SIP Message Tampering
THE SQL code INJECTION attack

Dimitris Geneiatakis, Georgios Kambourakis, Costas Lambrinoudakis,
Tasos Dagiuklas and Stefanos Gritzalis

Laboratory of Information and Communication Systems Security
Department of Information and Communication Systems Engineering
University of the Aegean, Karlovassi, GR-83200 Samos, Greece
Tel:+30-22730-82247
Fax: +30-22730-82009
Email:{dgen,gkamb,ntan,clam,sgritz}@aegean.gr

Abstract – **As Internet Telephony and Voice over IP (VoIP) are considered advanced Internet applications/services, they are vulnerable to attacks existing in Internet applications/services. For instance HTTP digest authentication attacks, malformed messages, message tampering with malicious code, SQL injection and more, can be launched against any Internet application/service. In this paper, we describe, analyze and demonstrate the inheritance of message tampering attacks, focusing on SQL injection, in the SIP protocol. This type of attack has been successfully launched in Internet environments, with very little cost, effort and specialized knowledge. However, in the context of the SIP protocol, no works or research efforts are reported until now. The paper provides an in-depth analysis of SQL injection in SIP realms, discussing implementation details, constraints and possibilities for the attacker. In addition, we provide some indicative experimental results by triggering this style of attack against a properly designed SIP-based test-bed environment. Finally, specific countermeasures, remedies and new signature-oriented framework are suggested for identifying and counter fight against this attack.**

I. INTRODUCTION

The advent and proliferation of the Internet Telephony-Voice over IP (VoIP) promises new evolutionary telephony services. Telecommunication and Internet Telephony Service Providers (ITSPs) enjoy several advantages [15], thus being able to offer advanced telephony services like *instant messaging, Internet conferencing rooms, personalized call transfer, etc.* On the other hand, the Public Switch Telephone Network (PSTN) is dedicated to a single service (namely voice) considered to be reliable and secure. As a result, VoIP must ensure at least the same level of reliability, availability and security as in the PSTN.

PSTN is mainly based on a closed network. On the contrary, Internet is an open network thus being vulnerable to various security threats. Consequently, VoIP inherits all these problems. In addition, the utilization of VoIP signaling standards like Session Initiation Protocol (SIP) [1], H.323 [19] and MGCP/MEGACO [19] combined with the Internet distributed architecture creates a hostile environment, as aggressors can exploit the aforementioned standards in order to launch several types of attacks, targeting to compromise VoIP security.

SIP seems to overwhelm other standards, mainly due to the fact that it has been adopted by various standardization organizations (i.e. IETF, ETSI, 3GPP) as the protocol for both wireline and wireless world in the Next Generation Networks (NGN) era. SIP similarities with HTTP enable malicious users to attack VoIP services with the same methods that HTTP applications are being attacked. For instance, HTTP digest authentication attacks [7], SQL injection [11] or message tampering with malicious code [21], are attacks that can be launched in a SIP-based network infrastructure to compromise the provided service.

The aim of this paper is to introduce, describe and analyse an attack, known as *SQL injection,* in a SIP-based network architecture. We name this attack *SIP message tampering (or injection) with SQL code*. Although, various injection style attacks have been successfully launched over Internet environments, to the best of our knowledge, no SIP-oriented specific case have been reported until now. In this context, the paper provides an in-depth analysis of SIP-oriented SQL injection, discussing implementation details, constraints and possibilities for the attacker to exploit. Moreover, the paper introduces an initial signature-based framework that can assist the identification of this attack.

The rest of the paper is structured as follows: Section II briefly presents background information for SQL injection in the Internet and an overview of the SIP protocol. Section III describes the SIP database infrastructure, which is the most prominent component regarding this attack. Section IV focuses on SIP message tampering with SQL code, introducing some counter measures for protection and a signature-based scheme capable of identifying this kind of threat. Last section concludes the paper and gives pointers to future work.

II. BACKGROUND

A. SQL injection

In order to administer dynamic content, Web applications utilize server side script technologies like PHP, ASP, JSP and databases. Script technologies are able to transfer data from a database to a Web page and vice versa utilizing the aforementioned technologies. This architecture is not employed only for data administration, but also for providing security services such as authentication to the corresponding application. Authentication services that utilize these technologies employ a data-entry form for the username, the password and the appropriate server side script responsible to handle user's input. The server side script to be able to authenticate the user includes SQL statements like the following:

SELECT Last_Name FROM subscriber WHERE User_ID='<username>' AND User_Password='<password>';

Therefore, whenever a legitimate user submits a valid username (e.g. dim) and his corresponding password (for example: !#$), the server side script generates the following SQL statement:

SELECT Last_Name FROM subscriber WHERE User_ID='dim' AND User_Password='!#$';

Note, that the input data (User_ID & Password field) have not been validated. Consequently, a perpetrator can exploit this vulnerability by injecting malicious SQL code to the input data. This attack is known as *SQL injection* [11]. For example, the attacker can complement the User_ID field in the equivalent Web form with the following SQL code:

"'--; DROP TABLE subscriber"

The resulting SQL statement will look like:

SELECT Last_Name FROM subscriber WHERE User_ID='' -- AND User_Password='<password>';

DROP TABLE subscriber;

In this example, two SQL statements will be executed towards the database: the SELECT and DROP statements. The execution of the SELECT statement fails as the User_ID value is null and the characters '- -' are commenting out the rest of the SELECT statement. However, it is clear, that the latter statement will delete the Subscriber table. Several methods have been proposed to identify such vulnerabilities and provide remedies for input verification. A detailed analysis on WWW SQL injection attacks can be found in [11],[12],[13],[14]. Before addressing this vulnerability in the context of a SIP-based network, an overview of the SIP architecture is provided here below.

B. Overview of the SIP architecture

SIP is an application-layer signaling protocol [1] for handling multimedia sessions over the Internet. In a typical SIP-based network infrastructure, the following subsystems are involved:

- Registrar: The Registrar server is responsible for user registration in VoIP services (e.g. instant messaging, presence).
- Proxy: The proxy server is responsible either to deliver a SIP message to the callee or forward the message to another proxy.
- Redirect: The Redirect server is responsible to inform a registered user to connect directly to another proxy or to the registrar server.

In order to enable communication among users, SIP introduces various types of messages similarly to the HTTP message structure. SIP messages must identify the requested resource, which corresponds to a unique address. SIP addresses follow the general form of the HTTP addressing scheme that is: "address_scheme:resource". An example of a SIP address is: 'sip:dgen@aegean.gr'. A typical SIP message, depicted in Figure 1, consists of a request or status line followed by header fields and a message body.

```
INVITE sip:dgen@aegean.gr  SIP/2.0
To: Geneiataki Dimitri <dgen@aegean.gr>
From: Karopoulos Georgios <sip:gkar@aegean.gr>;tag=76341
CSeq: 2 INVITE
Authorization: Digest username="gkar",
realm="195.251.164.23", algorithm="md5",                    SIP
uri="SIP:195.251.164.23",                                   header
nonce="41352a56632c7b3d382b39e0179ca5f98b9fa03b",
response="a6466dce70e7b098d127880584cd57"
Contact:  <SIP:195.251.166.73:9384>;>
Content-Type: application/sdp

v=0
o=Tesla 2890844526 IN IP4 lab.high-voltage.org
c=IN IP4 100.101.102.103
t=0 0                                                       Session
m=audio 49170 RTP/AVP 0                                     Description
a=rtpmap:0 PCMU/8000
```

Figure 1. Structure of SIP (INVITE) message

REGISTER and INVITE are the most common messages in SIP. The REGISTER message (Figure 2) is used by a user to sign in a corresponding VoIP service.

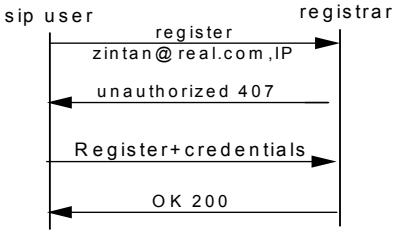

Figure 2. Register flow

On the other hand INVITE (Figure 3) is used to invite another user to participate in a session.

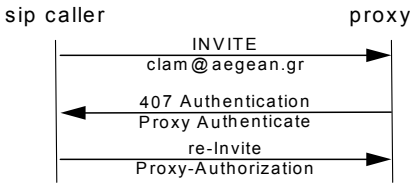

Figure 3. Invite flow

III. THE SIP DATABASE INFRASTRUCTURE

It must be mentioned that SIP servers rely on databases such as MySQL [3], Postgress [5] or Oracle [4] and utilize SQL statements in order to store and administer users' credentials and appropriate data for providing VoIP services. For example, open-source SIP implementations (e.g. SIP Express Router (SER) [6]) provide build-in modules in order to support MySQL and Postgress databases. This database schema is composed of various data tables. Among them, "Subscriber" and "Location" tables are of major importance, as they store critical data required for smooth VoIP operation. More specifically, the "Subscriber" table stores the appropriate data such as user name, domain, password etc, (see Table 1) for the legitimate users, while the "Location" table stores all the data representing the current available contact addresses for the legitimate subscribers (signed in a VoIP service).

Therefore, in case this attack is triggered against a SIP installation like SER, any corruption in the integrity of database and especially in the "Subscriber" and "Location" tables drives the provisioning of VoIP services to fail. Furthermore, the utilization of Web interfaces for the provision of VoIP services makes this attack more attractive to the potential perpetrators.

SUBSCRIBER					
User name	*domain*	*password*	*First name*	*Last name*	*Phone*
dgen	aegean	!#ertFGgh	Dimitris	Soccer	123@sip.aeg.com

Table 1. A subscriber record contained in table "Subscriber"

It is to be noted that the communication with the SIP database mandates the existence of a database user, having the appropriate privileges, who acts on behalf of the corresponding SIP server as depicted in Figure 4.

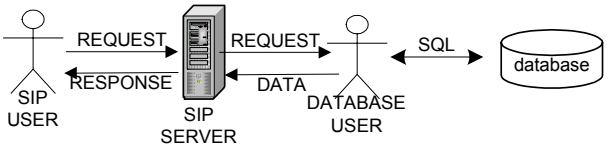

Figure 4. SIP Database Architecture

IV. SQL INJECTION IN SIP

A. Attack description

The text-based nature of SIP messages offers the opportunity for message tampering attacks [2] in SIP telephony services, similarly to HTTP messages. This attack is not only targeting in data modification, but also in the downfall of database services to cause a Denial of Service (DoS). The concept of SQL injection in SIP is quite similar to the aforementioned attack described in section II.A. The attack can be triggered every time a SIP network entity (e.g. SIP UA, SIP Proxy) is asking for authentication. When this situation occurs, the User Agent (UA) on behalf of the authorized user computes the appropriate credentials based on the HTTP Digest mechanism [7]. The result of this computation (credentials) is included in the message's authorization header (see Figure 1). Then the message is forwarded to the corresponding proxy server, which has to authenticate the received message. Thus, it re-calculates user's credentials using the user's password stored in the "Subscriber" table as presented in Table 1. To accomplish this task, it generates an SQL statement according to the following syntax:

Select password from subscriber where username='gkar' and realm='195.251.164.23'

In case a malicious user attempts to launch an attack in the SIP architecture, exploiting SQL injection, he tampers the SIP message and inserts the malicious SQL code in its Authorization header (see Figure 5). The candidate for injection message can be any SIP method, requiring authentication by a SIP server. The malicious code can be embodied either in the username or in realm fields in the Authorization header.

```
Authorization:Digest username="gkar';
    Update subscriber set first_name='malicious'
    where username='gkar'--",
    realm="195.251.164.23", algorithm="md5",
    uri="sip:195.251.164.23",
    nonce="41352a56632c7b3d382b5f98b9fa03b",
    response="a6466dce70e7b098d127880584cd57
```

Figure 5. SQL Injection in the SIP protocol

As soon as the proxy receives a SIP message with an 'infected' Authorization header, as illustrated in Figure 5, it generates and executes the following SQL statement:

Select password from subscriber where user name= 'gkar';
Update subscribe set first_name='malicius' where username='gkar'--

As a result, albeit message authentication fails, due to the fact that the attacker does not know the legitimate user's password, the second command manages to change '`gkar's first_name`' to '`malicious`'. It is also possible for a malicious user to attempt to employ similar SQL commands, aiming to make database useless and cause a DoS to the provided VoIP service.

The previous example it requires, from the SQL user that acts on behalf of the calling party (see Figure 4), to have the UPDATE privilege in the SIP subscribers' database. However, this is not a major restriction, as described further down in Section IV.B. Additionally, the malicious user can be an authorized "insider" (e.g. a legitimate subscriber), injecting the Authorization header with the SQL code as depicted in the Figure 6.

```
Authorization:Digest username=gkar, realm=1.23.4.5
    UNION SELECT FROM subscriber
    WHERE username=charlie and realm='195.251.164..23'
    algorithm="md5",
    uri="sip:195.251.164.23",
    nonce="41352a56632c7b3d382b5f98b9fa03b",
    response="a6466dce70e7b098d127880584cd57
```

Figure 6. Alternative example of SQL injection

Under these circumstances, the SIP proxy will generate the following SQL statement which prerequisite for the database user acting on behalf of SER to have *only* the READ privilege:

Select password from subscriber where user name= 'gkar' and realm='1.23.4.5'
UNION
Select password from subscriber where user name= 'Charlie' and realm='195.251.164.23';

When this statement is executed, Charlie (the attacker) will be registered as gkar (the legitimate user) in the provided SIP service. This attack is executed successfully because the first statement will deliberately fail as the malicious user has spoofed the realm and has injected the message with the second SELECT statement: *"UNION SELECT FROM SUBSCRIBER WHERE username='charlie' and realm='195.251.164.23' "*. Eventually, the whole message (the two Selects) will be executed as one statement. In addition, the malicious user has re-computed the corresponding credentials to the authorization header too.

B. Moderate limitations on the Attack

The SQL injection attack is independent from the underlying database and the specific implementation of the SIP server. The only restriction is coming from the Application Programming Interface (API) that is being utilized. For instance, the MySQL C API up to version 4.1 is quite immune to this type of attack, since only one SQL statement can be executed during one system call as pointed out in MySQL documentation [3], and some of the SQL injection attacks require the execution of more than one SQL statement (see the first example of Section IV.A).

In order for this attack to be successful, the hijacked SQL user (that acts on behalf of e.g. SER) must have the appropriate SQL authorization privileges to execute the malicious statement. Thus, the attacker may attempt, from the first place, to spoof user permissions table before launching the attack. Of course, he can also passively wait or actively keep trying until he locates the competent SQL user that holds the right privileges. However, VoIP providers, similarly to other Internet applications, allow their users to register, modify or even delete their current settings on-the-fly. This means that the administrator of the provided service must convey, to the SQL user that acts on behalf of the corresponding proxy (see Figure 4), the INSERT, UPDATE and/or DELETE privileges for the appropriate tables in the database. As a result, even this restriction is not a rigorous one.

C. Experimentation

Figure 7 depicts the test-bed that we have been utilized in order to provide some 'real-usage' experimentation results for the aforementioned attack. It consists of two laptop machines running SIP client software and the appropriate combination of SIP server and database. One of the SIP clients is the well-known Microsoft product implementing SIP stack named *Portrait* [9], while the other employs an open source SIP client called *Osip* [10].

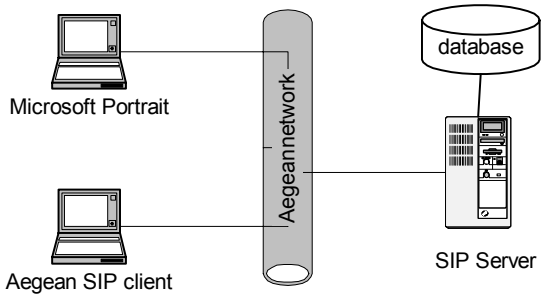

Figure 7. SIP test-bed architecture

The SIP client *Portrait* has been used, in order to discover if the SQL attack can also be launched through Commercial Off-The-Shelf (COTS) SIP clients. The attacker does not need to have any special skills for successfully performing the injection. In the case of Portrait, the malicious SQL code is added in one of the text boxes appearing in the corresponding authentication window. On the other hand, using the *Osip* client, the aggressor has simply to execute the application and modify the appropriate file that contains the SQL code. We were able to inject several SQL statements as described hereafter:

1. **Update subscribe set password='malicious122' where username='gkar'**
2. **Drop table Subscriber**
3. **Delete from Subscriber where username='gkar'**
4. **Insert into subscriber (Username, domain,phone Values ('charlie','1.1.1.1','sip:ch@voip.com)**

The damage provoked in the corresponding database depends on the malicious SQL code (UPDATE, INSERT, DROP, DELETE, CREATE) and the user's privileges that acts on behalf of SIP server to execute the SQL statements. As already mentioned in Section IV.B, the attacker can attempt to spoof user's permissions to the database in order to have the injected SQL code executed. Of course, there are also other prosperous ways [22][23] to manipulate SQL user's credentials, like doing some hacking in the database permission table. However, a detailed analysis on this issue is outside the scope of this paper.

Regarding the SIP server and the corresponding database we employed the following architectures:

1. 'Simulated' architecture: In this architecture we simulate the SIP server's stack and the Microsoft SQL database to administer user data.
2. Lab architecture A: In this architecture we use SER as the SIP server. The corresponding database infrastructure that connected with the SER employs MySQL in version 4.0.20.
3. Lab architecture B: The same as the previous architecture which utilizes the newer MySQL version 4.1.3.

In the first ('Simulated' architecture) and the third architecture (Lab architecture B) both of the malicious SQL statements described in Sub-Section A of the present section executed successfully. On the contrary, in the second architecture (Lab architecture A) due to the restriction in MySQL's C language API, as already mentioned in section IV.B, we did not manage to successfully execute the first of the examples (due to the restriction for execution of two statements in one system call). However, the execution of the second example statement was successful, giving access to a different user instead to the legitimate one. It is also worthy of note, that in this last example there is no need for the attacker to have any special permission to the database. Only the SELECT (read from the database) permission, which is either way compulsory for the VoIP service to operate, is sufficient for the attacker to execute the corresponding statement.

These outcomes confirm that this threat can become extremely dangerous as the aggressor is in position to effectively delete or modify crucial data needed for the accurate operation of the corresponding SIP-based VoIP service. For instance, among other things, the attacker can modify or delete subscribers' billing data. More importantly, this attack can be triggered from anywhere using cheap and easy to find commercial or open source software, as those used in our examples.

D. Protection from SQL injection in SIP

To facilitate the development of a robust and secure SIP based VoIP service, highly immune to SQL injection attacks one has to employ a number of prevention and detection mechanisms. Having these mechanisms acting simultaneously, it is possible to create a more secure environment. Various researchers have proposed a broad range of defense or shielding strategies [17],[18],[20],[21], aiming to protect Web architectures from equivalent SQL injection attacks as well (see Section II.A).

1) Prevention, Remedies and Countermeasures

Input validation procedures must be considered vital for the security of SIP based VoIP services. As already described in Section IV.A, the lack of any validation in data input process (e.g. in SIP message Authorization header) is responsible for security flaws. The employment of gateways to filter malicious input at the application level has also been studied [21]. Current firewall technologies incorporate packet inspection [16] for validating input data. The same techniques can be applied in the corresponding SIP architectures using the Middlebox communication approach [8].

Another technique for preventing SQL tampering in SIP is to digitally sign the messages that are exchanged. As a result, any modification in a SIP message can be detected, having the message automatically discarded by the SIP server. Generally, digital signatures can protect SIP messages from any sort of tampering attack. Nevertheless, digital signatures scheme requires the installation of a global or layered Public Key Infrastructure (PKI) beforehand. Moreover, this method is totally ineffective against "insiders". Finally, in order to avoid errors in input validation or to prevent any other malicious attempt, the SQL account that the SIP server uses to connect to the database must have only the minimum-required privileges.

2) Detection Framework

No matter how strong the existing security prevention mechanisms employed in current SIP based VoIP Services are, there is always the possibility for a malicious user to manage to by-pass them. So, in case an insider launches an intrusion attack, it is quite probable that no one of the existing prevention mechanisms will trigger an alarm. For example, consider a legitimate SIP user who is injecting malevolent SQL code in a message and then signs it with his private key. There is no doubt that this attack can be hardly defeated by the usual detection or prevention mechanisms.

To avoid such situations, the employment of an Intrusion Detection System (IDS), for the offered SIP based VoIP services is considered mandatory. On the other hand, in some cases, it is more economical to prevent only the uppermost attacks and detect the rest, than trying to prevent everything in a much higher cost. Besides that, lately, it has become a common belief that the use of a detection system is itself sufficient for protecting Web applications from SQL injection. For example, well know open source IDS tools like Snort (www.snort.org), have been employed to effectively detect this assault [17]. In those systems any distinct attack of this form is described through some specific static structure, known as the attack's '*signature'*. Thus, SQL injection attack in SIP architectures can be similarly confronted by identifying, categorizing and prototyping the corresponding signatures.

After several trial signature combinations that facilitate the detection of the SQL injection attack in the SIP protocol, we ended up in the one depicted in Figure 8. The proposed signature is based on the SIP message syntax, which is fully specified in RFC 3261 [1]. Since all SIP messages are based on this syntax, it will be more easily to embody a light SIP IDS mechanism in a slightly modified SIP protocol stack. Alternatively, it is possible to add this signature scheme in existing open source IDSs without making any modifications to the SIP stack.

Moreover, SIP messages, which include malicious SQL code, do not conform to SIP specifications. As a result, it can be characterized as a special case of illegal messages. Thus, the basic idea is to construct a general identification framework that is able to apply to any malicious SIP message, which is not compatible with SIP specifications. Each signature is composed by the malicious SIP message optionally followed by some additional rules. Based on this general architecture

we suggest a signature to identify SIP messages that include SQL code (see Figure 8). Note that the proposed signature is very similar to valid SIP messages. The main difference is that the message is characterized as malicious whenever any SIP method requires authentication and the corresponding Authorization header contains an SQL statement. Moreover, to make this signature more robust, we added an additional rule that checks the length of the Authorization header. For instance, if also the authorization header has length bigger than then expected, this can help to identify possible SQL injection when an attacker tries to evade the IDS. However, this parameter must be utilized very wisely because any misuse of this attribute will trigger a false alarm. Finally, note that this is an indicative signature for a wide range of similar attacks, as it is well known that attacker will try many different ways to evade the detection system.

```
METHOD SIP-URI | SIPS-URI MESSAGE HEADER+
MESSAGE HEADER =Via | Max-Forwards | From |To
                | Call-Id |CSeq | Contac |User-agent
                |Authorization |Event |Content-Length
Authorization = Digest username=".+(';SQL-STATM COMMENT)"
                realm=" Ipaddress" |
Authorization = Digest username=".+" realm="Ipaddress
                (';SQL-STATM COMMENT)"  |
Authorization = Digest username=".+(';SQL-STATM COMMENT)"
                realm=" Ipaddress (';SQL-STATM COMMENT)"
SQL-STATM= UPDATE | INSERT | UNION
COMMENT = "--|#"
UPDATE = SEE SQL 92 syntax
INSERT = SEE SQL 92 syntax
additional rules
size_of(Authorization)> %constant% e.g 100 bytes
```

Figure 8. SQL injection attack signature in SIP

V. CONCLUSIONS AND FUTURE WORK

As SIP based VoIP services are becoming more and more popular, the Internet-inherited security problems and threats will become more severe. Attackers can cause serious problems in regular VoIP operation, by exploiting a wide range of existing malicious tools. In this paper, we shifted the SQL injection attack, already found in other Internet applications, to the SIP architecture. As this attack is not yet reported in SIP environments, we demonstrated how malicious users could take advantage of it, in order to compromise sensitive SIP components and induce commotion to or paralyze the provided service. We employed two different SIP clients and properly designed test-beds, to demonstrate that the attack is profitable even with little know-how and resources. Moreover, we introduced countermeasures, remedies and a detection signature-based framework against this kind of attack.

However, the overheads, in terms of performance, introduced in SIP as a result of the proposed solutions are still under study. In addition, we esteem that a slight modification of this aggression can also be applicable in any VoIP service, independently from the underlying signaling protocol used. For this reason, there is an urgent need for a general identification framework capable of detecting similar attacks, to be established. The accomplishment of this goal, currently under inquiry, will contribute a great deal in VoIP security and reliability.

Acknowledgments

This work has been performed in the framework of the IST-2004-005892 project SNOCER (www.snocer.org), which is funded by the European Union.

REFERENCES

[1] Rosenberg J., Schulzrinne H., Camarillo G., Johnston A., Peterson J., Spark R., Handley M., Schooler E., "Session Initiation Protocol", RFC 3261, June 2002.

[2] Geneiatakis D., Kambourakis G., Dagiuklas T., Lambrinoudakis C. and Gritzalis S., "SIP Security Mechanisms: A state-of-the-art review", to be presented in the Fifth International Network Conference (INC 2005), Samos, Greece, July 2005.

[3] "MySQL open source database", http://www.mysql.com

[4] "Oracle database", http://www.oracle.com

[5] "Postgress database", http://www.postgreSQL.org

[6] "SIP Express Router", http://www.iptel.org/ser

[7] Franks J., Hallam-Baker P., Hostetler J., Lawrence S., Leach P., Luotonen A. and Stewart L., "HTTP Authentication: Basic and Digest Access Authentication" IETF, RFC 2617, June 1999.

[8] Srisuresh P., Kuthan J., Rosenberg J., Molitor A. and Rayan A: "Middlebox Communication Architecture and framework", IETF, RFC 3303, August 2002.

[9] "Microsoft Portrait" http:// research.microsoft.com/~jiangli/portrait/.

[10] "oSIP ", open source implementation of SIP stack, http://www.osip.org

[11] Anley C, "Advanced SQL Injection In SQL Server Applications." An NGSSoftware Insight Security Research (NISR) Publication, 2002.

[12] Spett K, "Blind SQL Injection", http://ww.spidynamics.com/whitepapers/Blind_SQLInjection.pdf, 2003.

[13] Ofer Maor, "SQL Injection Signatures Evasion", http://www.imperva.com/application_defense_center/white_papers/sql_injection_signatures_evasion.html, April 2004.

[14] Finnigan P., "SQL Injection and Oracle, Part One", http://www.securityfocus.com/infocus/1644, November 2002.

[15] Varshney U, Snow A, McGivern Mt, and Christi H, "Voice Over IP" Communications of the ACM Vol. 45, 2002.

[16] Dharmapurikar S., Krishnamurthy P., Sproull T., and Lockwood J., "Deep Packet Inspection Using Parallel Bloom Filters." In Proceedings 11th Symposium of High Performance Interconnects (HOTI'03), pages 44-71, 2003.

[17] Mookhey K. K. and Burghate N, "Detection of SQL Injection and Cross-site Scripting Attacks" http://www.securityfocus.com/infocus/1768#ref1, March 2004.

[18] "Preventing HTML form tampering", http://advosys.ca/papers/form−tampering.htm January 2005.

[19] Black U., "Voice Over IP", Prentice Hall, August 1999.

[20] Scott D. and Sharp R, "Developing Secure Web Applications", IEEE Internet Computing, pp. 38-45, November 2002.

[21] Scott D. and Sharp R., "Abstracting Application-Level Web Security," Proc. 11th Int'l World Wide Web Conf., ACM Press, New York, May 2002, pp. 396-407.

[22] "MySQL UDF Dynamic Library Exploit", http:// www.securiteam.com/exploits/6G00P1PC0U.html

[23] "MySQL DoS and local root exploits" http://www.sfu.ca/~siegert/linux-security/msg00017.html