



ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΙΓΑΙΟΥ
ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΑΚΩΝ ΚΑΙ ΕΠΙΚΟΙΝΩΝΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ

ΑΣΦΑΛΕΙΑ ΠΛΗΡΟΦΟΡΙΑΚΩΝ ΚΑΙ ΕΠΙΚΟΙΝΩΝΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

**“Σχεδίαση & Υλοποίηση ενός Ασφαλούς Συστήματος CI/CD
Βασισμένου σε Δοχεία”**

**“Design and Implementation of a Secure Container-Based CI/CD
System”**

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

ΤΡΙΑΝΤΑΦΥΛΛΟΠΟΥΛΟΥ Λάμπρου

Επιβλέπων : Δρ. Κρητικός Κυριάκος

Μέλη εξεταστικής επιτροπής:

Κ. Κρητικός (Αναπλ. Καθηγητής), Σ. Κοκολάκης (Καθηγητής), Χ. Σκιάνης (Καθηγητής)

Σάμος, Φεβρουάριος 2024

Πρόλογος και ευχαριστίες

Η εκπόνηση της παρούσας διπλωματικής εργασίας πραγματοποιήθηκε στο πλαίσιο του Προγράμματος Μεταπτυχιακών Σπουδών στην Ασφάλεια Πληροφοριακών και Επικοινωνιακών Συστημάτων του Τμήματος Μηχανικών Πληροφοριακών και Επικοινωνιακών Συστημάτων του Πανεπιστημίου Αιγαίου.

Ξεκινώντας τη Διπλωματική αυτή εργασία, θα ήθελα να ευχαριστήσω από την καρδιά μου τη σύζυγό μου Σοφία, η οποία με ενθάρρυνε να συμμετάσχω σε ένα τόσο απαιτητικό μεταπτυχιακό πρόγραμμα σπουδών, γνωρίζοντας τις ώρες και την ενέργεια που αυτό θα απαιτούσε για να ολοκληρωθεί σε καθεστώς πλήρους φοίτησης, όπως και έγινε.

Στη συνέχεια θα ήθελα να ευχαριστήσω των επιβλέποντα καθηγητή μου Δρ. Κυριάκο Κρητικό, για την καθοδήγηση και ευκαιρία που έδωσε να αναλάβω μια τόσο ενδιαφέρουσα και τεχνολογικά αιχμηρή διπλωματική εργασία, η οποία αναμφίβολα θα με ανελίξει επαγγελματικά.

Συνολικά θέλω να ευχαριστήσω όλη την κοινότητα των καθηγητών και φοιτητών του Μεταπτυχιακού προγράμματος, για την ώθηση που μου δόθηκε να ακολουθήσω νέες κατευθύνσεις που θα εξελίσσουν την καριέρα μου, για τα ερεθίσματα αλλά και την υποστήριξη που έλαβα.

© 2024

του

Λάμπρου Τριανταφυλλόπουλου

Τμήμα Μηχανικών Πληροφοριακών και Επικοινωνιακών Συστημάτων

ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΙΓΑΙΟΥ

Πίνακας περιεχομένων

1 Εισαγωγή.....	11
1.1 Υφιστάμενη κατάσταση.....	11
1.2 Αντικείμενο διπλωματικής.....	14
1.3 Περιορισμοί & Προκλήσεις.....	14
1.4 Δομή της διπλωματικής.....	15
2 Υπόβαθρο.....	17
2.1 DevOps.....	17
2.2 DevSecOps.....	18
2.3 CI/CD.....	20
2.3.1 Συνεχής Ενσωμάτωση - Continuous Integration.....	20
2.3.2 Συνεχής Παράδοση - Continuous Delivery.....	21
2.3.3 Συνεχής Εγκατάσταση - Continuous Deployment.....	21
2.3.4 Πλατφόρμες CI/CD.....	22
2.3.5 Πλατφόρμα Gitlab.....	26
2.4 Εικονικοποίηση - Virtualization.....	29
2.4.1 Εικονικές Μηχανές - Virtual Machines (VMs).....	29
2.4.2 Δοχεία - Containers.....	30
3 Σχετικές Εργασίες.....	32
3.1 Εργασίες περί DevOps.....	32
3.2 Εργασίες σκλήρυνσης αγωγών CI/CD.....	33
3.3 Σύγκριση με τη Διπλωματική εργασία.....	36
4 Απειλές.....	38
4.1 Απειλές ενάντια σε γενικού τύπου συστήματα CI/CD.....	38
4.1.1 Παθητικοί/Ενεργητικοί Κρυφακουστές (Eavesdroppers).....	38
4.1.2 Μη εξουσιοδοτημένη πρόσβαση στο σύστημα.....	39
4.1.3 Εκτέλεση αγωγού από χρήστη χωρίς την ανάλογη εξουσιοδότηση.....	39
4.1.4 Τροποποίηση του αγωγού από απλό χρήστη.....	40
4.1.5 Αποθήκευση συνθηματικών σε μορφή κειμένου.....	40
4.1.6 Άρνηση υπηρεσίας (Denial of Service).....	40
4.1.7 Εκτέλεση και εγκατάσταση παλαιού στιγμιότυπου του κώδικα.....	41
4.1.8 Τροποποίηση του αγωγού/Πρόσβαση συνθηματικών από λογαριασμό διαχειριστή.....	41
4.1.9 Θέαση συνθηματικών πρόσβασης σε τρίτα συστήματα από απλούς χρήστες.....	41
4.1.10 Εκτέλεση απομακρυσμένου κώδικα στον εξυπηρετητή CI/CD.....	42
4.1.11 Φόρτωση κακόβουλων εκτελέσιμων/βιβλιοθηκών στο αποθετήριο.....	42
4.1.12 Χρήση κακόβουλου κώδικα από τρίτες βιβλιοθήκες.....	42
4.1.13 Έγχυση κακόβουλων βιβλιοθηκών.....	43
4.1.14 Μη εξουσιοδοτημένη πρόσβαση στο δοκιμαστικό/παραγωγικό περιβάλλον.....	43
4.1.15 Υποβολή κακόβουλου κώδικα από εξουσιοδοτημένο χρήστη.....	43
4.1.16 Υποβολή ευάλωτου κώδικα από εξουσιοδοτημένο χρήστη.....	44

4.1.17 Ευάλωτοι εξυπηρετητές.....	44
4.2 Ειδικές απειλές ενάντια σε συστήματα που βασίζονται σε δοχεία (Containers).....	45
4.2.1 Επικίνδυνα διαμορφωμένες εικόνες δοχείων.....	46
4.2.2 Ευάλωτος εξυπηρετητής κατά την οικοδόμηση του δοχείου.....	49
4.2.3 Επίθεση εφοδιαστικής αλυσίδας (Supply Chain Attack).....	49
4.2.4 Επικίνδυνες ρυθμίσεις κατά την εκτέλεση του δοχείου.....	50
4.2.5 Εκτεθειμένα μυστικά/συνθηματικά.....	53
4.2.6 Μη ασφαλής δικτύωση.....	53
4.3 Καταγραφίσεις επιθέσεις.....	54
4.3.1 Στόχος: Πηγαίος κώδικας της PHP Απειλή: Μη εξουσιοδοτημένη πρόσβαση στο σύστημα.....	54
4.3.2 Στόχος: Mercedes - Benz Απειλή: Μη εξουσιοδοτημένη πρόσβαση στο σύστημα & Αποθήκευση συνθηματικών σε μορφή κειμένου.....	55
4.3.3 Στόχος: Homebrew package manager για MACOS Απειλή: Εκτέλεση αγωγού από απλό χρήστη.....	55
4.3.4 Στόχος: Η βιβλιοθήκη rc του αποθετηρίου npm Απειλή: Χρήση κακόβουλου κώδικα από τρίτες βιβλιοθήκες & Μη εξουσιοδοτημένη πρόσβαση στο σύστημα.....	56
4.3.5 Στόχος: Samsung Απειλή: Αποθήκευση συνθηματικών σε μορφή κειμένου & Μη εξουσιοδοτημένη πρόσβαση στο σύστημα/αποθετήριο.....	56
4.3.6 Στόχος: Nissan Απειλή: Μη εξουσιοδοτημένη πρόσβαση στο σύστημα.....	56
4.3.7 Στόχος: CodeCov, Atlassian, Kubernetes, Python Απειλή: Επίθεση εφοδιαστικής αλυσίδας (Supply Chain Attack) & Αποθήκευση συνθηματικών σε μορφή κειμένου & Φόρτωση κακόβουλων εκτελέσιμων/βιβλιοθηκών στο αποθετήριο.....	56
4.3.8 Στόχος: SolarWinds, State Department, NSA, Intel, Microsoft, Cisco Απειλή: Επίθεση εφοδιαστικής αλυσίδας (Supply Chain Attack) & Φόρτωση κακόβουλων εκτελέσιμων/βιβλιοθηκών στο αποθετήριο.....	57
5 Ανάπτυξη Συστήματος.....	58
5.1 Απαιτήσεις του Συστήματος.....	58
5.1.1 Μη λειτουργικές απαιτήσεις.....	58
5.1.2 Λειτουργικές απαιτήσεις.....	60
5.2 Σχεδιασμός του Συστήματος.....	63
5.2.1 Σχεδιασμός Λογισμικού.....	63
5.2.1.1 Σχεδιασμός Επέκτασης Πλατφόρμας Gitlab.....	63
5.2.1.2 Σχεδιασμός Αγωγού CI/CD.....	63
5.2.2 Σχεδιασμός Υλικού.....	64
5.2.3 Αφηρημένη Αρχιτεκτονική Συστήματος.....	65
5.3 Υλοποίηση.....	66
5.3.1 Υλοποίηση Λογισμικού.....	67
5.3.1.1 Στάδιο: Δοκιμή κώδικα (Test Code).....	67
5.3.1.2 Στάδιο: Έλεγχος Κώδικα (Scan Code).....	70
5.3.1.3 Στάδιο: Οικοδόμηση Εικόνας δοχείου (Build Image).....	75
5.3.1.4 Στάδιο: Έλεγχος εικόνας (Scan Image).....	81

5.3.1.5 Στάδιο: Προώθηση εικόνας (Push Image).....	85
5.3.1.6 Στάδιο: Έλεγχος προωθημένης εικόνας (Scan Pushed Image).....	86
5.3.1.7 Στάδιο: Υπογραφή εικόνας (Sign Image).....	91
5.3.1.8 Στάδιο: Εγκατάσταση Δοχείων (Deploy Containers).....	92
5.3.1.9 Χειροκίνητο Στάδιο: Δόμηση προφίλ SECCOMP (Seccomp profiling).....	96
5.3.2 Υλοποίηση υλικού, υπερεπόμενη και λειτουργικών συστημάτων.....	100
5.3.2.1 Πρόσβαση μέσω ssh.....	101
5.3.2.2 Αξιολόγηση ασφαλείας με το εργαλείο Lynis.....	101
5.3.2.3 Εγκατάσταση ανιχνευτή κακόβουλου λογισμικού ClamAV.....	102
5.3.2.4 Χρήση αναχώματος ασφαλείας (Firewall) και συστήματος πρόληψης εισβολής (Intrusion Prevention System).....	103
5.3.3 Υλοποίηση της Αρχιτεκτονικής του Συστήματος.....	103
5.3.3.1 Αρχιτεκτονική του συστήματος Gitlab.....	103
5.3.3.2 Αρχιτεκτονική της συστοιχίας Kubernetes & Διασύνδεση με αγωγό.....	105
5.3.4 Υλοποίηση συστήματος παρακολούθησης & ειδοποίησης.....	112
5.3.4.1 Γενικά.....	112
5.3.4.2 Μέτρα παρακολούθησης και ειδοποίησης της συστοιχίας Kubernetes.....	112
6 Εγκατάσταση, Διαμόρφωση & Επίδειξη Συστήματος.....	120
6.1 Εγκατάσταση.....	120
6.1.1 Εγκατάσταση υπερεπόμενη Proxmox.....	120
6.1.2 Εγκατάσταση Gitlab Server.....	120
6.1.2.1 Εγκατάσταση podman & podman-compose.....	120
6.1.2.2 Εκτέλεση αρχείου Podman-Compose.yaml.....	120
6.1.3 Εγκατάσταση Gitlab Runner ως δοχείο.....	121
6.1.3.1 Εγκατάσταση Docker & ρυθμίσεις χρηστών.....	121
6.1.3.2 Εγκατάσταση δοχείου Gitlab Runner & σύνδεση με Gitlab Server.....	121
6.1.4 Λήψη της εφαρμογής Java που περιγράφει τον αγωγό.....	121
6.1.5 Εγκατάσταση SonarQube με τη μορφή δοχείου.....	121
6.1.6 Εγκατάσταση συστοιχίας Kubernetes.....	121
6.1.7 Διασύνδεση Gitlab Server & Kubernetes.....	121
6.1.8 Εργαλείο ασφαλείας συστοιχίας Kubernetes Kyverno.....	121
6.1.8.1 Για την εγκατάσταση του Kyverno.....	121
6.1.8.2 Για την επιλογή των πολιτικών ασφαλείας που θα εφαρμοστούν έγινε χρήση του παρακάτω συνδέσμου:.....	122
6.1.9 Εργαλεία Prometheus, Grafana & Alert Manager.....	122
6.1.10 Εργαλείο καταγραφής περιστατικών ασφαλείας Falco.....	122
6.1.11 Λογισμικό IPS για προστασία των εικονικών μηχανών.....	122
6.2 Ασφαλής διαμόρφωση του έργου λογισμικού στο Gitlab.....	122
6.3 Σενάρια χρήσης.....	127
6.3.1 Απώρριψη του λογισμικού/δοχείου από τον αγωγό.....	129
6.3.1.1 Σφάλματα λόγω αποτυχίας κατά το στάδιο δοκιμών του κώδικα.....	129
6.3.1.2 Σφάλματα λόγω αποτυχίας κατά το στάδιο ελέγχου του κώδικα.....	130
6.3.1.3 Σφάλματα λόγω αποτυχίας κατά το στάδιο ελέγχου της εικόνας.....	133
6.3.1.4 Σφάλματα λόγω αποτυχίας κατά το στάδιο ελέγχου της προωθημένης εικόνας.....	135

6.3.1.5 Σφάλματα σε λοιπές ενέργειες.....	137
6.3.2 Απόρριψη του δοχείου από τους κανόνες της συστοιχίας Kubernetes.....	137
6.3.2.1 Απόρριψη εγκατάστασης δοχείου με τη σημαία privileged.....	137
6.3.2.2 Απόρριψη εγκατάστασης δοχείου που εκτελείται από το χρήστη root.....	137
6.3.3 Έγκριση του δοχείου.....	138
7 Συμπεράσματα και μελλοντική εργασία.....	139
7.1 Κάλυψη απαιτήσεων και προστασία από απειλές.....	139
7.1.1 Κάλυψη μη λειτουργικών απαιτήσεων.....	139
7.1.2 Κάλυψη λειτουργικών απαιτήσεων.....	141
7.1.3 Προστασία ενάντια σε καταγεγραμμένες απειλές.....	143
7.2 Αποτίμηση & Μελλοντική εργασία.....	147
Βιβλιογραφία.....	149

Λίστα Εικόνων

Εικόνα 1 - Μοντέλο Καταρράκτη	11
Εικόνα 2 - Μοντέλο Agile.....	12
Εικόνα 3 - DevOps.....	18
Εικόνα 4 - Στάδια DevSecOps.....	19
Εικόνα 5 - Continuous Integration/Continuous Delivery & Deployment	22
Εικόνα 6 - Δημοφιλή πλατφορμών CI/CD	23
Εικόνα 7 - Απεικόνιση των Stages & Jobs στο Gitlab UI.....	28
Εικόνα 8 - Εποπτεία του αγωγού στο Gitlab UI.....	28
Εικόνα 9 - Κειμενογράφος αγωγών στο Gitlab UI.....	29
Εικόνα 10 - Εικονικές Μηχανές & Δοχεία	31
Εικόνα 11 - Βασική εικόνα.....	46
Εικόνα 12 - Επίσημη εικόνα οργανισμού.....	46
Εικόνα 13 - Επιδιόρθωση ευπαθειών σε βασική εικόνα.....	47
Εικόνα 14 - Multi-Stage Dockerfile.....	48
Εικόνα 15 - Non-root user στο Dockerfile.....	49
Εικόνα 16 - Δυνατότητες μη προνομιούχου (unprivileged) δοχείου.....	50
Εικόνα 17 - Δυνατότητες προνομιούχου (privileged) δοχείου.....	51
Εικόνα 18 - Σύνδεση φακέλων δοχείου.....	51
Εικόνα 19 - Σύνδεση του Docker Daemon socket με δοχείο.....	52
Εικόνα 20 - Δικτυακοί κανόνες επικοινωνίας δοχείων.....	54
Εικόνα 21 - Αφηρημένη αρχιτεκτονική συστήματος.....	66
Εικόνα 22 - Αποτελέσματα δοκιμών μονάδας και δοκιμών ενοποίησης.....	70
Εικόνα 23 - Αποτελέσματα ελέγχου κάλυψης κώδικα.....	70
Εικόνα 24 - Αγωγός CI/CD.....	99
Εικόνα 25 - Περιβάλλον χρήστη Proxmox.....	100
Εικόνα 26 - Κανόνες πρόσβασης ssh.....	101
Εικόνα 27 - Αποτελέσματα 1ου ελέγχου του εργαλείου Lynis.....	101
Εικόνα 28 - Αποτελέσματα 2ου ελέγχου του εργαλείου Lynis.....	102
Εικόνα 29 - Αποτελέσματα ημερήσιας σάρωσης με το εργαλείο ClamAV.....	102
Εικόνα 30 - Διάγραμμα φυσικής διάταξης Πλατφόρμας Gitlab & Gitlab Runners.....	105
Εικόνα 31 - Αρχιτεκτονική Gitlab - Kubernetes.....	108
Εικόνα 32 - Συλλογή δεδομένων από το Prometheus.....	113
Εικόνα 33 - Γραφήματα του εργαλείου Grafana.....	114
Εικόνα 34 - UI του Alert Manager σε περίπτωση παραβίασης κανόνα.....	116
Εικόνα 35 - Ειδοποίηση για παραβίαση ορίου μνήμης δοχείου από το Grafana.....	117
Εικόνα 36 - Ειδοποίηση για προσπέλαση του ευρετηρίου /etc/ από το Grafana.....	118
Εικόνα 37 - Συνολική αρχιτεκτονική συστήματος.....	119
Εικόνα 38 - Επιλογές για εγγραφή νέων χρηστών.....	123
Εικόνα 39 - Επιλογές για τη σύνδεση των χρηστών.....	123
Εικόνα 40 - Άμυνα εναντίον επιθέσεων ωμής βίας.....	124
Εικόνα 41 - Άμυνα εναντίον επιθέσεων εξάντλησης πόρων.....	124

Εικόνα 42 - Προστασία του κύριου κλαδιού μιας εφαρμογής.....	125
Εικόνα 43 - Ορισμός προστατευμένου αποθετηρίου αγωγού.....	126
Εικόνα 44 - Προστασία των μυστικών.....	126
Εικόνα 45 - Προστασία από επανεκτέλεση παλαιού αγωγού.....	127
Εικόνα 46 - Δημιουργία Αιτήματος Συγχώνευσης.....	127
Εικόνα 47 - Καταχώρηση στοιχείων αιτήματος συγχώνευσης.....	128
Εικόνα 48 - Εποπτεία αλλαγών.....	128
Εικόνα 49 - Αποδοχή συγχώνευσης από τον υπεύθυνο.....	128
Εικόνα 50 - Κατάσταση αγωγού του συγχωνευμένου κλαδιού.....	129
Εικόνα 51 - Αποτυχία δοκιμών μονάδας.....	129
Εικόνα 52 - Αποτυχία ποσοστού κάλυψης κώδικα.....	130
Εικόνα 53 - Αποτυχία δοκιμών ενοποίησης.....	130
Εικόνα 54 - Αποτυχία ελέγχου στατικής ανάλυσης.....	131
Εικόνα 55 - Ανάλυση αποτελεσμάτων SonarQube.....	131
Εικόνα 56 - Αποτυχία κατά την ανίχνευση μυστικών.....	132
Εικόνα 57 - Αποτυχία κατά τον έλεγχο ευπαθειών βιβλιοθηκών.....	133
Εικόνα 58 - Αναφορά ελέγχου ευπαθειών βιβλιοθηκών.....	133
Εικόνα 59 - Αποτυχία κατά τον έλεγχο του Dockerfile από το Hadolint.....	134
Εικόνα 60 - Αποτυχία κατά τον έλεγχο του Dockerfile για χρήστη root.....	134
Εικόνα 61 - Αποτυχία κατά τον έλεγχο ευπαθειών σε βιβλιοθήκες/πακέτα.....	135
Εικόνα 62 - Σελίδα αποτελεσμάτων ελέγχου SCAP.....	136
Εικόνα 63 - Αποτυχία κατά τον έλεγχο ρυθμίσεων εγκατάστασης δοχείων.....	136
Εικόνα 64 - Αποτυχία κατά την εγκατάσταση λόγω της σημαίας privileged.....	137
Εικόνα 65 - Αποτυχία κατά την εγκατάσταση λόγω χρήστη root.....	138
Εικόνα 66 - Εγκατάσταση δοχείου.....	138

Λίστα Πινάκων

Πίνακας 1 - Σύγκριση πλατφορμών CI/CD	25
Πίνακας 2 - Κάλυψη μη λειτουργικών Απαιτήσεων.....	139
Πίνακας 3 - Κάλυψη λειτουργικών Απαιτήσεων.....	141
Πίνακας 4 - Προστασία από απειλές.....	143

Ακρωνύμια

BDST	Behaviour Driven Security Testing
CD	Continuous Delivery
CI	Continuous Integration
CRI	Container Runtime Interface
DAST	Dynamic Application Security Testing
DevOps	Development - Operations
DevSecOps	Development - Security - Operations
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
MFA	Multi Factor Authentication
OS	Operating System
OSINT	Open Source Intelligence
RBAC	Role Based Access Control
SAS	Security API Scanning
SaaS	Software as a Service
SCM	Source Code Manager
SSH	Secure Socket Shell
STRIDE	Spoofing, Tampering, Repudiation, Information disclosure, Denial of service, Elevation of privilege
UI	User Interface
SDLC	Software Development Lifecycle
WAST	Web Application Security Scanning
VCS	Version Control System
VM	Virtual Machine

Περίληψη

Στο σύγχρονο περιβάλλον ανάπτυξης εφαρμογών, με δεδομένο τον υψηλό ανταγωνισμό που υφίσταται ανάμεσα στους παρόχους εφαρμογών & ψηφιακών υπηρεσιών, έχει θεσπιστεί ως θεμελιώδης ανάγκη η ταχεία και αυτοματοποιημένη δημιουργία, παράδοση και εγκατάσταση κώδικα.

Με τη σταδιακή εγκατάλειψη του μοντέλου καταρράκτη για την ανάπτυξη λογισμικού, και την εξάπλωση του ευέλικτου μοντέλου ανάπτυξης (agile software development model), η ανάγκη για συχνή και σταδιακή ενσωμάτωση αλλαγών στον κώδικα απαίτησε τη συσσωμάτωση δυο παραδοσιακά διαφορετικών χώρων της βιομηχανίας ανάπτυξης λογισμικού, αυτόν της ανάπτυξης κώδικα και αυτόν της λειτουργίας (operations) του κώδικα σε έναν ενιαίο, με τίτλο devops.

Στον πυρήνα της λειτουργίας του devops βρίσκεται η έννοια των αγωγών (pipelines), μέσα από τους οποίους εκτελούνται οι κρίσιμες λειτουργίες της συνεχούς ενσωμάτωσης και συνεχούς παράδοσης (continuous integration & continuous delivery - CI/CD). Με αυτόν τον τρόπο, ο κώδικας αποθηκεύεται, ελέγχεται, δοκιμάζεται και τελικά διανέμεται στους καταναλωτές σε περιβάλλοντα παραγωγής.

Γίνεται μεμιάς φανερό, ότι οποιαδήποτε επίθεση στους CI/CD αγωγούς, θέτει σε κίνδυνο όχι μόνο τους καταναλωτές του λογισμικού, αλλά και τις υποδομές πάνω από τις οποίες εκτελούνται οι αγωγοί. Στο πλαίσιο της παρούσας διπλωματικής εργασίας, θα γίνει μια βασική αναφορά στις τεχνολογίες που συνθέτουν το σύγχρονο περιβάλλον του κύκλου ζωής λογισμικού, θα αναλυθούν οι κίνδυνοι που ελλοχεύουν μέσα και έξω από τους αγωγούς, και τελικά θα προταθεί ένα σύστημα CI/CD βασισμένο σε δοχεία (containers), ικανό να προστατέψει τον κώδικα και τις υποδομές από επίδοξους εισβολείς.

Λέξεις Κλειδιά: *devops, CI/CD, δοχεία, αγωγοί, ασφάλεια*

Abstract

The current trends and standards of software development, given the high competition between the providers of applications and digital services, are all about rapid and automated code creation and installation.

While the classic waterfall model of software development was being driven out, and its place ceded to the agile model, the need for frequent and gradual code implementation warranted the unification of two different fields of the software lifecycle, those of development and operations, under the newly created space of devops.

At its core, one can find the notion of pipelines, constructs through which the pillars of devops, i.e., continuous integration and continuous delivery (CI/CD) are given shape and form. Alongside these pipelines, the code is stored, verified, tested and finally delivered to its consumers (in production environments).

It becomes immediately obvious, that any successful attack upon the CI/CD pipelines, endangers not only the software consumers, but the infrastructure on which the pipelines are executed as well. As a part of this Master’s thesis, a brief introduction to the basic notions of current software development will be supplied, a catalogue of threats against pipelines will be defined, and finally we will propose a container-based CI/CD system, capable of fending off invaders while protecting the code, the pipelines and their infrastructure.

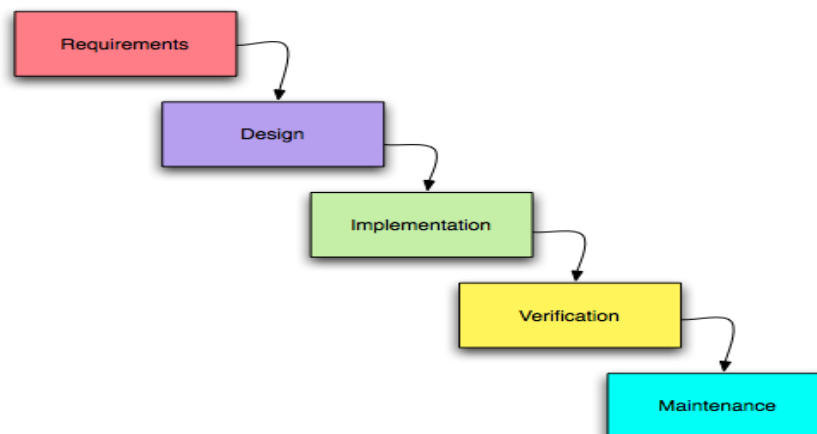
Keywords: *devops, CI/CD, containers, pipelines, security*

1

Εισαγωγή

1.1 Υφιστάμενη κατάσταση

Ο στόχος με το μεγαλύτερο βάρος σε έναν οργανισμό που παρέχει υπηρεσίες λογισμικού, είναι η πλέον ταχεία και αποτελεσματική συγγραφή κώδικα, η οποία οδηγεί σε πιο ανταγωνιστικά προϊόντα για τους πελάτες του. Η άμεση ενσωμάτωση νέων λειτουργιών και η ανατροφοδότηση από τους χρήστες οδήγησε τους οργανισμούς να εγκαταλείψουν τα παραδοσιακά μοντέλα τύπου καταρράκτη [1], τα οποία απαιτούσαν το σύνολο των απαιτήσεων να είναι γνωστά στη φάση του σχεδιασμού, ενώ παράλληλα ήταν εξαιρετικά δυσκίνητα σε περίπτωση αλλαγής των αρχικών απαιτήσεων [2].



Εικόνα 1 - Μοντέλο Καταρράκτη
(Πηγή: https://commons.wikimedia.org/wiki/File:Waterfall_model.png)

Ως συνέπεια αυτής της ανάγκης για ευελιξία, μέσα στο χώρο της δημιουργίας λογισμικού δημιουργήθηκε και το ομώνυμο νέο κύμα της ευέλικτης ανάπτυξης (agile development) [3], σύμφωνα με το οποίο ο κύκλος ζωής του λογισμικού διαιρείται σε υπο-περιόδους των 2 -συνήθως- εβδομάδων (sprints), μέσα στους οποίους καθορίζονται μεσοπρόθεσμοι στόχοι προς

υλοποίηση και ενσωμάτωση. Με τη συντόμευση και επαναληψιμότητα του κύκλου ζωής του λογισμικού, η ικανότητα του οργανισμού για αλλαγή και προσαρμογή γνώρισε νέα μεγέθη, αδύνατα μέχρι τότε [4].



Εικόνα 2 - Μοντέλο Agile (Πηγή:

https://www.researchgate.net/figure/Agile-Methodology-in-System-Development-source-Okeke2021-retrieved-from_fig1_354310848)

Για την περαιτέρω επιτάχυνση και βελτίωση της συνεργασίας μεταξύ των τμημάτων ενός οργανισμού, ο χώρος του λογισμικού γνώρισε μια ακόμα μεγάλη αλλαγή που ακούει στο όνομα *devops* [5]. Στον όρο αυτό περιλαμβάνεται η ενοποίηση δυο παραδοσιακά διαφορετικών χώρων, αυτών της ανάπτυξης λογισμικού (*development*) και εκείνου της λειτουργίας του λογισμικού (*operations*), που περιλαμβάνει, εκτός από την κατάλληλη λειτουργία του λογισμικού, την διαχείριση των υποδομών και των περιβαλλόντων πάνω από τα οποία εκτελείται το λογισμικό. Μέχρι τότε, οι μηχανικοί και οι operators (δηλ. Οι υπεύθυνοι της λειτουργίας του λογισμικού) δε συνδέονταν επιχειρησιακά, παρά μόνο με ένα μηχανισμό *ticketing* [75]. Αυτή η αλλαγή επέτρεψε στην παραγωγή του νέου κώδικα να συνδέεται στενά με την εγκατάσταση και διάθεσή του σε διαφορετικά περιβάλλοντα (ανάπτυξης, δοκιμής και παραγωγής) και άρα ακόμη και στους καταναλωτές [6]. Αυτή η διαδικασία, σε συνδυασμό με τη ταχεία ενσωμάτωση νέου κώδικα είναι γνωστή με τα αρχικά CI/CD (*Continuous Integration/ Continuous Delivery*) (Συνεχής Ενοποίηση / Συνεχής Παράδοση).

Ο ζωτικός κύκλος της συνεχούς ενσωμάτωσης και παράδοσης πραγματοποιείται μέσα από αυτοματοποιημένες ροές εργασίας, ονοματιζόμενες αγωγοί (*pipelines*), οι οποίες επιτρέπουν στον πολύτιμο χρόνο του ανθρώπινου δυναμικού να αξιοποιηθεί αποδοτικότερα σε άλλες διαδικασίες (δηλ. κυρίως στην ανάπτυξη του λογισμικού). Οι αγωγοί αποτελούνται από μικρότερα στάδια (*stages*), το καθένα από τα οποία είναι υπεύθυνο για ένα διαφορετικό χειρισμό του πηγαίου

κώδικα. Αυτά τα στάδια έχει καθιερωθεί να περιλαμβάνουν την κλωνοποίηση του τροποποιημένου κώδικα στο τρέχον περιβάλλον, την δοκιμή του με δοκιμές όπως: (Unit tests, Static Application Security Testing - SAST, Dynamic Application Security Testing - DAST, Integration tests), την οριστική ενσωμάτωση των αλλαγών σε κάποιο κατάλληλο κλαδί (branch) του κώδικα στο σύστημα διαχείρισης διαμόρφωσης και τελικά την παράδοση/εγκατάσταση του. Τα παραπάνω στάδια μπορούν να εκτελεστούν σε πληθώρα διαφορετικών τεχνολογιών, και να αλληλεπιδράσουν με όλους τους παρόχους υπηρεσιών νέφους, προσφέροντας μια ενοποιημένη διαχείριση στο προσωπικό του devops [7].

Σε αυτό το σημείο, πρέπει βέβαια να σημειωθεί ότι για όλα υπάρχει ένα αντίτιμο. Στην περίπτωση του devops, το αντίτιμο ήταν η μείωση του επιπέδου ασφαλείας στην παραγωγή κώδικα. Προηγουμένως, στο ευέλικτο μοντέλο ανάπτυξης, το αρμόδιο τμήμα ασφάλειας του εκάστοτε οργανισμού ήλεγχε τον κώδικα και τις υποδομές μετά την υλοποίηση του κώδικα και πριν την εγκατάσταση του. Με τους καινοφανείς, όμως, και σύντομους χρόνους παράδοσης στην εποχή του devops, αυτό ο τρόπος ελέγχου και παράδοσης του λογισμικού δεν ήταν πλέον βιώσιμος. Επιπρόσθετα, η οποιαδήποτε παρέμβαση της ασφάλειας κατά τη φάση της υλοποίησης παρουσιαζόταν συχνά σαν εμπόδιο στην παραγωγικότητα των μηχανικών λογισμικού [76].

Με την κουλτούρα devops να έχει υιοθετηθεί από το 74% των οργανισμών πληροφορικής [8], την ελεύθερη πλέον πρόσβαση σε πληροφορίες, τεχνικές, εκμετάλλευση ευπαθειών, πληροφορίες ανοικτών πηγών (OSINT), malware (κακόβουλο λογισμικό) και exploits (κώδικας/script εκμετάλλευσης ευπαθειών), καθώς και πληθώρα επιτυχημένων επιθέσεων σε pipelines οργανισμών με κόστος εκατομμυρίων δολαρίων [9], ο χώρος της πληροφορικής γρήγορα αντιλήφθηκε ότι η ασφάλεια έπρεπε να αποτελεί ένα οργανικό κομμάτι του devops, καθώς η ταχύτητα αλλαγής του κώδικα και των υποδομών δεν ήταν πλέον διαχειρίσιμη από παραδοσιακές προσεγγίσεις ασφαλείας. Έτσι, λοιπόν, γεννήθηκε ο χώρος του DevSecOps.

Η κεντρική φιλοσοφία του DevSecOps είναι η νοοτροπία της κύλισης προς τα αριστερά (shift left) [10]. Με άλλα λόγια, κηρύσσει ότι η ασφάλεια θα πρέπει να είναι εγγενές συστατικό ενός αγωγού λογισμικού, ρυθμισμένου κατάλληλα ώστε να ανιχνεύει τυχόν ευπάθειες πριν ο νέος κώδικας φτάσει στο στάδιο της παραγωγής. Επιπρόσθετα, ο αγωγός δε θα πρέπει απλά να ανιχνεύει ευπάθειες στον κώδικα, αλλά ταυτόχρονα να εντοπίζει ευπάθειες σε τρίτες βιβλιοθήκες, και να είναι και ο ίδιος σωστά ρυθμισμένος, ώστε να αποτυγχάνουν και οι επιθέσεις σε βάρος του ίδιου και του περιβάλλοντός του.

1.2 Αντικείμενο διπλωματικής

Όπως ήδη υπογραμμίσθηκε, ένας CI/CD αγωγός αποτελεί ίσως το πιο κρίσιμο συστατικό ενός οργανισμού παραγωγής λογισμικού, με δεδομένο ότι η οποιαδήποτε αστοχία σχετική με τον αγωγό μπορεί δυνητικά να προκαλέσει ανεπανόρθωτη ζημιά στη φήμη ή τα οικονομικά μεγέθη του οργανισμού αυτού. Ο κίνδυνος γίνεται ακόμα μεγαλύτερος εάν αναλογιστεί κανείς τη λειτουργικότητα που συγκεντρώνεται σε τόσο μικρό μηχανικό χώρο, σε συνδυασμό με το μεγάλο πλήθος των κινητών μερών αυτού του πολύπλοκου μηχανισμού: αποθετήρια, εξυπηρετητές, εικονικές μηχανές και δοχεία [11]. Με τόσες ετερόκλητες τεχνολογίες στο τεχνολογικό μείγμα, ο αγωγός πρέπει να είναι σε θέση να προστατέψει ανάλογα ευρεία επιφάνεια επίθεσης από τους επίδοξους εισβολείς.

Ο σκοπός της παρούσας διπλωματικής εργασίας είναι διπλός:

- Αρχικά θα γίνει μια αποτίμηση των κυριότερων κινδύνων που αντιμετωπίζει ένας αγωγός CI/CD, τόσο σε επίπεδο κώδικα, όσο και σε επίπεδο υποδομών/περιβαλλόντων, κατόπιν μελέτης της σχετικής βιβλιογραφίας και του τεχνολογικού περιβάλλοντος.
- Στη συνέχεια, βάσει της παραπάνω αποτίμησης θα σχεδιαστεί και θα υλοποιηθεί ένα σύστημα CI/CD βασισμένο σε δοχεία (containers), με σκοπό να είναι σε θέση να αμυνθεί ενάντια στις περιγραφείσες απειλές. Η τεχνολογία των δοχείων επιλέχθηκε καθώς αποτελούν ταχείες, αμετάβλητες και στεγανοποιημένες διεργασίες, δημιουργημένες για να εγκαθίστανται και να μεταφέρονται εύκολα. Θα δοθεί έμφαση όχι μόνο στην ασφάλεια του κώδικα, αλλά και του ίδιου του αντίστοιχου αγωγού και του περιβάλλοντός του με εφαρμογή των κατάλληλων μέτρων, εργαλείων και ρυθμίσεων, που θα επιτύχουν την απαραίτητη σκλήρυνση (hardening) του συστήματος συνολικά και αυτοματοποιημένα.

1.3 Περιορισμοί & Προκλήσεις

Οι περιορισμοί που περιχαράκωνουν την υλοποίηση του τεχνικού τμήματος της παρούσας διπλωματικής κυρίως έγκεινται στην ανάγκη του συστήματος να μην επιβαρύνει σε μεγάλο βαθμό τόσο χρονικά, όσο και υπολογιστικά (δηλ. ως προς την χρήση των υποκείμενων πόρων) τον αγωγό CI/CD. Η μετάβαση από μια παραδοσιακή προσέγγιση DevOps σε μια τύπου DevSecOps

απαιτεί την τροποποίηση μιας τυπικής αλυσίδας διαδικασιών και ενεργειών ενός αγωγού, και την ενσωμάτωση όλων εκείνων των στοιχείων ασφαλείας, προκειμένου αυτή να καταστεί ασφαλής [12].

Επιπρόσθετα, λόγω της μεγάλης ανάπτυξης του χώρου τόσο στο επίπεδο devops όσο και των δοχείων, παρουσιάζεται μεγάλη ετερογένεια στις διαθέσιμες τεχνολογίες και εργαλεία που χρειάζεται να διαλειτουργήσουν.

1.4 Δομή της διπλωματικής

Κεφάλαιο 2 - Υπόβαθρο

Σε αυτό το κεφάλαιο αναλύονται οι κατάλληλες έννοιες και ορισμοί που είναι απαραίτητοι για την καλύτερη κατανόηση του υπόλοιπου σκέλους της εργασίας.

Κεφάλαιο 3 - Σχετικές εργασίες

Σε αυτό το κεφάλαιο θα γίνει ανάλυση εργασιών σχετικών με την ανάπτυξη ασφαλών CI/CD συστημάτων και σύγκρισή τους με το σύστημα που αναπτύχθηκε στο πλαίσιο της παρούσας διπλωματικής εργασίας.

Κεφάλαιο 4 - Απειλές

Σε αυτό το κεφάλαιο θα γίνει αναγνώριση και απαρίθμηση των κυριότερων απειλών σε βάρος των αγωγών, καθώς και πραγματικών περιστατικών στα οποία κακόβουλοι χρήστες επέφεραν ζημιά μέσω αυτών.

Κεφάλαιο 5 - Ανάπτυξη συστήματος

Σε αυτό το κεφάλαιο αναλύονται όλα τα αποτελέσματα της εφαρμογής των δραστηριοτήτων της διαδικασίας ανάπτυξης του συστήματος, συμπεριλαμβανομένων των απαιτήσεων, σχεδιαστικών διαγραμμάτων αρχιτεκτονικής και βασικών λεπτομερειών υλοποίησης.

Κεφάλαιο 6 - Εγκατάσταση, Διαμόρφωση & Επίδειξη συστήματος

Σε αυτό το κεφάλαιο θα γίνει μια σύντομη παρουσίαση του τρόπου εγκατάστασης και εκτέλεσης του προτεινόμενου συστήματος ασφαλούς εκτέλεσης αγωγών καθώς και επίδειξη της λειτουργίας του με συγκεκριμένα σενάρια χρήσης.

Κεφάλαιο 7 - Συμπεράσματα και μελλοντική εργασία

Σε αυτό το κεφάλαιο θα συγκεντρωθούν τα συμπεράσματα που παρήχθησαν και οι εμπειρίες που αποκομίσθηκαν από την εκπόνηση της διπλωματικής εργασίας, καθώς επίσης θα αποτυπωθούν πιθανές μελλοντικές επεκτάσεις αυτής.

2

Υπόβαθρο

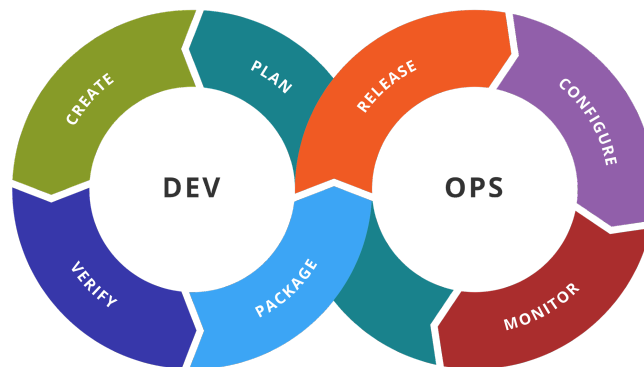
2.1 DevOps

Ο πρώτος που κατέγραψε την ανάγκη επινόησης ενός χώρου devops μέσα στον κύκλο ζωής της δημιουργίας λογισμικού ήταν ο Patrick Debois [13]. Η στεγανοποίηση ανάμεσα στα τμήματα της συγγραφής κώδικα και των λειτουργιών πληροφορικής, ειδικά μετά την εφαρμογή του ευέλικτου μοντέλου (agile) αναγνωρίστηκε ως δυσλειτουργική, με το υπάρχον σύστημα των αιτημάτων και στεγανοποίησης (compartmentalization) να αποδεικνύεται ανεπαρκές να ικανοποιήσει τις ολοένα πιο δυναμικές ανάγκες που προέκυπταν [13]. Μέχρι τότε, οι ομάδες των προγραμματιστών είχαν σαν κύριο μέλημα την ταχεία παραγωγή κώδικα και την υλοποίηση νέων λειτουργικοτήτων, ενώ το τμήμα των λειτουργιών στόχευε στη μέγιστη σταθερότητα και διαθεσιμότητα των συστημάτων. Αυτό ακριβώς το κενό γεφυρώθηκε με τη νέα εργασιακή φιλοσοφία devops [14], στην οποία η παραγωγή κώδικα και οι λειτουργίες (operations) συμβαίνουν παράλληλα.

Σε αδρές γραμμές, εντοπίζονται δύο βασικοί πυλώνες στους οποίους βασίζεται το μοντέλο devops: η συνεργασία και η αυτοματοποίηση [77]. Υπάρχουν αυτοί που υποστηρίζουν ότι με τον όρο devops περιγράφονται κυρίως οι εργασιακές σχέσεις, η κουλτούρα, η συνεργασία και η επικοινωνία ανάμεσα στο προσωπικό ενός οργανισμού. Η μείωση των διαδικαστικών καθυστερήσεων, άρα και η βελτίωση της συνεργασίας, μπορεί να επιτευχθεί με αμεσότερη επικοινωνία μέσω εξειδικευμένων προγραμμάτων, όπως π.χ. slack, jira, redmine ή τηλεδιασκέψεις, όπου απαιτείται [15]. Ανεξάρτητα από τα εργαλεία, ωστόσο, πρέπει πάντοτε να γίνεται ενεργή προσπάθεια για διαμοιρασμό της γνώσης και της επιτευξης των κοινών στόχων, ώστε το κάθε μέλος της ομάδας να αισθάνεται κοινωνός του ευρύτερου στόχου.

Αδιαμφισβήτητα όμως και πέρα από την ανθρώπινη διάσταση, στον πυρήνα του devops εντοπίζονται τεχνικά χαρακτηριστικά, διαδικασίες και τεχνικές, οι οποίες αποσκοπούν στην αυτοματοποίηση του κύκλου ζωής του λογισμικού, με τελικό προορισμό την έγκαιρη παροχή

υψηλής ποιότητας υπηρεσιών. Τα διαφορετικά στάδια, όπως η ανάπτυξη, η δοκιμή και η εγκατάσταση, εκτελούνται αυτόματα και επαναληπτικά, χωρίς την ανθρώπινη παρέμβαση, όπου αυτό είναι εφικτό και ασφαλές. Με αυτό τον τρόπο, εξοικονομείται πολύτιμος χρόνος, προκειμένου να αξιοποιηθεί σε άλλες κρίσιμες διαδικασίες, όπως είναι η ανάπτυξη νέων λειτουργιών ή η αποσφαλμάτωση. Σε αυτό το σημείο εισέρχεται η έννοια του CI/CD και των αγωγών (pipelines).



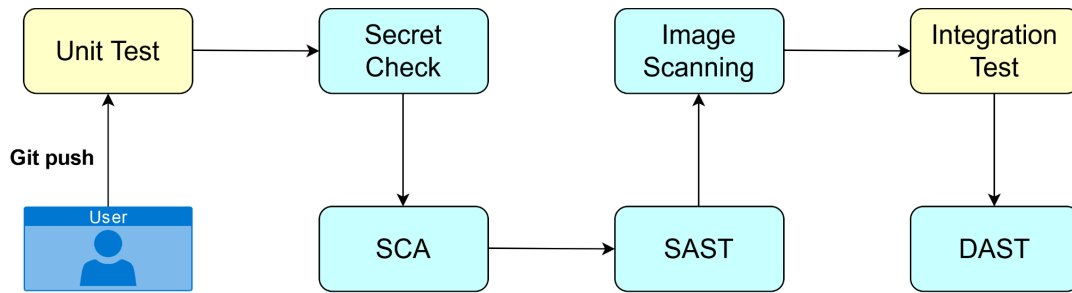
Εικόνα 3 - DevOps (Πηγή:

<https://upload.wikimedia.org/wikipedia/commons/thumb/0/05/Devops-toolchain.svg/2560px-Devops-toolchain.svg.png>)

2.2 DevSecOps

Ήδη αναπτύχθηκε η έννοια και η λειτουργία του DevOps, σύμφωνα με το οποίο ο κώδικας υπόκειται σε συνεχείς αλλαγές, οι οποίες αδιάλειπτα φτάνουν στην επιχειρησιακή λειτουργία ενός οργανισμού. Όμως, εμφανίστηκε και μια ανεπιθύμητη παρενέργεια: ο έλεγχος ασφαλείας από το αρμόδιο προσωπικό, ο οποίος παραδοσιακά λάμβανε χώρα μετά από τη συγγραφή κώδικα, πλέον δεν προλάβαινε να ακολουθήσει την καινούρια ταχεία ροή των διεργασιών CI/CD. Η φυσική εξέλιξη ήταν η “κύλιση προς τα αριστερά” του στοιχείου της ασφάλειας μέσα στον κύκλο ζωής του λογισμικού [19]. Σε πρακτικούς όρους, αυτό σήμαινε ότι ήταν πλέον επιβεβλημένο η διεργασία CI/CD του DevOps να εμπλουτιστεί με αυτοματοποιημένα εργαλεία που συμβάλλουν στην ασφάλεια του λογισμικού.

Στο DevSecOps, οι απαιτήσεις ασφαλείας πρέπει να τίθενται από τα αρμόδια τμήματα, με την κατάλληλη παραμετροποίηση του αγωγού. Ενώ ο κάθε οργανισμός συντηρεί τη δική του μεθοδολογία DevSecOps, ένα γενικό μοντέλο που περιχαράκώνει το παραγόμενο λογισμικό ακολουθεί τα εξής βήματα [20]:



Εικόνα 4 - Στάδια DevSecOps

Unit Test: Εκτελούνται αυτοματοποιημένες δοκιμές στις πιο απλές λειτουργίες του λογισμικού (πχ. με την μορφή μεθόδων και συναρτήσεων) για να ελεγχθεί ότι το σύνολο των λειτουργιών αυτών εκτελείται σωστά. Είναι τμήμα και ενός τυπικού DevOps αγωγού.

Secret Check: Ο κώδικας ελέγχεται για ύπαρξη τυχόν κωδικών ή συνθηματικών, τα οποία δεν επιτρέπεται να βρίσκονται διαθέσιμα μέσα στο αποθετήριο.

Software Composition Analysis (SCA): Ελέγχονται οι τρίτες βιβλιοθήκες που αξιοποιούνται από τον κώδικα για τυχόν ευπάθειες [84]. Αυτή η διαδικασία αναγνωρίζει τα συνθετικά μέρη μιας εφαρμογής είτε διαβάζοντας ένα αρχείο ρυθμίσεων όπως π.χ τύπου json, είτε αναγνωρίζοντας τις υπογραφές των διαφόρων πακέτων μέσα στο (βασικό) τέχνημα (της εφαρμογής). Στη συνέχεια, τα ευρήματα διασταυρώνονται με βάσεις αδειών χρήσης και ευπαθειών, για να κριθεί εάν η χρήση τους είναι νόμιμη και ασφαλής, σύμφωνα με την πολιτική ασφάλειας του οργανισμού.

Static Application Security Testing (SAST): Ελέγχεται ο πηγαίος κώδικας μέσω στατικών τεχνικών ανάλυσης για ευπάθειες και κακές πρακτικές (ανάπτυξης) που πρέπει να αποφεύγονται.

Image Scanning: Ελέγχεται για ευπάθειες τόσο η εικόνα βάσης ενός δοχείου, όσο και τυχόν πρόσθετα πακέτα σε αυτή, με τα αποτελέσματα να συγκρίνονται με γνωστές βάσεις ευπαθειών.

Dynamic application security testing (DAST): Εκτελούνται δοκιμές διείσδυσης (penetration tests) σε μια δοκιμαστική έκδοση του λογισμικού σε ένα περιβάλλον δοκιμής, προκειμένου να αναγνωριστούν ευπάθειες.

Integration Test: Εκτελούνται αυτοματοποιημένες δοκιμές για να πιστοποιηθεί ότι τα διακριτά μέρη του λογισμικού διαλειτουργούν σωστά καθώς και όλο το λογισμικό. Είναι τμήμα και ενός τυπικού DevOps αγωγού.

2.3 CI/CD

Ο αγωγός CI/CD είναι ο αυτοκινητόδρομος πάνω στον οποίο κινείται η διαδικασία ανάπτυξης λογισμικού στους σύγχρονους οργανισμούς. Αποτελείται από τα εξής τμήματα:

2.3.1 Συνεχής Ενσωμάτωση - Continuous Integration

Η συνεχής ενσωμάτωση είναι εκείνη η διαδικασία με την οποία οι μηχανικοί κώδικα υποβάλλουν τις προσθήκες/διορθώσεις που κάνουν στον πηγαίο κώδικα του λογισμικού. Γι αυτό το σκοπό είναι απαραίτητο να τηρείται ένα κεντρικό αποθετήριο κώδικα, το οποίο βασίζεται σε σύστημα διαχείρισης εκδόσεων git ή svn. Με την πλάστιγγα της αποδοχής να έχει γείρει δραματικά υπέρ του συστήματος git [80], είναι πλέον το πλέον αποδοτικό σύστημα διαχείρισης εκδόσεων. Οι σύγχρονες πρακτικές απαιτούν από τους μηχανικούς την ημερήσια - ή συχνότερη - υποβολή κώδικα, σε αντίθεση με παλαιότερες πρακτικές όπου σπανιότερα υποβάλλονταν ένα μεγάλο τμήμα κώδικα στο κεντρικό αποθετήριο.

Συνοπτικά η διαδικασία της ενσωμάτωσης νέου κώδικα σε αποθετήριο βασισμένο στο σύστημα git μπορεί να περιγραφεί ως εξής [81]:

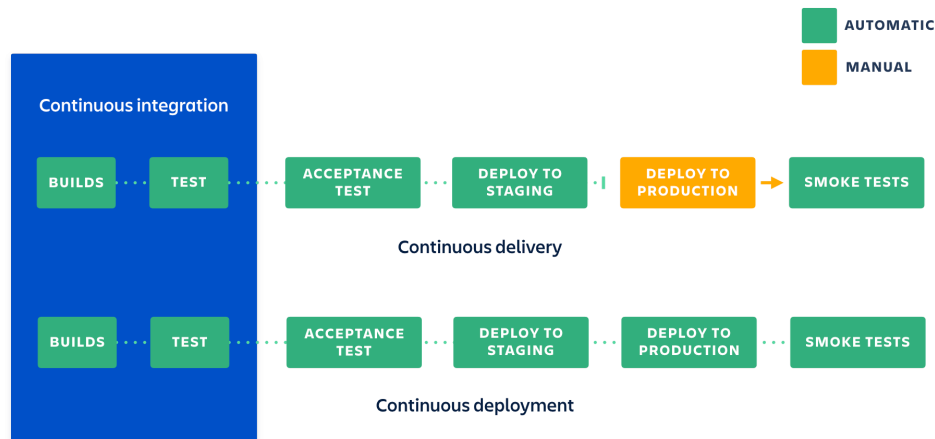
- Το αποθετήριο κάθε εφαρμογής αποτελείται από διακριτά αντίγραφα του πηγαίου κώδικα, τα οποία ονομάζονται κλαδιά (branches).
- Ο κώδικας που εκτελείται στο επιχειρησιακό περιβάλλον βρίσκεται σε ένα μη τροποποιήσιμο κύριο κλαδί που συνήθως ονομάζεται master/main. Σε αυτό το κλαδί απαγορεύεται η απευθείας τροποποίηση (από τους προγραμματιστές).
- Όταν παρουσιαστεί η ανάγκη, είτε λόγω προβλήματος (bug) ή λόγω νέων απαιτήσεων, για αλλαγή του κώδικα, οι μηχανικοί που αναλαμβάνουν την υλοποίηση δημιουργούν ένα νέο κλαδί για το στοιχείο που θα τροποποιηθεί ή θα αναπτυχθεί (feature branch).
- Ο τροποποιημένος κώδικας του νέου κλαδιού υποβάλλεται όσο πιο συχνά γίνεται στο αποθετήριο, με αποτέλεσμα να ενεργοποιούνται οι σχεδιασμένοι έλεγχοι επί αυτού.
- Όταν η υλοποίηση ολοκληρωθεί, δημιουργείται μια αίτηση συνένωσης του νέου κλαδιού με το κύριο κλαδί. Αφού όλοι οι έλεγχοι εκτελεστούν με επιτυχία, ο υπεύθυνος της ομάδας εγκρίνει το αίτημα συνένωσης, και πλέον το ενημερωμένο κύριο κλαδί (master/main) περνά στη φάση της εγκατάστασης.

2.3.2 Συνεχής Παράδοση - Continuous Delivery

Με την υποβολή του κώδικα στο αποθετήριο, αρχικά θα τρέξουν σαφώς καθορισμένες δοκιμές (Unit tests, Component tests, Integration tests), προκειμένου να ελεγχθεί η εγκυρότητά του και να επιλυθούν τυχόν προβλήματα αμέσως μετά τη συγγραφή [16]. Εφόσον ο κώδικας περάσει επιτυχώς τις δοκιμές, ενσωματώνεται στο τέχνημα (artifact) που προωθείται για εγκατάσταση στο δοκιμαστικό περιβάλλον, και απομένει μόνο μια χειροκίνητη έγκριση/δοκιμή για να προωθηθεί στην παραγωγή η νέα έκδοση. Εκτός του ίδιου του πηγαίου κώδικα, προϊόν της συνεχούς ενσωμάτωσης μπορεί να είναι και τεχνήματα, δηλαδή σχετικά εκτελέσιμα αρχεία διαφορετικών τύπων (.jar, .war, docker container images), τα οποία με τη σειρά τους μπορεί και αυτά να αποθηκεύονται σε ειδικό αποθετήριο (πχ. Maven repository, Docker Hub). Πέραν της επιτάχυνσης της όλης διαδικασίας, μια θετική παρενέργεια που προκύπτει από τη συνεχή παράδοση είναι η μεγαλύτερη διαφάνεια και επικοινωνία ανάμεσα στο προσωπικό των dev, και εκείνο των ops [17].

2.3.3 Συνεχής Εγκατάσταση - Continuous Deployment

Η επιπρόσθετη διάσταση “CD” είναι αυτή της συνεχούς εγκατάστασης. Σε αυτό το στάδιο, τα νέα αρχεία που πέρασαν τις δοκιμές και αποθηκεύτηκαν στα αποθετήριά μας πλέον μπορούν να διατεθούν στην παραγωγή και να αξιοποιηθούν από τους χρήστες, αυτόματα, χωρίς καμία ανθρώπινη παρέμβαση. Η διαφοροποίηση ανάμεσα στην έννοια της συνεχούς εγκατάστασης και της συνεχούς παράδοσης, εντοπίζεται στην ολοκληρωτικά αυτοματοποιημένη εγκατάσταση της νέας έκδοσης του κώδικα, σε αντίθεση με τη συνεχή παράδοση, όπου απαραίτητα παρεμβάλλεται ο ανθρώπινος παράγοντας. Με αλλαγές κώδικα που μπορεί να φτάνουν και στις χιλιάδες μέσα σε μια μόνο μέρα, γίνεται φανερό ότι μειώνεται δραματικά ο χρόνος επίλυσης προβλημάτων και προσθήκης νέων λειτουργιών, ανάμεσα στην παραγωγή και την κατανάλωση του λογισμικού.



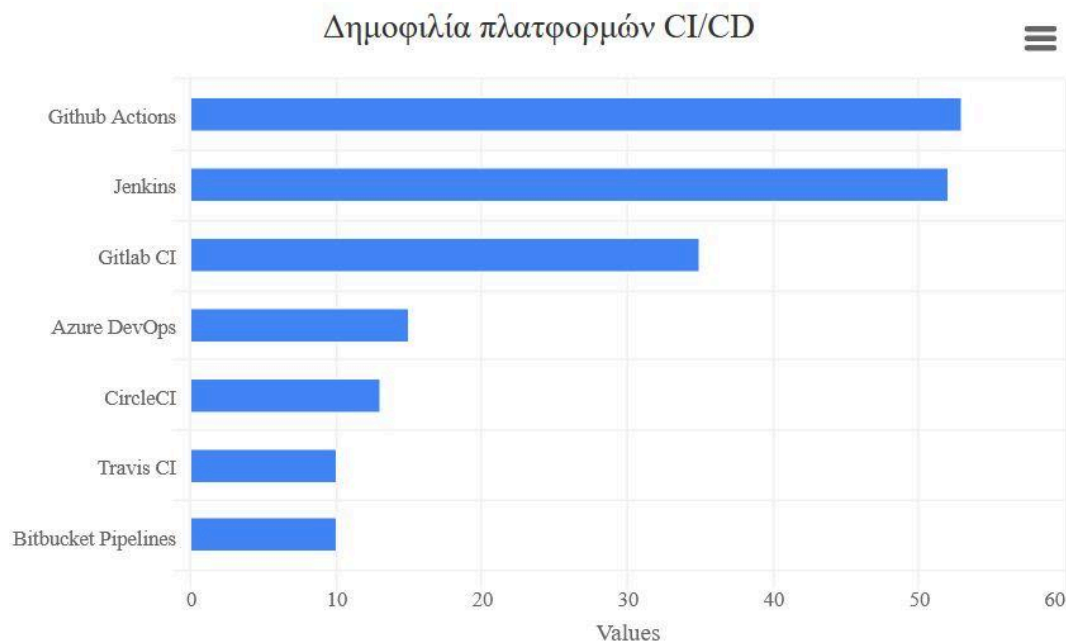
Εικόνα 5 - Continuous Integration/Continuous Delivery & Deployment

(Πηγή: <https://www.atlassian.com/continuous-delivery/principles/continuous-integration-vs-delivery-vs-deployment>)

2.3.4 Πλατφόρμες CI/CD

Είναι γεγονός ότι την τελευταία δεκαετία γνώρισε ιδιαίτερη άνθηση ο τομέας του CI/CD [78]. Με την αύξηση της ζήτησης, είναι επόμενο ο αρχικά κενός χώρος να καλυφθεί από μεγαλύτερο αριθμό παρόχων, οι οποίοι, προκειμένου να προσελκύσουν μεγαλύτερο τμήμα της αγοράς, ολοένα και αυξάνουν ή βελτιώνουν τις υπηρεσίες που παρέχουν. Ενδεικτικά της αλματώδους ανάπτυξης αυτού του χώρου είναι τα οικονομικά μεγέθη [21], τα οποία δείχνουν ότι από 1 δισ. δολάρια το 2023, ο τζίρος σχετικός με διεργασίες CI/CD θα εκτιναχθεί στα 3 δισεκατομμύρια μέχρι το 2028.

Με τόσο μεγάλο πλήθος διαθέσιμων επιλογών, αξίζει επιγραμματικά να αναφερθούν κάποιες εξ αυτών, καθώς επίσης και να παρουσιαστεί μια σύγκριση ανάμεσα τους. Σύμφωνα, λοιπόν, με μια έρευνα του 2023 [22], οι πλέον δημοφιλείς πλατφόρμες CI/CD είναι οι παρακάτω:



Εικόνα 6 - Δημοφιλότητα πλατφορμών CI/CD

(Πηγή: <https://blog.jetbrains.com/teamcity/2023/07/best-ci-tools/>)

Ακολουθεί μια μικρή ανάλυση για κάθε μια από αυτές τις πλατφόρμες CI/CD.

Github Actions: Η πιο δημοφιλής πλατφόρμα για CI/CD, η οποία είναι διαθέσιμη μόνο σαν SaaS (Software as a Service) (λογισμικό ως υπηρεσία). Είναι συνυφασμένη με την πιο δημοφιλή πλατφόρμα VCS Github [23], η οποία πρόσφατα αγοράστηκε από τη Microsoft [79]. Οι αγωγοί εκτελούνται μόνο σε κώδικα που έχει αποθηκευτεί στο Github, και ενεργοποιούνται κατόπιν συγκεκριμένων πράξεων του χρήστη (όπως είναι η υποβολή/commit κώδικα). Προσφέρει δωρεάν απεριόριστα λεπτά εκτέλεσης για αποθετήρια που είναι ανοικτά και διαθέσιμα στο ευρύ κοινό, όμως χρεώνει τους αγωγούς που εκτελούνται σε ιδιωτικά αποθετήρια. Για τον ορισμό του αγωγού χρησιμοποιείται YAML σύνταξη.

Jenkins [24]: Μέχρι πρότινος η πιο δημοφιλής πλατφόρμα CI/CD, με μεγάλο ποσοστό υιοθέτησης σε μεγάλους οργανισμούς. Πρόκειται για λογισμικό ανοικτού κώδικα γραμμένο σε γλώσσα Java, προορισμένο για τοπική εγκατάσταση και διαχείριση. Υποστηρίζεται από μια μεγάλη ενεργή κοινότητα προγραμματιστών, ενώ έχει εμπλουτιστεί από χιλιάδες πρόσθετα (plugins), προκειμένου να εξυπηρετήσει οποιαδήποτε ανάγκη. Ως λογισμικό ανοικτού κώδικα δεν επιφέρει χρεώσεις, εκτός και αν προσφέρεται σαν υπηρεσία από κάποιο μεταπωλητή. Για τον ορισμό του αγωγού υποστηρίζονται 2 διαφορετικά συντακτικά: (α) δηλωτικό (declarative) που

παρέχει πιο πλούσια συντακτικά χαρακτηριστικά και έχει σχεδιαστεί για την ευκολία στο γράψιμο και στην ανάγνωση του ορισμού ενός αγωγού - το συντακτικό αυτό είναι κυρίως κειμενικό (textual) και (β) προγραμματιστικό (scripted) μέσω της χρήσης της γλώσσας Apache Groovy.

Gitlab CI [25]: Η πλατφόρμα Gitlab αρχικά σχεδιάστηκε ως σύστημα VCS, με έμφαση στη διευκόλυνση της συνεργασίας ανάμεσα στα μέλη της ίδιας ομάδας και στην οριοθέτηση των δικαιωμάτων των χρηστών, πατώντας πάνω στο πολύ επιτυχημένο μοντέλο του Git. Από το 2014 άρχισε να ενσωματώνει επιπλέον λειτουργίες, όπως οι δυνατότητες για CI/CD, καθώς και η ενσωματωμένη λειτουργία αποθετηρίου εικόνων για δοχεία. Προσφέρεται ως SaaS με διάφορα μοντέλα δωρεάν έως κάποιες ώρες εκτέλεσης, αλλά και επί πληρωμή με επιπλέον δυνατότητες. Μπορεί επίσης να φιλοξενηθεί τοπικά. Και αυτή η πλατφόρμα βασίζεται στη σύνταξη YAML για τον ορισμό των αγωγών.

Azure DevOps [26]: Η γνωστή πλατφόρμα παροχής υπηρεσιών νέφους της Microsoft, η οποία παρέχει ένα σύνολο από υπηρεσίες DevOps. Αυτές περιλαμβάνουν υπηρεσίες εκτέλεσης αγωγών (Azure Pipelines), φιλοξενίας αποθετηρίων κώδικα, κα. Επίσης, υποστηρίζει την συνεχή παράδοση σε οποιοδήποτε περιβάλλον νέφους. Ξεκινά δωρεάν για μικρό αριθμό χρηστών και έπειτα ακολουθεί συγκεκριμένο μοντέλο χρέωσης μόλις αυτός ο αριθμός αυξηθεί. Οι αγωγοί συντάσσονται με βάση τη σύνταξη YAML.

Circle CI [27]: Μπορεί να εκτελεστεί με βάση γεγονότα από γνωστά συστήματα VCS όπως Github, Gitlab ή Bitbucket. Προσφέρεται τόσο με τη μορφή SaaS, όσο και με τοπική εγκατάσταση σε ιδίο εξοπλισμό. Προσφέρει εύχρηστο UI. Δεν υπάρχει χρέωση για τα πρώτα 6000 λεπτά χρήσης. Ο ορισμός των αγωγών βασίζεται και εδώ στη σύνταξη YAML.

Πίνακας 1 - Σύγκριση πλατφορμών CI/CD

(Πηγή: <https://blog.jetbrains.com/teamcity/2023/07/best-ci-tools/>)

Πλατφόρμα CI/CD	Ανοικτού Κώδικα	Τοπική Εγκατάσταση	Δωρεάν Έκδοση	Υποστηριζόμενα OS	Γλώσσα Σύνταξης	Αποθετήριο Κώδικα	Αποθετήριο Εικόνων/Τεχνημάτων
GitHub Actions	Όχι	Όχι	Ναι	Linux, Windows, macOS Docker ¹	yaml	Ναι	Ναι
Jenkins	Ναι	Ναι	Ναι	Linux, Windows, macOS, Docker	jenkinsfile, Groovy	Όχι	Όχι
GitLab CI	Ναι	Ναι	Ναι	Linux, Windows, macOS, Docker	yaml	Ναι	Ναι
Azure DevOps	Όχι	Ναι	Ναι	Linux, Windows, macOS	yaml	Ναι	Ναι
CircleCI	Όχι	Ναι	Ναι	Linux, macOS, Windows, GPU, ARM, Docker	yaml	Όχι	Όχι

Κατόπιν ανάλυσης και σύγκρισης των ανωτέρω κύριων πλατφορμών CI/CD για την υλοποίηση του τεχνικού μέρους της Διπλωματικής Εργασίας (δείτε παραπάνω πίνακα με την αποτίμηση των πλατφορμών με βάση συγκεκριμένα κριτήρια), επιλέχθηκε το Gitlab CI/CD για τους εξής λόγους:

- Υποστηρίζει την τοπική εγκατάσταση, τόσο της κυρίας εφαρμογής, όσο και των runners, ήτοι των προγραμμάτων που απαιτούνται για την εκτέλεση των διαφόρων σταδίων ενός αγωγού. Επιπλέον, μπορεί να εκτελεστεί και σαν δοχείο Docker, κατεύθυνση που θα ακολουθηθεί στο πλαίσιο της εργασίας. Η δυνατότητα για πλήρη τοπική εγκατάσταση κρίθηκε απαραίτητη, για τη μέγιστη δυνατή παραμετροποίηση του περιβάλλοντος, αλλά

¹ Μόνο για τους Runners και όχι για το κύριο σύστημα

και λόγω των απαιτήσεων ασφαλείας στα οποία μπορεί μια τέτοια υλοποίηση να βρει εφαρμογή.

- Καθώς ξεκίνησε σαν επέκταση του συστήματος VCS git, εμπεριέχει το δικό του αποθετήριο κώδικα. Έχοντας δώσει ιδιαίτερη σημασία σε λειτουργικότητες συνεργασίας ανάμεσα στα μέλη μιας ομάδας, η πλατφόρμα Gitlab εμπεριέχει λειτουργίες διάρθρωσης/ιεράρχησης έργων, διαχείρισης ζητημάτων ή ανάθεσης αυτών σε υπαλλήλους, έγκρισης/απόρριψης κώδικα από εξουσιοδοτημένο υπάλληλο, καθώς και πρόσβασης με βάση διακριτούς ρόλους και διακριτούς φακέλους.
- Εμπεριέχει αποθετήρια τεχνημάτων & δοχείων (artifact registry & container registry), για να αποθηκεύονται τα εκτελέσιμα αρχεία/δοχεία προς εγκατάσταση.
- Για τη διαμόρφωση του αγωγού χρησιμοποιεί τη διαδεδομένη γλώσσα σύνταξης YAML.
- Είναι ανοικτού κώδικα οπότε επιδέχεται οποιοδήποτε είδος επέκτασης, η οποία μπορεί να κριθεί απαραίτητη στα πλαίσια της τρέχουσας διπλωματικής εργασίας

2.3.5 Πλατφόρμα Gitlab

Μια από τις πλέον διαδεδομένες γλώσσες σύνταξης αρχείων ρυθμίσεων στο χώρο του υπολογιστικού νέφους και λογισμικού είναι η γλώσσα YAML. Δημιουργήθηκε από τον Clark Evans το 2005 [29], με τα αρχικά του ονόματος να σημαίνουν “Yet Another Markup Language”, σαν αστείο σε μια περίοδο όπου τέτοιες γλώσσες γνώρισαν υπέρμετρη άνθιση [28]. Θεωρείται ένα υπερσύνολο του γνωστού προτύπου JSON, άμεσα κατανοητό από τον άνθρωπο. Σε γενικές γραμμές μπορεί να θεωρηθεί σαν ένα λεξικό, με κάθε εγγραφή αρχικά να δηλώνει το κλειδί, και στη συνέχεια την τιμή του κλειδιού αυτού (η οποία μπορεί να είναι απλή, ένας πίνακας ή μια άλλη εγγραφή).

Στο οικοσύστημα του Gitlab, παρατηρούμε τα εξής θεμελιώδη δομικά τμήματα σε ένα αρχείο gitlab-ci.yml, στο οποίο περιγράφεται η δομή και η λειτουργία ενός αγωγού CI/CD [30]:

Stages: Κάθε αγωγός αναγκαστικά απαρτίζεται από ένα ή περισσότερα στάδια (stages). Το κάθε στάδιο περιλαμβάνει κάποιες ενέργειες που λειτουργικά συνδέονται μεταξύ τους, οι οποίες και ονομάζονται jobs. Η αποτυχία μιας τέτοιας ενέργειας συνήθως οδηγεί στον πρόωρο τερματισμό / αποτυχία της εκτέλεσης του αγωγού. Ένας τυπικός αγωγός στο περιβάλλον του Gitlab συνήθως αποτελείται από τα εξής βασικά στάδια:

- **Build:** Περιέχει εντολές κατάλληλες για την κατασκευή ενός εκτελέσιμου αρχείου ή εικόνας δοχείου προς εγκατάσταση ή αποθήκευση σε ένα αποθετήριο.

- **Lint:** Σε αυτό το στάδιο γίνεται τυχόν μορφοποίηση του κώδικα σύμφωνα με προσυμφωνημένα πρότυπα.
- **Test:** Σε αυτό το στάδιο εκτελούνται αυτοματοποιημένοι έλεγχοι του κώδικα ή μορφοποίησης αυτού, καθώς και έλεγχοι ασφαλείας (ιδιαίτερα στην περίπτωση DevSecOps).
- **Deploy:** Αυτό το στάδιο αναλαμβάνει την εγκατάσταση/διάταξη του κώδικα, ανάλογα με το περιβάλλον που έχει επιλεγεί.

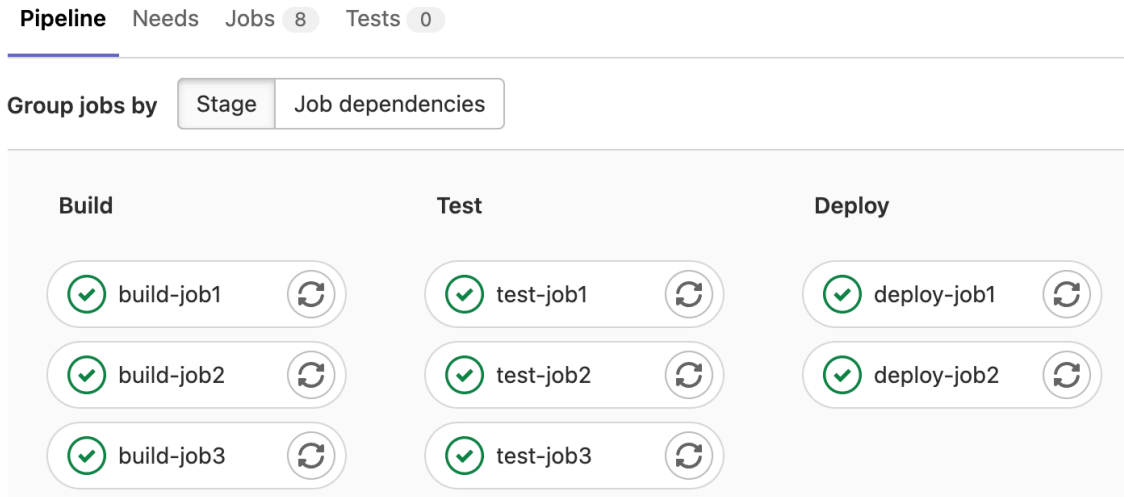
Δεν υπάρχει περιορισμός στο πλήθος των σταδίων που περιγράφονται σε ένα αρχείο `gitlab-ci.yml`.

Jobs: Προηγουμένως είδαμε ότι στα στάδια περιέχονται ενέργειες (jobs). Οι ενέργειες αυτές υποδιαίρουν ακόμα περισσότερο την ιεραρχία του αγωγού που πρέπει να εκτελεστεί. Κάθε στάδιο πρέπει υποχρεωτικά να περιέχει τουλάχιστον μια ενέργεια, η οποία με τη σειρά της περιέχει και τη μικρότερη δομική μονάδα ενός αγωγού, που είναι η απλή εντολή (command).

Εντολές: Οι εντολές που εκτελούνται στο πλαίσιο μιας ενέργειας (job) είναι τελικά αυτές που θα εκτελέσουν και το πραγματικό έργο που απαιτείται. Έχουν την ίδια δομή και σύνταξη με την αλληλουχία εντολών που θα εκτελούσε ένας άνθρωπος, εάν ήθελε να επιτύχει το επιθυμητό αποτέλεσμα σε μια κονσόλα τερματικού. Οι εντολές αυτές εκτελούνται από εξειδικευμένα προγράμματα που ονομάζονται Gitlab Runners², τα εγκαθίστανται σε κάποιο σύστημα και συνεχώς ελέγχουν εάν υπάρχουν εκκρεμείς εντολές προς εκτέλεση. Συνεπώς, ανάλογα με το περιβάλλον που προσομοιώνεται από τον Gitlab Runner (Linux, Windows, Εντολές μέσα σε δοχεία κ.α), απαιτούνται και οι ανάλογες εντολές.

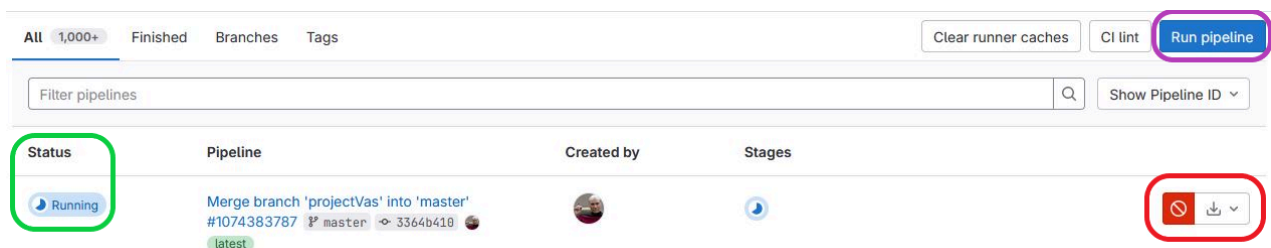
Ένα από τα πιο χρήσιμα χαρακτηριστικά της πλατφόρμας Gitlab είναι το εύχρηστο UI της, μέσα από το οποίο είναι άμεσα ορατά τα διαφορετικά στάδια και ενέργειες καθώς και οι καταστάσεις τους, όταν εκτελείται ένας αγωγός, όπως παρακάτω:

² <https://docs.gitlab.com/runner/>



Εικόνα 7 - Απεικόνιση των Stages & Jobs στο Gitlab UI

Πλην της διαχείρισης και προβολής των σταδίων ενός αγωγού, προσφέρεται και UI για την κατάσταση του ίδιου του αγωγού (Πράσινο πλαίσιο), πάυση ή λήψη των τεχνημάτων (artifacts) που παρήχθησαν κατά τη διάρκεια αυτού (Κόκκινο Πλαίσιο), καθώς και εκκίνηση νέου αγωγού (Μωβ πλαίσιο), όπως παρακάτω:



Εικόνα 8 - Εποπτεία του αγωγού στο Gitlab UI

Μια από τις πιο χρήσιμες λειτουργικότητες του Gitlab UI είναι ο ενσωματωμένος κειμενογράφος για τον ορισμό και πλέον άμεση δοκιμή της λειτουργίας του αγωγού (Σε κόκκινο πλαίσιο η άμεση ανίχνευση συντακτικών σφαλμάτων), χωρίς να απαιτείται συγγραφή σε προσωπικό υπολογιστή και στη συνέχεια ανέβασμα στο αποθετήριο git, όπως φαίνεται παρακάτω:



Εικόνα 9 - Κειμενογράφος αγωγών στο Gitlab UI

2.4 Εικονικοποίηση - Virtualization

Κυρίαρχο ρόλο στον χώρο του υπολογιστικού νέφους αλλά και σε αυτόν του devops διαδραματίζει η εικονικοποίηση. Είναι μια από τις πιο διαδεδομένες τεχνολογίες στο σύγχρονο κόσμο της πληροφορικής και των επικοινωνιών, καθώς επιτρέπει την συνύπαρξη και συλλειτουργία πολλαπλών διαφορετικών (εικονικών) συστημάτων (ή μηχανών) σε ένα και μοναδικό φυσικό εξυπηρετητή. Αρχικά στο χώρο αυτό εμφανίστηκαν οι Εικονικές Μηχανές (Virtual Machines), ενώ στη συνέχεια έκαναν την εμφάνισή τους και τα δοχεία (containers).

2.4.1 Εικονικές Μηχανές - Virtual Machines (VMs)

Πρόκειται για μια αναπαράσταση ενός υπολογιστικού συστήματος που μπορεί να εκτελέσει προγράμματα και λειτουργίες σε ένα απομονωμένο χώρο ενός φυσικού εξυπηρετητή. Η απομόνωση επιτυγχάνεται από ένα εξειδικευμένο πρόγραμμα που ονομάζεται υπερεπόπτης (hypervisor), το οποίο και αναλαμβάνει την εκχώρηση συγκεκριμένων δυνατοτήτων και πόρων σε κάθε μια από τις εικονικές μηχανές. Η κάθε εικονική μηχανή χρησιμοποιεί τους φυσικούς πόρους που της έχουν ανατεθεί, όμως χρησιμοποιεί το δικό της ξεχωριστό κέλυφος (kernel) λειτουργικού συστήματος.

Η εικονικοποίηση για τη λειτουργία μιας εικονικής μηχανής διακρίνεται στα παρακάτω είδη [82]:

- **Πλήρης εικονικοποίηση (Full Virtualization)**

Το φιλοξενούμενο λειτουργικό σύστημα δεν γνωρίζει ότι εκτελείται σε εικονικό υλικό, με τις εντολές του να εκτελούνται τελικά από τον υπερεπόπτη του εξυπηρετητή φιλοξενίας.

- **Παραεικονικοποίηση (Paravirtualization)**

Το φιλοξενούμενο λειτουργικό σύστημα γνωρίζει ότι φιλοξενείται, με αποτέλεσμα να μηνύει τις εντολές του απευθείας στο λειτουργικό σύστημα του οικοδεσπότη για ορισμένα είδη πόρων. Για τα άλλα είδη πόρων, οι εντολές εφαρμόζονται και πάλι από τον υπερεπόπτη.

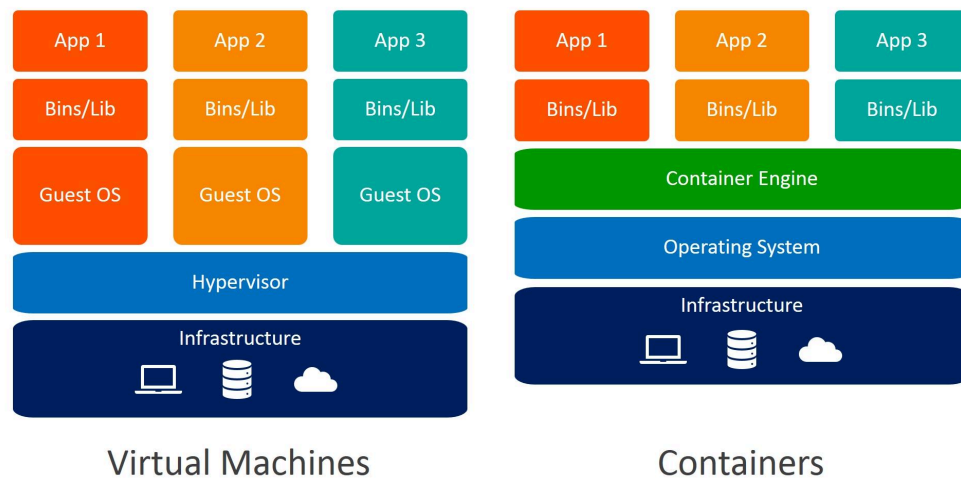
2.4.2 Δοχεία - Containers

Το δοχεία ή καλύτερα η δοχειοποίηση (containerisation) είναι και αυτή μια τεχνική εικονικοποίησης στο επίπεδο του λειτουργικού συστήματος που επιτρέπει τη γρήγορη εκτέλεση προγραμμάτων, ανεξάρτητα από την παραμετροποίηση του συστήματος στο οποίο εκτελούνται. Κάθε δοχείο είναι μια ειδική διεργασία συγκεκριμένου χώρου χρηστών στο λειτουργικό σύστημα, χαλαρά απομονωμένη από τις έτερες που εκτελούνται στον ίδιο εξυπηρετητή, ενώ για τη λειτουργία του δεν απαιτείται υπερεπόπτης [18]. Σε αντίθεση με τις εικονικές μηχανές, τα δοχεία βασίζονται στο κέλυφος (Kernel) του λειτουργικού συστήματος του εξυπηρετητή (είτε αυτός είναι φυσικός είτε εικονικός) στον οποίο εκτελούνται, και γι αυτό το λόγο είναι πολύ πιο ευέλικτα και γρήγορα στην εκκίνηση, πιο ελαφριά, και επιτρέπουν την ευέλικτη διαχείριση των υποκείμενων πόρων. Όμως, προσφέρουν πιο αδύναμη απομόνωση [83]. Για την διαχείριση και εκτέλεση των δοχείων, το πιο διαδεδομένο εργαλείο/πλαίσιο είναι το Docker, όμως υπάρχουν και τρίτες λύσεις όπως podman³ ή containerd, καθώς και λύσεις για την ενορχήστρωση αυτών όπως kubernetes⁴, openshift⁵ κ.α.

³ <https://podman.io/>

⁴ <https://kubernetes.io/>

⁵ <https://www.redhat.com/en/technologies/cloud-computing/openshift>



Εικόνα 10 - Εικονικές Μηχανές & Δοχεία

(Πηγή: <https://www.weave.works/blog/a-practical-guide-to-choosing-between-docker-containers-and-vms>)

Όπως γίνεται φανερό και από την Εικόνα 10, ενώ μια εικονική μηχανή διαθέτει πάντοτε το δικό της φιλοξενούμενο λειτουργικό σύστημα και βασίζεται στον υπερεπόπτη του οικοδεσπότη εξυπηρετητή για την εκτέλεση των εντολών της, το ίδιο δεν ισχύει για τα δοχεία, όπου χρησιμοποιούν το ίδιο λειτουργικό σύστημα από κοινού.

3

Σχετικές Εργασίες

Σε αυτό το κεφάλαιο θα παρουσιαστούν συνοπτικά, εργασίες σχετικές με το αντικείμενο της παρούσας Διπλωματικής εργασίας, και στη συνέχεια θα συγκριθούν ως προς τους στόχους και το περιεχόμενό τους.

3.1 Εργασίες περί DevOps

Στο [32], η Saarinen ανέλυσε τους κινδύνους που πρέπει να αντιμετωπιστούν τόσο κατά τον κύκλο ζωής του λογισμικού, όσο και κατά τις διαδικασίες devops ενός φορέα, ποσοτικοποιημένα σύμφωνα με αναγνωρισμένα πρότυπα. Η διατριβή ήταν διεξοδική αναφορικά με τα θεωρητικά στάδια προσδιορισμού των κινδύνων αλλά και πολιτικών ασφαλείας, όμως δεν περιέγραψε κάποια τεχνική υλοποίηση/εφαρμογή αυτών.

Με το [37], οι Desai και Nisha μέσα από περιπτώσεις χρήσης πρότειναν ένα θεωρητικό μοντέλο εύρεσης ευπαθειών, γενικότερα σε μια διαδικασία devops και όχι ειδικά για έναν αγωγό CI/CD. Δεν προτάθηκε ωστόσο καμία τεχνική υλοποίηση για την εφαρμογή/υλοποίηση αυτού του μοντέλου.

Με το [38], από τους Rahman και Williams αναδείχθηκαν οι θετικές συνέπειες μια διαδικασίας devops στον τομέα της ασφάλειας. Ιδιαίτερο ενδιαφέρον δόθηκε στον τομέα της συνεργασίας μεταξύ των ομάδων, παρουσιάζοντας δείκτες επίτευξης συνεργασίας ανάμεσα σε μέλη των ομάδων προγραμματιστών και ομάδων ασφαλείας, καθώς επίσης και στον αυτοματισμό των διαδικασιών (δοκιμών, επισκόπησης κώδικα, παρακολούθησης συστημάτων κ.α). Ωστόσο δεν προτάθηκε κάποια τεχνική λύση ή εργαλεία προς το σκοπό αυτό.

Οι Mao κ.α [40] περιέγραψαν στο άρθρο τους τις διακριτές δραστηριότητες μιας διαδικασίας DevOps και εξέτασαν τις δυσκολίες που έχουν οι οργανισμοί ως προς την υιοθέτηση του νέου αυτού μοντέλου λειτουργίας. Επιπλέον, μελέτησαν κενά ασφαλείας που εισάγονται, καθώς και νέες ευκαιρίες για βελτίωση του επιπέδου ασφαλείας των παρεχόμενων υπηρεσιών του

εκάστοτε οργανισμού, χωρίς να περιγράφεται μια πρακτική υλοποίηση ενός τέτοιου μοντέλου devops.

Οι Chernyshev, Baig, και Zeadally [41] κυρίως βασίστηκαν σε ανάλυση κινδύνου μιας DevOps διαδικασίας, παρουσιάζοντας τα πλεονεκτήματα αυτής, και ιδίως τη συμβολή των μέτρων αντιμετώπισης αυτών των κινδύνων σε κάθε στάδιο της διαδικασίας. Μπορεί να ειπωθεί ότι το άρθρο περισσότερο συνόψισε ήδη εδραιωθείσες πληροφορίες από πηγές τόσο βιβλιογραφικές όσο και διαδικτυακές, χωρίς να προσφέρει απτή υλοποίηση.

Στο άρθρο [42], η Battina επισημαίνει την αναγκαιότητα της εύρεσης τυχόν ευπαθειών όσο το δυνατόν νωρίτερα σε μια διαδικασία DevOps, προτείνοντας παράλληλα προτάσεις για σκλήρυνση (αυστηρότερη επιβολή ελέγχου πρόσβασης και ταυτοποίησης) της όλης διαδικασίας σε θεωρητικό επίπεδο. Δεν προσφέρεται κάποια τεχνική υλοποίηση ή πρόταση συγκεκριμένων εργαλείων.

Με το άρθρο [44], οι Casola, De Benedictis, Rak, και Villano πρότειναν μια μέθοδο διαμόρφωσης ενός, από το σχεδιασμό (by design), ασφαλούς συστήματος DevOps. Το μεγαλύτερο βάρος της έρευνας δόθηκε στην αναγνώριση και ανάλυση κινδύνου σύμφωνα με το μοντέλο STRIDE (ενδεικτικά: Cross-Site Request Forgery, Cross-Site Scripting, Man in the Middle). Δεν παρουσιάστηκε κάποια τεχνική υλοποίηση. Όμοια και στο [49], ο Koorman ακολούθησε μια μοντελοποιημένη προσέγγιση ανάλυσης ρίσκου και κατέγραψε αναλυτικά όλους τους πιθανούς κινδύνους που μπορεί να απειλήσουν μια DevOps διαδικασία ενός οργανισμού (ενδεικτικά: Κοινοί λογαριασμοί προγραμματιστών στο VCS, Man in the Middle επιθέσεις, τεχνήματα περιέχοντα κακόβουλο λογισμικό).

Στη μεταπτυχιακή διατριβή [52], ο Škurla κάνει μια γενική αναφορά στη δυσκολία (αδυναμία των δοκιμών ασφαλείας να προλάβουν την ταχύτερη παραγωγή λογισμικού, ανεπαρκής έλεγχος πρόσβασης, απουσία αυτοματοποιημένων δοκιμών ασφαλείας κ.α) που εντοπίζεται στην εισαγωγή των στοιχείων ασφαλείας σε μια τυπική DevOps διαδικασία, ενώ στη συνέχεια προτείνει κάποια εργαλεία (WhiteSource για εύρεση ευπαθειών κώδικα, Aqua Security για εύρεση ευπαθειών λογισμικού, ZAP για δοκιμές διείσδυσης, κ.α) που μπορούν να αξιοποιηθούν προς το σκοπό αυτό. Η εργασία ωστόσο στερείται τεχνικής υλοποίησης.

3.2 Εργασίες σκλήρυνσης αγωγών CI/CD

Ο Gonçalves στο [31] σχεδίασε και υλοποίησε ένα σύστημα αγωγού CI/CD ικανό να αυξάνει την ασφάλεια του παραγόμενου κώδικα, βασισμένο σε εργαλεία και τεχνολογίες ανοικτού

κώδικα, χρησιμοποιώντας την πλατφόρμα Jenkins σε συνδυασμό με εργαλεία, όπως Sonarqube, OWASP Zap και Selenium. Αυτά τα εργαλεία είναι εξαιρετικά χρήσιμα για την προστασία του λογισμικού από ευπάθειες και τρωτότητες, ωστόσο ο διαμορφωθείς αγωγός δεν ήταν σχεδιασμένος με βάση δοχεία, μια από τις βασικές απαιτήσεις της παρούσας διπλωματικής εργασίας, καθώς τα δοχεία κυριαρχούν στο σημερινό τεχνολογικό χώρο CI/CD, και επιπρόσθετα δεν είχε σχεδιαστεί με γνώμονα την παράλληλη ασφάλεια του ίδιου του αγωγού.

Στο [33] οι Shamim, Bhuiyan και Rahman & στο [34] οι Minna, Blaise, Rebecchi, Chandrasekaran και Massacci προσέγγισαν την ασφάλεια μιας CI/CD διαδικασίας μέσα από το πρίσμα των ευπαθειών που μπορεί να παρουσιαστούν σε μια εγκατάσταση λογισμικού μέσω ενός συστήματος Continuous Delivery/Deployment και ειδικότερα ενορχήστρωσης δοχείων, όπως είναι το εργαλείο kubernetes, καθώς και μέτρων αντιμετώπισης αυτών. Μια τέτοια εγκατάσταση όμως είναι το τελευταίο βήμα της όλης διαδικασίας CI/CD, αφού το δοχείο έχει δοκιμαστεί με αυτοματοποιημένο τρόπο στα προηγούμενα στάδια αυτής, ενώ ευπάθειες πλήττουν όλο το εύρος της διαδικασίας (βλ. Κεφάλαιο 4).

Στο [35] οι Pecka, Othmane και Valani εντόπισαν τέσσερις τρόπους για να επιτευχθεί αλλοίωση της εγκατάστασης λογισμικού μέσω κακής ρύθμισης είτε του αποθετηρίου κώδικα, είτε του ενορχηστρωτή δοχείων Kubernetes κατά το στάδιο CD. Στη συνέχεια, ενάντια σε όλες αυτές τις ενδεχόμενες επιθέσεις, πρότειναν για αντίμετρο τη σωστή διαμόρφωση ελέγχου πρόσβασης στα εμπλεκόμενα συστήματα (VCS, Jenkins & Kubernetes). Στο εν λόγω άρθρο δεν εξετάστηκαν τρόποι σκλήρυνσης του κώδικα, αλλά ούτε και των δοχείων.

Οι Lombardi και Fanton στο [36] συνέταξαν μια μοντελοποιημένη λίστα απειλών κατά συστημάτων CI/CD, και στη συνέχεια υλοποίησαν έναν αγωγό ικανό να αντιμετωπίσει ευπάθειες σε βιβλιοθήκες και πηγαίο κώδικα, κάνοντας χρήση λογισμικού όπως Jenkins, git-secrets, OWASP Dependency Check, Bandit, Sonarqube, Burp & OWASP ZAP. Τα αντίμετρα ασφαλείας δεν ήταν προσανατολισμένα σε όλο το εύρος του αγωγού, αλλά ούτε είχαν στο επίκεντρο την τεχνολογία δοχείων.

Οι συγγραφείς του [39], Bass, Holz, Rimba, Tran, και Zhu, συνέταξαν πίνακα με τα διακριτά τμήματα ενός αγωγού CI/CD, τα κατηγοριοποίησαν ανάλογα με το πόσο ευπρόσβλητα είναι, σήμαναν τις συνέπειες τυχόν επίθεσης σε βάρος τους, και προσέφεραν θεωρητικές λύσεις (προσδιορισμός απαιτήσεων ασφαλείας, αναγνώριση αξιόπιστων -και μη- συστατικών του αγωγού, τροποποίηση αναξιόπιστων συστατικών ή αντικατάσταση με αξιόπιστα). Το άρθρο δεν περιείχε τεχνική υλοποίηση, έλεγχο κώδικα ή εμβάθυνση σε τεχνολογία δοχείων.

Με το [43], από τους Ahmed και Shoba πραγματοποιήθηκε μερική ενσωμάτωση στοιχείων DevSecOps σε έναν παραδοσιακό αγωγό DevOps, χρησιμοποιώντας το γνωστό εργαλείο για ανίχνευση ευπαθειών σε κώδικα και τρίτες βιβλιοθήκες, Snyk. Ωστόσο, η τεχνική υλοποίηση είναι πολύ περιορισμένη σε βεληνεκές, με αποτέλεσμα να μην καλύπτει από άκρο σε άκρο ένα σύγχρονο αγωγό βασισμένο σε δοχεία.

Στο άρθρο [45], οι ερευνητές Rangnau, Buijtenen, Fransen, και Turkmen σχεδίασαν και υλοποίησαν έναν αγωγό CI/CD βασισμένο σε δοχεία, με μοναδική έμφαση στο δυναμικό έλεγχο (DAST - Dynamic Application Security Testing) μιας εφαρμογής. Χρησιμοποιήθηκαν γνωστά εργαλεία, όπως το OWASP ZAP, JMeter & Selenium, ώστε αυτοματοποιημένα ή χειροκίνητα να διεξαχθούν έλεγχοι τύπου WAST (Web Application Security Scanning), APS (Api Security Testing) & BDST (Behaviour Driven Security Testing). Το άρθρο παρουσίασε τις ευπάθειες (πχ. Έκχυσης SQL υψηλού επιπέδου) που φανερώθηκαν και τους τρόπους αντιμετώπισής τους (πχ. τροποποίηση του κώδικα για εξυγίανση των παραμέτρων που εισάγει ο χρήστης).

Με το [46] έργο τους, οι ερευνητές Sojan, Rajan και Kuvaja αποτύπωσαν τα σχέδια μιας αρχιτεκτονικής παρακολούθησης συστημάτων βασισμένων σε δοχεία, αξιοποιώντας λογισμικά ανοικτού κώδικα, όπως Elastic, Kibana, Logstash, Grafana & Prometheus, με σκοπό να απεικονίζεται σε πραγματικό χρόνο τόσο η χρήση του υπό παρακολούθηση συστήματος, όσο και των αρχείων καταγραφής που παράγονται, με σκοπό να ανιχνεύονται τυχόν επιθέσεις ή παράτυπες δραστηριότητες στο δοχείο ή τον εξυπηρετητή. Η θεωρητική αυτή αποτύπωση δε συνοδεύτηκε από τεχνική υλοποίηση και συμπεράσματα.

Ο Singh στο [47] σχεδίασε και υλοποίησε έναν αγωγό CI/CD βασισμένο σε δοχεία, ο οποίος ήταν σε θέση να εκτελεί ελέγχους τύπου unit test & end-to-end test, χωρίς όμως να δίνει έμφαση στην ασφάλεια του ίδιου του αγωγού.

Στο άρθρο τους [48], οι Ullah, Raft, Shahin, Zahedi, και Ali Babar, επικεντρώνονται σε προβλήματα ασφαλείας ενός αγωγού CI/CD βασισμένου σε εικονικές μηχανές και το σύστημα Jenkins. Προτείνουν τρόπους για ασφαλή διαχείριση λογαριασμών και πρόσβασης, καθώς και χρήση εργαλείων DAST (Qualys OWASP Scan, ZAP Scanner) ενώ παρουσιάζουν και ποσοτικά ευρήματα (πολλαπλές ευπάθειες insecure cookies, XSS, autocomplete σε πεδία κωδικών). Το συγκεκριμένο άρθρο δε χρησιμοποιεί τεχνολογία δοχείων, ενώ ταυτόχρονα δεν αμύνεται ενάντια σε ευρύ φάσμα απειλών.

Στη διατριβή [50], η συγγραφέας Paule σχημάτισε έναν αναλυτικό κατάλογο με ευάλωτα σημεία ενός αγωγού CI/CD, τα οποία μπορεί να προσβάλει ένας επιτιθέμενος προκειμένου να επιφέρει ζημία σε έναν οργανισμό, ακολουθώντας τη μέθοδο STRIDE. Για κάθε τρωτό σημείο

προτείνει και από ένα αντίμετρο ασφαλείας (π.χ εργαλείο git secrets για εύρεση ευαίσθητων δεδομένων στο αποθετήριο, εργαλείο OWASP Dependency Check για εντοπισμό ευπαθών βιβλιοθηκών λογισμικού, εργαλείο Dagda για ευπάθειες δοχείων κ.α), είτε χειροκίνητο είτε αυτοματοποιημένο. Τέλος, παρουσιάζει 2 περιπτώσεις χρήσης, στις οποίες εφαρμόστηκε η χειροκίνητη μέθοδος ανίχνευσης τρωτοτήτων, σε συνδυασμό με τα ευρήματα αυτών. Συνοψίζοντας, δεν παρουσιάστηκε μια ολοκληρωμένη υλοποίηση ενός ασφαλούς συστήματος αγωγών CI/CD.

Με τη μεταπτυχιακή διατριβή [51], οι Pham και Chalishhafshejani σχεδίασαν και υλοποίησαν έναν αγωγό CI/CD, δίνοντας έμφαση στην ασφάλεια του λογισμικού αλλά και του ίδιου του αγωγού, χρησιμοποιώντας αυτοματοποιημένα εργαλεία ανοικτού κώδικα. Για το σκοπό αυτό αξιοποίησαν την πλατφόρμα Github Actions, σε συνδυασμό με εργαλεία όπως το Snyk, Sonarqube, καθώς και scripts για ανίχνευση ανοικτών ευάλωτων θυρών και λανθασμένων πιστοποιητικών. Συνολικά κατόρθωσαν να επιτύχουν ένα υψηλό επίπεδο ασφάλειας, ενώ το μόνο έλλειμμα ήταν η μη αντιμετώπιση/σκλήρυνση της ασφάλειας των δοχείων.

Στο άρθρο [53], στο οποίο οι συγγραφείς Brady, Moon, Nguyen, και Coffman ενσωμάτωσαν σε έναν αγωγό CI/CD εργαλεία στατικής ανάλυσης εικόνων δοχείων για ευπάθειες, καθώς και δυναμικής ανάλυσης των δοχείων κατά το χρόνο εκτέλεσής τους. Με τη βοήθεια συμπληρωματικών δοχείων που έχουν προσαρτηθεί στα κύρια, κατέγραψαν τις δικτυακές κινήσεις αλλά και τις διεργασίες των κύριων δοχείων, με σκοπό να ανιχνεύσουν ασυνήθιστη δραστηριότητα. Ενώ το άρθρο παρουσιάζει τα αποτελέσματα εφαρμογής του αγωγού σε δοχεία τύπου nginx και Postgres, δυστυχώς δεν αναλύει σε βάθος τον τρόπο υλοποίησης των συμπληρωματικών δοχείων. Η συγκεκριμένη υλοποίηση αγωγού παρέχει προστασία ενάντια σε ευπάθειες δοχείων, κακόβουλο λογισμικό και παράτυπες δικτυακές κλήσεις, ωστόσο δεν παρέχει συνολική προστασία του αγωγού.

3.3 Σύγκριση με τη Διπλωματική εργασία

Μελετώντας τις αξιόλογες εργασίες που συγκεντρώθηκαν για τη συγγραφή αυτού του κεφαλαίου, πρέπει να τονισθεί ότι εξήχθησαν πολύτιμα συμπεράσματα για τον τρόπο προσέγγισης της υλοποίησης ενός ασφαλούς συστήματος CI/CD βασισμένου σε δοχεία. Η έρευνα στις τεχνολογίες και εργαλεία αιχμής, σε συνδυασμό με τους τεχνολογικούς χώρους που αφέθηκαν κενοί από τους συγγραφείς των έτερων εργασιών, οδήγησε σε ένα - κατά το δυνατόν - πλήρες μείγμα μέτρων ασφαλείας στους εξής τομείς:

- Σκλήρυνση εικονικών μηχανών
- Ασφάλεια αποθετηρίου κώδικα
- Εξασφάλιση λειτουργικότητας κώδικα
- Ασφάλεια Λογισμικού
- Ασφάλεια Δοχείων
- Ασφάλεια εγκατάστασης δοχείων
- Παρακολούθηση δοχείων

Ενώ παρατηρήθηκε ότι οι περισσότερες εργασίες καταπιάστηκαν με την ασφάλεια σε 1-2 από τους ανωτέρω τομείς, ενώ κάποιες όπως η [51] έστρεψαν το ενδιαφέρον τους σε 3 τομείς, η παρούσα Διπλωματική εργασία αποπειράται να δώσει ένα ικανοποιητικό επίπεδο ασφάλειας και στους 7 τομείς, προσφέροντας μια τεχνολογική λύση ικανή να αποτρέψει μια μεγάλη πλειονότητα απειλών σε βάρος του συστήματος CI/CD.

4

Απειλές

Σε αυτό το κεφάλαιο θα παρουσιαστούν οι κύριες απειλές που μπορεί να προκύψουν - σκόπιμα ή και ακούσια - ενάντια σε ένα σύστημα CI/CD. Οι κίνδυνοι που θα περιγραφούν αφορούν τόσο τα γενικού τύπου συστήματα DevOps, όσο και συγκεκριμένα εκείνα που βασίζονται στην τεχνολογία των δοχείων, εφόσον αυτά αποτελούν τον κορμό του συστήματος της Διπλωματικής εργασίας

Σύμφωνα με την Liz Rice [56], οι παράγοντες απειλών εναντίον των δοχείων διαιρούνται στις εξής κατηγορίες:

- **Εξωτερικοί επιτιθέμενοι** που προσπαθούν να αποκτήσουν πρόσβαση σε ένα σύστημα
- **Εσωτερικοί επιτιθέμενοι** που ήδη έχουν στον έλεγχο τους μέρος του συστήματος, όπως προγραμματιστές και διαχειριστές.
- **Ακούσιοι εσωτερικοί παράγοντες**, οι οποίοι μπορεί να προκαλέσουν βλάβη άθελα τους

4.1 Απειλές ενάντια σε γενικού τύπου συστήματα CI/CD

Για να εντοπιστεί ένα ολοκληρωμένο σύνολο από τρωτότητες απέναντι σε ένα CI/CD σύστημα μελετήθηκαν τα άρθρα [49], [54], [39], [48], [55], [50]. Με βάση αυτά τα άρθρα έχουμε τα παρακάτω ευρήματα, τα οποία παρουσιάζονται στις επόμενες υπο-ενότητες της Ενότητας 4.1.

4.1.1 Παθητικοί/Ενεργητικοί Κρυφακουστές (Eavesdroppers)

Η κίνηση ανάμεσα στα διακριτά μέρη ενός συστήματος CI/CD είναι ευάλωτη σε επιτιθέμενους που επιχειρούν να κρυφακούσουν είτε παθητικά, είτε ενεργητικά με ενδεχόμενη αλλοίωση δεδομένων. Εάν οι μεταξύ των μερών επικοινωνίες βασίζονται σε μη κρυπτογραφημένα πρωτόκολλα, όπως το απλό HTTP πρωτόκολλο, ένας επιτιθέμενος με πρόσβαση στο δίκτυο θα

μπορέσει να διαβάσει ευαίσθητες πληροφορίες, ή/και να τις παραλλάξει. Για να αντιμετωπιστεί αυτή η απειλή, θα πρέπει τα κανάλια επικοινωνίας να βασίζονται στο ασφαλές πρωτόκολλο HTTPS/τελευταία έκδοση πρωτοκόλλου TLS, ώστε ένας κρυφακουστής να μην είναι σε θέση να διαβάσει/αλλοιώσει την ανταλλασσόμενη πληροφορία.

Παράγοντες απειλών: Εξωτερικοί, Εσωτερικοί

4.1.2 Μη εξουσιοδοτημένη πρόσβαση στο σύστημα

Σε συστήματα που δεν υλοποιούν ισχυρούς μηχανισμούς αυθεντικοποίησης χρήστη, υπάρχει σοβαρός κίνδυνος μη εξουσιοδοτημένα άτομα να αποκτήσουν πρόσβαση όχι μόνο στο VCS, αλλά και στον ίδιο τον αγωγό CI/CD. Οι κωδικοί πρόσβασης πρέπει όχι μόνο να είναι ισχυροί, αλλά να απαιτείται και αυθεντικοποίηση πολλαπλών παραγόντων (Multi-factor Authentication - MFA) ή κρυπτογραφικά κλειδιά για την αυθεντικοποίηση ενός χρήστη. Σε περίπτωση που ένας επιτιθέμενος αποκτήσει πρόσβαση στο σύστημα, ανάλογα με τα δικαιώματα του έγκυρου χρήστη που σφετερίζεται, μπορεί να διαβάσει ευαίσθητους κωδικούς, να αλλάξει τον κώδικα της εφαρμογής ή ακόμη και να παύσει την παροχή της υπηρεσίας της εφαρμογής προς τους χρήστες. Επιπρόσθετα, λόγω ελλιπούς παραμετροποίησης του συστήματος ελέγχου ταυτότητας και απόδοσης δικαιωμάτων, ένας έγκυρος χρήστης μπορεί να αποκτήσει αυξημένα δικαιώματα, με αποτέλεσμα να είναι σε θέση να εκτελεί διαχειριστικές εργασίες στο σύστημα. Για να αντιμετωπιστεί αυτή η απειλή, πρέπει ο έλεγχος πρόσβασης σε όλες τις δραστηριότητες να είναι ενιαίος και να λειτουργεί με βάση τους ίδιους κανόνες και την αρχή των ελάχιστων δικαιωμάτων (least privileges principle).

Παράγοντες απειλών: Εξωτερικοί, Εσωτερικοί

4.1.3 Εκτέλεση αγωγού από χρήστη χωρίς την ανάλογη εξουσιοδότηση

Σε κανονικές συνθήκες, ο αγωγός που εκτελείται μετά από συνένωση 2 κλαδιών (πχ. feature & main branch) του κώδικα πρέπει προηγουμένως να έχει πάρει έγκριση από έναν χρήστη με αυξημένα καθήκοντα, ο οποίος είναι επιφορτισμένος με την αρμοδιότητα να εγκρίνει ή να απορρίπτει τις αλλαγές στον κύριο πηγαίο κώδικα της εφαρμογής. Εάν οι αγωγοί ξεκινούν ανεξέλεγκτα, ένας χρήστης θα μπορούσε να τροποποιήσει το κύριο κλαδί του αποθετηρίου με κακόβουλο κώδικα, προκαλώντας ζημιά στον οργανισμό, είτε λόγω μη παροχής υπηρεσίας προς τους πελάτες, είτε ακόμα και με τη διαρροή μυστικών/δεδομένων πελατών.

Παράγοντες απειλών: Εξωτερικοί, Εσωτερικοί, Ακούσιοι εσωτερικοί παράγοντες

4.1.4 Τροποποίηση του αγωγού από απλό χρήστη

Εάν το σύστημα CI/CD εκτελείται δυναμικά με βάση αρχείο ορισμού αγωγού CI/CD που υποβάλλεται μαζί με τον υπόλοιπο πηγαίο κώδικα, υπάρχει ο κίνδυνος ένας απλός χρήστης να τροποποιήσει το αρχείο αυτό ακούσια ή και κακόβουλα. Αυτή η τροποποίηση μπορεί να οδηγήσει σε αφαίρεση ή αλλοίωση των σταδίων του αγωγού, σε φανέρωση συνθηματικών ή σε εγκατάσταση κακόβουλου κώδικα σε τρίτα συστήματα. Για την αντιμετώπιση αυτού του κινδύνου, το αρχείο που περιγράφει τον αγωγό πρέπει να είναι προστατευμένο από τροποποιήσεις απλών χρηστών.

Παράγοντες απειλών: Εξωτερικοί, Εσωτερικοί, Ακούσιοι εσωτερικοί παράγοντες

4.1.5 Αποθήκευση συνθηματικών σε μορφή κειμένου

Είναι μια συνηθισμένη κακή πρακτική για τους μηχανικούς κώδικα να αποθηκεύουν συνθηματικά και κωδικούς πρόσβασης μέσα στον πηγαίο κώδικα, για διευκόλυνση της ανάπτυξης του λογισμικού. Δε χρειάζεται να τονισθεί ιδιαίτερα ότι αυτή η πρακτική εγκυμονεί σοβαρότατους κινδύνους για την ακεραιότητα των εμπλεκόμενων συστημάτων, καθώς οποιοσδήποτε υπάλληλος/επισκέπτης που έχει έστω δικαίωμα θέασης, είναι σε θέση να καταγράψει αυτά τα διαπιστευτήρια και να τα διαθέσει σε τρίτους ή να τα εκμεταλλευτεί προς ιδίους σκοπούς. Συνεπώς, τα όποια συνθηματικά δύναται να χρησιμοποιηθούν, πρέπει οπωσδήποτε να βρίσκονται ασφαλώς κρυπτογραφημένα, και να μην εμφανίζονται είτε καθόλου ή να τροποποιούνται μόνο κατ’απαιτήση χρηστών που κατέχουν την πιο υψηλή διαβάθμιση.

Παράγοντες απειλών: Εξωτερικού, Εσωτερικοί, Ακούσιοι εσωτερικοί παράγοντες

4.1.6 Άρνηση υπηρεσίας (Denial of Service)

Εφόσον ο αγωγός εκτελείται μετά από αίτημα για συνένωση κλαδιών του κώδικα, είναι δυνατόν ένας κακόβουλος χρήστης να προκαλέσει την εκτέλεση τόσων στιγμιotypών του αγωγού, ώστε να παραλύσει όλο το σύστημα CI/CD. Γι αυτό το λόγο γίνεται φανερή η ανάγκη επιβολής ενός μηχανισμού ελέγχου της εκτέλεσης των αγωγών, όπως είναι η έγκριση από χρήστη ανώτερου επιπέδου.

Επιπρόσθετα, ένας επιτιθέμενος θα μπορούσε να εξαπολύσει μια παραδοσιακή επίθεση άρνησης υπηρεσίας στο CI/CD σύστημα. Προς αποφυγή αυτού του κινδύνου, κατάλληλα φυσικά/λογισμικά αναχώματα (IDS / firewalls/ Όρια κατανάλωσης πόρων) πρέπει να προστατεύουν την υφιστάμενη υποδομή.

Παράγοντες απειλών: Εξωτερικοί, Εσωτερικοί

4.1.7 Εκτέλεση και εγκατάσταση παλαιού στιγμιότυπου του κώδικα

Σε περίπτωση που η επικοινωνία με τον εξυπηρετητή CI/CD γίνεται σε μη κρυπτογραφημένα κανάλια, ένας επιτιθέμενος θα μπορούσε να καταγράψει ένα αίτημα εκτέλεσης αγωγού, και σε ύστερο χρόνο να το επαναλάβει με παραλήπτη τον εξυπηρετητή, με συνέπεια να εγκατασταθεί στο περιβάλλον δοκιμής / παραγωγής ένα παλαιότερο στιγμιότυπο της εκάστοτε εφαρμογής. Αυτός ο κίνδυνος εύκολα αποφεύγεται με χρήση του πρωτοκόλλου HTTPS / τη χρήση χρονοσφραγίδων / τον έλεγχο για παλαιότερη έκδοση κλαδιού κατά την επικοινωνία με τον εξυπηρετητή.

Παράγοντες απειλών: Εξωτερικοί, Εσωτερικοί

4.1.8 Τροποποίηση του αγωγού/Πρόσβαση συνθηματικών από λογαριασμό διαχειριστή

Σε περίπτωση που ο μηχανισμός αυθεντικοποίησης δεν είναι επαρκώς ισχυρός (εφαρμογή πολιτικών πολύπλοκων συνθηματικών / υποχρεωτική χρήση MFA), ένας επιτιθέμενος θα μπορούσε να παραβιάσει το λογαριασμό ενός διαχειριστή του εξυπηρετητή CI/CD και να τροποποιήσει το αρχείο που καθορίζει τα βήματα του αγωγού, ή να αποκτήσει πρόσβαση στα συνθηματικά, εφόσον αυτά εμφανίζονται στον διαχειριστή, που απαιτούνται για την εγκατάσταση του κώδικα στα διάφορα συστήματα. Εδώ και πάλι η επιφάνεια επίθεσης μπορεί να συρρικνωθεί χρησιμοποιώντας MFA για την ασφαλή ταυτοποίηση των διαχειριστών και εργαλεία που επιτρέπουν την μετατροπή αλλά όχι την εμφάνιση των μυστικών.

Παράγοντες απειλών: Εξωτερικοί, Εσωτερικοί

4.1.9 Θέαση συνθηματικών πρόσβασης σε τρίτα συστήματα από απλούς χρήστες

Στον εξυπηρετητή CI/CD είναι συχνά απαραίτητο να αποθηκεύονται κρυπτογραφικά κλειδιά ή συνθηματικά, προκειμένου να είναι δυνατή η πρόσβαση και η εγκατάσταση από τον αγωγό σε τρίτα συστήματα. Αυτά τα συνθηματικά πρέπει να προστατεύονται και να μην είναι

ορατά από απλούς χρήστες, καθώς και να μην εμφανίζονται σε κανένα αρχείο καταγραφής κατά την εκτέλεση του αγωγού.

Παράγοντες απειλών: Εξωτερικοί, Εσωτερικοί, Ακούσιοι εσωτερικοί παράγοντες

4.1.10 Εκτέλεση απομακρυσμένου κώδικα στον εξυπηρετητή CI/CD

Εάν η πλατφόρμα CI/CD που χρησιμοποιείται δεν αποτρέπει την χρήση εντολών σε πεδία κειμένου, όπως για παράδειγμα κατά την υποβολή σχολίων, από τους χρήστες της, υπάρχει η ευκαιρία για κακόβουλους χρήστες να εκμεταλλευτούν τυχόν ευπάθειες και να προκαλέσουν την εκτέλεση κώδικα στον κεντρικό εξυπηρετητή. Αυτή η απειλή περιορίζεται πρωτίστως με την επιλογή του πλέον ασφαλούς λογισμικού CI/CD και δευτερευόντως με την τακτική αναβάθμιση αυτού. Με αυτόν τον τρόπο, προστατεύονται αυτά τα πεδία καθαρίζοντας και κωδικοποιώντας κατάλληλα το περιεχόμενο αυτών, προκειμένου να μειώνεται η διαθέσιμη επιφάνεια επίθεσης.

Παράγοντες απειλών: Εξωτερικοί, Εσωτερικοί

4.1.11 Φόρτωση κακόβουλων εκτελέσιμων/βιβλιοθηκών στο αποθετήριο

Εάν κάποιος κακόβουλος χρήστης αποκτήσει δικαίωμα πρόσβασης στο αποθετήριο τεχνημάτων/εικόνων, θα είναι ικανός να ανεβάσει κακόβουλα αρχεία, θέτοντας σε κίνδυνο την ασφάλεια των μελλοντικών αγωγών, εάν αυτά τα αρχεία αξιοποιηθούν. Εδώ και πάλι η επιφάνεια επίθεσης μπορεί να συρρικνωθεί χρησιμοποιώντας ισχυρότερες μεθόδους αυθεντικοποίησης όπως MFA, σε συνδυασμό με την πρόσβαση στο αποθετήριο με βάση αυστηρούς πρόσβασης με βάση τους ρόλους.

Παράγοντες απειλών: Εξωτερικοί, Εσωτερικοί

4.1.12 Χρήση κακόβουλου κώδικα από τρίτες βιβλιοθήκες

Σχεδόν αναπόφευκτα οι εφαρμογές που υλοποιούνται σήμερα στηρίζουν μεγάλο βαθμό της λειτουργικότητας τους σε έτοιμο κώδικα με τη μορφή χρήσιμων βιβλιοθηκών που στεγάζεται σε κοινά αποθετήρια, τα οποία και αξιοποιούνται από τους προγραμματιστές και ενσωματώνονται στο τελικό προϊόν. Είναι πολύ πιθανό αυτές οι βιβλιοθήκες να εμπεριέχουν κάποια ευπάθεια, ή να εισάγουν σκόπιμα μια τρωτότητα, η οποία τελικά να θέσει σε κίνδυνο την ακεραιότητα/εμπιστευτικότητα ή διαθεσιμότητα των υπηρεσιών της εφαρμογής. Γι' αυτό το σκοπό είναι απαραίτητο πλην του πηγαίου κώδικα, να ελέγχονται και όλες οι τρίτες βιβλιοθήκες που

εμπεριέχονται στην εφαρμογή, πριν ενσωματωθούν στον κύριο κορμό του αποθετηρίου. Με αυτόν τον τρόπο θωρακίζεται η διαδικασία DevOps, με εξαίρεση βέβαια της ευπάθειας zero-day, για τις οποίες δεν μπορεί να υπάρξει αντίχενυση άμεσα.

Παράγοντες απειλών: Εξωτερικοί, Εσωτερικοί, Ακούσιοι εσωτερικοί παράγοντες

4.1.13 Έγχυση κακόβουλων βιβλιοθηκών

Αυτή η απειλή παρουσιάζεται όταν ένας ενδιάμεσος (Man-in-the-Middle) αντικαθιστά μια βιβλιοθήκη ή ένα αρχείο με ένα δικό του παραγόμενο αρχείο, όταν το σύστημα την/το ανακτά από ένα αποθετήριο. Ο επιτιθέμενος στέκεται ανάμεσα στο σύστημα CI/CD και τα κοινώς προσβάσιμα αποθετήρια, και αντικαθιστά την έγκυρη βιβλιοθήκη με μια δικής του κατασκευής. Όταν τελικά αυτή η βιβλιοθήκη ενσωματωθεί με τον παραγόμενο κώδικα, τίθεται σε κίνδυνο η λειτουργία του τελικού προϊόντος αλλά και των χρηστών αυτού. Σαν αντίμετρο, πρέπει να γίνει έλεγχος με διαγνωστικό εργαλείο ευπαθειών κατά την εκτέλεση του αγωγού και παραγωγή του λογισμικού, καθώς και hash-based έλεγχος ταυτότητας της βιβλιοθήκης κατά τη λήψη της, ώστε να βεβαιωθεί η μη τροποποίηση αυτής, ή χρήση ασφαλούς επικοινωνίας με αυθεντικοποίηση και των 2 μερών.

Παράγοντες απειλών: Εξωτερικοί, Εσωτερικοί

4.1.14 Μη εξουσιοδοτημένη πρόσβαση στο δοκιμαστικό/παραγωγικό περιβάλλον

Λόγω ελλειπών/αδύναμων κανόνων πρόσβασης/αυθεντικοποίησης, ένας επιτιθέμενος θα μπορούσε να αποκτήσει πρόσβαση σε εικονικές μηχανές που υποστηρίζουν τη δοκιμαστική ή ακόμα και την παραγωγική λειτουργία. Σε αυτό το σενάριο θα μπορούσε να προκαλέσει ζημιά ή να αποκαλύψει μυστικά. Η πρόσβαση σε αυτά τα μηχανήματα (δηλ. τους εξυπηρετητές) θα πρέπει να επιτρέπεται μέσω του τείχους προστασίας μόνο από συγκεκριμένες πηγές που κατέχουν το ορθό κρυπτογραφικό κλειδί.

Παράγοντες απειλών: Εξωτερικοί, Εσωτερικοί

4.1.15 Υποβολή κακόβουλου κώδικα από εξουσιοδοτημένο χρήστη

Σε κάποιες περιπτώσεις, είναι δυνατόν ένας εσωτερικός και εξουσιοδοτημένος χρήστης να υποβάλλει σκόπιμα κακόβουλο πηγαίο κώδικα για λόγους εκδίκησης ή παράνομου οφέλους. Για να αποφευχθούν τέτοιες περιπτώσεις, πλην των αυτοματοποιημένων ελέγχων που

διενεργούνται κατά την εκτέλεση του αγωγού, υπάρχει επιπλέον ο χειροκίνητος έλεγχος νέου κώδικα από υπεύθυνο χρήστη πριν την ενσωμάτωση (git merge) των νέων κλαδιών στο βασικό.

Παράγοντες απειλών: Εσωτερικοί

4.1.16 Υποβολή ευάλωτου κώδικα από εξουσιοδοτημένο χρήστη

Στον πυρήνα κάθε είδους λογισμικού βρίσκεται ο πηγαίος κώδικας που εκτελεί τελικά τις βασικές λειτουργίες της εκάστοτε εφαρμογής. Εάν αυτός ο κώδικας περιέχει ευπάθειες, οι οποίες ανοίγουν τον δρόμο σε έναν επιτιθέμενο να τις εκμεταλλευτεί, ολόκληρη η υποδομή καθίσταται ευάλωτη. Το ίδιο ισχύει και για τις βιβλιοθήκες στις οποίες στηρίζεται το λογισμικό, καθώς μια ευπάθεια είναι εκμεταλλεύσιμη από έναν κακόβουλο χρήστη, ανεξάρτητα από την πηγή της. Για αυτό το λόγο, τόσο ο πηγαίος κώδικας όσο και οι τρίτες βιβλιοθήκες του λογισμικού πρέπει να υπόκεινται σε έλεγχο όχι μόνο κατά τη συγγραφή ή την υποβολή στο αποθετήριο, αλλά και περιοδικά, εφόσον ευπάθειες ανακαλύπτονται συνεχώς, χωρίς να εξαρτώνται από τη δραστηριότητα των μηχανικών λογισμικού.

Παράγοντες απειλών: Εσωτερικοί, Ακούσιοι εσωτερικοί παράγοντες

4.1.17 Ευάλωτοι εξυπηρετητές

Όπως ήδη υπογραμμίστηκε στο κεφάλαιο των βασικών εννοιών της παρούσας διπλωματικής εργασίας, σε αντίθεση με τις παραδοσιακές εικονικές μηχανές, τα δοχεία μοιράζονται το κέλυφος (kernel) του εξυπηρετητή που τα φιλοξενεί. Είναι λοιπόν αδήριτη η ανάγκη ο εξυπηρετητής να έχει περιχαρακωθεί υπό το πρίσμα της ασφάλειας στο μέτρο του δυνατού. Προς το σκοπό αυτό επιστρατεύονται οι εξής μέθοδοι:

- Συρρίκνωση του λειτουργικού συστήματος που εγκαθίσταται στον εξυπηρετητή. Με την ελαχιστοποίηση των πακέτων και υπηρεσιών που βρίσκονται διαθέσιμα σε ένα λειτουργικό σύστημα, μειώνεται ταυτόχρονα και η διαθέσιμη επιφάνεια επίθεσης. Σαφώς οι λειτουργικότητες ασφαλείας παραμένουν λειτουργικές στο ακέραιο. Μάλιστα, υπάρχουν διαθέσιμες ειδικές εκδόσεις του λειτουργικού Linux (CoreOS, PhotonOS), ελαχιστοποιημένες και ειδικά διαμορφωμένες για τη χρήση δοχείων, με ειδικούς μηχανισμούς επιβολής κανόνων περιορισμένης πρόσβασης σε πόρους και φακέλους του συστήματος (AppArmor, SELinux).

- Χρήση ειδικών εργαλείων αξιολόγησης ασφαλείας του λειτουργικού συστήματος. Κάνοντας χρήση εργαλείων, όπως πχ. το Lynis⁶, μπορεί να δοθεί μια καλή εικόνα για τυχόν τρωτότητες ενός λειτουργικού συστήματος (ΛΣ). Επιπρόσθετα, το επιλεγμένο ΛΣ πρέπει να υπόκειται σε όλες τις διαθέσιμες αναβαθμίσεις με αυτοματοποιημένο τρόπο, ο οποίος θα εφαρμόζεται όχι σαν πρόσθετο σε υφιστάμενες εικονικές μηχανές, αλλά με πλήρη αντικατάσταση της εικονικής μηχανής με νέα, ενημερωμένη στην τελευταία επίσημη έκδοση μέσω του υπερεπόπτη (hypervisor). Τακτικές και αυτοματοποιημένες αναβαθμίσεις λογισμικού πρέπει να εφαρμόζονται και στον ίδιο τον υπερεπόπτη.
- Ο εξυπηρετητής πρέπει να προστατεύεται από υλικό ή λογισμικό αναχώματος (firewall) / ανίχνευσης εισβολής (Intrusion Detection System), προκειμένου να περιορίζεται ο κίνδυνος επιθέσεων τύπου (D)DoS. Επιπρόσθετα, πρέπει η δικτυακή παραμετροποίηση να μην επιτρέπει την πρόσβαση σε σύστημα από χώρους που δεν δικαιούνται αντίστοιχα δικαιώματα.
- Τέλος, εφόσον η διαχείριση των δοχείων θα γίνεται αποκλειστικά μέσω του ενορχηστρωτή, οι εικονικές μηχανές καθώς και τα δοχεία που στεγάζουν θα πρέπει να βρίσκονται υπό συνεχή παρακολούθηση από λογισμικό καταγραφής, το οποίο θα αναφέρει τυχόν απροσδόκητες προσβάσεις ή μη προβλεπόμενες διεργασίες που εκτελούνται.

Παράγοντες απειλών: Εξωτερικοί, Εσωτερικοί, Ακούσιοι εσωτερικοί παράγοντες

4.2 Ειδικές απειλές ενάντια σε συστήματα που βασίζονται σε δοχεία (Containers)

Πέραν από τις απειλές που μπορεί να δεχτεί ένα σύστημα CI/CD υπάρχουν και ιδιαίτεροι κίνδυνοι που στοχοποιούν ειδικά τα συστήματα που βασίζονται σε δοχεία. Σε αυτή την ενότητα θα αναλυθούν αυτοί οι κίνδυνοι, οι οποίοι μπορεί σε ένα βαθμό να επικαλύπτονται με τις απειλές της προηγούμενης ενότητας. Οι επιφάνειες που προσφέρονται για την εκδήλωση επίθεσης σε βάρος ενός συστήματος CI/CD, το οποίο βασίζεται σε δοχεία, όπως διαβάζουμε στα άρθρα [56] & [57], είναι οι παρακάτω:

⁶ <https://cisofy.com/lynis>

4.2.1 Επικίνδυνα διαμορφωμένες εικόνες δοχείων

Η εικόνα είναι ένα σύνολο οδηγιών που περιγράφουν πως θα σχηματιστεί και θα εκτελεστεί ένα δοχείο. Η εικόνα ενός δοχείου σχηματίζεται συνήθως με βάση ένα αρχείο Dockerfile. Η διαμόρφωση αυτών των οδηγιών απαιτεί μεγάλη προσοχή, καθώς σε αυτό το στάδιο μπορεί να εισαχθούν ευπάθειες, τις οποίες μπορούν - και έχουν - εκμεταλλευτεί στο παρελθόν επιτιθέμενοι. Κάποιες από τις ευπάθειες αυτές είναι οι εξής:

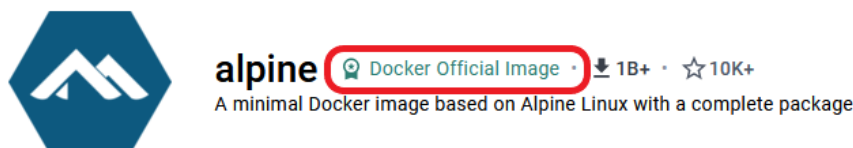
- **Στη βασική εικόνα**

Η κάθε εικόνα ενός δοχείου ξεκινά να οικοδομείται πάνω σε μια βασική εικόνα. Συνήθως είναι η πρώτη μη σχολιασμένη γραμμή που περιέχεται στο αρχείο Dockerfile και σημαίνεται με τη λέξη FROM ως εξής:

```
#Stage 1 Image that will create production files  
FROM node:14.4-alpine AS NODE_BUILD_IMAGE
```

Εικόνα 11 - Βασική εικόνα

Η εικόνα αυτή πρέπει οπωσδήποτε να προέρχεται από ένα αξιόπιστο αποθετήριο, όπως είναι για παράδειγμα το Docker Hub [58]. Πέραν του αποθετηρίου, μιας και αποτελεί το βασικό στρώμα της εικόνας, η βασική εικόνα πρέπει να είναι όσο το δυνατόν πιο αξιόπιστη γίνεται. Για το σκοπό αυτό, οι εικόνες που προέρχονται από έμπιστους οργανισμούς είναι συχνά κατάλληλα επισημασμένες, όπως φαίνεται και στην παρακάτω εικόνα.



Εικόνα 12 - Επίσημη εικόνα οργανισμού

Επιπρόσθετα, μια καλή πρακτική για την μείωση της επιφάνειας επίθεσης μιας βασικής εικόνας είναι η εικόνα αυτή να περιέχει μόνο τα απολύτως απαραίτητα εργαλεία και λειτουργικότητες για το σκοπό για τον οποίο προορίζεται. Όπως είναι η εικόνα alpine στην Εικόνα 12, από την οποία έχουν αφαιρεθεί όλες οι πλεονάζουσες λειτουργικότητες, και ως αποτέλεσμα, όλα τα πιθανά πλεονάζοντα σημεία στα οποία μπορεί να εμφανιστεί νέα

ευπάθεια. Τέλος, σε σχέση με την βασική εικόνα, θα πρέπει πάντοτε να γίνεται έλεγχος για νέα έκδοση αυτής, καθώς είναι πολύ πιθανό να έχουν ανακαλυφθεί / επιδιορθωθεί τυχόν ευπάθειες με μια νεότερη έκδοση.

TAG

[alpine3.17](#)

Last pushed 3 months ago by [doijanky](#)

DIGEST

[24d2410f6efb](#)

[ccf066d2cfec](#)

[ea3dd18d6068](#)

+4 more...

OS/ARCH

linux/386

linux/amd64

linux/arm/v6

VULNERABILITIES

1 C

5 H

2 M

1 L

1 C

5 H

2 M

1 L

1 C

5 H

2 M

1 L

TAG

[alpine3.18](#)

Last pushed a month ago by [doijanky](#)

DIGEST

[ab4363f4aff0](#)

[7e528502b614](#)

[a7b96c5fde81](#)

+4 more...

OS/ARCH

linux/386

linux/amd64

linux/arm/v6

VULNERABILITIES

2 H

2 M

0 L

2 H

2 M

0 L

2 H

2 M

0 L

Εικόνα 13 - Επιδιόρθωση ευπαθειών σε βασική εικόνα
(πηγή: <https://hub.docker.com/>)

Αυτό γίνεται φανερό με μια σύντομη σύγκριση εκδόσεων της ανωτέρω εικόνας alpine (δείτε Εικ. 13), όπου η νεότερη έκδοση επιδιόρθωσε ευπάθειες, εκ των οποίων η μια ήταν μάλιστα κρίσιμη. Όπως έγινε ήδη φανερό, ακόμα και έμπιστες εικόνες μπορεί να περιέχουν ευπάθειες. Γι αυτό το σκοπό, απαιτείται ένα σύστημα CI/CD που να διασφαλίζει ότι αυτές οι ευπάθειες δε θα φτάσουν στο στάδιο εγκατάστασης σε δοκιμαστικό / παραγωγικό περιβάλλον.

- **Σε εικόνα με πλεονάζοντα πακέτα**

Όπως σημειώθηκε και στο προηγούμενο σημείο, η κάθε εικόνα αποτελείται από ιεραρχικά τοποθετημένα στρώματα, σε μορφή πυραμίδας. Οι πιο βελτιστοποιημένες εικόνες, όπως είναι η alpine, έχουν απορρίψει τις μη αναγκαίες λειτουργικότητες, μειώνοντας έτσι δραματικά το μέγεθός τους αλλά και την διαθέσιμη επιφάνεια επίθεσης. Σε πολλές περιπτώσεις όμως, οι μηχανικοί λογισμικού δημιουργούν μια εικόνα, η οποία μπορεί να

περιέχει τόσο τις βιβλιοθήκες και τα εργαλεία που απαιτούνται για να παραχθεί ένα εκτελέσιμο αρχείο, όσο και τις βιβλιοθήκες και τα εργαλεία που είναι αναγκαία για την εκτέλεση του ίδιου του εκτελέσιμου αρχείου. Αυτή η κακή πρακτική όχι μόνο διογκώνει το μέγεθος της εικόνας και άρα το χρόνο εκτέλεσης του αγωγού, αλλά επιπλέον αυξάνει την επιφάνεια επίθεσης. Η σωστή πρακτική σε τέτοιες περιπτώσεις απαιτεί το Dockerfile να υλοποιείται με την τεχνική **Multi-Stage Build (κατασκευή πολλαπλών φάσεων)**, κατά την οποία τα προκαταρκτικά στρώματα της εικόνας απορρίπτονται μόλις ολοκληρώσουν τη λειτουργικότητά τους, αφήνοντας προς αποθήκευση ένα μικρότερο και ασφαλές δοχείο, όπως φαίνεται στην παρακάτω εικόνα.

```
#####Frontend Dockerfile#####  
#Stage 1 Image that will create production files  
FROM node:14.4-alpine AS NODE_BUILD_IMAGE  
WORKDIR /app  
COPY package*.json ./  
RUN npm install;  
COPY ./ .  
RUN npm run build  
  
#Stage 2 Image that will host the files in an nginx http server  
FROM nginx as foreigners_frontend  
RUN mkdir /app  
COPY --from=NODE_BUILD_IMAGE /app/dist /app  
COPY nginx.conf /etc/nginx/nginx.conf  
COPY default.conf.template /etc/nginx/templates/  
#####
```

Εικόνα 14 - Multi-Stage Dockerfile

- **Η εικόνα εκτελείται ως διαχειριστής (root)**

Με το [59], οι Jian and Chen έδειξαν ότι είναι πιθανό, με τις κατάλληλες συνθήκες και εκδόσεις, παρά τη στεγανοποίηση που συμβαίνει ανάμεσα στα δοχεία και τον (φυσικό/εικονικό) εξυπηρετητή, ένας κακόβουλος χρήστης να κατορθώσει να αποδράσει από ένα δοχείο. Εξ'ορισμού, τα δοχεία εκτελούνται με δικαιώματα διαχειριστή (root) εσωτερικά, όμως παραδόξως, εάν δεν έχουν ρυθμιστεί κατάλληλα οι χώροι ονομάτων (namespaces) [62] του εξυπηρετητή, ο διαχειριστής εντός του δοχείου μπορεί να ισοδυναμεί με διαχειριστή και εκτός του δοχείου. Εύκολα λοιπόν εξάγεται το συμπέρασμα, ότι η εκτέλεση ενός δοχείου ως διαχειριστής αποτελεί απειλή, η οποία όμως μπορεί εύκολα να αντιμετωπιστεί καθορίζοντας έναν διαφορετικό χρήστη, όπως φαίνεται παρακάτω (Εικ. 15).

```

COPY default.conf.template /etc/nginx/templates/
RUN touch /var/run/nginx.pid && \
    chown -R 1001:1001 /var/run/nginx.pid && \
    chown -R 1001:1001 /var/cache/nginx
USER 1001
#####

```

Εικόνα 15 - Non-root user στο Dockerfile

Παράγοντες απειλών: Εξωτερικοί, Εσωτερικοί, Ακούσιοι εσωτερικοί παράγοντες

4.2.2 *Ευάλωτος εξυπηρετητής κατά την οικοδόμηση του δοχείου*

Εάν το μηχάνημα που αναλαμβάνει τη κατασκευή των δοχείων έχει παραβιαστεί, είναι εξαιρετικά πιθανό ότι ένας επιτιθέμενος θα μπορέσει να εισάγει το δικό του κώδικα ή βιβλιοθήκες μέσα σε ένα δοχείο και την αντίστοιχη εικόνα του, και ως εκ τούτου να αποκτήσει τελικά πρόσβαση σε οποιοδήποτε μηχάνημα εκτελεστεί τελικά αυτό το δοχείο, εάν δεν έχουν τεθεί κανόνες που να αποτρέπουν κάτι τέτοιο, όπως για παράδειγμα ο κανόνας που απαγορεύει σε ένα δοχείο να εκτελείται ως διαχειριστής (root), όπως είδαμε προηγουμένως στην Ενότητα 4.2.1. Λύση απέναντι στη συγκεκριμένη απειλή αποτελεί η σκλήρυνση του εικονικού μηχανήματος-εξυπηρετητή, ώστε να μειωθούν οι πιθανότητες παραβίασης αυτού.

Παράγοντες απειλών: Εξωτερικοί, Εσωτερικοί, Ακούσιοι εσωτερικοί παράγοντες

4.2.3 *Επίθεση εφοδιαστικής αλυσίδας (Supply Chain Attack)*

Μετά το κτίσιμο της εικόνας από το εξειδικευμένο μηχάνημα, η εικόνα αποθηκεύεται σε ένα αποθετήριο. Είναι πιθανό, ένας επιτιθέμενος να αντικαταστήσει την έγκυρη εικόνα με μια δικής του κατασκευής, ενεργώντας είτε ως ενδιάμεσος (Man in the Middle) είτε αποκτώντας πρόσβαση στο ίδιο το αποθετήριο. Η λύση σε τέτοιες απειλές είναι η υπογραφή της εικόνας και η επαλήθευση αυτής με βάση το δημόσιο γνωστό κλειδί του έγκυρου παραγωγού λογισμικού (δηλ. του οργανισμού που παράγει και διαθέτει την εικόνα) κατά τη λήψη από το αποθετήριο, συνήθως κάνοντας χρήση του εργαλείου Notary [60], το οποίο βασίζεται στο The Update Framework⁷.

Παράγοντες απειλών: Εξωτερικοί, Εσωτερικοί

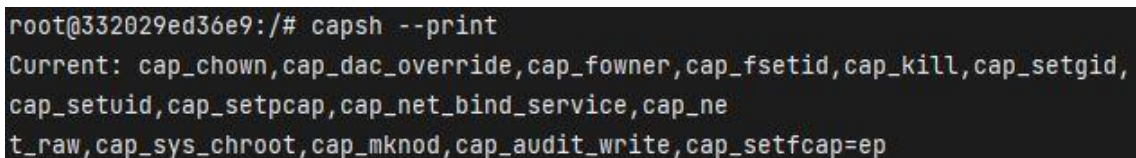
⁷ <https://theupdateframework.io/>

4.2.4 Επικίνδυνες ρυθμίσεις κατά την εκτέλεση του δοχείου

Ήδη έγινε αναφορά σε λανθασμένες ρυθμίσεις ή οδηγίες μέσα σε ένα Dockerfile, από τις οποίες μπορεί να προκύψει ευπάθεια σε μια εικόνα, και ως αποτέλεσμα σε ένα δοχείο που θα δημιουργηθεί από την εικόνα-πηγή. Ακόμη όμως και αν οι προδιαγραφές της εικόνας είναι σωστές, υπάρχει ο κίνδυνος το δοχείο να εκτελεστεί με τέτοιο τρόπο, που τελικά θα καταστεί επικίνδυνο. Οι τρόποι αυτοί είναι οι εξής:

- **Η εκτέλεση με τη σημαία “privileged”**

Συνολικά στον πυρήνα του Linux εντοπίζονται περίπου 30 δυνατότητες (capabilities). Οι ενέργειες που επιτρέπεται να εκτελέσει ένα thread ή ένα (εκτελέσιμο) αρχείο εξαρτώνται άμεσα από τις δυνατότητες που του έχουν δοθεί. Εάν μια διεργασία δεν εκτελείται από τον διαχειριστή του συστήματος (root), οι δυνατότητες που της ανατίθενται από τον πυρήνα είναι δραστηκά μειωμένες. Ο Docker daemon από προεπιλογή ελαττώνει τις δυνατότητες που αποδίδει σε ένα δοχείο, όταν αυτό εκτελείται, στις άκρως απαραίτητες [61]. Σε αντιδιαστολή αυτού, όταν ένα δοχείο εκτελείται με τη σημαία “privileged” (προνομιούχο), του αποδίδονται όλες οι δυνατότητες του πυρήνα, όσες ακριβώς διαθέτει και ένας πραγματικός διαχειριστής του συστήματος. Ενδεικτικά μπορούμε να συγκρίνουμε τις δυνατότητες του ίδιου δοχείου, αρχικά χωρίς και στη συνέχεια με την εν λόγω σημαία (δείτε τις επόμενες δύο εικόνες, αντίστοιχα).



```
root@332029ed36e9:/# capsh --print
Current: cap_chown,cap_dac_override,cap_fowner,cap_fsetid,cap_kill,cap_setgid,
cap_setuid,cap_setpcap,cap_net_bind_service,cap_net_raw,cap_sys_chroot,cap_mknod,cap_audit_write,cap_setfcap=ep
```

Εικόνα 16 - Δυνατότητες μη προνομιούχου (unprivileged) δοχείου


```
root@5f69b0dd2064:/# capsh --print
Current: =ep
Bounding set =cap_chown,cap_dac_override,cap_dac_read_search,cap_fowner,cap_fs
etid,cap_kill,cap_setgid,cap_setuid,cap_setpcap,cap_linux_immutable,cap_net_bi
nd_service,cap_net_broadcast,cap_net_admin,cap_net_raw,cap_ipc_lock,cap_ipc_ow
ner,cap_sys_module,cap_sys_rawio,cap_sys_chroot,cap_sys_ptrace,cap_sys_pacct,c
ap_sys_admin,cap_sys_boot,cap_sys_nice,cap_sys_resource,cap_sys_time,cap_sys_t
ty_config,cap_mknod,cap_lease,cap_audit_write,cap_audit_control,cap_setfcap,ca
p_mac_override,cap_mac_admin,cap_syslog,cap_wake_alarm,cap_block_suspend,cap_a
udit_read,cap_perfmon,cap_bpf,cap_checkpoint_restore
```

Εικόνα 17 - Δυνατότητες προνομιούχου (privileged) δοχείου

Εύκολα γίνεται αντιληπτό το μέγεθος της βλάβης που μπορεί να καταφέρει ένας επιτιθέμενος, εάν πάρει τον έλεγχο ενός δοχείου που έχει πλήρεις δυνατότητες στον εξυπηρετητή, γι’ αυτό και η προκείμενη σημαία πρέπει να αποφεύγεται.

- **Η σύνδεση του δοχείου με κρίσιμους φακέλους του συστήματος**

Κατά την έναρξή του, ένα δοχείο μπορεί να συνδεθεί αμφίδρομα με έναν φάκελο του συστήματος του εξυπηρετητή, προκειμένου να μπορέσει να αντλήσει και να γράψει δεδομένα, τα οποία και θα επιβιώνουν ανεξάρτητα από τον κύκλο ζωής του δοχείου. Αυτή η σύνδεση γίνεται με τη σημαία **-v** κατά την εκτέλεση του δοχείου. Με την παρακάτω εντολή, για παράδειγμα, ο φάκελος **/usr/data** του εξυπηρετητή συγχρονίζεται αμφίδρομα με τον φάκελο **/data** του δοχείου.

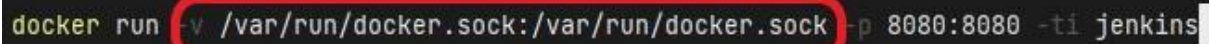
```
docker run -v /usr/data:/data --name=nginx-container -d nginx
```

Εικόνα 18 - Σύνδεση φακέλων δοχείου

Ωστόσο, πίσω από τη χρήση αυτής της σημαίας κρύβονται δυνητικές απειλές, καθώς εάν ακούσια - ή σκόπιμα - γίνει σύνδεση κάποιου κρίσιμου φακέλου του συστήματος, όπως είναι ο **/** ή ο **/etc**, ο επιτιθέμενος μπορεί να γίνει διαχειριστής του συστήματος, να αλλάξει κωδικούς κ.ο.κ. Για την αντιμετώπιση αυτής της απειλής, απαιτείται η εφαρμογή κανόνων του λειτουργικού συστήματος, που απαγορεύουν τη σύνδεση κρίσιμων φακέλων με ένα δοχείο, όπως είναι οι τεχνολογίες AppArmor/SELinux.

- **Σύνδεση του δοχείου με το Socket του Docker**

Σε συστήματα που εκτελούνται δοχεία μέσω του εργαλείου Docker, αξίζει να υπογραμμίσουμε ότι οποιαδήποτε εντολή καταχωρείται στο τερματικό καταλήγει στον Docker Daemon (Δαίμονας του Docker), ο οποίος υποχρεωτικά εκτελείται με δικαίωμα διαχειριστή [85]. Πρόκειται για μια διεργασία που εκτελείται στο παρασκήνιο και διαχειρίζεται δοχεία, καθώς και την επικοινωνία ανάμεσα στα δοχεία και τον εξυπηρετητή. Ο μηχανισμός περιλαμβάνει ένα ψευδο-αρχείο που βρίσκεται στο μονοπάτι `/var/run/docker.sock`, μέσα από το οποίο μια διεργασία μπορεί να στέλνει πακέτα στον Δαίμονα αυτό. Γίνεται εύκολα φανερό, ότι εάν μια εντολή όπως η παρακάτω



```
docker run -v /var/run/docker.sock:/var/run/docker.sock -p 8080:8080 -ti jenkins
```

Εικόνα 19 - Σύνδεση του Docker Daemon socket με δοχείο

αποδώσει δικαιώματα σε ένα δοχείο να τροποποιεί το ψευδο-αρχείο αυτό (το ονομαζόμενο socket), αυτό αποκτά πλήρη δικαιώματα πάνω στον Δαίμονα του Docker ώστε να εκτελεί όποιο δοχείο επιθυμεί. Υπάρχουν περιπτώσεις (δοχεία που εσωτερικά εκτελούν τη λειτουργία Docker, ονομαζόμενα Docker-in-Docker), όπως σε συστήματα CI/CD, όπου αυτό πρακτικά επιβάλλεται από την τεχνολογία που χρησιμοποιείται, όμως ακόμα κι έτσι δεν παύει να αποτελεί δυνητική απειλή. Σε κάθε περίπτωση, προτείνεται η εύρεση εναλλακτικών λύσεων / εργαλείων προκειμένου να αποφεύγεται η ανωτέρω σύνδεση.

- **Κοινοί χώροι ονομάτων (namespaces) μεταξύ δοχείου και του εξυπηρετητή**

Οργανικό τμήμα της στεγανοποίησης που προσφέρει ο χώρος των δοχείων και οι συναφείς τεχνολογίες (Docker, Podman κτλ.) είναι και η χρήση των χώρων ονομάτων (namespaces) σε λειτουργικά συστήματα Linux [62]. Από προεπιλογή, όταν εκτελείται ένα δοχείο, ενώ η διεργασία του, όπως αναφέραμε, ανήκει στον χρήστη root (ριζικό χρήστη), η μηχανή φιλοξενίας παράγει ένα σύνολο προσβάσεων που είναι ξεχωριστό για το κάθε δοχείο. Αυτές οι προσβάσεις (namespaces) καθορίζονται σε επίπεδα διεργασιών, τομέα (domain), δικτύου, αρχείων κ.ο.κ. Εάν κατά την εκτέλεση ενός δοχείου δοθεί η σημαία `--pid=host`, τότε ο χώρος ονομάτων αναφορικά με τις διεργασίες εξομοιώνεται με εκείνον του host, δηλαδή του εξυπηρετητή. Υπάρχουν περιπτώσεις όπου κάτι τέτοιο είναι επιθυμητό (δοχεία που επιτηρούν τη χρήση άλλων δοχείων). Όμως κάθε τέτοια περίπτωση πρέπει να προσεγγίζεται με προσοχή, καθώς πλέον το δοχείο έχει πρόσβαση στις

διεργασίες που εκτελούνται όχι μόνο σε επίπεδο εξυπηρετητή, αλλά επιπρόσθετα και σε κάθε άλλο δοχείο που φιλοξενείται σε αυτόν.

Παράγοντες απειλών: Εξωτερικοί, Εσωτερικοί, Ακούσιοι εσωτερικοί παράγοντες

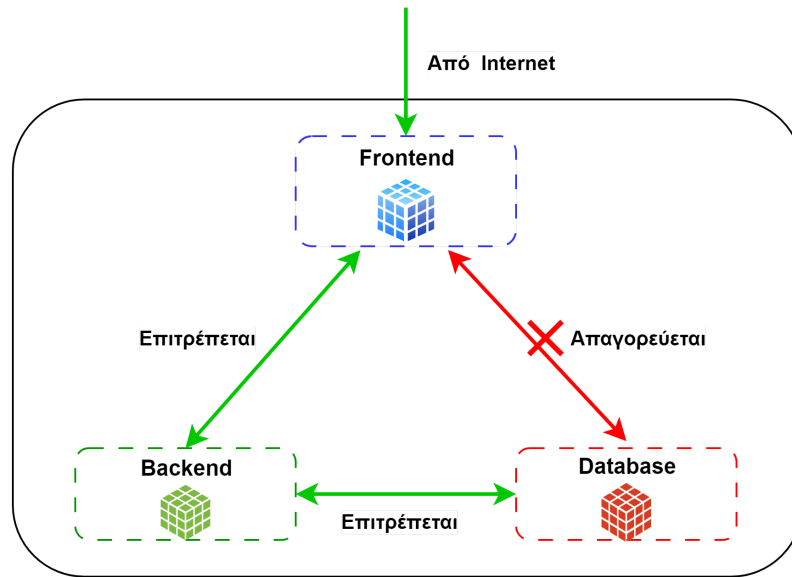
4.2.5 *Εκτεθειμένα μυστικά/συνθηματικά*

Ένα δοχείο που προορίζεται να προσφέρει υπηρεσίες σε πραγματικούς καταναλωτές/πελάτες συνήθως απαιτεί διασύνδεση με μια βάση δεδομένων ή ένα τρίτο σύστημα, για την οποία / το οποίο χρησιμοποιούνται κωδικοί ή συνθηματικά. Είναι μια συνηθισμένη κακή πρακτική αυτά τα μυστικά είτε να αποθηκεύονται αυτούσια μέσα στον κώδικα, είτε να βρίσκονται στο μηχανήμα (εικονικό ή φυσικό) που θα φιλοξενήσει τελικά το δοχείο. Αυτές είναι πρακτικές που πρέπει να αποφεύγονται καθώς υπάρχει κίνδυνος διαρροής. Σε κάθε περίπτωση, τα μυστικά πρέπει να φυλάσσονται σε ειδική εφαρμογή, η οποία τα κρατά κρυπτογραφημένα και τα ενσωματώνει στα δοχεία μόνο κατά την εγκατάσταση αυτών.

Παράγοντες απειλών: Εξωτερικοί, Εσωτερικοί, Ακούσιοι εσωτερικοί παράγοντες

4.2.6 *Μη ασφαλής δικτύωση*

Είναι γεγονός ότι η εφαρμογές που δομούνται σήμερα σε τεχνολογία δοχείων βασίζονται ολοένα και περισσότερο στην αρχιτεκτονική των μικρο-υπηρεσιών (microservices). Έχοντας λοιπόν τη λειτουργικότητα μιας εφαρμογής κατακερματισμένη ανάμεσα σε πολλαπλά δοχεία, γίνεται προφανές ότι δεν απαιτείται όλα τα δοχεία να είναι προσβάσιμα από τον έξω κόσμος, καθώς επίσης και ότι δεν είναι απαραίτητο ένα δοχείο να είναι καν προσβάσιμο από όλα τα υπόλοιπα δοχεία. Σε αντίθετη περίπτωση (μη ασφαλούς & κατάλληλης δικτύωσης), αυξάνεται η διαθέσιμη επιφάνεια επίθεσης από τα εξωτερικά δίκτυα προς το εσωτερικό της αρχιτεκτονικής. Συνεπώς, υπάρχει ανάγκη για κανόνες δικτύωσης, οι οποίοι συνήθως ρυθμίζονται από τον ενορχηστρωτή (πχ. Docker Compose, Kubernetes) των δοχείων (μέσω πχ. μοντέλων που παρέχονται από τον χρήστη/μηχανικό), ενώ μπορούν ακόμη και να αξιοποιηθούν Virtual Private Clouds (Εικονικά Ιδιωτικά Νέφη) / Web Application Firewalls (Αναχώματα Εφαρμογών Ιστού) [63] για μεγαλύτερο επίπεδο ασφάλειας. Μια ενδεικτική αρχιτεκτονική για εφαρμογές ιστού παρατίθεται στην Εικ. 20.



Εικόνα 20 - Δικτυακοί κανόνες επικοινωνίας δοχείων

Παράγοντες απειλών: Εξωτερικοί, Εσωτερικοί, Ακούσιοι εσωτερικοί παράγοντες

4.3 Καταγραφίσεις επιθέσεις

Σε αυτήν την ενότητα θα παρατεθούν οι σημαντικότερες και πιο επιβλαβείς επιθέσεις που έχουν καταγραφεί σε συστήματα CI/CD, προκειμένου να τονισθεί κατάλληλα ο βαθμός επικινδυνότητας που διατρέχει ένα τέτοιο σύστημα, εάν δεν τηρηθούν σωστά τα μέτρα που απαιτούνται για την εξουδετέρωση όλων των δυνητικών απειλών, όπως αυτών που αναφέρθηκαν προηγουμένως.

4.3.1 Στόχος: Πηγαίος κώδικας της PHP

Απειλή: Μη εξουσιοδοτημένη πρόσβαση στο σύστημα

Τον Απρίλιο του 2021 [64], άγνωστοι κατόρθωσαν να ενσωματώσουν και να εγκαταστήσουν κώδικα στο κεντρικό αποθετήριο της πολύ διαδεδομένης γλώσσας PHP, η οποία σύμφωνα με εκτιμήσεις [65], χρησιμοποιείται από το 80% των ιστοσελίδων παγκοσμίως. Οι επιτιθέμενοι ανακάλυψαν ότι ήταν επιτρεπτή η αυθεντικοποίηση μέσω HTTPS (ουσιαστικά υποστηρίζεται μόνο server-side αυθεντικοποίηση σε αυτή την περίπτωση), ενώ οι διαχειριστές είχαν την πεποίθηση ότι ήταν επιτρεπτή η αυθεντικοποίηση μόνο μέσω SSH (όπου τόσο ο πελάτης όσο και ο εξυπηρετητής αυθεντικοποιούνται και άρα αυτό αποτελεί μια πιο ασφαλή μέθοδο αυθεντικοποίησης). Το

αποτέλεσμα ήταν ότι οι επιτιθέμενοι κατόρθωσαν να εισάγουν δικό τους πηγαίο κώδικα, τον οποίο και πρόλαβαν να κατεβάσουν χιλιάδες χρήστες ανά τον κόσμο.

4.3.2 Στόχος: Mercedes - Benz

Απειλή: Μη εξουσιοδοτημένη πρόσβαση στο σύστημα & Αποθήκευση συνθηματικών σε μορφή κειμένου

Το Μάιο του 2020 [67], ο Ελβετός μηχανικός Till Kottmann ανακάλυψε ένα σύστημα VCS της εταιρείας Mercedes - Benz προσβάσιμο από το διαδίκτυο (Internet). Το σύστημα χρησιμοποιούσε τη γνωστή πλατφόρμα Gitlab, η οποία λόγω λανθασμένης παραμετροποίησης επέτρεπε την αυτο-εγγραφή ενός χρήστη, χωρίς να επικυρώνει εάν η δηλωμένη email διεύθυνση ήταν πραγματική και άνηκε στον τομέα (domain) της εταιρείας. Η συνέπεια αυτού του ευρήματος ήταν ότι οποιοσδήποτε χρήστης του διαδικτύου ήταν σε θέση να κατεβάσει 580 αποθετήρια πηγαίου κώδικα, στον οποίο μάλιστα περιέχονταν και μυστικά/κωδικοί για εσωτερικά συστήματα της γνωστής αυτοκινητοβιομηχανίας.

4.3.3 Στόχος: Homebrew package manager για MACOS

Απειλή: Εκτέλεση αγωγού από απλό χρήστη

Τον Απρίλιο του 2021 [66], ένας μηχανικός ασφαλείας με το ψευδώνυμο Ryottak, εξέθεσε ένα κενό ασφαλείας της πλατφόρμας GitHub Actions, την οποία αξιοποιεί η πλατφόρμα διαχείρισης πακέτων για υπολογιστές Mac, Homebrew. Ο μηχανικός κατόρθωσε να ενσωματώσει τον κώδικα του στο κύριο κλαδί του αποθετηρίου του Homebrew, παρακάμπτοντας τον υφιστάμενο μηχανισμό έγκρισης από χρήστη με αυξημένα δικαιώματα. Ευτυχώς, ο επιτιθέμενος ήταν καλόβουλος, και η επίθεση έγινε με τη συναίνεση της πλατφόρμας στο πλαίσιο ελέγχου Red Team Penetration Testing (Δοκιμή Διείσδυσης), ως Proof of Concept.

4.3.4 Στόχος: Η βιβλιοθήκη rc του αποθετηρίου nrm

Απειλή: Χρήση κακόβουλων κώδικα από τρίτες βιβλιοθήκες & Μη εξουσιοδοτημένη πρόσβαση στο σύστημα

Το Νοέμβριο του 2021 [68] επιτιθέμενοι κατάφεραν να εισάγουν κακόβουλο κώδικα υποκλοπής κωδικών στη πολύ δημοφιλή βιβλιοθήκη rc του αποθετηρίου nrm, παραβιάζοντας τον αδύναμο κωδικό ενός μηχανικού της ομάδας rc. Σύμφωνα με τα επίσημα στατιστικά, την βιβλιοθήκη καλούν 14 εκατομμύρια χρήστες ανα εβδομάδα.

4.3.5 Στόχος: Samsung

Απειλή: Αποθήκευση συνθηματικών σε μορφή κειμένου & Μη εξουσιοδοτημένη πρόσβαση στο σύστημα/αποθετήριο

Σύμφωνα με τον Whittaker [69], το 2019 ένα εργαστήριο της Samsung ρύθμισε λανθασμένα ένα αποθετήριο κώδικα git, το οποίο και κατέστη κοινώς προσβάσιμο από το Διαδίκτυο. Αποτέλεσμα αυτής της ακούσιας ενέργειας ήταν η φανέρωση πολλαπλών μυστικών κωδικών, οι οποίοι από ελλιπή προσοχή των μηχανικών ήταν αποθηκευμένοι σε μορφή απλού κειμένου.

4.3.6 Στόχος: Nissan

Απειλή: Μη εξουσιοδοτημένη πρόσβαση στο σύστημα

Τον Ιανουάριο του 2021 [70], διέρρευσε σε κοινότητες στο Διαδίκτυο ο πηγαίος κώδικας των εφαρμογών κινητών και οχημάτων της Nissan. Η αιτία ήταν χρήση προεπιλεγμένων συνθηματικών για το λογαριασμό του διαχειριστή σε έναν εξυπηρετητή Bitbucket που ήταν προσβάσιμος από το Διαδίκτυο.

4.3.7 Στόχος: CodeCov, Atlassian, Kubernetes, Python

Απειλή: Επίθεση εφοδιαστικής αλυσίδας (Supply Chain Attack) & Αποθήκευση συνθηματικών σε μορφή κειμένου & Φόρτωση κακόβουλων εκτελέσιμων/βιβλιοθηκών στο αποθετήριο

Μια από τις πλέον γνωστές επιθέσεις σε διαδικασίες CI/CD περιγράφεται στο άρθρο [72]. Σε αυτό, ο Even-Zohar εξηγεί πως με ένα συνδυασμό ακρυπτογράφων μυστικών και αντικατάστασης τεχνητάς βιβλιοθήκης που ενσωματώνεται στο κύριο

λογισμικό, οι επιτιθέμενοι παράλλαξαν τη διασημότετη βιβλιοθήκη ελέγχου κώδικα για δοκιμές CodeCon, προκειμένου να συλλέξουν συνθηματικά από λογισμικά που τη χρησιμοποιούν. Αυτό το επέτυχαν εξαιτίας κωδικού που είχε ενσωματωθεί σε Dockerfile, με τον οποίο απέκτησαν πρόσβαση στον αποθηκευτικό χώρο του οργανισμού στο νέφος της Google. Οι πιθανοί λήπτες αυτής της βιβλιοθήκης υπολογίζονται στους 29.000.

4.3.8 Στόχος: SolarWinds, State Department, NSA, Intel, Microsoft, Cisco

Απειλή: Επίθεση εφοδιαστικής αλυσίδας (Supply Chain Attack) & Φόρτωση κακόβουλων εκτελέσιμων/βιβλιοθηκών στο αποθετήριο

Ίσως η πιο σημαντική επίθεση που καταγράφηκε ποτέ [71], συνέβη το Δεκέμβριο του 2020. Ξεκίνησε από διάφορες εταιρείες, φαινομενικά ασύνδετες μεταξύ τους, στις οποίες άρχισαν να καταγράφονται συμβάντα ασφαλείας. Τα κρούσματα πολλαπλασιάζονταν, ώσπου τελικά η εταιρεία κυβερνοασφάλειας FireEye κατόρθωσε να εντοπίσει την πηγή των περιστατικών: το λογισμικό Orion της Solarwinds. Η Solarwinds είναι μια εταιρεία παγκόσμιου βεληνεκού, της οποίας το λογισμικό Orion διαχειρίζεται τις υποδομές οργανισμών. Οι επιτιθέμενοι κατόρθωσαν να πλήξουν την εφοδιαστική αλυσίδα κατά την παραγωγή των εκτελέσιμων αρχείων του λογισμικού και να αντικαταστήσουν μια βιβλιοθήκη-τέχνημα με μια δικής τους κατασκευής. Φημολογείται ότι αυτό επετεύχθη λόγω διαρροής κωδικών του περιβάλλοντος παραγωγής τεχνημάτων. Το αποτέλεσμα ήταν ότι 18.000 οργανισμοί και εταιρείες, μεταξύ των οποίων κυβερνήσεις και τεχνολογικοί κολοσσοί, έπεσαν θύματα αυτής της επίθεσης.

5

Ανάπτυξη Συστήματος

Σε αυτό το κεφάλαιο θα παρουσιαστεί η τεχνική λύση της παρούσας διπλωματικής εργασίας, και συγκεκριμένα ένα ασφαλές σύστημα που υλοποιεί την προσέγγιση DevSecOps. Αρχικά, θα γίνει παράθεση των λειτουργικών και μη λειτουργικών απαιτήσεων του υπό υλοποίηση συστήματος. Στη συνέχεια, θα αναφερθούν οι τεχνολογίες και βιβλιοθήκες που χρησιμοποιήθηκαν, και τελικά θα γίνει αναλυτική παρουσίαση τόσο της αρχιτεκτονικής του συστήματος όσο και της υλοποίησης αυτού

5.1 Απαιτήσεις του Συστήματος

Οι απαιτήσεις που έχουν τεθεί από το σύστημα διαχωρίζονται σε μη λειτουργικές και λειτουργικές. Ας δούμε με τη συγκεκριμένη σειρά τι στόχους περιμένουμε να καλύψει η προτεινόμενη λύση.

5.1.1 Μη λειτουργικές απαιτήσεις

Για τις μη λειτουργικές απαιτήσεις δεν εντοπίστηκε κάτι συγγενές σε κάποια δημοσίευση, συνεπώς λήφθηκαν υπόψη οι γενικές οδηγίες που περιγράφονται στα [73] και [74]. Η λίστα περιλαμβάνει τα εξής στοιχεία:

- **Κλιμακωσιμότητα**

Το σύστημα θα πρέπει να ανταποκρίνεται πάντοτε με την ίδια ταχύτητα, ανεξάρτητα από το φόρτο. Αυτό πρέπει να επιτυγχάνεται μέσω οριζόντιας ή κάθετης κλιμάκωσης, ανάλογα με την παραμετροποίηση.

- **Υψηλή διαθεσιμότητα**

Όλα τα τμήματα του συστήματος πρέπει να διατίθενται σε υψηλή διαθεσιμότητα.

- **Απόδοση**

Το σύστημα θα πρέπει να εκτελεί τους ελέγχους, την οικοδόμηση των τεχνημάτων αλλά και την εγκατάστασή τους με τέτοιο τρόπο ώστε να ελαχιστοποιείται η αναμονή.

- **Αξιοπιστία**

Το σύστημα θα πρέπει να είναι ανθεκτικό απέναντι σε σφάλματα, ώστε να αποφεύγονται παύσεις στη ροή των εργασιών.

- **Ασφάλεια**

Η πρόσβαση σε όλα τα στοιχεία του συστήματος πρέπει να είναι πλήρως ελεγχόμενη και βασισμένη σε ρόλους, με δυνατότητα χρήσης MFA, και κρυπτογραφικών κλειδιών για πρόσβαση σε εικονικές μηχανές.

- **Συντήρηση**

Το σύστημα πρέπει να είναι εύκολο στη συντήρηση, με εύκολα προσβάσιμες οδηγίες και ευκολία στην αναβάθμιση εκδόσεων.

- **Παρακολούθηση και Καταγραφή**

Το σύστημα θα πρέπει να παρέχει επαρκή μέσα παρακολούθησης της ευρωστίας του αγωγού καθώς και αρχεία καταγραφής της πορείας των διεργασιών.

- **Διαλειτουργικότητα**

Το σύστημα θα πρέπει να υποστηρίζει την απρόσκοπτη διαλειτουργικότητα και ενσωμάτωση γνωστών βιβλιοθηκών, εργαλείων CI/CD και ενορχηστρωτών, όπως Kubernetes, Rancher, K3s, Openshift, AWS Fargate κ.α.

- **Αξιοποίηση πόρων**

Το σύστημα θα πρέπει να μεγιστοποιεί την αξιοποίηση των διαθέσιμων πόρων, ανάλογα με τις φάσεις του αγωγού που εκτελούνται.

- **Εμπειρία Χρήστη / Χρηστικότητα**

Το σύστημα θα πρέπει να είναι φιλικό προς το χρήστη, με εύχρηστο μενού και ροές που αυξάνουν την παραγωγικότητα.

- **Διαχείριση κώδικα**

Το σύστημα θα πρέπει να υποστηρίζει αποθετήρια κώδικα, βασισμένα στο τεχνικό πρότυπο git. Το αποθετήριο θα μπορεί να κρατά πολλαπλές εκδόσεις του ίδιου

κώδικα και να επιτρέπει στους προγραμματιστές την εύκολη διαχείριση αυτού μέσα από γραφικό περιβάλλον.

- **Γλώσσες**

Το σύστημα θα πρέπει να υποστηρίζει την δόμηση και αποθήκευση τεχνημάτων σε ιεραρχημένο ευρετήριο σε όλες τις διαδεδομένες προγραμματιστικές γλώσσες.

- **Δικτύωση**

Το σύστημα θα πρέπει να είναι διαμορφωμένο σύμφωνα με την αρχή της ελάχιστης πρόσβασης, με τους δικτυακούς κανόνες να επιτρέπουν την κίνηση αυστηρά και μόνο όταν αυτή απαιτείται.

- **Απομόνωση - Σκλήρυνση δοχείων**

Τα δοχεία του συστήματος θα πρέπει να είναι επαρκώς στεγανοποιημένα μεταξύ τους και ως προς το εικονικό μηχάνημα που τα φιλοξενεί. Επιπλέον, το σύστημα θα εφαρμόζει πολιτικές αυστηρής χρήσης των πόρων που απαιτούνται από τα δοχεία μέσω δημιουργίας πολιτικών (AppArmor, SELinux).

5.1.2 Λειτουργικές απαιτήσεις

Κώδικας - Συνεργασία

- **Συνεργασία**

Το σύστημα θα πρέπει να προσφέρει εξελιγμένες λειτουργίες συλλογικής διαχείρισης κώδικα, όπως έγκριση για αλλαγές και ενσωμάτωση κώδικα στο κύριο κλαδί.

- **Παρακολούθηση ζητημάτων**

Το σύστημα θα πρέπει να προσφέρει τη δυνατότητα δημιουργίας, διαχείρισης και προτεραιοποίησης ζητημάτων, όπως προβλήματα και προσθήκες στη λειτουργικότητα. Η δημιουργία, ανάθεση και διαχείριση των ζητημάτων πρέπει να προσφέρεται εύχρηστα μέσα από γραφικό περιβάλλον.

- **Χρήσιμες πληροφορίες**

Το σύστημα θα πρέπει να προσφέρει λειτουργία για αποθήκευση αρχείων και χρήσιμων πληροφοριών για την εκάστοτε εφαρμογή.

- **Αποθήκευση εκτελέσιμων/εικόνων δοχείων**

Το σύστημα θα πρέπει να υποστηρίζει την αποθήκευση των παραγόμενων

εκτελέσιμων αρχείων ή εικόνων δοχείων που παράγονται από τη διαδικασία, σε εσωτερικά ή εξωτερικά αποθετήρια.

Αγωγός CI/CD

- **Συνεχής ενσωμάτωση**

Το σύστημα θα πρέπει να υποστηρίζει εγγενώς ή μέσω προσθέτων την εκτέλεση αγωγών CI/CD, με πλήρη ελευθερία διαμόρφωσης των σταδίων και βημάτων που θα εκτελούνται, καλύπτοντας όλο το εύρος του κύκλου ζωής λογισμικού.

- **Συνεχής εγκατάσταση**

Το σύστημα θα πρέπει να υποστηρίζει την εγκατάσταση του παραγόμενου λογισμικού σε πολλαπλά ήδη διαμορφωμένα περιβάλλοντα και με πολλαπλούς/ετερογενείς ενορχηστρωτές.

- **Στάδια αγωγών**

Το σύστημα θα πρέπει να διαθέτει αγωγούς με τουλάχιστον τα εξής στάδια: ελέγχων, σχηματισμού τεχνημάτων και εγκατάστασης.

- **Αποδεκτά όρια**

Το σύστημα θα πρέπει να είναι παραμετροποιήσιμο ως προς τα αποδεκτά όρια επικινδυνότητας των ευρημάτων, πάνω από τα οποία ο αγωγός θα τερματίζεται ανεπιτυχώς.

- **Στάδιο δοκιμών**

Το σύστημα θα πρέπει να υποστηρίζει διαφορετικά είδη δοκιμών του πηγαίου κώδικα, όπως δοκιμές μονάδων (unit testing) και δοκιμές ενοποίησης (integration testing) κάνοντας χρήση εργαλείων όπως π.χ Maven και εκ των προτέρων διαμορφωθείσας υποδομής.

- **Έλεγχος μυστικών**

Το σύστημα θα πρέπει να ελέγχει τον πηγαίο κώδικα και τις παραγόμενες εικόνες για μυστικά, και να ενημερώνει τον χρήστη σε περίπτωση ευρήματος.

- **Έλεγχος κώδικα**

Το σύστημα θα πρέπει να ελέγχει τον πηγαίο κώδικα ως προς την ποιότητά του, αλλά και τυχόν ευπάθειες σε αυτόν, καθώς και να ενημερώνει τον χρήστη σε περίπτωση ευρήματος.

- **Έλεγχος εικόνων**

Το σύστημα θα πρέπει αυτοματοποιημένα να ελέγχει τις παραγόμενες εικόνες για τυχόν ευπάθειες, και να ενημερώνει τον χρήστη σε περίπτωση επικίνδυνου ευρήματος.

- **Έλεγχος εξαρτήσεων**

Το σύστημα θα πρέπει να ελέγχει τις εξαρτήσεις του κώδικα αλλά και των παραγόμενων εικόνων για τυχόν ευπάθειες, ειδοποιώντας τον αρμόδιο χρήστη σε περίπτωση επικίνδυνου ευρήματος.

- **Παρακολούθηση δοχείων**

Το σύστημα θα πρέπει να διαθέτει μέσα παρακολούθησης της ορθής λειτουργίας των δοχείων.

- **Προστασία από DoS**

Το σύστημα θα πρέπει να είναι σε θέση να αμύνεται επαρκώς απέναντι σε επιθέσεις τύπου Denial of Service, κάνοντας χρήση εργαλείων όπως αναχώματα ασφαλείας, εργαλεία ελέγχου συμπεριφοράς κ.α.

- **Ειδοποιήσεις**

Σε περίπτωση σφάλματος ή επικίνδυνου ευρήματος, το σύστημα θα πρέπει να ειδοποιεί τους αρμόδιους χειριστές μέσω email και αναφορών που παράγονται από το σύστημα. Η διαχείριση των αγωγών και η ανάγνωση των αναφορών παρέχεται μέσα από τις εγγενείς δυνατότητες της πλατφόρμας Gitlab, το οποίο επεκτείνεται και παραμετροποιείται στο πλαίσιο της Διπλωματικής εργασίας.

Αυθεντικοποίηση - Μυστικότητα

- **Ασφάλεια - Αυθεντικοποίηση**

Το σύστημα θα πρέπει να επιτρέπει την είσοδο στο περιβάλλον χρήστη με χρήση Αυθεντικοποίησης πολλαπλών βημάτων (MFA), ενώ την υποβολή κώδικα μόνο μέσω εξουσιοδοτημένων κλειδιών SSH.

- **Μυστικά**

Το σύστημα θα πρέπει να είναι σε θέση να αποθηκεύει και να διαχειρίζεται με ασφαλή τρόπο μυστικά ή συνθηματικά κρυπτογραφημένα, χωρίς αυτά να φανερώνονται σε οποιοδήποτε αρχείο καταγραφής.

5.2 Σχεδιασμός του Συστήματος

Σε αυτή την ενότητα θα παρουσιαστεί μια υψηλού επιπέδου περιγραφή του προτεινόμενου συστήματος της Διπλωματικής εργασίας.

5.2.1 Σχεδιασμός Λογισμικού

5.2.1.1 Σχεδιασμός Επέκτασης Πλατφόρμας Gitlab

Με δεδομένο ότι θα χρησιμοποιηθεί η πλατφόρμα Gitlab, στο πλαίσιο της εργασίας σχεδιάστηκε μια αρχιτεκτονική, στην οποία τόσο το κέντρικό σύστημα όσο και τα προγράμματα, τα οποία απαιτούνται για την εκτέλεση του αγωγού CI/CD, θα εκτελούνται σε μορφή δοχείου σε εικονικές μηχανές. Στη συνέχεια, θα ληφθούν όλα τα απαραίτητα μέτρα και οι σχετικές ρυθμίσεις, ώστε να διασφαλιστεί ότι η εν γένει λειτουργία της πλατφόρμας αλλά και η διαχείριση των αποθετηρίων και αγωγών θα γίνει με τον πλέον ασφαλή τρόπο. Τέλος, η λειτουργία CI/CD θα επεκταθεί με το σχεδιασμό και δημιουργία του αγωγού που περιγράφεται στην επόμενη ενότητα.

5.2.1.2 Σχεδιασμός Αγωγού CI/CD

Για να σχεδιαστεί ο αγωγός που θα είναι σε θέση να παράγει και να εγκαθιστά ασφαλή δοχεία λογισμικού, σε αυτόν συμπεριλήφθηκαν τα εξής στάδια:

- **Στάδιο δοκιμής κώδικα**

Εδώ θα πραγματοποιούνται οι απαραίτητες δοκιμές για τη διασφάλιση της ποιότητας του λογισμικού, ώστε να επιβεβαιωθεί ότι το σύνολο της λειτουργικότητας της εκάστοτε εφαρμογής προσφέρεται ορθά, καθώς και ότι καλύπτεται επαρκώς από αυτοματοποιημένες δοκιμές.

- **Στάδιο ελέγχου κώδικα**

Σε αυτό το στάδιο ελέγχεται ο κώδικας και οι ενσωματωμένες βιβλιοθήκες της υπό ανάπτυξη εφαρμογής από ένα εύρος διαθέσιμων εργαλείων, προκειμένου να επαληθευτεί ότι δεν περιέχουν γνωστές ευπάθειες ή προβλήματα σημεία.

- **Στάδιο οικοδόμησης εικόνας δοχείου**

Σε αυτό το στάδιο αφού ελεγχθεί η ασφάλεια του Dockerfile, οικοδομείται η εικόνα δοχείου και αποθηκεύεται προσωρινά στον αγωγό.

- **Στάδιο ελέγχου εικόνας**

Σε αυτό το στάδιο γίνονται όλοι οι απαραίτητοι έλεγχοι στο προσωρινό αρχείο εικόνας ώστε να επαληθευτεί ότι δεν περιέχει ευπάθειες ή κακόβουλο λογισμικό.

- **Στάδιο προώθησης εικόνας**

Σε αυτό το στάδιο η ασφαλής εικόνα αποθηκεύεται στο αποθετήριο εικόνων του συστήματος.

- **Στάδιο ελέγχου προωθημένης εικόνας**

Σε αυτό το στάδιο γίνονται έλεγχοι στην αποθηκευμένη εικόνα ώστε να συγκριθεί με συγκεκριμένα πρότυπα ασφαλείας τύπου SCAP, καθώς και με τις ρυθμίσεις ασφαλείας που έχουν επιλεγεί από τον διαχειριστή, για ασφαλή εγκατάσταση δοχείων.

- **Στάδιο υπογραφής εικόνας**

Σε αυτό το στάδιο η αποθηκευμένη εικόνα υπογράφεται, ώστε να μπορεί στη συνέχεια να επαληθευτεί η προέλευση της από τον ενορχηστρωτή.

- **Στάδιο εγκατάστασης δοχείων**

Σε αυτό το στάδιο πρώτα γίνεται επαλήθευση των ασφαλών ρυθμίσεων εγκατάστασης, και τελικά η ασφαλής πλέον εικόνα εγκαθίσταται στη συστοιχία του ενορχηστρωτή σε ήδη διαμορφωμένο περιβάλλον.

- **Χειροκίνητο Στάδιο δόμησης προφίλ SECCOMP**

Σε αυτό το χειροκίνητο στάδιο παράγεται ένα προφίλ ασφαλείας δοχείων τύπου SECCOMP, το οποίο περιορίζει τις κλήσεις που μπορεί να εκτελέσει ένα δοχείο προς το κέλυφος του λειτουργικού συστήματος.

5.2.2 Σχεδιασμός Υλικού

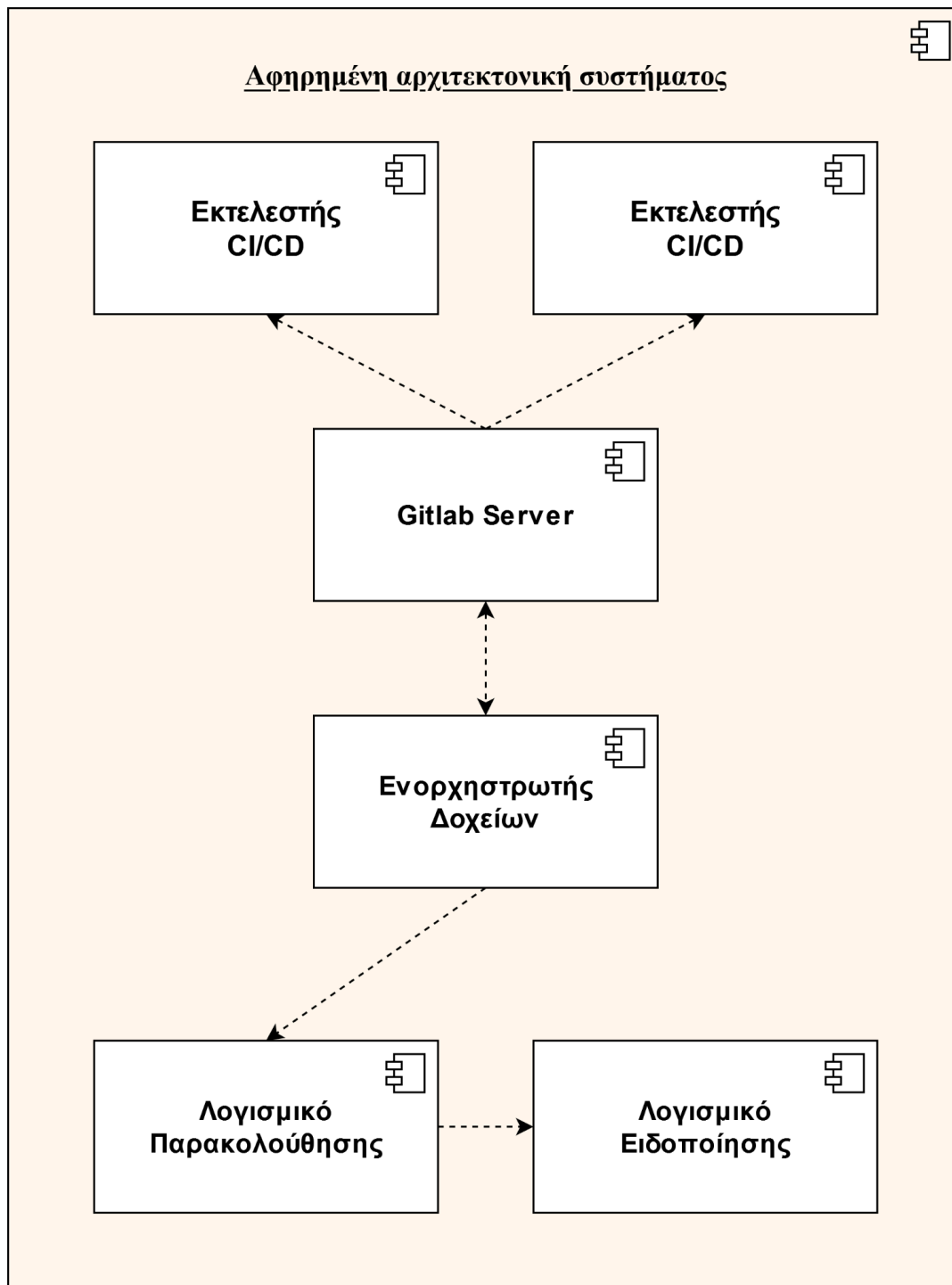
Με βάση το υφιστάμενο υλικό, αξιοποιήθηκαν τεχνολογίες υπερεπόπτη για τη δημιουργία και διαχείριση εικονικών μηχανών, στις οποίες εγκαταστάθηκε η προτεινόμενη αρχιτεκτονική. Στην κάθε εικονική μηχανή ελήφθησαν όλα τα πρόσφορα μέσα για την σκλήρυνση τόσο του λειτουργικού συστήματος, όσο και των διασυνδέσεων σε επίπεδο δικτύου. Το απαραίτητο λογισμικό εγκαταστάθηκε με τη μορφή δοχείων, όπου αυτο ήταν εφικτό, στην εκάστοτε εικονική μηχανή.

5.2.3 Αφηρημένη Αρχιτεκτονική Συστήματος

Σε αφηρημένο επίπεδο, υπάρχουν 4 κύρια τμήματα της αρχιτεκτονικής που θα πρέπει να διαλειτουργήσουν:

- Η πλατφόρμα Gitlab που αποθηκεύει τον πηγαίο κώδικα και τον κώδικα του αγωγού CI/CD.
- Τα απαραίτητα προγράμματα που συνδέονται με το Gitlab, που αναλαμβάνουν την εκτέλεση του αγωγού.
- Ο ενορχηστρωτής δοχείων που αναλαμβάνει την ασφαλή εκτέλεση των δοχείων που παρέχουν την υπηρεσία λογισμικού.
- Τα λογισμικά παρακολούθησης και ειδοποίησης που επιβλέπουν την ασφαλή λειτουργία των εικονικών μηχανών και δοχείων.

Το κάθε τμήμα θα συνδεθεί και θα διαλειτουργήσει με ασφάλεια με τα υπόλοιπα, προκειμένου το κύκλωμα να παρέχει μια ολοκληρωμένη και ασφαλή λειτουργία, όπως φαίνεται στην Εικόνα 21.



Εικόνα 21 - Αφηρημένη αρχιτεκτονική συστήματος

5.3 Υλοποίηση

Σε αυτήν την ενότητα θα δοθούν διάφορες λεπτομέρειες υλοποίησης, από την επιλογή συγκεκριμένων εργαλείων, πλαισίων και βιβλιοθηκών μέχρι την ανάλυση της δομής του κώδικα

5.3.1 Υλοποίηση Λογισμικού

Για τις ανάγκες διαμόρφωσης ενός ασφαλούς αγωγού, ο οποίος έχει ως στόχο να παράγει και να εγκαθιστά ασφαλή δοχεία δημιουργήθηκε μια υποτυπώδης εφαρμογή χρησιμοποιώντας τη γλώσσα προγραμματισμού Java στην έκδοση 17 σε συνδυασμό με το πλαίσιο εφαρμογών ιστού (web framework) Spring Boot⁸ έκδοση 3 και Maven έκδοση 3.8.4. Με βάση τον κώδικα αυτής της εφαρμογής αναπτύχθηκε και ο αγωγός, ο οποίος μπορεί να επαναχρησιμοποιηθεί για άλλες εφαρμογές βασισμένες στο Maven. Ο αγωγός, ο οποίος θα μπορούσε να αποτελέσει πρότυπο για τον έλεγχο όλων των εφαρμογών Java - Maven, περιλαμβάνει τα εξής στάδια:

5.3.1.1 Στάδιο: Δοκιμή κώδικα (Test Code)

Σε αυτό το στάδιο πραγματοποιούνται οι απαραίτητες δοκιμές για τη διασφάλιση της ποιότητας του λογισμικού, ώστε να επιβεβαιωθεί ότι το σύνολο της λειτουργικότητας της εκάστοτε εφαρμογής δεν έχει επηρεαστεί αρνητικά από τον νέο κώδικα που έχει υποβληθεί, καθώς και ότι ένα επαρκές τμήμα του κώδικα καλύπτεται από αυτοματοποιημένες δοκιμές. Αυτό το στάδιο κάνει χρήση ενός προσωρινού δοχείου με τη γλώσσα Java 17 και την πλατφόρμα Maven, ενώ περιλαμβάνει 2 ενέργειες (jobs):

➤ Ενέργεια: Δοκιμές μονάδας (Unit Tests)

Οι δοκιμές μονάδας (Unit Tests) είναι μικρές δοκιμές που κάθε μια ελέγχει τον κώδικα μιας μονάδας ή ενός μικρού τμήματος του προγράμματος. Σκοπός τους είναι να επιβεβαιώσουν ότι οι μονάδες του κώδικα λειτουργούν σωστά και εκτελούν τις αναμενόμενες λειτουργίες τους. Εάν έστω και μια δοκιμή μονάδας αποτύχει, η εκτέλεση του αγωγού σταματά.

Ο κώδικας για την υλοποίηση της ενέργειας είναι ο παρακάτω:

```
❖ #Unit tests
❖ Unit-Tests:
❖   stage: test-code
❖   image: ${MAVEN_BUILD_IMAGE}
❖   script:
❖   - mvn test
❖   artifacts:
❖     reports:
❖       junit:
❖         - target/surefire-reports/TEST-*.xml
```

⁸ <https://spring.io/projects/spring-boot>

Για την υλοποίηση και την εκτέλεση των δοκιμών μονάδας επιλέχθηκε η βιβλιοθήκη Surefire⁹. Για την εκτέλεση των δοκιμών στο πλαίσιο του αγωγού, επιλέχθηκε μια ελαχιστοποιημένη εικόνα που φέρει τα απαραίτητα πακέτα, όπως είναι το Maven στην έκδοση 3.8.4 και το Java Development Kit στην έκδοση 17 (παραμετροποιήσιμη από τον χρήστη). Με την εικόνα Maven να είναι παραμετροποιήσιμη από τον χρήστη, γίνεται φανερό ότι αυτός ο αγωγός θα μπορούσε να χρησιμεύσει σαν πρότυπο για όλες τις εφαρμογές που βασίζονται στο Maven.

Η εντολή με την οποία ελέγχεται η εφαρμογή σε αυτό το στάδιο είναι η `mvn test`. Η εντολή χρησιμοποιείται στο πλαίσιο του Maven για τη μεταγλώττιση του κώδικα και την εκτέλεση των δοκιμών μονάδας που έχουν οριστεί σε μια εφαρμογή Java. Αυτή η διαδικασία εκτελεί τις δοκιμές μονάδας, ελέγχοντας τη συμπεριφορά του κώδικα και διασφαλίζοντας τη σωστή λειτουργία των κλάσεων. Με την εντολή `mvn test`, οι προγραμματιστές μπορούν να διασφαλίσουν ότι ο κώδικας τους πληροί τις προδιαγραφές των δοκιμών, προάγοντας την ποιότητα του λογισμικού.

Για να θεωρείται επιτυχής η ενέργεια, πρέπει όλες οι δοκιμές που έχουν οριστεί στον κώδικα να αποβούν επιτυχείς, διαφορετικά η ενέργεια αποτυγχάνει και η εκτέλεση του αγωγού σταματά.

Η έξοδος της εντολής είναι ένα σύνολο αρχείων τύπου XML, τα οποία ο αγωγός αποθηκεύει σαν αναφορές, ώστε μετά το πέρας της εκτέλεσης να είναι διαθέσιμες στο UI των αποτελεσμάτων του αγωγού.

➤ **Ενέργεια: Δοκιμές ένοποίησης & Έλεγχος κάλυψης κώδικα (Integration Tests & Code Coverage)**

Οι δοκιμές ένοποίησης (integration tests) είναι δοκιμές που ελέγχουν τον τρόπο με τον οποίο τα διάφορα συστατικά του λογισμικού συνδυάζονται και αλληλεπιδρούν μεταξύ τους. Σκοπός τους είναι να διασφαλίσουν ότι τα συστατικά του λογισμικού ενσωματώνονται σωστά και αλληλεπιδρούν κατάλληλα ώστε να εκτελούνται σωστά οι λειτουργίες του λογισμικού. Ο έλεγχος κάλυψης κώδικα επαληθεύει ότι ένα επαρκές ποσοστό (ορισμένο από τον χρήστη) κώδικα ελέγχεται κατά τη φάση των δοκιμών από αυτοματοποιημένα σενάρια, όπως αυτά που εκτελούνται στις μέχρι τώρα δυο ενέργειες. Αυτοί οι δυο έλεγχοι διεκπεραιώνονται ταυτόχρονα λόγω του ότι εκτελούνται από το ίδιο εργαλείο, για μεγαλύτερη εξοικονόμηση χρόνου κατά την εκτέλεση του αγωγού. Εάν έστω και μια δοκιμή ένοποίησης ή η κάλυψη κώδικα είναι μικρότερη

⁹ <https://maven.apache.org/surefire/maven-surefire-plugin/>

του προκαθορισμένου ορίου, η εκτέλεση του αγωγού σταματά.

Ο κώδικας είναι ο παρακάτω:

```
❖ #Integration Tests
❖ Integration-Tests-Code-Coverage:
❖   stage: test-code
❖   image: ${MAVEN_BUILD_IMAGE}
❖   needs:
❖     - Unit-Tests
❖   script:
❖     - mvn verify && mvn jacoco:report
❖   artifacts:
❖     reports:
❖       junit:
❖         - target/failsafe-reports/TEST-*.xml
❖   paths:
❖     - target/site/jacoco/index.html
```

Και για την τρέχουσα ενέργεια επιλέχθηκε μια ελαχιστοποιημένη εικόνα που φέρει τα απαραίτητα πακέτα όπως είναι το Maven στην έκδοση 3.8.4 και το Java Development Kit στην έκδοση 17 (παραμετροποιήσιμη από τον χρήστη). Όπως φαίνεται και από τον κώδικα, για να εκτελεστεί η ενέργεια πρέπει προηγουμένως να έχει ολοκληρωθεί με επιτυχία η ενέργεια **Unit-Tests**. Για την υλοποίηση και εκτέλεση των δοκιμών ενοποίησης και του ελέγχου κάλυψης κώδικα έγινε χρήση των βιβλιοθηκών Failsafe¹⁰ και Jacoco¹¹. Το πρόσθετο Failsafe χρησιμοποιείται στο Maven για τη διαχείριση των δοκιμών ενοποίησης (integration tests). Επιτρέπει την εκτέλεση δοκιμών που αφορούν τον συντονισμό και τον έλεγχο των διαφορετικών συστατικών ενός έργου λογισμικού. Το JaCoCo είναι ένα εργαλείο μέτρησης κάλυψης κώδικα για τη γλώσσα Java. Χρησιμοποιείται για τον έλεγχο του ποσοστού κάλυψης του κώδικα από τις δοκιμές, προσφέροντας αναλυτικές αναφορές. Η εντολή **mvn verify && mvn jacoco:report** στο Maven εκτελεί τις δοκιμές ενοποίησης και κάλυψης κώδικα.

Για να θεωρείται επιτυχής η ενέργεια, πρέπει όλες οι δοκιμές που έχουν οριστεί στον κώδικα να αποβούν επιτυχείς, καθώς επίσης και η κάλυψη του κώδικα από δοκιμές να είναι ίση ή μεγαλύτερη από την τιμή που έχει δηλωθεί στο αρχείο pom.xml μιας εφαρμογής Java. Σε διαφορετική περίπτωση, η ενέργεια αποτυγχάνει και η εκτέλεση του αγωγού σταματά.

Η έξοδος της εντολής είναι ένα σύνολο αρχείων τύπου XML, τα οποία ο αγωγός αποθηκεύει ως αναφορές, ώστε μετά το πέρας της εκτέλεσης να είναι διαθέσιμες στο UI των

¹⁰ <https://maven.apache.org/surefire/maven-failsafe-plugin/>

¹¹ <https://www.eclemma.org/jacoco/trunk/doc/maven.html>

αποτελεσμάτων του αγωγού (Εικ. 22), καθώς και ένα αρχείο html, το οποίο ο προγραμματιστής μπορεί να λάβει για να ενημερωθεί για το αποτέλεσμα των δοκιμών κάλυψης (Εικ. 23).

Summary

8 tests 0 failures 0 errors 100% success rate 1.86s

Jobs

Job	Duration	Failed	Errors	Skipped	Passed	Total
Unit-Tests	1.74s	0	0	0	6	6
Integration-Tests-Code-Coverage	120.00ms	0	0	0	2	2

Εικόνα 22 - Αποτελέσματα δοκιμών μονάδας και δοκιμών ενοποίησης

java-maven-junit-2

Element	Missed Instructions	Cov.	Missed Branches	Cov.
com.example.javamavenjunit-helloworld	2892	76%	17	87%
com.example.javamavenjunit-helloworld.controllers	11	0%		n/a
Total	39 of 131	70%	1 of 8	87%

Created with [JaCoCo](#) 0.8.11.202310140853

Εικόνα 23 - Αποτελέσματα ελέγχου κάλυψης κώδικα

5.3.1.2 Στάδιο: Έλεγχος Κώδικα (Scan Code)

Σε αυτό το στάδιο ελέγχεται ο πηγαίος κώδικας σε συνδυασμό μαζί με τις ενσωματωμένες βιβλιοθήκες της υπό ανάπτυξη εφαρμογής από ένα εύρος διαθέσιμων εργαλείων, προκειμένου να επαληθευτεί ότι δεν περιέχουν γνωστές ευπάθειες ή κακές πρακτικές. Αυτό το στάδιο περιλαμβάνει 3 ενέργειες (jobs) :

➤ Ενέργεια: Στατική Ανάλυση Ασφάλειας Λογισμικού (SAST-SonarQube)

Για να επιτευχθεί η στατική ανάλυση ασφαλείας του παραγόμενου λογισμικού, δημιουργήθηκε εικονική μηχανή, στην οποία εγκαταστάθηκε το λογισμικού ανοικτού κώδικα SonarQube σε μορφή δοχείου. Ο έλεγχος που πραγματοποιείται σε αυτή την ενέργεια επιτελεί τις ακόλουθες λειτουργίες:

- **Συμμόρφωση με τα Πρότυπα Κώδικα:** Ελέγχει αν ο κώδικας ακολουθεί τα αναγνωρισμένα πρότυπα και κανόνες προγραμματισμού.

- **Ανίχνευση Κώδικα για Κινδύνους Ασφαλείας:** Αναζητά πιθανούς κινδύνους ασφαλείας στον κώδικα, όπως ευπάθειες σε επιθέσεις.
- **Ποιότητα Κώδικα:** Αξιολογεί τη γενική ποιότητα του κώδικα, συμπεριλαμβανομένης της αναγνώσιμης δομής και των βέλτιστων πρακτικών.
- **Απόδοση Εφαρμογής:** Ανιχνεύει πιθανές προβληματικές περιοχές που μπορεί να επηρεάσουν την επίδοση.
- **Ευανάγνωστος Κώδικας:** Εξετάζει τον κώδικα για να επιβεβαιώσει ότι είναι καλά σχολιασμένος και κατανοητός.

Σε περίπτωση που ο έλεγχος δεν ικανοποιήσει τα προκαθορισμένα κατώφλια ποιότητας (Quality Gates), η εκτέλεση του αγωγού σταματά.

Ο κώδικας είναι ο παρακάτω:

```
❖ sast-sonarqube:
❖ stage: scan-code
❖ image: ${MAVEN_BUILD_IMAGE}
❖ Variables:
❖ # Defines the location of the analysis task cache
❖ SONAR_USER_HOME: "${CI_PROJECT_DIR}/.sonar"
❖ # Instruct git to fetch all the commits of the project, required by the
  analysis task
❖ GIT_DEPTH: "0"
❖ Script: - mvn verify sonar:sonar -Dsonar.qualitygate.wait=true
  -Dsonar.host.url=$SONAR_HOST_URL -Dsonar.login=$SONAR_TOKEN
❖ allow_failure: false
❖ cache:
❖ key: "${CI_JOB_NAME}"
❖ paths:
❖ - .sonar/cache
```

Και για την τρέχουσα ενέργεια επιλέχθηκε μια ελαχιστοποιημένη εικόνα που φέρει τα απαραίτητα πακέτα όπως είναι το Maven στην έκδοση 3.8.4 και το Java Development Kit στην έκδοση 17 (παραμετροποιήσιμη από τον χρήστη). Για τους σκοπούς αυτής της ενέργειας έχουν ρυθμιστεί οι απαραίτητες εξουσιοδοτήσεις ανάμεσα στην πλατφόρμα Gitlab και το SonarQube για αμοιβαία πρόσβαση.

Ο κώδικας της ενέργειας αρχικά ορίζει το προσωρινό ευρετήριο στο οποίο θα αποθηκευτούν τα προϊόντα της ανάλυσης. Ορίζοντας το επίπεδο git στο 0, επιλέγουμε να

σαρωθεί το επιλεγμένο κλαδί, με το πλήρες ιστορικό των υποβολών κώδικα, σε αντιδιαστολή από έναν ρηχό κλώνο (shallow clone). Μέσα στην Java εφαρμογή μας έχει οριστεί ο στόχος (goal) του Maven και το Sonar plugin στο αρχείο pom.xml, το οποίο και καλείται για την ανάλυση. Ο αγωγός με την ίδια εντολή ρυθμίζεται ώστε να περιμένει την απάντηση από το SonarQube, και συγκεκριμένα την έγκριση ή όχι από τις πύλες ποιότητας (Quality Gates).

Οι πύλες ποιότητας αναφέρονται σε ένα σύνολο προκαθορισμένων κριτηρίων ή συνθηκών που ένα έργο λογισμικού πρέπει να πληροί προτού προχωρήσει στο επόμενο στάδιο. Στο δοχείο του SonarQube είναι ενεργοποιημένες οι προεπιλεγμένες και προεγκατεστημένες πύλες ποιότητας, οι οποίες κρίθηκαν επαρκείς καθώς λειτουργούν με τα εξής κριτήρια:

- Δεν υπάρχουν νέα θέματα, όπως Bugs κώδικα που μπορεί να οδηγήσουν σε διακοπή του προγράμματος ή Code Smells που μπορεί να δυσκολέψουν την συντήρηση του κώδικα.
- Το ποσοστό διπλότυπου κώδικα είναι μικρότερο του 3%.
- Το ποσοστό κάλυψης κώδικα από δοκιμές είναι άνω του 80%.
- Τα νέα τρωτά σημεία, από άποψη ασφαλείας, έχουν επιδιορθωθεί ή σημειωθεί ως ασφαλή από τη σελίδα εποπτείας και διαχείρισης τρωτών σημείων του SonarQube.

Για να θεωρείται επιτυχής η ενέργεια, πρέπει ο νέος κώδικας που υποβλήθηκε να τύχει της έγκρισης μέσα από την πύλη ποιότητας του SonarQube. Σε διαφορετική περίπτωση, η ενέργεια αποτυγχάνει και η εκτέλεση του αγωγού σταματά.

Η αξιολόγηση που έχει παραχθεί είναι διαθέσιμη μέσα στο περιβάλλον του SonarQube, εφόσον ο χρήστης επισκεφτεί τη σελίδα που έχει διαμορφωθεί για την εφαρμογή, της οποίας ο σύνδεσμος παρέχεται μέσα από το αρχείο καταγραφής της ενέργειας.

➤ **Ενέργεια: Ανίχνευση μυστικών (Secret Detection)**

Όπως υπογραμμίστηκε στην Ενότητα 4.2.5, ένας συνηθισμένος τρόπος να παρακαμφθεί κάθε μηχανισμός ασφαλείας ενός συστήματος CI/CD είναι η φανέρωση - ακούσια ή εκούσια - μυστικών ή συνθηματικών μέσα στον κώδικα της εφαρμογής. Η τρέχουσα ενέργεια, ελέγχει το αποθετήριο για εκτεθειμένα μυστικά με βάση γνωστά μοτίβα, και εάν τέτοια ανιχνευθούν, σταματά την εκτέλεση του αγωγού τυπώνοντας ανάλογο σφάλμα.

Ο κώδικας είναι ο παρακάτω:

- ❖ *#Scans for secrets (.crt, .pems) revealed in our Code using gitleaks tool*
- ❖ `secret-detection:`

```
❖ stage: scan-code
❖ image:
❖   name: "zricethezav/gitleaks"
❖   entrypoint: [""]
❖ script:
❖   #Scan the repos current state in verbose manner, hide any secrets from
❖   #Logs and place the findings in a exposed-secrets.json file
❖   - |
❖       echo -e "\e[94mChecking for exposed secrets..\e[0m"
❖       set +e
❖       gitleaks detect -v -s . --redact --no-git > exposed-secrets.json
❖       GITLEAKS_EXIT_CODE=$?
❖       set -e
❖       cat exposed-secrets.json
❖       case "${GITLEAKS_EXIT_CODE}" in
❖           0)
❖               echo -e "\e[92mNo secrets have been exposed\e[0m"
❖               ;;
❖           1)
❖               echo -e "\e[31mExposed secrets have been detected, exiting with
❖ error.\e[0m"
❖               exit 1
❖               ;;
❖       esac
❖ artifacts:
❖   reports:
❖     secret_detection: exposed-secrets.json
❖   expire_in: 7 days
```

Για την εκτέλεση της ανίχνευσης μυστικών χρησιμοποιείται το γνωστό εργαλείο Gitleaks¹² σε μορφή δοχείου. Σε αυτόν ειδικά τον χώρο υπάρχουν και έτερες αξιόλογες λύσεις όπως π.χ. Το Trufflehog ή το Git-Secrets. Η επιλογή του συγκεκριμένου εργαλείου έγινε με γνώμονα τόσο την υψηλότερη δημοφιλία του (συγκρίνοντας τα αποθετήρια github), αλλά και άρθρα όπως το [90]. Ο κώδικας παρακάμπτει την αρχική εντολή του δοχείου, θέτοντας ένα κενό entrypoint, για

¹² <https://github.com/gitleaks/gitleaks>

μεγαλύτερη ευελιξία στον καθορισμό των ενεργειών, ενώ παράλληλα σε κάθε σημείο εμφανίζει ανάλογο μήνυμα, το οποίο καθιστά το αρχείο καταγραφής της ενέργειας πιο κατανοητό.

Αρχικά μέσω της εντολής `set +e` απενεργοποιούμε τη δυνατότητα εξόδου του προγράμματος με κωδικό αποτυχίας (1) σε περίπτωση ευρημάτων, ώστε να τυπωθεί στη συνέχεια το κατάλληλο ενημερωτικό μήνυμα. Με την εντολή `gitleaks detect -v -s . --redact --no-git > exposed-secrets.json`, εκτελείται σάρωση για μυστικά με τα ακόλουθα ορίσματα:

- **-v** : Τυπώνει αναλυτικά τα αποτελέσματα
- **-s** : Ορίζει το τρέχον ευρετήριο για σάρωση
- **--redact** : Αποκρύβει τα μυστικά από το αρχείο καταγραφής
- **--no-git** : Ελέγχει όλα τα αρχεία, όχι μόνο τα τροποποιημένα
- **> exposed-secrets.json** : Δημιουργεί ένα αρχείο json με τα αποτελέσματα.
- **GITLEAKS_EXIT_CODE=\$?** : Αποθηκεύει τον κωδικό εξόδου της προηγούμενης εντολής σε μεταβλητή (1 για εύρεση μυστικών, 0 για μη εύρεση)

Στη συνέχεια η μεταβλητή που περιέχει τον κωδικό εξόδου ελέγχεται ώστε να τυπωθούν τα ανάλογα μηνύματα που θα φανούν στο αρχείο καταγραφής, και ακολούθως η ενέργεια τερματίζει επιτυχώς ή ανεπιτυχώς, ανάλογα με την τιμή.

Η ενέργεια σε κάθε περίπτωση παράγει ένα αρχείο .json, το οποίο αποθηκεύεται ώστε να ελεγχθεί από τον χρήστη μέσα από το UI του Gitlab.

➤ **Ενέργεια: Έλεγχος Εξαρτήσεων (Dependency Check)**

Στην Ενότητα 4.1.12 είδαμε την κρισιμότητα που έχει η ασφάλεια των βιβλιοθηκών από τις οποίες εξαρτάται το υπό ανάπτυξη λογισμικού. Κατά την εκτέλεση αυτής της ενέργειας, ο έλεγχος συνίσταται στην ανάλυση του λογισμικού στις συστατικά του, και ακολούθως στη σάρωση της βάσης των γνωστών ευπαθειών (CVE Database) για κάθε μια από αυτές τις συνιστώσες. Η εκτέλεση του αγωγού σταματά εάν ανιχνευθούν ευπάθειες, των οποίων το συνολικό σκορ είναι ανώτερο της προκαθορισμένης τιμής. Για τη συνεχή αναβάθμιση της έκδοσης του εργαλείου, αλλά κυρίως την επιτάχυνση της διαδικασίας ώστε οι τελευταίες ενημερώσεις να είναι ανά πάσα στιγμή διαθέσιμες πριν την εκτέλεση του αγωγού, το δοχείο που χρησιμοποιείται για αυτό το σκοπό παράγεται από ένα ξεχωριστό αποθετήριο και αγωγό, ο οποίος εκτελείται κάθε βράδυ.

Ο κώδικας είναι ο παρακάτω:

```
❖ #Scans for vulnerabilities in our code dependencies/libraries
❖ dependency-check:
❖ stage: scan-code
❖ image: 192.168.1.101:5050/aegean-apps/helper-apps/dependency-checker:latest
❖ script:
❖ - mvn org.owasp:dependency-check-maven:check -DdataDirectory=/owasp
  -DfailBuildOnCVSS=8
❖ allow_failure: false
❖ artifacts:
❖   when: always
❖   paths:
❖     - target/dependency-check-report.html
```

Κατά την εκτέλεση αυτής της ενέργειας γίνεται χρήση ενός δοχείου με το εργαλείο Owasp Dependency-check. Το δοχείο που χρησιμοποιείται δομείται κάθε βράδυ σε ξεχωριστό αποθετήριο και αγωγό στην πλατφόρμα Gitlab, ώστε να ελαχιστοποιηθεί ο χρόνος εκτέλεσης, καθώς η ανανέωση των βάσεων δεδομένων έχει ήδη πραγματοποιηθεί.

Σύμφωνα με τις εντολές αυτής της ενέργειας, εκτελείται για άλλη μια φορά ένας στόχος του πρόσθετου maven org.owasp:dependency-check-maven προκειμένου να ελεγχθούν σύμφωνα με τη βάση ευπαθειών οι βιβλιοθήκες της εφαρμογής. Ορίζεται το ευρετήριο στο οποίο θα αποθηκευτούν τα δεδομένα, ενώ η συνθήκη αποτυχίας της ενέργειας είναι η εύρεση ευπαθούς βιβλιοθήκης με κρισιμότητα ίση ή ανώτερη του 8. (Κλίμακα 1-10).

Σε περίπτωση όπου βρεθεί τέτοια ευπάθεια, η ενέργεια αποτυγχάνει και η εκτέλεση του αγωγού τερματίζεται. Σε κάθε περίπτωση, η ενεργεια αποθηκεύει μια αναφορά ευπαθειών, την οποία στη συνέχεια μπορεί να λάβει και να επιθεωρήσει ο χρήστης.

5.3.1.3 Στάδιο: Οικοδόμηση Εικόνας δοχείου (Build Image)

Πριν την οικοδόμηση της εικόνας, θα διενεργηθούν έλεγχοι ώστε να βεβαιωθεί ότι το Dockerfile είναι ασφαλές προς οικοδόμηση.

➤ Ενέργεια: Έλεγχος του Dockerfile (Hadolint Dockerfile)

Η ενέργεια ελέγχει ότι το Dockerfile αρχείο που χρησιμοποιήθηκε τηρεί τις βέλτιστες πρακτικές ασφαλείας, σε συνδυασμό με εξατομικευμένες επιλογές του χρήστη. Όπως είδαμε στην Ενότητα 4.2.1, η σωστή ρύθμιση του αρχείου Dockerfile είναι κρίσιμης σημασίας για την ασφάλεια της παραγόμενης εικόνας. Για το λόγο αυτό, τόσο αυτή η ενέργεια όσο και η επόμενη σταματούν την εκτέλεση του αγωγού εάν ο έλεγχος φανερώσει

σχετικές παραβιάσεις των πολιτικών ασφαλείας.

Ο κώδικας είναι ο παρακάτω:

```
❖ #Lint and check integrity of Dockerfile
❖ hadolint-dockerfile:
❖   stage: build-image
❖     name: hadolint/hadolint:latest-debian
❖     #Override the entry point
❖     entrypoint: [""]
❖     script:
❖       - |
❖         echo -e "\e[94mChecking dockerfile with Hadolint..\e[0m"
❖         set +e
❖         mkdir -p reports
❖         hadolint -t error --trusted-registry 192.168.1.101 --trusted-registry
❖         docker.io --require-label author:text --require-label version:semver --error
❖         DL3006 --error DL3007 --error DL3022 --error DL3030 --error DL3045 -f
❖         gitlab_codeclimate ./Dockerfile > reports/hadolint-$(md5sum Dockerfile | cut
❖         -d" " -f1).json
❖         HADOLINT_EXIT_CODE=$?
❖         echo -e "$$(cat reports/hadolint-$(md5sum Dockerfile | cut -d" "
❖         -f1).json)"
❖         set -e
❖         case "${HADOLINT_EXIT_CODE}" in
❖           0)
❖             echo -e "\e[92mHadolint check was successull.\e[0m"
❖             ;;
❖           1)
❖             echo -e "\e[31mDockerfile violations were detected. Please follow
❖             Hadolint rules.\e[0m"
❖             exit 1
❖             ;;
❖         esac
❖     artifacts:
❖       name: "$CI_JOB_NAME artifacts from $CI_PROJECT_NAME on
❖       $CI_COMMIT_REF_SLUG"
❖       expire_in: 1 day
❖       when: always
❖       reports:
❖         codequality:
❖           - "reports/*"
```


- ❖ paths:
- ❖ - "reports/*"

Η ενέργεια χρησιμοποιεί την τελευταία έκδοση του δοχείου με το εργαλείο Hadolint¹³ για να ελέγξει ότι το Dockerfile αρχείο που χρησιμοποιήθηκε τηρεί τις βέλτιστες πρακτικές ασφαλείας, σε συνδυασμό με εξατομικευμένες επιλογές του χρήστη.

Ο κώδικας υλοποιεί τα εξής:

- **mkdir -p reports** : Αρχικά δημιουργείται ένας φάκελος στον οποίο θα αποθηκευτούν οι παραγόμενες αναφορές
- **Hadolint -t error** : Εκτελεί τη σάρωση, και θα παράξει κωδικό αποτυχίας εάν υπάρχει εύρημα επιπέδου “error”
- **trusted-registry 192.168.1.101 --trusted-registry docker.io** : Μόνο αυτά τα αποθετήρια εικόνων που καθορίζονται είναι αποδεκτά για άντληση εικόνων βάσης
- **--error DL3006 --error DL3022 --error DL3030 --error DL3045** : Διαφοροποιεί την εκτέλεση της σάρωσης από τις προεπιλεγμένες ρυθμίσεις, και απαιτεί η εικόνα βάσης να φέρει σαφώς ορισμένη ετικέτα (tag), να μην υπάρχει προβληματική σύνταξη σε περίπτωση κατασκευής πολλαπλών φάσεων (Multi-Stage Dockerfile), να μην προκύπτει αναμονή για είσοδο χρήστη κατά την εκτέλεση του Dockerfile και να είναι πάντα σαφώς ορισμένο το ευρετήριο αντιγραφής αρχείων.
- **-f gitlab_codeclimate Dockerfile > reports/hadolint-\$(md5sum Dockerfile | cut -d" " -f1).json** : Αποθηκεύει τα αποτελέσματα σε μορφή json με συγκεκριμένο όνομα αρχείου.

Η ενέργεια αποτυγχάνει και η εκτέλεση του αγωγού σταματά εάν ανιχνευθεί σφάλμα σύμφωνα με τα δοσμένα κριτήρια, ενώ ο χρήστης μπορεί να κατεβάσει το αρχείο που παρήχθη μέσα από τη διεπαφή χρήσης (User Interface - UI) του αγωγού.

➤ **Ενέργεια: Έλεγχος για root χρήστη εντός του Dockerfile (No Root dockerfile)**

Όπως είδαμε επίσης στην Ενότητα 4.2.1, η εκτέλεση του δοχείου σαν χρήστης root (ριζικός) μπορεί να οδηγήσει σε σημαντικά αυξημένη επιφάνεια επίθεσης. Ενώ η προηγούμενη ενέργεια απορρίπτει αρχεία που ρητά δηλώνουν σαν χρήστη του δοχείου τον χρήστη root, εντοπίστηκε αδυναμία να εφαρμοσθεί ο κανόνας όταν δεν υπάρχει καμία τέτοια δήλωση στο αρχείο, οπότε και σιωπηρά ο χρήστης παραμένει ο root. Μάλιστα, αυτή

¹³ <https://github.com/hadolint/hadolint>

η σοβαρή έλλειψη είναι επίσημα καταγεγραμμένη στην ιστοσελίδα του εργαλείου¹⁴. Για αυτό το λόγο σε ένα δοχείο εκτελείται ένα αυτόνομο script που δημιουργήθηκε εξ αρχής από τον συγγραφέα, και το οποίο διασφαλίζει ότι στο Dockerfile ή στη βασική εικόνα αυτού υπάρχει ρητή δήλωση για χρήστη διάφορο του root.

Ο κώδικας είναι ο παρακάτω:

```
❖ #Hadolint currently cant detect a missing USER command (Github Issue: #328),  
  so it needs be verified via script  
❖ no-root-dockerfile:  
❖   stage: build-image  
❖   image: alpine:latest  
❖   before_script:  
❖     #Add curl  
❖     - apk add --no-cache --no-check-certificate curl  
❖     #Add jq to manipulate json objects  
❖     - apk add --no-cache --no-check-certificate jq  
❖   script:  
❖     #Hadolint currently cant detect a missing USER command (Github Issue:  
#328), so it needs to be verified via script  
❖     #Following script checks the Dockerfile if the last USER command is not  
for the root user  
❖     #So there needs to be at least USER command, the last of which must not be  
USER root or USER 0  
❖     #-i is case insensitive, -c is number of occurrences, -q means dont print  
the result, only execute it  
❖     - |  
❖     if (grep -i "USER" Dockerfile | tail -1 | grep -iq root) || (grep -i  
"USER" Dockerfile | tail -1 | grep -iq ' \0' ) ;  
❖     then  
❖       echo -e "\e[31mUser in Dockerfile is set to ROOT.\e[0m Please use  
the \e[31m USER command to change User\e[0m, as running a container as root  
is a security vulnerability."  
❖       exit 1  
❖       #If there is no USER command, check the base image for inheritance of  
the USER  
❖       elif [ $(grep -c "USER" Dockerfile) -lt 1 ] ;  
❖       then  
❖         #Since Dockerfile did not meet the non-root criteria, we will check  
if the base image is non root so we dont have a false positive result
```

¹⁴ <https://github.com/hadolint/hadolint/issues/328>

```
❖      #Lets find the Base Image, and if its a Multi-Stage Dockerfile, keep
the Last 'FROM entry'
❖      base_image=$(grep -o 'FROM [^[:space:]]*' Dockerfile | tail -1 | awk
'{print $2 $3}')
❖      echo -e "\e[33mDockerfile final image detected: $base_image.\e[0m"
❖      #Lets format it to two parts of the later URL
❖      image=$(echo "$base_image" | cut -d ':' -f1)
❖      # Check if before_colon does not contains a "/", else append
/library to it to match dockerhub's syntax for official images
❖      if [[ $image = */* ]]; then
❖          image="library/$image"
❖      fi
❖      tag=$(echo "$base_image" | cut -d ':' -f2)
❖      url="https://hub.docker.com/v2/repositories/$image/tags/$tag/images"
❖      echo -e "\e[33mChecking base image layers from: $url.\e[0m"
❖      # Use curl to make the HTTP request, pipe the response to jq, and
save "instruction" values in a variable
❖      instructions=$(curl -s -S -X GET $url | jq -r
'.[0].layers[].instruction')
❖      #In the instructions there needs to be at least USER command, the
last of which must not be USER root or USER 0
❖      echo -e "\e[33mDockerfile, did not contain a USER Command. Checking
the layers of the base image on Dockerhub for the USER command\e[0m"
❖      echo "$instructions"
❖      #So there needs to be at least USER command, the last of which must
not be USER root or USER 0
❖      if [ $(echo "$instructions" | grep -c "USER") -lt 1 ] || (echo
"$instructions" | grep -i "USER" | tail -1 | grep -iq root) || (echo
"$instructions" | grep -i "USER" | tail -1 | grep -iq ' \0') ;
❖          then
❖              echo -e "\e[31mUser in Base Image is set to ROOT.\e[0m Please use
the \e[31m USER command to change User\e[0m, as running a container as root
is a security vulnerability."
❖              exit 1
❖          fi
❖      else
❖          echo -e "\e[32mUser in Dockerfile is not root, proceeding."
❖          exit 0
❖      fi
```

Για να επιδιορθωθεί αυτό το κενό ασφαλείας, δημιουργήθηκε η τρέχουσα ενέργεια, η οποία εκτελείται σε ένα δοχείο alpine linux, στο οποίο εγκαθίστανται οι λειτουργικότητες curl για προσπέλαση ιστοσελίδων και jq για διαχείριση στοιχείων JSON. Αρχικά, ο κώδικας ψάχνει το αρχείο Dockerfile για όλες τις δηλώσεις χρήστη, και σε περίπτωση που ο τελευταίος δηλωμένος χρήστης είναι ο root ή ‘0’, η ενέργεια αποτυγχάνει και η εκτέλεση του αγωγού σταματά.

Στη συνέχεια, εφόσον δεν υπάρχει καμία τέτοια δήλωση χρήστη, ο κώδικας εντοπίζει στο Dockerfile την εικόνα βάσης, και αναζητά τα στρώματα (layers) από τα οποία συντίθεται στο Docker Hub. Αφού εντοπίσει τα στρώματα, για άλλη μια φορά εκτελεί τον ίδιο έλεγχο ώστε να ανιχνευθεί ο τελευταίος δηλωμένος χρήστης.

Εάν και πάλι λείπει τέτοια δήλωση, ή ο χρήστης είναι ο root ή ο ‘0’, η ενέργεια αποτυγχάνει και η εκτέλεση του αγωγού σταματά, ενημερώνοντας τον χρήστη με κατάλληλα μηνύματα.

➤ Ενέργεια: Οικοδόμηση Εικόνας (Build Image)

Εφόσον όλες οι προηγούμενες ενέργειες του σταδίου στέφθηκαν με επιτυχία, αυτή η ενέργεια οικοδομεί την εικόνα του δοχείου. Η εικόνα που παράγεται δεν αποθηκεύεται σε αυτό το στάδιο στο αποθετήριο εικόνων, αλλά τροφοδοτείται με τη συμπιεσμένη μορφή .tar στις επόμενες ενέργειες του αγωγού για να ελεγχθεί περαιτέρω, πριν την υποβολή της.

Ο κώδικας είναι ο παρακάτω:

```
❖ #Builds the Image into a tar archive, without needing privileged Docker in Docker
❖ build-image:
❖   stage: build-image
❖   image:
❖   name:
❖     gcr.io/kaniko-project/executor@sha256:97c78eedb0560b8fcf64900abdb810f84f9882d033421f4aee1e6559f42b7e87
❖     needs:
❖       - hadolint-dockerfile
❖       - no-root-dockerfile
❖     entrypoint: [""]
❖     script:
❖       - /kaniko/executor
❖       --context "${CI_PROJECT_DIR}"
❖       --dockerfile "${CI_PROJECT_DIR}/Dockerfile"
❖       --no-push --tar-path "image.tar"
```

- ❖ artifacts:
- ❖ paths:
- ❖ - **image.tar**
- ❖ expire_in: **2 hrs**
- ❖ when: **on_success**

Για αυτή την ενέργεια δόθηκε ιδιαίτερη έμφαση στην αποφυγή χρήσης τυπικών μεθοδολογιών και ενεργειών, με τις οποίες για την οικοδόμηση της εικόνας απαιτείται χρήση ενός privileged (προνομιούχου) Gitlab Runner για να εκτελεστεί το πρόγραμμα Docker μέσα σε δοχείο (Docker in Docker). Ως εκ τούτου αξιοποιήθηκε ένα δοχείο με το λογισμικό Kaniko, το οποίο εκτελείται σε έναν απλό Gitlab Runner. Επιπρόσθετα, για να αποφευχθεί άσκοπη υποβολή της εικόνας στο αποθετήριο εικόνων, αυτή αποθηκεύεται σε συμπιεσμένη μορφή .tar και περνά σαν τέχνημα στα υπόλοιπα στάδια του αγωγού προς έλεγχο.

Σύμφωνα με τις εντολές, εκτελούνται τα εξής:

- **/kaniko/executor** : Καλείται το εκτελέσιμο αρχείο Kaniko
- **--context "\${CI_PROJECT_DIR}"** : Ορίζεται ο φάκελος στον οποίο περιέχεται ο κώδικας της εφαρμογής μέσω της προ-ορισμένης μεταβλητής CI_PROJECT_DIR
- **--dockerfile "\${CI_PROJECT_DIR}/Dockerfile"** : Ορίζεται η τοποθεσία του Dockerfile, βάση του οποίου θα σχηματιστεί η εικόνα (βλ. Ενότητα 4.2.1)
- **--no-push** : Δηλώνεται η μη υποβολή της εικόνας στο αποθετήριο εικόνων
- **--tar-path "image.tar"** : Δηλώνεται η αποθήκευση της εικόνας σε συμπιεσμένη μορφή .tar

Σύμφωνα με τον κώδικα (τελευταίες 5 γραμμές), το τέχνημα της εικόνας παραμένει διαθέσιμο στον αγωγό για 2 ώρες, μετά τις οποίες διαγράφεται για εξοικονόμηση χώρου.

5.3.1.4 Στάδιο: Έλεγχος εικόνας (Scan Image)

Σε αυτό το στάδιο γίνονται όλοι οι απαραίτητοι έλεγχοι στο αρχείο εικόνας .tar ώστε να επαληθευτεί ότι οι προδιαγραφές της εικόνας πληρούν όλους τους καθορισμένους κανόνες ασφαλείας που επιβάλλονται από τις απαιτήσεις της εργασίας, και οι οποίοι είναι παραμετροποιήσιμοι από τον εκάστοτε χρήστη, πριν η εικόνα να αποθηκευτεί τελικά στο αποθετήριο εικόνων. Το στάδιο απαρτίζεται από τις παρακάτω ενέργειες, οι οποίες ομαδοποιήθηκαν κυρίως με θεματικό κριτήριο και εκτελούνται παράλληλα για μέγιστη

εξοικονόμηση χρόνου, εφόσον όλες είναι συναφείς με τις ιδιότητες και το περιεχόμενο της εικόνας:

➤ **Ενέργεια: Έλεγχος για κακόβουλο λογισμικό (Scan Image Malware)**

Στην Ενότητα 4.3.4 είδαμε την κρισιμότητα που έχει η ακεραιότητα της τελικής εικόνας για έναν οργανισμό. Ο σκοπός της τρέχουσας ενέργειας είναι να διασφαλιστεί η ασφαλής εκτέλεση της εικόνας χωρίς κίνδυνο ζημίας από παρεισφρέον κακόβουλο λογισμικό. Ο έλεγχος συνίσταται στη σύγκριση των υπογραφών των αρχείων της εικόνας με τη βάση υπογραφών γνωστών κακόβουλων λογισμικών. Η εκτέλεση του αγωγού σταματά εάν ανιχνευθεί οποιοδήποτε γνωστό κακόβουλο λογισμικό.

Ο κώδικας είναι ο παρακάτω:

```
❖ #Scan the image with ClamAV malware scanner
❖ scan-image-malware:
❖   image: 192.168.1.101:5050/aegean-apps/helper-apps/clam-antivirus:latest
❖   stage: scan-image
❖   allow_failure: false
❖   script:
❖     #Scan the image and print only the infected files while saving the report
    to antivirus.log
❖     #The job will fail if it finds a virus (default behaviour exit code 1)
❖     - clamscan --log=antivirus.log --infected -r ./image.tar
❖   artifacts:
❖     reports:
❖       container_scanning: antivirus.log
```

Κατά την εκτέλεση αυτής της ενέργειας γίνεται χρήση ενός δοχείου με το εργαλείο ClamAV. Για τη συνεχή αναβάθμιση της έκδοσης του εργαλείου, αλλά κυρίως την επιτάχυνση της διαδικασίας ώστε οι τελευταίες ενημερώσεις της βάσης υπογραφών να είναι ανά πάσα στιγμή διαθέσιμες πριν την εκτέλεση του αγωγού, το δοχείο που χρησιμοποιείται για αυτό το σκοπό παράγεται από ένα ξεχωριστό αποθετήριο (/aegean-apps/helper-apps/clam-antivirus) και αγωγό, ο οποίος εκτελείται κάθε βράδυ.

Ο κώδικας της τρέχουσας ενέργειας σαρώνει τα αρχεία που περιέχονται στο τέχνημα της εικόνας που έχει παραχθεί σε προγενέστερο στάδιο για υπογραφές γνωστών ιών και κακόβουλου λογισμικού. Εάν ανιχνευθούν τέτοιες υπογραφές, τα μολυσμένα αρχεία εγγράφονται στο αρχείο antivirus.log, η ενέργεια αποτυγχάνει και η εκτέλεση του αγωγού σταματά.

Σε κάθε περίπτωση, το αρχείο καταγραφής είναι διαθέσιμο μέσα από το UI του αγωγού.

➤ **Ενέργεια: Έλεγχος για ευάλωτες βιβλιοθήκες/πακέτα (Scan Image Vulnerabilities)**

Στην Ενότητα 4.2.1 κατέστη φανερή η σημασία της χρήσης ενημερωμένων εικόνων και βιβλιοθηκών για την κατασκευή της εικόνας του δοχείου. Για το σκοπό αυτό, κατά την εκτέλεση της τρέχουσας ενέργειας γίνεται ανίχνευση αυτών των δύο συστατικών της εικόνας, και σύγκρισή αυτών με τη βάση ευπαθειών (CVE Mitre). Σε περίπτωση που ανιχνευθούν ευαλωτότητες επιπέδου υψηλού ή κρίσιμου, η εκτέλεση του αγωγού σταματά.

Ο κώδικας είναι ο παρακάτω:

```
❖ #scan the image with trivy vulnerability scanner
❖ scan-image-vulnerabilities:
❖   image: 192.168.1.101:5050/aegean-apps/helper-apps/trivy-scanner:latest
❖   stage: scan-image
❖   allow_failure: false
❖   script:
❖     - set +e # Disable immediate job failure
❖     # Fail on vulnerability level HIGH & Critical without an available fix,
and print them on screen
❖     - |
❖         trivy image --timeout 10m0s --exit-code 1 --severity CRITICAL
--severity HIGH --ignore-unfixed --template "@/contrib/gitlab.tpl" -o
vulnerability-log.json --cache-dir .trivycache/ --input ./image.tar
❖         TRIVY_EXIT_CODE=$?
❖         echo "${TRIVY_EXIT_CODE}"
❖         set -e
❖         case "${TRIVY_EXIT_CODE}" in
❖             0)
❖                 echo -e "\e[92mNo High or Critical vulnerabilities were detected\e[0m"
❖                 ;;
❖             1)
❖                 echo -e "\e[31mHigh or Critical vulnerabilities detected. Please
update the Dockerfile libraries\e[0m"
❖                 exit 1
❖                 ;;
❖             esac
❖   cache:
❖   paths:
❖     - .trivycache/
```

- ❖ artifacts:
- ❖ reports:
- ❖ #paths:
- ❖ container_scanning: vulnerability-log.json

Για τη συνεχή αναβάθμιση της έκδοσης του εργαλείου, αλλά κυρίως την επιτάχυνση της διαδικασίας ώστε οι τελευταίες ενημερώσεις της βάσης ευπαθειών να είναι ανά πάσα στιγμή διαθέσιμες πριν την εκτέλεση του αγωγού, το δοχείο που χρησιμοποιείται για αυτό το σκοπό παράγεται από ένα ξεχωριστό αποθετήριο και αγωγό, ο οποίος εκτελείται κάθε βράδυ.

Ο κώδικας λειτουργεί ως εξής:

- **- set +e** : Απενεργοποιείται η άμεση έξοδος (κωδικός 1) της ενέργειας, ώστε να γίνει πιο σωστή διαχείριση της ροής
- **trivy image --timeout \${TRIVY_TIMEOUT}** : Εντολή για σάρωση εικόνας, η οποία δε μπορεί να διαρκέσει πάνω από 10 λεπτά (παραμετροποιήσιμο από τον χρήστη), ένα επαρκές χρονικό διάστημα για να αποφευχθεί ενδεχόμενο παράλυσης του δοχείου
- **--exit-code 1 --severity CRITICAL --severity HIGH** : Εάν ανιχνευθούν ευπάθειες υψηλού ή κρίσιμου επιπέδου, η ενέργεια αποτυγχάνει
- **--ignore-unfixed** : Δε λαμβάνονται υπ όψιν ευπάθειες, για τις οποίες δεν υπάρχει γνωστή αντιμετώπιση
- **--input ./image.tar** : Η εικόνα προς σάρωση είναι το τέχνημα του προηγούμενου σταδίου
- **--template "@/contrib/gitlab.tpl" -o vulnerability-log.json** : Η ενέργεια παράγει ένα αρχείο json με συγκεκριμένη μορφοποίηση
- **TRIVY_EXIT_CODE=\$?** : Ο κωδικός εξόδου της ενέργειας αποθηκεύεται σε μια μεταβλητή

Στη συνέχεια γίνεται έλεγχος της τιμής της μεταβλητής, και σε περίπτωση κωδικού αποτυχίας, τυπώνεται το κατάλληλο μήνυμα, η ενέργεια αποτυγχάνει και η εκτέλεση του αγωγού σταματά.

Σε κάθε περίπτωση, το αρχείο vulnerability-log.json είναι διαθέσιμο μέσα από το UI του αγωγού.

5.3.1.5 Στάδιο: Προώθηση εικόνας (Push Image)

➤ Ενέργεια: Προώθηση Εικόνας (Push Image)

Αφού ολοκληρωθούν με επιτυχία οι έλεγχοι επί της εικόνας, η τρέχουσα ενέργεια αναλαμβάνει την προώθηση της ασφαλούς πλέον εικόνας στο αποθετήριο εικόνων της πλατφόρμας Gitlab.

Ο κώδικας είναι ο παρακάτω:

```
❖ #Pushes image to the registry without needing a privileged Runner
❖ push-image:
❖   stage: push-image
❖   image:
❖     name: gcr.io/go-containerregistry/crane:debug
❖     entrypoint: [""]
❖   script:
❖     #Load the Certificate into a file so crane can accept it
❖     - echo "$CA_CERT" > registry.crt
❖     - crane auth login -u $CI_REGISTRY_USER -p $CI_REGISTRY_PASSWORD
❖       $CI_REGISTRY
❖     - crane push --insecure image.tar
❖       ${CI_REGISTRY_IMAGE}:${CI_COMMIT_SHORT_SHA}
```

Για αυτή την ενέργεια δόθηκε ιδιαίτερη έμφαση στην αποφυγή χρήσης τυπικών μεθοδολογιών και ενεργειών, με τις οποίες για την υποβολή της εικόνας στο αποθετήριο απαιτείται χρήση ενός privileged Gitlab Runner για να εκτελεστεί το πρόγραμμα Docker μέσα σε δοχείο (Docker in Docker). Ως εκ τούτου αξιοποιήθηκε ένα δοχείο με το λογισμικό crane¹⁵, το οποίο εκτελείται σε έναν απλό Gitlab Runner.

Ο κώδικας λειτουργεί ως εξής:

- **echo "\$CA_CERT" > registry.crt** : Αποθηκεύει το δημόσιο κλειδί TLS της πλατφόρμας Gitlab μέσα στο δοχείο, ώστε να μην απορριφθεί κατά την επικοινωνία με το αποθετήριο εικόνων.
- **crane auth login -u \$CI_REGISTRY_USER -p \$CI_REGISTRY_PASSWORD \$CI_REGISTRY** : Αυθεντικοποιείται με το αποθετήριο εικόνων που βρίσκεται στην τοποθεσία πίσω από τη μεταβλητή CI_REGISTRY, χρησιμοποιώντας εφήμερα διαπιστευτήρια μιας χρήσης, τα οποία παράγονται και καταστρέφονται στον πλαίσιο μιας και μόνο εκτέλεσης αγωγού.
- **Crane push --insecure image.tar \${CI_REGISTRY_IMAGE}:\${CI_COMMIT_SHORT_SHA}** : Υποβάλλει την εικόνα της εφαρμογής στο αποθετήριο με ετικέτα τη σύντομη έκδοσή του αναγνωριστικού υποβολής κώδικα, ώστε η κάθε εικόνα να είναι σαφώς ορισμένη. Η σημαία (flag) insecure είναι αναγκαία για να επιτραπεί το πιστοποιητικό TLS.

¹⁵ <https://github.com/google/go-containerregistry/blob/main/cmd/crane/doc/crane.md>

5.3.1.6 Στάδιο: Έλεγχος προωθημένης εικόνας (Scan Pushed Image)

Σε αυτό το στάδιο πραγματοποιούνται οι τελευταίοι έλεγχοι της εικόνας πριν την εγκατάστασή της στο δοκιμαστικό/παραγωγικό περιβάλλον. Και εδώ η ομαδοποίηση έγινε με θεματικό κριτήριο, καθώς είναι το τελευταίο βήμα πριν την εγκατάσταση, και λόγω του ότι ο έλεγχος SCAP πρέπει να γίνει σε αποθηκευμένη σε αποθετήριο εικόνα λόγω του περιορισμού του εργαλείου.

➤ Ενέργεια: Έλεγχος SCAP (SCAP Scan)

Στη συγκεκριμένη ενέργεια εκτελείται αξιολόγηση SCAP (Security Content Automation Protocol) της εικόνας στον προνομιούχο (privileged) Gitlab Runner με docker executor. Δυστυχώς, δεν εντοπίστηκε λύση που να εκτελείται σε απλό Gitlab Runner χωρίς ειδικά προνόμια. Το SCAP είναι ένα πρωτόκολλο αυτοματοποιημένης αξιολόγησης ασφάλειας που χρησιμοποιείται για τον έλεγχο της συμμόρφωσης των συστημάτων υπολογιστών με προδιαγραφές ασφάλειας. Τα προφίλ SCAP καθορίζουν τους κανόνες και τις ρυθμίσεις που πρέπει να ελεγχθούν, με στόχο την επιτυχή συμμόρφωση με πρότυπα ασφάλειας. Τα εργαλεία αξιοποιούν διάφορες μεθοδολογίες, όπως CCE (Common Configuration Enumeration), XCCDF (eXtensible Configuration Checklist Description Format), και OVAL (Open Vulnerability and Assessment Language), για να παρέχουν μια ολοκληρωμένη εικόνα της ασφάλειας του συστήματος. Ο έλεγχος γίνεται με βάση το πρότυπο XCCDF, καθώς είναι αποδοτικότερο για τον έλεγχο συστημάτων με βάση προκαθορισμένα κριτήρια, χρησιμοποιώντας το προφίλ Standard System Security Profile. Για τη συνεχή αναβάθμιση της έκδοσης του εργαλείου, αλλά κυρίως την επιτάχυνση της διαδικασίας ώστε οι τελευταίες ενημερώσεις των προδιαγραφών να είναι ανά πάσα στιγμή διαθέσιμες πριν την εκτέλεση του αγωγού, η εικόνα του δοχείου που χρησιμοποιείται για αυτό το σκοπό παράγεται από ένα ξεχωριστό αποθετήριο και αγωγό, ο οποίος εκτελείται κάθε βράδυ.

Ο κώδικας είναι ο παρακάτω:

```
❖ #SCAP SCan for Security Compliance with international standards of choice
❖ # An earlier job/stage must have build and pushed a docker image to
  $IMAGE_NAME that the scap job will scan
❖ scap-scan:
❖   stage: scan-pushed-image
❖   #Only the privileged SCAP Runner will execute this
❖   tags:
❖     - scap
❖   image:
```

```
❖ name: 192.168.1.101:5050/aegean-apps/helper-apps/scap-scanner:latest #  
    Automatically updates every midnight  
❖ entrypoint: [""]  
❖ variables:  
❖ # Instruct Testcontainers to use the daemon of DinD.  
❖ DOCKER_HOST: "tcp://docker:2375"  
❖ #Needed to pass our Self-Signed TLS Certificate  
❖ CA_CERTIFICATE: "$CA_CRT"  
❖ DOCKER_TLS_CERTDIR: ""  
❖ SCAP_SECURITY_GUIDE_VERSION: "placeholder"  
❖ MICROSOFT_SARIF_MULTITool_VERSION: "4.3.5"  
❖ MITRE_SAF_VERSION: "1.2.33"  
❖ SARIF_CONVERTER_VERSION: "0.7.1"  
❖ IMAGE_NAME: ${CI_REGISTRY_IMAGE}:${CI_COMMIT_SHORT_SHA}  
❖ services:  
❖ - name:  
    docker:24.0.4-dind@sha256:594b3a4dc012e5160fed2a5c47a36ac5af6fa25f0c155bd9cf  
    f23d4f8c6fec16  
❖ alias: docker  
❖ # explicitly disable tls to avoid docker startup interruption as of  
    docker 20.10.9  
❖ # there is so security concern because docker is only accessed from  
    within the same system;  
❖ # there is no external network communication to worry about.  
❖ # See https://github.com/testcontainers/testcontainers-java/pull/4573  
❖ #command: ["--tls=false"]  
❖ #Needed to pass our Self-Signed TLS Certificate  
❖ command:  
❖ - /bin/sh  
❖ - -c  
❖ - echo "$CA_CERTIFICATE" > /usr/local/share/ca-certificates/my-ca.crt  
    && update-ca-certificates && dockerd-entrypoint.sh || exit  
❖ script:  
❖ - |  
❖ ssgdir="/ssg"  
❖ printf "\e[0Ksection_start:%s:docker[collapsed=true]\r\e[0KPulling the  
    docker image to scan...\n" "$(date +%s)"  
❖ # oscan-docker requires the image to have been pushed beforehand  
❖ docker login -u "${CI_REGISTRY_USER}" -p "${CI_REGISTRY_PASSWORD}"  
    "${CI_REGISTRY}"  
❖ docker pull "${IMAGE_NAME}"  
❖ printf "\e[0Ksection_end:%s:docker\r\e[0K\n" "$(date +%s)"
```

```
❖
❖     printf "\e[0Ksection_start:%s:scan\r\e[0KRunning openscap...\n"
"$$(date +%s)"
❖
❖     # extract /etc/os-release
❖     #Start a container from the Image
❖     container_id=$(docker create "${IMAGE_NAME}")
❖     #Location of os version inside the container, it is dynamic so we need
to filter the tar files, and keep the last row in case of a symbolic link
❖     os_release_path=$(docker export "${container_id}" | tar -tvf - |
grep '/os-release$' | tail -n 1 | awk '{print $NF}')
❖     echo "$os_release_path"
❖     #If the path is empty, abort with message
❖     if [ -z "$os_release_path" ] ; then
❖         >&2 echo "The operating system used by ${IMAGE_NAME} could not be
detected."
❖         >&2 echo "Images that are not based on an operating system (such as
distroless images) cannot be scanned by SCAP."
❖         exit 1
❖     fi
❖     #Copies Container/etc/os-release to our current directory
❖     echo "Copy the release"
❖     docker cp "$container_id:$os_release_path" .
❖     docker rm "$container_id"
❖     unset container_id
❖     # determine which ssg to use based on /etc/os-release
❖     echo "Compute ssg"
❖     # see https://www.freedesktop.org/software/systemd/man/os-release.html
❖     version_id=$(awk -F= '$1=="VERSION_ID" { print $2 ;}' os-release | sed
's/"//g')
❖     echo "$version_id"
❖     id=$(awk -F= '$1=="ID" { print $2 ;}' os-release | sed 's/"//g')
❖     echo "$id"
❖     cd /ssg
❖     #ls ssg
❖     echo $(find . -type d -name '*scap-security-guide*' -print -quit |
xargs basename)
❖     #Version of the xml our project will be scanned against
❖     SCAP_SECURITY_GUIDE_VERSION=$(find . -type d -name
'*scap-security-guide*' -print -quit | xargs basename)
❖     ssg="${SCAP_SECURITY_GUIDE_VERSION}/ssg-${id}${version_id//./}-ds.xml"
❖     #We need to correct some edge case, like alpine to match the xml name
```

```
❖ if [[ $ssg == *"pine"* ]]; then
❖   #echo -e "\e[31mHigh or /critical vulnerabilities detected. Please
    update the Dockerfile libraries\e[0m"
❖   echo -e "\e[30mAlpine is not currently supported, exiting
    successfully\e[0m"
❖   exit 0;
❖ fi
❖ echo "ssg is $ssg"
❖ # Select the profile to use. The first profile that exists in the ssg
    is used.
❖ for profile in xccdf_org.ssgproject.content_profile_cis_level2_server
    xccdf_org.ssgproject.content_profile_cis
    xccdf_org.ssgproject.content_profile_standard; do
❖   if oscap info --profiles "${ssgdir}/${ssg}" | grep -qF
    "${profile}:"; then
❖     echo "Selected profile: ${profile}"
❖     break;
❖   fi
❖ done
❖ cd /builds/aegean-apps/demonstration-apps/junit-app-2
❖ set +e
❖ echo "Using oscap-docker"
❖ oscap-docker image "${IMAGE_NAME}" xccdf eval --verbose ERROR
    --fetch-remote-resources --profile "${profile}" --results
    "openscap-report.xml" --report "openscap-report.html" "${ssgdir}/${ssg}"
❖ OSCAP_EXIT_CODE=$?
❖ set -e
❖
❖ echo "To view the openscap report:
    ${CI_JOB_URL}/artifacts/external_file/openscap-report.html"
❖
❖ case "${OSCAP_EXIT_CODE}" in
❖   0)
❖     echo "All rules passed"
❖   ;;
❖   1)
❖     >&2 echo "An error occurred during evaluation"
❖     exit 2
❖   ;;
❖   2)
❖     echo "There is at least one rule with either fail or unknown
    result"
```

```
❖      ;;
❖      *)
❖      >&2 echo "openscap returned an unexpected exit status of
$OSCAP_EXIT_CODE"
❖      exit "$OSCAP_EXIT_CODE"
❖      ;;
❖      esac
❖      printf "\e[0Ksection_end:%s:scan\r\e[0K\n" "$(date +%s)"
❖
❖      printf "\e[0Ksection_start:%s:xml_to_hdf\r\e[0KConverting xml to
hdf...\n" "$(date +%s)"
❖      saf convert xccdf_results2hdf -i "openscap-report.xml" -o
openscap-report.hdf
❖      printf "\e[0Ksection_end:%s:xml_to_hdf\r\e[0K\n" "$(date +%s)"
❖
❖      printf "\e[0Ksection_start:%s:hdf_to_sarif\r\e[0KConverting hdf to
sarif...\n" "$(date +%s)"
❖      DOTNET_SYSTEM_GLOBALIZATION_INVARIANT=1 sarif-multitool convert -t Hdf
-o openscap-report.sarif openscap-report.hdf.json
❖      printf "\e[0Ksection_end:%s:hdf_to_sarif\r\e[0K\n" "$(date +%s)"
❖
❖      printf "\e[0Ksection_start:%s:sarif_to_gitlab\r\e[0KConverting sarif
to GitLab SAST JSON...\n" "$(date +%s)"
❖      /sarif-converter --type sast openscap-report.sarif gl-sast-report.json
❖      printf "\e[0Ksection_end:%s:sarif_to_gitlab\r\e[0K\n" "$(date +%s)"
❖      artifacts:
❖      when: always
❖      paths:
❖      - "openscap-report.html"
```

Για τη συνεχή αναβάθμιση της έκδοσης του εργαλείου, αλλά κυρίως την επιτάχυνση της διαδικασίας ώστε οι τελευταίες ενημερώσεις της βάσης ευπαθειών να είναι ανά πάσα στιγμή διαθέσιμες πριν την εκτέλεση του αγωγού, το δοχείο που χρησιμοποιείται για αυτό το σκοπό παράγεται από ένα ξεχωριστό αποθετήριο και αγωγό, ο οποίος εκτελείται κάθε βράδυ.

Στη συγκεκριμένη ενέργεια εκτελείται αξιολόγηση **SCAP** (Security Content Automation Protocol) της εικόνας στον privileged Gitlab Runner με docker executor. Δυστυχώς, δεν εντοπίστηκε λύση που να εκτελείται σε απλό Gitlab Runner χωρίς ειδικά προνόμια.

Ακολουθεί μια σύνοψη της λειτουργικότητας του κώδικα. Αρχικά, θέτει μεταβλητές που απαιτούνται για την εκτέλεση Docker in Docker και αντιγράφει το πιστοποιητικό TLS του

αποθετηρίου εικόνων μέσα στο δοχείο ώστε να γίνει αποδεκτό κατά τη χειραψία. Στη συνέχεια η διεργασία Docker μέσα στο δοχείο αυθεντικοποιείται στο αποθετήριο εικόνων με τα διαπιστευτήρια μιας χρήσης που είδαμε και παραπάνω. Στη συνέχεια, ανιχνεύεται το είδος και η έκδοση του λειτουργικού συστήματος στο υπό εξέταση δοχείο, και βάση αυτού θα γίνει η αξιολόγηση **XCCDF** (eXtensible Configuration Checklist Description Format) με γνώμονα το προφίλ Standard System Security Profile. Δυστυχώς, διαπιστώθηκε πως το εργαλείο OpenSCAP¹⁶ δεν υποστηρίζει την αξιολόγηση του πολύ δημοφιλούς λειτουργικού Alpine Linux ακόμα. Σε κάθε στάδιο της διαδικασίας τυπώνονται φιλικά προς το χρήστη μηνύματα ώστε να είναι κατανοητή η ροή και τυχόν προβλήματα.

Η συγκεκριμένη ενέργεια επιτρέπεται να αποτύχει, ανεξάρτητα από τα ευρήματα αυτής, καθώς επιλέχθηκε να αυτά (τα ευρήματα) είναι συμβουλευτικά και όχι δεσμευτικά. Ως συνέπεια η εκτέλεση του αγωγού συνεχίζεται σε κάθε περίπτωση.

Η αναφορά που παράγεται openscap-report.html είναι διαθέσιμη για αξιολόγηση μέσα από το UI του αγωγού, ώστε εάν είναι επιθυμητό, ο προγραμματιστής να προβεί σε τροποποιήσεις του δοχείου.

5.3.1.7 Στάδιο: Υπογραφή εικόνας (Sign Image)

Σε αυτό το στάδιο πραγματοποιείται η υπογραφή της εικόνας στο αποθετήριο.

➤ Ενέργεια: Υπογραφή εικόνας (Cosign Image)

Αφού έχουν επιτύχει όλες οι προηγούμενες ενέργειες και η εικόνα βρίσκεται ήδη στο αποθετήριο, η συγκεκριμένη ενέργεια αξιοποιεί προσθέτει μια υπογραφή στην εικόνα με βάση το ιδιωτικό κλειδί που έχει εκδοθεί και βρίσκεται διαθέσιμο ως μυστικό στον αγωγό CI/CD. Η υπογραφή αυτή επιτρέπει, όπως θα δούμε σε επόμενη ενότητα, την εικόνα τελικά να εγκατασταθεί με επιτυχία στη συστοιχία Kubernetes.

Ο κώδικας είναι ο παρακάτω:

```
❖ #Scan the image with Cosign
❖ cosign-image:
❖ image: 192.168.1.101:5050/aegean-apps/helper-apps/cosigner:latest
❖ stage: sign-image
❖ allow_failure: false
```

¹⁶ <https://github.com/OpenSCAP>

```
❖ script:
❖   #Login to our Registry
❖   - cosign login -u "${CI_REGISTRY_USER}" -p ${CI_REGISTRY_PASSWORD}
    ${CI_REGISTRY}
❖   #Sign the image, do not send a record to online Rekor and trust our own
    registry
❖   - cosign sign --key env://COSIGN_PRIVATE_KEY
    --allow-insecure-registry=true "${CI_REGISTRY_IMAGE}":${CI_COMMIT_SHORT_SHA}
```

Η ενέργεια κάνει χρήση δοχείου με το εργαλείο **cosign** και ο κώδικας έχει την εξής λειτουργικότητα:

- Το **cosign** αυθεντικοποιείται στο αποθετήριο εικόνων κάνοντας χρήση των διαπιστευτηρίων μιας χρήσης
- Στη συνέχεια προσθέτει υπογραφή στην εικόνα που υποβλήθηκε στο αποθετήριο σε προηγούμενη ενέργεια κάνοντας χρήση του μυστικού ιδιωτικού κλειδιού **COSIGN_PRIVATE_KEY**, το οποίο φυλάσσεται στα μυστικά της πλατφόρμας Gitlab.

5.3.1.8 Στάδιο: Εγκατάσταση Δοχείων (Deploy Containers)

Σε αυτό το στάδιο γίνεται ο έλεγχος των σωστών ρυθμίσεων εγκατάστασης, και τελικά και η εγκατάσταση της υπογεγραμμένης εικόνας στη συστοιχία Kubernetes.

➤ Ενέργεια: Έλεγχος ρυθμίσεων εγκατάστασης δοχείων (Scan K3s Manifests)

Έχοντας ως δεδομένη μια προϋπάρχουσα συστοιχία Kubernetes στο δοκιμαστικό περιβάλλον, κατά την εκτέλεση αυτής της ενέργειας γίνεται χρήση δοχείου που πραγματοποιεί στατική ανάλυση ασφαλείας στα αρχεία του αποθετηρίου που χρησιμεύουν στην εγκατάσταση της εφαρμογής στο περιβάλλον αυτό, με δυνατότητα παραμετροποίησης των ήδη προεπιλεγμένων κανόνων του. Σε περίπτωση παραβίασης κάποιου κανόνα η εκτέλεση του αγωγού σταματά.

Ο κώδικας είναι ο παρακάτω:

```
❖ #Scans the k3s manifests for misconfigurations/vulnerabilities
❖ scan-k3s-manifests:
❖   stage: deploy-containers
❖   image:
❖     name: zegl/kube-score:latest
❖     #Override the entry point
❖     entrypoint: [""]
```



```
❖ allow_failure: false
❖   #Save results to a file named manifests-scan.txt
❖   #Sort the file, and colour the critical parts with red
❖   #Job will fail if a [CRITICAL] row is printed
❖ script:
❖   - |
❖       echo -e "\e[94mChecking K3s manifests..\e[0m"
❖       set +e
❖       kube-score score -o ci --ignore-test
❖       deployment-has-poddisruptionbudget ${MANIFESTS_LOCATION}/*.yaml >
❖       temp-scan.txt
❖
❖       grep "\[CRITICAL\]" temp-scan.txt | sed 's/\[CRITICAL\]/'$(printf
❖       '\033[31m')"'&'$(printf '\033[0m')"'/' > k3s-manifests-scan.txt
❖       grep "\[OK\]" temp-scan.txt >> k3s-manifests-scan.txt
❖       grep "\[SKIPPED\]" temp-scan.txt >> k3s-manifests-scan.txt
❖       echo -e "$(cat k3s-manifests-scan.txt)"
❖
❖       ! grep -q "CRITICAL" k3s-manifests-scan.txt
❖       KUBE_SCORE_EXIT_CODE=$?
❖       set -e
❖       case "${KUBE_SCORE_EXIT_CODE}" in
❖           0)
❖               echo -e "\e[92mK3s manifests are according to specs\e[0m"
❖               ;;
❖           1)
❖               echo -e "\e[31mHigh or critical misconfigurations detected. Please
❖       update the K3s manifests\e[0m"
❖       exit 1
❖       ;;
❖       esac
❖ artifacts:
❖   reports:
❖     container_scanning: k3s-manifests-scan.txt
❖     expire_in: 7 days
```

Κατά την εκτέλεση αυτής της ενέργειας γίνεται χρήση του εργαλείου **Kube-score** σε μορφή δοχείου. Η προεπιλεγμένη λειτουργία του Kube-score δεν ανταποκρινόταν απόλυτα στις απαιτήσεις του αγωγού, ήτοι να παράγεται αρχείο με τα αποτελέσματα και να αποθηκεύεται αυτό ώστε να είναι προσβάσιμο από τον προγραμματιστή, ως εκ τούτου ακολουθήθηκε μια

ημι-χειροκίνητη προσέγγιση. Εξ’ ορισμού, το Kube-score έχει ενεργοποιημένες πολλές απαιτήσεις ασφαλείας για την αξιολόγηση των αρχείων .yaml που καθορίζουν τον τρόπο εγκατάστασης μιας εφαρμογής σε μια συστοιχία Kubernetes. Ο κώδικας τυπώνει σε κάθε βήμα κατανοητά μηνύματα ώστε να είναι διαφανής η διαδικασία από το αρχείο καταγραφής της ενέργειας, και εκτελείται ως εξής:

- **set +e** : Απενεργοποιείται η άμεση έξοδος της ενέργειας σε περίπτωση που προκύψουν σφάλματα (Κωδικός 1)
- **kube-score score -o ci** : Εκτελείται η εντολή για αξιολόγηση των αρχείων yaml, των οποίων τα αποτελέσματα θα παράγονται σε μορφή συμβατή με εκτέλεση μέσα σε αγωγό
- **--ignore-test deployment-has-poddisruptionbudget** : Τροποποιούνται οι απαιτήσεις ώστε να μην απαιτείται ρύθμιση για Pod Disruption Budget, δηλαδή το ελάχιστο πλήθος δοχείων που απαιτούνται για την ομαλή παροχή υπηρεσιών λογισμικού, δεδομένων των περιορισμένων πόρων του συστήματος
- **k3s/*.yaml** : Το Kube-score θα ελέγξει όλα τα αρχεία τύπου yaml που βρίσκονται μέσα στον φάκελο k3s (παραμετροποιήσιμος από τον χρήστη) του αποθετηρίου της εφαρμογής.
- **> temp-scan.txt** : Το αρχείο καταγραφής θα εγγραφεί σε ένα προσωρινό αρχείο, το οποίο και θα τύχει περαιτέρω επεξεργασίας.
- Με χρήση των εντολών **grep** και **sed**, για λόγους αναγνωσιμότητας τοποθετούνται σε ένα νέο αρχείο **k3s-manifests-scan.txt** πρώτα τα σφάλματα, μετά οι έλεγχοι που παραλήφθηκαν και τέλος οι επιτυχείς έλεγχοι.
- **! grep -q "CRITICAL" k3s-manifests-scan.txt** : Ελέγχεται εάν στο νέο αρχείο που δημιουργήθηκε υπάρχει η λέξη **CRITICAL**
- **KUBE_SCORE_EXIT_CODE=\$?** : Το αποτέλεσμα της αναζήτησης αποθηκεύεται στη μεταβλητή **KUBE_SCORE_EXIT_CODE**.
- Στη συνέχεια, ανάλογα με την τιμή της μεταβλητής, τυπώνεται το ανάλογο μήνυμα επιτυχούς/ανεπιτυχούς ελέγχου και η ενέργεια τερματίζεται, μαζί με τον αγωγό σε περίπτωση σφαλμάτων.

Το αρχείο **k3s-manifests-scan.txt** είναι διαθέσιμο στον χρήστη μέσα από το UI του αγωγού.

➤ Ενέργεια: Εγκατάσταση στο K3s (Deploy K3s)

Αυτή η ενέργεια εκτελείται σε δοχείο που επικοινωνεί με τη συστοιχία Kubernetes, και αναλαμβάνει να εφαρμόσει τα κατάλληλα αρχεία (manifests) ώστε να εγκατασταθεί η νέα έκδοση της εφαρμογής στη συστοιχία αυτή. Σε περίπτωση αντικατάστασης δοχείων με νεότερη έκδοση, αυτή γίνεται σταδιακά.

Ο κώδικας είναι ο παρακάτω:

```
❖ #Deploys our image to the K3s Cluster
❖ deploy-k3s:
❖   stage: deploy-containers
❖   image:
❖     name: bitnami/kubectl:latest
❖     entrypoint: ['']
❖   script:
❖     # Set our context based on the yaml of the Connector project
❖     - kubectl config use-context
❖       aegean-apps/helper-apps/k3s-connector:k3s-connector
❖     #Apply our k3s manifests from k3s dir to deploy the application
❖     - echo "Deploying Application to the designated Cluster using these
❖       manifests:"
❖     #Apply our changes to the Cluster
❖     - kubectl apply -f $CI_PROJECT_DIR/${MANIFESTS_LOCATION}/.
❖     #Begin a rolling update of the containers with no downtime
❖     - kubectl rollout restart deploy junit-app-2
```

Αυτή η ενέργεια εκτελείται σε δοχείο που περιέχει το εργαλείο **Kubectl** για την επικοινωνία με μια συστοιχία Kubernetes. Ο κώδικας εκτελεί τις εξής εντολές:

- - **kubectl config use-context aegean-apps/helper-apps/k3s-connector:k3s-connector** : Με αυτή την εντολή ορίζεται ότι θα χρησιμοποιηθεί το αποθετήριο-πράκτορας (Agent aegean-apps/helper-apps/k3s-connector) που έχει δημιουργηθεί για το σκοπό της σύνδεσης του αγωγού CI/CD με τη συστοιχία Kubernetes, όπως αυτός περιγράφηκε στην Ενότητα 5.2.4.2.
- - **kubectl apply -f \$CI_PROJECT_DIR/k3s/** : Με την εντολή **apply** εφαρμόζονται όλα τα αρχεία **yaml**, ώστε να εισαχθούν/αντικατασταθούν στη συστοιχία K8s οι ρυθμίσεις εγκατάστασης.

- - **kubectrl rollout restart deploy junit-app-2** : Εντολή για σταδιακή αντικατάσταση τυχόν ήδη λειτουργικών δοχείων με τη νέα έκδοση.

5.3.1.9 Χειροκίνητο Στάδιο: Δόμηση προφίλ SECCOMP (Seccomp profiling)

Σε αυτό το χειροκίνητο στάδιο στάδιο διαμορφώνεται η βάση για ένα προφίλ **Seccomp** για το υπό διαμόρφωση δοχείο.

➤ Χειροκίνητη Ενέργεια: Δόμηση προφίλ SECCOMP (Build Seccomp Profile)

Αυτή η ενέργεια εκτελείται στον προνομιούχο (privileged) Gitlab Runner με shell executor, προκειμένου, για να δημιουργήσει ένα σύνολο-βάση με τις ελάχιστες δυνατές κλήσεις syscalls που απαιτεί το δοχείο. Αυτές οι κλήσεις καταγράφονται σε ένα προφίλ seccomp [86], και δεν επιτρέπουν σε ένα δοχείο να χρησιμοποιεί άλλες που δεν περιέχονται σε αυτό, εφόσον αυτό εφαρμοστεί από τον ενορχηστρωτή. Αυτή η ενέργεια είναι εκτός της κανονικής ροής καθώς ενεργοποιείται με χειροκίνητη ενέργεια του χρήστη. Επίσης, αν και παράγει ένα προφίλ seccomp, αυτό δεν θα είναι πλήρες εάν δε γίνουν οι απαραίτητες δοκιμές όλων των πιθανών σεναρίων χρήσης από έναν χρήστη κατά την εκτέλεση του δοχείου.

Ο κώδικας είναι ο παρακάτω:

```
❖ #Makes a base Seccomp profile by having the container run
❖ #User can download the json in order to incorporate it into the k3s
  Deployment Security context
❖ # Needs to placed in the Master node at : /var/lib/kubelet and then declared
  in the Deployment.yaml
❖ #The profile can be extended by manual testing and logging of the SysCalls
❖ seccomp-profiling:
❖   stage: build-seccomp-profile
❖   #Only the privileged Seccomp Shell Runner will execute this
❖   tags:
❖     - seccomp
❖   when: manual
❖   script:
❖     #Imports our tar image into a proper OCI image and can then run it
❖     #- podman import
❖     # Decode password to from Base 64
❖     - echo ${SECCOMP_PASSWORD} | base64 -d | sudo -S su -
❖     - sudo podman login -u $CI_REGISTRY_USER -p $CI_REGISTRY_PASSWORD
    $CI_REGISTRY
```

- ❖ - `sudo podman pull ${CI_REGISTRY_IMAGE}:latest`
- ❖ - `sudo rm -rf /home/gitlab-runner/seccomp-profiles`
- ❖ - `sudo mkdir /home/gitlab-runner/seccomp-profiles`
- ❖ *#- sudo podman run --annotation*
io.containers.trace-syscall=of:/home/gitlab-runner/seccomp-profiles/junit-app-2.json fedora:35 ls / > /dev/null
- ❖ *#This commands needs an absolute path-not a variable- for the output json, while the job has not visibility outside the CI_PROJECT_DIR*
- ❖ - `sudo podman run --timeout 25 -d --annotation`
`io.containers.trace-syscall=of:/home/gitlab-runner/builds/PyQSdxVVt/0/aegean-apps/demonstration-apps/junit-app-2/SECCOMP-junit-app-2.json`
`192.168.1.101:5050/aegean-apps/demonstration-apps/junit-app-2 java -jar target/java-maven-junit-2-1.0-SNAPSHOT.jar`
- ❖ *# Sleep for 20 seconds*
- ❖ - `sleep 25`
- ❖ `artifacts:`
- ❖ `paths:`
- ❖ - `SECCOMP-junit-app-2.json`

Αυτή η ενέργεια εκτελείται στον privileged Gitlab Runner με shell executor, προκειμένου, χρησιμοποιώντας το εργαλείο **podman** σε συνδυασμό με το **oci-seccomp-bpf-hook**, ώστε να είναι σε θέση να δημιουργήσει ένα αρχείο με τις ελάχιστες δυνατές κλήσεις syscalls που απαιτεί το δοχείο. Σε κάθε περίπτωση, τα καταγεγραμμένα syscalls αποτελούν μόνο μια βάση, καθώς μετά τη λειτουργία που περιγράφεται στον κώδικα, θα είναι απαραίτητη η κανονική δοκιμή όλων των λειτουργιών της υπό δημιουργία εφαρμογής από χρήστες, με σκοπό να καταγραφούν όλες οι κλήσεις που απαιτούνται για όλες τις περιπτώσεις χρήσης του δοχείου.

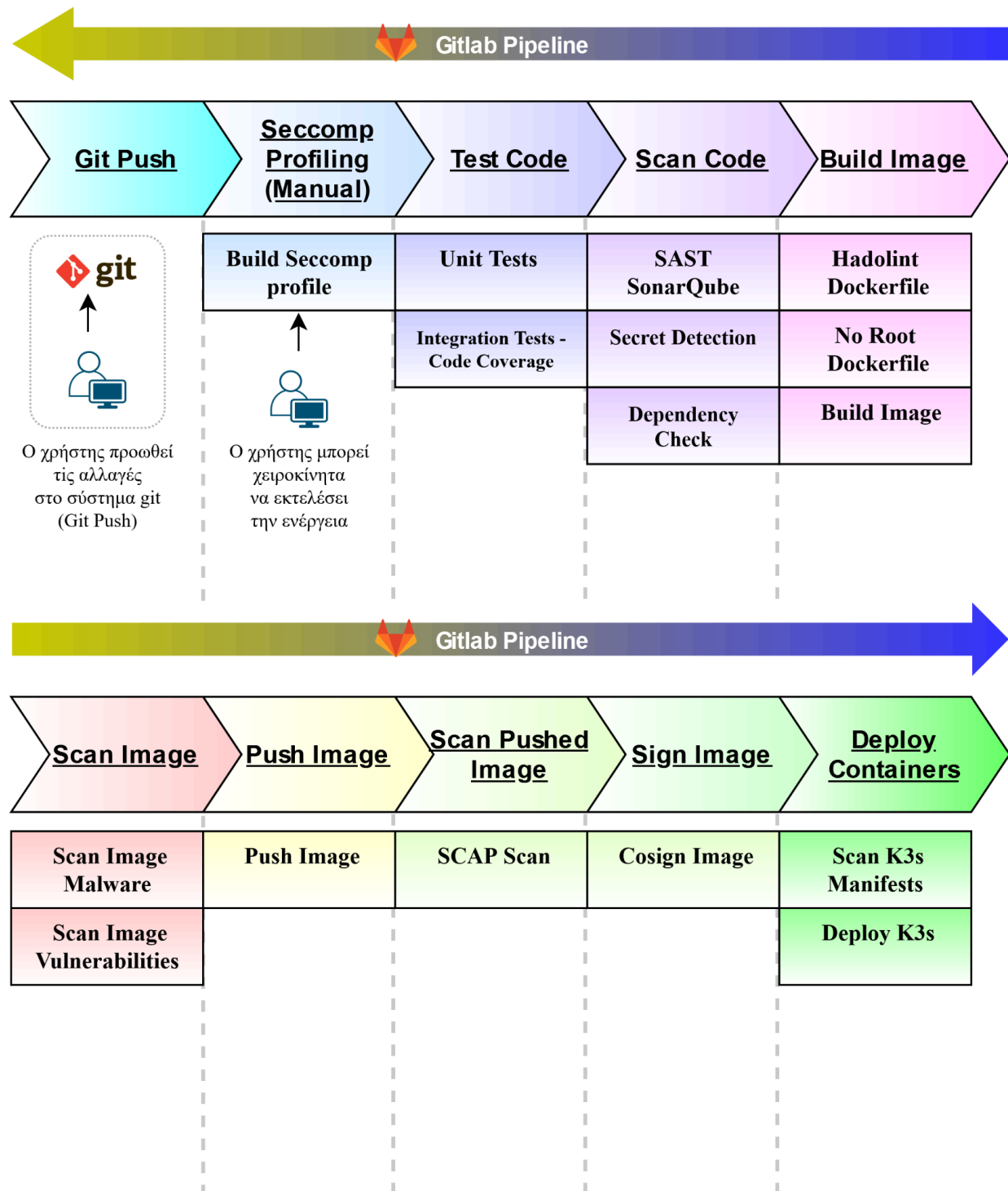
Ο κώδικας λειτουργεί ως εξής:

- **Tags: seccomp** : Εκτελείται σε εικονικό μηχάνημα με privileged shell executor. Για παραγωγικά συστήματα, αυτό το εικονικό μηχάνημα πρέπει να είναι εφήμερο και να αναδημιουργείται κατ'απαίτηση, λόγω προβληματισμών ασφαλείας.
- **when: manual** : Δηλώνει ότι αυτή η ενέργεια εκτελείται όχι αυτοματοποιημένα, αλλά κατ'απαίτηση του χρήστη.
- - `echo ${SECCOMP_PASSWORD} | base64 -d | sudo -S su` : Για τους σκοπούς της καταγραφής των syscalls, είναι απαραίτητο το δοχείο να εκτελεστεί ως χρήστης root. Με την εντολή αυτή, ο χρήστης αναβαθμίζεται σε root.

- - **sudo podman login -u \$SCI_REGISTRY_USER -p \$SCI_REGISTRY_PASSWORD \$SCI_REGISTRY** : Το εργαλείο podman αυθεντικοποιείται στο αποθετήριο εικόνων με τα διαπιστευτήρια μιας χρήσης.
- - **sudo podman pull \${CI_REGISTRY_IMAGE}:\${COMMIT_SHORT_SHA}** : Γίνεται λήψη της τελευταίας της εικόνας του αποθετηρίου που αποθηκεύτηκε από τον αγωγό.
- - **sudo rm -rf /home/gitlab-runner/seccomp-profiles**
- **sudo mkdir /home/gitlab-runner/seccomp-profiles** : Αναδημιουργούνται τα ευρετήρια για να εξαλειφθούν τυχόν κατάλοιπα.
- - **sudo podman run --timeout 25 -d --annotation io.containers.trace-syscall=of:/home/gitlab-runner/builds/PyQSdxVVt/0/aegean-apps/demonstration-apps/junit-app-2/SECCOMP-junit-app-2.json 192.168.1.101:5050/aegean-apps/demonstration-apps/junit-app-2 java -jar target/java-maven-junit-2-1.0-SNAPSHOT.jar** : Εκτελείται το δοχείο της εφαρμογής μας με ενισχυμένα δικαιώματα ώστε να καταγραφούν τα syscalls με τη βοήθεια του εργαλείου oci-seccomp-bpf-hook, και να αποθηκευτούν στο αρχείο **SECCOMP-junit-app-2.json**. Δυστυχώς το εργαλείο δεν επιτρέπει την αποθήκευση σε οποιοδήποτε φάκελο, με συνέπεια η χρήση του ανωτέρω φακέλου να είναι δεσμευτική.
- - **sleep 25** : Το δοχείο σταματά μετά από 25 δευτερόλεπτα

Το αρχείο SECCOMP-junit-app-2.json είναι διαθέσιμο στον χρήστη μέσα από το UI του αγωγού, ώστε να σταλεί στον διαχειριστή της συστοιχίας Kubernetes προς εγκατάσταση.

Συνολικά, τα στάδια και οι ενέργειες του αγωγού εμφανίζονται στην Εικόνα 24.



Εικόνα 24 - Αγωγός CI/CD

5.3.2 Υλοποίηση υλικού, υπερεπόπτη και λειτουργικών συστημάτων

Αρχικά, πρέπει να αναφερθεί ότι για την υλοποίηση του εν θέματι συστήματος διαμορφώθηκε ένα αυτοσχέδιο testbed, αποτελούμενο από 2 παλαιούς επιτραπέζιους

υπολογιστές με επεξεργαστές i5-6500 και συνολικά 24Gb μνήμης RAM.

Για τη δημιουργία και διαχείριση των εικονικών μηχανών εγκαταστάθηκε και χρησιμοποιήθηκε ο υπερεπόπτης Proxmox¹⁷. Το Proxmox Virtual Environment (Proxmox VE) είναι μια πλατφόρμα εικονικοποίησης που σχεδιάστηκε για να παρέχει λύσεις εικονικοποίησης και διαχείρισης εικονικών μηχανών. Αναπτύσσεται από την εταιρεία Proxmox Server Solutions GmbH και βασίζεται στο Debian GNU/Linux. Το Proxmox VE συνδυάζει δύο κύριες τεχνολογίες εικονικοποίησης, το KVM (Kernel-based Virtual Machine) για εικονικές μηχανές και το LXC (Linux Containers) για ελαφριά εικονικά περιβάλλοντα.

Με το Proxmox VE, ο χρήστης μπορεί να διαχειριστεί εύκολα και να δημιουργήσει εικονικές μηχανές, να τις εξάγει σαν πρότυπο για να επαναχρησιμοποιηθούν για τη δημιουργία νέων μηχανών, να παρακολουθεί την απόδοση του συστήματος, και να κρατά αντίγραφα ασφαλείας. Επιπλέον, παρέχει ένα περιβάλλον χρήστη σε μορφή ιστοσελίδας, καθιστώντας τον έλεγχο των εικονικών μηχανών και των πόρων εύκολο και αποτελεσματικό, όπως φαίνεται και στην Εικόνα 25.



Εικόνα 25 - Περιβάλλον χρήστη Proxmox

5.3.2.1 Πρόσβαση μέσω ssh

Η πρόσβαση στα εικονικά μηχανήματα μέσω ssh επιτράπη μόνο μέσω προκαθορισμένων κλειδιών και όχι μέσω κωδικού, ενώ το εγγενές ανάχωμα ασφαλείας του υπερεπόπτη επιτρέπει τη χρήση της πόρτας 22 του πρωτοκόλλου ssh μόνο από επιλεγμένες IP. (Τον προσωπικό υπολογιστή

¹⁷ <https://www.proxmox.com/en/>

του συγγραφέα) Αυτή η λειτουργία θα μπορούσε να επιτρέψει την πρόσβαση μόνο σε ένα εν δυνάμει εταιρικό δίκτυο.. Ενδεικτικά παρατίθεται η Εικόνα 26.

On	Type	Action	Macro	Interface	Protocol	Source	S.Port	Destination	D.Port
☒	in	DROP			tcp				22
☒	in	ACCEPT			tcp	192.168.1.10			22

Εικόνα 26 - Κανόνες πρόσβασης ssh

5.3.2.2 Αξιολόγηση ασφαλείας με το εργαλείο Lynis

Το Lynis¹⁸ είναι ένα εργαλείο αξιολόγησης ασφάλειας για συστήματα όπως Linux, macOS, BSD κ.α. Πραγματοποιεί μια εκτενή ανάλυση ασφαλείας και εκτελείται στο ίδιο το σύστημα. Ο κύριος στόχος είναι να δοκιμάσει τα αντίμετρα ασφαλείας και να παρέχει συμβουλές για περαιτέρω σκλήρυνση του συστήματος. Αναζητά πληροφορίες γενικά για το σύστημα, τυχόν ευάλωτα πακέτα λογισμικού και πιθανές άστοχες ρυθμίσεις. Μετά την εγκατάσταση και την εκτέλεση, ο πίνακας των αποτελεσμάτων στο επιλεγμένο λειτουργικό σύστημα CentOS Stream 9, μας παρότρυνε να εγκαταστήσουμε ένα πρόγραμμα ανίχνευσης κακόβουλου λογισμικού, όπως φαίνεται και στην Εικόνα 27.

```
Lynis security scan details:

Hardening index : 70 [##### ]
Tests performed : 208
Plugins enabled : 0

Components:
- Firewall [V]
- Malware scanner [X]

Lynis Modules:
- Compliance Status [?]
- Security Audit [V]
- Vulnerability Scan [V]

Files:
- Test and debug information : /var/log/lynis.log
- Report data : /var/log/lynis-report.dat
```

Εικόνα 27 - Αποτελέσματα 1ου ελέγχου του εργαλείου Lynis

Μετά την εγκατάσταση του εργαλείου ανίχνευσης κακόβουλου λογισμικού ClamAV, τα αποτελέσματα ενημερώθηκαν όπως φαίνονται στην Εικόνα 28.

¹⁸ <https://github.com/CISOfy/lynis>

```
Lynis security scan details:

Hardening index : 82 [#####          ]
Tests performed : 208
Plugins enabled : 0

Components:
- Firewall          [V]
- Malware scanner   [V]

Lynis Modules:
- Compliance Status [?]
- Security Audit    [V]
- Vulnerability Scan [V]

Files:
- Test and debug information : /var/log/lynis.log
- Report data                : /var/log/lynis-report.dat
```

Εικόνα 28 - Αποτελέσματα 2ου ελέγχου του εργαλείου Lynis

5.3.2.3 Εγκατάσταση ανιχνευτή κακόβουλου λογισμικού ClamAV

Όπως ήδη περιγράφηκε στην παράγραφο 5.2.3, για να ενισχυθεί η ασφάλεια του λειτουργικού συστήματος που θα χρησιμοποιηθεί εγκαταστάθηκε το γνωστό λογισμικό ανίχνευσης κακόβουλου λογισμικού ClamAV¹⁹. Για να παρέχεται συνεχώς ένα επαρκές επίπεδο προστασίας, δημιουργήθηκαν cron jobs, τα οποία εκτελούνται κάθε βράδυ στις 03:00, με τα οποία ενημερώνεται η βάση δεδομένων για κακόβουλες υπογραφές και εκτελείται πλήρης σάρωση του συστήματος. Κάθε φορά που εκτελείται η σάρωση, στο αρχείο καταγραφής προστίθεται μια καταχώρηση, όπως αυτή που εμφανίζεται στην Εικόνα 29.

```
----- SCAN SUMMARY -----
Known viruses: 8560185
Engine version: 0.103.2
Scanned directories: 204
Scanned files: 4340
Infected files: 0
Data scanned: 378.32 MB
Data read: 231.95 MB (ratio 1.63:1)
Time: 62.941 sec (1 m 2 s)
```

Εικόνα 29 - Αποτελέσματα ημερήσιας σάρωσης με το εργαλείο ClamAV

¹⁹ <https://github.com/Cisco-Talos/clamav>

5.3.2.4 Χρήση αναχώματος ασφαλείας (Firewall) και συστήματος πρόληψης εισβολής (Intrusion Prevention System)

Όλες οι θύρες των εικονικών μηχανών που δεν απαιτούνται για χρήση κάποιας λειτουργίας παραμένουν αποκλεισμένες από το εγγενές ανάχωμα ασφαλείας firewalld. Πέραν αυτού, γίνεται επιλέον χρήση του λογισμικού πρόληψης εισβολής και ανίχνευσης συμπεριφοράς CrowdSec²⁰, το οποίο εγκαθίσταται στο επίπεδο της εικονικής μηχανής, και του οποίου οι κύριες λειτουργίες είναι οι εξής:

- **Ανίχνευση Απειλών:** Το CrowdSec αναλύει την κίνηση στο δίκτυο για να ανιχνεύσει επιθέσεις, όπως προσπάθειες ανεπιτυχούς σύνδεσης, DDoS επιθέσεις, ή άλλες μορφές κακόβουλης δραστηριότητας.
- **Αυτοματοποιημένο Σύστημα Αποκλεισμού:** Με βάση τις απειλές που ανιχνεύει, το CrowdSec μπορεί να αποκλείει αυτόματα τους κακόβουλους χρήστες ή τις κακόβουλες IP διευθύνσεις από την πρόσβαση στο σύστημα. Η ενημέρωση της βάσης των κακόβουλων διευθύνσεων γίνεται αυτόματα, ενώ παράλληλα δεν προκύπτει σύγκρουση με το λογισμικό ανίχνευσης κακόβουλου λογισμικού.
- **Συλλογή Δεδομένων και Κοινότητα:** Το CrowdSec συλλέγει δεδομένα σχετικά με απειλές από χρήστες που τρέχουν το σύστημα παγκόσμια, και αυτά τα δεδομένα χρησιμοποιούνται για τη βελτίωση της ανίχνευσης και τη δημιουργία μιας κοινότητας ασφαλείας.

5.3.3 Υλοποίηση της Αρχιτεκτονικής του Συστήματος

Όπως ήδη αναφέρθηκε στην ενότητα 3.2.5, η τρέχουσα Διπλωματική εργασία αξιοποίησε τις δυνατότητες της πλατφόρμας ανοικτού κώδικα CI/CD Gitlab Community Edition, ακολουθώντας τις καλύτερες πρακτικές για τη σκλήρυνση κάθε τμήματος αυτής. Συγκεκριμένα, τηρήθηκε η αρχή του ελάχιστου δικαιώματος, χρησιμοποιώντας παράλληλα ρυθμίσεις ή εργαλεία που ελαχιστοποιούν την επιφάνεια επίθεσης.

5.3.3.1 Αρχιτεκτονική του συστήματος Gitlab

Η εγκατάσταση της κύριας πλατφόρμας πραγματοποιήθηκε κάνοντας χρήση της τεχνολογίας δοχείων, εγκαθιστώντας την σε περιβάλλον podman²¹ με χρήση ενός αρχείου

²⁰ <https://github.com/crowdsecurity/crowdsec>

²¹ <https://podman.io/>

docker-compose.yaml. Για την εγκατάσταση προτιμήθηκε η χρήση του εργαλείου podman για τους παρακάτω λόγους:

1. Δεν απαιτεί τη χρήση Daemon:

Η σημαντικότερη διαφορά στο επίπεδο ασφαλείας ανάμεσα στο διαδεδομένο εργαλείο Docker και το Podman είναι ότι το πρώτο χρησιμοποιεί έναν δαίμονα (Daemon) με αυξημένα δικαιώματα για τη δημιουργία και διαχείριση δοχείων. Αντίθετα, το εργαλείο podman εκτελείται ως διεργασία χρήστη, μειώνοντας με αυτόν τον τρόπο την διαθέσιμη επιφάνεια επίθεσης.

2. Rootless Containers:

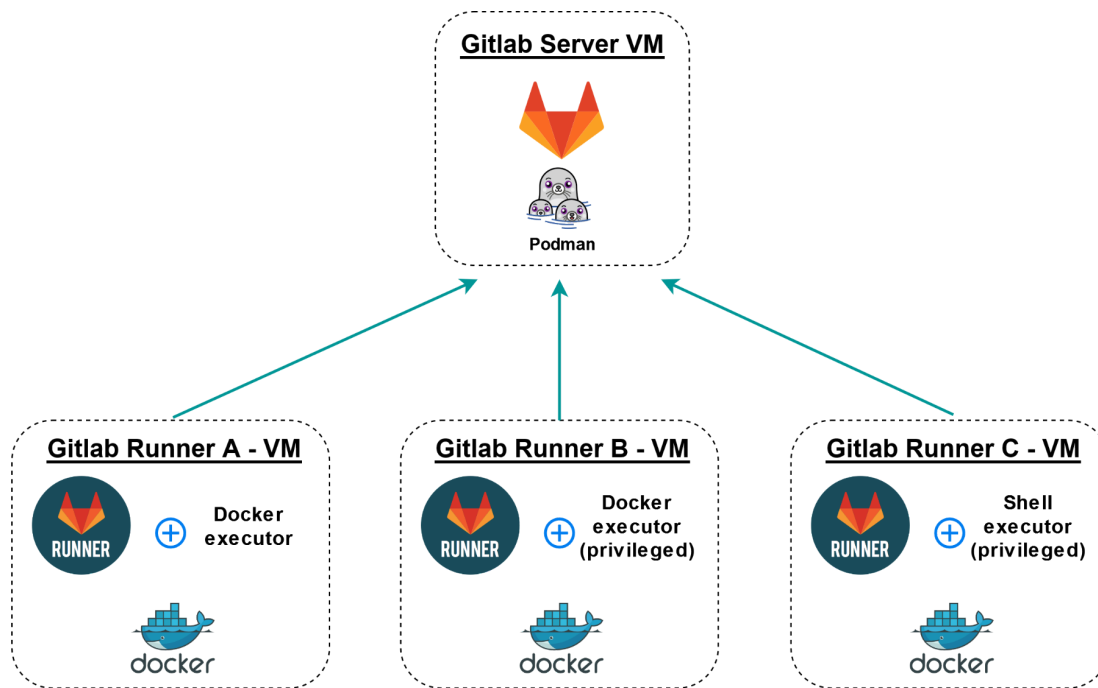
Το εργαλείο podman υποστηρίζει - χωρίς καμία τροποποίηση - δοχεία που εκτελούνται από απλούς χρήστες του λειτουργικού συστήματος χωρίς την ανάγκη για αυξημένα δικαιώματα και χωρίς να ανήκει σε συγκεκριμένη ομάδα χρηστών. Αυτό αυτόματα μειώνει τη διαθέσιμη επιφάνεια επίθεσης. Τελευταία και το εργαλείο Docker έχει ενσωματώσει τη δυνατότητα για δοχεία από απλούς χρήστες, με κάποιες τροποποιήσεις να απαιτούνται προς το σκοπό αυτό.

Πέραν της εγκατάστασης της κεντρικής εφαρμογής Gitlab, για την εκτέλεση των αγωγών απαιτούνται μια ή περισσότερες εγκαταστάσεις της βοηθητικής, δοχειοποιημένης εφαρμογής Gitlab Runner²². Η εφαρμογή Gitlab Runner εγκαταστάθηκε σε 3 εικονικά μηχανήματα, πάντα με τη μορφή δοχείου. Τα στιγμιότυπα εγκαθίστανται σε κάποιο σύστημα (εικονικό μηχάνημα με δικτυακή πρόσβαση στην πλατφόρμα Gitlab) και συνεχώς ελέγχουν εάν υπάρχουν εκκρεμείς εντολές (στάδια ή ενέργειες/βήματα αγωγών) προς εκτέλεση. Συνεπώς, ανάλογα με το περιβάλλον που προσομοιώνεται από τον Gitlab Runner (Linux, Windows, Εντολές μέσα σε δοχεία κ.α), απαιτούνται και οι ανάλογες εντολές.

Κατά το σχεδιασμό του συστήματος καταβλήθηκε η μέγιστη προσπάθεια να εκτελούνται οι Runners με κατά το δυνατόν λιγότερες δυνατότητες, αποφεύγοντας τη χρήση της σημαίας privileged, για τους λόγους που είδαμε στην Ενότητα 4.2.4. Ωστόσο κάποια τμήματα του αγωγού, ο οποίος διαμορφώθηκε εξ'ολοκλήρου στο πλαίσιο της διπλωματικής εργασίας, όπως είναι ο έλεγχος SCAP και η δημιουργία προφίλ SECCOMP, απαιτούσαν την εκτέλεση από privileged runner, χωρίς να κατορθωθεί να αποφευχθεί αυτό. Δημιουργήθηκαν τρεις runners στο περιβάλλον ανάπτυξης, 2 με Docker executor και ένας με Shell Executor, ο καθένας σε ξεχωριστή εικονική

²² <https://docs.gitlab.com/runner/>

μηχανή για καλύτερη απομόνωση, μαζί με ένα στιγμιότυπο της κεντρικής εφαρμογής του Gitlab (Gitlab Server) όπως φαίνεται στην παρακάτω Εικόνα 30.



Εικόνα 30 - Διάγραμμα φυσικής διάταξης Πλατφόρμας Gitlab & Gitlab Runners

5.3.3.2 Αρχιτεκτονική της συστοιχίας Kubernetes & Διασύνδεση με αγωγό CI/CD

5.3.3.2.1 Γενικά

Για την αυτοματοποιημένη εγκατάσταση των παραγόμενων δοχείων, και λόγω σημαντικού περιορισμού στους διαθέσιμους πόρους, ως τεχνολογία ενορχήστρωσης (δοχείων) επιλέχθηκε το λογισμικό K3s²³. Το K3s είναι ένα ελαφρύ και απλοποιημένο σύστημα διαχείρισης δοχείων που βασίζεται στο Kubernetes. Σχεδιάστηκε για να είναι ελαφρύ και ευέλικτο, καθιστώντας το κατάλληλο για χρήση σε περιβάλλοντα με περιορισμένους πόρους, όπως συστήματα edge (στις παρυφές του δικτύου) και IoT (Internet of Things - Διαδικτύου των Πραγμάτων). Το K3s προσφέρει τη λειτουργικότητα του Kubernetes, αλλά με μειωμένες απαιτήσεις σε υπολογιστικούς πόρους και χώρο αποθήκευσης, καθιστώντας το ευκολότερο στην εγκατάσταση, διαχείριση και συντήρηση. Τα σημαντικότερα χαρακτηριστικά του μπορούν να συνοψιστούν ως εξής:

- **Ελαφρύ:** Το K3s σχεδιάστηκε να είναι ελαφρύ και αποτελεσματικό ως προς τους πόρους, καθιστώντας το κατάλληλο για περιβάλλοντα με περιορισμένους πόρους.

²³ <https://k3s.io/>

- **Απλοποιημένη Εγκατάσταση:** Το K3s παρέχει μια απλοποιημένη διαδικασία εγκατάστασης, μειώνοντας την πολυπλοκότητα που συνήθως σχετίζεται με τη δημιουργία μια συστοιχίας Kubernetes.
- **Ελάχιστες Εξαρτήσεις:** Έχει ελάχιστες εξαρτήσεις, επιταχύνοντας με αυτόν τον τρόπο την εγκατάσταση.
- **Υψηλή Φορητότητα:** Λόγω του ελαφρού του χαρακτήρα, το K3s μπορεί να τρέξει σε διάφορες αρχιτεκτονικές, γεγονός που το καθιστά ευέλικτο για διάφορα σενάρια ανάπτυξης.
- **Ασφάλεια από Προεπιλογή:** Συμπεριλαμβάνει χαρακτηριστικά ασφαλείας από προεπιλογή, εξασφαλίζοντας ότι η συστοιχία έχει ρυθμιστεί με βάση τις βέλτιστες πρακτικές ασφαλείας.
- **Οριζόντια επέκταση:** Παρέχει δυνατότητες επέκτασης, ενώ παραμένει ελαφρύ, επιτρέποντας στους χρήστες να επεκτείνουν τις εφαρμογές τους σύμφωνα με τις ανάγκες τους.

5.3.3.2.2 Αρχιτεκτονική

Λόγω υψηλών περιορισμών σε υπολογιστικούς πόρους, υλοποιήθηκε μια αρχιτεκτονική με μόνο ένα Master node (κόμβος) και δυο Worker nodes. Στο πλαίσιο του οικοσυστήματος Kubernetes, ένας "worker node" (κόμβος - εργάτης) και ένας "master node" (κόμβος διαχείρισης - κύριος κόμβος) είναι δύο βασικοί τύποι κόμβων που συνεργάζονται για τη λειτουργία μιας συστοιχίας. Για αυτές τις βασικές έννοιες μπορούμε να αναφέρουμε τα εξής:

- **Κόμβος Διαχείρισης (Master Node):** Ο κόμβος διαχείρισης είναι υπεύθυνος για τον έλεγχο και τη διαχείριση της συστοιχίας Kubernetes. Σε αυτόν τον κόμβο τρέχουν βασικές υπηρεσίες, όπως ο "kube-apiserver" που διαχειρίζεται τα API του Kubernetes, ο "kube-controller-manager" που χειρίζεται τους controllers (ελεγκτές), ο "kube-scheduler" που επιλέγει σε ποιον worker node θα τρέξει μια εργασία, καθώς και άλλα συστατικά που συντονίζουν τη λειτουργία της συστοιχίας. Επιπλέον, ο κόμβος διαχείρισης διασφαλίζει τη συνοχή της κατάστασης του συστήματος.
- **Κόμβος - εργάτης (Worker Node):** Ο κόμβος - εργάτης είναι υπεύθυνος για την εκτέλεση των εφαρμογών και των εργασιών που ανατίθενται στη συστοιχία Kubernetes. Σε αυτόν τον κόμβο τρέχουν τα δοχεία των εφαρμογών, και αυτός είναι υπεύθυνος για τη διαχείριση

των πόρων του όπως CPU, μνήμη και αποθηκευτικός χώρος, και τη δέσμευσή τους στα δοχεία που φιλοξενεί.

Για την επικοινωνία ανάμεσα στη συστοιχία Kubernetes και τον αγωγό CI/CD χρησιμοποιήθηκε το ενδιάμεσο λογισμικό Gitlab Agent για Kubernetes.

Το GitLab Agent για Kubernetes είναι ένα εργαλείο που ανήκει στο οικοσύστημα GitLab και χρησιμοποιείται για τον αυτοματισμό και την διαχείριση εγκαταστάσεων σε περιβάλλοντα Kubernetes. Ο πράκτορας είναι σχεδιασμένος για να λειτουργεί ως ενδιάμεσος μεταξύ του αγωγού CI/CD και της συστοιχίας Kubernetes.

Ουσιαστικά, ο GitLab Agent επιτρέπει την εκτέλεση εγκαταστάσεων δοχείων σε περιβάλλοντα Kubernetes, διευκολύνοντας τη διαδικασία παράδοσης λογισμικού. Με τη χρήση του GitLab Agent, ο οποίος καλείται κατά την ενέργεια της εγκατάστασης δοχείων και παρέχει την ασφαλή σύνδεση με τη συστοιχία, αυτοματοποιείται η διαχείριση και παράδοση λογισμικού.

Η διαδικασία εγκατάστασης του GitLab Agent για Kubernetes είναι σχετικά απλή και περιλαμβάνει τα εξής βήματα:

Στην πλατφόρμα Gitlab: Πρέπει να δημιουργηθεί ένα νέο αποθετήριο για τον πράκτορα, στο οποίο θα δημιουργηθεί ένα token για τη σύνδεση με τη συστοιχία Kubernetes.

Στη συστοιχία Kubernetes: Ο GitLab Agent εγκαθίσταται με τη βοήθεια του εργαλείου helm²⁴, κάνοντας χρήση του παραπάνω token.

Οι κύριες λειτουργίες του GitLab Agent για Kubernetes είναι οι εξής:

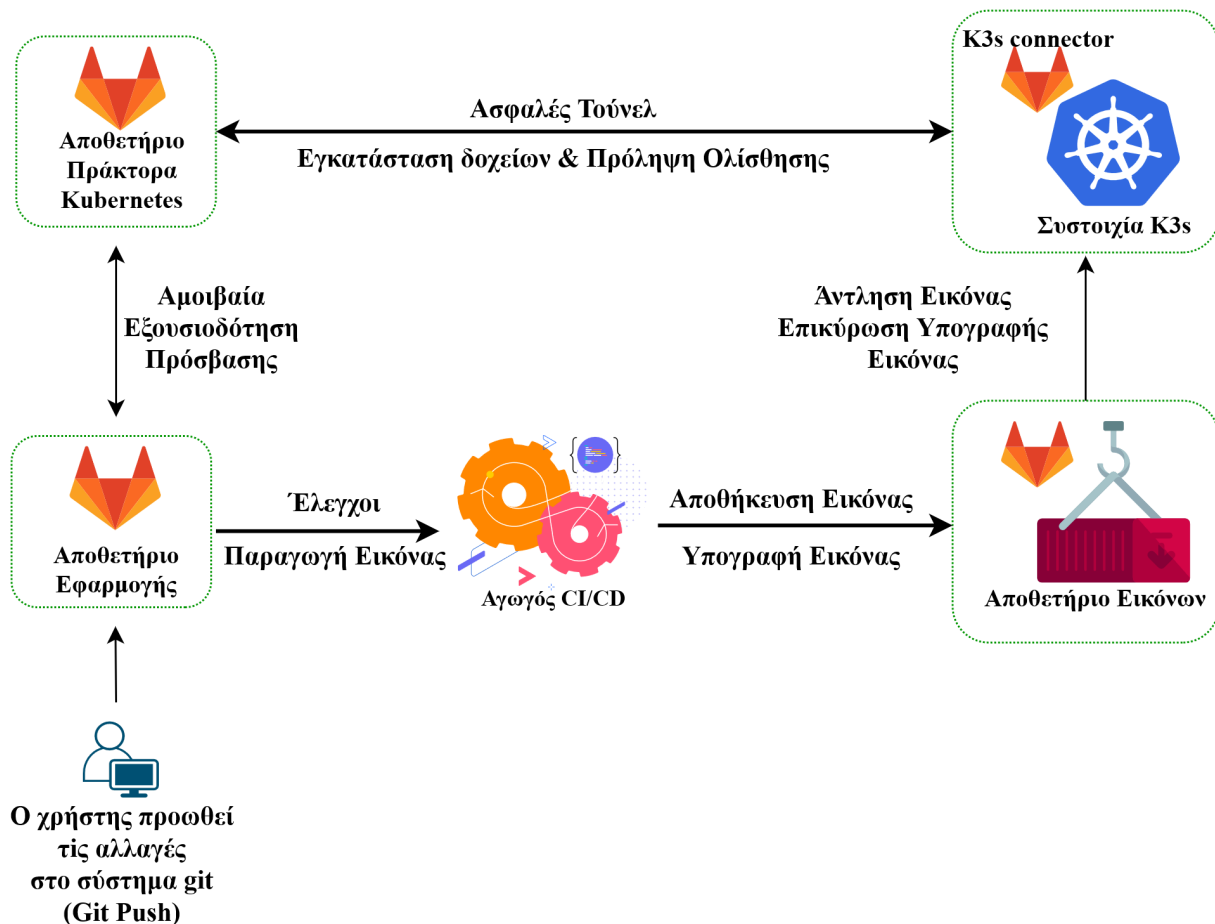
Εκτέλεση εγκατάστασης δοχείων σε Kubernetes: Ο πράκτορας δίνει τη δυνατότητα για την αυτοματοποίηση της διαδικασίας CD σε περιβάλλοντα Kubernetes.

Πρόληψη Ολίσθησης (Drift Prevention): Ο πράκτορας παρέχει πληροφορίες στον αγωγό σχετικά με την κατάσταση εκτέλεσης των δοχείων στη συστοιχία Kubernetes, βοηθώντας στον έλεγχο της ποιότητας και της εναρμόνισης των δοχείων με τις προδιαγραφές που θα έπρεπε να τηρούν.

²⁴ <https://github.com/helm/helm>

Συνεργασία με GitLab CI/CD: Ο GitLab Agent ενσωματώνεται στο οικοσύστημα CI/CD του GitLab, προσφέροντας ομαλή συνεργασία με τα άλλα εργαλεία και υπηρεσίες της πλατφόρμας.

Στην Εικόνα 31 παρουσιάζεται η αρχιτεκτονική ανάμεσα στον αγωγό CI/CD, το αποθετήριο εικόνων, τον Kubernetes Agent και τη συστοιχία Kubernetes.



Εικόνα 31 - Αρχιτεκτονική Gitlab - Kubernetes

5.3.3.3 Μέτρα σκλήρυνσης της συστοιχίας Kubernetes

Για την εφαρμογή μέτρων ασφάλειας στη συστοιχία Kubernetes χρησιμοποιήθηκε το εργαλείο Kyverno²⁵. Πρόκειται για ένα μέσο καθορισμού και ελέγχου επιβολής των κανόνων λειτουργίας δοχείων, απαγορεύοντας την εγκατάσταση ή επιβάλλοντας την δυναμική προσαρμογή της συμπεριφοράς τους. Με το Kyverno, οι πολιτικές μπορούν να καθορίζονται ως κώδικας

²⁵ <https://github.com/kyverno/kyverno>

YAML και να εφαρμόζονται σε επίπεδο αντικειμένων Kubernetes, προσφέροντας έναν ευέλικτο τρόπο διαχείρισης των πόρων και της ασφάλειας.

Με βάση αρχεία YAML που προτείνονται από την κοινότητα, εφαρμόστηκαν οι παρακάτω πολιτικές ασφαλείας για τα δοχεία που εγκαθίστανται στη συστοιχία:

5.3.3.3.1 *Απαγόρευση των χώρων ονομάτων του οικοδεσπότη (Host Namespaces)*

Αυτή η πολιτική απαγορεύει σε ένα δοχείο να δηλώσει πρόσβαση στο χώρο ονομάτων του εξυπηρετητή σε επίπεδο δικτύου και διεργασιών.

5.3.3.3.2 *Απαγόρευση των δοχείων με τη σημαία privileged*

Είδαμε ήδη στην Ενότητα 4.2.4 ότι η εκτέλεση δοχείων με τη συγκεκριμένη σημαία είναι σοβαρό κενό ασφαλείας. Με αυτή την πολιτική αυτή η δυνατότητα αφαιρείται.

5.3.3.3.3 *Απαγόρευση της σύνδεσης δοχείων με host paths*

Με αυτόν τον κανόνα απαγορεύεται σε ένα δοχείο να χρησιμοποιεί φακέλους του οικοδεσπότη (δηλ. του μηχανήματος φιλοξενίας που ανήκει στην συστοιχία), καθώς προτιμάται η λύση των persistent volumes (των συνεπών/επίμονων αποθηκευτικών χώρων) ως πιο ασφαλής.

5.3.3.3.4 *Απαγόρευση της χρήσης θυρών του οικοδεσπότη*

Αυτός ο κανόνας απαγορεύει την απευθείας σύνδεση ενός δοχείου με θύρες του οικοδεσπότη.

5.3.3.3.5 *Απαγόρευση χρήσης procMount πλην του βασικού*

Κατά την εκτέλεση ενός δοχείου σε περιβάλλον Linux δημιουργείται ένας εικονικός φάκελος /proc, στον οποίο βρίσκονται πληροφορίες για το κέλυφος και τις διεργασίες του. Αυτός ο κανόνας επιβάλλει στα νέα δοχεία τη χρήση του προεπιλεγμένου αυτού φακέλου σε λειτουργία μόνο για ανάγνωση.

5.3.3.3.6 *Παρακολούθηση των δυνατοτήτων των δοχείων*

Σε περίπτωση που σε κάποιο δοχείο αποδοθεί δυνατότητα Linux έξω από τη λίστα [AUDIT_WRITE, CHOWN, DAC_OVERRIDE, FOWNER, FSETID, KILL, MKNOD, NET_BIND_SERVICE, SETFCAP, SETGID, SETPCAP, SETUID, SYS_CHROOT], δηλαδή τις δυνατότητες των μη προνομιούχων δοχείων), στην κονσόλα του χρήστη εμφανίζεται προειδοποίηση.

5.3.3.3.7 *Απαγόρευση τροποποίησης της ετικέτας SELinux*

Το SELinux (Security-Enhanced Linux) είναι ένα σύστημα ελέγχου πρόσβασης που ενσωματώνεται στον πυρήνα του Linux σε συστήματα, όπως CentOS, RHEL και Fedora, παρέχοντας επιπρόσθετη λειτουργικότητα ασφαλείας σε επίπεδο πυρήνα. Το SELinux επιτρέπει τον καθορισμό αποτρεπτικών πολιτικών ασφαλείας, προστατεύοντας το σύστημα από μη εξουσιοδοτημένες προσβάσεις.

Το SELinux ενισχύει την ασφάλεια των δοχείων, ελέγχοντας και περιορίζοντας τα δικαιώματα πρόσβασης των διαδικασιών του εκάστοτε δοχείου σε συγκεκριμένους πόρους του συστήματος. Εξ'ορισμού, συστήματα όπως το CentOS (που χρησιμοποιήθηκε στο πλαίσιο της διπλωματικής) επιβάλλουν σε κάθε δοχείο την ετικέτα 'container_t', περιορίζοντας σημαντικά τα δικαιώματα πρόσβασης αυτού. Ο τρέχον κανόνας απαγορεύει σε κάποιο δοχείο να τροποποιήσει τη συγκεκριμένη ετικέτα και να αλλάξει τα δικαιώματά πρόσβασής του.

5.3.3.3.8 *Απαγόρευσης εκτέλεσης του δοχείου σαν χρήστης root*

Ήδη στην Ενότητα 4.2.1 έγινε φανερό η επικινδυνότητα της εκτέλεσης ενός δοχείου σαν χρήστης root. Με τον τρέχοντα κανόνα απαγορεύεται η εκτέλεση δοχείων με τον συγκεκριμένο χρήστη.

5.3.3.3.9 *Απαγόρευση εκτέλεσης δοχείου χωρίς εξατομικευμένο προφίλ SECCOMP*

Όπως είδαμε στην Ενότητα 5.2.3.9, στον αγωγό υπάρχει ένα χειροκίνητο στάδιο, το οποίο παράγει μια βάση για ένα προφίλ SECCOMP, με μια περιοριστική λίστα επιτρεπόμενων κλήσεων προς το κέλυφος του λειτουργικού συστήματος Linux. Ο συγκεκριμένος κανόνας απαγορεύει την εκτέλεση οποιουδήποτε δοχείου που δεν έχει σημειωθεί για χρήση με ένα εξατομικευμένο προφίλ. Ακόμα και το γενικό προφίλ για δοχεία με όνομα RuntimeDefault απορρίπτεται.

5.3.3.3.10 *Απαγόρευση επικίνδυνων ρυθμίσεων sysctls*

Τα "sysctls" είναι παράμετροι ελέγχου του συστήματος που επιτρέπουν την δυναμική ρύθμιση πολλών παραμέτρων του πυρήνα του λειτουργικού συστήματος. Στο πλαίσιο των δοχείων, αυτές οι παράμετροι χρησιμοποιούνται για να προσαρμοστούν οι ρυθμίσεις του πυρήνα, προσφέροντας έτσι έναν τρόπο ελέγχου της συμπεριφοράς των εφαρμογών που εκτελούνται μέσα στα δοχεία.

Η ανεξέλεγκτη ρύθμιση των sysctls μπορεί να προκαλέσει προβλήματα ασφαλείας, εάν για παράδειγμα, οι ρυθμίσεις δεν είναι κατάλληλα διαμορφωμένες, μπορεί να εκτεθούν ευαίσθητες λειτουργίες του πυρήνα, προκαλώντας αστάθεια στο σύστημα ή ακόμη και δυνητικά ανοίγοντας

τον δρόμο για ρήγματα ασφαλείας. Αυτός ο κανόνας επιτρέπει μόνο ένα αυστηρά καθορισμένο σύνολο systcls ρυθμίσεων.

5.3.3.3.11 Απαγόρευση εγκατάστασης εικόνας με την ετικέτα (tag) latest

Το δοχείο που εκτελείται κάθε φορά πρέπει να έχει μια σαφώς ορισμένη εικόνα με ετικέτα διάφορη του latest, προκειμένου να είναι σαφές ποιος κώδικας εκτελείται ανά πάσα στιγμή. Αυτός ο κανόνας επιτρέπει μόνο σε δοχεία με σαφώς ορισμένη ετικέτα να εγκαθίστανται.

5.3.3.3.12 Απαγόρευση σύνδεσης του δοχείου με το Socket του Container CRI

Στην Ενότητα 4.2.4 έγινε φανερή η επικινδυνότητα της σύνδεσης του socket του Container CRI (Container Runtime Interface) με ένα δοχείο, καθώς μπορεί να οδηγήσει σε απόδραση και κατάληψη του οικοδεσπότη από έναν επιτιθέμενο. Αυτός ο κανόνας δεν επιτρέπει σε κανένα δοχείο με αυτή τη σύνδεση να εγκατασταθεί.

5.3.3.3.13 Απαγόρευση ανάθεσης σε δοχείο της δυνατότητας Linux CAP_NET_RAW

Αυτός ο κανόνας απαιτεί από όλα τα δοχεία να έχουν ρητά απεμπολήσει τη δυνατότητα Linux CAP_NET_RAW, καθώς μπορεί να χρησιμοποιηθεί για πλαστογράφηση δικτυακών πακέτων [88] και σύνδεση με δικτυακές διεπαφές εκτός εύρους του δοχείου, ανοίγοντας το δρόμο για επιθέσεις ενδιάμεσου (Man in the Middle).

5.3.3.3.14 Απαγόρευση εγγραφής στο root σύστημα αρχείων του δοχείου (root filesystem)

Η επιφάνεια επίθεσης ενός δοχείου μειώνεται δραστικά [87], εάν το root σύστημα αρχείων του δεν είναι διαθέσιμο για εγγραφή, παρά μόνο για ανάγνωση. Με αυτό τον τρόπο, ακόμη και αν κάποιος επιτιθέμενος πάρει τον έλεγχο του δοχείου, δεν είναι σε θέση να τροποποιήσει αρχεία ή να εγκαταστήσει νέα, σε τοποθεσίες διαφορετικές των persistent volumes που έχουν οριστεί.

5.3.3.3.15 Απαγόρευση άντλησης εικόνας από μη προκαθορισμένο αποθετήριο εικόνων

Με αυτόν τον κανόνα παρατίθενται περιοριστικά μέσα στο αρχείο yaml τα επιτρεπτά αποθετήρια εικόνων, από τα οποία επιτρέπεται η εγκατάσταση δοχείων.

5.3.4 Υλοποίηση συστήματος παρακολούθησης & ειδοποίησης

5.3.4.1 Γενικά

Η παρακολούθηση μιας συστοιχίας Kubernetes είναι ζωτικής σημασίας για τη διασφάλιση της ομαλής λειτουργίας, αξιοπιστίας και ασφάλειας του συστήματος [89]. Αυτό περιλαμβάνει την παρακολούθηση των πόρων, τη διαχείριση της απόδοσης, καθώς και την ανίχνευση και αντιμετώπιση προβλημάτων. Οι πιο σημαντικοί λόγοι για την παρακολούθηση μιας συστοιχίας Kubernetes είναι οι παρακάτω:

- **Απόδοση συστήματος:** Η παρακολούθηση βοηθά στην αναγνώριση τυχόν υψηλού φόρτου που αντιμετωπίζεται κυρίως με οριζόντια κλιμάκωση, και άλλα θέματα που επηρεάζουν την απόδοση της συστοιχίας.
- **Ανίχνευση προβλημάτων:** Η παρακολούθηση επιτρέπει την άμεση ανίχνευση προβλημάτων, όπως σφάλματα εφαρμογών, προβλήματα δικτύου, ή αποτυχίες κόμβων.
- **Αυξημένη Διαθεσιμότητα:** Η παρακολούθηση επιτρέπει τη λήψη προληπτικών μέτρων για την αποφυγή πιθανών προβλημάτων που μπορεί να επηρεάσουν τη διαθεσιμότητα των εφαρμογών.
- **Ασφάλεια:** Η παρακολούθηση βοηθά στον εντοπισμό ασυνήθιστων δραστηριοτήτων που ενδέχεται να υποδεικνύουν προβλήματα ασφαλείας ή επιθέσεις.

Συνολικά, η παρακολούθηση της συστοιχίας Kubernetes είναι θεμελιώδους σημασίας για τη διασφάλιση ότι οι εφαρμογές εκτελούνται αποτελεσματικά, με αξιοπιστία και υψηλή ασφάλεια.

5.3.4.2 Μέτρα παρακολούθησης και ειδοποίησης της συστοιχίας Kubernetes

Για να επιτευχθούν οι παραπάνω στόχοι επιστρατεύτηκαν τα παρακάτω μέτρα.

5.3.4.2.1 Χρήση του εργαλείου Prometheus για τη συγκέντρωση δεδομένων απόδοσης

Το Prometheus²⁶ είναι ένα σύστημα παρακολούθησης (monitoring) και συλλογής δεδομένων (μέτρησης) υπολογιστικών συστημάτων. Αναπτύχθηκε αρχικά από την εταιρεία SoundCloud και αποτελεί ένα ευέλικτο εργαλείο παρακολούθησης, ιδανικό για περιβάλλοντα μεγάλης κλίμακας.

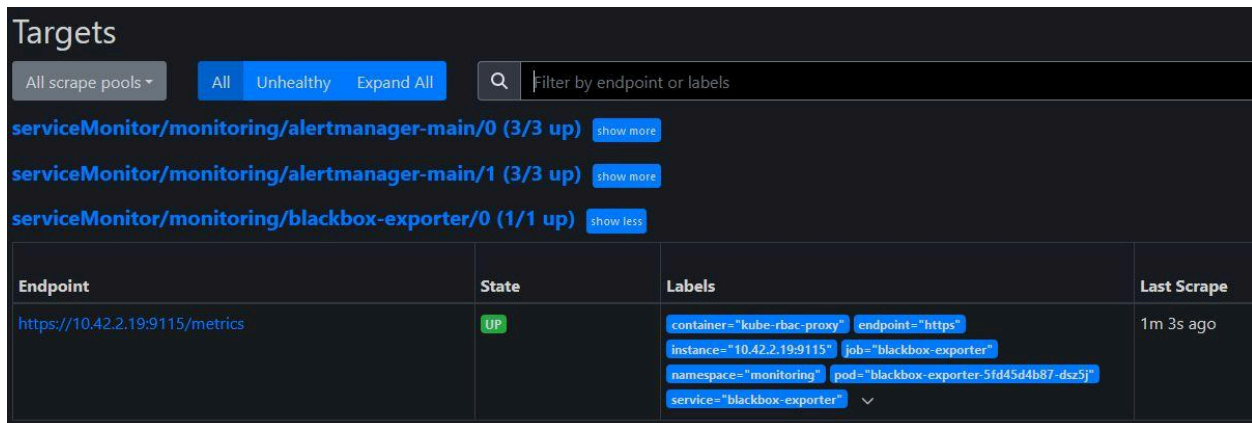
Το Prometheus λειτουργεί ως συλλέκτης δεδομένων, συλλέγοντας πληροφορίες από διάφορες υπηρεσίες και συστήματα. Αυτό περιλαμβάνει μετρήσεις από εφαρμογές, μέρη του

²⁶ <https://github.com/prometheus/prometheus>

λειτουργικού συστήματος και άλλες παραμέτρους από το περιβάλλον εκτέλεσης. Οι μετρήσεις αυτές αποθηκεύονται σε μια βάση δεδομένων του Prometheus και μπορούν να χρησιμοποιηθούν για τη δημιουργία γραφημάτων, ειδοποιήσεων και αναλύσεων απόδοσης.

Ένα από τα χαρακτηριστικά του Prometheus είναι η επεκτασιμότητά του, καθώς μπορεί να ενσωματώνει διάφορα είδη συλλογής δεδομένων και να υποστηρίζει πολλά μοντέλα παρακολούθησης. Είναι επίσης συχνά χρησιμοποιούμενο στο πλαίσιο DevOps.

Για να πραγματοποιηθεί η διασύνδεση του Prometheus με τη συστοιχία Kubernetes, έγινε εγκατάσταση, υπό τη μορφή pods (δοχείων), του εργαλείου Prometheus Adapter²⁷, το οποίο αναλαμβάνει να παράσχει δεδομένα της συστοιχίας μέσω API, του Prometheus Operator²⁸ για την εγκατάσταση δοχείων σε όλους τους κόμβους, καθώς και του Prometheus Node Exporter²⁹ για την εξαγωγή των μετρικών δεδομένων από κάθε κόμβο. Στην Εικόνα 32 βλέπουμε το UI συλλογής μετρικών δεδομένων του Prometheus.



Endpoint	State	Labels	Last Scrape
https://10.42.2.19:9115/metrics	UP	container="kube-rbac-proxy" endpoint="https" instance="10.42.2.19:9115" job="blackbox-exporter" namespace="monitoring" pod="blackbox-exporter-5fd45d4b87-dsz5j" service="blackbox-exporter"	1m 3s ago

Εικόνα 32 - Συλλογή δεδομένων από το Prometheus

5.3.4.2.2 Χρήση του εργαλείου Grafana για την απεικόνιση των μετρικών δεδομένων

Το Grafana³⁰ είναι μια πλατφόρμα ανοικτού κώδικα για απεικόνιση δεδομένων και παρακολούθηση συστημάτων. Σχεδιασμένο για να επιτρέπει τη δημιουργία όμορφων και ευέλικτων γραφημάτων, το Grafana αποτελεί ιδανική επιλογή για την οπτικοποίηση δεδομένων από διαφορετικές πηγές.

Με το Grafana, οι χρήστες μπορούν να δημιουργήσουν πίνακες ελέγχου (dashboards) που συνδυάζουν δεδομένα από ποικίλες πηγές, όπως βάσεις δεδομένων, συστήματα παρακολούθησης,

²⁷ <https://github.com/kubernetes-sigs/prometheus-adapter>

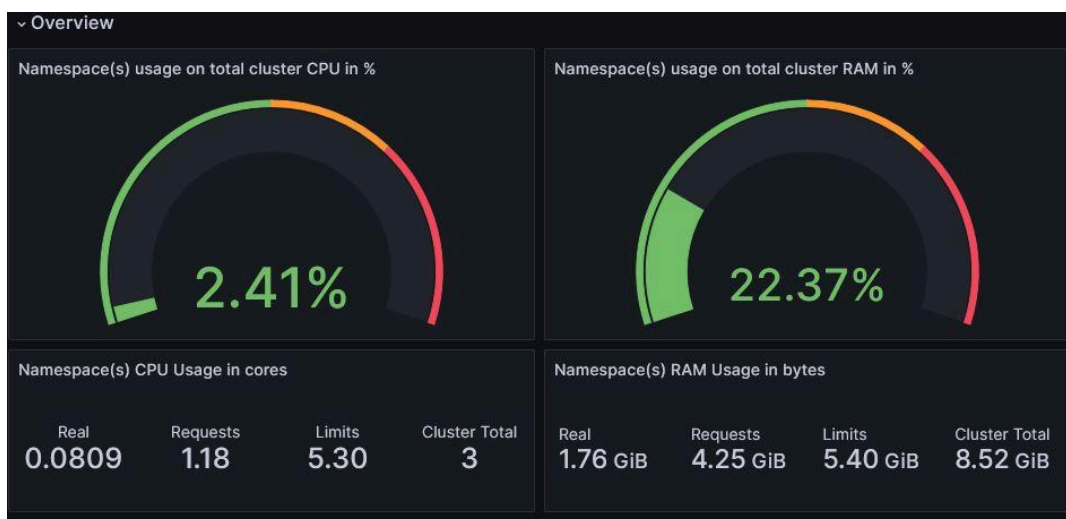
²⁸ <https://github.com/prometheus-operator/prometheus-operator>

²⁹ https://github.com/prometheus/node_exporter

³⁰ <https://github.com/grafana/grafana>

υπηρεσίες νέφους κ.α. Η πλατφόρμα υποστηρίζει πολλούς τύπους γραφημάτων και προσφέρει πολλές επιλογές προσαρμογής για την εξατομίκευση του εμφανιζόμενου περιεχομένου.

Ένα ισχυρό χαρακτηριστικό του Grafana, το οποίο εγκαθίσταται μέσα στη συστοιχία Kubernetes, είναι η δυνατότητα επέκτασής του μέσω πρόσθετων (plugins), τα οποία προσθέτουν νέες δυνατότητες και ενσωματώνουν διάφορες υπηρεσίες. Επιπλέον, το Grafana χρησιμοποιείται ευρέως σε συνδυασμό με συστήματα παρακολούθησης, όπως το Prometheus, το οποίο αναγνωρίζει ως πηγή μετρικών δεδομένων, επιτρέποντας την εύκολη απεικόνιση και ανάλυση δεδομένων παρακολούθησης. Στην Εικόνα 33 μπορούμε να δούμε ένα στιγμιότυπο του Grafana, στο οποίο είναι διαθέσιμα σημαντικά γραφήματα για την υγεία της συστοιχίας, όπως ο φόρτος του επεξεργαστή και η χρήση της μνήμης.



Εικόνα 33 - Γραφήματα του εργαλείου Grafana

5.3.4.2.3 Χρήση του εργαλείου Falco για την παρακολούθηση των γεγονότων και διεργασιών της συστοιχίας Kubernetes

Το Falco³¹ είναι ένα ανοικτού κώδικα σύστημα ανίχνευσης απειλών για περιβάλλοντα υπολογιστών βασισμένα σε Linux. Σχεδιάστηκε ειδικά για την ανίχνευση γεγονότων ασφαλείας και την παρακολούθηση των δραστηριοτήτων εφαρμογών σε περιβάλλοντα που χρησιμοποιούν δοχεία, όπως το Kubernetes.

³¹ <https://falco.org/docs/>

Οι κύριες λειτουργίες του Falco είναι οι εξής:

- **Ανίχνευση Παραβιάσεων:**

Το Falco παρακολουθεί τις ενέργειες στο σύστημα, όπως προσπάθειες πρόσβασης, χρήση ευαίσθητων πόρων, ή άλλες πιθανές παραβιάσεις ασφαλείας.

- **Προσαρμοστικοί Κανόνες:**

Οι κανόνες ανίχνευσης του Falco είναι προσαρμοστικοί και επιτρέπουν τον καθορισμό ποικίλων κριτηρίων για το τι θεωρείται ως ανωμαλία στη συμπεριφορά του συστήματος.

- **Ενημέρωση σε Πραγματικό Χρόνο:**

Παρέχει ενημερώσεις σχετικά με πιθανές παραβιάσεις σε πραγματικό χρόνο, επιτρέποντας γρήγορη αντίδραση σε γεγονότα ασφαλείας.

- **Ενσωμάτωση με Άλλα Εργαλεία:**

Μπορεί να ενσωματωθεί με άλλα εργαλεία παρακολούθησης και ασφαλείας, όπως το Kubernetes και το Prometheus, προσφέροντας μια ολοκληρωμένη λύση ασφαλείας.

- **Υποστήριξη δοχείων:**

Είναι ειδικά σχεδιασμένο για την αντιμετώπιση περιβαλλόντων που χρησιμοποιούν δοχεία, καθιστώντας το κατάλληλο για σύγχρονες αρχιτεκτονικές εφαρμογών.

Για τους σκοπούς της διπλωματικής οι κανόνες του Falco ορίστηκαν ώστε να παρακολουθούν, μεταξύ άλλων, τα εξής:

- Κλιμάκωση δικαιωμάτων σε δοχεία
- Αλλαγές σε χώρους ονομάτων (namespaces)
- Πρόσβαση σε φακέλους όπως /etc, /usr/bin
- Δημιουργία Symlinks
- Απροσδόκητες δικτυακές συνδέσεις
- Εκτέλεση διεργασιών όπως ssh, scp, κα.

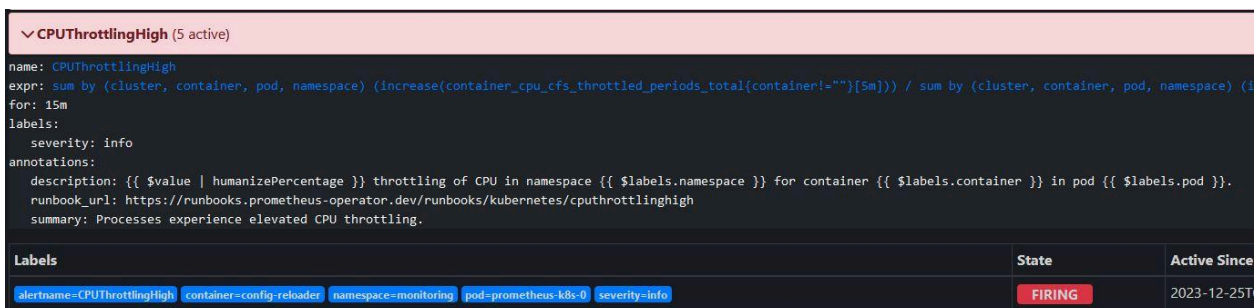
Με την εγκατάσταση του Falco και τη σύνδεση του τόσο με το Prometheus όσο και με το Grafana, πλέον παρακολουθούνται τα γεγονότα ασφαλείας τόσο του εξυπηρετητή οικοδεσπότη, όσο και των δοχείων που εκτελούνται στο πλαίσιο της συστοιχίας Kubernetes.

5.3.4.2.4 Χρήση του εργαλείου Trivy για την περιοδική σάρωση των εγκατεστημένων δοχείων για νέες ευπάθειες

Στο πλαίσιο παροχής ολοκληρωμένης προστασίας από ευπάθειες, υπάρχει η δυνατότητα εγκατάστασης στη συστοιχία Kubernetes του εργαλείου trivy-operator³², το οποίο αναλαμβάνει να εκτελεί περιοδικές σαρώσεις ευπαθειών για όλα τα δοχεία που εκτελούνται. Δυστυχώς, λόγω περιορισμού στους διαθέσιμους πόρους του συστήματος, αυτή η λειτουργία αφαιρέθηκε, αλλά προτείνεται η χρήση της σε συστήματα με επαρκείς δυνατότητες.

5.3.4.2.5 Ειδοποιήσεις σε περίπτωση γεγονότος ασφαλείας

Για την καλύτερη εποπτεία και ειδοποίηση του χρήστη σε περίπτωση κάποιου ανώμαλου γεγονότος, αξιοποιήθηκαν τόσο οι εγγενείς δυνατότητες του εργαλείου Grafana, όσο και η δυνατότητα εξωγενούς ειδοποίησης του χρήστη με μήνυμα email κ.α μέσα από το εργαλείο Alert Manager³³, το οποίο και αυτό εγκαθίσταται στη συστοιχία Kubernetes. Οι κανόνες που ρυθμίζουν υπό ποιες συνθήκες παραβιάζεται η σωστή λειτουργία της συστοιχίας, αλλά και με ποιο τρόπο πρέπει να ενημερωθεί ο χρήστης καθορίζονται με βάση αρχεία YAML. Στην Εικόνα 34 μπορούμε να δούμε μια λίστα με κανόνες που έχουν παραβιαστεί και έχουν οδηγήσει σε ειδοποίηση του χρήστη, όπως είναι ο κανόνας περί υψηλής χρήσης CPU από έναν χώρο ονομάτων δοχείων.



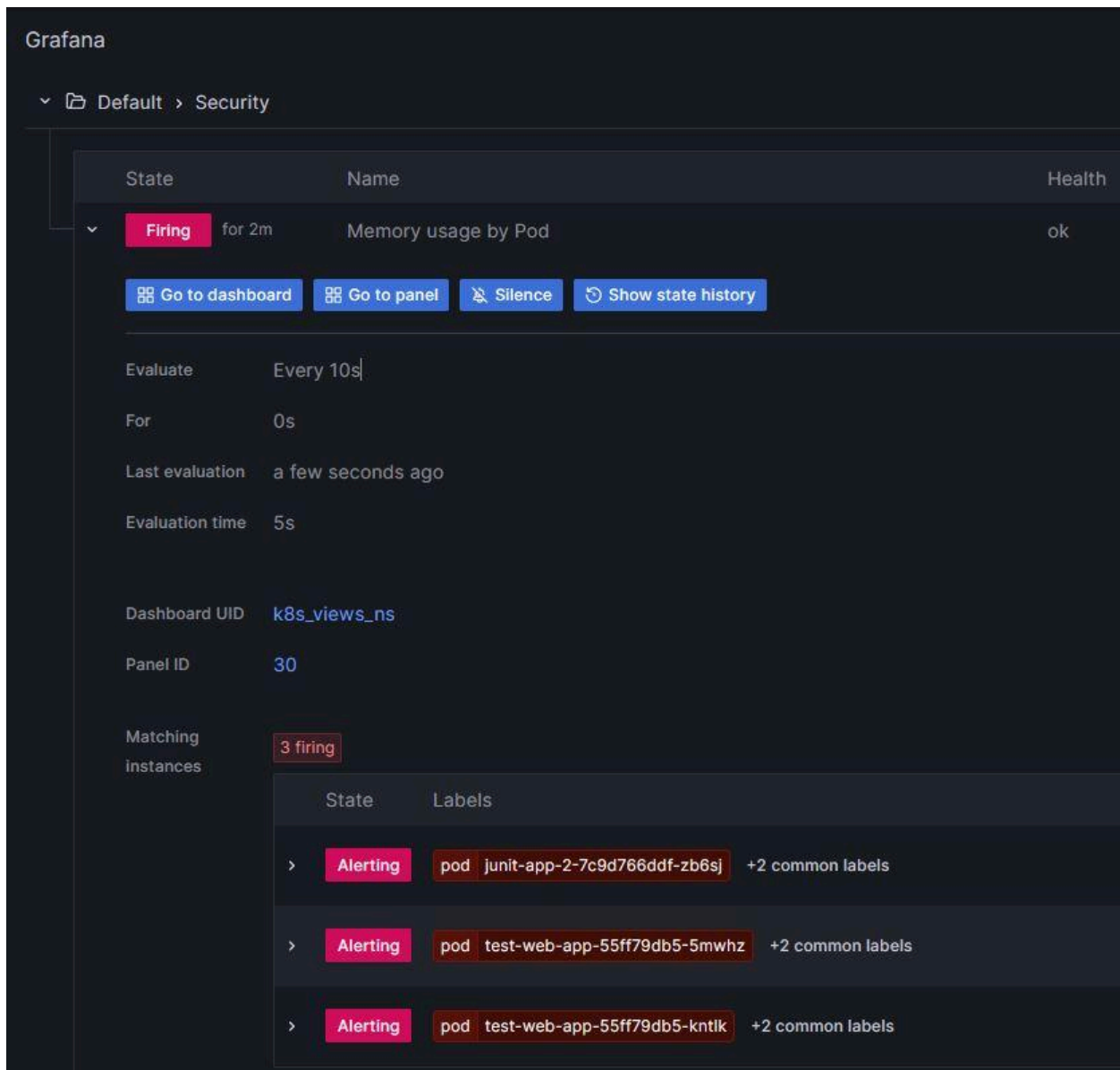
▼ CPUThrottlingHigh (5 active)		
<pre>name: CPUThrottlingHigh expr: sum by (cluster, container, pod, namespace) (increase(container_cpu_cfs_throttled_periods_total{container!=""}[5m])) / sum by (cluster, container, pod, namespace) (container_cpu_cfs_throttled_periods_total{container!=""}[5m]) for: 15m labels: severity: info annotations: description: '{{ \$value humanizePercentage }} throttling of CPU in namespace {{ \$labels.namespace }} for container {{ \$labels.container }} in pod {{ \$labels.pod }}. runbook_url: https://runbooks.prometheus-operator.dev/runbooks/kubernetes/cputhrottlinghigh summary: Processes experience elevated CPU throttling.</pre>		
Labels	State	Active Since
alertname=CPUThrottlingHigh container=config-reloader namespace=monitoring pod=prometheus-k8s-0 severity=info	FIRING	2023-12-25T

Εικόνα 34 - UI του Alert Manager σε περίπτωση παραβίασης κανόνα

Την ίδια λειτουργικότητα μπορεί να επιτελέσει και το εργαλείο Grafana, το οποίο δίνει τη δυνατότητα μέσα από το γραφικό του περιβάλλον για δημιουργία ειδοποιήσεων και αποστολής μηνύματος στο χρήστη, παράλληλα με την ειδοποίηση που προκύπτει από τον πίνακα ελέγχου. Παρέχει μάλιστα και UI για άμεση επιτήρηση των ειδοποιήσεων, όπως φαίνεται και στην Εικόνα 35.

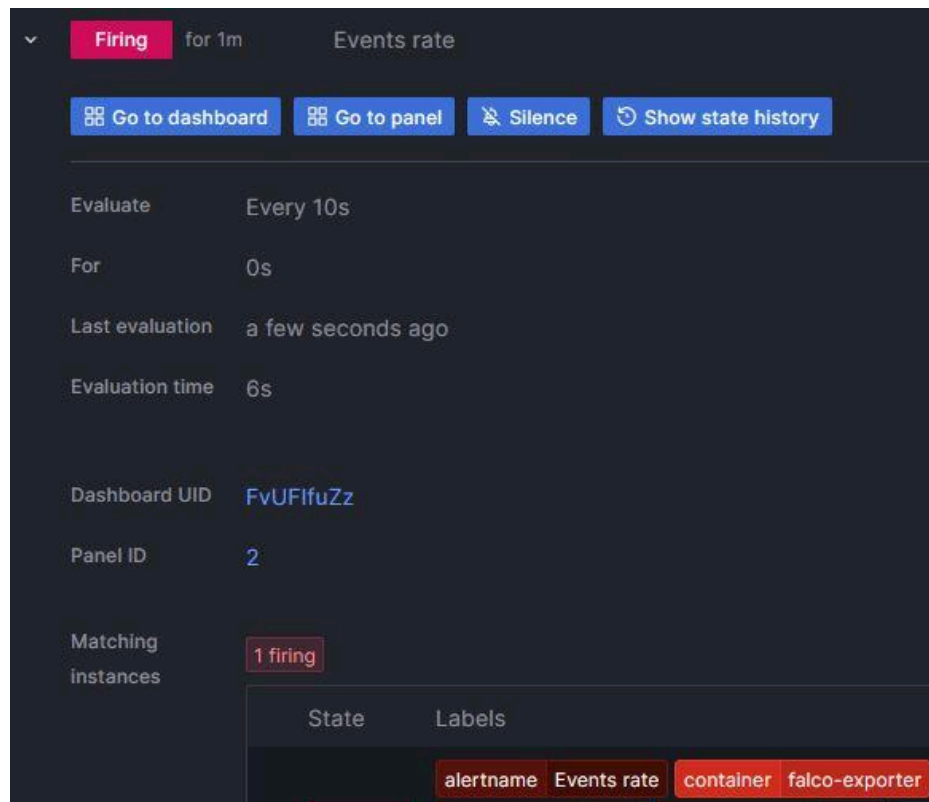
³² <https://github.com/aquasecurity/trivy-operator>

³³ <https://github.com/prometheus/alertmanager>



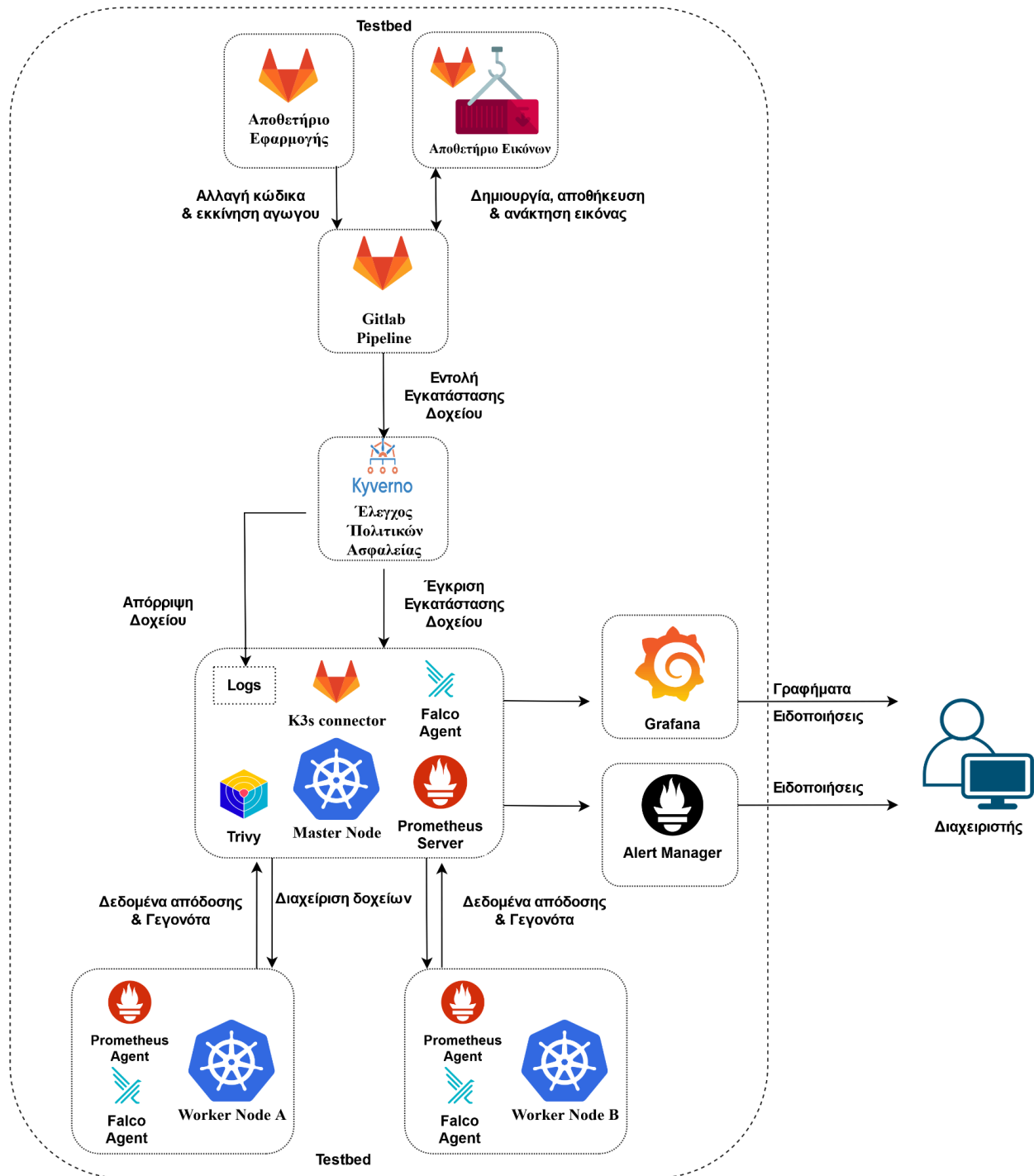
Εικόνα 35 - Ειδοποίηση για παραβίαση ορίου μνήμης δοχείου από το Grafana

Με τη διαλειτουργικότητα ανάμεσα στο Falco, Prometheus & Grafana, μπορούμε εύκολα να διαπιστώσουμε ότι εκτελώνας μια ενέργεια, όπως η προσπέλαση του ευρετηρίου /etc/shadow του οικοδεσπότη, παράγεται μια ειδοποίηση ασφαλείας στο Grafana, όπως στην Εικόνα 36.



Εικόνα 36 - Ειδοποίηση για προσπέλαση του ευρετηρίου /etc/shadow από το Grafana

Συνολικά, η αρχιτεκτονική της συστοιχίας Kubernetes και αποθετηρίων κώδικα και εικόνων, σε συνδυασμό με τα εργαλεία, ασφάλειας παρακολούθησης και ειδοποίησης φαίνεται στην Εικόνα 37.



Εικόνα 37 - Συνολική αρχιτεκτονική συστήματος

6

Εγκατάσταση, Διαμόρφωση

& Επίδειξη Συστήματος

Σε αυτό το κεφάλαιο θα δοθούν συνοπτικές οδηγίες για την εγκατάσταση του συστήματος, θα παρασχεθούν καλές πρακτικές για την ασφαλή διαμόρφωση του έργου λογισμικού στο Gitlab, και τελικά θα γίνει μια σύντομη επίδειξη της λειτουργίας του συστήματος βάσει ενδεικτικών σεναρίων.

6.1 Εγκατάσταση

Η εγκατάσταση των διαφόρων τμημάτων του συστήματος έγινε με βάση τις παρακάτω οδηγίες/συνδέσμους:

6.1.1 Εγκατάσταση υπερεπόπτη Proxmox

<https://www.proxmox.com/en/proxmox-virtual-environment/get-started>

6.1.2 Εγκατάσταση Gitlab Server

6.1.2.1 Εγκατάσταση podman & podman-compose

- ❖ `yum -y install podman`
- ❖ `sudo curl -o /usr/local/bin/podman-compose`
`https://raw.githubusercontent.com/containers/podman-compose/devel/podman_compose.py`
`sudo chmod +x /usr/local/bin/podman-compose`

6.1.2.2 Εκτέλεση αρχείου Podman-Compose.yaml

Για να εκτελεστεί το δοχείο με τον Gitlab Server

(<https://github.com/LampTrilos/aegean-thesis-junit-app/Gitlab-Compose.yaml>), ακολούθως εκτελείται η εντολή:

❖ Podman-compose up -f Gitlab-Compose.yaml -d

6.1.3 Εγκατάσταση Gitlab Runner ως δοχείο

6.1.3.1 Εγκατάσταση Docker & ρυθμίσεις χρηστών

- <https://docs.docker.com/engine/install/centos/>
- <https://docs.docker.com/engine/install/linux-postinstall/>

6.1.3.2 Εγκατάσταση δοχείου Gitlab Runner & σύνδεση με Gitlab Server

- <https://docs.gitlab.com/runner/install/docker.html>

6.1.4 Λήψη της εφαρμογής Java που περιγράφει τον αγωγό

Η εφαρμογή που υλοποιήθηκε στο πλαίσιο της διπλωματικής εργασίας βρίσκεται στο εξής αποθετήριο github: <https://github.com/LampTrilos/aegean-thesis-junit-app>

6.1.5 Εγκατάσταση SonarQube με τη μορφή δοχείου

Για την εγκατάσταση του λογισμικού SonarQube ως δοχείου ακολουθήθηκαν οι οδηγίες του παρακάτω συνδέσμου:

<https://www.promyze.com/deploy-sonarqube-docker-ubuntu-gitlab-integration/>

6.1.6 Εγκατάσταση συστοιχίας Kubernetes

Για την εγκατάσταση της συστοιχίας ακολουθήθηκαν οι οδηγίες του ακόλουθου συνδέσμου:

<https://www.baeldung.com/ops/k3s-getting-started>

6.1.7 Διασύνδεση Gitlab Server & Kubernetes

Για την επίτευξη της ασφαλούς διασύνδεσης ανάμεσα στα μέρη του τίτλου, οι οδηγίες βρίσκονται στον παρακάτω σύνδεσμο:

<https://docs.gitlab.com/ee/user/clusters/agent/gitops.html>

6.1.8 Εργαλείο ασφαλείας συστοιχίας Kubernetes Kyverno

6.1.8.1 Για την εγκατάσταση του Kyverno αξιοποιήθηκε ο ακόλουθος σύνδεσμος:

<https://medium.com/@seifeddinerajhi/sign-and-verify-container-images-with-cosign-and-kyverno>

[o-a-complete-guide-b32b1f6e6264](#)

6.1.8.2 Για την επιλογή των πολιτικών ασφαλείας που θα εφαρμοστούν έγινε χρήση του παρακάτω συνδέσμου:

<https://kyverno.io/docs/kyverno-policies/>

6.1.9 *Εργαλεία παρακολούθησης και ειδοποίησης Prometheus, Grafana & Alert Manager*

Ακολουθήθηκαν οι οδηγίες του παρακάτω αποθετηρίου:

<https://github.com/prometheus-operator/kube-prometheus>

6.1.10 *Εργαλείο καταγραφής περιστατικών ασφαλείας Falco*

Οι οδηγίες περιέχονται στον παρακάτω σύνδεσμο:

<https://falco.org/blog/falco-kind-prometheus-grafana/>

6.1.11 *Λογισμικό IPS για προστασία των εικονικών μηχανών*

Οι οδηγίες βρίσκονται στο:

<https://github.com/crowdsecurity/crowdsec>

6.2 Ασφαλής διαμόρφωση του έργου λογισμικού στο Gitlab

Για τη σκλήρυνση των λειτουργιών της πλατφόρμας Gitlab κρίθηκε απαραίτητο να υλοποιηθούν οι παρακάτω παραμετροποιήσεις από τον διαχειριστή, αφού έχει εκτελεστεί το δοχείο. Δυστυχώς δεν εντοπίστηκε μέσο ώστε να είναι ενεργοποιημένες κατά την εγκατάσταση του δοχείου, κάτι που θα οδηγούσε σε ασφαλέστερη εγκατάσταση και παραμετροποίηση:

- **Εγγραφή νέων χρηστών**

Για την εγγραφή νέων χρηστών απενεργοποιήθηκε πλήρως η δυνατότητα να εγγράφονται οι χρήστες μόνοι τους, και αντί αυτού η εγγραφή γίνεται μόνο κατόπιν ενέργειας του διαχειριστή της πλατφόρμας. Επιπρόσθετα, οι νέοι χρήστες δε μπορούν να συνδεθούν, εάν προηγουμένως δεν έχουν επιβεβαιώσει την εγκυρότητα της διεύθυνσης ηλεκτρονικού ταχυδρομείου που δήλωσαν στον διαχειριστή. Στην Εικόνα 38 φαίνονται οι σχετικές ρυθμίσεις.

Sign-up restrictions

Configure the way a user creates a new account.

- ☐ Sign-up enabled
Any user that visits `https://192.168.1.101/users/sign_in` can create an account.
- ☒ Require admin approval for new sign-ups
Any user that visits `https://192.168.1.101/users/sign_in` and creates an account must be explicitly approved by an administrator before they can sign in. Only effective if sign-ups are enabled.

Email confirmation settings

- ☐ Off
New users can sign up without confirming their email address.
- ☐ Soft
Send a confirmation email during sign up. New users can log in immediately, but must confirm their email within three days.
- ☒ Hard
Send a confirmation email during sign up. New users must confirm their email address before they can log in.

Εικόνα 38 - Επιλογές για εγγραφή νέων χρηστών

- **Σύνδεση των χρηστών στην πλατφόρμα**

Για να επιτευχθεί μεγαλύτερο επίπεδο ασφάλειας, για τη σύνδεση των χρηστών στην πλατφόρμα μέσω του UI απαιτείται η χρήση μιας εφαρμογής Αυθεντικοποίησης 2 Παραγόντων (2 Factor Authentication), ενώ για την υποβολή κώδικα η αυθεντικοποίηση γίνεται αποκλειστικά με τη χρήση προσωπικών μυστικών (Personal Access Tokens) και όχι του κωδικού χρήστη. Αυτός ο κανόνας επιβάλλεται με ενέργεια του διαχειριστή του Gitlab, και στη συνέχεια ο κάθε χρήστης πρέπει να εκδώσει ένα προσωπικό μυστικό. Αυτό ενισχύει το συνολικό επίπεδο ασφάλειας καθώς τα μυστικά αυτά μπορούν να φέρουν μόνο ένα υποσύνολο από τα συνολικά δικαιώματα του χρήστη, είναι μεταβλητά και μπορούν να ακυρωθούν ανά πάσα στιγμή, προκαλώντας ελεγχόμενη ζημία στην πλατφόρμα σε περίπτωση που διαρρεύσουν. Οι ρυθμίσεις αυτές εμφανίζονται στην Εικόνα 39.

Sign-in restrictions

Set sign-in restrictions for all users. [Learn more.](#)

- ☒ Allow password authentication for the web interface
Clear this checkbox to use an external authentication provider instead.
- ☐ Allow password authentication for Git over HTTP(S)
Clear this checkbox to use a personal access token instead.

Two-factor authentication

- ☒ Enforce two-factor authentication
Enforce two-factor authentication for all user sign-ins. [Learn more.](#)

Εικόνα 39 - Επιλογές για τη σύνδεση των χρηστών

- **Προστασία από επιθέσεις ωμής βίας (Brute force attacks)**

Η προστασία ενάντια σε επιθέσεις ωμής βίας παρέχεται κατά την εγκατάσταση ή επανεκκίνηση του δοχείου της πλατφόρμας Gitlab, αρκεί να αποδοθούν οι κατάλληλες μεταβλητές περιβάλλοντος στο αρχείο docker-compose.yaml, όπως αυτές που φαίνονται στην Εικόνα 40, με τις οποίες μετά από 5 αποτυχημένες απόπειρες κωδικού η διεύθυνση πηγής αποκλείεται για διάστημα μιας ώρας.

```
#Anti-Brute force lock out period of 1 hour after 5 password attempts
gitlab_rails['rack_attack_git_basic_auth'] = {
  'enabled' => true,
  'ip_whitelist' => ["127.0.0.1"],
  'maxretry' => 5,
  'findtime' => 60,
  'bantime' => 3600
}
```

Εικόνα 40 - Άμυνα εναντίον επιθέσεων ωμής βίας

- **Προστασία από επιθέσεις εξάντλησης πόρων**

Επιπρόσθετα, η πλατφόρμα Gitlab διαθέτει - εφόσον ενεργοποιηθεί - και δυνατότητα περιορισμού των κλήσεων ανά διεύθυνση, με ορισμό ενός μέγιστου ορίου ανά χρονική περίοδο, όπως φαίνεται στην Εικόνα 41.

☒ Enable unauthenticated web request rate limit

Maximum unauthenticated web requests per rate limit period per IP

240

Unauthenticated web rate limit period in seconds

3600

☒ Enable authenticated API request rate limit

Maximum authenticated API requests per rate limit period per user

240

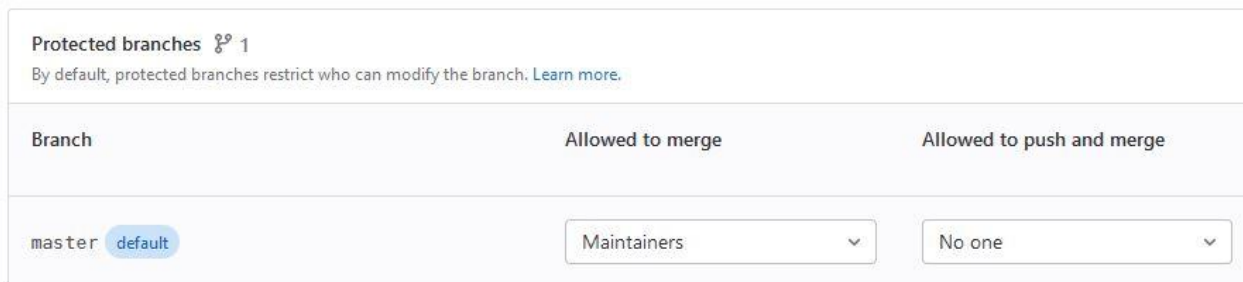
Authenticated API rate limit period in seconds

3600

Εικόνα 41 - Άμυνα εναντίον επιθέσεων εξάντλησης πόρων

- **Προστασία του κύριου κλαδιού των αποθετηρίων**

Σε κάθε νέο αποθετήριο που δημιουργείται, εξ’ορισμού απενεργοποιείται η δυνατότητα για προώθηση νέου κώδικα απευθείας στο κύριο κλαδί από απλούς χρήστες, ενώ η δυνατότητα για συγχώνευση 2 κλαδιών θα πρέπει να ανήκει μόνο στους διαχειριστές του αποθετηρίου. Προτείνεται και η απενεργοποίηση προώθησης κώδικα απευθείας στο κύριο κλαδί και από τους διαχειριστές του εκάστοτε αποθετηρίου, όπως φαίνεται στην Εικόνα 42.



Εικόνα 42 - Προστασία του κύριου κλαδιού μιας εφαρμογής

- **Ιεράρχηση των χρηστών και αποθετηρίων με βάση την αρχή του ελάχιστου δικαιώματος**

Τα αποθετήρια κάθε εφαρμογής και η αντιστοίχιση των χρηστών με αυτά έγινε με βάση την αρχή του ελάχιστου δικαιώματος. Όλα τα αποθετήρια, οι ομάδες και οι υποομάδες αποθετηρίων προτείνεται να είναι εξ’ορισμού ιδιωτικά, όπως δίνεται και αυτή η δυνατότητα, και η πρόσβαση σε αυτά δίνεται από τον εκάστοτε ιδιοκτήτη, ώστε να είναι σίγουρο ότι ο κάθε χρήστης έχει δικαίωμα να βλέπει και να χειρίζεται αποκλειστικά τις οντότητες που του είναι απαραίτητες.

- **Προστασία του αρχείου καθορισμού του αγωγού gitlab-ci.yml**

Όπως είδαμε στην Ενότητα 2.3.5, ο αγωγός για κάθε αποθετήριο εφαρμογής καθορίζεται από το αρχείο gitlab-ci.yml. Με δεδομένη την κρισιμότητα αυτού του αρχείου, τόσο ως προς την ασφαλή παραγωγή νέων τεχνημάτων, αλλά και την ασφάλεια του ίδιου του αγωγού, αποτελεί απαραίτητη - υπό το πρίσμα της ασφάλειας - πρακτική η προστασία του από μη εξουσιοδοτημένη μετατροπή. Για το σκοπό αυτό, δημιουργήθηκε έτερο αποθετήριο που περιέχει το αρχείο ορισμού του αγωγού, στο οποίο κανένας απλός χρήστης δεν έχει δικαίωμα πρόσβασης. Το αποθετήριο της εκάστοτε εφαρμογής

ρυθμίζεται κατάλληλα ώστε να αξιοποιεί το προστατευμένο πλέον αρχείο `gitlab-ci.yml`, όπως φαίνεται και στην Εικόνα 43.



Εικόνα 43 - Ορισμός προστατευμένου αποθετηρίου αγωγού

- **Προστασία των μυστικών (secrets)**

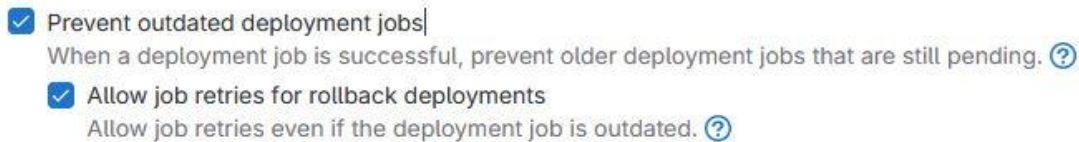
Ήδη από την Ενότητα 5.2.5 έγινε φανερή η ανάγκη ασφαλούς διαχείρισης των μυστικών που χρησιμοποιούνται κατά την εκτέλεση του αγωγού ή εκτέλεση των δοχείων. Για το σκοπό αυτό αξιοποιείται η εγγενής δυνατότητα της πλατφόρμας Gitlab για διαχείριση μυστικών, με το κάθε αποθετήριο ή ομάδα αποθετηρίων να έχει πρόσβαση αποκλειστικά στα μυστικά που χρειάζεται. Τα μυστικά επίσης μπορούν να είναι διαθέσιμα μόνο στα κύρια κλαδιά, καθώς και να μην τυπώνονται στα αρχεία καταγραφής, όπως φαίνεται στην Εικόνα 44.



Εικόνα 44 - Προστασία των μυστικών

- **Προστασία από επανεκτέλεση παλαιότερου αγωγού**

Στην Ενότητα 4.1.7 διαπιστώθηκε ότι μια από τις απειλές ενάντια σε συστήματα CI/CD είναι η επανεκτέλεση παλαιού αγωγού, που μπορεί να οδηγήσει σε εγκατάσταση παλαιού και απαρχαιωμένου στιγμιότυπου του κώδικα. Ενεργοποιώντας τη ρύθμιση που εμφανίζεται στην Εικόνα 45, αυτός ο κίνδυνος απαλείφεται.

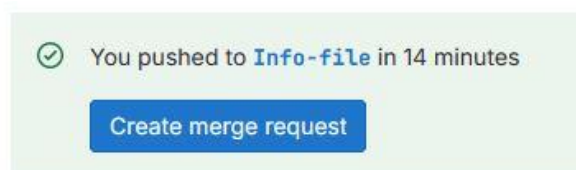


Εικόνα 45 - Προστασία από επανεκτέλεση παλαιού αγωγού

6.3 Σενάρια χρήσης

Η μεθοδολογία βασίστηκε στην εφαρμογή Maven που ήδη έχει περιγραφεί στην εργασία, η οποία υποβάλλεται σταδιακά στους διάφορους ελέγχους χωρίς να αλλάξει ο κώδικάς της, ώστε να είναι δυνατόν να παρατηρηθούν τα τυχόν σφάλματα.

Όλα τα σενάρια χρήσης ξεκινούν με τον προγραμματιστή να προσθέτει/τροποποιεί/διαγράφει οποιοδήποτε αρχείο στο αποθετήριο κώδικα. Με δεδομένο ότι το κύριο κλαδί του αποθετηρίου είναι προστατευμένο, ο μόνος τρόπος για να τροποποιηθεί ο πηγαίος κώδικας είναι η δημιουργία νέου κλαδιού και στη συνέχεια να υποβληθεί αίτημα για συγχώνευση, όπως φαίνεται στις Εικόνες 46 και 47.



Εικόνα 46 - Δημιουργία Αιτήματος Συγχώνευσης

New merge request

From `Info-file` into `main` [Change branches](#)

Title (required)

Info file added

☐ Mark as draft

Drafts cannot be merged until marked ready.

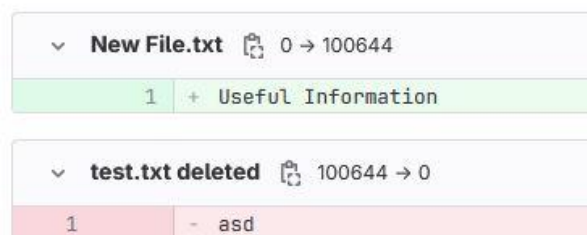
Description

Preview **B** *I* U | `</>` [🔗](#) [📌](#) [📋](#) [📄](#) [📁](#) [🔍](#) [🔖](#) [🔖](#)

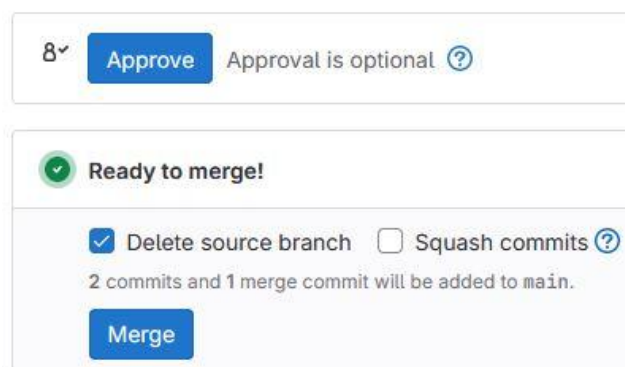
New file to explain functionalities was added as per User Story #268

Εικόνα 47 - Καταχώρηση στοιχείων αιτήματος συγχώνευσης

Στη συνέχεια, ο υπεύθυνος της εφαρμογής, αφού εκτελεστεί με επιτυχία ο αγωγός, θα πρέπει να αξιολογήσει τις αλλαγές που πρόκειται να ενσωματωθούν στον κώδικα από τη σελίδα αξιολόγησης του αιτήματος συγχώνευσης (Εικ. 48), και στη συνέχεια να εγκρίνει ή να απορρίψει τη συγχώνευση με το κύριο κλαδί (Εικ. 49).

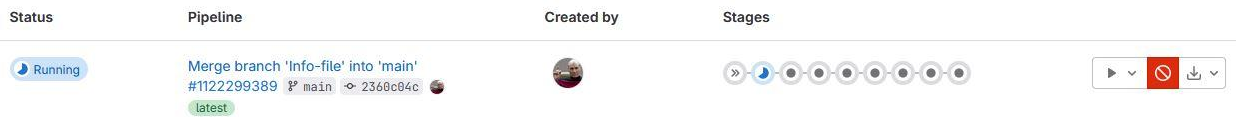


Εικόνα 48 - Εποπτεία αλλαγών



Εικόνα 49 - Αποδοχή συγχώνευσης από τον υπεύθυνο

Αμέσως μετά την έγκριση της συγχώνευσης, ξεκινά η εκτέλεση του αγωγού στο κύριο κλαδί του αποθετηρίου, του οποίου η κατάσταση φαίνεται από τη σελίδα διαχείρισης των αγωγών, όπως φαίνεται στην Εικόνα 50.



Εικόνα 50 - Κατάσταση αγωγού του συγχωνευμένου κλαδιού

6.3.1 Απόρριψη του λογισμικού/δοχείου από τον αγωγό

6.3.1.1 Σφάλματα λόγω αποτυχίας κατά το στάδιο δοκιμών του κώδικα

Σε περίπτωση που ο κώδικας που εισαχθεί οδηγήσει σε μη ορθή εκτέλεση των δοκιμών μονάδας, το αρχείο καταγραφής που εμφανίζεται στο χρήστη απεικονίζεται στην Εικόνα 51.

```
[ERROR] Failures: |
[ERROR]   HelloTest.testSayHello:33
Expected: is "2"
        but: was "Hello!\n"
[INFO]
[ERROR] Tests run: 6, Failures: 1, Errors: 0, Skipped: 0
[INFO]
[INFO] -----
[INFO] BUILD FAILURE
[INFO] -----
[INFO] Total time: 10.092 s
[INFO] Finished at: 2023-12-30T18:51:02Z
```

Εικόνα 51 - Αποτυχία δοκιμών μονάδας

Σε περίπτωση που ο κώδικας που εισαχθεί οδηγήσει σε μη συμμόρφωση με τις απαιτήσεις για το ποσοστό κάλυψης κώδικα, το αρχείο καταγραφής που εμφανίζεται στο χρήστη απεικονίζεται στην Εικόνα 52.

```
[INFO] Analyzed bundle 'java-maven-junit-2' with 3 classes
[WARNING] Rule violated for package com.example.javamavenjunit-hello-world: lines covered ratio is
0.7, but expected minimum is 0.9
[INFO] -----
[INFO] BUILD FAILURE
[INFO] -----
[INFO] Total time: 14.632 s
[INFO] Finished at: 2023-12-30T18:59:53Z
[INFO] -----
```

Εικόνα 52 - Αποτυχία ποσοστού κάλυψης κώδικα

Σε περίπτωση που ο κώδικας που εισαχθεί οδηγήσει σε μη ορθή εκτέλεση των δοκιμών ενοποίησης, το αρχείο καταγραφής που εμφανίζεται στο χρήστη απεικονίζεται στην Εικόνα 53.

```
[INFO] Results:
[INFO]
[ERROR] Failures:
[ERROR]   HelloWithTestsIT.doesItSayHelloTest:44
Expected: is "2"
but: was "Hello!\n"
[ERROR]   HelloWithTestsIT.doesItSayHelloTest3:57
Expected: is "2"
but: was "Hello!\nHello!\nHello!\n"
[INFO]
[ERROR] Tests run: 2, Failures: 2, Errors: 0, Skipped: 0
[INFO]
[INFO]
[INFO] --- maven-failsafe-plugin:3.2.2:verify (default) @ java-maven-junit-2 ---
[INFO] -----
[INFO] BUILD FAILURE
[INFO] -----
```

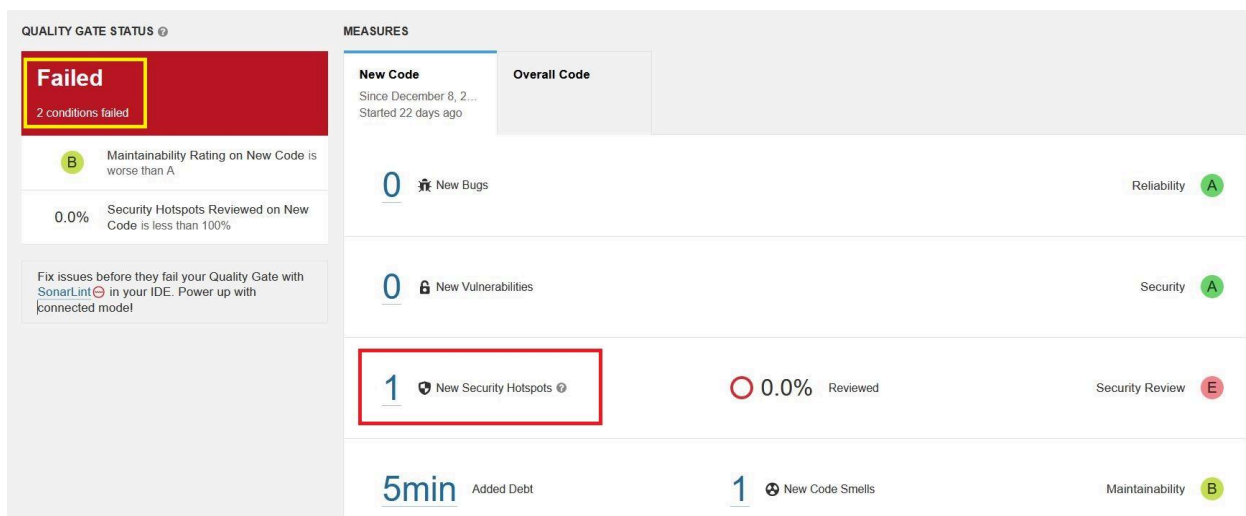
Εικόνα 53 - Αποτυχία δοκιμών ενοποίησης

6.3.1.2 Σφάλματα λόγω αποτυχίας κατά το στάδιο ελέγχου του κώδικα

Σε περίπτωση που ο κώδικας που εισαχθεί οδηγήσει σε απόρριψη από τις πύλες ποιότητας του SonarQube, το αρχείο καταγραφής που εμφανίζεται στο χρήστη απεικονίζεται στην Εικόνα 54, ενώ επισκεπτόμενοι το URL που εμφανίζεται, έχουμε αναλυτικά τους λόγους που απέτυχε η ενέργεια και δεν πέρασε ο κώδικας από τις αντίστοιχες πύλες ποιότητας, στην Εικόνα 55.


```
[INFO] ----- Check Quality Gate status
[INFO] Waiting for the analysis report to be processed (max 300s)
[INFO] -----
[INFO] BUILD FAILURE
[INFO] -----
[INFO] Total time: 02:01 min
[INFO] Finished at: 2023-12-30T19:19:53Z
[INFO] -----
[ERROR] Failed to execute goal org.sonarsource.scanner.maven:sonar-maven-plugin:3.7.0.1746:sonar
(default-cli) on project java-maven-junit-2: QUALITY GATE STATUS: FAILED - View details on htt
p://192.168.1.105:9000/dashboard?id=com.example%3Ajava-maven-junit-2&branch=master -> [Help 1]
```

Εικόνα 54 - Αποτυχία ελέγχου στατικής ανάλυσης



Εικόνα 55 - Ανάλυση αποτελεσμάτων SonarQube

Σε περίπτωση που ο κώδικας που εισαχθεί οδηγήσει σε μη συμμόρφωση με τον κανόνα μη φανέρωσης μυστικών μέσα στο αποθετήριο κώδικα, το αρχείο καταγραφής που εμφανίζεται στο χρήστη απεικονίζεται στην Εικόνα 56.

```

Checking for exposed secrets..

  ○
  | \
  |  ○
  ○ █
  █  gitLeaks

7:36PM INF scan completed in 2.01s
7:36PM WRN leaks found: 2
Finding:      REDACTED
Secret:       REDACTED
RuleID:       private-key
Entropy:      6.020492
File:         src/main/resources/key.pem
Line:         1
Fingerprint:  src/main/resources/key.pem:private-key:1
Finding:      REDACTED
Secret:       REDACTED
RuleID:       private-key
Entropy:      6.020492
File:         target/classes/key.pem
Line:         1
Fingerprint:  target/classes/key.pem:private-key:1
Exposed secrets have been detected, exiting with error.

```

Εικόνα 56 - Αποτυχία κατά την ανίχνευση μυστικών

Σε περίπτωση που εισαχθούν ευπάθειες βιβλιοθηκών βαθμολογίας υψηλότερης του 8, το αρχείο καταγραφής που εμφανίζεται στο χρήστη απεικονίζεται στην Εικόνα 57 ενώ η αναλυτική αναφορά είναι διαθέσιμη σε μορφή ιστοσελίδας, όπως φαίνεται στην Εικόνα 58.

```
[INFO] BUILD FAILURE
[INFO] -----
[INFO] Total time: 01:09 min
[INFO] Finished at: 2023-12-30T19:48:59Z
[INFO] -----
[ERROR] Failed to execute goal org.owasp:dependency-check-maven:9.0.7:check (default-cli) on project java-maven-junit-2:
[ERROR]
[ERROR] One or more dependencies were identified with vulnerabilities that have a CVSS score greater than or equal to '8.0':
[ERROR]
[ERROR] snakeyaml-1.33.jar: CVE-2022-1471(9.8)
[ERROR] spring-boot-3.0.5.jar: CVE-2023-20873(9.8)
[ERROR] spring-boot-devtools-3.0.5.jar: CVE-2023-20873(9.8)|
[ERROR] spring-boot-starter-web-3.0.5.jar: CVE-2023-20873(9.8)
[ERROR]
[ERROR] See the dependency-check report for more details.
```

Εικόνα 57 - Αποτυχία κατά τον έλεγχο ευπαθειών βιβλιοθηκών

Summary

Display: [Showing Vulnerable Dependencies \(click to show all\)](#)

Dependency	Vulnerability IDs	Package	Highest Severity	CVE Count	Confidence	Evidence Count
jackson-databind-2.14.2.jar	cpe:2.3:a:fasterxml:jackson-databind:2.14.2:***** cpe:2.3:a:fasterxml:jackson-modules-java8:2.14.2:*****	pkg:maven/com.fasterxml.jackson.core/jackson-databind@2.14.2	MEDIUM	1	Highest	41
logback-core-1.4.6.jar	cpe:2.3:a:qos.logback:logback-core:1.4.6:*****	pkg:maven/ch.qos.logback/logback-core@1.4.6	HIGH	1	Highest	38
snakeyaml-1.33.jar	cpe:2.3:a:snakeyaml:project:snakeyaml:1.33:*****	pkg:maven/org.yaml/snakeyaml@1.33	CRITICAL	1	Highest	40
spring-boot-3.0.5.jar	cpe:2.3:a:vmware:spring_boot:3.0.5:*****	pkg:maven/org.springframework.boot/spring-boot@3.0.5	CRITICAL	3	Highest	38
spring-boot-devtools-3.0.5.jar	cpe:2.3:a:vmware:spring_boot_tools:3.0.5:***** cpe:2.3:a:vmware:spring_tools:3.0.5:*****	pkg:maven/org.springframework.boot/spring-boot-devtools@3.0.5	CRITICAL	3	Highest	40
spring-boot-starter-web-3.0.5.jar	cpe:2.3:a:vmware:spring_boot:3.0.5:***** cpe:2.3:a:web_project:web:3.0.5:*****	pkg:maven/org.springframework.boot/spring-boot-starter-web@3.0.5	CRITICAL	3	Highest	36
spring-core-6.0.7.jar	cpe:2.3:a:pivotal:software:spring_framework:6.0.7:***** cpe:2.3:a:springsource:spring_framework:6.0.7:***** cpe:2.3:a:vmware:spring_framework:6.0.7:*****	pkg:maven/org.springframework/spring-core@6.0.7	HIGH	2	Highest	37
spring-web-6.0.7.jar	cpe:2.3:a:pivotal:software:spring_framework:6.0.7:***** cpe:2.3:a:vmware:spring_framework:6.0.7:***** cpe:2.3:a:web_project:web:6.0.7:*****	pkg:maven/org.springframework/spring-web@6.0.7	HIGH	2	Highest	35
tomcat-embed-core-10.1.7.jar	cpe:2.3:a:apache:tomcat:10.1.7:***** cpe:2.3:a:apache_tomcat:apache_tomcat:10.1.7:*****	pkg:maven/org.apache.tomcat.embed/tomcat-embed-core@10.1.7	HIGH*	6	Highest	63

Εικόνα 58 - Αναφορά ελέγχου ευπαθειών βιβλιοθηκών

6.3.1.3 Σφάλματα λόγω αποτυχίας κατά το στάδιο ελέγχου της εικόνας

Σε περίπτωση που το Dockerfile, βάσει του οποίου διαμορφώνεται η εικόνα, δεν συμμορφώνεται με τους κανόνες που τέθηκαν στο εργαλείο Hadolint, το αρχείο καταγραφής που εμφανίζεται στο χρήστη απεικονίζεται στην Εικόνα 59, στο κόκκινο πλαίσιο της οποίας διακρίνεται το σφάλμα για χρήση της ετικέτας ‘latest’.

```
Checking dockerfile with Hadolint..
[{"categories":["Bug Risk"],"check_name":"DL3007","description":"Using latest is prone to errors if the image will ever update. Pin the version explicitly to a release tag","fingerprint":"edcb144f02f0e43ff21f2378e26820ea1c82b2d1","location":{"lines":{"begin":21,"end":21},"path":"Dockerfile"},"severity":"blocker","type":"issue"}, {"categories":["Bug Risk"],"check_name":"DL3049","description":"Label `author` is missing.","fingerprint":"9cbb1ee8ff7801df6098d888c2f15a2ba5fe36cf","location":{"lines":{"begin":21,"end":21},"path":"Dockerfile"},"severity":"info","type":"issue"}, {"categories":["Bug Risk"],"check_name":"DL3049","description":"Label `version` is missing.","fingerprint":"b49893ba20a5c15e491cd03a911576aede6f1a3b","location":{"lines":{"begin":21,"end":21},"path":"Dockerfile"},"severity":"info","type":"issue"}]
Dockerfile violations were detected. Please follow Hadolint rules.
Uploading artifacts for failed job 00:01
```

Εικόνα 59 - Αποτυχία κατά τον έλεγχο του Dockerfile από το Hadolint

Σε περίπτωση που ο χρήστης που εκτελεί την κύρια διεργασία του δοχείου είναι ο root, το αρχείο καταγραφής που εμφανίζεται στο χρήστη απεικονίζεται στην Εικόνα 60.

```
Checking dockerfile with no-root script..
No USER Command found in Dockerfile. Checking base image..
Dockerfile base image detected: adoptopenjdk/openjdk11:jre-11.0.9_11-alpine.
Checking base image layers from: https://hub.docker.com/v2/repositories/adoptopenjdk/openjdk11/tags/jre-11.0.9_11-alpine/images.
Dockerfile, did not contain a USER Command. Checking the layers of the base image on Dockerhub for the USER command
User in Base Image is set to ROOT. Please use the USER command to change User, as running a container as root is a security vulnerability.
```

Εικόνα 60 - Αποτυχία κατά τον έλεγχο του Dockerfile για χρήση root

Σε περίπτωση που η εικόνα που διαμορφώθηκε ανιχνευθεί ότι περιέχει γνωστές ευπάθειες πακέτων ή βιβλιοθηκών με κρισιμότητα υψηλότερη του αποδεκτού που έχει οριστεί στον αγωγό, το αρχείο καταγραφής που εμφανίζεται στο χρήστη απεικονίζεται στην Εικόνα 61, στην οποία εμφανίζονται 4 ανιχνευθείσες ευπάθειες κρίσιμου επιπέδου και 32 υψηλού επιπέδου.

```

2023-12-31T10:22:19.610Z    INFO    Detected OS: alpine
2023-12-31T10:22:19.610Z    INFO    Detecting Alpine vulnerabilities...
2023-12-31T10:22:19.658Z    INFO    Number of language-specific files: 1
2023-12-31T10:22:19.658Z    INFO    Detecting jar vulnerabilities...
2023-12-31T10:22:19.894Z    WARN    This OS version is no longer supported by the distribution: alpine 3.12.1
2023-12-31T10:22:19.895Z    WARN    The vulnerability detection may be insufficient because security updates are not provided
2023-12-31T10:22:19.912Z    INFO    Table result includes only package filenames. Use '--format json' option to get the full path to the package file.
High or Critical vulnerabilities detected. Please update the Dockerfile libraries
./image.tar (alpine 3.12.1)
=====
Total: 36 (HIGH: 32, CRITICAL: 4)

```

Library	Vulnerability	Severity	Status	Installed Version	Fixed Version	Title
apk-tools	CVE-2021-36159	CRITICAL	fixed	2.10.5-r1	2.10.7-r0	libfetch: an out of boundary read while libfetch uses strtol to parse... https://avd.aquasec.com/nvd/cve-2021-36159
	CVE-2021-30139	HIGH			2.10.6-r0	In Alpine Linux apk-tools before 2.12.5, the tarball parser allows a buffer... https://avd.aquasec.com/nvd/cve-2021-30139
busybox	CVE-2021-28831			1.31.1-r19	1.31.1-r20	invalid free or segmentation fault via malformed gzip data https://avd.aquasec.com/nvd/cve-2021-28831
	CVE-2021-42378				1.31.1-r21	busybox: use-after-free in awk applet leads to denial of service and possibly... https://avd.aquasec.com/nvd/cve-2021-42378
	CVE-2021-42379					busybox: use-after-free in awk applet leads to denial of service and possibly... https://avd.aquasec.com/nvd/cve-2021-42379

Εικόνα 61 - Αποτυχία κατά τον έλεγχο ευπαθειών σε βιβλιοθήκες/πακέτα

6.3.1.4 Σφάλματα λόγω αποτυχίας κατά το στάδιο ελέγχου της προωθημένης εικόνας

Ο έλεγχος SCAP δεν οδηγεί σε αποτυχία την εκτέλεση του αγωγού, και ο χρήστης μπορεί να επιθεωρήσει το αποτέλεσμα του ελέγχου σε μια ιστοσελίδα, ώστε να λάβει τα μέτρα ασφαλείας που κρίνει σκόπιμα, όπως αυτή που απεικονίζεται στην Εικόνα 62, στην οποία εντοπίζονται τα εξής αποτελέσματα μεσαίου επιπέδου:

- **Default (C) shell Unmask:** Η συγκεκριμένη μάσκα ορίζει τα προνόμια των χρηστών επάνω σε νεοδιαμορφωμένα αρχεία. Μια ελαστική μάσκα μπορεί να οδηγήσει σε μη εξουσιοδοτημένη πρόσβαση σε αρχεία.
- **Προνόμια χρηστών σε αρχεία καταγραφής:** Εάν τα προνόμια των χρηστών σε αρχεία καταγραφής δεν είναι σωστά ορισμένα, υπάρχει ο κίνδυνος ή της μη εξουσιοδοτημένης πρόσβασης, είτε της αδυναμίας εργαλείων ασφαλείας να τα αξιολογήσουν.

Rule results

80 passed

3 failed

Severity of failed rules

3 medium

Score

Scoring system	Score	Maximum	Percent
urn:xccdf:scoring:default	97.152779	100.000000	97.15%

Rule Overview

Group	Severity	Result
▼ result = fail		
Ensure the Default C Shell Umask is Set Correctly	medium	fail
Ensure the Default Umask is Set Correctly in /etc/profile	medium	fail
Verify permissions of log files	medium	fail

Εικόνα 62 - Σελίδα αποτελεσμάτων ελέγχου SCAP

Σε περίπτωση που τα αρχεία YAML που προορίζονται για την εγκατάσταση του δοχείου στη συστοιχία Kubernetes δεν ανταποκρίνονται στις προδιαγραφές της ενέργειας “Έλεγχος ρυθμίσεων εγκατάστασης δοχείων”, το αρχείο καταγραφής που εμφανίζεται στο χρήστη απεικονίζεται στην Εικόνα 63. Στη συγκεκριμένη εικόνα εμφανίζονται τα παρακάτω σοβαρά σφάλματα:

- Το δοχείο μπορεί να τροποποιήσει το ριζικό (root) σύστημα αρχείων του, με αποτέλεσμα ένας επιτιθέμενος να μπορέσει να εγκαταστήσει δικές του λειτουργικότητες.
- Το δοχείο εκτελείται με χαμηλό δείκτη χρήστη η ομάδας χρήστη, με αποτέλεσμα να προκύπτουν οι κίνδυνοι που περιγράφονται στην απειλή της Ενότητας 4.2.1, δηλαδή της απόδρασης στον οικοδεσπότη με δικαιώματα ριζικού (root) χρήστη.
- Δεν εντοπίστηκε συνδεδεμένη υπηρεσία (service) Kubernetes.

```
Checking K3s manifests...
[CRITICAL] junit-app-2/default apps/v1/Deployment: (junit-app-2) The pod has a container with a writable root filesystem
[CRITICAL] junit-app-2/default apps/v1/Deployment: (junit-app-2) The container is running with a low user ID
[CRITICAL] junit-app-2/default apps/v1/Deployment: (junit-app-2) The container running with a low group ID
[CRITICAL] test-web-app-ingress networking.k8s.io/v1/Ingress: (/web) No service match was found
High or critical misconfigurations detected. Please update the K3s manifests
```

Εικόνα 63 - Αποτυχία κατά τον έλεγχο ρυθμίσεων εγκατάστασης δοχείων

6.3.1.5 Σφάλματα σε λοιπές ενέργειες

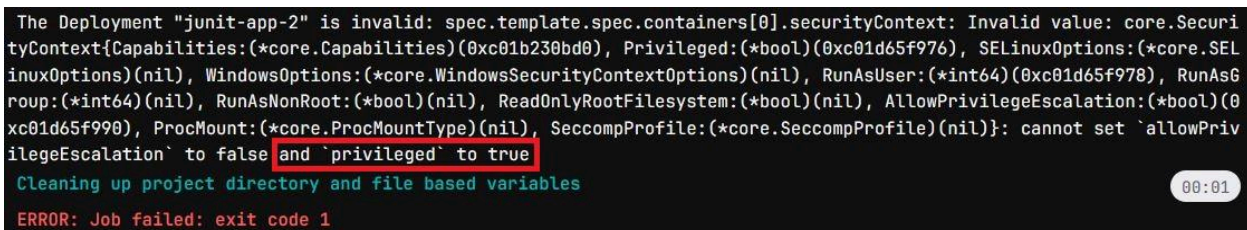
Σε κάθε περίπτωση, εάν συμβεί ένα απρόοπτο σφάλμα (π.χ. Αποτυχία κατά την επικοινωνία) που δεν εντάσσεται στον έλεγχο ασφαλείας του λογισμικού/δοχείου, εμφανίζεται ένα κατάλληλο μήνυμα ώστε ο χρήστης να διερευνήσει το αίτιο και να το επιλύσει.

6.3.2 Απόρριψη του δοχείου από τους κανόνες της συστοιχίας Kubernetes

Σε περίπτωση που όλοι οι έλεγχοι του αγωγού είναι επιτυχείς, αλλά το δοχείο απορριφθεί από τους κανόνες ασφαλείας της ίδιας της συστοιχίας Kubernetes, στο αρχείο καταγραφής εμφανίζεται και το αντίστοιχο μήνυμα απόρριψης. Ενδεικτικά παρατίθενται δυο παραβιάσεις και τα αντίστοιχα μηνύματα που αυτές παράγουν.

6.3.2.1 Απόρριψη εγκατάστασης δοχείου με τη σημαία privileged

Ακόμα και στην περίπτωση όπου ένας έλεγχος της Ενότητας 5.3.7 δε λειτουργήσει ορθά, όπως είναι η απαγόρευση privileged δοχείων, το δοχείο απορρίπτεται από τους κανόνες του εργαλείου Kyverno, όπως εμφανίζεται στο αρχείο καταγραφής της Εικόνας 64.



```
The Deployment "junit-app-2" is invalid: spec.template.spec.containers[0].securityContext: Invalid value: core.SecurityContext{Capabilities:(*core.Capabilities)(0xc01b230bd0), Privileged:(*bool)(0xc01d65f976), SELinuxOptions:(*core.SELinuxOptions)(nil), WindowsOptions:(*core.WindowsSecurityContextOptions)(nil), RunAsUser:(*int64)(0xc01d65f978), RunAsGroup:(*int64)(nil), RunAsNonRoot:(*bool)(nil), ReadOnlyRootFilesystem:(*bool)(nil), AllowPrivilegeEscalation:(*bool)(0xc01d65f990), ProcMount:(*core.ProcMountType)(nil), SeccompProfile:(*core.SeccompProfile)(nil)}: cannot set `allowPrivilegeEscalation` to false and `privileged` to true
Cleaning up project directory and file based variables
ERROR: Job failed: exit code 1
```

Εικόνα 64 - Αποτυχία κατά την εγκατάσταση λόγω της σημαίας privileged

6.3.2.2 Απόρριψη εγκατάστασης δοχείου που εκτελείται από το χρήστη root

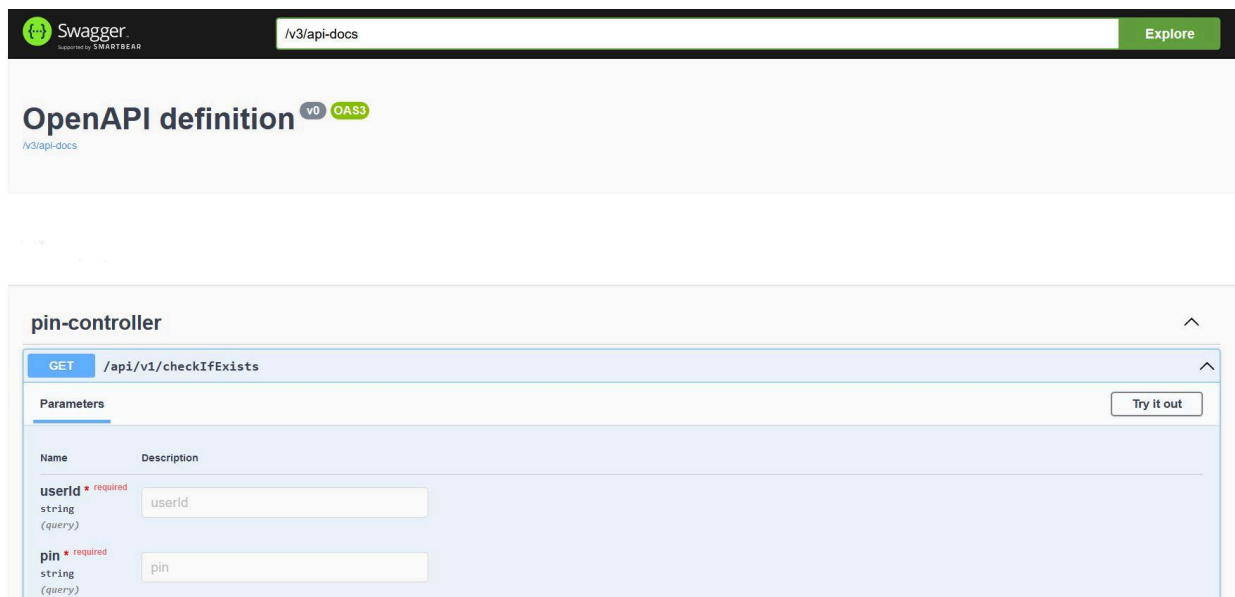
Μια ακόμα περίπτωση που θα εξεταστεί είναι η εγκατάσταση δοχείου με χρήστη root, η οποία όμως και πάλι αποτρέπεται από τους κανόνες του εργαλείου Kyverno με το αρχείο καταγραφής να εμφανίζεται στην Εικόνα 65.

```
Error from server: error when creating "/builds/aegean-apps/demonstration-apps/junit-app-2/k3s/depl-junit-app-2.yaml": admission webhook "validate.kyverno.svc-fail" denied the request: resource Deployment/default/junit-app-2 was blocked due to the following policies
require-run-as-non-root-user:
autogen-run-as-non-root-user: 'validation error: Running as root is not allowed.
The fields spec.securityContext.runAsUser, spec.containers[*].securityContext.runAsUser,
spec.initContainers[*].securityContext.runAsUser, and spec.ephemeralContainers[*].securityContext.runAsUser
must be unset or set to a number greater than zero. . rule autogen-run-as-non-root-user
failed at path /spec/template/spec/containers/0/securityContext/runAsUser/'
Cleaning up project directory and file based variables
ERROR: Job failed: exit code 1
```

Εικόνα 65 - Αποτυχία κατά την εγκατάσταση λόγω χρήστη root

6.3.3 Έγκριση του δοχείου

Στην περίπτωση που το δοχείο κατασκευαστεί με επιτυχία από τον αγωγό και εγκατασταθεί στη συστοιχία Kubernetes, μπορούμε να δούμε ότι προσφέρεται για χρήση όπως στην παρακάτω εικόνα:



Εικόνα 66 - Εγκατάσταση δοχείου

7

Συμπεράσματα**και μελλοντική εργασία**

Σε αυτό το κεφάλαιο θα συγκεντρωθούν τα συμπεράσματα που παρήχθησαν και οι εμπειρίες που αποκομίσθηκαν από την εκπόνηση της διπλωματικής εργασίας, όπως επίσης θα αποτυπωθούν πιθανές μελλοντικές επεκτάσεις αυτής.

7.1 Κάλυψη απαιτήσεων και προστασία από απειλές

Ο στόχος της Διπλωματικής εργασίας είναι να υλοποιήσει ένα ασφαλές σύστημα CI/CD βασισμένο σε δοχεία. Για τον καθορισμό της επιτυχίας η μή του εγχειρήματος, θα χρησιμοποιηθούν τόσο οι λειτουργικές και μη λειτουργικές απαιτήσεις που περιγράφηκαν στο 5ο κεφάλαιο αλλά και οι απειλές που εντοπίστηκαν στο 4ο κεφάλαιο.

7.1.1 Κάλυψη μη λειτουργικών απαιτήσεων

Οι μη λειτουργικές απαιτήσεις, όπως περιγράφηκαν στην Ενότητα 5.1.1, και η κάλυψη αυτών συνοψίζεται στον παρακάτω Πίνακα 2.

Πίνακας 2 - Κάλυψη μη λειτουργικών Απαιτήσεων

Μη Λειτουργική Απαίτηση	Ικανοποίηση	Δικαιολόγηση
Επεκτασιμότητα		Η ενέργεια ‘Έλεγχος ρυθμίσεων εγκατάστασης δοχείων’ απαιτεί την ύπαρξη οριζόντιας επέκτασης, η οποία και υλοποιείται με βάση την κατανεμημένη αρχιτεκτονική του Gitlab, με την οποία είναι εύκολη η επέκταση του συστήματος μέσω της δημιουργίας

		περισσότερων runners. Αυτό θα μπορούσε να αυτοματοποιηθεί μέσω της αυτόματης οριζόντιας κλιμάκωσης των εικονικών μηχανών που θα εκτελούν τους runners και μπορεί εύκολα να επιτευχθεί σε ένα νέφος.
Υψηλή διαθεσιμότητα	✓	Όλα τα μέρη του συστήματος (Gitlab & Gitlab Runners) μπορούν να εγκατασταθούν σε διάταξη υψηλής διαθεσιμότητας, με δεδομένη την ύπαρξη του απαραίτητου υλικού.
Απόδοση	✓	Όλες οι ενέργειες του αγωγού μέσα στο ίδιο στάδιο μπορούν να εκτελούνται παράλληλα, αν δεν έχουν εξαρτήσεις, εφόσον έχει προσφερθεί το κατάλληλο πλήθος από Runners. Επιπρόσθετα, σε ορισμένες ενέργειες τίθενται και χρονικά όρια εκτέλεσης για εξασφάλιση αποδεκτής απόδοσης.
Αξιοπιστία	✓	Η συστοιχία Kubernetes προσφέρει ανθεκτικότητα απέναντι σε σφάλματα [91].
Ασφάλεια	✓	• Έχει εφαρμοστεί στρατηγική RBAC (Role Based Access Control) • Ο αγωγός έχει σκληρυνθεί ενάντια σε γνωστές απειλές • Η συστοιχία έχει σκληρυνθεί ενάντια σε γνωστές απειλές • Η εικονικές μηχανές έχουν σκληρυνθεί ενάντια σε γνωστές απειλές • Το παραγόμενο λογισμικό έχει σκληρυνθεί ενάντια σε γνωστές απειλές
Συντήρηση	✓	Για το κάθε τμήμα και εξωτερικό εργαλείο του συστήματος υπάρχει εκτενής τεκμηρίωση και οδηγίες στο διαδίκτυο.
Παρακολούθηση & Καταγραφή	✓	Για κάθε βήμα του αγωγού και της λειτουργίας των δοχείων υπάρχει αναλυτική παρακολούθηση και καταγραφή, μέσω των εργαλείων Prometheus, Grafana και AlertManager.
Διαλειτουργικότητα	✓	Λόγω της αρθρωτής του αρχιτεκτονικής, σε συνδυασμό με τη χρήση δοχείων, το σύστημα επιτρέπει τη διαλειτουργικότητα με πολλαπλές βιβλιοθήκες και εργαλεία.
Αξιοποίηση πόρων	✓	Μέσα από τον αξιοποιημένο υπερεπόπτη Proxmox, οι πόροι του συστήματος διαμοιράζονται δυναμικά (ballooning) στο

		μέγιστο δυνατό βαθμό ανάμεσα στις διάφορες εικονικές μηχανές.
Εμπειρία Χρήστη		Το UI του συστήματος είναι κατανοητό και φιλικό προς το χρήστη με εύχρηστο μενού και μηνύματα.

7.1.2 Κάλυψη λειτουργικών απαιτήσεων

Οι λειτουργικές απαιτήσεις, όπως περιγράφηκαν στην ενότητα 5.1.2 και η κάλυψη αυτών συνοψίζεται στον παρακάτω Πίνακα 3.

Πίνακας 3 - Κάλυψη λειτουργικών Απαιτήσεων

Λειτουργική Απαίτηση	Ικανοποίηση	Δικαιολόγηση
Διαχείριση κώδικα		Προσφέρεται υπηρεσία διαχείρισης αποθετηρίων κώδικα βασισμένο στο πρότυπο .git.
Συνεργασία		Προσφέρονται λειτουργίες συνεργασίας μελών ομάδας, με διαχείριση κλαδιών και συνενώσεων.
Παρακολούθηση ζητημάτων		Προσφέρεται η δυνατότητα δημιουργίας και επίλυσης ζητημάτων, ως εγγενής δυνατότητα της πλατφόρμας Gitlab.
Γλώσσες		Προσφέρεται η δυνατότητα αποθήκευσης αρχείων σε οποιαδήποτε γλώσσα
Χρήσιμες πληροφορίες		Προσφέρεται η δυνατότητα δημιουργίας Wiki για κάθε εφαρμογή ή ομάδα εφαρμογών
Αποθήκευση εκτελέσιμων/εικόνων δοχείων		Το σύστημα αποθηκεύει τις εκτελέσιμες εικόνες/δοχεία στο αποθετήριο εικόνων που διαθέτει
Συνεχής ενσωμάτωση		Προσφέρεται η λειτουργία της συνεχούς ενσωμάτωσης με παραμετροποιήσιμα στάδια και ενέργειες.
Συνεχής εγκατάσταση		Με την καταλληλή παραμετροποίηση, η συνεχής εγκατάσταση μέσω αγωγού μπορεί να γίνει με οποιοδήποτε εντοπιστή (δοχείων)

		σε ήδη διαμορφωμένα περιβάλλοντα.
Στάδια αγωγών	✓	Το σύστημα προσφέρει επέκταση του Gitlab ώστε να υποστηρίζεται ο έλεγχος, ο σχηματισμός εκτελέσεων και η εγκατάσταση Maven-based Java εφαρμογών, με πληθώρα άλλων σταδίων.
Στάδιο δοκιμών	✓	Ο αγωγός προσφέρει πολλαπλά στάδια και ενέργειες για τον έλεγχο των λειτουργιών δοκιμής όπως Δοκιμές μονάδας και Δοκιμές ενοποίησης.
Έλεγχος κώδικα	✓	Ο αγωγός ελέγχει τον πηγαίο κώδικα ως προς την ποιότητα του και για ευπάθειες μέσω στατικής ανάλυσης κώδικα (SAST), με κατάλληλα μηνύματα προς το χρήστη για κάθε σενάριο.
Έλεγχος εικόνων	✓	Ο αγωγός ελέγχει τις παραγόμενες εικόνες για τυχόν ευπάθειες μέσω ελέγχου του Dockerfile, των παραμέτρων εγκατάστασης και για τυχόν κακόβουλο λογισμικό, και ενημερώνει τον χρήστη κατάλληλα.
Έλεγχος εξαρτήσεων	✓	Ο αγωγός ελέγχει τις εξαρτήσεις τόσο του κώδικα όσο και των παραγόμενων εικόνων με εργαλεία ανάλυσης και ανίχνευσης ευπαθειών εξαρτήσεων, και ενημερώνει τον χρήστη με κατάλληλα μηνύματα.
Παρακολούθηση δοχείων	✓	Το σύστημα παρακολουθεί τα εγκατεστημένα δοχεία με εργαλεία όπως το Prometheus, Grafana & Falco, μέσω των οποίων παρακολουθούνται τόσο οι απαιτήσεις σε πόρους του συστήματος, όσο και γεγονότα ασφαλείας.
Δικτύωση	✓	Η δικτύωση ανάμεσα στις εικονικές μηχανές και τον έξω κόσμο περιορίζεται από τον υπερεπόπτη Proxmox. Η κίνηση ανάμεσα στα δοχεία, και συγκεκριμένα οι κανόνες επικοινωνίας ανάμεσα στα δοχεία και τον έξω κόσμο, υποχρεωτικά πρέπει να περιορίζεται και ελέγχεται κατά την ενέργεια Έλεγχος ρυθμίσεων εγκατάστασης δοχείων .
Απομόνωση - Σκλήρυνση δοχείων	✓	Στα εγκατεστημένα δοχεία της συστοιχίας Kubernetes εφαρμόζεται υποχρεωτικά το γενικό προφίλ SELinux, ενώ απόπειρα παράκαμψης του χειροκίνητα αποτρέπεται από κανόνα του

		εργαλείου Kyverno. Επιπρόσθετα, έχουν σκληρυνθεί και οι ίδιες οι εικονικές μηχανές.
Προστασία από DoS	✓	Οι εικονικές μηχανές του συστήματος προστατεύονται από το IPS εργαλείο Crowdsec και κανόνες αναχώματος ασφαλείας (Firewall)
Αποδεκτά όρια	✓	Κάθε έλεγχος και κανόνας του συστήματος είναι παραμετροποιήσιμος ως προς τα αποδεκτά όρια.
Ειδοποιήσεις	✓	Εάν η εκτέλεση του αγωγού αποτύχει ή υπάρξει ένδειξη ανώμαλου/επικίνδυνου γεγονότος κατά την εκτέλεση των δοχείων ο χρήστης ειδοποιείται ανάλογα μέσω αρχείων καταγραφής και email.
Ασφάλεια - Αυθεντικοποίηση	✓	Το σύστημα μπορεί να παραμετροποιηθεί ώστε να επιτρέπει την είσοδο στο Gitlab μόνο με Αυθεντικοποίηση πολλαπλών παραγόντων (MFA), ενώ η υποβολή κώδικα γίνεται μόνο μέσω κλειδιού.
Μυστικά	✓	Η πλατφόρμα Gitlab προσφέρει εγγενώς τη δυνατότητα διαχείρισης μυστικών, χωρίς αυτά να φανερώνονται σε κανένα αρχείο καταγραφής.

7.1.3 Προστασία ενάντια σε καταγεγραμμένες απειλές

Οι απειλές εναντίον συστημάτων CI/CD, όπως περιγράφηκαν στο 4ο κεφάλαιο και η προστασία από αυτές, συνοψίζεται στον παρακάτω Πίνακα 4.

Πίνακας 4 - Προστασία από απειλές

Γενικές Απειλές		
Απειλή	Προστασία	Δικαιολόγηση
Παθητικοί/Ενεργητικοί Κρυφακουστές	✓	Η επικοινωνία ανάμεσα στην πλατφόρμα Gitlab και τους Runners είναι κρυπτογραφημένη. Η επικοινωνία ανάμεσα στην πλατφόρμα Gitlab και τη συστοιχία Kubernetes είναι κρυπτογραφημένη. Δεν προσφέρεται αμοιβαία ταυτοποίηση.

		Η επικοινωνία ανάμεσα στα δοχεία της συστοιχίας μπορεί να κρυπτογραφηθεί με χρήση του εργαλείου Istio ³⁴ , ωστόσο δεν ενσωματώθηκε στην εργασία λόγω περιορισμού πόρων.
Μη εξουσιοδοτημένη πρόσβαση στο σύστημα		Για την είσοδο στην πλατφόρμα Gitlab μπορεί να απαιτηθεί αυθεντικοποίηση πολλαπλών σταδίων (MFA), σε συνδυασμό με χρήση αναχώματος ασφαλείας για σύνδεση από επιτρεπτές διευθύνσεις.
Εκτέλεση αγωγού από χρήστη χωρίς την ανάλογη εξουσιοδότηση		Έχει εφαρμοστεί RBAC και για τη συνένωση κλαδιών απαιτούνται αυξημένα δικαιώματα χρήστη. Η αντιμετώπιση της απειλής πληρώνεται με ενέργειες του χρήστη και όχι με προκαθορισμένο τρόπο.
Τροποποίηση του αγωγού από απλό χρήστη		Ο ορισμός του αγωγού γίνεται σε έτερο αποθετήριο, στο οποίο απλοί χρήστες δεν έχουν πρόσβαση
Αποθήκευση συνθηματικών σε μορφή κειμένου		Στον αγωγό εκτελείται η ενέργεια ‘ Ανίχνευση Μυστικών ’, η οποία απορρίπτει κώδικα που περιέχει μυστικά.
Άρνηση υπηρεσίας		Η πλατφόρμα έχει σκληρυνθεί ενάντια σε επιθέσεις εξαντλήσεως πόρων (Ενότητα 5.2.2.2), ενώ παράλληλα το δικαίωμα εκτέλεσης αγωγού ανήκει μόνο σε εξουσιοδοτημένο χρήστη. Παράλληλα, γίνεται χρήση IPS που μπορεί να αποτρέψει μια τέτοια επίθεση.
Εκτέλεση και εγκατάσταση παλαιού στιγμιότυπου του κώδικα		Η πλατφόρμα έχει σκληρυνθεί ενάντια σε επιθέσεις επανεκτέλεσης παλαιών αγωγών (εν. 5.2.2.2). Η προστασία αυτή παρέχεται μετά από ενέργεια του χρήστη.
Τροποποίηση του αγωγού/συνθηματικών από λογαριασμό διαχειριστή		Για την είσοδο στην πλατφόρμα Gitlab μπορεί να απαιτηθεί αυθεντικοποίηση πολλαπλών σταδίων (MFA), σε συνδυασμό με χρήση RBAC.

34 <https://github.com/istio/istio>

Θέαση συνθηματικών πρόσβασης σε τρίτα συστήματα από απλούς χρήστες		Τα μυστικά δεν είναι προσβάσιμα από απλούς χρήστες λόγω λόγω ικανοτήτων αυθεντικοποίησης & RBAC, ενώ δεν εμφανίζονται σε κανένα αρχείο καταγραφής, παρά μόνο σε διαχειριστές.
Εκτέλεση απομακρυσμένου κώδικα στον εξυπηρετητή CI/CD		Η πλατφόρμα Gitlab εκτελείται ως δοχείο μέσα από ένα Docker-Compose.yaml αρχείο. Η αναβάθμιση της έκδοσής του γίνεται με απλή αλλαγή μιας γραμμής στον εν λόγω αρχείο και επανεκτέλεση του δοχείου.
Φόρτωση κακόβουλων εκτελέσιμων/βιβλιοθηκών στο αποθετήριο		Για την είσοδο στην πλατφόρμα Gitlab απαιτείται αυθεντικοποίηση πολλαπλών σταδίων (MFA), ενώ για την υποβολή τεχνημάτων στο αποθετήριο τηρείται η αρχή του ελάχιστου δικαιώματος, με τον κάθε χρήστη να έχει πρόσβαση στα αποθετήρια που δικαιούται, σύμφωνα με το ρόλο του.
Χρήση κακόβουλου κώδικα από τρίτες βιβλιοθήκες		Ο αγωγός απορρίπτει ευπαθείς/ευάλωτες βιβλιοθήκες/πακέτα κατά την εκτέλεση της ενέργειας ‘Έλεγχος Εξαρτήσεων’.
Έγχυση κακόβουλων βιβλιοθηκών		Ο αγωγός καλύπτει όλες τις περιπτώσεις έγχυσης κακόβουλων βιβλιοθηκών μέσα από την πιστοποίηση της εγκυρότητας βιβλιοθηκών του εργαλείου maven.
Μη εξουσιοδοτημένη πρόσβαση στο δοκιμαστικό/παραγωγικό περιβάλλον		Το περιβάλλον που στεγάζει τη συστοιχία Kubernetes επιτρέπει την πρόσβαση μόνο μέσω SSH κλειδιών. Επιπρόσθετα, άλλες υλοποιήσεις Kubernetes που δεν εγκαταστάθηκαν λόγω περιορισμού πόρων (π.χ Rancher, Openshift) προσφέρουν και RBAC για την πρόσβαση στα διάφορα αντικείμενα της συστοιχίας.
Υποβολή κακόβουλου κώδικα από εξουσιοδοτημένο χρήστη		Όλα τα αιτήματα συγχώνευσης στο κύριο κλαδί εγκρίνονται από εξουσιοδοτημένο χρήστη χειροκίνητα, με εποπτεία του νέου κώδικα. Επιπρόσθετα, αγωγός μπορεί να εντοπίσει τυχόν κακόβουλο κώδικα και να αναχαιτίσει τη συγχώνευση.

Υποβολή ευάλωτου κώδικα από εξουσιοδοτημένο χρήστη		Τυχόν ευάλωτος κώδικας/βιβλιοθήκες θα απορριφθούν κατά τις ενέργειες ‘ Στατική Ανάλυση Ασφάλειας ’ και ‘ Έλεγχος Εξαρτήσεων ’.
Ευάλωτοι εξυπηρετητές		Το λειτουργικό σύστημα των εικονικών μηχανών λειτουργεί με βάση το μηχανισμό προστασίας SELinux, έχει σκληρυνθεί κάνοντας χρήση του λογισμικού Lynis, προστατεύεται με σαρωτή κακόβουλου λογισμικού ClamAV και παρακολουθείται για ασυνήθιστες διεργασίες από το Falco. Ταυτόχρονα, προστατεύεται από το εργαλείο IPS Crowdsec.
<u>Ειδικές απειλές ενάντια σε δοχεία</u>		
Απειλή	Προστασία	Δικαιολόγηση
Επικίνδυνα διαμορφωμένες εικόνες δοχείων		Το υπό διαμόρφωση δοχείο είναι ασφαλές εφόσον περάσει επιτυχώς από το στάδιο ‘ Έλεγχος Εικόνας ’, το οποίο εγγυάται ότι τηρούνται όλες οι ασφαλείς προδιαγραφές που έχει θέσει ο διαχειριστής.
Ευάλωτος εξυπηρετητής κατά την οικοδόμηση του δοχείου		Το λειτουργικό σύστημα των εικονικών μηχανών λειτουργεί με βάση το μηχανισμό προστασίας SELinux, έχει σκληρυνθεί κάνοντας χρήση του λογισμικού Lynis, προστατεύεται με σαρωτή κακόβουλου λογισμικού ClamAV και παρακολουθείται για ασυνήθιστες διεργασίες από το Falco. Ταυτόχρονα, προστατεύεται από το εργαλείο IPS Crowdsec.
Επίθεση εφοδιαστικής αλυσίδας		Οι εικόνες που παράγονται από τον αγωγό υπογράφονται με ιδιωτικό κλειδί, ενώ η συστοιχία Kubernetes δεν επιτρέπει την εγκατάσταση εικόνων που δεν επαληθεύονται με το αντίστοιχο δημόσιο κλειδί. Επιπρόσθετα, η πρόσβαση στο αποθετήριο προστατεύεται με σύστημα ρόλων και εξουσιοδότησης.
Επικίνδυνες ρυθμίσεις κατά την εκτέλεση του δοχείου		Όλες οι επικίνδυνες ρυθμίσεις που περιγράφονται στην Ενότητα 4.2.4 οδηγούν σε αποτυχία κατά την εκτέλεση του αγωγού αλλά και κατά την εγκατάσταση από τους κανόνες που ορίστηκαν στο εργαλείο Kyverno

Εκτεθειμένα μυστικά/συνθηματικά		Κατά την ενέργεια ‘Ανίχνευση Μυστικών’, εάν ανιχνευτούν αρχεία που περιέχουν μυστικά, ο αγωγός οδηγείται σε έξοδο με σφάλμα
Μη ασφαλής δικτύωση		Μια από τις ρυθμίσεις της ενέργειας ‘Έλεγχος ρυθμίσεων εγκατάστασης δοχείων’ είναι η απαίτηση για ύπαρξη κανόνων επιτρεπτής κίνησης (Network Policy), για τους οποίους υπεύθυνος είναι ο χρήστης

7.2 Αποτίμηση & Μελλοντική εργασία

Με την αποτίμηση της προηγούμενης ενότητας έγινε φανερό ότι το σύστημα CI/CD της παρούσας διπλωματικής εργασίας μπορεί να θεωρηθεί ασφαλές. Αυτό οφείλεται στη μεγάλη απόσταση που καλύφθηκε από την εκκίνηση της εκπόνησης μέχρι τη συγγραφή του παρόντος κεφαλαίου, καθώς απαιτήθηκε βαθιά έρευνα τόσο στα αίτια που καθιστούν τα συστήματα CI/CD ευάλωτα, είτε αυτά είναι τεχνικά είτε ανθρωποκεντρικά, όσο και στις πλέον σύγχρονες τάσεις και εργαλεία σκλήρυνσης των διαδικασιών και τεχνημάτων.

Με την μεθοδική προσέγγιση και τον συγκερασμό όλων των ετερόκλητων τεχνολογιών που απαιτήθηκαν, υλοποιήθηκε το προϊόν της παρούσας Διπλωματικής Εργασίας, από το οποίο αποκομίσθηκαν μεγάλα οφέλη σε γνώση και εμπειρία στον χώρο του CI/CD.

Ένα τμήμα που δεν ήταν δυνατόν να εισαχθεί χρονικά στο πλαίσιο της Διπλωματικής εργασίας ενδεχομένως να ήταν η επέκταση του συστήματος ώστε να μπορεί να παράγει εφήμερα εικονικά μηχανήματα σε συνδυασμό με την παραμετροποίηση αυτών χρησιμοποιώντας τεχνολογίες όπως Ansible και Terraform, ώστε να υπάρχει απεριόριστη οριζόντια κλιμάκωση πόρων για παρεχόμενες υπηρεσίες. Έτερη κατεύθυνση θα ήταν η δημιουργία επιπρόσθετων αγωγών για εφαρμογές βασισμένες σε διαφορετικές γλώσσες/εργαλεία προγραμματισμού, καθώς και η πλήρης έξωθεν παραμετροποίηση των ορίων αποδοχής του αγωγού, ο οποίος είναι ήδη σε μεγάλο βαθμό παραμετρικός, χωρίς να απαιτείται σε κανέναν σημείο παρέμβαση στον κώδικα του λογισμικού.

Ένα τμήμα που θα μπορούσε να συμπληρώσει τον αγωγό της εργασίας θα ήταν η διενέργεια Δυναμική ανάλυσης λογισμικού (DAST) κάνοντας χρήση εργαλείων όπως του OWASP Zap, το οποίο δεν υλοποιήθηκε λόγω ανεπάρκειας πόρων.

Τέλος, θα μπορούσε να παρασχεθεί ο αγωγός σε μορφή προτύπου, όσο και οι ενέργειες σε μορφή προτύπου, για χρήση από οποιοδήποτε αποθετήριο.

Σε κάθε περίπτωση, οι εντατικοί μήνες που δαπανήθηκαν για τη μελέτη και υλοποίηση αυτού του συστήματος αποτέλεσαν ένα μακρύ ταξίδι γεμάτο εμπειρία, και είναι σίγουρο ότι θα συνεχίσει να είναι παραγωγικό και ωφέλιμο για το υπόλοιπο της σταδιοδρομίας του συγγραφέα.

Βιβλιογραφία

- [1] W. Royce, “Managing the Development of Large Software Systems,” in *Proceedings of IEEE WESCON*, 1970, vol. 26, pp. 328–388.
- [2] J. Highsmith and A. Cockburn, “Agile software development: the business of innovation,” *Computer*, vol. 34, no. 9, pp. 120–127, 2001, doi: 10.1109/2.947100.
- [3] D. Leffingwell, *Agile Software Requirements: Lean Requirements Practices for Teams, Programs, and the Enterprise*. Addison-Wesley Professional, 2010.
- [4] L. W. Khong, L. Y. Beng, T. W. Yip, and T. Soofun, “Software Development Life Cycle AGILE vs Traditional Approaches,” *unknown*, Feb. 25, 2012.
https://www.researchgate.net/publication/268334807_Software_Development_Life_Cycle_AGILE_vs_Traditional_Approaches
- [5] L. Bass, I. Weber, and L. Zhu, *Devops: A Software Architect’s Perspective*. Addison-Wesley Professional, 2015.
- [6] J. Wettinger, U. Breitenbücher, and F. Leymann, “DevOpSlang – Bridging the Gap between Development and Operations,” *Springer Berlin Heidelberg*, Jan. 01, 2014.
https://link.springer.com/chapter/10.1007/978-3-662-44879-3_8 (accessed Aug. 24, 2023).
- [7] G. Kim, J. Humble, P. Debois, J. Willis, and N. Forsgren, *The DevOps Handbook: How to Create World-Class Agility, Reliability, & Security in Technology Organizations*. IT Revolution, 2021.
- [8] J. McKendrick, “DevOps adoption almost doubles in five years, Covid crisis accelerated adoption,” *ZDNET*, Feb. 26, 2021. Accessed: Aug. 24, 2023. [Online]. Available: <https://www.zdnet.com/article/devops-adoption-almost-doubles-in-five-years-covid-crisis-accelerates/>
- [9] OWASP, “OWASP Top 10 CI/CD SEcurity Risks,” 2020.
- [10] T. H.-C. Hsu, *Hands-On Security in DevOps: Ensure continuous security, deployment, and delivery with DevSecOps*. Packt Publishing Ltd, 2018.
- [11] S. J. Bigelow, “CI/CD pipelines explained: Everything you need to know,” *TechTarget*, May 13, 2021. Accessed: Aug. 25, 2023. [Online]. Available:

<https://www.techtarget.com/searchsoftwarequality/CI-CD-pipelines-explained-Everything-you-ne-ed-to-know>

[12] F. Lombardi and A. Fanton, “From DevOps to DevSecOps is not enough. CyberDevOps: an extreme shifting-left architecture to bring cybersecurity within software security lifecycle pipeline,” *Software Quality Journal*, vol. 31, no. 2, pp. 619–654, Apr. 2023, doi: 10.1007/s11219-023-09619-3.

[13] P. Debois, “Agile Infrastructure and Operations: How Infra-gile are You?,” in *Agile 2008 Conference*, 2008. Accessed: Aug. 26, 2023. [Online]. Available: <http://dx.doi.org/10.1109/agile.2008.42>

[14] R. Jabbari, N. bin Ali, K. Petersen, and B. Tanveer, “What is DevOps?,” in *Proceedings of the Scientific Workshop Proceedings of XP2016*, May 2016. Accessed: Aug. 26, 2023. [Online]. Available: <http://dx.doi.org/10.1145/2962695.2962707>

[15] J. Bunnell, “Team Collaboration Strategies for Hybrid DevOps Teams,” *Semaphore*, Mar. 29, 2023. <https://semaphoreci.com/blog/hybrid-devops-collaboration-strategies> (accessed Aug. 26, 2023).

[16] M. Virmani, “Understanding DevOps & bridging the gap from continuous integration to continuous delivery,” in *Fifth International Conference on the Innovative Computing Technology (INTECH 2015)*, May 2015. Accessed: Aug. 27, 2023. [Online]. Available: <http://dx.doi.org/10.1109/intech.2015.7173368>

[17] “What is CI/CD?” <https://www.redhat.com/en/topics/devops/what-is-ci-cd> (accessed Aug. 27, 2023).

[18] S. Deochake, S. Maheshwari, R. De, and A. Grover, “Comparative Study of Virtual Machines and Containers for DevOps Developers,” *SSRN Electronic Journal*, 2023, doi: 10.2139/ssrn.4404383.

[19] H. Myrbakken and R. Colomo-Palacios, “DevSecOps: A Multivocal Literature Review,” in *Communications in Computer and Information Science*, Cham: Springer International Publishing, 2017, pp. 17–29. Accessed: Aug. 29, 2023. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-67383-7_2

[20] F. Lombardi and A. Fanton, “From DevOps to DevSecOps is not enough. CyberDevOps: an extreme shifting-left architecture to bring cybersecurity within software security lifecycle pipeline,” *Software Quality Journal*, vol. 31, no. 2, pp. 619–654, Apr. 2023, doi: 10.1007/s11219-023-09619-3.

- [21] “Continuous Integration Tools Market Size & Share Analysis - Industry Research Report,” *Growth Trends*.
<https://www.mordorintelligence.com/industry-reports/continuous-integration-tools-market>
(accessed Aug. 30, 2023).
- [22] O. Bedrina, “Best Continuous Integration Tools for 2023 – Survey Results,” *The JetBrains Blog*, Jul. 07, 2023. <https://blog.jetbrains.com/teamcity/2023/07/best-ci-tools/> (accessed Aug. 30, 2023).
- [23] “GitHub: Let’s build from here,” *GitHub*. <https://github.com/> (accessed Aug. 30, 2023).
- [24] “Jenkins,” *Jenkins*. <https://www.jenkins.io/> (accessed Aug. 30, 2023).
- [25] “GitLab,” *GitLab*. <https://gitlab.com/> (accessed Aug. 30, 2023).
- [26] “DevOps Solutions,” *Microsoft Azure*. <https://azure.microsoft.com/en-us/solutions/devops>
(accessed Aug. 31, 2023).
- [27] “Continuous Integration and Delivery,” *CircleCI*. <https://circleci.com/> (accessed Aug. 31, 2023).
- [28] Contributors to Wikimedia projects, “YAML,” *Wikipedia*, Jul. 31, 2023.
<https://en.wikipedia.org/wiki/YAML> (accessed Aug. 31, 2023).
- [29] “YAML Ain’t Markup Language (YAMLTm) Version 1.1.” <https://yaml.org/spec/1.1/>
(accessed Aug. 31, 2023).
- [30] C. Cowell, N. Lotz, and C. Timberlake, *Automating DevOps with GitLab CI/CD Pipelines: Build efficient CI/CD pipelines to verify, secure, and deploy your code using real-life examples*. Packt Publishing Ltd, 2023.
- [31] D. H. A. Gonçalves, “DevSecOps for web applications: a case study,” Jan. 01, 2022.
https://www.oasisbr.ibict.br/vufind/Record/RCAP_1a577f52263b5a69deaf891daa1d6945
- [32] H. Saarinen, “Security Activities Integrated into DevOps Software Development and Operation Processes,” 2022.
https://www.theseus.fi/bitstream/handle/10024/781693/Saarinen_Helena.pdf
- [33] Md. S. I. Shamim, F. A. Bhuiyan, and A. Rahman, “XI Commandments of Kubernetes Security: A Systematization of Knowledge Related to Kubernetes Security Practices,” *arXiv.org*, Jun. 27, 2020. <https://arxiv.org/abs/2006.15275>
- [34] F. Minna, A. Blaise, F. Rebecchi, B. Chandrasekaran, and F. Massacci, “Understanding the Security Implications of Kubernetes Networking,” *IEEE Security & Privacy*, vol. 19, no. 5, pp. 46–56, Sep. 2021, doi: 10.1109/msec.2021.3094726.

- [35] N. Pecka, L. ben Othmane, and A. Valani, “Making Secure Software Insecure without Changing Its Code: The Possibilities and Impacts of Attacks on the DevOps Pipeline,” *arXiv.org*, Jan. 30, 2022. <https://arxiv.org/abs/2201.12879>
- [36] F. Lombardi and A. Fanton, “From DevOps to DevSecOps is not enough. CyberDevOps: an extreme shifting-left architecture to bring cybersecurity within software security lifecycle pipeline,” *Software Quality Journal*, vol. 31, no. 2, pp. 619–654, Apr. 2023, doi: 10.1007/s11219-023-09619-3.
- [37] R. Desai and T. N. Nisha, “Best Practices for Ensuring Security in DevOps: A Case Study Approach,” *Journal of Physics: Conference Series*, vol. 1964, no. 4, p. 042045, Jul. 2021, doi: 10.1088/1742-6596/1964/4/042045.
- [38] A. A. Ur Rahman and L. Williams, “Software security in DevOps,” in *Proceedings of the International Workshop on Continuous Software Evolution and Delivery*, May 2016. Accessed: Sep. 24, 2023. [Online]. Available: <http://dx.doi.org/10.1145/2896941.2896946>
- [39] L. Bass, R. Holz, P. Rimba, A. B. Tran, and L. Zhu, “Securing a Deployment Pipeline,” in *2015 IEEE/ACM 3rd International Workshop on Release Engineering*, May 2015. Accessed: Sep. 24, 2023. [Online]. Available: <http://dx.doi.org/10.1109/releng.2015.11>
- [40] R. Mao *et al.*, “Preliminary Findings about DevSecOps from Grey Literature,” in *2020 IEEE 20th International Conference on Software Quality, Reliability and Security (QRS)*, Dec. 2020. Accessed: Sep. 24, 2023. [Online]. Available: <http://dx.doi.org/10.1109/qrs51102.2020.00064>
- [41] M. Chernyshev, Z. Baig, and S. Zeadally, “Cloud-Native Application Security: Risks, Opportunities, and Challenges in Securing the Evolving Attack Surface,” *Computer*, vol. 54, no. 11, pp. 47–57, Nov. 2021, doi: 10.1109/mc.2021.3076537.
- [42] D. S. Battina, “The Challenges and Mitigation Strategies of Using DevOps during Software Development.” https://papers.ssrn.com/sol3/papers.cfm?abstract_id=4004335
- [43] Z. Ahmed and Shoba. C. Francis, “Integrating Security with DevSecOps: Techniques and Challenges,” in *2019 International Conference on Digitization (ICD)*, Nov. 2019. Accessed: Sep. 24, 2023. [Online]. Available: <http://dx.doi.org/10.1109/icd47981.2019.9105789>
- [44] V. Casola, A. De Benedictis, M. Rak, and U. Villano, “A novel Security-by-Design methodology: Modeling and assessing security by SLAs with a quantitative approach,” *Journal of Systems and Software*, vol. 163, p. 110537, May 2020, doi: 10.1016/j.jss.2020.110537.
- [45] T. Rangnau, R. v. Buijtenen, F. Fransen, and F. Turkmen, “Continuous Security Testing: A

- Case Study on Integrating Dynamic Security Testing Tools in CI/CD Pipelines,” in *2020 IEEE 24th International Enterprise Distributed Object Computing Conference (EDOC)*, Oct. 2020. Accessed: Sep. 25, 2023. [Online]. Available: <http://dx.doi.org/10.1109/edoc49727.2020.00026>
- [46] A. Sojan, R. Rajan, and P. Kuvaja, “Monitoring solution for cloud-native DevSecOps,” in *2021 IEEE 6th International Conference on Smart Cloud (SmartCloud)*, Nov. 2021. Accessed: Sep. 25, 2023. [Online]. Available: <http://dx.doi.org/10.1109/smartcloud52277.2021.00029>
- [47] V. Singh, “DEVELOPING A CI/CD PIPELINE WITH GITLAB,” Turku University of Applied Sciences, 2022.
- [48] F. Ullah, A. J. Raft, M. Shahin, M. Zahedi, and M. Ali Babar, “Security Support in Continuous Deployment Pipeline,” in *Proceedings of the 12th International Conference on Evaluation of Novel Approaches to Software Engineering*, 2017. Accessed: Sep. 25, 2023. [Online]. Available: <http://dx.doi.org/10.5220/0006318200570068>
- [49] M. Koopman, “A Framework for Detecting and Preventing Security Vulnerabilities in Continuous Integration/Continuous Delivery pipelines,” University of Twente, 2019.
- [50] C. Paule, “Securing DevOps — Detection of vulnerabilities in CD pipelines,” University of Stuttgart, 2018.
- [51] B. K. Pham and S. Chalishhafshejani, “Automated software security activities in a continuous delivery pipeline,” University of Stavanger, 2021.
- [52] A. Škurla, “DevOps Integration of Security Practices,” Masaryk University, 2020.
- [53] K. Brady, S. Moon, T. Nguyen, and J. Coffman, “Docker Container Security in Cloud Computing,” in *2020 10th Annual Computing and Communication Workshop and Conference (CCWC)*, Jan. 2020. Accessed: Sep. 26, 2023. [Online]. Available: <http://dx.doi.org/10.1109/ccwc47524.2020.9031195>
- [54] V. Gruhn, C. Hannebauer, and C. John, “Security of public continuous integration services,” in *Proceedings of the 9th International Symposium on Open Collaboration*, Aug. 2013. Accessed: Sep. 26, 2023. [Online]. Available: <http://dx.doi.org/10.1145/2491055.2491070>
- [55] P. Rimba, L. Zhu, L. Bass, I. Kuz, and S. Reeves, “Composing Patterns to Construct Secure Systems,” in *2015 11th European Dependable Computing Conference (EDCC)*, Sep. 2015. Accessed: Sep. 26, 2023. [Online]. Available: <http://dx.doi.org/10.1109/edcc.2015.12>
- [56] L. Rice, *Container Security: Fundamental Technology Concepts that Protect Containerized Applications*. O’Reilly Media, 2020.
- [57] teamanchore, “Docker Security Best Practices: A Complete Guide • Anchore,” *Anchore*,

May 24, 2022. <https://anchore.com/blog/docker-security-best-practices-a-complete-guide/> (accessed Sep. 29, 2023).

[58] “Docker.” <https://hub.docker.com/> (accessed Sep. 30, 2023).

[59] Z. Jian and L. Chen, “A Defense Method against Docker Escape Attack,” in *Proceedings of the 2017 International Conference on Cryptography, Security and Privacy*, Mar. 2017. Accessed: Sep. 30, 2023. [Online]. Available: <http://dx.doi.org/10.1145/3058060.3058085>

[60] ““Content trust in Docker,”” *Docker Documentation*, Aug. 22, 2023. <https://docs.docker.com/engine/security/trust/> (accessed Sep. 30, 2023).

[61] “Chapter 8. Linux Capabilities and Seccomp Red Hat Enterprise Linux Atomic Host 7,” *Red Hat Customer Portal*. https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux_atomic_host/7/html/container_security_guide/linux_capabilities_and_seccomp (accessed Sep. 30, 2023).

[62] A. Chourasia, “Understanding How Containers Work Behind the Scenes,” *Kubesimplify*, Apr. 29, 2023. Accessed: Sep. 30, 2023. [Online]. Available: <https://blog.kubesimplify.com/understanding-how-containers-work-behind-the-scenes>

[63] “What is a WAF?,” *Cloudflare*. <https://www.cloudflare.com/learning/ddos/glossary/web-application-firewall-waf/> (accessed Oct. 01, 2023).

[64] N. Popov, “php.internals: Update on git.php.net incident.” <https://news-web.php.net/php.internals/113981> (accessed Oct. 01, 2023).

[65] L. Lingo, “10 Reasons Why PHP is Still Relevant in 2023 - London Lingo,” *Medium*, May 19, 2023. Accessed: Oct. 01, 2023. [Online]. Available: <https://medium.com/@london.lingo.01/10-reasons-why-php-is-still-relevant-in-2023-d533ed6dc687>

[66] reitermarkus, “Security Incident Disclosure,” *reitermarkus*, Apr. 21, 2021. <https://brew.sh/2021/04/21/security-incident-disclosure/> (accessed Oct. 01, 2023).

[67] C. Cimpanu, “Mercedes-Benz onboard logic unit (OLU) source code leaks online,” *ZDNET*, May 18, 2020. Accessed: Oct. 01, 2023. [Online]. Available: <https://www.zdnet.com/article/mercedes-benz-onboard-logic-unit-olu-source-code-leaks-online/>

[68] “GHSA-g2q5-5433-rhrf - GitHub Advisory Database,” *GitHub*. <https://github.com/advisories/GHSA-g2q5-5433-rhrf> (accessed Oct. 02, 2023).

[69] Z. Whittaker, “Samsung spilled SmartThings app source code and secret keys,” *Join*

TechCrunch+, May 08, 2019. <https://techcrunch.com/2019/05/08/samsung-source-code-leak/>

[70] C. Cimpanu, “Nissan source code leaked online after Git repo misconfiguration,” *ZDNET*, Jan. 06, 2021. Accessed: Oct. 02, 2023. [Online]. Available:

<https://www.zdnet.com/article/nissan-source-code-leaked-online-after-git-repo-misconfiguration/>

[71] N. Patel, “Hard lessons of the SolarWinds hack,” *The Verge*, Jan. 26, 2021. Accessed: Oct. 02, 2023. [Online]. Available:

<https://www.theverge.com/2021/1/26/22248631/solarwinds-hack-cybersecurity-us-menn-decoder-podcast>

[72] A. Even-Zohar, “The Codecov Breach - Development Infrastructure is the Weakest Link & its Now Rapidly Being Exploited,” *Cycode*, Apr. 20, 2021.

<https://cycode.com/blog/the-codecov-breach-development-infrastructure-is-the-weakest-link-its-now-rapidly-being-exploited/> (accessed Oct. 02, 2023).

[73] L. Sharma, “10 nonfunctional requirements to consider in your enterprise architecture,” *Enable Architect*, Aug. 04, 2022.

<https://www.redhat.com/architect/nonfunctional-requirements-architecture> (accessed Oct. 02, 2023).

[74] “Nonfunctional requirements: A checklist,” *IBM Cloud Architecture Center*.

<https://www.ibm.com/cloud/architecture/architecture/practices/nonfunctional-requirements-checklist/> (accessed Oct. 02, 2023).

[75] D. Mangot, “Why you should burn your IT ticketing system,” *TechBeacon*, Oct. 08, 2020. Accessed: Oct. 05, 2023. [Online]. Available:

<https://techbeacon.com/enterprise-it/why-you-should-burn-your-ticketing-system>

[76] C. Romeo, “Why developers dislike security—and what you can do about it,” *TechBeacon*, May 18, 2021. Accessed: Oct. 05, 2023. [Online]. Available:

<https://techbeacon.com/security/why-developers-dislike-security-what-you-can-do-about-it>

[77] N. Azad and S. Hyrynsalmi, “DevOps critical success factors — A systematic literature review,” *Information and Software Technology*, vol. 157, p. 107150, May 2023, doi: 10.1016/j.infsof.2023.107150.

[78] L. Chen, “Continuous Delivery: Overcoming adoption challenges,” *Journal of Systems and Software*, vol. 128, pp. 72–86, Jun. 2017, doi: 10.1016/j.jss.2017.02.013.

[79] B. Evans, “Microsoft’s Killer Acquisition: GitHub Soars 70%,” *Acceleration Economy*, May 06, 2021. Accessed: Oct. 05, 2023. [Online]. Available:

<https://accelerationeconomy.com/cloud/microsofts-killer-acquisition-github-soars-70/>

[80] “Who are the top Git competitors?,” *6sense*.

<https://6sense.com/tech/version-control/git-market-share> (accessed Oct. 15, 2023).

[81] R. Wieruch, “How to Git as a Team,” Aug. 22, 2019.

<https://www.robinwieruch.de/git-team-workflow/> (accessed Oct. 15, 2023).

[82] P. Gupta, “On Virtualization and Containers - Pranshu Gupta,” *Medium*, Mar. 03, 2019.

Accessed: Oct. 15, 2023. [Online]. Available:

https://medium.com/@pranshugupta_61104/on-virtualization-and-containers-6f9683fe1c1

[83] S. Sultan, I. Ahmad, and T. Dimitriou, “Container Security: Issues, Challenges, and the Road Ahead,” *IEEE Access*, vol. 7, pp. 52976–52996, 2019, doi: 10.1109/access.2019.2911732.

[84] “What is Software Composition Analysis?,” *Sonatype*.

<https://www.sonatype.com/launchpad/what-is-software-composition-analysis> (accessed Oct. 16, 2023).

[85] “Linux post-installation steps for Docker Engine,” *Docker Documentation*, Oct. 20, 2023. <https://docs.docker.com/engine/install/linux-postinstall/> (accessed Nov. 23, 2023).

[86] R. McCune, “Container security fundamentals part 6: seccomp,” *Datadog Security Labs*, Sep. 11, 2023. <https://securitylabs.datadoghq.com/articles/container-security-fundamentals-part-6/> (accessed Dec. 18, 2023).

[87] Datadog, “The container’s root filesystem should be set to read-only,” *Datadog Infrastructure and Application Monitoring*.

https://docs.datadoghq.com/security/default_rules/cis-docker-1.2.0-5.12/ (accessed Dec. 21, 2023).

[88] Frankpromiseedah, “Kubernetes SecurityContext Capabilities - Frankpromiseedah,” *Medium*, Oct. 02, 2023. Accessed: Dec. 21, 2023. [Online]. Available:

<https://medium.com/@frankpromiseedah/kubernetes-securitycontext-capabilities-b343f859c633>

[89] H. Dhaduk, “Kubernetes Monitoring: Metrics, Tools and Best Practices,” *Simform - Product Engineering Company*, Sep. 06, 2022.

<https://www.simform.com/blog/kubernetes-monitoring/> (accessed Dec. 22, 2023).

[90] Navin, “Secret Scanner Comparison: Finding Your Best Tool - Navin,” *Medium*, Jan. 15, 2024. Accessed: Jan. 20, 2024. [Online]. Available:

<https://medium.com/@navinwork21/secret-scanner-comparison-finding-your-best-tool-ed899541b9b6>

[91] O. Pochapsky, “Kubernetes fault tolerance mechanisms - Oleksandr Pochapsky,” *Medium*, Apr. 05, 2023. Accessed: Jan. 22, 2024. [Online]. Available: <https://medium.com/@olexandr.pochapskiy/building-fault-tolerant-application-in-kubernetes-cce31c485f34>