File Inclusion Vulnerabilities

Sankalana Gunawardhana,
Faculty of Graduate Studies and Research
Sri Lanka Institute of Information Technology
Colombo, Sri Lanka
replacefakes@gmail.com

***Abstract*— File inclusion vulnerabilities allow an attacker to include a file in the web application that was not intended to be included by the developers. Using this an attacker is able to output the contents of restricted files but also depending on the severity of the vulnerability it can also lead to more dangerous attacks such as Remote Code Execution, Local code execution and even Denial of Service.**

***Index Terms*— file inclusion, local file inclusion, remote file inclusion**

I. Introduction

This is a type of vulnerability that can be commonly found among web applications. More precisely web applications that use scripting at run time. This type of vulnerability can look like it goes hand in hand with directory traversal attacks. But the methods of execution for both vary drastically. In directory traversal, unauthorized access to the file system of a server is obtained through a vulnerability present in a web application but file inclusion disrupts the execution sequence in a web app. Using this type of vulnerability, it is possible to even obtain remote code execution on the said web server. File inclusion mainly divides into 2 categories as remote file inclusion and local file inclusion vulnerability. Remote file inclusion happens when a remote file or a file that is currently not in the web server is executed by the web application without a developer including it. Local file inclusion has the same concept but instead of remote files the server uses local files for execution. These local files could be either original server files or files that were uploaded to the server by an attacker. Both types of attacks yield the same result given the same remote or local file but the method of attack is used depending on what is vulnerable on the web app. It is possible for the developer to safeguard against remote file inclusions and the local file inclusion might go unchecked.

II. the Vulnerability

Both remote file inclusion and local file inclusion can happen due to unvalidated user input. This can happen either at the client side or the server side. If it happens at the server side, it usually depends on the scripting language used. For example, in the PHP language if there was an unvalidated use of user input along with a filesystem function this type of vulnerability occurs. In PHP language there are directives which allow filesystem functions to utilize a URL to obtain files from the internet. These directives are ‘allow_url_fopen’ and ‘allow_url_include’. This directive is disabled by default and must be explicitly enabled by the developer. This can cause a remote file inclusion vulnerability if enabled and used without properly sanitizing user input. A common method that this type of directive is used with file system functions is when GET methods are used for page navigation. Consider the following URL.

http://www.example.com/page.php?page=checkout

This type of web application uses GET arguments for page navigation. If directives such as ‘allow_url_include’ is enabled, then this web application is vulnerable for remote file inclusion. The same scenario can be utilized if the web app uses POST arguments for navigation. This is a very unlikely scenario but consider the following code snippets.

```
<form method= ”post” action=”page.php” >
<input type=”submit”>
<input type=”hidden” name=”nextPage”
value=”checkout”>
</form>
```

And at the server side the next page value would be obtained as

```
$nextpage = $_POST[‘nextPage’];
Header(“Location:
http://www.exmaple.com/”.$nextpage.”)”;
```

In php a page redirect could be done using method but this would again be vulnerable for a file inclusion vulnerability if the user input is not properly sanitized. This is a highly unlikely scenario to occur but the concept still stands.

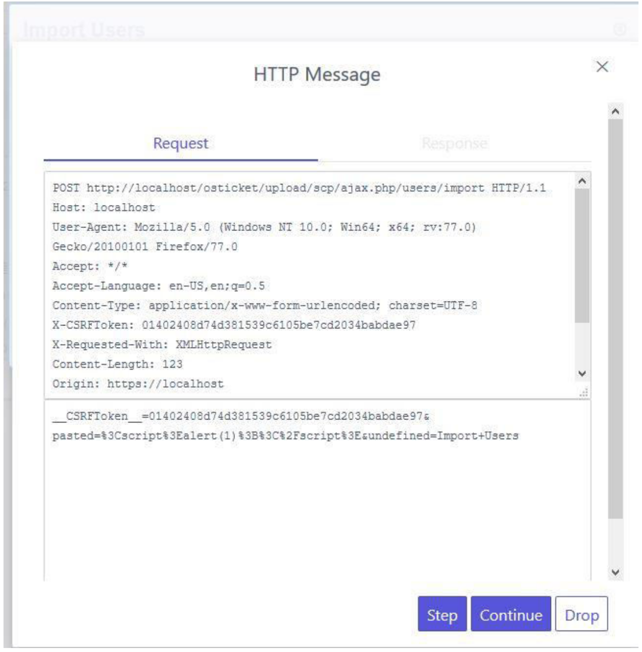
Another scripting language that would be vulnerable to such an attack would be JavaServer pages or JSP. This can include files at runtime the same way as PHP does it using the *‘@include’* statement. If this is used in conjunction with unsanitized user input then it may present a remote or local file inclusion vulnerability.

III. Existing Exploits

Exploits for file inclusion vulnerabilities are very scarce and rare. A main reason for this is that this only occurs purely because of bad development of the web app and all the vulnerable features such as ‘allow_url_include’ are disabled by default and has to be explicitly enabled by the developer. If someone is to enable that they would know the risk of such a feature as well. But even with these constraints in place there are many web applications developed that include file inclusion vulnerabilities. This may come purely as file inclusion vulnerabilities or as a chain of exploits that could result in file inclusion or use file inclusion to chain the other exploits. The best place to look for existing exploits regarding this is exploit-db. Simply tagging as file inclusion would yield around 3500 current exploits and filtering them for web apps would yield only around 35.

IV. Exploitation

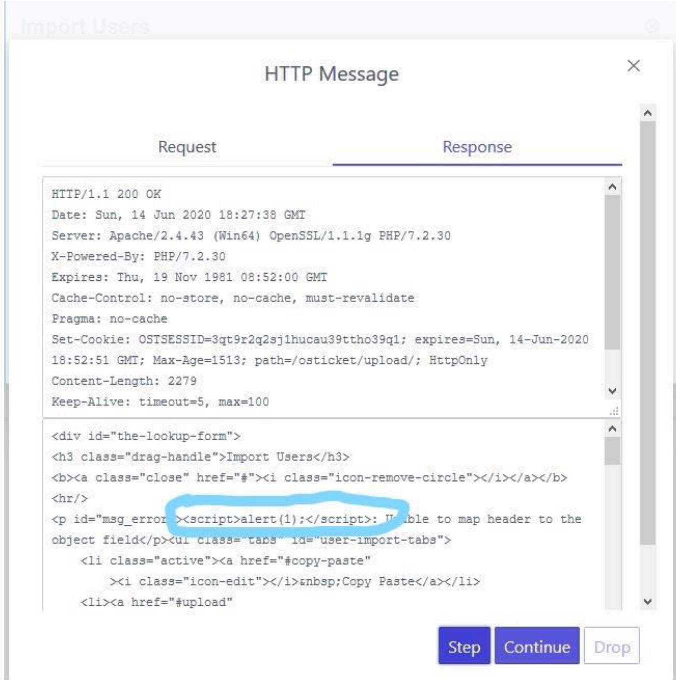
The following exploit will be discussed in this paper https://www.exploit-db.com/exploits/46753. The vulnerable web application utilized in this exploit is OsTicket which is a live chat web application. The vulnerable versions are any below v.1.12 therefore v.1.11 has been used for this demonstration. This exploitation arises from a single point and as a result can be utilized in many ways and areas to achieve a (Kind of) stored XSS and Local file inclusion vulnerability. The main weakness is located in /scp/users.php#users/import which results in an XSS attack as shown below.



```
POST http://localhost/osticket/upload/scp/ajax.php/users/import HTTP/1.1
Host: localhost
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:77.0)
Gecko/20100101 Firefox/77.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
X-CSRFToken: 01402408d74d381539c6105be7cd2034babdae97
X-Requested-With: XMLHttpRequest
Content-Length: 123
Origin: https://localhost
```

```
__CSRFToken__=01402408d74d381539c6105be7cd2034babdae97&
pasted=%3Cscript%3Ealert(1)%3B%3C%2Fscript%3E&undefined=Import+Users
```

Figure 1 - XSS Request



```
HTTP/1.1 200 OK
Date: Sun, 14 Jun 2020 18:27:38 GMT
Server: Apache/2.4.43 (Win64) OpenSSL/1.1.1g PHP/7.2.30
X-Powered-By: PHP/7.2.30
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
Set-Cookie: OSTSESSID=3qt9r2q2sj1hucau39ttho39q1; expires=Sun, 14-Jun-2020
18:52:51 GMT; Max-Age=1513; path=/osticket/upload/; HttpOnly
Content-Length: 2279
Keep-Alive: timeout=5, max=100
```

```
<div id="the-lookup-form">
<h3 class="drag-handle">Import Users</h3>
<b><a class="close" href="#"><i class="icon-remove-circle"></i></a></b>
<hr/>
<p id="msg_error"><script>alert(1);</script>: Unable to map header to the
object field</p><ul class="tabs" id="user-import-tabs">
    <li class="active"><a href="#copy-paste"
        ><i class="icon-edit"></i>&nbsp;Copy Paste</a></li>
    <li><a href="#upload"
```

Figure 2 - XSS Response

An error message is returned with the exact string as the user input appended to the html. Therefore, a simple JS code entered as the input would be executed by the browser as normal JS code sent by the server itself. This can be utilized with the fact that users are able to store their attachments in the ticket conversations in the server side as clear text files. Those stored attachments can be used to invoke a Local File Inclusion Vulnerability since the script can be executed from the server side directly. For this, the attacker only needs access to the agent-panel in the web app.

This exploitation can be executed in numerous ways such as,

- In a situation where there’s an inside agent to execute the script
- In a situation where the support agent is tricked into clicking a link that will execute the script
- Hijacking an already logged in customer support agent’s session.

Since the script is stored in the OsTicket Server it can be used to obtain information within the server itself.

The basic exploitation occurs as follows. When a malicious Customer agent with already access to the agent panel decides to obtain server information, he or she is able to do so with the help of the XSS vulnerability present in the users.php file. In order to do that,

- They must first upload a malicious JS file to the server either via the user-import panel as a csv file or simply create a ticket and upload the file as an attachment in the ticket. Both these methods yield the same results.
- Then the inside agent with access to the agent panel must execute that script using the XSS vulnerability present.

The simple theory behind this exploitation is that it utilizes two vulnerabilities present within the web application; Reflective XSS and Local File Inclusion. This exploitation was found by [4] Ozkan who is a penetration tester in Turkey and it is EDB verified. The overall effectiveness of this exploit depends on the depth an attacker is willing to go. For example, in the original exploit mechanism suggested by Ozkan, an inside agent is required to carry out the exploitation. But if an attacker is willing to carry out the exploitation without an inside agent it is also possible but the exploitation would be much more complicated. This is because of the security mechanisms implemented such as CSRF Tokens. In order to do that an attacker would have to obtain the required information such as CSRF Tokens and Session IDs from a currently logged in Customer Agent using other attack methods.

V. References

[1] P. Mutton, "NetCraft," [Online]. Available: https://news.netcraft.com/archives/2014/04/28/fraudsters-modify-ebay-listings-with-javascript-redirects-and-proxies.html.

[2] L. Dingan, "ZDNET," [Online]. Available: https://www.zdnet.com/article/obama-site-hacked-redirected-to-hillary-clinton/.

[3] G. PODJARNY, "synk," [Online]. Available: https://snyk.io/blog/xss-attacks-the-next-wave/#xss-attacks-on-the-rise-for-a-year-grew-39-in-q1

[4] Ozkan, "Whoami," [Online]. Available: https://pentest.com.tr/whoami/.

[5] Ozkan, "Pentest Blog," [Online]. Available: https://pentest.com.tr/exploits/osTicket-v1-11-XSS-to-LFI.html

[6] Ozkan, "CVE-2019-11537," [Online]. Available: https://nvd.nist.gov/vuln/detail/CVE-2019-11537.

[7] S. Singh, "5 Practical Scenarios for XSS Attacks," [Online]. Available: https://pentest-tools.com/blog/xss-attacks-practical-scenarios/.

[8] KirstenS, "OWASP," [Online]. Available: https://owasp.org/www-community/attacks/xss/.

[9] OWASP, "OWASP," [Online]. Available: https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/07-Input_Validation_Testing/11.1-Testing_for_Local_File_Inclusion.