Understanding File Upload Security for Web Applications

Karishma Pooj[#1], Sonali Patil[*2]

[#]*Student,* [*]*Professor, Department of Information Technology (Information Security),*
K.J. Somaiya College of Engineering (Aff. To Mumbai University),
Vidyavihar East, Mumbai, India

Abstract- *In today's times the web model has become an important mechanism in terms of information and services delivery over the internet. With the success of the internet, it becomes important to take into account the security of the web application layer from various unauthorized user attacks.*
The main reason for security awareness is due to lack of trustworthiness of the applications programming logic or input validation. The best way of preventing application exploitability is to enforce good security policies through the applications. This can be done only when the client and server collaborate to achieve the desired security goals eliminating the possibility of such attacks. In this paper we focus on file upload exploits with respect to web application security. Various test cases will be explained along with the impact which will help security testers and application developers to maintain the confidentiality and integrity of user data. Finally, potential steps for mitigation will be provided in order to restrict such attacks.

Keywords— *Web Application Security, Malicious File Upload, File Upload Security*

I. INTRODUCTION

World Wide Web is considered as the main infrastructure of the global information society on which the world is highly dependent. The Web platform is a complex ecosystem composed of a large number of components and technologies, including HTTP protocol, web server and server-side application development technologies, web browser and client-side technologies [1]. The internet and its services are now easily accessible to us on portable devices Web applications have had a huge impact on fields such as business, education, health and social life, drastically changing the cultural norms and individual behaviors. Every week thousands of new web applications with the power to simplify and ease the human activity process hit the market. But it is also important to understand that with such power it also becomes the responsibility of application builders to be vigilant about security to protect users. With the growing popularity of the good guys developing the applications, there are thousands of hackers working hard to break into these apps to try to phish for user information or implant malware. According to a report by the Web Application Security Consortium, about 49% of the web applications being reviewed contain vulnerabilities of high risk level and more than 13% of the websites can be compromised completely automatically [2].
There are many factors due to which it becomes difficult to secure applications which have be taken into consideration to improve application security. Insecure applications are built due to shortcomings of many factors such as security testing done too late in the SDLC, skipping out on security testing because of the release rush, budget restraints and more commonly, the lack of security awareness by developers. The lack of developer awareness of secure coding standards along with the lack of budget spent on mobile application security are two of the scariest issues. The primary goal of this paper is for developers and testers to understand the common vulnerabilities on file upload functionality which leads to attacks and their respective mitigations for future secure development.

II. LITERATURE REVIEW

The following papers were referred to understand web application security:

A. A Survey on Web Application Security

X. Li | Y. Xue [1] have conducted surveys with respect to web application security techniques. They have categorized three essential properties: state integrity, input validation and logic correctness required for application security along with the future scope of the research.

B. End-to-end Web Application Security

U. Erlingsson | B. Livshits | Y. Xie [3] support the argument that there should be collaboration between the server and client to improve security. They also provide examples mechanisms in order to achieve end-to-end security.

C. A review on Application Security Vulnerabilities

A. Garg | S. Singh [4] provides a look at common web application vulnerabilities such as remote code execution, SQL injection, format string vulnerabilities, cross site scripting, username

enumeration along with their examples and mitigations to overcome their shortcomings.

D. *Systematic Review of Web Application Security Vulnerabilities Detection Methods*

S. Rafique | M. Humayun | Z. Gul | H. Javed [5] explain the cause of vulnerabilities related to web application layer. The paper also provides a review on techniques, stages, approach and tools to detect vulnerabilities.

E. *Web Server Security and Survey on Web Application Security*

S. Almin [6] has described the importance of web server along with the threats posed by hackers. Countermeasures against web server threats are also explained.

F. *Security Testing of Web Applications: Issues and Challenges*

A. Jaiswal | G. Raj | D. Singh [7] provide an insight on the challenges and issues that occur during security testing of web applications. This is done in order to provide inputs to testers and managers with respect to their projects.

This paper adds to the list of issues highlighted by other authors based on OWASP top 10 categorization [8].

III. HOW FILE UPLOAD WORKS

File upload in simple words can be described as transferring a file (photo, audio file, etc.) to a server on the web. To upload data to a server, the Client first starts communication with a server by initiating a TCP/IP connection from the client to the server called the handshake. In this communication, the client starts any communication and not the server. When a connection is established between the client and servers, data transfer can take place between them. This does not need any port forwarding to send/receive data to/from a server. Now the client needs a file to be uploaded and form in a Web page through which the file is sent to the server. This lets the user include one or more files into the form submission. The below is a simple example of file upload form:

<FORM METHOD=" post" ECTYPE=" multipart/form-data" ACTION="">
<INPUT TYPE=" file" NAME=" Example.exe">
<INPUT TYPE =" Submit" VALUE=" Send File">
</FORM>

Once the form is sent over the channel to the server it is often processed so that the files are stored onto the disk of the Web server. Now the server-side script is to be executed once the file is received on the server. The server knows how to handle such a request and stores the data. It saves the file onto the server's disk under some name, but it might just as well process the data only by extracting some information from it.

IV. RESULTS

Different ways in which file upload functionality can be exploited are as explained below:

A. *Case 1 – No Filter*

Summary:

No validation is performed at client end or server end.

Steps:

Select an executable file (e.g. Calc.exe) to be uploaded. Submit the file in the upload feature and observe if the file is uploaded successfully.

Vulnerability Reason:

In this type we directly upload an executable/malicious file. The possibility of this vulnerability occurring is when no validation is applied in application at the client and server end.

B. *Case 2 – Bypass client side validation*

Summary:

Validations applied at client side can be bypassed using developer options.

Steps:

Select an executable file (e.g. Calc.exe) to be uploaded. Before uploading, select the developer options using the F12 button.

In the JavaScript file, search for the function which validates the type of upload and apply break points.

Start debugging and submit the file in the upload feature.

During execution, change the extension of allowable file type to malicious type in JavaScript function and observe if the file is uploaded successfully.

Another way to perform this is to return a true value from the JavaScript function which performs file validation.

Vulnerability Reason:

The breakpoints applied are to change the allowable extension from a particular type (say .pdf) to another type(.exe) which is disallowed. The file is uploaded due to validations performed only at client side.

C. *Case 3 – Perform stored XSS on file name*

Summary:

An attacker is able to perform stored XSS using file upload feature.

Steps:

Select a white-listed file **(e.g. Test.txt)** and upload the file using the submit feature.

As soon as the submit button is clicked, intercept the request using a proxy tool.

Change the file name from **"Test.txt"** to **"XSS<img src="" onerror=alert(document.cookie)>Test.txt"** and forward the request to the server.

Vulnerability Reason:
The attack is possible as there is no sanitization on file name.

D. Case 4 – No file content validation

Summary:
No validation is performed to check the contents of the file uploaded by the end user.

Steps:
Select a white-listed file (e.g. Test.txt) and upload the file using the submit feature.
As soon as the submit button is clicked, intercept the request using a proxy tool.
Add EICAR value in the file body and forward the request to the server.

Vulnerability Reason:
Since file is a part of allowable extensions, client side validation allows the file to be uploaded. No content validation is performed at server side; hence the file is uploaded.

E. Case 5 – No file size validation

Summary:
No validation is performed to check the size of the file uploaded by the end user.

Steps:
Select a white-listed file whose size should be larger than required, based on the business logic (say 100Mb). Upload the file using the submit feature.
As soon as the submit button is clicked, intercept the request using a proxy tool.
Change the value of size parameter in the request and forward the request to the server.
Another way to do is to change the value of the JavaScript function which validates the size of file to true.

Vulnerability Reason:
Since file is a part of allowable extensions, client side validation allows the file to be uploaded. No file size validation is performed at server side; hence the file is uploaded.

F. Case 6 – Bypassing validation based on content type/mime type

Summary:
When the validation is based just on content type, attack can be made by manipulating the content-type of a file which specifies the nature of data.

Steps:
Select the executable file (Test.exe) to be uploaded. Upload this file into the upload feature by clicking on the submit button also intercepting the request in a proxy tool.
Now we change the content-type of an executable file from application/x-msdownload to an allowable content-type (say text/plain) and forward the request.

Vulnerability Reason:
This vulnerability is possible when the validation is done only based on the content-type, but the body contains executable functions.

G. Case 7 – Bypassing blacklisting using Multiple extension (Type I)

Summary:
This type is possible by using more than one type of file extension.

Steps:
Select an executable file (Test.exe) and rename it to an allowable file extension (Test.exe.jpg).
This can also be done by intercepting the request using a proxy tool and changing the values.
Upload this file/forward the request and observe the results.

Vulnerability Reason:
Assuming that .htaccess file has following line of code:

AddHandler php5-script.php

This line checks only if the uploaded extension is a PHP; it doesn't necessarily check what order it is placed in. For example, it would execute all the following files as PHP due to vulnerable code in .htaccess file:

Test.php.jpg, test.php.pdf, etc.

H. Case 8- Bypassing blacklisting using Multiple extension (Type II)

Summary:
This type is performed by separating file extensions using Semi colons. This attack is possible on IIS server 6 or prior.

Steps:
Select an executable file (Test.exe) and rename it(Test.exe;.jpg) to an allowable file extension.
This can also be done by intercepting the request using a proxy tool and changing the values.
Upload this file/forward the request and observe the results.

Vulnerability Reason:
During file upload, when Test.exe;.jpg is uploaded, server will only check the first dot from the right. When it sees .jpg, the server allows the file to get successfully uploaded concluding that this extension is not in the list of dangerous extension. IIS server executes Text.exe;.jpeg as Text.exe. Also "test.exe/file.txt" is later executed as test.exe.

I. Case 9 – Bypassing blacklisting using multiple extensions (Type III)

Summary:
This type is performed by using forbidden file extensions along with file extension which is not permitted by the application.

Steps:

Select an executable file (Test.exe) and rename it (Test.exe.aabbcc) with a file extension which the server or client does not understand.

This can also be done by intercepting the request using a proxy tool and changing the values.

Upload this file/forward the request and observe the results.

Vulnerability Reason:

When last extension (in our example .aabbcc), is not specified in the list of mime-types known to the web server, Test.exe.aabbcc will be interpreted as Test.exe and will be executed.

J. Case 10 – Bypass blacklist using uncommon executable extensions

Summary:

Blacklisting can often be bypassed using uncommon executable extensions such as php3, php4, php5, shtml, phtml, cgi which are understood by server.

Steps:

Upload any server executable file in the file upload feature of the application.

Vulnerability Reason:

The vulnerability is possible since these file extensions are default files of the server and are accepted when they are uploaded. These extension codes can be modified with malicious content.

K. Case 11 – Bypass blacklist by changing case in extension

Summary:

Blacklisting is bypassed by changing a number of letters to their capital forms to bypass case sensitive rules (e.g. "file.aSp" or "file.PHp3").

Steps:

Select any malicious file which is blacklisted by the server (Test.php).

Change some letters in the extension to their capital form (say Test.pHp or Test.PHp) and upload the file.

Vulnerability Reason:

This vulnerability occurs when validation applied for filtering disallowed files is not proper. Security checks made for filtering disallowed files should be case insensitive.

L. Case 12 – Bypass blacklist type by adding neutral spaces

Summary:

Blacklisting is bypassed by adding neutral spaces or dots in Windows file system and slash and dots in Unix file system.

Steps:

Select any malicious file which is blacklisted by the server (say Test.php).

Add some trailing spaces or dots after the extension (say Test.php……) and upload the file in file upload functionality of the application.

Vulnerability Reason:

Finding neutral characters after a filename such as trailing spaces and dots in Windows file system or dot and slash characters in a Linux file system are removed automatically. These characters at the end of a filename will be removed automatically (e.g. "file.asp ... ...", "file.asp ", or "file.asp.").

M. Case 13 – Bypass blacklist using Null Byte

Summary:

This attack is possible by using NULL Byte in the allowed file extensions.

Steps:

Select executable file (Test.exe) and rename it (Test.exe%00.jpg).

Another way to add the NULL byte is by intercepting the request in a proxy tool.

Observe if the file is uploaded successfully.

Vulnerability Reason:

Web application will accept the file as jpg. Null byte (0x00) is used as a string terminator. When web server tries to read it stops at Test.exe as it encounters a null byte and the file is treated as executable file.

N. Case 14 – Bypass using embedded executable extension in excel sheet

Summary:

In this type we try to embed an executable file within an excel file and upload it onto the server which allows .xls or .xlsx formats.

Steps:

First we embed an executable file into the excel file using the Object option from the toolbar.

Now we write a small script/formula in the cell such that the executable file embedded is executed as soon as the excel file is opened.

Thus an excel file is created where malicious file and code is written.

Upload this file and observe if the file was uploaded successfully.

Vulnerability Reason:

The attack is possible since there is no validation done at client side to check the body contents of the attached file.

O. Case 15 – Bypass using embedded executable in pdf file

Summary:

In this type we try to embed an executable file within a PDF file and upload it onto the server that allows .pdf format.

Steps:

First we embed an executable file into a PDF file using the path View>Comment>Annotations from the toolbar.

From annotations, select attach file and place the cursor in the document and browse the file you want to attach. Attach an excel file with an executable file embedded into it.

Thus a pdf file is created where malicious excel file is embedded.

Upload this file and observe if the file was uploaded successfully.

Vulnerability Reason:

The attack is possible since there is no validation done at client side to check the body contents of the attached file.

P. Case 16 – Overwriting critical files

Summary:

In this type we try to overwrite critical files on the webserver such as .htaccess or web.config file.

Steps:

Find the path of the .htaccess file onto the server using information gathering.

Following line can be added in .htaccess file before uploading:

AddType application/x-httpd-php .gif

Once the path is found, the above manipulated .htaccess file can be uploaded on the desired path using the file upload feature.

Vulnerability Reason:

If the webserver allows to modify sensitive files such as .htaccess file or web.config file, we can upload files to modify how different files should be executed on the server. The .htaccess file contains restrictions for a particular folder. Now there are locations where the server allows the user to upload and overwrite files. Using this feature of over writing an attacker can replace the .htaccess file with a manipulated one allowing permissions to execute scripts.

The above malicious line (AddType application/x-httpd-php .gif) explained would basically execute every gif file inside the webserver as a PHP. So once an attacker uploads .htaccess file containing that code, attacker can rename any malicious file as Test.gif which will be interpreted as a PHP file by the webserver.

V. MITIGATION

- The application should use a whitelist of allowed file types. This list determines the types of files that can be uploaded, and rejects all files that do not match approved types.
- Only allow authorized and authenticated users to use the feature.
- Serve fetched files from your application rather than directly via the web server.
- Write to the file when you store it to include a header that makes it non-executable.
- Define a .htaccess file that will only allow access to files with allowed extensions.
- Do not place the .htaccess file in the same directory where the uploaded files will be stored. It should be placed in the parent directory.
- A typical .htaccess which allows only gif, jpg, jpeg and png files should include the following (adapt it for your own need). This will also prevent double extension attacks.
- The most important thing is to keep uploaded files in a location that can't access though the Internet. This can be done either by storing uploaded files outside of the web root or configuring the web server to deny access to the uploads directory.
- Prevent overwriting of existing files (to prevent the .htaccess overwrite attack).
- Don't rely on client-side validation only, since it is not enough. Ideally one should have both server-side and client-side validation implemented. The application should use client- and server-side input validation to ensure evasion techniques have not been used to bypass the whitelist filter.
- Set a pre-defined size and file name length.
- Files that are uploaded should be scanned by antivirus software.
- The application should not use the file name supplied by the user. Instead, the uploaded file should be renamed according to a predetermined convention. Thus, the attacker will encounter problems trying to determine the name of the file in the uploaded folder.

VI. CONCLUSION

In this paper we have highlighted the importance of application security and how users could be affected by such data loss. Due to large user base of web applications it becomes necessary to make organizations aware of application security practices to prevent these types of break-ins. We have demonstrated various ways to bypass the file upload vulnerability using open source tools, along with their mitigations. Security is not a one-time event due to which it insufficient to perform security analysis on the application just once. An application can meet the security requirements only when all the stages of an application development cycle are analyzed securely by developers and testers. This paper aims at providing awareness to apply security measures for file uploads at client and server side which will reduce the security testing cost by itself.

ACKNOWLEDGEMENT

I offer my profound gratitude towards all the staff members of K. J. Somaiya College of Engineering,

Vidyavihar, Mumbai for providing me all academic assistance required to complete this paper.
I would like to thank my colleagues, who have contributed to ease the understanding of this project and this paper by giving their time and taking a keen interest in making this a success.

REFERENCES

[1] X Lie and Y Xue. " A Survey on Web Application Security." Vanderbilt University, "http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.434.7174&rep=rep1&type=pdf".

[2] Web Application Security Statistics, "http://projects.webappsec.org/w/page/13246989/WebApplication SecurityStatistics."

[3] Ulfar Erlingsson, Benjamin Livshits, Yinglian Xie, " Microsoft Reasearch", "http://research-srv.microsoft.com/en-us/um/people/livshits/papers/pdf/hotos07.pdf".

[4] Ashwani Garg, Shekhar Singh. "A Review on Web Application Security Vulnerabilities." International Journal of Advanced Research in Computer Science and Software Engineering (2013): 222-226.

[5] Rafique, Sajjad, Mamoona Humayun, Zartasha Gul, Ansar Abbas, and Hasan Javed. "Systematic Review of Web Application Security Vulnerabilities Detection Methods." Journal of Computer and Communications 03.09 (2015): 28-40.

[6] B. Shaikh, "Web Server Security and Survey on Web Application Security," International Journal on Recent and Innovation Trends in Computing and Communication, vol. 2, no. 1, pp. 114–119, Jan. 2014.

[7] Jaiswal, Arunima, Gaurav Raj, and Dheerendra Singh. "Security Testing of Web Applications: Issues and Challenges." International Journal of Computer Applications88.3 (2014): 26-32.

[8] OWASP Top 10-2013, "https://www.owasp.org/index.php/Top_10_2013-Top_10."