Web malware that targets web applications

Chapter 12
Web Malware that Targets Web Applications

Ammar Alazab
Deakin University, Australia

Jemal Abawajy
Deakin University, Australia

Michael Hobbs
Deakin University, Australia

ABSTRACT

Web applications have steadily increased, making them very important in areas, such as financial sectors, e-commerce, e-government, social media network, medical data, e-business, academic an activities, e-banking, e-shopping, e-mail. However, web application pages support users interacting with the data stored in their website to insert, delete and modify content by making a web site their own space. Unfortunately, these activities attracted writers of malicious software for financial gain, and to take advantage of such activities to perform their malicious objectives. This chapter focuses on severe threats to web applications specifically on Structure Query Language Injection Attack (SQLIA) and Zeus threats. These threats could adopt new obfuscation techniques to evade and thwart countermeasures Intrusion Detection Systems (IDS). Furthermore, this work explores and discusses the techniques to detect and prevent web application malware.

INTRODUCTION

Malware is a very broad term that describes a kind of malicious software (Valli & Brand, 2008). However, numerous definitions have been proposed to define malware. For example, malware is software that harmfully attacks the software of others (Kramer & Bradfield, 2010). For the purpose of this research, malware is defined as any piece of code or string that causes harm to information systems without a user's permission.

Recent trends in web application malware have become a major threat and they are increasing in complexity and evolving rapidly as systems

DOI: 10.4018/978-1-4666-3926-3.ch012

provide more opportunities for more automated activities. Furthermore, the damages caused by web application malware to individuals and businesses have dramatically increased in 2010 (RSA, 2011).

Today, writers of malicious software (malware) either develop sophisticated techniques to conceal their attacks or constantly change their method of attack to evade detection software. New study identifies a new attack that infected nearly 300,000 web pages, with the infection containing malicious code that revealed client information and directed clients to a fake web site (Jie et al., 2010). However, this type of attack event was just one of a series of malicious activities targeting web applications. Research has indicated that fraud detection has steadily increased over recent years (BitDefender, 2010).

Even though attackers who achieve unauthorized access to financial systems cause huge losses to the financial sector, there is not one single technique that can stop them. However, a threat that was once utilized by individual criminals is now the focus of major organised crime crossing international boundaries and jurisdictions. A report by the Australian government warns that attacks will become more prevalent as more persistent techniques are adopted (RSA, 2011).

Generally, an attacker develops new and sophisticated techniques to target and hack the web application. The result is that attackers gain access to the data of other users. To prevent web application attacks, different approaches have been suggested but they do have limitations. Indeed, some of these approaches have yet to be implemented and in the approaches that have been, most cannot prevent or detect every single type of attack.

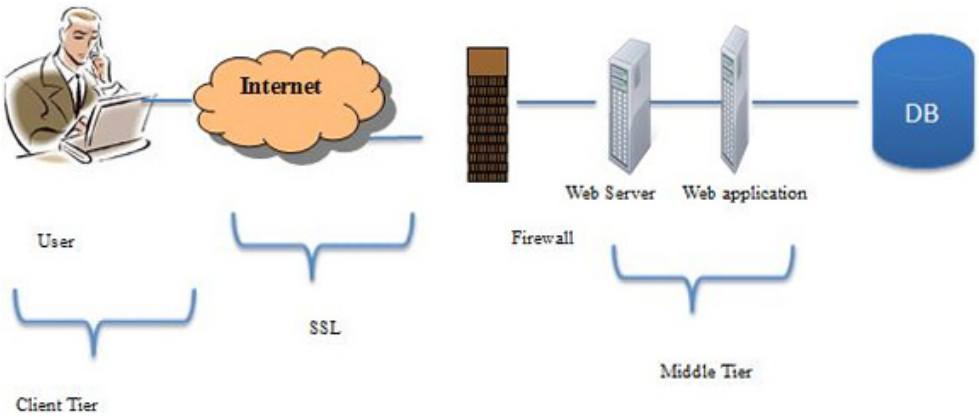
There are several different types of malware. These include viruses, worms, Trojan horses, spyware, rootkits and backdoors, etc. This chapter will focus on malware that specifically targets web applications using SQLIAs. This research will make the following contributions:

- Significant investigation exploring new evasion techniques used by hackers to compromise web applications.
- Exploration of existing detection and prevention techniques against SQL injection.

BACKGROUND

Web applications are applications that run over a network (such as the internet or an intranet) that enable a website to become dynamic by making connections within the database. The high level system component of a web application is shown in Figure 1. Included in web application architec-

Figure 1. Web application architecture

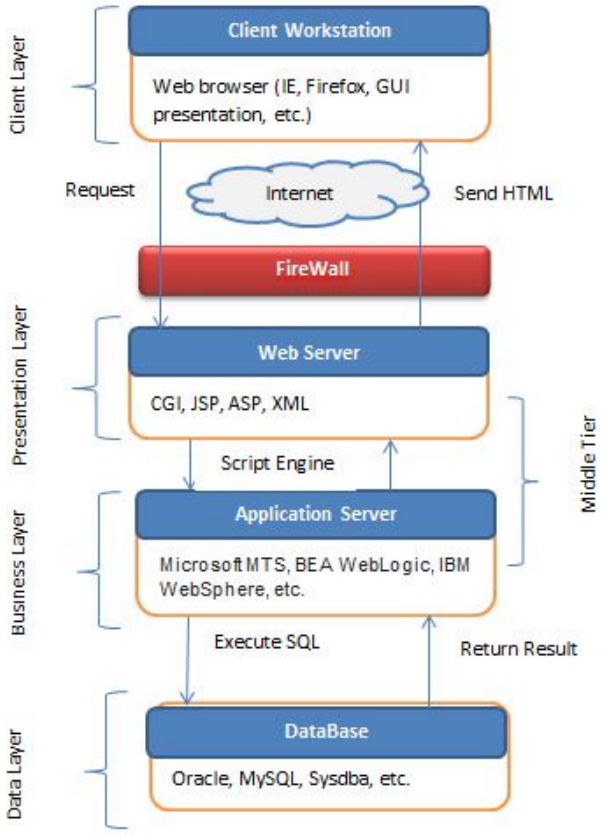


ture are browsers, a network, a web server, a web application and a database.

Figure 2 explains the web application process model in detail. Firstly, the client requests either a static or dynamic page. Secondly, the web browser passes this request through the firewall to the web server. Thirdly, the web server handles this request based on initial configuration such as HTTP, HTTPS, etc. The web server can also handle these requests by "decoding" the webpage. Fourthly, the web server passes this request to the web application server. Next, the web application passes these requests to the database. In addition, the web application processes commands and verifies security access to the database through middleware such as Java Database Connectivity (JDBC), Structured Query Language for Java (SQLJ), Java Data Object (JDO), Open Database Connectivity (ODBC), etc. and makes the logical decisions. After verifying database access, the web application server sends the Structured Query Language (SQL) request to the database server. Finally, the database server manipulates this request by allowing storage, deletion, or updating of the data. Depending upon the SQL query, it sends back the results to the application server.

Generally, web applications use query statements to generate strings to interact with the database. Usually, these queries are generated by web application servers such as ASP, JSP and PHP. A string contains both the query itself and its parameters which are usually the user name and password. Then, the string is forwarded to the database server for checking as a single SQL statement. If the received string is compromised

Figure 2. Web application model works



or injected, it will cause data leakage. Therefore, it is important to protect the web application from illegal access.

Web applications are infamous for security vulnerabilities that can be utilised by writers of malware, and also by hackers. The global accessibility of web applications is a serious problem which renders them vulnerable to attack. One of the most known and widespread threats to web applications is SQLIAs (Bisht, Madhusudan, & Venkatakrishnan, 2010).

Web applications offer a great opportunity to access the database through the internet (Peng, Chen, Chung, Roy-Chowdhury, & Srinivasan, 2010), which has provided the required service to customers. However, these advantages have raised a number of security vulnerabilities from improper code that have resulted in Structured Query Language Vulnerabilities (SQLV) that give hackers the ability to influence the Structured Query Language (SQL) a web application passes to the back-end of a database. The insertion of malicious code into strings to gain unauthorized access to a database is used to either retrieve information or destroy the database (DB) where all the data is sensitive.

Web application security generally focuses on identifying vulnerabilities and malicious strings within web applications layers. Firewalls and a Secure Sockets Layer (SSL) protect information transferred between the site and client, but do not protect information against web application hackers because it is built on top of the web application infrastructure (Yu, Aravind, & Supthaweesuk, 2006). Therefore, it is easy to append data and commands into a SQL statement. Even normal users can attempt direct connections to the database through specific ports by bypassing the security mechanism (Vella, 2007).

WEB APPLICATION ATTACK

Web applications are designed to assist any client to connect to the Database (DB) through a user's web Brower. However, if the user's input is not handled correctly, the web application can be left vulnerable to a malicious attack.

In addition, attackers develop tools to look for popular vulnerabilities to launch attacks against web applications. This availability of tools, and the existence of motivations to launch attacks, makes web applications a critical area to study. According to the report of Common Vulnerabilities and Exposures (CVE), the numbers of web-related vulnerabilities increased from 2005 to 2010 (Corporation, 2010).

Moreover, web applications use Rapid Application Development (RAD) to accelerate building complex web application systems but this makes web applications more vulnerable to attacks. According to the Open Web Application Security Project (OWASP), the top ten application security risks are presented in Table 1 (OWASP, 2010). This table shows all of the popular web application attacks.

Another report from the Web Application Security Consortium's Statistics Project (WACSP) stated that in 2010, the popular vulnerabilities in web application found by penetration testers were (Gordeychik 2010). See Table 2.

We note the improper input handling is equal to 14.65% and insufficient authentication is equal to 13.38% which helps to increase SQL injection.

The result is that a SQL injection attack (SQLI) dominates all other vulnerabilities and is still the best known type of attack. It has risen from number two in 2007 to number one in 2010 according to Open Web Application Security (OWASP, 2010).

Furthermore, the WHID announced that the Web Application Security Consortium's Statistics Project revealed the weaknesses that are exploited by various attack methods (http://

Table 1. Attack methods in web applications

No.	Web Application Security Risks	When does it occur?	Impact
1	Injection	The hacker sends a malicious string as a query statement to the database.	Data loss, data stolen, modified, deleted, corruption or denial of access
2	Cross-Site Scripting (XSS)	Attacker executed malicious code in the user's browser	Redirect users to malicious web sites, the result sensitive information
3	Broken Authentication and Session Management	When a session ID is visible due to leaks in the authentication process	Attacker could do anything with the victim's privileged accounts
4	Insecure Direct Object References	When An attacker changes the URL parameters	Compromises all the data
5	Cross-Site Request Forgery (CSRF)	Attacker sends fake HTTP requests to trick a victim into submitting them via image tags.	Loss or modified data
6	Security Misconfiguration	Attackers try to find unused pages, unprotected files and unprotected directories, etc. to reach for sensitive data.	The web application could be completely compromised without knowledge. All data could be stolen or modified
7	Insecure Cryptographic Storage	Here attackers don't break the cryptographic, but disclose something else to get the original text.	Lost data such as health records, credentials, personal data, credit cards, etc.
8	Failure to Restrict URL access	Simply changes the URL to a privileged page	Attacker can do anything the victim can do
9	Insufficient Transport Layer Protection	The attacker simply monitors network traffic (like an open wireless or neighbourhood cable modem network), and observes an authenticated victim's session cookie. Attacker then replays this cookie and takes over the user's session.	Phishing, resulting in a stolen account
10	Invalidated Redirects and Forwards	Attacker redirects links to invalidate and trick victims into clicking it. Victims will probably click on it, with the security mechanism bypassed.	Redirections such as these may attempt to install malware or trick victims into disclosing passwords or other sensitive information

Table 2. Web vulnerabilities

Web Application Vulnerability	Percentage
Improper output handling	22.29%
Insufficient anti automation	15.29%
Improper input handling	14.65%
Insufficient authentications	13.38%
Unknown	8.92%
Application misconfiguration	6.37%
Insufficient process validation	6.36%

projects.Webappsec.org/Threat-Classification) and further research has identified an increase in professional criminals using a combination of attacks (Gordeychik, 2010).

WEB APPLICATION VULNERABILITIES EXPLOITED

Web applications use Rapid Application Development (RAD) to accelerate building complex web application system; unfortunately this makes web applications more vulnerable to attacks. A web application has many features and characteristics which could lead to infection of SQLAs: (i) the input query strings contain usernames and passwords; it may inject with malicious code; (ii) Dynamic query statements also help SQLAs; (iii) SQL queries don't contain any information about the source; (iv) in addition, the problem of SQL injection is the integrity of information fl t (Jie, Phan, Whitley, & Parish, 2010). Furthermore, malicious application writers use combination

attacks such as botnets, hacking tool kits and web application vulnerabilities to make cybercriminals more resilient and powerful than ever before.

Today, most industries are moving toward the use of web applications. Even though the features of web applications make them convenient to support and maintain, many of these same features also make them vulnerable to attack. As a result, attackers target web applications to access the database through SQLIAs because this type of attack can compromise the confidentiality and integrity of information in the databases, resulting in the theft of confidential data and the breaking of data integrity or affecting the availability of the web application.

The Open Web Application Security Project (OWASP) revealed that many web application vulnerabilities cause input validation problems. This means the process of checking all the input in an application before using it for the input of the client (OWASP, 2010). However, this is a weakness and can lead to SQL attacks. SQLIAs are not only introduced via any user inputs but also any external strings or outside input that malicious writer uses to build a query string like cookies, a web application variable or a server variable. Attackers have various techniques to employ to find vulnerabilities based on the input source that can be used to cause a database to crash, steal data, or to deface the web. These vulnerabilities can breach the security mechanism that is designed to prevent, detect and recover (Anderson & Moore, 2007) if there is a threat to a web application. In the case of vulnerabilities, there is a lack of enforced security mechanisms, a lack of configuration of security mechanisms and a lack of antivirus detection systems. See Figure 3.

However, the main problem causing web application vulnerabilities are insufficient input validation to exploit invalidate in web applications, the hacker need to inject a malicious string into web applications (Anderson & Moore, 2007). The most commonly used methods hackers use to inject web applications are*:*

- **Parameter Tampering:** These rely on the browser; the attacker changes the parameters in Uniform Resource Locator (URL) to bypass the security mechanism. For example in online shopping when the developer use hidden field to store item product statues, attackers take advantage from this hidden file and change the information such as shown in the following example:

  <input type=”hidden” id=”321” name= ”cost” value=”2.00”>

In the above example the attacker can change the cost of a product to lowering cost.

- **Cookie Poisoning:** The attacker puts a malicious string into cookies to submit to web application. Web browsers submit the request using GET or POST methods. If the method GET is used, it means all parameters and values will display in the browser. The result is hackers can tamper this query statement. For example the following URL is submitted: http://www.website.com/

Figure 3. Relation between threat, vulnerabilities, and mechanisms

page.asp?UserId=6543&value=1, the hacker can modify the URL parameters (User ID and value) in order to add another account like http://www.website.com/example.asp?UserID=1243&value=9999

In general, hackers develop advanced techniques that compromise a database using SQL statements to steal information from a database that help to compromise web application successfully.

WEB APPLICATIONS THREAT

Recent trends in web application malware have become a major threat and are increasing in complexity as systems provide more opportunities for automated activities. Furthermore, damages caused by malware to individuals and businesses have also significantly increased in the last few years (RSA, 2011).

SQLIAs are one of the biggest threats to web applications. New evasion techniques allow attackers to take control of a database on an application. The result increasingly causes financial fraud, website defacement, denial of service and possibly the manipulation of data.

The general use of the Internet has made it increasingly difficult to find a high level of security because most web applications work as distributed systems and are built on multiple heterogeneous technologies. In addition, hackers increasingly target web applications because they connect with databases that contain sensitive data. The threats related to database security include (Connolly, Begg, & Strachan 1996)

1. Loss of confidentiality when information can be accessed by someone who shouldn't access it.
2. Loss of privacy which refers to exposure of personal information.
3. Loss of integrity which may be due to corrupt data since unauthorized individuals make the integrity loss happen.
4. Loss of availability which means the services or systems cannot be reached due to Denial-of-Service attacks.
5. However, Connolly *et al* also define risk as any effect that might damage a database with the system crashing being the end result. This threat can be classified as:
    a. **Interruption:** An attempt to destroy an asset or make it unusable.
    b. **Interception:** The database becomes unavailable.
    c. **Modification:** For example, a hacker can change the values in a database.
    d. **Fabrication:** A hacker can insert records into an existing database.

SQL INJECTION ATTACKS (SQLIA)

In order to demonstrate the web application threat, we conducted a second study of a SQL Injection Attack. In this section, I will define SQL, SQL injection, and provide examples of a SQL injection, plus various SQL injection vulnerabilities and evasion methods.

Structured Query Language (SQL)

SQL stands for Structured Query Language. It was defined by ANSI (American National Standards Institute) and has also been adopted as a standard by ISO (International Organization for Standardization). SQL is standard language for accessing and manipulating relational database systems (w3schools, 2011).

SQL is divided into three parts: the Data Manipulation Language (DML), Data Definition Language (DDL) and Data Control Language (DCL).

- **DML:** Used to perform, SELECT statement, change existing data - UPDATE statement, create new records holding data - INSERT INTO statement and delete data - DELETE statement.
  - ***SELECT:*** Used for retrieving information from one or more tables in the database and displaying it. The syntax of the SELECT statement is given below:

    SELECT [DISTINCT|ALL] *|column_expression][,...] FROM table_name [alias][,...]

    [WHERE condition] [GROUP BY column_list][HAVING condition]
  - ***INSERT:*** Used for adding new data rows in a table. The syntax of the INSERT statement is given here: INSERT INTO Table_name [(column_list)] VALUES (data).
  - ***UPDATE:*** Used for updating data rows in a table. The syntax of the *UPDATE* statement is given: UPDATE table_name SET column_name1 = data_value1.
- **DDL:** DDL statements are used to build and change the structure of tables, records, fields, and other objects in the database.
  - ***CREATE:*** Creates an object inside the database, the syntax of the CREATE TABLE is CREATE [TEMPORARY] TABLE *[table name]* ([*column definitions]*) *[table parameters]*;
    ALTER TABLE <table name>
  - ***DROP:*** Removes an object from database DROP TABLE <table name>;.
  - ***DCL:*** Statements.
  - ***GRANT:*** Gives user's privileges select, insert, update, or delete on the database.
  - ***REVOKE:*** Withdraw user's privileges select, insert, update, or delete on the database.

SQL Injection Attack (SQLIA) is a type of attack on a web application that occurs when an attacker inputs malicious strings as parameters in legitimate SQL statements (Torrano-Gimenez, Perez-Villegas, & Alvarez, 2010). The SQLIA allows the hacker to gain complete access to the database server resulting in a serious threat to the web application.

Impact of SQL Injection Attacks

SQL injection is a serious threat that can cause the following:

1. **Web Defacements:** Zone-H is a web site that records web site defacements (http://www.zone-h.org/news/id/4735). This site has recorded the web applications that have been hacked over the years using SQL injection vulnerabilities. Records show that there were 33,920 attacks in 2010 and 57,797 attacks in 2009 (Almeida, 2010).
2. **Financial Fraud:** Attacks have been responsible for loss of credentials to fraudsters.
3. **Spoof:** To impersonate a user in order to manipulate a database that allows an attacker to intercept the database.
4. **Tampering:** Updating or deleting data in a database without authorization.
5. **Authentication Bypass:** When the hacker logs onto the web application without a valid username and password.
6. **Information Leakage:** When an application reveals sensitive data, such as technical details of the web application. The result is the theft of sensitive data, such as credit card information and user identities (driver's license number, birth date, passport number, social security numbers, etc.) and user-specific information (passwords, username).

7. **Denial of Service (DoS):** This can occur when the hacker deletes or modifies the database. The end result is the unavailability of the website. This means legitimate users are unable to access or use the web application. For example, if the attacker enters input as "; SHUTDOWN; - -" The query will be: "SELECT * FROM username WHERE userTable=' '; SHUTDOWN; --'and password='whatever";

The result of the above SQL statement means the database will be shut down and a DoS attack will be conducted on the web application because the database becomes unavailable. The '--' character sequence means the comment and the character ";" is the end of one query and the beginning of another SQL statement, which is used to stop the database service.

Example of SQL injection

The code in Box 1 shows a JSP login page that is vulnerable, leading to a SQLIA.

The above code is a simple web application that contains a SQL injection. In this example, the web developer uses input parameters to login using two parameters: username and password, in order to dynamically build an SQL statement. They check the username and password at the login stage against the database as shown in Table 1.

When a user logs in with their user name and password in a web browser, the request is submitted as a dynamic URL, such as http://<mysite>/show.jsp?login=username&pass=password, and sent to the web server. The web server checks the parameters of the login username and the password against the database. If they match, then the account information is returned. Otherwise, a null set is returned by the database and the authentication fails. If the SQLA input "user1'--"as the user name "x" as the password, the database server returns the recode despite the password being incorrect because the database server ignores the statement after the comment "--".

From this example, it follows that the problem is in the structure of the query syntax used by programmers.

SQL Injection Vulnerabilities

A SQL injection attack exploits vulnerabilities in input validation to execute a malicious statement in the database. The problems of SQLI become complicated if the database server doesn't use

Box 1.

```
1. String username = request.getParameter ("username");
2. String password = request.getParameter ("password");
3. Statement stmt = connection.createStatement();
4. String sql = new String(SELECT password FROM Tableusers WHERE Username=";//
sql query to retrieve 6.values from the secified table
7. sql += username + "' AND password=" + password;
8. stmt = Conn.prepareStatement(sql)
9. ResultSet Rs = stmt.executeQuery() //execute query
10. if (Rs != null)
11. displayAccount(Rs); // Show Information
12. else
13. sendAuthFailed(); // Authentication failed
```

Table 3. Result of attack type

Attack Type	Result
Unauthorized data access	Allows the SQLIA to trick the web application in order to obtain information from the database that is not supposed to be returned or is not allowed to be seen by this user.
Authentication bypass	Allows the attacker to access the database-driven application and observe data from the database without presenting proper credentials.
Database modification	Allows the attacker to change the database by insert, modify, or destroy data content.

privileged accounts to connect to the database. In this case it is applicable to use the database server to run operating system commands.

Vulnerabilities in web applications allow malicious users to obtain unlimited access to private and confidential information from any database-driven applications written for web applications. This occurs when end user input is a malicious string that literally evades characters embedded in SQL statements. The results are shown in Table 3.

Detection and Prevention Techniques

Researchers have proposed several methods for detection and prevention to assist developers and Anti-Virus (AV) vendors to find defects in defensive coding.

Static Analysis: Pixy (Barnes, Marateo, & Ferris, 2007) is an open source prototype implementation aimed at detecting SQL injection, cross-site scripting, or command injection based on flow-sensitive, inter procedural and context-sensitive data flow analysis. In addition, Pixy uses literal analysis to improve the rightness and precision of its results.

Combined Static and Dynamic Analysis: AMNESIA (Halfond, Orso, & Manolios, 2006) is a technique that combines dynamic and static for preventing and detecting web application vulnerabilities at runtime. AMNESIA uses static analysis to generate different types of query statements. In the dynamic phase, AMNESIA interprets all queries before they are sent to the database and validates each query against the statically built models. In other words, AMNESIA stops all queries before they are sent to the database and validates each query statement against the AMNESIA models. However, the primary limitation in AMNESIA according to Ramaraj et al. (IndraniBalasundaram & Ramaraj, 2011), is a technique that depends on the accuracy of its static analysis for building query models to successfully prevent SQL injection. Furthermore, AMNESIA doesn't consider certain types of code obfuscation or query development techniques that could make this step less precise and result in both false positives and false negatives.

Moreover, Martin, Livshits, and Lam (2005) proposed Program Query Language (PQL) that uses static analysis and dynamic techniques to detect vulnerabilities in web applications. In static analysis, information flow techniques detect when malicious input has been used to generate a SQL query statement which are then flagged as SQLIA vulnerabilities. According to Ramaraj et al. (IndraniBalasundaram & Ramaraj, 2011) the limitation of this approach is that it can only detect known patterns of SQLIA.

Taint Based Approaches: Huang et al. (2004) proposed WebSSARI (Web application Security via Static Analysis and Runtime Inspection) to: a) statically validate existing web applications and legacy web application code without any extra effort for programmer; and b) automatically protect potentially defective code. In this model, static analysis is applied to validate infected runs versus given conditions for sensitive formulations.

Code Checkers: This is based on static analysis of web applications that can reduce SQL injection vulnerabilities and detect type errors. For instance, the JDBC-Checker (Gould, Su, & Devanbu, 2004) is a tool used to check code for statically validating the accuracy of dynamically-generated

SQL queries. However, researchers have also developed particular packages that can be applied to make SQL query statements safe (McClure & Krüger, 2005). These techniques are good but they need extra effort from programmers to build query statements using Application Programme Interfaces (APIs), especially for legacy web applications due to a lack of information about the programmers' intent.

Tainted Data Tracking: This method was proposed by Halfond et al. (Halfond et al., 2006) and is based on track tainted-ness of data. It specifically checks for dangerous content that comes from user input. According to Nguygen-Tuong et al. (2005), this can be done via instrumenting the run time environment or the interpreter of the back-end scripting language so when a SQL statement is sent to the database server, its syntax tree is examined first. However, this approach does not provide any way to check the accuracy of the input validation routines (Bandhakavi, Bisht, Madhusudan, & Venkatakrishnan, 2007). That programs that do use incomplete input checking routines may still pass these checks yet remain vulnerable to injection attacks (Bisht et al., 2010).

Static Analysis: Static analysis uses a statistical approach with data and uses a flow analysis to detect tainted data. This method is not efficient because it can lead to false positives and scalability issues.

Table 4 summarizes the results for detection and prevention techniques against the different types of SQLIA attacks as presented in Section 3.6 and 3.7. Most of these techniques failed to detect stored procedure and new evasion techniques because these techniques focus on queries generated within the application. Only four techniques, Candid, AMNESA, SqlCheckS and parse tree address SQLA attacks based on stored procedure because these techniques use database layer to interpret a SQL statement in a web application layer in the same way as it interprets a database layer.

Table 4. Summary of results for detection and prevention techniques against types of SQLIA

Techniques	Prevent	Detect	Tautology	Illegal	Union	Piggy-Backed	Store Procedure	Timing	Evasion Tech
Candid (Bisht, et al., 2010)	YES	NO	YES	YES	YES	YES	YES	YES	NO
AMNESA (Halfond & Orso, 2006)	YES	YES	YES	YES	YES	YES	YES	YES	NO
CSSE (Pietraszek & Berghe, 2006)	YES	YES	YES	YES	YES	YES	NO	YES	NO
IDS (Valeur, Mutz, & Vigna, 2005)	YES	NO	YES	YES	YES	YES	NO	YES	NO
Dynamic Taint Propagation for Java (Haldar, Chandra, & Franz, 2005)	YES	NO	YES	YES	YES	YES	NO	YES	NO
SqlCheckS (Su & Wassermann, 2006)	YES	NO	YES	YES	YES	YES	YES	YES	NO
Using parse tree validation to prevent SQL injection attacks (Buehrer, Weide, & Sivilotti, 2005)	YES	NO	YES	YES	YES	YES	YES	YES	NO
WebSSARI (Huang, et al., 2004)	NO	YES	YES	YES	YES	YES	NO	YES	NO

CASE SCENARIO: THE ZEUS BOTNET THREAT

Crime tool kit exposes a widespread a variety of attack vectors. Many different attack kits are presented with a range of exploits and a wide array of attack vectors. Advancing in amount of crime tool kit increase the probability for success attack on web application, because there are a more chances that the hacker discovers the web application vulnerability. There are different crime tool kits existing on underground marketplaces.

In order to demonstrate the web application threat, we conducted a study of Zeus Malware. The Zeus Trojan is also called Zbot, NTOS, WSNPOEM, or PRG, and is currently the most dangerous category of financial malware in circulation, both in terms of infection size and effectiveness. Furthermore, it is the biggest and most sophisticated threat to internet security. The Zeus Trojan is estimated to be responsible for about 90% of banking fraud worldwide (RSA, 2011) and guilty of 44% of the world's banking malware infections (Trusteer, 2009). Symantec Corporation describes it as "Zeus, King of the Underground Crimeware Toolkits."

The Zeus Trojan software, which comes with a friendly interface toolkit, is available in underground online forums for $1,500 – $20,000 USD. Currently, it is causing serious problems because it enables cybercriminals to configure and create malicious software to affect user systems, and allows them to take control of a compromised computer to harm the data, log keystrokes, and execute unauthorized transactions in online banking. Reports and studies (Trusteer, 2009) show that since last year, Zeus has been embroiled in more than half of the banking malware infections in the world.

The Zeus Trojan carries a very light footprint and is designed to steal sensitive data stored on computers or transmitted through web browsers. Once infected with Zeus, the computer sends the stolen data to a bot command and control (C&C) server via encrypted HTTP POST requests where the data is stored. It also allows cybercriminals to inject content into a bank's web page as it is displayed in the infected computer browser in real time. It is setup so stolen data is sent to a "drop server" controlled by an attacker called a botmaster and allows cybercriminals to control the infected systems remotely. To compound this problem, Zeus is highly dynamic and applies obfuscation methods such as polymorphic and metamorphic encryption in a network of bots. In each infection, it automatically re-encrypts itself to create a new signature that defeats signature-based detection that makes the signature difficult to comprehend. The Windows Zeus is increasingly hard to stop as it can successfully evade commercial detection engines and is able to hide malicious features such as string and API function calls. The Zeus Trojan is still developing and has versions and new plug-in releases that can also infect the latest operating systems such as Windows 7 and Vista systems (Binsalleeh et al., 2010).

According to numerous research labs and hacker forums, the Zeus botnet has recently combined (RSA, 2011) with the new release of the 2010 'Spy Eye Trojan' source code to create more sophisticated bots and take the threat to a new level. This new toolkit is reportedly available for purchase in the underground market and version 1.4.1 was published in January 11, 2011. This new version combines two versions of a control panel used for committing fraud and managing compromised systems. These trends indicate that self-learning and self-updating by observing system anomalies and behaviour patterns is much warranted in malware detection systems of the future.

According to Trend Micro (Micro's, 2009), the Zeus attack continues to persist due to the following file compression variant: Compressed or packed for executable (UPX), used compression software that is capable of encrypting actual.EXE application code. The result makes applications difficult to analyze.

Furthermore, Zeus has Rootkit capabilities, which allows variants to hide themselves and their malicious routines from users in order to perform stealth routines. Also, Zeus has varying social engineering techniques and has an effective "business model" which means an attacker can develop the software and create crime toolkits, then sell the kits through an established network of money mules to transfer money.

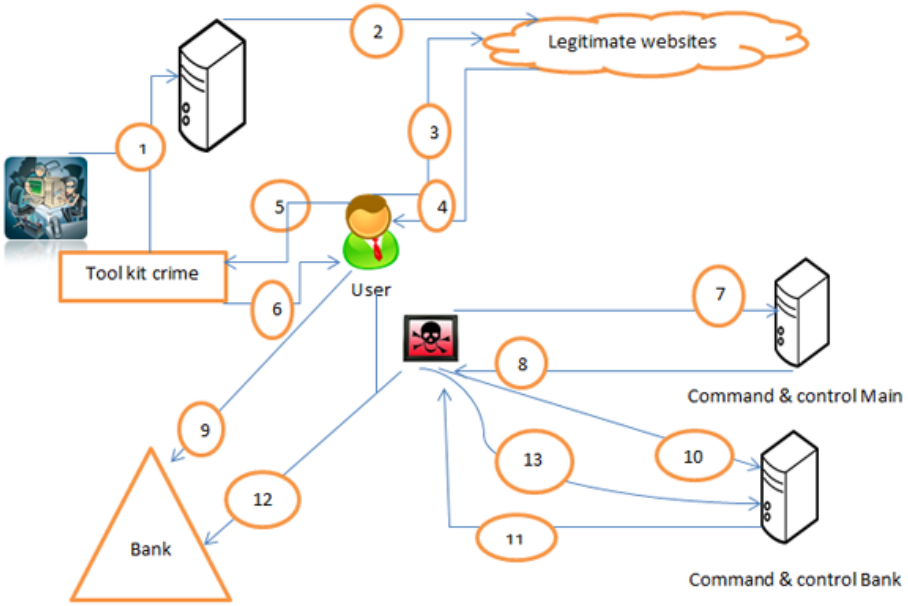
Figure 4 shows how attackers compromise a web application. Firstly, an attacker deploys a malicious string to the web application servers. Secondly, once a malicious application is published on a legitimate site, the user clicks a link on the infected website. Thirdly, after the user clicks on the infected web site, the website redirects the user to the malicious kit. Fourthly, once the user is redirected to the malicious kit that contains the botnet, the users' enter their personal identification on the infected web site. Fifthly, the Trojan reports for a new bot to the C&C (control and command server). Finally, the C&C sends instructions to the Trojan to observe the user, especially the user's access to an e-bank system or financial institution.

FUTURE RESEARCH DIRECTIONS

Researchers have proposed a wide range of techniques to address malware on web applications. However, many of these solutions have limitations that affect their performance and practicality. There are a number of reports showing SQL injection is steadily increasing and that clients are still confused when choosing the appropriate tools because of the difficulty in integrating information vulnerabilities. For example, one of the most common solutions is based on signature detection that has proven unsuccessful due to the development of new evasion techniques by hackers. Most SQL query statements are legitimate keywords that can be displayed if the user makes a HTTP request. The result is a high rate of false positives.

Researchers have been proposing anomaly-based detection systems to overcome the limitation of signature-based detection systems. The main feature of an anomaly-based detection system is its ability to detect unknown attacks (i.e. Zero day attack). Nevertheless, most intrusion detection systems used today are signature-based, with a few anomaly based detection systems used. The

Figure 4. A malicious upload to a web application



reason for this is signature-based detection systems are easier to implement, easier to set up and more manageable than anomaly-based detection (Bolzoni, 2009; Kruegel & Toth, 2003).

However, the weakness of exiting mechanism to deal of these threats, calling for new mechanisms to detect advance threats that target web application.

In general, prevention and detection web malware is an active research field in both industry and academia, and there are many solutions and tools that have been implemented. Unfortunately not all of them guarantee a high level of security on web applications. Focus should be on the development of a mechanism that is easy to implement, with no modification of code needed, yet is efficient and has a high performance rate for existing web applications

CONCLUSION

Increasingly, web applications are developed over the internet. Securing these web applications is becoming increasingly important as they hold critical security features. The malicious writer takes advantage of a web application vulnerability to promote his attack. These attacks can target the financial and non-financial sectors such as Facebook, Twitter, Gmail, and Yahoo! and against cloud application such as Salesforce and Google Apps. On the other hand, the new generation of the crime toolkits have ability to evade from exiting detection by Polymorphism and Dynamic injection to web page. Additionally, hackers are always trying different and new techniques to hack into your computer system.

Preventing attacks is not a sufficient solution in itself and intrusion detection systems in web applications are necessary, otherwise hackers will repeatedly attack web applications until they find vulnerability. Also, the current exiting detection system and the traditional firewalls are not enough solution to protect your computer system from the latest challenges. Web application protection is still necessary to protect against advanced zero day threat. Therefore, Security awareness and user education are important steps that really help to prevent most of the online attacks.

REFERENCES

Anderson, R., & Moore, T. (2007). Information security economics–And beyond. *Advances in Cryptology-CRYPTO*, *2007*, 68–91.

Bandhakavi, S., Bisht, P., Madhusudan, P., & Venkatakrishnan, V. (2007). CANDID: Preventing sql injection attacks using dynamic candidate evaluations. *ACM Conference on Computer and Communications Security* (pp. 12-24).

Barnes, K., Marateo, R. C., & Ferris, S. P. (2007). Teaching and learning with the net generation. *Innovate Journal of Online Education, 3*(4).

Binsalleeh, H., Ormerod, T., Boukhtouta, A., Sinha, P., Youssef, A., Debbabi, M., & Wang, L. (2010, 17-19 August). *On the analysis of the Zeus botnet crimeware toolkit.* Paper presented at the Privacy Security and Trust (PST).

Bisht, P., Madhusudan, P., & Venkatakrishnan, V. (2010). CANDID: Dynamic candidate evaluations for automatic prevention of SQL injection attacks. *ACM Transactions on Information and System Security*, *13*(2), 1–39. doi:10.1145/1698750.1698754

Buehrer, G., Weide, B. W., & Sivilotti, P. A. G. (2005). *Using parse tree validation to prevent SQL injection attacks*. Paper presented at the Proceedings of the 5th international workshop on Software engineering and middleware. Lisbon, Portugal.

Corporation, M. (2010). Common vulnerabilities and exposures. Retrieved April, 2011, from http://cve.mitre.org/

Gould, C., Su, Z., & Devanbu, P. (2004). JDBC checker: A static analysis tool for SQL/JDBC applications. In *Proceedings of the 26th International Conference on Software Engineering*.

Halfond, W. G. J., & Orso, A. (2006). Preventing SQL injection attacks using AMNESIA. In *Proceedings of the 28th International Conference on Software Engineering*.

Halfond, W. G. J., Orso, A., & Manolios, P. (2006). Using positive tainting and syntax-aware evaluation to counter SQL injection attacks. In *Proceedings of the 14th ACM SIGSOFT International Symposium on Foundations of Software Engineering*.

Huang, Y.-W., Yu, F., Hang, C., Tsai, C.-H., Lee, D.-T., & Kuo, S.-Y. (2004). *Securing web application code by static analysis and runtime protection*. Paper presented at the Proceedings of the 13th international conference on World Wide Web, New York, NY, USA.

IndraniBalasundaram, & Ramaraj. (2011). An approach to detect and prevent SQL injection attacks in database using web service. *International Journal of Computer Science and Network Security*, *11*, 197–205.

Jie, W., Phan, R. C. W., Whitley, J. N., & Parish, D. J. (2010). Research on Multi-Level Security Framework for OpenID. In *Third International Symposium on Electronic Commerce and Security*.

Kramer, S., & Bradfield, J. C. (2010). A general definition of malware. *Journal in Computer Virology*, *6*(2), 105–114. doi:10.1007/s11416-009-0137-1

Martin, M., Livshits, B., & Lam, M. S. (2005). Finding application errors and security flaws using PQL: A program query language. *ACM SIGPLAN Notices*, *40*(10), 365–383. doi:10.1145/1103845.1094840

McClure, R. A., & Krüger, I. H. (2005). SQL DOM: Compile time checking of dynamic SQL statements. In *27th International Conference on Software Engineering*.

Nguyen-Tuong, A., Guarnieri, S., Greene, D., Shirley, J., & Evans, D. (2005). Automatically hardening web applications using precise tainting. *Security and Privacy in the Age of Ubiquitous Computing*, 295-307.

OWASP. (2010). The top 10 most critical web application security risks. Retrieved April 2011, from owasp.org

Peng, C. S., Chen, S. K., Chung, J. Y., Roy-Chowdhury, A., & Srinivasan, V. (2010). Accessing existing business data from the World Wide Web. *IBM Systems Journal*, *37*(1), 115–132. doi:10.1147/sj.371.0115

Pietraszek, T., & Berghe, C. (2006). Defending against injection attacks through context-sensitive string evaluation. In Valdes, A., & Zamboni, D. (Eds.), *Recent advances in intrusion detection* (*Vol. 3858*, pp. 124–145). Springer. doi:10.1007/11663812_7

RSA. (2011). The current state of cybercrime and what to expect in 2011. USA: EMC corporation.

Su, Z., & Wassermann, G. (2006). *The essence of command injection attacks in web applications*. Paper presented at the Conference record of the 33rd ACM SIGPLAN-SIGACT symposium on Principles of programming languages. Charleston, South Carolina, USA.

Torrano-Gimenez, C., Perez-Villegas, A., & Alvarez, G. (2010). WASAT-A new web authorization security analysis tool. *Web Application Security*, 39-49.

Trend Micro. (2009). 2009's most persistent malware threats. Retrieved February 3, 2011, from http://us.trendmicro.com/imperia/md/content/us/trendwatch/researchandanalysis/2009s_most_persistent_malware_threats__march_2010_.pdf

Trusteer. (2009). Banking malware Zeus sucessfully bypasses anti-virus detection. Retrieved March, 2011, from http://www.ecommerce-journal.com/news/18221_zeus_increasingly_avoids_pcs_detection

Valeur, F., Mutz, D., & Vigna, G. (2005). A learning-based approach to the detection of SQL attacks. In Julisch, K., & Kruegel, C. (Eds.), *Detection of intrusions and malware, and vulnerability assessment* (*Vol. 3548*, pp. 533–546). Springer. doi:10.1007/11506881_8

Valli, C., & Brand, M. (2008). *The malware analysis body of knowledge*. MABOK.

Vella, K. J. (2007, Jun 11). Web applications: What are they? What about them? Retrieved 2011, from http://www.windowsecurity.com/articles/Web-Applications.html?printversion

W3schools. (2011). Introduction to SQL. Retrieved April, 2011, from http://www.w3schools.com/sql/sql_intro.asp

Yu, W. D., Aravind, D., & Supthaweesuk, P. (2006). *Software vulnerability analysis for web services software systems*.

ADDITIONAL READING

Alazab, M., Venkataraman, S., & Watters, P. (2010). Towards understanding malware behaviour by the extraction of api calls. *Cybercrime and Trustworthy Computing Workshop*.

Alazab, M., Ventatraman, S., Watters, P., Alazab, M., & Alazab, A. (2011). *Cybercrime: The case of obuscated malware.* Paper presented at the 7th International Conference on Global Security, Safety & Sustainability. Thessaloniki, Greece.

Clarke, J. (2009). *SQL injection attacks and defense*. Syngress.

Stuttard, D., & Pinto, M. (2011). *The web application hacker's handbook: Finding and exploiting security flaws*. Wiley.

KEY TERMS AND DEFINITIONS

Authentication: The techniques that use to verify and identity of users to the Web application and website, typically based on a username and password.

Buffer Overflow: A buffer overflow attack happens when attackers overload systems to damage the execution stack of a web application. In this attack the attacker sending malicious input to web application, an attacker can cause the web application to execute arbitrary code causing damage over all the system.

Cross-Site Request Forgery (CSRF): A type of attack on web application that takes advantage of a lack of authorization of Web application. This attack permits an attacker to perform malicious commands without user's knowledge. In this attack the attackers cheating an internet user into clicking on a malicious link which is planned to perform a malicious activity on behalf of the users. For example, a user's may click on a malicious link. Once the users clicked on this linked.The linked redirect the user to illegitimate web site that forces the victim to transfer money from the victim's bank account to an attacker's bank account.

Cross-Site Scripting (XSS): A type of attack on web application which occurs when an attacker inserts malicious string or code into a hyper link that appears in legitimate web site. When internet user clicks on this link, the embedded programming is submitted as part of the client's Web request and can execute on the user's computer. The result from this is permitting the attacker to gain critical information.

Insecure Cryptographic Storage: Cryptographic Storage regularly occurs when the developers encrypt the information wrongly. As the web application encrypts sensitive information in a database to avoid revelation to end users. Though, the database is having functionality to decrypt queries against the sensitive information. The system should have been configured to allow only back end applications to decrypt them, not the front end web application. Otherwise, allowing the hackers to attack the database to steal all the critical information.

Security Misconfiguration: A type of attack on web application which occurs when an attacker exploit configuration weaknesses found in web applications. Many web applications have unnecessary features, such as default username. These features may help an attacker to lunch his activities in order to get access to personal information.

SQL Injection Attack (SQLIA): A type of attack on web application which occurs when an attacker inputs malicious strings as parameters in legitimate SQL statement. The SQLIA allow the hacker to gain complete access to the database server a serious threat to the web application.

Web Application Vulnerability: Any weakness in web application architecture, web application design, web application configuration, web application code. That permits an attacker to comprise a web application.

Web Malware: Malicious software intended to harm an internet users system without the user's permission. "Malware" is the general term that describes the various kinds of threats to the computer system.