

INTRODUCTION GENERAL

INTRODUCTION GENERAL

SQL injection is the vulnerability That results when you give an attacker the ability to influence the Structured Query Language (SQL) queries that an application passes to a back-end database. By being able to influence what is passed to the database, the attacker can leverage the syntax and capabilities of SQL itself, as well as the power and flexibility of supporting database functionality and operating system functionality available to the database. SQL injection is not a vulnerability that exclusively affects Web applications; any code that accepts input from an untrusted source and then uses that input to form dynamic SQL statements could be vulnerable (e.g., “fat client” applications in a client/server architecture).

A lot of works have been engaged to remedy to SQL attacks, the goal is to build a scanner that can reveal the vulnerability included in the web sites. Several approaches are proposed; they can be categorized. In our research we focus on the structure of the web page rather than the content, so, we proposed a method based on the variation of the number of the tags between the original page and the resulted one after being injected by an SQL attack.

The experiments realized to evaluate the proposed detection SQL injection method is done on web sites proposed by the researchers’ community in the area. Three scanners are considered in order to appreciate the performance of our scanner. The results reveal a better performance.

The dissertation is structured as follow, In the first chapter we have seen a view in web security is presented and the major threats are discussed, In the second chapter a view in SQL injection, what is SQL injection, understanding SQL Injection, how it happens and more things about SQL injection is a full and comprehensive definition, In the third chapter we discuss the proposed approach is motivated and explained and all SQL Injection various types of attacks are explained, and a survey of the famous detection methods are listed, In the last chapter is like a series of experiments are done to evaluate the performance of the method.

CHAPITRE 01

WEB SECURITY

1. Introduction

The internet was considered to be safe in the early days, but after research by some people on its safety it was found to be very unsafe.

What is security? What circumstances will cause a security problem? How do we view security issues? Only by clearly understanding these basic issues can we get to understand all defense technologies and procedures we carry out.

When addressing security issues, all must people say "there is no straightforward solution", In general, people try to avoid trouble. Security is a troublesome thing, but we cannot escape it forever. Anyone who wants to solve security problems once and for all resorts to the wishful thinking and is unrealistic.

Since the Internet comes with security issues, attack and defense technologies undergo constant development. At the micro level, in a given period, a party may have prevailed, but from a macro point of view, a period of always using attack or defense techniques cannot always be effective. This is because attack techniques are constantly being upgraded to keep pace with developments in defense technology; the two are mutually reinforcing a dialectical relationship. Attack is a means of defense against the continuous development of technology and committing the error of disregarding the changing circumstances. In the area of security, there is no silver bullet.

Several security vendors show the user some really good blueprint when selling their products; it seems omnipotent, and the user can sleep well after purchase. But in fact, the security products themselves also need to be constantly upgraded and need someone to operate them. The product itself also needs a metabolic process otherwise it will not be effective. The automatic update feature in modern Internet products has become a standard configuration; a dynamic product will always continue to improve on its own.

When Vista released, Microsoft had vowed to ensure that this is the most secure operating system. We see the effort put in by Microsoft as security issues in Vista are much less than in its predecessors (Windows XP, Windows 2000, Windows 2003, etc.), especially with regard to high-risk vulnerabilities. In spite of this, hackers have managed to attack Vista in the Pwn2Own competition in 2008. The Pwn2Own contest is held every year.

Hackers continue to research and find new attack techniques, as does the defense side. Microsoft, in recent years, has improved the safety of products; it has

taken into account the safety aspects throughout the development process and the security checks across the entire software life cycle and has proven that this is viable. Each product now has a sustained implementation of stringent security checks, which is a valuable experience that Microsoft has taught the industry. Safety checks need to be constantly upgraded to counter new attack detection and prevention programs.

1.1 Definition of security?

Security Is protection against adversaries from those who would do harm, intentionally or otherwise is the objective. National security, for example, is a multilayered system that protects the sovereignty of a state, its assets, its resources, and its people. Achieving the appropriate level of security for an organization also requires a multifaceted system. A successful organization should have the following multiple layers of security in place to protect its operations:

- **Physical security**, to protect physical items, objects, or areas from unauthorized access and misuse
- **Personnel security**, to protect the individual or group of individuals who are authorized to access the organization and its operations
- **Operations security**, to protect the details of a particular operation or series of activities
- **Communications security**, to protect communications media, technology, and content
- **Network security**, to protect networking components, connections, and contents
- **Information security**, to protect the confidentiality, integrity and availability of information assets, whether in storage, processing, or transmission. It is achieved via the application of policy, education, training and awareness, and technology.

Security is fundamentally about protecting assets. Assets may be tangible items, such as a Web page or your customer database or they may be less tangible, such as your company's reputation.

Security is a path, not a destination. As you analyze your infrastructure and applications, you identify potential threats and understand that each threat presents a

degree of risk. Security is about risk management and implementing effective countermeasures.

1.2 The foundations of security

Security relies on the following elements:

- **Authentication**

Authentication addresses the question: who are you? It is the process of uniquely identifying the clients of your applications and services. These might be end users, other services, processes, or computers. In security parlance, authenticated clients are referred to as *principals*.

- **Authorization**

Authorization addresses the question: what can you do? It is the process that governs the resources and operations that the authenticated client is permitted to access. Resources include files, databases, tables, rows, and so on, together with system-level resources such as registry keys and configuration data. Operations include performing transactions such as purchasing a product, transferring money from one account to another, or increasing a customer's credit rating.

- **Auditing**

Effective auditing and logging is the key to non-repudiation. Non-repudiation guarantees that a user cannot deny performing an operation or initiating a transaction. For example, in an e-commerce system, non-repudiation mechanisms are required to make sure that a consumer cannot deny ordering 100 copies of a particular book.

- **Confidentiality**

Confidentiality, also referred to as *privacy*, is the process of making sure that data remains private and confidential, and that it cannot be viewed by unauthorized users or eavesdroppers who monitor the flow of traffic across a network. Encryption is frequently used to enforce confidentiality. Access control lists (ACLs) are another means of enforcing confidentiality.

- **Integrity**

Integrity is the guarantee that data is protected from accidental or deliberate (malicious) modification. Like privacy, integrity is a key concern, particularly for data passed across networks. Integrity for data in transit is typically provided by using hashing techniques and message authentication codes.

- **Availability**

From a security perspective, availability means that systems remain available for legitimate users. The goal for many attackers with denial of service attacks is to crash an application or to make sure that it is sufficiently overwhelmed so that other users cannot access the application.

1.3 Threats, vulnerabilities, and attacks defined

A threat is any potential occurrence, malicious or otherwise, that could harm an asset. In other words, a threat is any bad thing that can happen to your assets.

A vulnerability is a weakness that makes a threat possible. This may be because of poor design, configuration mistakes, or inappropriate and insecure coding techniques. Weak input validation is an example of an application layer vulnerability, which can result in input attacks.

An attack is an action that exploits a vulnerability or enacts a threat. Examples of attacks include sending malicious input to an application or flooding a network in an attempt to deny service.

To summarize, a threat is a potential event that can adversely affect an asset, whereas a successful attack exploits vulnerabilities in your system.

1.4 How to implement safety assessment?

Let us now begin to analyze and resolve security issues. A security assessment process can be divided into four stages: asset classification, threat analysis, risk analysis, and conformance to design (Figure I.1).

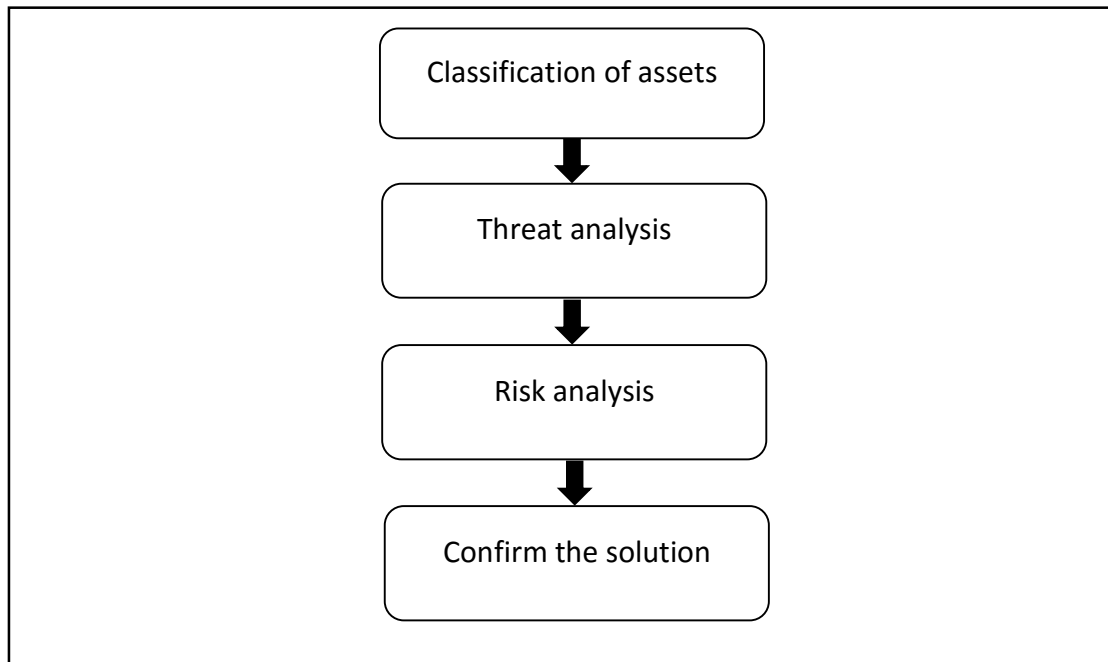


Figure 1.1 - Security assessment process.

1.4.1 Asset Classification

Asset classification is the basis for all work; this will make clear the objective that you want to protect. We mentioned formerly the three elements of security, confidentiality, and integrity of all data related to the availability, for which I use the term resources. Resources, as a concept, describes a more extensive range than data, but in many cases, the availability of resources can be understood as the availability of data. If the infrastructure of the Internet has been relatively complete, the core of the Internet is actually driven by the user data—user-generated business operations data. For Internet companies, in addition to some fixed assets, such as servers and other hardware, the core value is user data. -So the core issue of Internet security is the issue of data security. Does security relate to asset evaluation? Of course, Internet companies have an asset classification, the data classification. Some companies are most concerned about customer data; some companies are even more concerned about employee data; the focus is different depending on the respective business. In the process of asset classification, we need to communicate with the persons in charge of each business unit to understand the company's

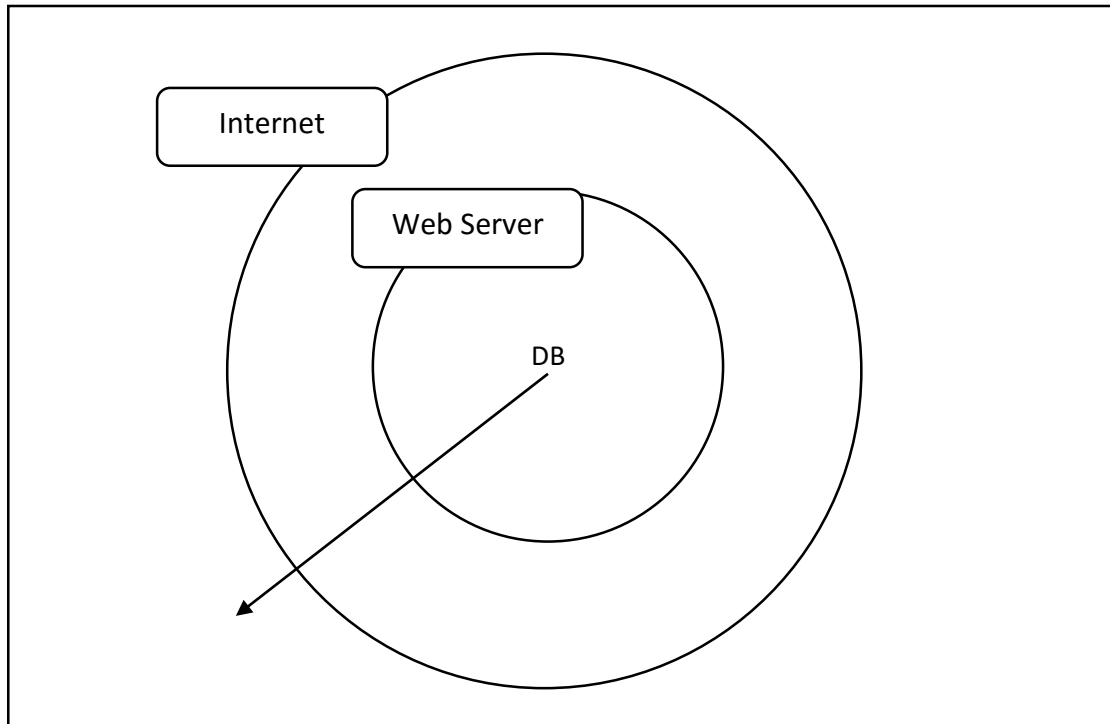


Figure 1.2- Simple website trust model.

Most important asset. After the interviews, security departments become familiar with and understand the company's business and its data, as well as the importance of different data, which directs the follow-up assessment process. After the completion of asset classification, the objective forms into a rough idea; the next step is to divide the trust domain and trust boundaries. Usually, we use one of the simplest divisions, which is based on network logic. For example, if the most important data is in the database, then we circle the database server; the web application can read/ write data from the database and external services, and then the web server is circled; most of what is outside the Internet cannot be trusted (Figure 1.2). This is the simplest example; we will encounter many more complicated problems than this in practice. For example, if the two similar applications exchange data, then we must consider the interaction—whether it is credible, whether we should draw a boundary between the two applications, and finally whether data flowing through the boundary need security checks.

1.4.2 Threat analysis

Having provided the definition of trusted domain, we can now determine where the danger comes from. In the security field, harmful elements to the source are known as threats, and losses that may occur are known as risks. Certain risks are associated with losses, but many professional security engineers often confuse these two concepts and mistake their identity in a written document. Distinguishing these two concepts can help us go to the next two stages, “threat modeling” and “risk analysis,” which are closely related. What is threat analysis? Threat analysis is finding out all threats. How to find them? By brainstorming. Of course, there are some other more scientific methods as well, such as the use of models to help us identify threats; this process can avoid omission and is known as threat modeling. We will introduce a threat modeling method called the STRIDE model, which was first proposed by Microsoft. STRIDE is made up of the initial letters of six words (see Table 1).

Thread	Definitions	Corresponding Security Attributes
Spoofing	Impersonate another person's identity	Identification
Tampering	Modify data or code	Integrity
Repudiation	Deny done before	Nonrepudiation
Information disclosure	Disclosure of confidential information	Confidentiality
Denial of service	Denial of service	Availability
Elevation of privilege	Unauthorized license	Authorization

Table1.1- Analysis of Threat Using the STRIDE Model

Threat analysis is a very important stage; a lot of time is also needed to regularly review and update the existing model. There may be many threats, but not all of them can cause a major loss. How much harm can a threat cause and how to measure this? This is why it is necessary to take into account the risk, which is done during the risk analysis process. At this stage, there are models that can help us scientifically assess the different types of risks.

1.4.3 Risk analysis

Risk is composed of the following factors:

Risk = Probability * Damage Potential or

$$\text{Risk} = \text{Threat} * \text{Vulnerability} / \text{Against} - \text{Measure}$$

The threat represents the type of action likely to harm in absolute terms, while the **Vulnerability**, represents the exposure face of the threat in a particular context. Finally, **the measure** is against all the means implemented prevention of threat. It is necessary to identify potential threats, and therefore to know and plan how to do the enemy to better understand how it is possible to limit risks

There are many standard methods of analysis, risk management and developing a security policy, the example of MEHARI or EBIOS.

Level	High (3)	Medium (2)	Low (1)
Damage potential	Get to fully verify the permissions; perform administrative operations; illegal to upload files	Disclosure of sensitive information	Disclosure of some other information
Reproducibility	Attacker can freely attack again	The attacker can repeat the attack, but within time constraints	Attacker has difficulty in repeating the attack process
Exploitability	Beginner can master the attack methods in a short time	Skilled attacker can complete the attack	Exploits are very harsh
Affected users	All users, the default configuration, the key user	Some users, no default configuration	Least number of users, anonymous users
Discoverability	The vulnerability is very conspicuous; attack conditions are easy to obtain	In the private area, some people can locate the vulnerability, but need to dig deeper to find it	Is extremely difficult to find the vulnerability

Table1.2- Level of Risk a Threat Can Lead To

When we consider the security issues, we need to weigh the possibility of occurrence in order to correctly determine the risk factor. How do we assess risk scientifically? I will here introduce the DREAD model, which has also been proposed by Microsoft. DREAD, like STRIDE, is also made up of the initials of words; it allows us to judge what level of risk a threat can lead to (Table 1.2). In the DREAD model,

each factor can be divided into three grades, high, medium, and low. In Table 1.1, high, medium, and low grades (3, 2, 1 scores, respectively) represent the weight; thus we can calculate the specific risk value of a threat.

DREAD is used to form part of the thinking behind the risk rating, and is used directly to sort the risks

DREAD is used to compute a risk value, which is an average of all five elements:

$$\text{Risk}_{\text{DREAD}} = (\text{DAMAGE} + \text{REPRODUCABILITY} + \text{EXPLOITABILITY} + \text{AFFECTED USERS} + \text{DISCOVERABILITY}) / 5$$

NOTE: This produces a number between 0 and 10. The higher the number, the more serious the risk.

For example, if the KMT brigade, after threat modeling, found two principal threats to circumventing the mountain a threat of stormy weather on the main path and a treacherous trail to contend with the corresponding risks could be calculated as follows:

The main path:

$$\text{Risk} = D(3) + R(3) + E(3) + A(3) + D(3) = 3+3+3+3+3=15$$

The mountain trail:

$$\text{Risk} = D(3) + R(1) + E(1) + A(3) + D(1) = 3+1+1+3+1=9$$

The level of risk is defined as follows:

High-risk: 12~15

Medium-risk: 8~11

Low-risk: 0~7

Thus, the main path is the highest risk and bound to be heavily guarded; however, the mountain trail too turns out to be at risk and therefore cannot be ignored. The reason why we regard the mountain trail as a medium-level risk in this model is because whenever it is broken, the loss is too large to afford. Let us now take a look at the security assessment of the overall process. Remember at all times that the model is the same; it is only the people who keep changing. No matter

How good the model, we need trained people to use it. Once the attack surface and risk level are determined, one needs an experienced hand. This is where a security engineer adds value. Like STRIDE and DREAD, there may be many different standards corresponding to different models; as long as we feel that these models can help us, we can use them. However, the model can only play a supporting role—decisions will ultimately have to be made by people.

1.4.4 Security programs Design

The output of security assessment is the security solution. Solutions must be target-specific, that is, they must be broken down by asset class, threat analysis, and risk analysis. To design solutions is not difficult; what is difficult is to design a good solution. Designing a good solution is the true test for security engineers.

2 Web Security

In this part takes a closer look at how security works and how it applies to web applications. If your application is on the Internet, it is on the front lines of your network. It is like a door to the outside world that allows visitors to come in and check out whatever you have to offer. Your application needs to be secure, and you need to be aware of the dangers an application can open to your network.

2.1 Targets and impacts of web attacks

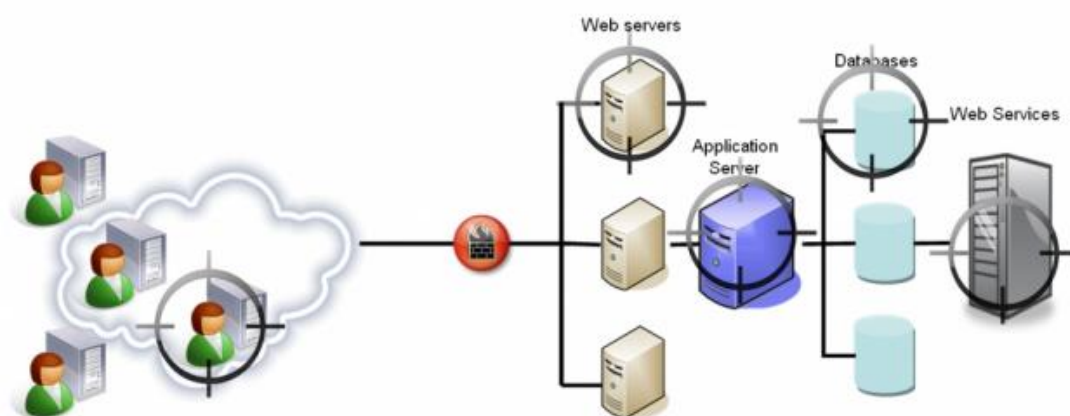


Figure 1.3- Targets of Web attacks

User: essentially, attacks against the user seeking to recover authentication credentials to impersonate and realize all actions that latter could have done.

Web server: is strategically important and the most frequent impacts are:

- Access to confidential data
- Changing Content
- Adding Bad Content
- Control the Web server (command execution)

The web application: in this case, the aim will be to ensure that the Web application is a different behavior than expected. For example: change the price of a product.

Databases: they are a prime target for hackers. The impacts range from theft, deletion or alteration of data.

Les services Web: they have a strong interaction with the database, which makes them an attractive target.

2.2 The web application

So, with a decade of web pages behind us the Web now is like a college graduate— beaming with excitement and curiosity and looking for a new job. Companies, lured by “free publishing” have flocked to the Web and are demanding more. Commerce! By the year 2000 web applications serving dynamic data were showing up everywhere and fueling the great climax of the dot-com era. For web pioneers, led by the likes of Amazon, eBay, Yahoo!, and Microsoft, the electronic world was their oyster. Web server vendors and technology providers, faced with the demands of an ever growing dynamic Web, were breaking new ground and innovating a whole new type of server. Figure 1-7 shows a typical application server environment.

2.3 96% of Tested Applications Have Vulnerabilities (Report of CENZIC) [10]

The application layer continues to be a soft target with increasing cyberattacks. 96% of all applications tested in 2013 have one or more serious security vulnerabilities. The median number of vulnerabilities per app has elevated to (14) from last year’s count of (13).

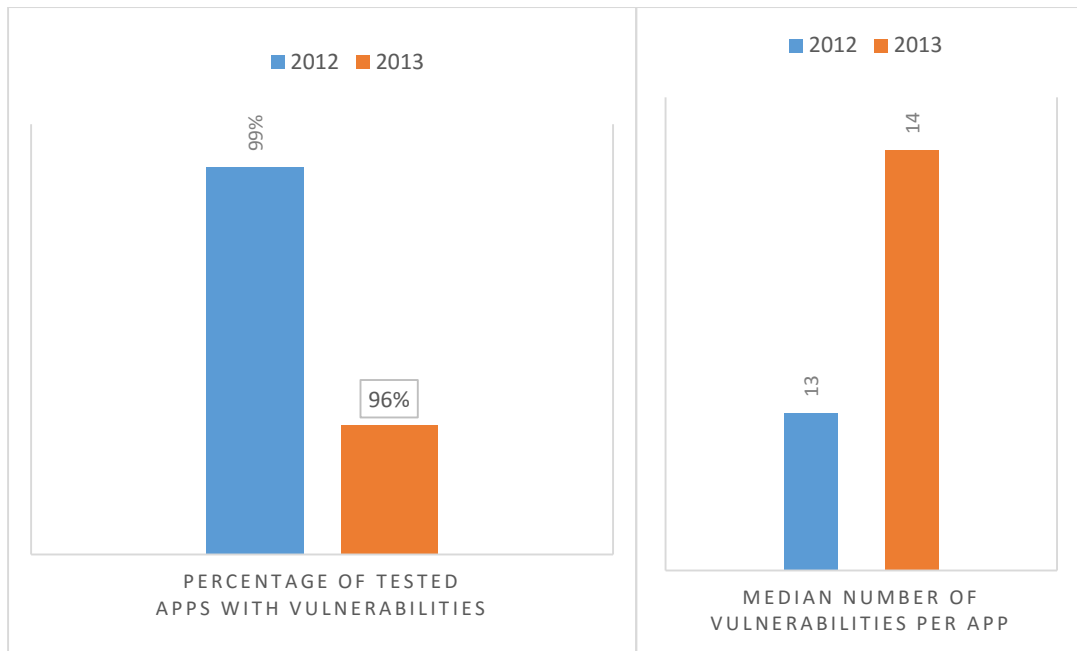


Figure 1.4 -: 2012-2013 Summary Statistics: Application Vulnerabilities

2.4 Top 10 OWASP

Top 10 OWASP
A1 - Injection
A2 - Broken Authentication and Session Management
A3 - Cross-Site Scripting (XSS)
A4 - Insecure Direct Object References
A5 - Security Misconfiguration
A6 - Sensitive Data Exposure
A7 - Missing Function Level Access Control
A8 - Cross-Site Request Forgery (CSRF)
A9 - Using Components with Known Vulnerabilities
A10 - Unvalidated Redirects and Forwards
Table 1.3 - List of security risks[11]

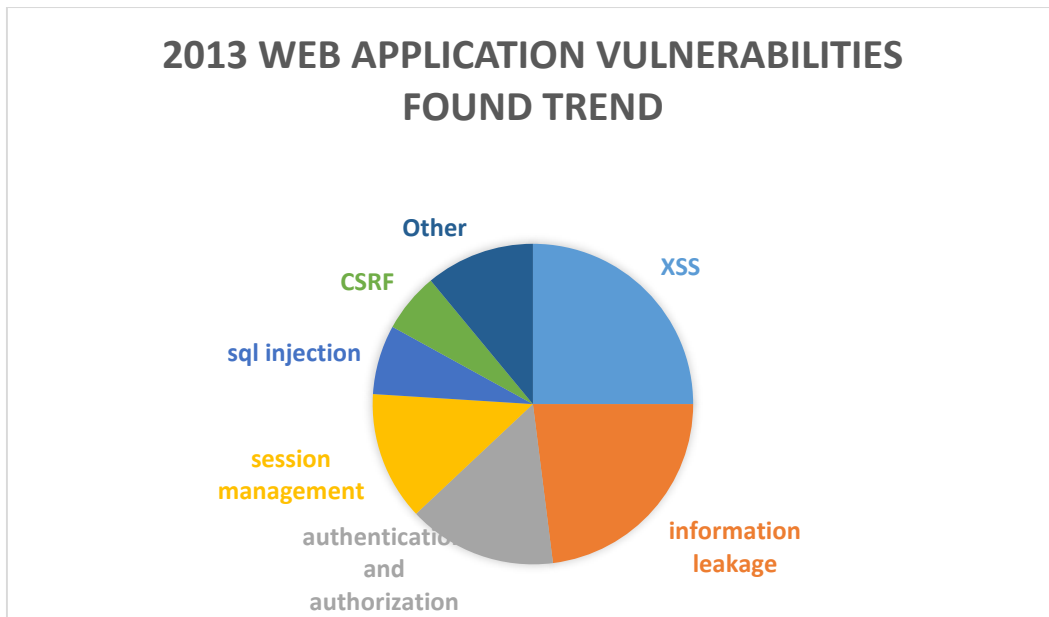


Figure 1.5-: 2013 Web Application Vulnerabilities Found Trend

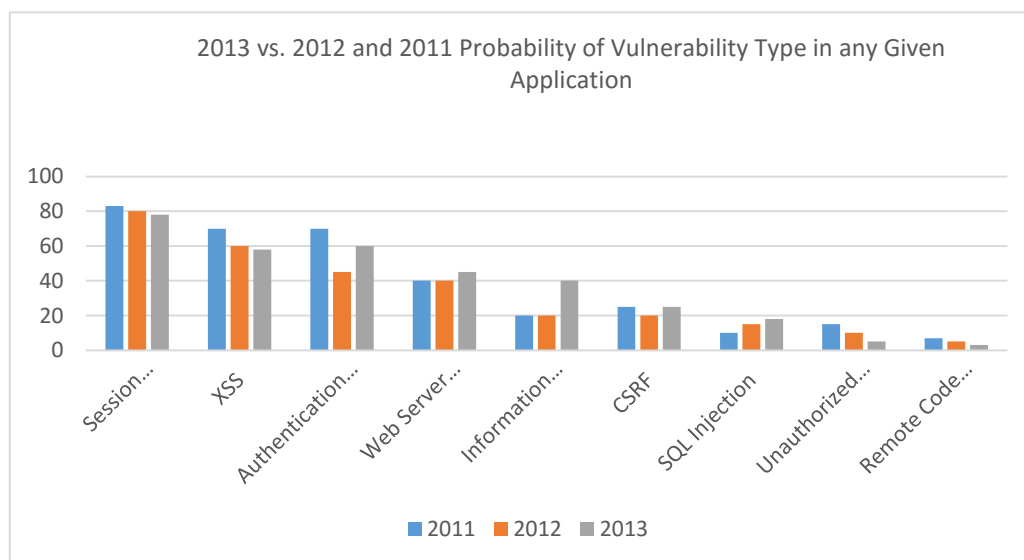


Figure 1.6- 2013 vs. 2012 and 2011 Web Application Vulnerabilities Found Trends

2.5 Understanding the threats of Insafe web applications

Vulnerabilities in web applications may take dozens of forms. Many attacks use fault injection, which exploits vulnerabilities in a web application's syntax and semantics. In simple terms, an attacker manipulates data in a web page Uniform Resource Indicator (URL) link to force an exploitable malfunction in the application. The two most common varieties are SQL injection and cross-site scripting. Later we'll dive into details of how these and other vulnerabilities work (and how to get rid of them!). For now, here's the basic idea. Consider a typical URL:

<http://example/foo.cgi?a=1>

Executing a SQL injection exploit simply requires modifying the URL. All that's needed might be one odd character to trigger a successful exploit, such as adding an apostrophe to the end of the URL:

<http://example/foo.cgi?a=1'>

The 'successful' outcome can give an attacker control over the application and easy access to the server, database, and other back-end IT resources. Needless to say, this access can trigger disastrous results.

Data is the object of desire for attackers – particularly data that converts their efforts into cash. The most lucrative source of this data is a business database containing information that can be sold or used directly by an attacker for profit. Business databases are like pots of gold brimming with bankable opportunities – all in one location. Some of these include:

- Strategic business plans.
- Product plans and other intellectual property.
- Competitive analysis.
- Employee rosters and their personal information.
- Confidential data from business partners.
- Confidential customer data.

Confidential customer data may be the highest value data because it's easy to sell and leverage. It includes personally identifiable data such as names, addresses, birth dates, payment card Primary Account Numbers, email addresses, and so on. Some of the worst data breaches have included the theft of millions of records containing this information. The massive scale of instant damage is unprecedented.

Web application attacks may also target individuals, one by one. Some attacks are executed by infiltrating a trusted website, which then injects malware into computers used by unsuspecting visitors. The malware might redirect links to rogue sites that steal personal information directly from the user's PC. It could trick users into revealing confidential passwords or payment card data. It may even hijack the user's PC and transform it into a spam server or other nefarious mechanism aimed to further the attacker's goals.

Either way, successful attacks on web applications can result in highly negative fallout.

2.6 Web security Rises

Web attack techniques can be divided into several stages. In the Web 1.0 era, people were more concerned about server-side dynamic scripting security issues, such as an executable script (commonly known as web shell) uploaded to the server to obtain permission. The popularity of dynamic scripting languages and insufficient cognition of web technologies on security issues in the early stages caused a lot of issues, such as the PHP language still having to rely on good code specifications to ensure that no file contains a loophole, but not on the language itself to prevent the occurrence of such security issues.

SQL injection is a milestone in the history of web security; it first appeared in about 1999 and quickly became a major threat to web security. Programmers worked hard to amend the loopholes in the system and to contain the attacks, as otherwise hackers can access important and sensitive data through SQL injection attacks and can even access the system through the database. SQL injection attack is as effective, if not better, than a direct attack, which makes it popular with hackers. Vulnerability to SQL injection attacks is therefore still an important concern in the web security field.

XSS (cross-site scripting) attack is another milestone in the history of web security. In fact, XSS and SQL injection appeared almost at the same time, but the former came into prominence only in about 2003. After the My Space XSS worm incident, the web security community took cognition of the large threat posed by XSS; it even made it to the top of the **OWASP** 2007 Top 10 threats. Along with the rise of Web 2.0, XSS and CSRF attacks have become even more powerful. Web attacks shifted from the server side to that of clients, browsers, and users. With more options at their disposal, hackers covered every aspect of the web.

With the growth of web technology and the Internet, several new scripting languages have developed such as Python, Ruby, Node JS, and so on. Besides, the development of mobile phone technology and the rise of the mobile Internet have brought new opportunities and challenges to HTML 5, which require web security technology to constantly evolve and remain up to date.

2.7 Educating Security in Web Applications

The good news is that most prevalent web application vulnerabilities can be easily detected with an automated scanner.

Scanning web applications acts to supplement and compliment manual testing by performing likely attacks on target applications.

Scanning web applications has even become a strategic requirement in some regulations.

For example, the Payment Card Industry Data Security Standard (PCI DSS) version 2.0 now requires all merchants accepting payment cards to pass quarterly scans for vulnerabilities in web applications.

Automated scanning can provide many benefits, including:

- Discovering and cataloging all web applications in your Enterprise.
- Lowering the total cost of operations by automating Repeatable testing processes.
- Identifying vulnerabilities of syntax and semantics in Custom web-applications.
- Performing authenticated scanning.
- Profiling the target application.
- Ensuring accuracy by effectively reducing false positives and false negatives.

2.8 Web application security lifecycle (S-SDLC)

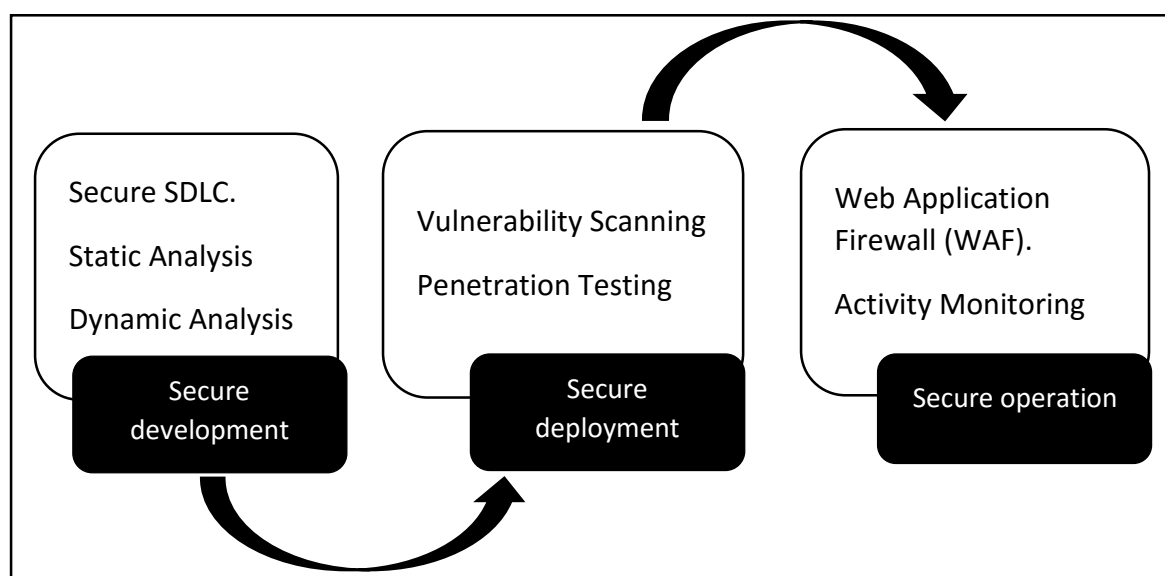


Figure 1.7- Web application security lifecycle (Source: Securosis).

“Securosis is the world’s leading independent security research and advisory firm, offering unparalleled insight and unique value to meet the challenges of managing security and compliance in a Web 2.0 world.”

2.8.1 Secure development

The Secure Development phase is all about building security into web applications right from their inception. This is what commercial software companies do because customers expect what they license to be secure. This isn’t an automatic process, however, so as your organization strives to be smart with its efforts to secure applications on its websites, it needs to provision for several elements. These include:

- **Secure SDLC**

The software development life cycle is where your organization establishes best practices for secure coding, testing, and proving that software is safe.

- **Static Analysis.**

This entails line-by-line analysis of code for errors or improper implementations of open source modules. So-called ‘white box’ tools (testing code structure) can help automate this process

- **Dynamic Analysis**

Tools of the ‘black box’ variety (functionality testing) attack and try to break . running applications. They don’t analyze source code.

2.8.2 Secure deployment

Web applications are deployed when deemed functional within specifications, and when they’re secure. Upon deployment, ensuring their security requires two new elements:

- **Vulnerability Scanning.**

Applications must be scanned as a credentialed user and as a non-credentialed user to simulate a full range of tests. Ideally, scanning should occur as part of enterprise vulnerability management. In addition to testing systems and networks, web applications are tested to assess exposure to known vulnerabilities, other threats, and for compliance with the organization’s security policy. Applications at risk must be fixed to eliminate vulnerabilities. Remote scanning can be software-based or Software-as-a-Service (SaaS). In

cases where SaaS is used, internal scanning often utilizes a trusted Scanner Appliance. This ensures that scanning is thorough from the inside out.

- **Penetration Testing.**

Usually done by skilled experts, penetration testing is a deep, targeted effort to break a web application. In exploring vulnerabilities, a ‘pen test’ helps to measure and prioritize their impact.

2.8.3 Secure operation

The operational phase includes detection and reaction to actual vulnerabilities and potential exploits. Ideally, the security team leads this process with participation by programmers and other application experts as required. New elements for the operational phase include:

- **Web Application Firewall (WAF).**

This tool can help provide visibility into web application traffic. It also can block known attacks.

- **Activity Monitoring.**

An organization needs perpetual visibility on the operations and security of the web applications, databases powering their operation, and systems that host operations and provide connectivity. Automated tools can provide this visibility and instantly alert the security team when policy is violated.

2.9 Detectable vulnerabilities

Vulnerability	Description
Cross Site Scripting (XSS)	An application allows attackers to send malicious scripts by relaying the script from an otherwise trusted URL. Detection: web application security scanner Block/Fix: coding standards, web application firewall
Information Leakage	An application inappropriately discloses sensitive data, such as technical details of the application, environment, or user specific data. Detection: web application security scanner Block/Fix: coding standards, web application firewall
Session Management	An application inappropriately allows attackers to interject themselves as valid website users Detection: web application security scanner Block/Fix: coding standards.
Authentication & Authorization	An application does not properly ensure for unreachable and unpayable authentication, and/or authorized access to data and capabilities is not properly enforced on the server side of the application. This includes enforcement of proper encrypted communication of credentials, password standards enforcement, feature and data access ACL enforcement, etc. Detection: web application security scanner

	Block/Fix: server configuration, coding standards
Cross Site Request Forgery (CSRF)	CSRF inappropriately leverages a current authenticated browser session and uses unauthorized commands with credentials that the application trusts. Detection: web application security scanner Block/Fix: Coding practices, web application firewall
SQL Injection	An attacker inputs SQL database commands into a data entry field, and tricks the application into delivering user data, destroying user data, planting malicious data, extracts infrastructure information, drops tables, removes users or can dump a database. Detection: web application security scanner Block/Fix: coding standards, web application firewall
Web Server Version	Exploits applications, servers and databases through unpatched versions of server software with known security issues. Detection: web application security scanner Block/Fix: server configuration
Remote Code Execution	An application allows any arbitrary commands to execute on a vulnerable device. Successful exploitation could result in an attacker both inappropriately controlling/executing commands on the device, and being able to do so with the same privileges as the logged in user. Detection: web application security scanner Block/Fix: Coding practices, server configuration
Web Server Configuration	Attackers exploit misconfigured servers or access to server configuration files, enabling further, more sophisticated attacks. Detection: web application security scanner Block/Fix: server configuration
Unauthorized Directory Access	Access to directory listings should be restricted. Unsecured directories can be traversed, accessed and viewed by an attacker who may be able to access or view the contents of files. Detection: web application security scanner Block/Fix: server configuration

Table 1.4 - Common detectable application vulnerabilities

3 Conclusion

Shortage of skilled application security professionals remains a major issue across the global business landscape. This is likely to continue and lead to the growth in managed security services as companies look to bring in specialists to augment their overextended teams. Ultimately this will lead to better protection for web, cloud and mobile applications.

CHAPITRE 2

SQL INJECTION

1. Introduction

SQL injection is one of the most devastating vulnerabilities to impact a business, as it can lead to exposure of all of the sensitive information stored in an application's database, including handy information such as usernames, passwords, names, addresses, phone numbers, and credit card details.

SQL injection, it is the vulnerability that results when you give an attacker the ability to influence the Structured Query Language (SQL) queries that an application passes to a back-end database. By being able to influence what is passed to the database, the attacker can leverage the syntax and capabilities of SQL itself, as well as the power and flexibility of supporting database functionality and operating system functionality available to the database. SQL injection is not a vulnerability that exclusively affects Web applications; any code that accepts input from an untrusted source and then uses that input to form dynamic SQL statements could be vulnerable

2. The SQL Injection

SQL injection is an attack in which SQL code is injected or appended into application/ user input parameters that are later passed to a back-end SQL server for parsing and execution. Any procedure that constructs SQL statements could potentially be vulnerable, as the diverse nature of SQL and the methods available for constructing it provide a wealth of coding options. The primary form of SQL injection consists of direct insertion of code into parameters that are concatenated with SQL commands and executed. A less direct attack injects malicious code into strings that are destined for storage in a table or as metadata. When the stored strings are subsequently concatenated into a dynamic SQL command, the malicious code is executed. When a Web application fails to properly sanitize the parameters which are passed to dynamically created SQL statements (even when using parameterization techniques) it is possible for an attacker to alter the construction of back-end SQL statements. When an attacker is able to modify an SQL statement, the statement will execute with the same rights as the application user; when using the SQL server to execute commands that interact with the operating system, the process will run with the same permissions as the component that executed the command (e.g., database server, application server, or Web server), which is often highly privileged.

Example:

To illustrate this, we have a simple test website, the page **Sqli.php** contents of **Id's**, this is the first **id** from list of **Id's**:

- **<http://localhost/dvwa/vulnerabilities/sqli/?id=1>**

This time, however, we are going to attempt to inject own SQL commands by appending them to the input parameter val. We can do this by appending the string **' or 1=1 --** to the URL:

- **[http://localhost/dvwa/vulnerabilities/sqli/? id=' OR 1 =1 --](http://localhost/dvwa/vulnerabilities/sqli/?id=' OR 1 =1 --)**

This time, the SQL statement that the PHP script builds and executes will return all of **Id's** in the database, this is because we have altered the logic of the query, this happens because the appended statement results in the OR operand of the query always returning **true**, that is, **1** will always be equal to **1**, Here is the query that was built and executed:

```
SELECT *  
FROM Users  
WHERE User_id = " OR 1 = 1 --
```

The preceding simple example demonstrates how an attacker can manipulate a dynamically created SQL statement that is formed from input that has not been validated or encoded to perform actions that the developer of an application did not foresee or intend, the example, however, perhaps does not illustrate the effectiveness of such a vulnerability; after all, we only used the vector to view all of the **Id's** in the database.

What if the same application can be remotely administered using a content management system (CMS)? A CMS is a Web application that is used to create, edit, manage, and publish content to a Web site, without having to have an in-depth understanding of or ability to code in HTML. we can use the following URL to access the CMS application:

- **<http://localhost/btslab?username=admin&password=password>**

The CMS application requires that you supply a valid username and password before you can access its functionality. Accessing the preceding URL would result in the error “*Username/Password is wrong*”. Here is the code for the **login.php** script:

```
// connect to the database
$con=mysql_connect($db_server,$db_user,$db_password
mysql_select_db($db_name,$con);
// dynamically build the sql statement with the input &
execute the query against the database
$result=mysql_query("select * from users where
username='$username' and password='$password' ") or
die(mysql_error());;
// check to see how many rows were returned from the database
if(mysql_num_rows($result))
// if a row is returned then the credentials must be valid,
so // forward the user to the admin pages
$data=mysql_fetch_array($result);
    session_start();
    $_SESSION['isLoggedIn']=1;
    $_SESSION['userid']=$data["ID"];
    $_SESSION['username']=$username;
    $_SESSION['avatar']=$data['avatar'];
    header("Location: index.php");
// if a row is not returned then the credentials must be
invalid
$em="Username/Password is wrong";
```

The *login.php* script dynamically creates an SQL statement that will return a record set if a username and matching password are entered. The SQL statement that the PHP script builds and executes is illustrated more clearly in the following code snippet. The query will return the *userid* that corresponds to the user if the user and password values entered match a corresponding stored value in the *Users* table.

```
SELECT *
FROM Users
WHERE Username = 'Admin' AND Password = 'Password';
```

In the previous injection example, we used the exploitable vector to change the meaning of the SQL query to always return *true*. If we use the same technique with the CMS application, But try to input ' **OR 1 = 1** – as a user input, The URL would look like this:

- **localhost/btslab/login.php?username=' or 1=1 -- &password=&Login=Login**

Here is the query that was built and executed:

```
SELECT *
FROM Users
WHERE Username = " or 1=1 -- AND Password = ";
```

The meaning of " - - " is making all thing after in as a single line comment, we will normally be logged in as the first user in the Users table because the appended statement results in the OR operand of the query always returning **true**.

3. How SQL Injection Happens

SQL is the standard language for accessing Microsoft SQL Server, Oracle, MySQL, Sybase, and Informix (as well as other) database servers. Most Web applications need to interact with a database, and most Web application programming languages, such as ASP, C#, .NET, Java, and PHP, provide programmatic ways of connecting to a database and interacting with it. SQL injection vulnerabilities most commonly occur when the Web application developer does not ensure that values received from a Web form, cookie, input parameter, and so forth are validated before passing them to SQL queries that will be executed on a database server. If an attacker can control the input that is sent to an SQL query and manipulate that input so that the data is interpreted as code instead of as data, the attacker may be able to execute code on the back-end database.

3.1. Example of Wrongly Handled Escape Characters

SQL databases interpret the quote character (‘) as the boundary between code and data. It assumes that anything following a quote is code that it needs to run and anything encapsulated by a quote is data. Therefore, you can quickly tell whether a Web site is -vulnerable to SQL injection by simply typing a single quote in the URL or within a field in the Web page or application.

Here is the URL:

- **localhost/btslab/vulnerability/ForumPosts.php?id=1**

By adding the single-quote character (‘) at the end of the URL the result of executing will be like this:

You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near ''' at line 1

The reason for the error is that the single-quote character has been interpreted as a string delimiter. Syntactically, the SQL query executed at runtime is incorrect (it has one too many string delimiters), and therefore the database throws an exception. The SQL database sees the single-quote character as a special character (a string delimiter). The character is used in SQL injection attacks to “escape” the developer’s query so that the attacker can then construct his own queries and have them executed.

The single-quote character is not the only character that acts as an escape character; for instance, in Oracle, the blank space (), double pipe (||), comma (,), period (.), (*), and double-quote characters (“”) have special meanings

4. Finding SQL Injection

SQL injection can be present in any front-end application accepting data entry from a system or user, which is then used to access a database server. In a Web environment, the Web browser is a client acting as a front end requesting data from the user and sending it to the remote server which will create SQL queries using the submitted data.

4.1. Testing by Inference

There is one simple rule for identifying SQL injection vulnerabilities: Trigger anomalies by sending unexpected data. This rule implies that:

- Identify all the data entry on the Web application.
- knows what kind of request might trigger anomalies.
- detect anomalies in the response from the server

It’s as simple as that. need to see how Web browser sends requests to the Web server. Different applications behave in different ways, but the fundamentals should be the same, as they are all Web-based environments.

Once we identify all the data accepted by the application, need to modify it and analyze the response from the server, Sometimes the response will include an SQL error directly from the database and will make the injection very easy.

4.2. Identifying data entry

Web environments are an example of client/server architecture. Your browser (acting as a client) sends a request to the server and waits for a response. The server receives the request, generates a response, and sends it back to the client.

The two most relevant ones for the purpose of discovering SQL injection: the GET and POST methods.

4.3. GET Requests

GET is an HTTP method that requests to the server whatever information is indicated in the URL. This is the kind of method that is normally used when you click on a link. Usually, the Web browser creates the GET request, sends it to the Web server, and renders the result in the browser. Although it is transparent to the user, the GET request that is sent to the Web server looks like this:

```
GET /S/ HTTP/1.1
Host: localhost
User-Agent: Mozilla/5.0 (Windows NT 10.0; rv:44.0)
Gecko/20100101 Firefox/44.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0
.9,*/*;q=0.8
Accept-Language: fr,fr-FR;q=0.8,en-
US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
DNT: 1
Connection: close
If-Modified-Since: Thu, 24 Mar 2016 22:53:30 GMT
If-None-Match: "82a-52ed350ad1d17"
Cache-Control: max-age=0
```

This kind of request sends parameters within the URLs in the following format:

```
?parameter1=value1&parameter2=value2&parameter3...
```

For GET requests, we can manipulate the parameters by simply changing them in our browser's navigation toolbar. Alternatively, we can also use a proxy tool, which I'll explain shortly.

4.4. POST Requests

POST is an HTTP method used to send information to the Web server. The action the server performs is determined by the target URL. This is normally the method used when you fill in a form in your browser and click the Submit button. Although your browser does everything for you, this is an example of what is sent to the remote Web server:

```
POST /S/test3.php HTTP/1.1
Host: localhost
User-Agent: Mozilla/5.0 (Windows NT 10.0; rv:44.0)
Gecko/20100101 Firefox/44.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0
.9,*/*;q=0.8
Accept-Language: fr,fr-FR;q=0.8,en-
US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
DNT: 1
Referer: http://localhost/S/test3.php
Connection: close
Content-Type: application/x-www-form-urlencoded
Content-Length: 44

username=admin&password=password&Login=Login
```

The values sent to the Web server have the same format explained for the GET request, but are now located at the bottom of the request.

You may be wondering how you modify data if the browser is not allowing you to do so. There are a couple of ways to do this:

- Browser modification extensions
- Proxy servers

4.5. Browser modification extensions

Browser modification extensions are plug-ins that run on your browser and allow you to perform some additional functionality, For example: **HTTP Headers**, Interesting extension available for Firefox. You can use Tamper Data to view and modify headers and POST parameters in HTTP and HTTPS requests.

4.6. Proxy servers

A local proxy is a piece of software that sits between browser and the server, as shown in Figure, the software runs locally on computer, the figure shows a logical representation of a local proxy setup.



Figure 2.1 - Proxy server

The proxy intercepts the request to the server and permits you to modify it at will. To do this you need only two things:

- Installation of a proxy server on your computer
- Configuration of your browser to use your proxy server

Example for Proxy Server: Paros Proxy, WebScarab, and Burp Suite.

4.7. Web request and Web response

It is important to have a clear understanding of how your data entry influences an SQL query and what kind of response you could expect from the server, the Figure shows how the data sent from the browser is used in creating an SQL statement and how the results are returned back to the browser.

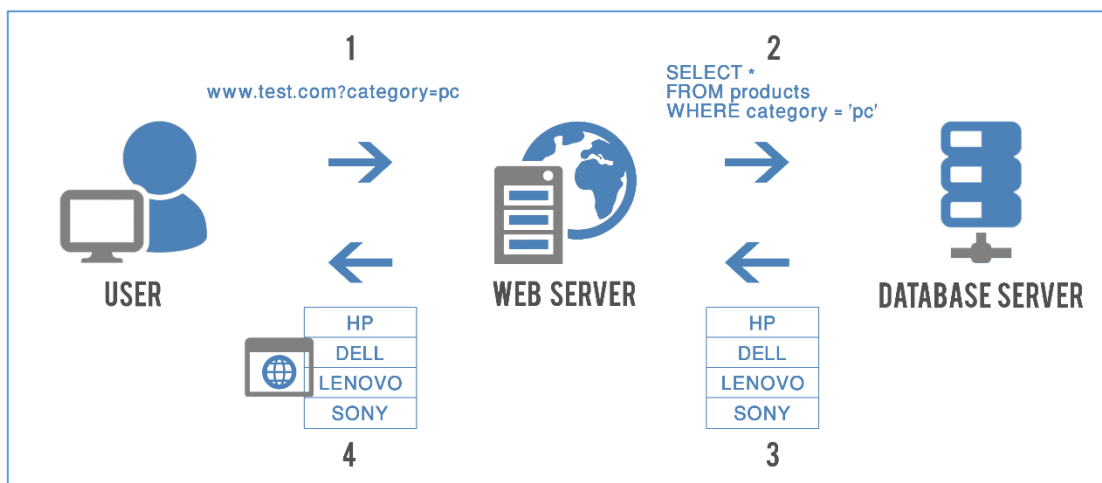


Figure 2.2-Web request and web response

1. The user sends a request to the Web server
2. The Web server retrieves user data, creates an SQL statement which contains the entry from the user, and then sends the query to the database server
3. The database server executes the SQL query and returns the results to the Web server. Note that the database server doesn't know about the logic of the application; it will just execute a query and return results.
4. The Web server dynamically creates an HTML page based on the database response.

4.8. Database Errors

The SQL injection happens at the database layer. Very important to know the different database errors that you may get from the Web server when testing for SQL injection vulnerabilities

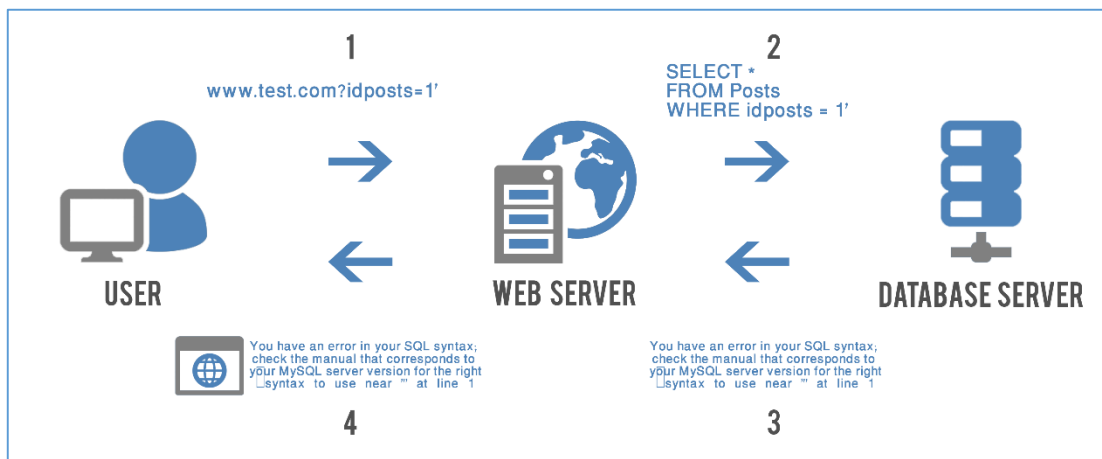


Figure 2.3- Database Errors

1. The user sends a request in an attempt to identify an SQL injection vulnerability. In this case, the user sends a value with a single quote appended to it.
2. The Web server retrieves user data and sends an SQL query to the database server.
3. The database server receives the malformed SQL query and returns an error to the Web server
4. The Web server receives the error from the database and sends an HTML response to the user

5. Differentiating Numbers and Strings

The databases have different data types. These types are represented in different ways.

- Number: represented without single quotes
- All the rest: represented with single quotes

5.1. Example:

```
SELECT * FROM Posts WHERE id_posts=1
SELECT * FROM company WHERE name = 'Microsoft'
```

When using a numeric value SQL statements don't use quotes, but alphanumeric values are enclosed between single quotes must need to take this into account when injecting SQL code into this types, it is possible to represent a numeric value between quotes, but the database will understand it as a string representation of a number; for example, '2'+2' = '22', not 4.

6. Inline SQL Injection

Inline injection happens when you inject some SQL code in such a way that all parts of the original query are executed, **Figure**, shows a representation of an inline SQL injection.

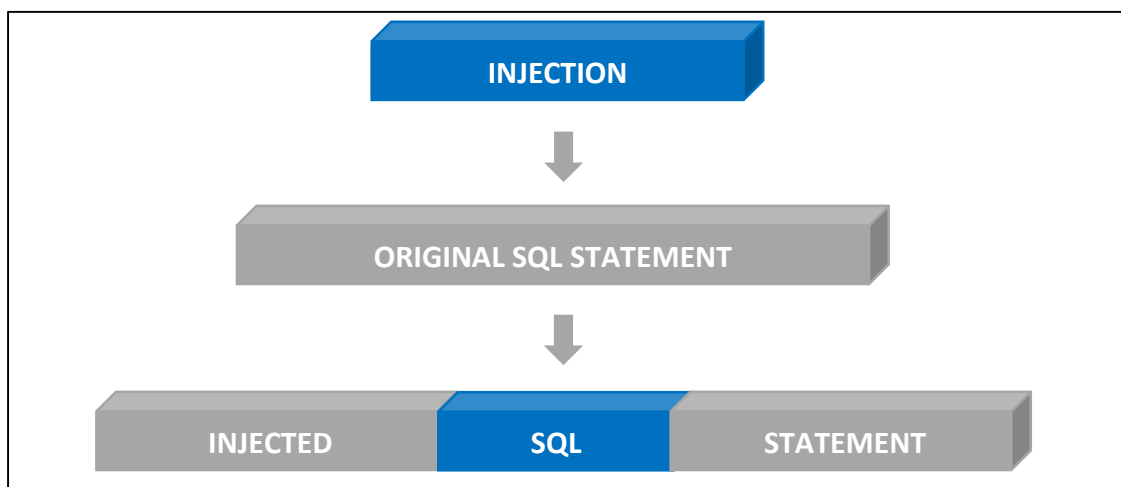


Figure 2.4-Inline SQL Injection

6.1. Injecting Strings Inline

Example of authentication form for accessing the administration part of its Web site. The authentication requires the user to enter a valid username and password. After sending a username and password, the application sends a query to the database to validate the user. The query has the following format:

```
SELECT *
FROM Administrators
WHERE Username = '[USER ENTRY]' AND Password = '[USER ENTRY]'
```

The Figure, shows the part of the SQL statement that you can manipulate.

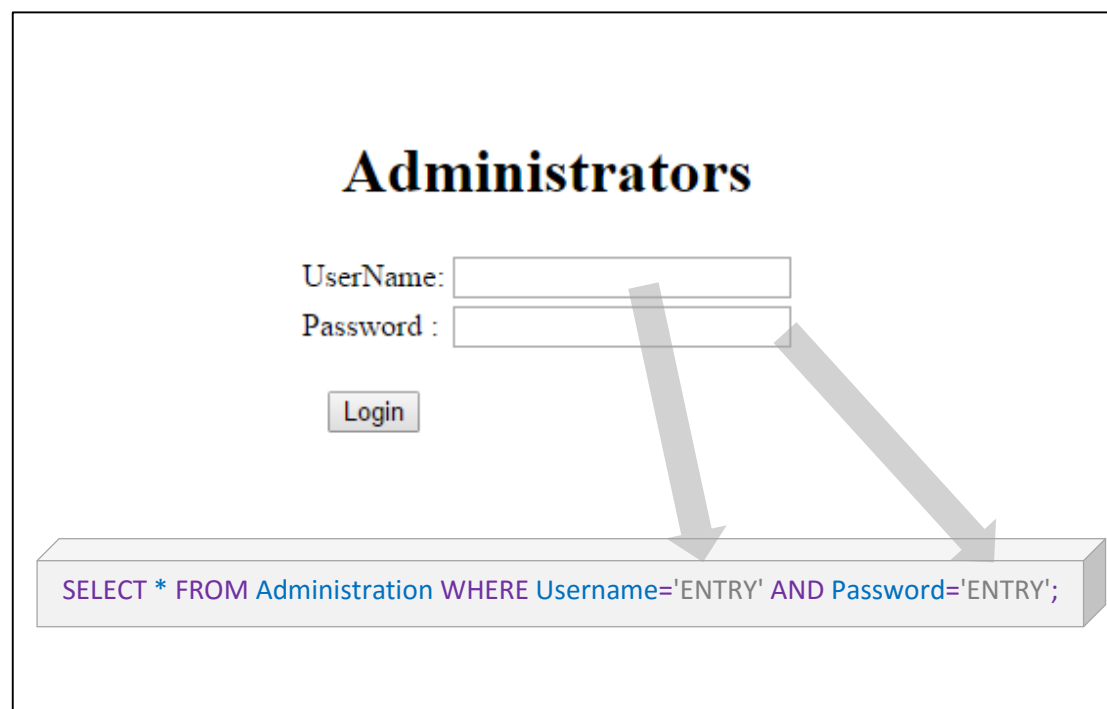


Figure 2.5-Injecting string inline

Entering a single quote in the *Username* field and clicking **Login** returns the following error:

```
You have an error in your SQL syntax; check the manual
that corresponds to your MySQL server version for the
right syntax to use near ''' and password='' at line 1
```

The error indicates that the form is vulnerable to SQL injection. The resultant SQL statement given the preceding input is as follows:

```
SELECT *
FROM Administrators
WHERE Username = "'" AND Password = "';
```

The syntax of the query is wrong due to the injected quote and the database throws an error, which the Web server sends back to the client, once we identify the vulnerability, also we can bypass the authentication control.

The **Table** provides you with a list of injection strings that you may need during the discovery and confirmation process of an inline injection in a string field.

Testing String	Expected Results
'	If successful, the database will return an error
1' or '1'=1	Always true condition, if successful, it returns every row in the table
1' and '1'=2	Always false condition. If successful, it returns no rows from the table

Table 2.1-List of injection string [4]

6.2. Injecting Numeric Values Inline

The URL have the following format:

- <http://localhost/S/test2.php?id=1>

When testing the *id* parameter sending just a single quote('), we get the following error:

```
You have an error in your SQL syntax; check the manual
that corresponds to your MySQL server version for the
right syntax to use near ''' at line 1
```

Based on the information retrieved, we can assert that the SQL code running on the server side should look like this:

```
SELECT *
FROM Posts
WHERE id= [USER ENTRY]
```

7. Ending SQL Injection

Injection-terminating an SQL statement is a technique whereby the attacker injects SQL code and successfully finalizes the statement by commenting the rest of the query.

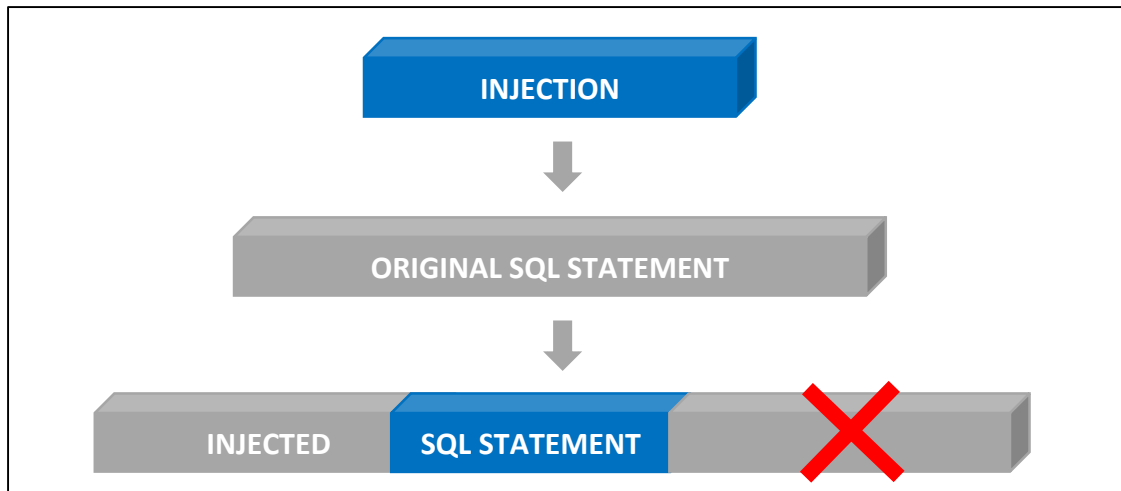


Figure 2.6 – Terminating SQL Injection

In The **Figure** the injected code terminates the SQL statement. Apart from terminating the statement need to comment out the rest of the query such that it is not executed.

7.1. Database Comment Syntax

Table shows the syntax for adding comments in SQL Server, Oracle, and MySQL databases.

Database	Comment	Observations
Microsoft SQL Server and Oracle	-- (double dash)	Used for single-line comments
	/* */	Used for multiline comments
MySQL	-- (double dash)	Used for single-line comments. It requires the second dash to be followed by a space or a control character such as tabulation, newline, etc.
	#	Used for single-line comments
	/* */	Used for multiline comments

Table 2.2 – Database Comment Syntax [4]

7.2. How Using Comments to terminate SQL statements

Represents the concept of terminating the SQL statement with the authentication.

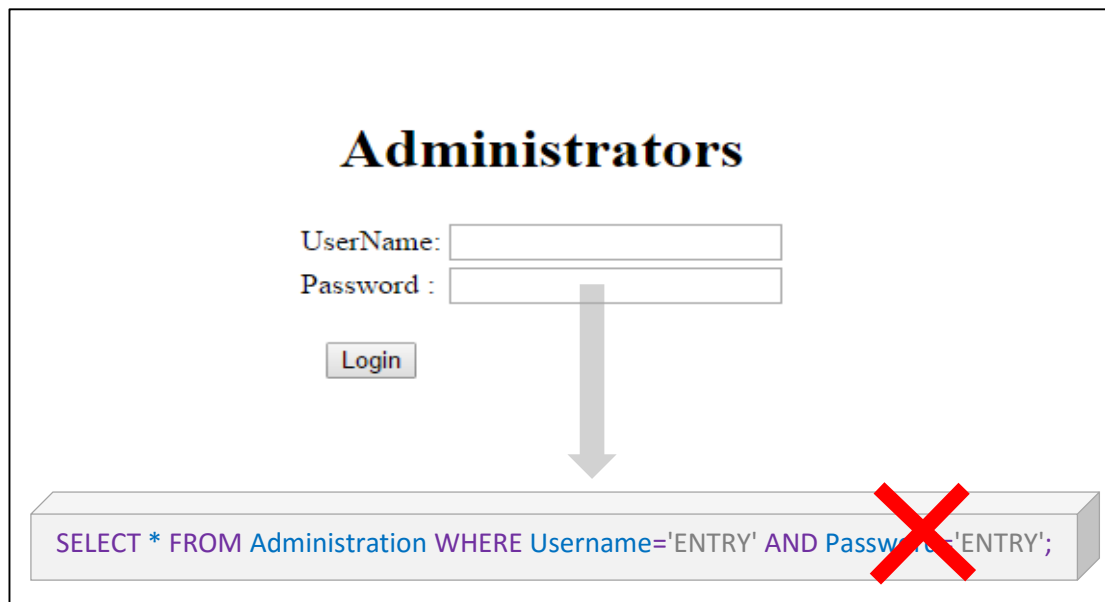


Figure 2.7- Using comments to terminate SQL statements

By using the code ' **or 1=1 --** ' into the username will create the following statement

```
SELECT *
FROM Administrators
WHERE Username = " or 1=1 -- ' AND password = ";
```

The result of this statement will be a successfully bypassing the authentication mechanism and it will ignore the part of the query after the comment, so we don't have to worry about the AND password=''.

The following **Table**, represent some of Signatures Using Database Comments

Testing String	Expected Results
Admin' --	Bypass authentication mechanism by returning the admin row set from the database
Admin' #	MySQL - Bypass authentication mechanism by returning the admin row set from the database
1 or 1=1 --	Return all rows injecting a numeric parameter

Table 2.3 – Some of signatures using comments [4]

8. Conclusion

SQL injection is an attack in which SQL code is inserted or appended into application/user input parameters that are later passed to a back-end SQL server for parsing and execution, without a sound understanding of the underlying database that they are interacting with or a thorough understanding and awareness of the potential security issues of the code that is being developed, application developers can often produce inherently insecure applications that are vulnerable to SQL injection.

There are three key aspects for finding SQL injection vulnerabilities: 1) identifying the data entry accepted by the application, 2) modifying the value of the entry including hazardous strings, and 3) detecting the anomalies returned by the server.

To confirm an SQL injection vulnerability and in prevision for later exploitation you need to craft a request that injects SQL code such that the application creates a syntactically correct SQL statement that is in turn executed by the database server without returning any error

CHAPTER 03

THE APPROACH PROPOSED

1. Introduction

SQL injection is an attack methodology that targets the data residing in a database through the firewall that shields it. The attack takes advantage of poor input validation in code and website administration. SQL Injection Attacks occur when an attacker is able to insert a series of SQL statements in to a „query“ by manipulating user input data in to a web-based application, attacker can take advantages of web application programming security flaws and pass unexpected malicious SQL statements, Query through a web application for execution by the back-end database. And attacker get full access of the backend database. In this way, SQL Injection Attack performed.

2. Definition of SQLIA

Most web applications today use a multi-tier design, usually with three tiers: a presentation, a processing and a data tier. The presentation tier is the HTTP web interface, the application tier implements the software functionality, and the data tier keeps data structured and answers to requests from the application tier. Meanwhile, large companies developing SQL-based database management systems rely heavily on hardware to ensure the desired performance. SQL injection is a type of attack which the attacker adds Structured Query Language code to input box of a web form to gain access or make changes to data. SQL injection vulnerability allows an attacker to flow commands directly to web applications underlying database and destroy functionality or confidentiality.

3. SQL Injection Attacks (SQLIA) Process

SQLIA is hacking technique which the attacker adds SQL statements through a web application's input field or hidden parameter to access to resources. Lack of input validation in web applications causes hacker to be successful. Basically SQL process structured in three phases:

- a) An attack sends the malicious HTTP request to the web application.
- b) Create the SQL Statements
- c) Submits the SQL statements to the back end database

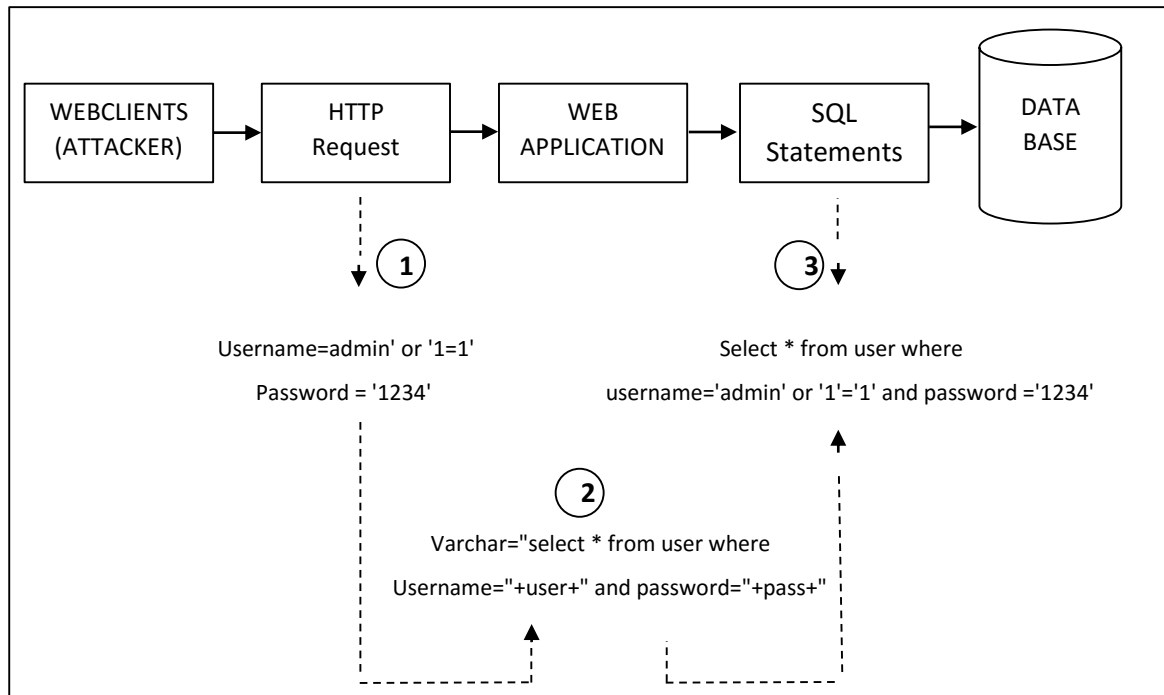


Figure 3.1 - Example for SQL injection attack data flow

4. Types of SQLIA

4.1. Tautologies

Attack Intent: Bypassing authentication, identifying inject able parameters, extracting data.

Description: This type of attack injects SQL tokens to the conditional query statement to be evaluated always true. This type of attack used to bypass authentication control and access to data by exploiting vulnerable input field which use WHERE clause. "SELECT * FROM employee WHERE userid = '118' and password ='aaa' OR '1'='1'" As the tautology statement (1=1) has been added to the query statement so it is always true.

4.2. Piggy-backed Query

Attack Intent: Extracting data, adding or modifying data, performing denial of service, executing remote commands.

Description: In the piggy-backed Query attacker tries to append additional queries to the original query string. On the successful attack the database receives and executes a query string that contains multiple distinct queries. In this method the first query is

original whereas the subsequent queries are injected. This attack is very dangerous; attacker can use it to inject virtually any type of SQL command. For example, `SELECT * FROM user WHERE id='admin' AND password='1234'; DROP TABLE user; --;` Here database treats above query string as two query separated by “;” and executes both. The second sub query is malicious query and it causes the database to drop the user table in the database.

4.3. Logically Incorrect

Attack Intent: Identifying inject able parameters, performing database finger-printing, extracting data.

Description: when a query is rejected, an error message is returned from the database including useful debugging information. This error messages help attacker to find vulnerable parameters in the application and consequently database of the application. In fact, attacker injects junk input or SQL tokens in query to produce syntax error, type mismatches, or logical errors by purpose. In this example attacker makes a type mismatch error by injecting the following text into the pin input field: 1) Original URL: `http://www.arch.polimi.it/eventi/?id_nav=8864` 2) SQL Injection: `http://www.arch.polimi.it/eventi/?id_nav=8864'` 3) Error message showed: `SELECT name FROM Employee WHERE id =8864\` From the message error we can find out name of table and fields: name; Employee; id. By the gained information attacker can organize more strict attacks.

4.4. Union query

Attack Intent: Bypassing Authentication, extracting data.

Description: By this technique, attackers join injected query to the safe query by the word UNION and then can get data about other tables from the application. Suppose for our examples that the query executed from the server is the following: `SELECT Name, Phone FROM Users WHERE Id=$id`. By injecting the following Id value: `$id=1 UNION ALL SELECT credit Card Number,1 FROM Credit sysTable` We will have the following query: `SELECT Name, Phone FROM Users WHERE Id=1 UNION ALL SELECT creditcardNumber,1 FROM Credit Sys Table` which will join the result of the original query with all the credit card users.

4.5. Stored Procedure

Attack Intent: Performing privilege escalation, performing denial of service, executing remote commands.

Description: In this technique, attacker focuses on the stored procedures which are present in the database system. Stored procedures run directly by the database engine. Stored procedure is nothing but a code and it can be vulnerable as program code. For authorized/unauthorized user the stored procedure returns true/false. As an SQLIA, intruder input “; SHUTDOWN; --” for username or password. Then the stored procedure generates the following query: For example, SELECT accounts FROM users WHERE login= '1111' AND pass='1234 '; SHUTDOWN; --; This type of attack works as piggyback attack. The first original query is executed and consequently the second query which is illegitimate is executed and causes database shut down.

4.6. Inference

Attack Intent: Identifying injectable parameters, extracting data, determining database schema.

Description: By this type of attack, intruders change the behavior of a database or application. There are two well-known attack techniques that are based on inference: blind injection and timing attacks.

Blind Injection: Sometimes developers hide the error details which help attackers to compromise the database. In this situation attacker face to a generic page provided by developer, instead of an error message. So the SQLIA would be more difficult but not impossible. An attacker can still steal data by asking a series of True/False questions through SQL statements. Consider two possible injections into the login field: SELECT accounts FROM users WHERE login=“doe” and 1=0 -- AND pass= AND pin=0
SELECT accounts FROM users WHERE login=“doe” and 1=1 -- AND pass= AND pin=0 If the application is secured, both queries would be unsuccessful, because of input validation. But if there is no input validation, the attacker can try the chance. First the attacker submit the first query and receives an error message because of "1=0". So the attacker does not understand the error is for input validation or for logical error in query. Then the attacker submits the second query which always true. If there is no login error message, then the attacker finds the login field vulnerable to injection.

Timing Attacks: A timing attack lets an attacker gather information from a database by observing timing delays in the database's responses. This technique by using if-then statement cause the SQL engine to execute a long running query or a time delay statement depending on the logic injected. This attack is similar to blind injection and attacker can then measure the time the page takes to load to determine if the injected statement is true. This technique uses an if-then statement for injecting queries. WAITFOR is a keyword along the branches, which causes the database to delay its response by a specified time. For example, declare @ varchar (8000) select @s = db_name () if (ascii (substring (@s, 1, 1)) & (power (2, 0))) > 0 wait for delay '0:0:5' Database will pause for five seconds if the first bit of the first byte of the name of the current database is 1. Then code is then injected to generate a delay in response time when the condition is true. Also, attacker can ask a series of other questions about this character. As these examples show, the information is extracted from the database using a vulnerable parameter.

4.7. Alternate Encodings

Attack Intent: Evading detection

Description: In this technique, attackers modify the injection query by using alternate encoding, such as hexadecimal, ASCII, and Unicode. Because by this way they can escape from developer's filter which scan input queries for special known "bad character". For example, attacker use char (44) instead of single quote that is a bad character. This technique with join to other attack techniques could be strong, because it can target different layers in the application so developers need to be familiar to all of them to provide an effective defensive coding to prevent the alternate encoding attacks. By this technique, different attacks could be hidden in alternate encodings successfully. In the following example the pin field is injected with this string: "0; exec (0x73587574 64 5f77 6e)," and the result query is: SELECT accounts FROM users WHERE login=" AND pin=0; exec (char(0x73687574646f776e)) This example use the char () function and ASCII hexadecimal encoding. The char () function takes hexadecimal encoding of character(s) and returns the actual character(s). The stream of numbers in the second part of the injection is the ASCII hexadecimal encoding of the attack string. This encoded string is translated into the shutdown command by database when it is executed.

5. The proposed approach

A web site is an ensemble of linked web pages architecture to face specifications of customer's needs. So, secure a web site is the result of securing its web pages. A web page is a contents structured with tags. Any contents had to be enveloped with specific tags. This vision leads us to a deep remark: "the change of a web page implies the change of its contents as well as its structure, which is, its tags". This observation is taken as the principle of our proposed detection approach of SQL injections. Another observation is the fact that: "The contents need to be tagged, and the tags enclose contents". This second statement reveals a strong informational connection between contents and its tags. This leads, to the fact, that any change appearing on the contents is sensitive by the tags and may be reflected by a change of the tags to represent the new contents arising or cutting as a result from the corresponding change. After this reflection, it seems natural to choose to articulate our approach upon the tags rather than the contents. Tags are more restricted in number and meaning because they are specifying by the norm HTML, than the content which obeys to the willing of the developer. Two information concerning the tags should be seeking: the tags as syntax and semantic and their number. The formers are fixed by the HTML language and they will not well have affected by the change of a web page, but, the number is directly affected by any change in the web page. So, the variation of the number of tags between the original web page and the resulted one from SQL injection is our index of SQL injection detection. The detection method as simple as it appears will reveals a great performance over the scanners offered by academicians and professionals.

6. The method – (Method Detector)

Our method as explained earlier is based on the variation of the number of the tags between the original page and the resulted one after injection attacks.

The method is summarized concisely in the following table:

	Tag / Original	Result
M1	<	In Safe
M2	0	Safe
M3	>	In Safe

Table 3.1 - The method proposed

M1: Represents the case where an error is returned by the SGBD due to the syntactic malformation of the queries.

M2: Reflects the state of secure page where no change in the page is engaged, this implies no

variation in the number of tags between the original page and the resulted one.

M3: Stands for the situation where the injection is a success and some information have been

extracted and new contents are formulated through new tags.

6.1. Example:

In order to explain well our method we present its operational semantic through an example.

6.2. Queries

The attack queries for scanning are chosen from the famous ones used in the academics and professional, also hacking area. The four following queries seem to be strong enough to build our scanner upon them. All scanners use a little number of queries; many of them chose only one.

The choice of the queries is motivated by the goal of stimulating an SGBD error elsewhere extract information from the database.

Req 1	' or 1=1 --	Req 3	' or 1'=1#
Req 2	'or'1=1' --	Req 4	' or 0=0 #

Table 3.2 - Queries

6.3. Links

Pages of the test-site are built according to the type of the injections that could be used.

We are interested in **six very common types of SQL injections**. The following table shows the test pages matched with the corresponding injection type.

N°	URL	Type
1	dvwa/vulnerabilities/sqli/	Injection Based

Table 3.3 - Links

6.4. Results of the test

WS	SITE			
	Low Security			
	Req1	Req2	Req3	Req4
1	M3	M3	M1	M3

Table 3.4 - Result of the test

6.5. Analyze of the results

In this part we use a test in a vulnerable page Regarding the results, we see that our approach can detect the vulnerability of the page used, by using **Req1**(as an attack vector) we specify the result as **M3** that's mean the number of tag is increases, as a result the page is **Insafe** also the same thing by using **Req2**, in adding to that by using **Req3** we see as a result The number of Tag is decreasing that's mean the page is **Insafe**.

Regarding the results, we can say that our approach can detect the vulnerability of pages that's prove the fact of this approach, we will see more analyses in the Chapter 4.

7. Related work

7.1. SWADDLER

Examines the internal state of a web application. It workings based on both single and multiple variables and shows an impressive way against complex attacks to web applications. First the approach describes the normal values for the application's state variables in critical points of the application's components. Then, during the detection phase, it monitors the application's execution to identify abnormal states.[5]

7.2. SQL Prevent

Is consists of an HTTP request interceptor. The original data flow is modified when SQL Prevent is deployed into a web server. The HTTP requests are saved into the current thread-local storage. Then, SQL interceptor intercepts the SQL statements that are made by web application and pass them to the SQLIA detector module. Consequently, HTTP request from thread-local storage is fetched and examined to

determine whether it contains an SQLIA. The malicious SQL statement would be prevented to be sent to database, if it is suspicious to SQLIA.

7.3. SQL Lrand

Boyd, Keromytis, proposed SQLrand which uses instruction set randomization of SQL statement to check SQL injection attack. It uses a proxy to an append key to SQL keyword. A de randomizing proxy then converts the randomized query to proper SQL queries for the database. The key is not known to the attacker, so the code injected by attacker is treated as undefined keywords and expressions which cause runtime exceptions and the query is not sent to database. The disadvantage of this system is its complex configuration and the security of the key. If the key is exposed, attacker can formulate queries for successful attack.

7.4. SAFELI

Propositions a Static Analysis Framework in order to detect SQL Injection Vulnerabilities, SAFELI framework aims at identifying the SQL Injection attacks during the compile-time, this static analysis tool has two main advantages. Firstly, it does a White-box Static Analysis and secondly, it uses a Hybrid-Constraint Solver. For the White-box Static Analysis, the proposed approach considers the byte-code and deals mainly with strings. For the Hybrid-Constraint Solver, the method implements an efficient string analysis tool which is able to deal with Boolean, integer and string variables.

7.5. SQLIPA

(SQL Injection Protector for Authentication) prototype was developed in order to test the framework. The user name and password hash values are created and calculated at runtime for the first time the particular user account is created [6].

7.6. SQLCheck

Su and Wassermann, implement their algorithm with SQLCHECK on a real time environment. It checks whether the input queries conform to the expected ones defined by the programmer. A secret key is used for the user input delimitation. The analysis of SQLCHECK shows no false positives or false negatives. Also, the overhead

runtime rate is very low and can be implemented directly in many other Web applications using different languages.

7.7. IDS (Intrusion Detection Systems)

IDS system is based on a machine knowledge technique that is trained using a set of typical application queries. The method builds models of the typical queries and then monitors the application at runtime to identify queries that do not match the model. In their evaluation, Valeur and colleagues have shown that their system is able to detect attacks with a high rate of success. However, the fundamental limitation of learning based techniques is that they can provide no guarantees about their detection abilities because their success is dependent on the quality of the training set used. A poor training set would cause the learning technique to generate a large number of false positive and negatives.

7.8. Detection based on removing SQL query attribute values

It's an approach based on static and dynamic analysis. This method eliminates the attribute values of SQL queries at runtime (dynamic method) and compares them with the SQL queries analyzed in advance (static method) to detect the SQL injection. When run the application each dynamical generated query is compared or performs XOR operation with fixed query if it results zero then that particular query allowed to the database and if it not results to zero then that query reported as abnormal query stop sending to database.

7.9. DIWeDa

DIWeDa, a prototype which acts at the session level rather than the SQL statement or transaction stage, to detect the intrusions in Web applications. The proposed framework is efficient and could identify SQL injections and business logic violations too.

7.10. Context Sensitive String Evaluation (CSSE)

The basic idea behind this approach is to find out the root cause of SQLIA, the root cause is the origin of the data (information about the data, termed as metadata) i.e., user-provided or developer-provided. Thus, any data provided by the user is marked as

untrusted and data provided by the applications are termed as trusted. The untrusted metadata are used for syntactic analysis based on 'Context Sensitive String Evaluation (CSSE)'. Injection may also occur due to programming flaws during developments. CSSE is basically based on syntactical analysis, which first distinguishes string constants (for e.g., select *from users where login='\$login_name') and numerical constants (e.g., select * from users where pin=\$pin). It then removes all unsafe characters (un-escaped quotes) in alphanumeric identifiers and non-numeric characters in numeric identifiers. This operation is performed before sending the query to the database server. Following issues are there in this approach (i) Initialization of the unsafe characters is dependent on the web programmer, and (ii) Removal of unsafe characters restricts the application functionality.

7.11. CANDID

It is a Dynamic Candidate Evaluations method for automatic prevention of SQL Injection attacks. This framework dynamically extracts the query structures from every SQL query location which are intended by the developer (programmer). Hence, it solves the issue of manually modifying the application to create the prepared statements.

7.12. Automated Approaches

Besides using manual approaches, MeiJunjin, also highlights the use of automated approaches. The author notes that the two main schemes are: Static analysis FindBugs and Web vulnerability scanning. Static analysis FindBugs approach detects bugs on SQLIAs, gives warning when an SQL query is made of variable. However, for the Web vulnerability scanning, it uses software agents to crawl, scans Web applications, and detects the vulnerabilities by observing their behavior to the attacks.

7.13. AMNESIA

Is a model-based technique that combines static analysis and runtime monitoring, in its static phase, AMNESIA uses static analysis to build models of the different types of queries an application can legally generate at each point of access to the database. In its dynamic phase, AMNESIA intercepts all queries before they are sent to the database and checks each query against the statically built models.

8. Comparison of SQL Injection detection techniques with respect to attack types

SQL Injection detection techniques														
	Method Detector													
Attack Types	AMNESIA	●	●	●	●	×	●	●	●	×	●	●	●	●
	Automated Approaches	●	●	●	●	×	●	●	●	×	●	●	×	×
	CANDID	○	○	○	○	○	○	○	○	○	○	○	○	○
	CSSE	●	●	●	●	×	●	●	●	×	●	●	×	×
	DIWeDa	×	×	×	×	×	×	×	×	×	×	×	×	×
	Detection based on removing SQL query	●	●	●	●	●	●	●	●	●	●	●	●	●
	IDS	○	○	○	○	○	○	○	○	○	○	○	○	○
	SQLcheck	●	●	●	●	●	×	●	●	×	●	●	●	●
	SQLIPA	●	×	×	×	×	×	×	×	×	×	×	×	×
SAFELI	×	●	●	●	●	●	●	●	●	●	●	●	●	
SQLrand	●	●	●	●	●	×	●	●	×	●	●	●	●	
SQL Prevent	●	●	●	●	●	●	●	●	●	●	●	●	●	
SWADDLER	○	○	○	○	○	○	○	○	○	○	○	○	○	

Table 3.5 - Comparison of SQL Injection detection techniques[6]

- (●) - Technique that can stop all attacks of that type
- (○) - Technique that can stop the attack only partially
- (×) - Technique that is not able to stop attacks of that type

9. Percentage of SQL Injection Detection Techniques

S.No.	Attack types	Technique that can stop all attacks of that type (•)	Technique that can stop the attack only partially(o)	Technique that is not able to stop attacks of that type (×)
1	Tautologies	59	31	9
2	Piggy-backed	54	31	13
3	Logically / Incorrect	54	31	13
4	Union	54	31	13
5	Stored Procedure	22	31	45
6	Inference	59	31	9
7	Alternate Encodings	45	31	22

Table 3.6 - Percentage of SQL Injection Detection Techniques

10. Conclusion

It is obvious from above description that SQL injection attacks are one of the largest classes of security problems, in this chapter, we represent our proposed approach and we also assessed detecting SQL injection attacks among current SQL Injection detection techniques. To perform our approach, we first identified the various types of SQLIAs known to date. Then we represent our approach also the method with simple example, after that we represent all related work such as "SWADDLER, SQL Prevent ... etc." and we compared these techniques in terms of their ability to stop SQLIA.

Regarding the results, we see that our proposed approach has the power to detect most of SQL Injection attacks also we see that our approach is better than some technique proposed and we see that some current techniques ability should be improved for stopping SQLI attacks, finally we can say that our approach has a fact to be one of the most detection technique of SQL Injection attacks.

CHAPTER 04

ANALYSIS, RESULTS AND EXPERIMENTAL

1. Introduction

The mathematics of the proposed approach and its design is so simple that thorough testing is required to validate its detection prediction SQL injection attacks, whether on forms or URLs. A series of attacks has been specified to achieve this objective.

Three of the most popular scanners have been used to conduct a comparative study of our approach. NETsparker and OWASP ZAP scanner.

2. Platform

2.1. Visual Studio

Microsoft Visual Studio is an integrated development environment (IDE) from Microsoft. It is used to develop computer programs for Microsoft Windows, as well as web sites, web applications and web services. Visual Studio uses Microsoft software development platforms such as Windows API, Windows Forms, Windows Presentation Foundation, Windows Store and Microsoft Silverlight.

Visual Studio supports different programming languages and allows the code editor and debugger to support (to varying degrees) nearly any programming language, provided a language-specific service exists. Built-in languages include C, C++ and C++/CLI (via Visual C++), VB.NET (via Visual Basic .NET), C# (via Visual C#), and F# (as of Visual Studio 2010).

2.2. C#

Pronounced "C sharp", Is a programming language that is designed for building a variety of applications that run on the .NET Framework, C# is simple, powerful, type-safe, and object oriented.

2.3. XAMPP

XAMPP is a free and open source cross-platform web server solution stack package developed by Apache, consisting mainly of the Apache HTTP Server, MariaDB database, and interpreters for scripts written in the PHP and Perl programming languages. XAMPP stands for Cross-Platform (X), Apache (A), MariaDB (M), PHP (P) and Perl (P).

2.4. PHP

PHP is a server-side scripting language designed for web development but also used as a general-purpose programming language. Originally created by Rasmus Lerdorf in 1994, the PHP reference implementation is now produced by The PHP Group. PHP originally stood for Personal Home Page, but it now stands for the recursive backronym PHP: Hypertext Preprocessor

2.5. MySQL

MySQL is an open-source relational database management system (RDBMS), in July 2013, it was the world's second most widely used RDBMS, and the most widely used open-source client–server model RDBMS.

3. Experimentation

We proceed to some experimentation to test our method. The tests are of two steps; we valid our method on some vulnerable web pages to observe it detection behavior. Secondly, we study the performance of the proposed algorithm compared to some famous scanners.

The results are collected and well represented, a series of deep analysis will lead to understand the method.

4. The Sites of experimentation

4.1.The BTS Lab

4.1.1. Presentation

BTS Lab is an open source vulnerable web application. It can be used to learn about many different types of web application vulnerabilities.

Currently, the app allows you to learn the following vulnerability types Including SQL Injection.

4.1.2. Architecture

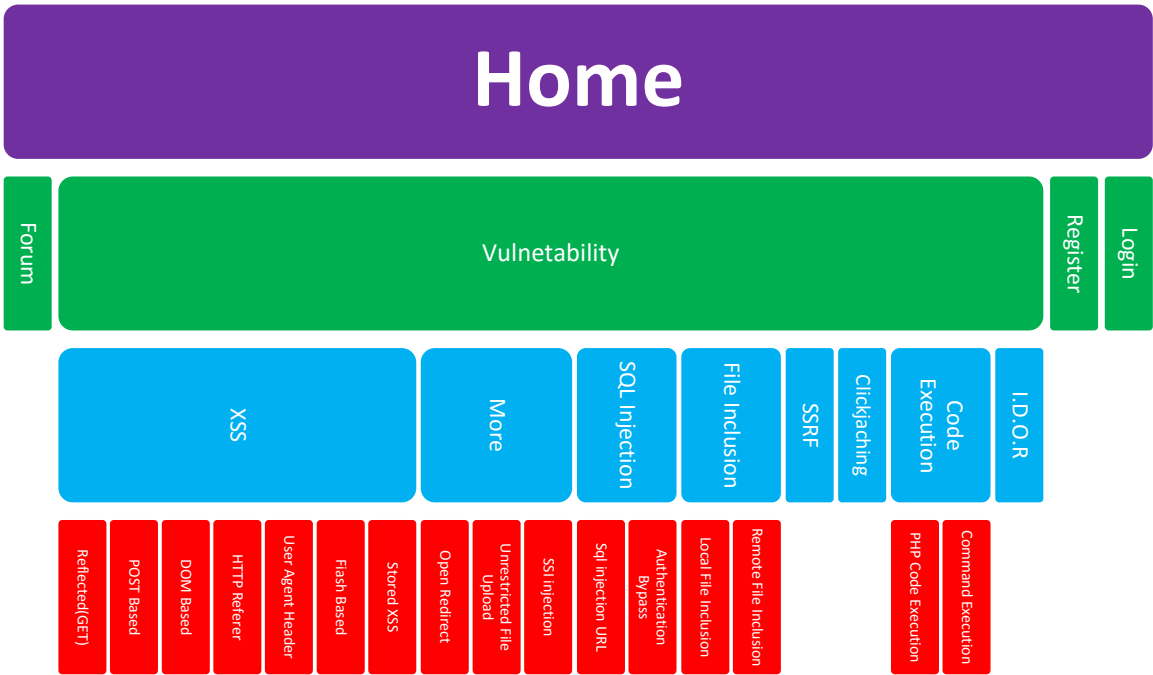


Figure 4.1- BTS Lab architecture

4.2.TEST Site

4.2.1. Presentation

Test-Lab Site is an open source vulnerable web application, is a simple site developed to test the vulnerability of SQL Injection with deferent types.

4.4.2. Architecture

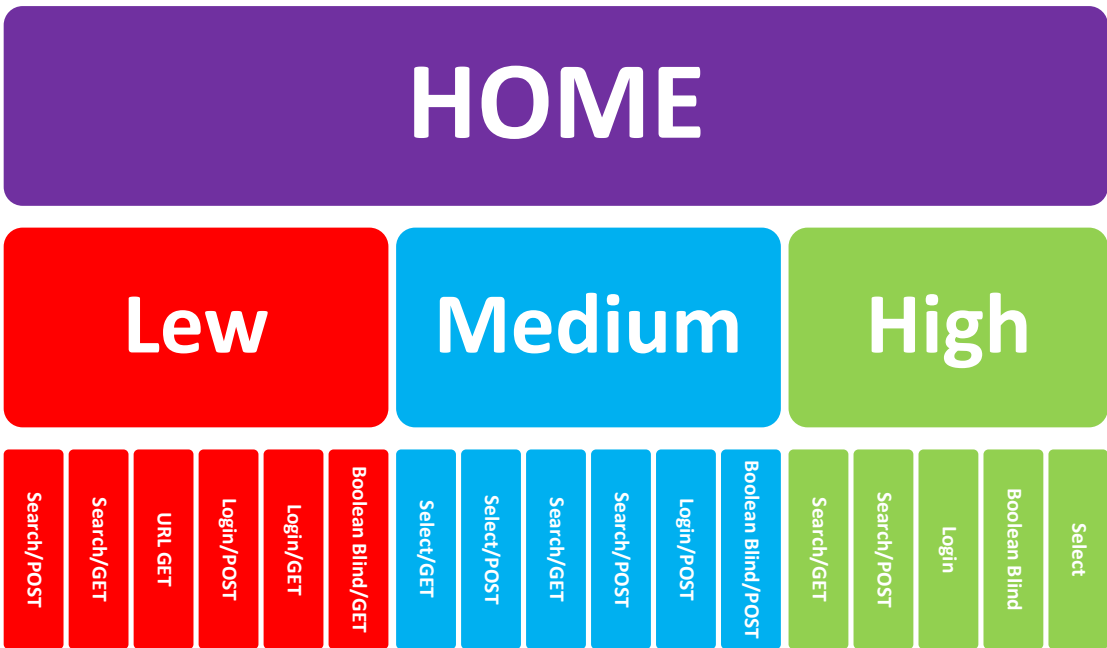


Figure 4.8 – TEST Site Architecture

4.3. Buggy Web Application (bWAPP)

4.3.1. Presentation

bWAPP, is a free and open source deliberately insecure web application, it helps security enthusiasts, developers and students to discover and to prevent web vulnerabilities, bWAPP prepares one to conduct successful penetration testing and ethical hacking projects.

4.3.2. Architecture

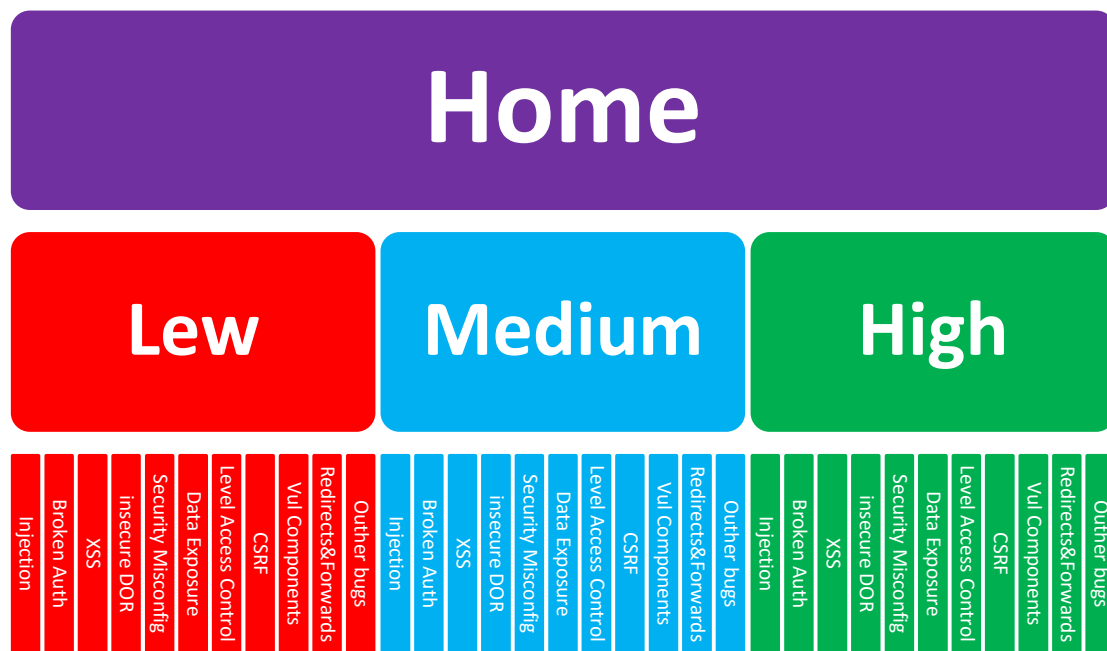


Figure 4.3 – bWAPP architecture

5. The scanners used in the experiment

5.1. Netsparker

Netsparker identified an SQL injection, which occurs when data input by a user is interpreted as an SQL command rather than as normal data by the backend database. This is an extremely common vulnerability and its successful exploitation can have critical implications. **Netsparker** confirmed the vulnerability by executing a test SQL query on the backend database.

5.2. OWASP ZAP 2.4.2

OWASP ZAP (short for Zed Attack Proxy) is an open-source web application security scanner. It is intended to be used by both those new to application security as well as professional penetration testers.

6. Experiment of the Method

6.1 Attacks vectors

The SQL queries chosen to construct the sql injection attack vector in order to test the web sites vulnerability are selected in the spirit of stimuli the Data Base Management System (DBMS) to response with generating error messages notification, which is a sign of vulnerability of the current web page, rather trying to extract data from the database because our aim is to test vulnerability, not hacking information.

Qrs 1	' or 1=1 --	Qrs 11	admin') or ('1'=1
Qrs 2	'or'1=1' --	Qrs 12	admin') or ('1'=1'#
Qrs 3	' or 1'=1#	Qrs 13	'
Qrs 4	' or 0=0 #	Qrs 14	' or 1'=1#
Qrs 5	admin' --	Qrs 15	') or ('a'=a
Qrs 6	admin'/*	Qrs 16	) or '1'=1--
Qrs 7	admin' or '1'=1'--	Qrs 17	' or a'=a#
Qrs 8	admin' or '1'=1'/*	Qrs 18	') or ('x'=x
Qrs 9	admin' or 1=1	Qrs 19	1') and '1'=1--
Qrs 10	admin' or 1=1/*	Qrs 20	' or 1'=2#

Table 4.1 - Queries

6.2.bWAPP

6.2.1. Links (bWAPP)

N°	URL	Type
1	http://localhost/bwapp/sqli_1.php	Injection(GET/Search)
2	http://localhost/bwapp/sqli_2.php	Injection(GET/Select)
3	http://localhost/bwapp/sqli_6.php	Injection(POST/Search)
4	http://localhost/bwapp/sqli_13.php	Injection(POST/Select)
5	http://localhost/bwapp/sqli_3.php	Injection(Login/Hero)
6	http://localhost/bwapp/sqli_7.php	Injection(Stored/Blog)
7	http://localhost/bwapp/sqli_16.php	Bypass Authentication
8	bwapp/sqli_4.php	Blind/Boolean-based

Table 4.2 – Links

6.2.2. Test method with queries

WS	bWAPP						M1%	M3%	M1%T	M3%T
	Low									
	URL 1	URL 2	URL 3	URL 4	URL 5	URL 6				
Qrs 1	M3	M1	M3	M1	M3	M1	50%	50%	62.50%	37.50%
Qrs 2	M3	M1	M3	M1	M3	M1	50%	50%		
Qrs 3	M1	M1	M1	M1	M1	M1	100%	0%		
Qrs 4	M3	M1	M3	M1	M3	M1	50%	50%		

Table 4.3 – Test with queries

- We see that the proposed algorithm has succeed to detect all vulnerabilities
- The detection is based on error messages notified by the DBMS by about 62.50% detections.

6.2.3 bWAPP – SQL Injection – Bypass Authentication

6.2.3.1. Test method with queries

WS	bWAPP	M1%Total
	Low	
	URL 7	
Qrs 5	M1	100%
Qrs 6	M1	
Qrs 7	M1	
Qrs 8	M1	
Qrs 9	M1	
Qrs 10	M1	
Qrs 11	M1	
Qrs 12	M1	

Table 4.4 – Test with queries

- Detection: full success
- DBMS error messages notification: success of 100%

6.2.4 bWAPP – SQL Injection – Blind – Boolean- Based

6.2.4.1 Test method with queries

WS	bWAPP	M1%Total
	Low	
	URL 8	
Qrs 13	M1	100%
Qrs 14	M1	
Qrs 15	M1	
Qrs 16	M1	
Qrs 17	M1	
Qrs 18	M1	
Qrs 19	M1	
Qrs 20	M1	

Table 4.5 – Test with queries

- The success of the method detects all vulnerabilities
- Successes of SQL injection attack vector to stimuli the DBMS error message notification.

6.3.BTS Lab Site

6.3.1. Links (BTS Lab)

N°	URL	Type
URL 1	btslab/vulnerability/ForumPosts.php?id=1	SQLi (Based)
URL 2	btslab/login.php	Bypass Authentication

Table 4.6 – Links

6.3.2 Test method with queries

WS	bWAPP		M1%	M3%	M1%Total	M3%Total
	Low					
	URL 1	URL 2				
Qrs 1	M1	M3	50%	50%	62.50%	37.50%
Qrs 2	M1	M3	50%	50%		
Qrs 3	M1	M1	100%	0%		
Qrs 4	M1	M3	50%	50%		

Table 4.7 – Test with queries

- Detection: full success
- DBMS error messages notification: success of 62.50 %

7. Comparison the method with different scanner

7.1. List of the URL (TEST Site)

In this second part of the test we proceed to the study of the performance of the proposed scanner named “T-Scan” compared to some sophisticated know scanners.

The chosen test sites are designed with three security levels: low, medium and high security. Each security level tests will have imposed an order on the scanners based on their detection performance. This way of doing allows showing gradually the strongest of each scanner performance.

The sites are chosen in such a manner that they include the major methods of handling data in web sites, as shown in the table below.

Site	Levels Security	URL-N	Type of page	URL
	Low Security	1	Search/POST	Test1.php
		2	Search/GET	Test5.php
		3	URL/GET	Test2.php
		4	Login/POST	Test3.php
		5	Login/GET	Test10.php
		6	Boolean Blind/GET	Test4_1.php
	Medium Security	1	Select/GET	Test7.php
		2	Select/POST	Test6.php
		3	Search/GET	Test11.php
		4	Search/POST	Test12.php
		5	Login/POST	Test13.php
		6	Boolean Blind/POST	Test4_2.php
	High Security	1	Search/GET	Test8.php
		2	Search/POST	Test9.php
		3	Select	Test16.php
		4	Login	Test14.php
		5	Boolean Blind	Test15.php

Table 4.8 – List of URL

7.2. Low Security

Low Security						
Scanner/URL	URL 1	URL 2	URL 3	URL 4	URL 5	URL 6
OWASP ZAP	1	1	1	1	1	1
Net Sparker	1	1	1	1	1	1
Detector Method	1	1	1	1	1	1

Table 4.9 - Low Security

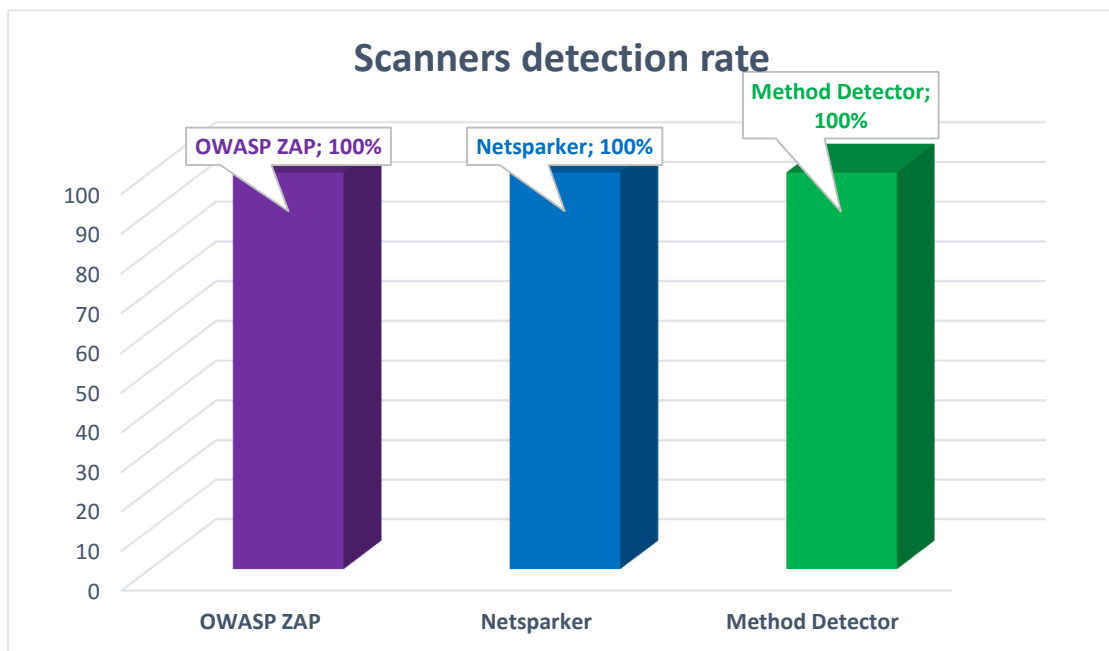


Figure 4.4 – Different detection Low security

In this graph, the scanners are ordered from the right to the left, the strongest scanner first. And we will observe if this order persists with the growth of the security level or not.

7.3. Medium Security

Medium Security						
Scanner/URL	URL 1	URL 2	URL 3	URL 4	URL 5	URL 6
Net Sparker	1	1	0	0	1	1
OWASP ZAP	1	1	0	0	1	1
Detector Method	1	1	1	1	1	1

Table 4.10 – Medium Security

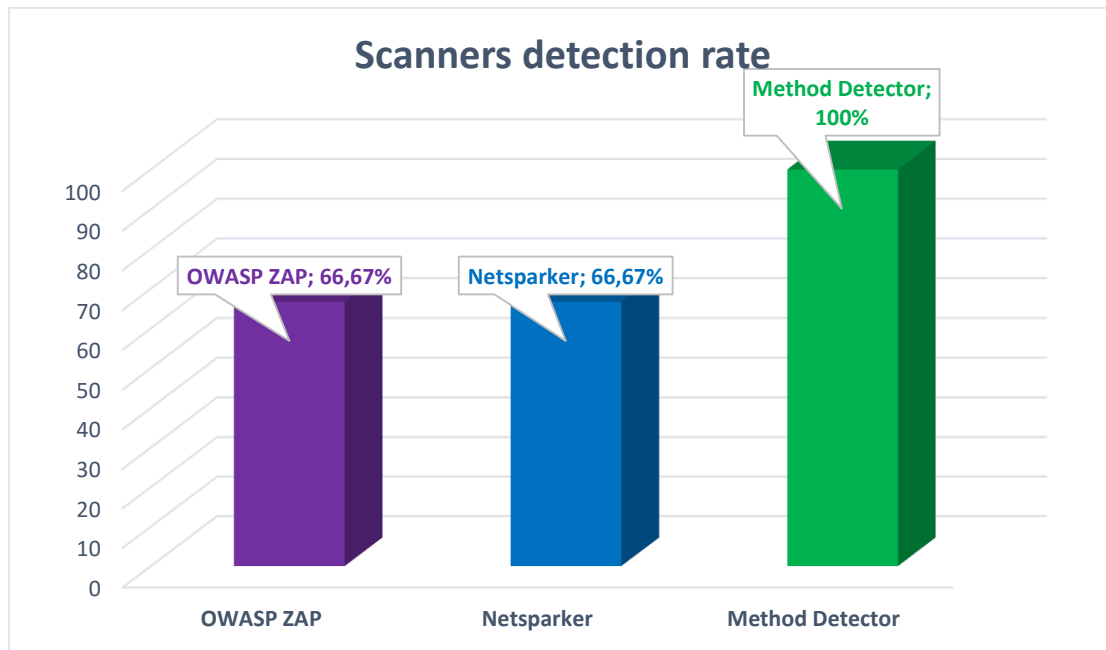


Figure 4.5 - Different detection Medium security

The medium security level induces changes in detection performance of the used scanners, but the proposed scanner: “T-Scan”, resists to it, and still provides high detection performance. The order of scanners according to their detection performance is preserved. T-Scan has the higher performance.

7.4. High Security

High Security					
Scanner/URL	URL 1	URL 2	URL 3	URL 4	URL 5
Net Sparker	0	0	0	0	0
OWASP ZAP	0	0	0	0	0
Detector Method	0	0	0	0	0

Table 4.11 - High Security

All scanners detect no vulnerabilities due to the simple fact, they don't exit. Because high security level use sophisticated security functions.

7.5 The result of the vulnerability (12 page vulnerable)

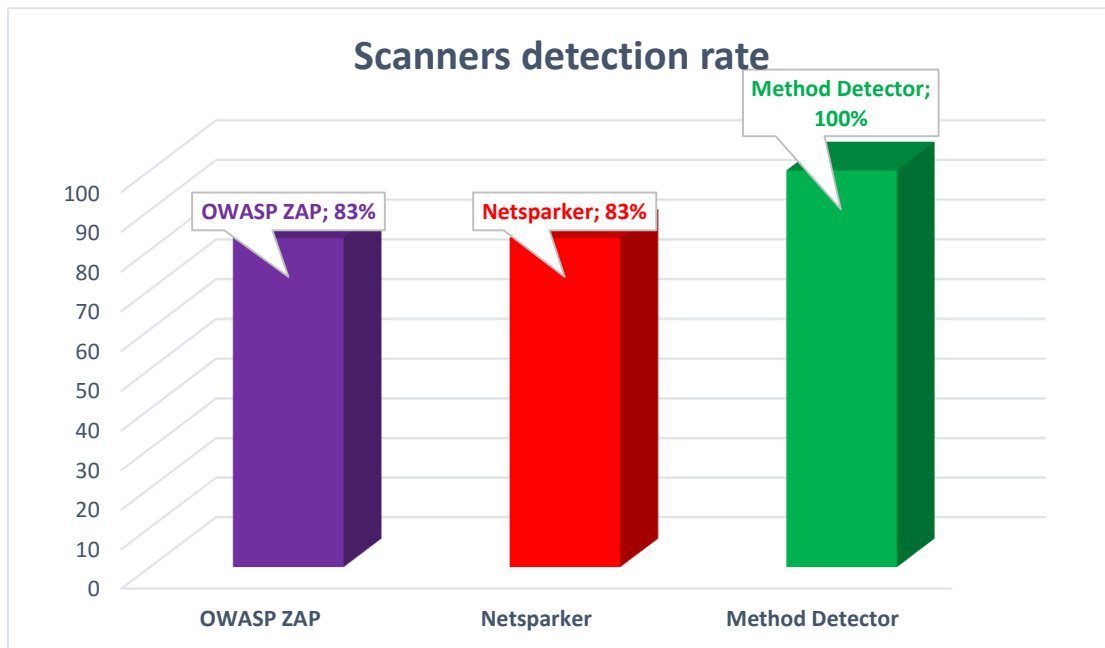


Figure 4.6 – Scanners detection rate

The global comparison between scanners reveals a high vulnerability detection of our scanner “T-Scan” compared to the used scanners, Netsparker come second, after it the OWASP ZAP, finally SQL-D.

7.6 The result of False positive/the vulnerability (12 page vulnerable)

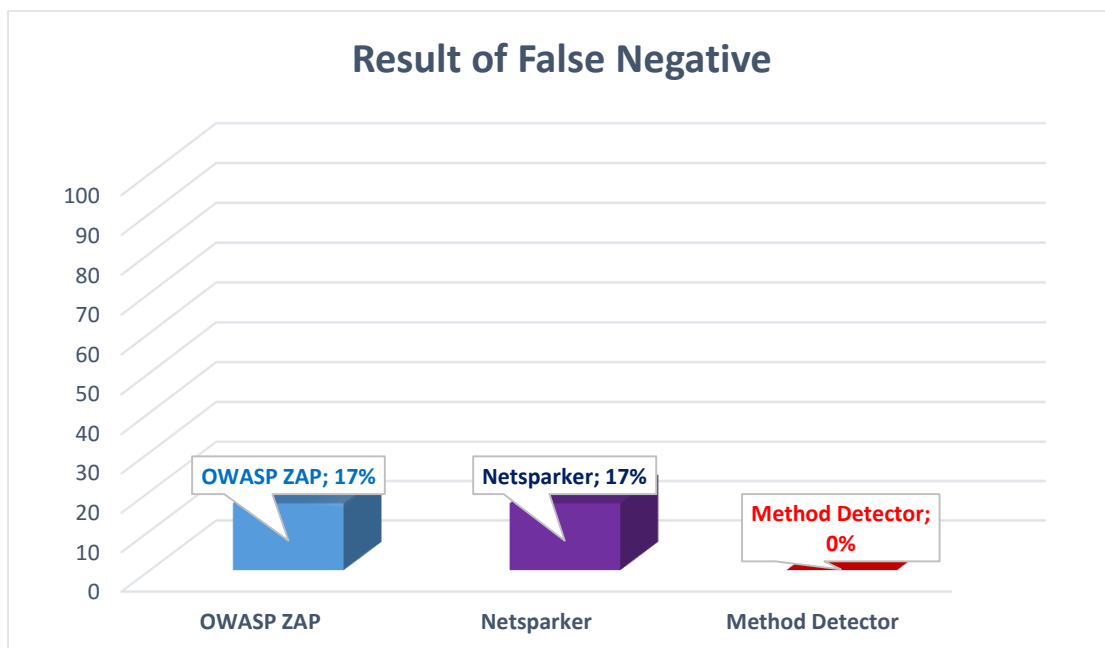


Figure 4.7 - Result of False Negative Detection

By seeing the graph of false negative we can see the fact that our scanner compared to other scanners.

8. Conclusion

The experimentations done to analyze the proposed method, have used test sites which were rich of methods of handling input and output data, also with a variety of security functions which they were chosen in such a manner to get three security levels, in order to get a nice idea about the detection performance of our method. The comparison uses three strong scanners, and the results confirm the strength of our scanner.

CONCLUSION GENERAL

CONCLUSION GENERAL

SQL injection is a serious threat with deep dangerous consequences on the integrity and security of web applications. In our work, we proposed a method designed upon the structure of the web page cached by its number of tags. The detection is based on the statement that “the behavior of any SQL attack changes the number of tags from the original page to the injected one”.

This statement reveals being a good one, because the detection performance is highly appreciated over the well-known scanners used by the community of researchers and professionals. a series of experiments have shown this fact.

REFERENCES

Books :

- [1] Hanqing Wu and Liz Zhao, **Web Security: A WhiteHat Perspective**, by Auerbach Publications, April 6, 2015

- [2] John Wiley & Sons, Ltd, **Web Application Security for Dummies**, by Qualys, West Sussex PO19 8SQ England, 2011.

- [3] Michael E. Whitman, Herbert J. Mattord, **Principles of Information Security**, Kennesaw State University, 20 Channel Center Boston, MA 02210 USA, 2012.

- [4] Justin clarke, **SQL Injection Attacks and Defense**, Rachel Roumelioti , July 2, 2012.

Journals and Survey:

- [5] Pushkar Y.Jane , M.S.Chaudhari, SQLIA: Detection And Prevention Techniques: A Survey, iosrjournals, 2, 2013, pp. 56-60.
- [6] V. Nithya,R.Regan , J.vijayaraghavan, A Survey on SQL Injection attacks, their Detection and Prevention Techniques, iosrjournals, Volume 2, April, 2013 pp, 886-905

Guide:

- [7] **The Ten Most Critical Web Application Security Risks**, OWASP, 2013
- [8] **A Guide to Building Secure Web Applications and Web Services**, OWASP.
- [9] **Web Application Security a Beginner's guide**, Bryan Sullivan Vincent Liu,

Rapport:

- [10] **Application Vulnerability Trends Report**, CENZIC, 2014

Website:

- [11] OWASP, www.owasp.org.

BIBLIOGRAPHY

[12] OWASP Zap Scanner, 2015

[13] Netsparker Web Application Security Scanner, Version 3.5.

[14] BTS Pentesting Lab Site.

[15] bWAPP Site.