reliable and uninterrupted DNS services.

- *Availability:* As concerns the facilitation of DNS amplification attacks, a thorough examination of the possible ways DNS can be exploited is highly appreciable. In the current thesis, we unveil two, previously uninvestigated, aspects that a potential aggressor can take advantage of with the purpose of amplifying and reflecting their attack traffic, namely DNSSEC-related RR and open forwarders respectively. Furthermore, the extent of the amplification factor provided by the upper DNS hierarchy was also examined. Nevertheless, it is expected that several more unexplored or semi-explored features of DNS can accommodate DDoS incidents. Such a research will aid the defenders to diminish the attack surface of the DNS service and infrastructure to the minimum possible, and hence reduce the implications of such events.

  Continuing from the previous point, the botnet issue is certain to be of high interest to the research community. That is, considering that botnets are on the rise, the need for providing a deeper understanding of their different facets and ways they construct their C&C channels is without doubt an urgent need. In the context of this PhD thesis, we focused on mobile and/or mixed bot agents. Apparently, many more architectures can be exploited. For instance, future researches may concentrate on additional covert channels for bot communication. It can be argued that the tendency is toward well-established, text-based network protocols. As an alternative, one may consider the coupling of Internet anonymization facilities and DNS protocol for completely hiding the C&C infrastructure inside the Tor overlay.

To do so, a *Tor Fluxing* approach for locating in a stealth and undetectable manner the C&C server can be followed.

# Bibliography


[1] Paul Mockapetris. RFC 1034: Domain names-concepts and facilities, 1987. URL http://www.ietf.org/rfc/rfc1034.txt.

[2] Paul Mockapetris. RFC 1035: Domain Names - Implementation and Specification, 1987. URL https://www.ietf.org/rfc/rfc1035.txt.

[3] Steven M. Bellovin. Using the Domain Name System for System Break-ins. In *Proceedings of the Fifth USENIX UNIX Security Symposium*, 1995.

[4] Dan Kaminsky. Its The End Of The Cache As We Know It. In *Black Hat USA 2008*, 2008.

[5] Robert Lemos. Largest-Ever DDoS Campaign Demonstrates Danger of New Attack Method, March 2013. http://www.eweek.com/security/largest-ever-ddos-campaign-demonstrates-danger-of-new-attack-method/.

[6] Jose Nazario and T. Holz. As the net churns: Fast-flux botnet observations. In *Malicious and Unwanted Software, 2008. MALWARE 2008. 3rd International Conference on*, pages 24–31, Oct 2008.

[7] Jon Oberheide, Manish Karir, and Z. Morley Mao. *Characterizing Dark DNS Behavior*, pages 140–156. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[8] Hyunsang Choi, Hanwoo Lee, Heejo Lee, and Hyogon Kim. Botnet Detection by Monitoring Group Activities in DNS Traffic. In *7th IEEE International Conference on Computer and Information Technology CIT 2007)*, pages 715–720, 2007.

[9] Keisuke Ishibashi, Tsuyoshi Toyono, Katsuyasu Toyama, Masahiro Ishino, Haruhiko Ohshima, and Ichiro Mizukoshi. Detecting Mass-mailing Worm Infected Hosts by Mining DNS Traffic Data. In *Proceedings of the 2005 ACM SIGCOMM*

*Workshop on Mining Network Data*, MineNet '05, pages 159–164, New York, NY, USA, 2005. ACM.

[10] David Whyte, Evangelos Kranakis, and Paul C. van Oorschot. DNS-based Detection of Scanning Worms in an Enterprise Network. In *Proceedings of the Network and Distributed System Security Symposium (NDSS 2005)*, pages 181–195, 2005.

[11] Manos Antonakakis, Roberto Perdisci, David Dagon, Wenke Lee, and Nick Feamster. Building a Dynamic Reputation System for DNS. In *19th Usenix Security Symposium*, 2010.

[12] Donald Eastlake and Charles Kaufman. RFC 2065: Domain Name System Security Extensions, 1997. URL https://www.rfc-editor.org/rfc/rfc2065.txt.

[13] Donald Eastlake. RFC 2535: Domain Name System Security Extensions, 1999. URL https://tools.ietf.org/html/rfc2535.

[14] Roy Arends, Rob Austein, Matt Larson, Dan Massey, and Scott Rose. RFC 4033: DNS security introduction and requirements, 2005. URL http://www.ietf.org/rfc/rfc4033.txt.

[15] Roy Arends, Rob Austein, Matt Larson, Dan Massey, and Scott Rose. RFC 4034: Resource records for the DNS security extensions, 2005. URL http://www.ietf.org/rfc/rfc4034.txt.

[16] Roy Arends, Rob Austein, Matt Larson, Dan Massey, and Scott Rose. RFC 4035: Protocol modifications for the DNS security extensions, 2005. URL http://www.ietf.org/rfc/rfc4035.txt.

[17] Marios Anagnostopoulos, Georgios Kambourakis, Elisavet Konstantinou, and Stefanos Gritzalis. *DNSSEC vs. DNSCurve: A Side-by-Side Comparison*, page 201. IGI Global, 2012.

[18] Marios Anagnostopoulos, Georgios Kambourakis, Panagiotis Kopanos, Georgios Louloudakis, and Stefanos Gritzalis. DNS Amplification Attack Revisited. *Computers & Security*, 39, Part B:475 – 485, 2013.

[19] Marios Anagnostopoulos, Georgios Kambourakis, and Stefanos Gritzalis. Never say never: Authoritative TLD nameserver-powered DNS amplification. In *Under Preparation*, 2016.

[20] Marios Anagnostopoulos, Georgios Kambourakis, and Stefanos Gritzalis. New facets of mobile botnet: architecture and evaluation. *International Journal of Information Security*, pages 1–19, 2015.

[21] Dimitrios Damopoulos, Georgios Kambourakis, Marios Anagnostopoulos, Stefanos Gritzalis, and J. H. Park. User-privacy and modern smartphones: A Siri(ous) dilemma. In *Proceedings of the FTRA AIM 2012 International Conference on Advanced IT, Engineering and Management*. FTRA, 2012.

[22] Dimitrios Damopoulos, Georgios Kambourakis, Marios Anagnostopoulos, Stefanos Gritzalis, and JH Park. User privacy and modern mobile services: are they on the same path? *Personal and ubiquitous computing*, 17(7):1437–1448, 2013.

[23] Apple, 2016. URL http://www.apple.com/ios/siri/.

[24] Mark Handley, Van Jacobson, and Colin Perkins. RFC 4566 : SDP session description protocol, 2006. URL https://tools.ietf.org/html/rfc4566.

[25] Jonathan Rosenberg, Henning Schulzrinne, Gonzalo Camarillo, Alan Johnston, Jon Peterson, Robert Sparks, Mark Handley, and Eve Schooler. RFC261: SIP Session Initiation Protocol, 2002. URL https://tools.ietf.org/html/rfc3261.

[26] Zisis Tsiatsikas, Marios Anagnostopoulos, Georgios Kambourakis, Sozon Lambrou, and Dimitris Geneiatakis. Hidden in Plain Sight. SDP-Based Covert Channel for Botnet Communication. In Simone Fischer-Hubner, Costas Lambrinoudakis, and Javier Lopez, editors, *Trust, Privacy and Security in Digital Business*, volume 9264 of *Lecture Notes in Computer Science*, pages 48–59. Springer International Publishing, 2015.

[27] Marc Blanchet and Lars-Johan Liman. RFC 7720: DNS Root Name Service Protocol and Deployment Requirements, 2015. URL https://tools.ietf.org/html/rfc7720.

[28] Jon Postel. RFC 1591: Domain Name System Structure and Delegation, 1994. URL http://https://tools.ietf.org/html/rfc1591.

[29] ISO 3166-1-alpha-2 code elements, English country names and code elements, October 2005.

[30] Joe Abley and Terry Manderson. RFC 5855: Nameservers for IPv4 and IPv6 Reverse Zones, 2010. URL https://tools.ietf.org/html/rfc5855.

[31] Liu Cricket and Paul Albitz. *DNS and BIND (Fifth Edition)*. O'Reilly Media, Inc., 2006.

[32] Rom Aitchison. *Pro DNS and BIND*. Apress, 2005.

[33] Paul Hoffman, Andrew Sullivan, and Kazunori Fujiwara. RFC 7719: DNS Terminology, 2015. URL https://tools.ietf.org/html/rfc7719.

[34] Edward Lewis. RFC 5936: DNS Zone Transfer Protocol (AXFR), 2010. URL https://tools.ietf.org/html/rfc5936.

[35] ICANN's Root Server System Advisory Committee (RSSAC). RSSAC001: Service Expectations of Root Servers, 2013. URL https://www.icann.org/en/system/files/files/rssac-001-draft-02may13-en.pdf.

[36] David Dagon, Niels Provos, Christopher P Lee, and Wenke Lee. Corrupted DNS resolution paths: The rise of a malicious resolution authority. In *Proceedings of Network and Distributed Security Symposium (NDSS08)*, 2008.

[37] Donald Eastlake. RFC 6895: Domain Name System (DNS) IANA Considerations, 2013. URL https://tools.ietf.org/html/rfc6895.

[38] Mark Andrews. RFC 2308: Negative Caching of DNS Queries (DNS NCACHE), 1998. URL https://tools.ietf.org/html/rfc2308.

[39] Internet Assigned Numbers Authority (IANA). Root files. URL https://www.iana.org/domains/root/files.

[40] Scott Kitterman. RFC 7208: Sender Policy Framework (SPF) for Authorizing Use of Domains in Email, 2014. URL https://tools.ietf.org/html/rfc7208.

[41] Wes Hardaker. RFC 7477: Child-to-Parent Synchronization in DNS, 2015. URL https://tools.ietf.org/html/rfc7477.

[42] Masataka Ohta. RFC 1995: Incremental Zone Transfer in DNS, 1996. URL https://tools.ietf.org/html/rfc1995.

[43] Paul Vixie. RFC 1996: A Mechanism for Prompt Notification of Zone Changes (DNS NOTIFY), 1996. URL https://www.ietf.org/rfc/rfc1996.txt.

[44] Paul Vixie, Susan Thomson, Yakov Rekhter, and Jim Bound. RFC 2136: Dynamic Updates in the Domain Name System (DNS UPDATE), 1997. URL http://www.ietf.org/rfc/rfc2136.txt.

[45] Bill Manning. DNSSEC & Operations, 2003. URL www.nordunet2003.is/smasidur/presentations/Manning.pps.

[46] Derek Atkins and Rob Austein. RFC 3833: Threat Analysis of the Domain Name System (DNS), 2004. URL https://tools.ietf.org/html/rfc3833.

[47] David Dagon, Manos Antonakakis, Kevin Day, Xiapu Luo, Christopher P Lee, and Wenke Lee. Recursive DNS architectures and vulnerability implications. In *Proceedings of 16th Network and Distributed System Security Symposium (NDSS)*, 2009.

[48] Paul Vixie, Olafur Gudmundsson, Donald E. Eastlake, and Brian Wellington. RFC 2485: Secret Key Transaction Authentication for DNS (TSIG), 2000. URL https://tools.ietf.org/html/rfc2845.

[49] David Dagon, Manos Antonakakis, Paul Vixie, Tatuya Jinmei, and Wenke Lee. Increased DNS forgery resistance through 0x20-bit encoding. In *15th ACM Conference on Computer and Communications Security (CCS 2008)*, pages 211–222. ACM, 2008.

[50] Bert Hubert and Remco van Mook. RFC 5452: Measures for Making DNS More Resilient against Forged Answers, 2009. URL https://www.ietf.org/rfc/rfc5452.txt.

[51] Mauro Conti, Nicola Dragoni, and Viktor Lesyk. A Survey of Man In The Middle Attacks. *IEEE Communications Surveys Tutorials*, PP(99):1–1, 2016.

[52] Suranjith Ariyapperuma and Chris J Mitchell. Security vulnerabilities in DNS and DNSSEC. In *The Second International Conference on Availability, Reliability and Security, 2007. ARES 2007.*, pages 335–342. IEEE, 2007.

[53] Robert Elz and Randy Bush. RFC 2181: Clarifications to the DNS Specification, 1997. URL https://tools.ietf.org/html/rfc2181.

[54] Sooel Son and Vitaly Shmatikov. *The Hitchhiker's Guide to DNS Cache Poisoning*, pages 466–483. Springer Berlin Heidelberg, 2010.

[55] Chad Dougherty. CERT Vulnerability Note VU 800113: Multiple DNS implementations vulnerable to cache poisoning, 2008. URL http://www.kb.cert.org/vuls/id/800113.

[56] Nikolaos Alexiou, Stylianos Basagiannis, Panagiotis Katsaros, Tushar Dashpande, and Scott A. Smolka. Formal Analysis of the Kaminsky DNS Cache-Poisoning Attack Using Probabilistic Model Checking. In *IEEE 12th International Symposium on High-Assurance Systems Engineering (HASE), 2010*, pages 94–103, Nov 2010.

[57] Daniel J. Bernstein. djbdns. http://cr.yp.to/djbdns.html.

[58] Michael Vittrup Larsen and Fernando Gont. RFC 6056: Recommendations for Transport-Protocol Port Randomization, 2011. URL https://tools.ietf.org/html/rfc6056.

[59] Donald Eastlake. RFC 4343: Domain Name System (DNS) Case Insensitivity Clarification, 2006. URL https://tools.ietf.org/html/rfc4343.

[60] Alexa. Alexa : The top 500 sites on the web. URL http://www.alexa.com/topsites.

[61] Roberto Perdisci, Manos Antonakakis, Xiapu Luo, and Wenke Lee. WSEC DNS: Protecting Recursive DNS Resolvers from Poisoning Attacks. In *2009 IEEE/IFIP International Conference on Dependable Systems Networks (DSN-DCCS'09)*, 2009.

[62] Roberto Perdisci, Manos Antonakakis, Xiapu Luo, and Wenke Lee. WSEC DNS: Protecting Recursive DNS Resolvers from Poisoning Attacks (extended). Technical report, 2009. URL http://roberto.perdisci.com/projects/wsecdns.

[63] Jonathan Trostle, Bill Van Besien, and Ashish Pujari. Protecting against DNS cache poisoning attacks. In *Secure Network Protocols (NPSec), 2010 6th IEEE Workshop on*, pages 25–30, 2010.

[64] Paul Mockapetris. RFC 882: Domain names - concepts and facilities, 1983. URL https://tools.ietf.org/html/rfc882.

[65] Paul Mockapetris. RFC 883: Domain Names - Implementation and Specification, 1983. URL https://tools.ietf.org/html/rfc883.

[66] Daniel J. Bernstein. DNSCurve, 2009. http://dnscurve.org.

[67] Daniel J. Bernstein. DNSCurve: Usable security for DNS, 2008. http://cr.yp.to/talks/2008.08.22/slides.pdf.

[68] IANIX. DNSCurve Software. URL https://dnscurve.io/dnscurve/dnscurve-software.html.

[69] Matthew Dempsky. DNSCurve: Link-Level Security for the Domain Name System. Internet Draft, 2010. https://tools.ietf.org/html/draft-dempsky-dnscurve-01.

[70] Ben Laurie, Geoffrey Sisson, Roy Arends, and David Blacka. RFC 5155: DNS Security (DNSSEC) Hashed Authenticated Denial of Existence, 2008. URL https://tools.ietf.org/html/rfc5155.

[71] Paul Vixie. RFC 2671: Extension Mechanisms for DNS (EDNS0), 1999. URL https://www.ietf.org/rfc/rfc2671.txt.

[72] Internet Systems Consortium. DNSSEC key generation tool, . http://ftp.isc.org/isc/bind/cur/9.9/doc/arm/man.dnssec-keygen.html.

[73] Internet Systems Consortium. DNSSEC zone signing tool, . http://ftp.isc.org/www/bind/arm95/man.dnssec-signzone.html.

[74] Olaf Kolkman, Jakob Schlyter, and Edward Lewis. RFC 3757: Domain Name System KEY (DNSKEY) Resource Record (RR) Secure Entry Point (SEP) Flag, 2004. URL http://www.ietf.org/rfc/rfc3757.txt.

[75] Samuel Weiler and Johan Ihren. RFC 4470: Minimally Covering NSEC Records and DNSSEC On-line Signing, 2006. URL https://tools.ietf.org/html/rfc4470.

[76] Geoffrey Sisson and Ben Laurie. RFC 4471: Derivation of DNS Name Predecessor and Successor, 2006. URL https://tools.ietf.org/html/rfc4471.

[77] Wes Hardaker. RFC 4509: Use of SHA-256 in DNSSEC Delegation Signer (DS) Resource Records (RRs), 2006. URL https://www.ietf.org/rfc/rfc4509.txt.

[78] Donald Eastlake. RFC 2537: RSA/MD5 KEYs and SIGs in the Domain Name System (DNS), 1999. URL https://tools.ietf.org/html/rfc2537.

[79] Donald Eastlake. RFC 3110: RSA/SHA-1 SIGs and RSA KEYs in the Domain Name System (DNS), 2001. URL https://tools.ietf.org/html/rfc3110.

[80] Scott Rose. RFC 6944: Applicability Statement: DNS Security (DNSSEC) DNSKEY Algorithm Implementation Status, 2013. URL https://tools.ietf.org/html/rfc6944.

[81] Donald Eastlake. RFC 2539: Storage of Diffie-Hellman Keys in the Domain Name System (DNS), 1999. URL https://tools.ietf.org/html/rfc2539.

[82] Donald Eastlake. RFC 2536: DSA KEYs and SIGs in the Domain Name System (DNS), 1999. URL https://tools.ietf.org/html/rfc2536.

[83] Scott Rose. RFC 6725: DNS Security (DNSSEC) DNSKEY Algorithm IANA Registry Updates, 2012. URL http://tools.ietf.org/html/rfc6725.

[84] Jelte Jansen. RFC 5702: Use of SHA-2 Algorithms with RSA in DNSKEY and RRSIG Resource Records for DNSSEC, 2009. URL http://tools.ietf.org/html/rfc5702.

[85] Vasily Dolmatov, Artem Chuprina, and Igor Ustinov. RFC 5933: Use of GOST Signature Algorithms in DNSKEY and RRSIG Resource Records for DNSSEC, 2010. URL http://tools.ietf.org/html/rfc5933.

[86] Paul Hoffman and Wouter C.A. Wijngaards. RFC 6605: Elliptic Curve Digital Signature Algorithm (DSA) for DNSSEC, 2012. URL http://tools.ietf.org/html/rfc6605.

[87] Paul Hoffman. RFC 6014: Cryptographic Algorithm Identifier Allocation for DNSSEC, 2010. URL http://tools.ietf.org/html/rfc6014.

[88] Joao Damas and Paul Vixie. RFC 6891: Extension Mechanisms for DNS (EDNS0), 2013. URL https://tools.ietf.org/html/rfc6891.

[89] David Conrad. RFC 3225: Indicating Resolver Support of DNSSEC, 2001. URL https://tools.ietf.org/html/rfc3225.

[90] Samuel Weiler and David Blacka. RFC 6840: Clarifications and Implementation Notes for DNS Security (DNSSEC), 2013. URL https://tools.ietf.org/html/rfc6840.

[91] Eric Osterweil, Dan Massey, Michael Ryan, and Lixia Zhang. Quantifying the operational status of the DNSSEC deployment. In *Proceedings of the 8th ACM SIGCOMM conference on Internet measurement*, ACM SIGCOMM, pages 231–242, Greece, 2008. ACM.

[92] Eric Osterweil, Dan Massey, and Lixia Zhang. Deploying and Monitoring DNS Security (DNSSEC). In *In IEEE Computer Society: 25th Annual Computer Security Applications Conference (ACSAC '09)*, pages 231–242, Honolulu, Hawaii, USA, 2009. IEEE.

[93] Fredrik Ljunggren, Tomofumi Okubo, Richard Lamb, and Jakob Schlyter. DNSSEC Practice Statement for the Root Zone KSK Operator, 2010. URL https://www.iana.org/dnssec/icann-dps.txt.

[94] Michael StJohns. RFC 5011: Automated Updates of DNS Security (DNSSEC) Trust Anchors, 2007. URL https://tools.ietf.org/html/rfc5011.

[95] Howard Eland, Russ Mundy, Steve Crocker, and Suresh Krishnaswamy. RFC 4986: Requirements Related to DNS Security (DNSSEC) Trust Anchor Rollover, 2007. URL https://tools.ietf.org/html/rfc4986.

[96] Paul Ebersman, Warren Kumari, Chris Griffiths, Jason Livingood, and Ralf Weber. RFC 7646: Definition and Use of DNSSEC Negative Trust Anchors, 2015. URL https://tools.ietf.org/html/rfc7646.

[97] Mark Andrews and Samuel Weiler. RFC 4431: The DNSSEC Lookaside Validation (DLV) DNS Resource Record, 2006. URL https://tools.ietf.org/html/rfc4431.

[98] Samuel Weiler. RFC 5074: DNSSEC Lookaside Validation (DLV), 2007. URL https://tools.ietf.org/html/rfc5074.

[99] Hao Yang, Eric Osterweil, Dan Massey, Songwu Lu, and Lixia Zhang. Deploying Cryptography in Internet-Scale Systems: A Case Study on DNSSEC. In *Dependable and Secure Computing, IEEE Transactions on*, volume 8, pages 656–669. IEEE, 2010.

[100] Internet Systems Consortium (ISC). DNSSEC Look-aside Validation Registry, . URL https://dlv.isc.org/.

[101] ICANN Research. TLD DNSSEC Report. URL http://stats.research.icann.org/dns/tld_report/.

[102] Internet Systems Consortium (ISC). DNSSEC Deployment Report, . URL https://rick.eng.br/dnssecstat/.

[103] Niels L. M. Adrichem, Norbert Blenn, Antonio Reyes Lúa, Xin Wang, Muhammad Wasif, Ficky Fatturrahman, and Fernando A. Kuipers. A measurement study of DNSSEC misconfigurations. *Security Informatics*, 4(1):1–14, 2015.

[104] Jason Livingood. Responsibility for Authoritative DNSSEC Mistakes. Internet Draft, 2015. URL https://tools.ietf.org/html/draft-livingood-dnsop-auth-dnssec-mistakes-03.

[105] Matthaeus Wander and Torben Weis. Measuring Occurrence of DNSSEC Validation. In *14th International Conference Passive and Active Measurement (PAM), In Proceedings of*, pages 125–134. Springer Berlin Heidelberg, Aug 2013.

[106] Yingdi Yu, Duane Wessels, Matt Larson, and Matt Zhang. Check-Repeat: A new method of measuring DNSSEC validating resolvers. In *International Conference on Computer Communications (INFOCOM) 2013, In Proceedings of IEEE*, pages 3147–3152, Aug 2013.

[107] Wilson Lian, Eric Rescorla, Hovav Shacham, and Stefan Savage. Measuring the Practical Impact of DNSSEC Deployment. In *22nd USENIX Security Symposium (USENIX Security 13)*, pages 573–588, Washington, D.C., August 2013. USENIX.

[108] Daniel J. Bernstein. High-speed cryptography and DNSCurve, 2009. http://cr.yp.to/talks/2009.06.27/slides.pdf.

[109] Daniel J. Bernstein. Cryptography in NaCl, 2009. http://cr.yp.to/highspeed/naclcrypto-20090310.pdf.

[110] Daniel J. Bernstein. *The Salsa20 family of stream ciphers*, pages 84–97. Springer Berlin Heidelberg, 2008.

[111] Daniel J. Bernstein. The poly1305-aes message-authentication code. In *12th International Workshop on Fast Software Encryption (FSE 2005)*, pages 32–49. Springer Berlin Heidelberg, 2005.

[112] Daniel J. Bernstein. Curve25519: new Diffie-Hellman speed records. In *Public Key Cryptography (PKC 2006), In Proceedings of the*, pages 207–228. Springer Berlin Heidelberg, 2006.

[113] IEEE P1363. Standard Specifications For Public-Key Cryptography, 2008. http://grouper.ieee.org/groups/1363/index.html.

[114] Olaf Kolkman, Matthijs Mekking, and Miek Gieben. RFC 6781: DNSSEC Operational Practices, Version 2, 2012. URL https://tools.ietf.org/html/rfc6781.

[115] ENISA. Algorithms, key size and parameters report 2014. Technical report, 2014.

[116] eSTREAM. The ECRYPT Stream cipher project, 2008. http://www.ecrypt.eu.org/stream/salsa20pf.html.

[117] Jason Bau and John C Mitchell. A Security Evaluation of DNSSEC with NSEC3. In *IACR Eprint archive*. IEEE, 2010. Revised and corrected version of conference paper in Network and Distributed Systems Security (NDSS) 2010.

[118] Eric Osterweil, Dan Massey, and Lixia Zhang. Observations from the DNSSEC Deployment. In *IEEE Workshop on Secure Network Protocols 2007 (NPSEC 2007)*, pages 1–6, Beijing, China, 2007. IEEE.

[119] Yan He, Eric Osterweil, Jon Hajdu, Jonas Acres, and Dan Massey. Limiting replay vulnerabilities in DNSSEC. In *4th Workshop on Secure Network Protocols 2008 (NPSec 2008)*, pages 3–8, 2008.

[120] Stephane Bortzmeyer. RFC 7626: DNS Privacy Considerations, 2015. URL https://tools.ietf.org/html/rfc7626.

[121] Sharon Goldberg, Moni Naor, Dimitrios Papadopoulos, Leonid Reyzin, Sachin Vasant, and Asaf Ziv. Stretching NSEC3 to the Limit: Efficient Zone Enumeration Attacks on NSEC3 Variants. Technical report, Boston University, 2015. URL https://www.cs.bu.edu/~goldbe/papers/nsec3attacks.pdf.

[122] Matthaeus Wander, Lorenz Schwittmann, Christopher Boelmann, and Torben Weis. GPU-Based NSEC3 Hash Breaking. In *Network Computing and Applications (NCA), 2014 IEEE 13th International Symposium on*, pages 137–144, August 2014.

[123] Sharon Goldberg, Moni Naor, Dimitrios Papadopoulos, Leonid Reyzin, Sachin Vasant, and Asaf Ziv. NSEC5: Provably Preventing DNSSEC Zone Enumeration. In *2015 Network and Distributed System Security Symposium (NDSS15)*, February 2015.

[124] Randal Vaughn and Gadi Evron. DNS amplification attacks (Preliminary Release), 2006. http://www.isotf.org/news/DNS-Amplification-Attacks.pdf.

[125] Joao Damas and Frederico Neves. RFC 5358: Preventing Use of Recursive Nameservers in Reflector Attacks, 2008. URL http://tools.ietf.org/html/rfc5358.

[126] Yuri Schaeffer, Benno J. Overeinder, and Matthijs Mekking. Flexible and Robust Key Rollover in DNSSEC. In *Proceedings of the Workshop on Securing and Trusting Internet Names (SATIN 2012)*, 2012.

[127] Warren Kumari, Olafur Gudmundsson, and George Barwood. RFC 7344: Automating DNSSEC Delegation Trust Maintenance, 2014. URL https://tools.ietf.org/html/rfc7344.

[128] Stephen Morris, Johan Ihren, John Dickinson, and Matthijs Mekking. RFC 7583: DNSSEC Key Rollover Timing Considerations, 2015. URL https://www.rfc-editor.org/rfc/rfc7583.txt.

[129] Bernhard Ager, Holger Dreger, and Anja Feldmann. Exploring the Overhead of DNSSEC, 2005. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.206.1442&rep=rep1&type=pdf.

[130] Olaf Kolkman and Miek Gieben. RFC 4641: DNSSEC Operational Practices, 2006. URL https://www.ietf.org/rfc/rfc4641.txt.

[131] Miek Gieben and Matthijs Mekking. RFC 7129: Authenticated Denial of Existence in the DNS, 2014. URL https://tools.ietf.org/html/rfc7129.

[132] Crypto++ 5.6.0 Benchmarks. Speed comparison of popular crypto algorithms, 2009. https://www.cryptopp.com/benchmarks.html.

[133] Olaf Kolkman. Measuring the resource requirements of DNSSEC. Technical report, 2005. URL http://search.belnet.be/pub/ftp.ripe.net/ripe/docs/ripe-352.pdf.

[134] Tirumaleswar Reddy, Dan Wing, and Prashanth Patil. Specification for DNS over Datagram Transport Layer Security (DTLS). Internet Draft, 2016. URL https://tools.ietf.org/html/draft-ietf-dprive-dnsodtls-12.

[135] Eric Rescorla and Nagendra Modadugu. RFC 6347: Datagram Transport Layer Security Version 1.2, 2012. URL https://tools.ietf.org/html/rfc6347.

[136] Marco Tiloca, Christian Gehrmann, and Ludwig Seitz. On improving resistance to Denial of Service and key provisioning scalability of the DTLS handshake. *International Journal of Information Security*, pages 1–21, 2016.

[137] Zi Hu, Liang Zhu, John Heidemann, Allison Mankin, Duane Wessels, and Paul Hoffman. RFC 7858: Specification for DNS over Transport Layer Security (TLS), 2016. URL https://tools.ietf.org/html/rfc7858.

[138] Tim Dierks and Eric Rescorla. RFC 5246: The Transport Layer Security (TLS) Protocol Version 1.2, 2008. URL https://tools.ietf.org/html/rfc5246.

[139] P. Morrissey, N. P. Smart, and B. Warinschi. *A Modular Security Analysis of the TLS Handshake Protocol*, pages 55–73. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[140] Fabrice J. Ryba, Matthew Orlinski, Matthias Waehlisch, Christian Rossow, and Thomas C. Schmidt. Amplification and DRDoS Attack Defense–A Survey and New Perspectives. *arXiv preprint arXiv:1505.07892*, 2015. URL http://arxiv.org/abs/1505.07892.

[141] Roland van Rijswijk-Deij, Anna Sperotto, and Aiko Pras. DNSSEC and Its Potential for DDoS Attacks: A Comprehensive Measurement Study. In *Proceedings of the 2014 Conference on Internet Measurement Conference*, IMC '14, pages 449–460, New York, NY, USA, 2014. ACM.

[142] Vern Paxson. An Analysis of Using Reflectors for Distributed Denial-of-service Attacks. *ACM SIGCOMM Computer Communication Review*, 31(3):38–47, July 2001.

[143] US-CERT. The Continuing Denial of Service Threat Posed by DNS Recursion (v2.0). Technical report, US-CERT, 2013. https://www.us-cert.gov/sites/default/files/publications/DNS-recursion033006.pdf.

[144] Don Jackson. DNS Amplification Variation Used in Recent DDoS Attacks, February 2009. http://www.secureworks.com/cyber-threat-intelligence/threats/dns-amplification.

[145] US-CERT. DNS Amplification Attacks. Technical report, US-CERT, 2013. https://www.us-cert.gov/ncas/alerts/TA13-088A.

[146] Bernhard Ager, Holger Dreger, and Anja Feldmann. Predicting the DNSSEC overhead using DNS traces. In *40th Annual Conference on Information Sciences and Systems (2006)*, pages 1484–1489, March 2006.

[147] Alex Cowperthwaite and Anil Somayaji. The futility of DNSSec. In *Proceedings of the 5th Annual Symposium on Information Assurance (ASIA)*, June 2010.

[148] Christian Rossow. Amplification Hell: Revisiting Network Protocols for DDoS Abuse. In *Proceedings of the 2014 Network and Distributed System Security Symposium (NDSS)*, 2014.

[149] Yuuki Takano, Ruo Ando, Takeshi Takahashi, Satoshi Uda, and Tomoya Inoue. A Measurement Study of Open Resolvers and DNS Server Version. In *Internet Conference (IEICE)*, 2013.

[150] Open Resolver Project. http://openresolverproject.org/.

[151] DNS - The Measurement Factory. http://dns.measurement-factory.com.

[152] Marc Kührer, Thomas Hupperich, Jonas Bushart, Christian Rossow, and Thorsten Holz. Going Wild: Large-Scale Classification of Open DNS Resolvers. In *Proceedings of the Internet Measurement Conference (IMC) October 2015, ACM*, pages 355–368, 2015.

[153] Andrew J. Kaizer and Minaxi Gupta. Open Resolvers: Understanding the Origins of Anomalous Open DNS Resolvers. In *Proceedings of the 16th International Conference Passive and Active Measurement (PAM 2015)*, pages 3–14. Springer International Publishing.

[154] Douglas C. MacFarland, Craig A. Shue, and Andrew J. Kalafut. Characterizing Optimal DNS Amplification Attacks and Effective Mitigation. In *Passive and Active Measurement Conference*. Springer, 2015.

[155] Paul Vixie and Vernon Schryver. DNS Response Rate Limiting (DNS RRL), June 2012. http://ss.vix.com/~vixie/isc-tn-2012-1.txt.

[156] Root Server Operators. Events of 2015-11-30, December 2015. http://www.root-servers.org/news/events-of-20151130.txt.

[157] Symantec. Internet Security Threat Report (ISTR20). 20, 2015.

[158] Anonymous. Operation global blackout, February 2012. http://pastebin.com/NKbnh8q8.

[159] Robert D. Graham. No, #Anonymous can't DDoS the root DNS servers, February 2012. http://erratasec.blogspot.gr/2012/02/no-anonymous-cant-ddos-root-dns-servers.html.

[160] Gadi Evron. Battling botnets and online mobs. *Georgetown Journal of International Affairs*, 9:121–126, 2008.

[161] Anthony Eden. Incident Report: DNS Outage due to DDoS Attack, June 2013. https://blog.dnsimple.com/2013/06/incident-report-20130603/.

[162] Scapy Project. http://www.secdev.org/projects/scapy/.

[163] NIST. Vulnerability Summary for CVE-2012-3411, March 2013. http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2012-3411.

[164] Marc Kührer, Thomas Hupperich, Christian Rossow, and Thorsten Holz. Exit from Hell? Reducing the Impact of Amplification DDoS Attacks. In *23rd USENIX Security Symposium (USENIX Security 14)*, pages 111–125, San Diego, CA, August 2014. USENIX Association.

[165] Ray Bellis. RFC 5966: DNS Transport over TCP - Implementation Requirements, 2010. URL https://tools.ietf.org/html/rfc5966.

[166] Gijs Van Den Broek, Roland Van Rijswijk-Deij, Anna Sperotto, and Aiko Pras. DNSSEC meets real world: dealing with unreachability caused by fragmentation. *IEEE Communications Magazine*, 52(4):154–160, April 2014.

[167] Internet System Consortium. *BIND 9 Administrator Reference Manual*, 2014.

[168] Manos Antonakakis, David Dagon, Xiapu Luo, Roberto Perdisci, Wenke Lee, and Justin Bellmor. A centralized monitoring infrastructure for improving dns security. In *13th International Conference on Recent Advances in Intrusion Detection*, pages 18–37. Springer, 2010.

[169] Claude Fachkha, Elias Bou-Harb, and Mourad Debbabi. Inferring distributed reflection denial of service attacks from darknet. *Computer Communications*, 62: 59 – 71, 2015.

[170] Olafur Gudmundsson and Marek Majkowski. Deprecating the DNS ANY meta-query type, March 2015. URL https://blog.cloudflare.com/deprecating-dns-any-meta-query-type/.

[171] John Dickinson, Sara Dickinson, Ray Bellis, Allison Mankin, and Duane Wessels. RFC 7766: DNS Transport over TCP - Implementation Requirements, 2016. URL https://tools.ietf.org/html/rfc7766.

[172] Paul Ferguson and Daniel Senie. RFC 2827: Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing, 2000. URL http://www.ietf.org/rfc/rfc2827.txt.

[173] Robert Beverly, Ryan Koga, and Kc Claffy. Initial Longitudinal Analysis of IP Source Spoofing Capability on the Internet, 2013. URL http://www.internetsociety.org/doc/initial-longitudinal-analysis-ip-source-spoofing-capability-internet.

[174] Spoofer Project: State of IP Spoofing. http://www.secdev.org/projects/scapy/.

[175] Georgios Kambourakis, Tassos Moschos, Dimitris Geneiatakis, and Stefanos Gritzalis. Detecting DNS amplification attacks. In *Critical Information Infrastructures Security*, pages 185–196. Springer, 2008.

[176] Georgios Kambourakis, Tassos Moschos, Dimitris Geneiatakis, and Stefanos Gritzalis. A Fair Solution to DNS Amplification Attacks. In *Digital Forensics and Incident Analysis, 2007. WDFIA 2007. Second International Workshop on*, pages 38–47, 2007.

[177] Sebastiano Di Paola and Dario Lombardo. Protecting against DNS reflection attacks with Bloom filters. In *Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 1–16. Springer, 2011.

[178] Changhua Sun, Bin Liu, and Lei Shi. Efficient and low-cost hardware defense against DNS amplification attacks. In *Global Telecommunications Conference, 2008. IEEE GLOBECOM 2008. IEEE*, pages 1–5. IEEE, 2008.

[179] Samaneh Rastegari, M. Iqbal Saripan, and Mohd Fadlee A. Rasid. Detection of Denial of Service Attacks against Domain Name System Using Neural Networks. *International Journal of Computer Science Issues*, 6:23–27, 2009.

[180] Samaneh Rastegari, M Iqbal Saripan, and Mohd Fadlee A Rasid. Detection of Denial of Service Attacks against Domain Name System Using Machine Learning Classifiers. *Lecture Notes in Engineering and Computer Science: Proceedings of The World Congress on Engineering 2010*, pages 444–447, 2010.

[181] Tong Guang Ni, Xiao Qing Gu, and Hong Yuan Wang. Detecting DDoS Attacks Against DNS Servers Using Time Series Analysis. *TELKOMNIKA Indonesian Journal of Electrical Engineering*, 12(1):753–761, 2014.

[182] Tushar Deshpande, Panagiotis Katsaros, Stylianos Basagiannis, and Scott A Smolka. Formal analysis of the DNS Bandwidth Amplification Attack and its countermeasures using probabilistic model checking. In *High-Assurance Systems Engineering (HASE), 2011 IEEE 13th International Symposium on*, pages 360–367. IEEE, 2011.

[183] Allison Mankin. Random drop congestion control. In *Proceedings of the ACM symposium on Communications architectures & protocols*, volume 20, pages 1–7. ACM, 1990.

[184] Michael Walfish, Mythili Vutukuru, Hari Balakrishnan, David Karger, and Scott Shenker. DDoS defense by offense. In *Proceedings of the 2006 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pages 303–314, 2006.

[185] Claude Fachkha, Elias Bou-Harb, and Mourad Debbabi. Fingerprinting Internet DNS Amplification DDoS Activities. In *2014 6th International Conference on New Technologies, Mobility and Security (NTMS)*, pages 1–5, March 2014.

[186] David Moore, Colleen Shannon, Douglas J. Brown, Geoffrey M. Voelker, and Stefan Savage. Inferring Internet Denial-of-service Activity. *ACM Transactions on Computer Systems*, 24(2):115–139, May 2006.

[187] Sergio S.C. Silva, Rodrigo M.P. Silva, Raquel C.G. Pinto, and Ronaldo M. Salles. Botnets: A survey. *Computer Networks*, 57(2):378 – 403, 2013.

[188] Felix C. Freiling, Thorsten Holz, and Georg Wicherski. Botnet Tracking: Exploring a Root-Cause Methodology to Prevent Distributed Denial-of-Service Attacks. In *Proceedings of 10th European Symposium on Research in Computer Security (ESORICS)*, pages 319–335. Springer, 2005.

[189] Anirudh Ramachandran and Nick Feamster. Understanding the Network-level Behavior of Spammers. In *Proceedings of the 2006 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '06, pages 291–302, New York, NY, USA, 2006. ACM.

[190] Thorsten Holz, Markus Engelberth, and Felix Freiling. Learning More About the Underground Economy: A Case-Study of Keyloggers and Dropzones. In *Proceedings of European Symposium on Research in Computer Security (ESORICS)*. Springer, 2009.

[191] N. Ianelli and A. Hackworth. Botnets as a vehicle for online crime. In *CERT Coordination Center*, 2005.

[192] Cormac Herley and Dinei Florêncio. *Nobody Sells Gold for the Price of Silver: Dishonesty, Uncertainty and the Underground Economy*, pages 33–53. Springer US, Boston, MA, 2010.

[193] Moheeb Abu Rajab, Jay Zarfoss, Fabian Monrose, and Andreas Terzis. A Multifaceted Approach to Understanding the Botnet Phenomenon. In *Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement*, IMC '06, pages 41–52, New York, NY, USA, 2006. ACM.

[194] Gunter Ollmann. Botnet Communication Topologies, 2009. URL https://www.damballa.com/downloads/r_pubs/WP_Botnet_Communications_Primer.pdf.

[195] David Dagon, Guofei Gu, Christopher P. Lee, and Wenke Lee. A Taxonomy of Botnet Structures. In *Computer Security Applications Conference, 2007 (ACSAC 2007). Twenty-Third Annual*, pages 325–339, Dec 2007.

[196] Ping Wang, Lei Wu, Baber Aslam, and Cliff Changchun Zou. A systematic study on peer-to-peer botnets. In *Computer Communications and Networks (ICCN2009), Proceedings of 18th Internatonal Conference on*, pages 1–8. IEEE, 2009.

[197] Ping Wang, Sherri Sparks, and Cliff Changchun Zou. An advanced hybrid peer-to-peer botnet. *Dependable and Secure Computing, IEEE Transactions on*, 7(2): 113–127, 2010.

[198] Evan Cooke, Farnam Jahanian, and Danny McPherson. The zombie roundup: Understanding, detecting, and disrupting botnets. In *Proceedings of the USENIX Steps to Reducing Unwanted Traffic on the Internet (SRUTI05) Workshop*, volume 39, page 44, 2005.

[199] Craig Schiller and James R. Binkley. *Botnets: The killer web applications*. Syngress, 2011.

[200] Nicole M Hands, Baijian Yang, and Raymond A. Hansen. A Study on Botnets Utilizing DNS. In *Proceedings of the 4th Annual ACM Conference on Research in Information Technology (RIIT '15)*, pages 23–28, 2015.

[201] Thorsten Holz, Christian Gorecki, Konrad Rieck, and Felix C. Freiling. Measuring and detecting fast-flux service networks, 2008.

[202] Manos Antonakakis, Roberto Perdisci, Yacin Nadji, Nikolaos Vasiloglou, Saeed Abu-Nimeh, Wenke Lee, and David Dagon. From Throw-Away Traffic to Bots: Detecting the Rise of DGA-Based Malware. In *Proceedings of the 21st USENIX Security Symposium (USENIX Security 12)*, pages 491–506, Bellevue, WA, 2012. USENIX.

[203] Brett Stone-Gross, Marco Cova, Lorenzo Cavallaro, Bob Gilbert, Martin Szydlowski, Richard Kemmerer, Christopher Kruegel, and Giovanni Vigna. Your Botnet is My Botnet: Analysis of a Botnet Takeover. In *Proceedings of the 16th ACM Conference on Computer and Communications Security*, CCS '09, pages 635–647, New York, NY, USA, 2009.

[204] Phillip Porras, Hassen Saidi, and Vinod Yegneswaran. A Foray into Conficker's Logic and Rendezvous Points. In *USENIX Workshop on Large-Scale Exploits and Emergent Threats*, 2009.

[205] Kui Xu, Patrick Butler, Sudip Saha, and Danfeng Yao. DNS for Massive-Scale Command and Control. *IEEE Transactions on Dependable and Secure Computing*, 10(3):143–153, May 2013.

[206] Guofei Gu, Roberto Perdisci, Junjie Zhang, Wenke Lee, et al. Botminer: Clustering analysis of network traffic for protocol-and structure-independent botnet detection. In *Proceedings of the 17th Conference on Security Symposium, USENIX Association*, pages 139–154, 2008.

[207] Goebel Jan and Thorsten Holz. Rishi: Identify Bot Contaminated Hosts by IRC Nickname Evaluation. In *Proceedings of USENIX HotBots'07*, 2007.

[208] Lucas Nussbaum, Pierre Neyron, and Olivier Richard. *On Robust Covert Channels Inside DNS*, pages 51–62. Springer Berlin Heidelberg, 2009.

[209] Christian J. Dietrich, Christian Rossow, Felix C. Freiling, Herbert Bos, Maarten van van Steen, and Norbert Pohlmann. On Botnets That Use DNS for Command and Control. In *Computer Network Defense (EC2ND) 2011, Seventh European Conference on*, pages 9–16, September 2011.

[210] Daan Raman, Bjorn De Sutter, Bart Coppens, Stijn Volckaert, Koen De Bosschere, Pieter Danhieux, and Erik Van Buggenhout. *DNS Tunneling for Network Penetration*, pages 65–77. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[211] Hamad Binsalleeh, A. Mert Kara, Amr Youssef, and Mourad Debbabi. Characterization of Covert Channels in DNS. In *New Technologies, Mobility and Security (NTMS), 2014 6th International Conference on*, pages 1–5, March 2014.

[212] Cathal Mullaney. Morto worm sets a (dns) record. Technical report, Aug 2011. URL http://www.symantec.com/connect/blogs/morto-worm-sets-dns-record.

[213] Axelle Apvrille. Symbian worm Yxes: Towards mobile botnets? *Journal in Computer Virology*, 8(4):117–131, 2012.

[214] Phillip Porras, Hassen Saidi, and Vinod Yegneswaran. An analysis of the iKee.B iPhone botnet. In *Security and Privacy in Mobile Information and Communication Systems*, pages 141–152. Springer, 2010.

[215] H. Pieterse and M.S. Olivier. Android botnets on the rise: Trends and characteristics. In *Information Security for South Africa (ISSA), 2012*, pages 1–5, Aug 2012.

[216] Yajin Zhou and Xuxian Jiang. Dissecting Android Malware: Characterization and Evolution. In *Security and Privacy (SP), 2012 IEEE Symposium on*, pages 95–109, May 2012.

[217] Mariantonietta La Polla, Fabio Martinelli, and Daniele Sgandurra. A survey on security for mobile devices. *Communications Surveys & Tutorials, IEEE*, 15(1): 446–471, 2013.

[218] Kapil Singh, Samrit Sangal, Nehil Jain, Patrick Traynor, and Wenke Lee. Evaluating bluetooth as a medium for botnet command and control. In *Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 61–80. Springer, 2010.

[219] Jingyu Hua and Kouichi Sakurai. A sms-based mobile botnet using flooding algorithm. In *Information Security Theory and Practice, International Workshop on Security and Privacy of Mobile Devices in Wireless Communication*, pages 264–279. Springer, 2011.

[220] Collin Mulliner and Jean-Pierre Seifert. Rise of the iBots: Owning a telco network. In *5th International Conference on Malicious and Unwanted Software (MALWARE) 2010*, pages 71–80. IEEE, 2010.

[221] Cui Xiang, Fang Binxing, Yin Lihua, Liu Xiaoyi, and Zang Tianning. Andbot: towards advanced mobile botnets. In *Proceedings of the 4th USENIX conference on Large-scale exploits and emergent threats*, pages 11–11. USENIX Association, Berkeley, CA, USA, 2011.

[222] Mohammad Reza Faghani and Mohammad Reza Nguyen. Socellbot: A new botnet design to infect smartphones via online social networking. In *25th IEEE Canadian Conference on Electrical Computer Engineering (CCECE) 2012*, pages 1–5, April 2012.

[223] Shuang Zhao, Patrick P. C. Lee, John C. S. Lui, Xiaohong Guan, Xiaobo Ma, and Jing Tao. Cloud-based Push-styled Mobile Botnets: A Case Study of Exploiting the Cloud to Device Messaging Service. In *Proceedings of the 28th Annual Computer Security Applications Conference*, ACSAC '12, pages 119–128, New York, NY, USA, 2012. ACM.

[224] Ragib Hasan, Nitesh Saxena, Tzipora Haleviz, Shams Zawoad, and Dustin Rinehart. Sensing-enabled Channels for Hard-to-detect Command and Control of Mobile Devices. In *Proceedings of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security*, ASIA CCS '13, pages 469–480, New York, NY, USA, 2013. ACM.

[225] David Dagon, Cliff Changchun Zou, and Wenke Lee. Modeling Botnet Propagation Using Time Zones. In *Proceedings of the 13th Annual Network and Distributed System Security Symposium (NDSS'06)*, volume 6, pages 2–13, 2006.

[226] Adrienne Porter Felt, Matthew Finifter, Erika Chin, Steve Hanna, and David Wagner. A survey of mobile malware in the wild. In *Proceedings of the 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices*, SPSM '11, pages 3–14, New York, NY, USA, 2011. ACM.

[227] Matthew Knysz, Xin Hu, Yuanyuan Zeng, and Kang G. Shin. Open WiFi networks: Lethal weapons for botnets? In *IEEE International Conference on Computer Communications (INFOCOM) 2012*, pages 2631–2635, March 2012.

[228] Thorsten Holz, Christian Gorecki, Konrad Rieck, and Felix C. Freiling. Measuring and detecting fast-flux service networks. In *Symposium on Network and Distributed System Security (NDSS08)*, 2008.

[229] Emanuele Passerini, Roberto Paleari, Lorenzo Martignoni, and Danilo Bruschi. *FluXOR: Detecting and Monitoring Fast-Flux Service Networks*, pages 186–206. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[230] Xin Hu, Matthew Knysz, and Kang G. Shin. RB-Seeker: Auto-detection of Redirection Botnets. In *Proceedings of 16th Network and Distributed System Security Symposium (NDSS)*, 2009.

[231] Roberto Perdisci, Igino Corona, David Dagon, and Wenke Lee. Detecting Malicious Flux Service Networks through Passive Analysis of Recursive DNS Traces. In

*Annual Computer Security Applications Conference (ACSAC '09)*, pages 311–320, 2009.

[232] Roberto Perdisci, Igino Corona, and Giorgio Giacinto. Early Detection of Malicious Flux Networks via Large-Scale Passive DNS Traffic Analysis. *IEEE Transactions on Dependable and Secure Computing*, 9(5):714–726, 2012.

[233] Brett Stone-Gross, Marco Cova, Bob Gilbert, Richard Kemmerer, Christopher Kruegel, and Giovanni Vigna. Analysis of a Botnet Takeover. *IEEE Security Privacy*, 9(1):64–72, January 2011.

[234] Kelly Burton. The Conficker Worm, 2012. URL https://www2.sans.org/security-resources/malwarefaq/conficker-worm.php.

[235] Moheeb Abu Rajab, Jay Zarfoss, Fabian Monrose, and Andreas Terzis. My botnet is bigger than yours (maybe, better than yours): why size estimates remain challenging. In *USENIX Workshop on Hot Topics in Understanding Botnet*, 2007.

[236] Juan Caballero, Pongsin Poosankam, Christian Kreibich, and Dawn Song. Dispatcher: Enabling Active Botnet Infiltration Using Automatic Protocol Reverse-engineering. In *Proceedings of the 16th ACM Conference on Computer and Communications Security (CCS '09)*, pages 621–634, 2009.

[237] Chia Yuan Cho, Juan Caballero, Chris Grier, Vern Paxson, and Dawn Song. Insights from the Inside: A View of Botnet Management from Infiltration. In *Proceedings of the USENIX Workshop on Large-Scale Exploits and Emergent Threat*, 2010.

[238] Cedric Halbronn and Jean Sigwald. iphone security model & vulnerabilities. In *Proceedings of Hack in the box sec-conference*, Kuala Lumpur, Malaysia, 2010.

[239] Jesse Burns. Exploratory android surgery. In *Black Hat Technical Security Conference USA*, 2009.

[240] Charlie Miller. Inside iOS Code Signing. In *Symposium on Security for Asia Network (SyScan)*, 2011.

[241] Stuart Cheshire and Marc Krochmal. RFC 6762: Multicast DNS, 2013. URL https://tools.ietf.org/html/rfc6762.

[242] Ryan Paul. How Applidium reverse engineered Siri's protocol, 2011. URL http://arstechnica.com/apple/2011/11/a-look-at-how-applidium-reverse-engineered-siris-protocol/.

[243] Collin Mulliner and Charlie Miller. Fuzzing the Phone in your Phone. In *Black Hat USA*, 2009.

[244] Pete Lamonica. Siriproxy, 2012. https://github.com/plamoni/.

[245] The iPhone Wiki. Theos, . URL http://iphonedevwiki.net/index.php/Theos.

[246] The iPhone Wiki. MobileSubstrate, . URL http://iphonedevwiki.net/index.php/Cydia_Substrate.

[247] Aubrey Derrick Schmidt and Sahin Albayrak. Malicious software for smartphones. Technical report, Technische Universitat Berlin - DAI-Labor, 2008.

[248] Nicolas Seriot. iPhone Privacy. In *Black Hat*, 2010.

[249] Francesco Di Cerbo, Andrea Girardello, Florian Michahelles, and Svetlana Voronkova. Detection of malicious applications on android OS. In *4th International Workshop on Computational Forensics*, 2010.

[250] Peter Koch. Confidentiality Aspects of DNS Data, Publication, and Resolution, 2013. URL https://tools.ietf.org/html/draft-koch-perpass-dns-confidentiality-00.