Sound and Precise Analysis of Web Applications For Injection Vulnerabilities

Hiren Chafekar, Kasireddi.Himabindu, Rahul Agarwal
Ajeenkya D.Y Patil University, Pune, Maharastra, India

Abstract:
Nowadays web applications are the major targets for the hackers. It is an application program which is stored remotely on a server. It can be utilized using web browser over the internet .There are different type of techniques such as SQL injection, cross-site scripting,etc. through which web applications vulnerabilities can be exploited. In this paper the authors have tried to provide mitigation to reduce the adverse effect of exploitation of web vulnerabilities.

Keywords:
Web application vulnerabilities, Mitigations, Techniques used to exploit, Security, Verification

Literature Review:
Web applications are popular targets of security attacks. Many researchers have focused on SQL and XSS injection attacks and their prevention. Faulty application code is exploited and is used to execute maliciously queries. Similarly, much research on analyzing other types of injection attacks along with the prevention methods is being done. Following, advancement in technology, loopholes is being found more as compared to the usefulness of the web application. Hence, end users need to be aware of such types of attacks and the preventive ways or mitigations to be followed.

I. Introduction

A Web application is a program that is utilized through a browser interface and is virtually stored on a remote server and delivered over the Internet. Web services are the best examples of web apps. It can be designed according to the need of the company or the users. The application can be used by end users and organisations for numerous reasons. Some of the common usage includes e-commerce sites, document converter, webmail, etc. Web applications have diverse range of uses along with many potential benefits. Some common benefits of Web applications are:

- Same version can be accessed by multiple users.
- Installation is not a prerequisite.
- It can be used on any device irrespective of hardware.
- Any browsers can be used to access the web application.

However, many new types of vulnerabilities are being found every day. These vulnerabilities are loopholes or weakness in the system which can used to exploit the user's or company's systems They don't validate form inputs, misconfigured web servers and due to flaws in application design. These vulnerabilities are not the same as other common types of vulnerabilities, such as network or asset. They arise because web applications need to interact with multiple users across multiple networks, and that level of accessibility is easily taken advantage of by hackers.

II. Problem Statement

To identify the different types of vulnerabilities found in web application which might lead to successful cyber-attack and financial loss.

III. Types of Injection Attacks

Major five types of injection attacks along with their mitigations have been explained in detail:

3.1 Code Injection

Remote Code Execution or RCE is another name for Code Injection. Its objective is to execute a malicious code on the host machine remotely through injection attack. These attacks vary from Command Injection attacks. Attacker's capability is dependent on the limits of the server-side interpreter. The attacker can move from Code Injection attack to Command Injection attack. Such type of attacks succeeds when an application evaluates the code without validating it. Following is an example in PHP with Code Injection bug:

```
/**
* Get the code from a GET input
* Example - http://example.com/?code=phpinfo(); */
$code = $_GET['code']; /**
* Unsafely evaluate the code
* $myvar = "varname";
* $x = %_GET['arg'];
* eval("\$myvar = \$x;");
*/
eval("\$code;");
```

The following command can be used by anyone to execute the code. Hence, the following info page would be displayed:

* *http://example.com/?code=phpinfo();*

The attacker can attempt to install malware, spyware or also execute OS commands on the host machine post successful code injection attack. .

* */index.php?arg=1; system('id')*

3.1.1 Mitigation

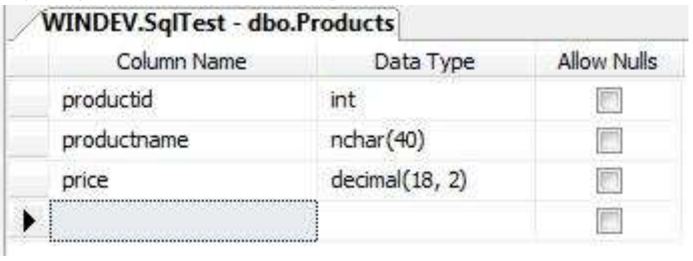
Utilisation of updated Application Program Interface (API) must be used. For instance: In Java, rather than using Runtime.exec() to issue a 'mail' command, Java API located at javax.mail.* can be used. Developer must scan for malicious characters in case there is no API available. A positive security model should be implemented for better efficiency. The legal characters are easy to define as compared to the illegal characters.

3.2 SQL Injection

SQL Injection (SQLi) is a type of injection attack for standardized query language. Execution of malicious SQL statements is performed in such attacks. A database server can be controlled by such malicious SQL queries. Such vulnerability can easily affect any website or web application as they rely on usage of SQL database, eg: MySQL, Oracle, SQL Server, etc. To initiate with the attack vulnerable user inputs are needed. To make an SQL Injection attack, the attacker creates a malicious payload. The malicious payload consists of input content which traps the user to feed vulnerable inputs. The malicious SQL commands are executed successfully post sending the content by the attacker in the database. Such successful attacks can create very serious repercussion.

- Credentials can be easily found within the database using SQL Injections. The attacker can masquerade himself/herself as the user. An administrator's system can also be compromised
- SQL allows the option for selecting and displaying output from the database. Hence, complete access is granted to the attacker.
- Alteration or addition of data can be done in SQL. For example, in a medical report, an attacker could use SQL Injection to tamper the data.
- The user can use SQL to delete records from a database, even drop tables. Even though backups are stored during failures, it takes significant amount of time to restore the system In addition to, the backup does not fully cover the most recent data.
- Accidently or incidentally the operating system can be accessed in some database servers. Here, SQL Injection as the initial vector is used and later, attack is done in the internal network behind a firewall.

In this example, the attacker can run any code they want on the database server. If an account which has full control of the database is injected then it becomes very easy for the attacker to drop tables, delete records, hence bringing down the website.


WINDEV.SqlTest - dbo.Products

Column Name	Data Type	Allow Nulls
productid	int	☐
productname	nchar(40)	☐
price	decimal(18, 2)	☐
		☐

Fig. 1: A sample database named "Products"

Here we create a simple database named "Products" consisting of three columns. The first column holds the product id, the second holds the product name, and the third holds the product's price. For our normal query we'll attempt to find every product containing widget in the name. The SQL to do this in the same pattern we've show would look like this:
*string sql = "SELECT * FROM Products WHERE productname LIKE '%" + searchterm + "%'";*

A typical website to access this would look like: *http://www.example.com/product?search=widget.*
Here the web page will simply cycle through each returned record and display it on the screen. In this case we see our widget product.

productid	productname	price
1	Widget	100

Let's change our query to *http://www.example.com/product?search=widget' OR 1=1;*-- and execute the same query. Now every record in the table can be seen.

productid	Productname	price
1	Widget	100
2	Thingy	50
3	Boxy	125

We have tricked the interpreter into running SQL code of our choice.
*SELECT * FROM Products WHERE productname LIKE '%widget' OR 1=1;—%'*
The result will be two statements:

1. *SELECT * FROM Products WHERE productname LIKE '%widget' OR 1=1;*
2. *—%'*

By adding the 1=1, which is always true, the where clause will be true for every row in the table and the resulting query will return every row. The -- at the start of the second statement turns the rest of the SQL statement into a comment which prevents the error message we'd otherwise see.

3.2.1 Mitigation

The SQL database or interpreter must be prevented from accepting untrusted SQL inputs. The inputs which do not belong to the system must be declared as untrusted and should not be allowed to operate. Foreign inputs should be considered as untrusted. Even data from other partner systems must be considered untrusted as you have no way to guarantee the other system does not suffer from security problems that would allow insertion of arbitrary data then passed on to your application.

3.3 CRLF Injection

When the user wants to view any website, a browser needs to sends a request to a web server. Response headers along with the website content is sent back in response by the web server. A carriage return and a line feed are used to separate these headers. Together, they are called as CRLF. The CRLF provides the idea to the web server that a new HTTP header has begun and another one has ended. It also gives knowledge on new line in a file or a text block. These are part of standard HTTP message, so it can be used by any type of web server. The attacker needs to insert both carriage return and linefeed characters in such attacks to gain access in the server. The web application or the user assumes that an object has been terminated and another one has started. Although CRLF sequences are not malicious, they can be used for malicious intent, for HTTP response splitting etc. The CRLF can be used as following:

- Adding a fake HTTP response header to fake the new response. Content-Length: 0. This causes the web browser to assume the ongoing response as a terminated response and begin parsing a new response.
- HTTP/1.1 200 OK. New response has begun here.
- Content-Type: text/html. It is utilized for parsing the content.
- Content-Length: 25. Only 25 bytes can be parsed in the web browser.
- XSS: <script>alert(1)</script>. Page content is added here but this content has exactly 25 bytes.
- Because of the Content-Length header, the web browser ignores the original content that comes from the web server.

http://www.example.com/somepage.php?page=%0d%0aContent-Length:%200%0d%0a%0d%0aHTTP/1.1%20200%20OK%0d%0aContent-Type:%20text/html%0d%0aContent-Length:%2025%0d%0a%0d%0a%3Cscript%3Ealert(1)%3C/script%3E

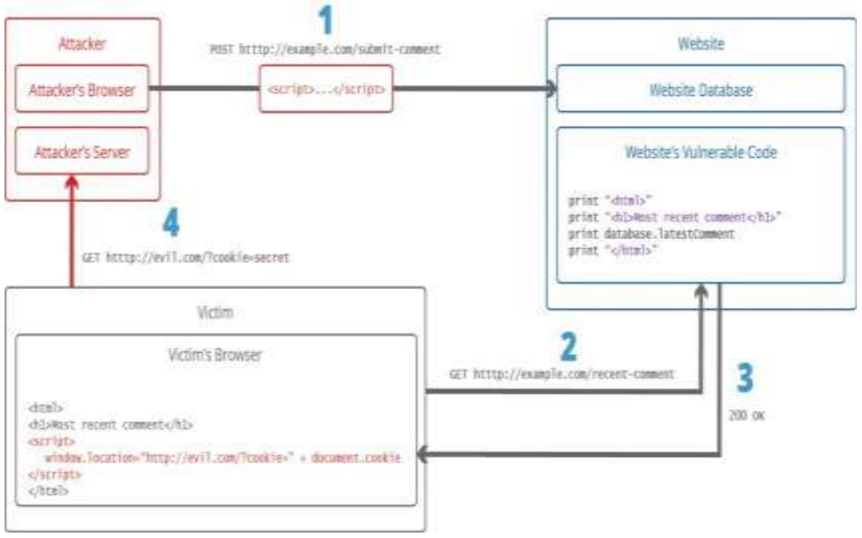
3.3.1 Mitigation

Input from users must not be directly inserted within the response headers. Alternatively, encoding the CRLF special characters can also be an effective step. Lastly, updating the programming language to latest version which does not entertain CRLF injection inside functions that sets the HTTP headers

3.4 Cross-Site Scripting

Cross-Site Scripting (XSS) attacks are one kind of injection attack. The main objective of such attacks is to inject malicious scripts into trusted and benign websites. In this attack, web application is used as a main tool to send the malicious code. This code typically, is embedded in the form of a browser side script and is sent to the end user. Such attacks are quite extensive nowadays. Whenever, the web application uses user's input without validating or encoding it, there are high chances of risk that a malicious script can be embedded in the input. XSS allows the attacker to gain access to victim's machine using such attacks. As the end user is clueless about the malicious script, he/she blindly trusts because it comes from a trusted source. Hence, the script is executed without any problems and can easily gain access to all the cookies, session tokens or other sensitive information. The HTML page can also be changed entirely by editing its content using these scripts. XSS attacks comprises of two stages:

1. The attacker needs to find a doorway or loophole into the webpage that the victim visits. The attacker can then inject the malicious code, i.e., payload.
2. Following the injection, the victim is needed or is fooled to visit the webpage. Social engineering or phishing are different which can also be used to target specific victims for spreading the URL.


Attacker
Attacker's Browser
Attacker's Server
1
POST http://example.com/submit-comment
<script>...</script>
Website
Website Database
Website's Vulnerable Code
4
GET http://evil.com/?cookie=secret
Victim
Victim's Browser
2
GET http://example.com/recent-comment
3
200 OK

```
print "<html>"
print "<h1>Most recent comment</h1>"
print database.latestComment
print "</html>"
```

```
<html>
<h1>Most recent comment</h1>
<script>
  window.location="http://evil.com/?cookie=" + document.cookie
</script>
</html>
```

Fig. 2: A pictorial example explaining the XSS attack

print "<h1>Most recent comment</h1>"
print database.latestComment
print "</html>"

Here, a script has been added to gain the up-to-date comment from the database. This script has been added in the Webpage. The web page presumes the comments to be safe and consisting only of text. Unaware of the malicious scripts, it accepts the comment and the scripts are run in the background. Hence, the webpage or website is vulnerable to XSS as anyone could submit a comment which consists of payload. For example:

<script>Script_running();</script>

The web server provides the following HTML code to users that visit this web page:

<html>
<h1>i recent comment</h1>
<script>Script_running();</script>
</html>

The script executes when the page is loaded in the host machine. Most often, the victim does not realize it and is unable to prevent such an attack.

3.4.1 Mitigation

Insert untrusted data in allowed locations only. HTML Escape must be used before inserting any kind of untrusted data. Attributing the Escape before it any untrusted data is inserted into HTML Common Attributes. Always use HTTPOnly cookie flag. URL Escape must be sanitized, i.e. cleaned with a particular library. Content Security Policy needs to be implemented and updated regularly.

3.5 OS Command Injection

OS command injection allows the attacker to perform operating system commands remotely. It is also called shell injection. Here, the host machine is using the application that is running on the server and the server has all the data of the application which can be compromised. Mostly, the attacker misuses the OS command injection to compromise the hosting infrastructure and exploits and destroys the trust relationship within the organization.

The user performes DNS lookup.It is the first variant of OS command injection.

```
use CGI qw(:standard);
$name = param('name');
$nslookup = "/path/to/nslookup";
print header;
if (open($fh, "$nslookup $name|")) {
while (<$fh>) {
print escapeHTML($_);
print "<bri>\n";
}
close($fh);
}
```

Suppose an attacker provides a domain name such as: *cwe.mitre.org%20%3B%20/bin/ls%20-l*

The ";" character is decoded by the "%3B" sequence and the %20 decodes to a space. The open() statement processes a string: /path/to/nslookup cwe.mitre.org ; /bin/ls –l. Execution of the command: "/bin/ls -l" is done and the attacker receives a list of all the files in the program's working directory. The code did not sanitize the user inputs.

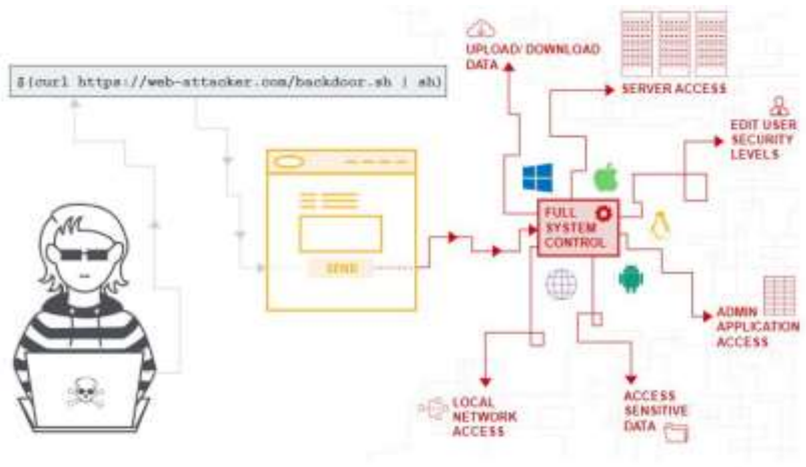


Fig. 3 : A pictorial example explaining the OS Injection attack

3.5.1 Mitigation

A developer must use existing APIs for their relevant programming languages. For example, In Java, the developer available Java API should be utilized which is located at javax.mail.*. Regex or a whitelist of accepted values should be used in case there is no such available API exists, the developer should validate the input using Regex or a whitelist of accepted values. This is much better than using Runtime.exec() to issue a 'mail' command.

```
Class Win32 extends OS {
        public void email (String subject, String body) throws Exception {
                String cmd = "cmd.exe /c start \"\"\"" + formatMailto(subject, body) + "\"";
                Runtime.getRuntime().exec(cmd);
        }
 }
```

Executing only static strings is the idel way to remain safe.

IV. Conclusion

As there quite number of vulnerabilities in the web applications through which the systems can be compromised, appropriate mitigation is strictly required. The web applications need to be patched after every vulnerability is discovered. The owners of the web applications can conduct an open source testing among tester to find the bugs in their applications and to help them increase their knowledge and experience. The organisation can select fromwhite box testing, grey box testing or black box testing to find the bugs depending whether the application has been deployed or not

Acknowledgment

We are really very thankful and we would also like to display our sincere recognition for our Head of the Department Dr.Shabnam Sharma and our supervisor Prof. Ratan Singh for the guidance, support and continuous encouragement in making this research possible. Their valuable guidance from initial to final level has helped us in completion of this research paper. Our sincere thanks to all the faculties who helped us to complete the research paper and gave their valuable advice and made our work easy.

VI. Refrences:

[1] https://www.acunetix.com/blog/articles/injection-attacks/

[2] https://searchsoftwarequality.techtarget.com/definition/Web-application-Web-app

[3] https://www.rapid7.com/fundamentals/web-application-vulnerabilities/

[4] `https:
//wikisites.cityu.edu.hk/sites/netcomp/articles/Pages/Hardening%20Steps%20to%20Mitigate%20Code%20Injectio
n.aspx

[5] https://code.tutsplus.com/articles/preventing-code-injection--net-36946

[6] https://www.netsparker.com/blog /web-security/crlf-http-header/

[7] https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html

[8] https://portswigger.net/web-security/os-command-injection

[9] https://www.checkmarx.com/knowledge/knowledgebase/OS-Command_Injection

[10] https://web.cs.ucdavis.edu/~su/publications/pldi07.pdf

[11] https://www.sciencedirect.com/science/article/abs/pii/S0950584914001700

[12] S. W. Boyd and A. D. Keromytis. SQLrand: Preventing SQL injection attacks. In International Conference on Applied Cryptography and Network Security (ACNS), LNCS, volume 2, 2004