Flare-On 4: Challenge 10 Solution – `shell.php`

Challenge Author: Dominik Weber (@Invalid_handle)

Summary

Modelled after a real-world web shell, this crypto level is a PHP page containing an encrypted HTML stage2 with 3 encrypted JavaScripts. The HTTP POST variable flag is the key. This level is intended to be a two-stage password attack. The first stage gets the MD5 of the password and decrypt stage2, then the second stage derives the flag with a known-plaintext attack.

Description:

Figure 1 depicts the essential contents of `shell.php`

```
<?php
$o__o_ = base64_decode('<Base64 block omitted>');
$o_o   = isset($_POST['o_o']) ? $_POST['o_o'] : "";
$o_o   = md5($o_o) . substr(MD5(strrev($o_o)), 0, strlen($o_o));
for ($o___o = 0; $o___o < 2268; $o___o++) {
    $o__o_[$o___o] = chr((ord($o__o_[$o___o]) ^ ord($o_o[$o___o])) % 256);
    $o_o .= $o__o_[$o___o];
}
if (MD5($o__o_) == '43a141570e0c926e0e3673216a4dd73d') {
    if (isset($_POST['o_o']))
        @setcookie('o_o', $_POST['o_o']);
    $o___o = create_function('', $o__o_);
    unset($o_o, $o__o_);
    $o___o();
} else {
 echo '<form method="post" action="shell.php"><input type="text" name="o_o" value="
"/><input type="submit" value="&gt;"/></form>';
 }
?>
```

Figure 1: Contents of `shell.php`

The HTTP post variable, o_o is the flag for the level. This string is then hashed with MD5 and the textual hash is the first 16 bytes of a decryption XOR key. The remainder of the key bytes is the MD5 of the reverse string array, truncated to password length size. <u>The resulting XOR key is password length dependent</u> and consists of characters `0-9` and `a-f`. This key decrypts the Base64 blob into a PHP function that gets executed. Looking at the code, we can see that the resulting plain text of this function has 2268 bytes. The MD5 of this data is should be 43a141570e0c926e0e3673216a4dd73d.

A Plan of Attack

The first thing we need to do, is replicate the decryption algorithm to use it in brute-forcing the encryption key. Also, we expect/hope the output to be in the range 20-7E with 0D 0A and possibly 09 (horizontal tab). Then we determine that the key length is between 33 and 64. This is done by looking at patterns in the encrypted data, when selecting a line length of 64 bytes, we see a lot of bytes line up well at 0x840, where 26 bytes line up. At this point we and brute force the key and generate an array of possibilities for every key character as we see in Figure 2.

1	cd	17	abcde	33	012356789	49	d
2	abe	18	abcdef	34	012345679	50	07
3	0123456789	19	bf	35	037	51	156
4	9	20	347	36	cd	52	012345679
5	256	21	35	37	def	53	abcf
6	01234678	22	56	38	047	54	abcdef
7	bf	23	012456789	39	25	55	abcdf
8	8	24	156	40	047	56	abcdef
9	04	25	8	41	123456789	57	012356789
10	abf	26	abcdef	42	034789	58	037
11	023456789	27	cd	43	ae	59	9
12	0123456789	28	9	44	af	60	aef
13	b	29	d	45	c	61	012367
14	9	30	9	46	037	62	abcef
15	03	31	012345679	47	9	63	aef
16	0123456789	32	abcdef	48	12356789	64	cd

Figure 2: possible characters for the key

Characters 28-30 are unique. We now have a key of all the unique characters:

```
---9---8----b9----------8--9d9-------------c-9-d---------9-----
```

We apply this to the encrypted input al look at the output, finding the string "b-:e64" that might be base64 we then update the key with the sequence 8a49d9. (It is useful to encounter the bytes 0D or 0A, since the next / previous byte is the other one.) We repeat this process to arrive at:

```
db6952b84a49b934acb436418ad9d93d237df05769afc796d067bccb379f2cac
```

This is the decryption key to get us stage 2. We still do not have the flag or the password, but we have its MD5 value: db6952b84a49b934acb436418ad9d93d

Stage 2

The second stage from Figure 3 is a PHP function that will decrypt and display a JavaScript animation:

```
$d   = '';
$key = "";
if (isset($_POST['o_o']))
    $key = $_POST['o_o'];
if (isset($_POST['hint']))
    $d = "www.p01.org";
if (isset($_POST['t'])) {
    if ($_POST['t'] == 'c'){
        $d   = base64_decode('<Base64 blob 1>');
        $key = preg_replace('/(.)../', '$1', $key);
    }
    if ($_POST['t'] == 's'){
        $d   = base64_decode('<Base64 blob 2>);
        $key = preg_replace('/.(.)./', '$1', $key);
    }
    if ($_POST['t'] == 'w'){
        $d   = base64_decode('<Base64 blob 3>');
        $key = preg_replace('/..(.)/', '$1', $key);
    }
    while (strlen($key) < strlen($d))
        $key = $key . $key;
    $d = $d ^ $key;
  }

if (strlen($d))
    echo $d;
else

 echo '<form action="shell.php" method="post"><input type="hidden" name="o_o" value
="' . $key . '"><input type="radio" name="t" value="c"> Raytraced Checkboard<br> <i
nput type="radio" name="t" value="s"> p01 256b Starfield<br> <input type="radio" na
me="t" value="w"> Wolfensteiny<br><input type="submit" value="Show"/></form>';
```

Figure 3: Contents of the `second stage`

The decryption key is derived from the flag by using every third character. For instance, if the flag would be ABC123, the key for `blob 1` would be A1, the one for `blob 2` would be B2 and the one for `blob 3` would be C3. Since the flag's length is 39 bytes, the key lengths are 13 and the way to derive this key is to adjust the line length to 13 as shown in Figure 4 (from 010 editor):

	0	1	2	3	4	5	6	7	8	9	A	B	C	0123456789ABC
0000h:	48	37	06	1E	0D	5F	79	55	08	12	1B	1B	0F	H7..._yU.....
000Dh:	11	61	20	12	18	15	06	3E	57	03	16	4F	20	.a ....>W..O
001Ah:	1C	3A	11	18	03	0E	15	2D	50	5A	5D	1B	0A	.:.....-PZ]..
0027h:	00	33	17	4D	6C	6B	48	7E	19	4B	52	22	02	.3.MlkH~.KR".
0034h:	00	37	1B	16	14	41	53	2F	04	57	55	4F	2B	.7...AS/.WUO+
0041h:	11	31	00	1A	41	4C	59	61	39	6C	4E	0D	0C	.1..ALYa9lN..
004Eh:	10	26	52	1A	05	5C	36	7F	40	03	0A	1B	5E	.&R..\6.@...^
005Bh:	07	31	1D	04	41	03	13	1C	5B	0A	1D	1D	5E	.1..A...[...^
0068h:	44	7F	14	1F	00	06	2B	32	5B	02	42	52	41	D.....+2[.BRA
0075h:	00	00	00	00	00	00	00	00	00	00	00	00	00	....
0082h:	56	61	7F	79	5D	11	06	3A	14	12	1B	1B	0F	Va.y]..:.....
008Fh:	11	62	50	31	18	41	39	3E	40	0E	1B	0A	16	.bP1.A9>@....
009Ch:	54	78	02	43	50	46	54	17	51	08	00	06	41	Tx.CPFT.Q...A
00A9h:	54	36	16	4E	11	5F	48	2C	57	14	1B	1F	17	T6.N._H,W....
00B6h:	4A	31	4F	00	04	15	3D	31	40	03	00	19	02	J1O...=1@....
00C3h:	18	77	50	15	0E	13	5C	31	1F	5B	45	43	0A	.wP...\1.[EC.
00D0h:	49	34	5E	23	5C	46	32	71	68	3A	1C	48	58	I4^#\F2qh:.HX
00DDh:	1D	72	4F	42	4E	0A	4F	0F	1F	5B	22	34	0A	.rOBN.O..["4.
00EAh:	51	6D	4D	5B	08	44	46	75	5E	4B	18	44	0D	QmM[.DFu^K.D.
00F7h:	5B	34	2C	19	48	47	45	65	06	3B	5B	05	5E	[4,.HGEe.;[.^
0104h:	1F	70	1B	48	11	4F	1D	31	5A	03	00	27	37	.p.H.O.1Z..'7
0111h:	39	13	4F	23	4F	12	18	36	57	03	5A	5B	4A	9.O#O..6W.Z[J
011Eh:	56	73	19	4E	57	55	5D	63	1B	15	11	1D	0A	Vs.NWU]c.....
012Bh:	04	2B	4C	7E	6B	5D	5B	3D	5B	02	0B	51	6E	.+L~k][=[..Qn
0138h:	7E	63	5D	1B	15	0C	18	61						~c]....a

Figure 4: Encrypted Blob 1

There are several hints; 0x75 has a line of just 0x00 and in the stage 2 source, we find an URL `www.p01.org` that leads us to a page from Mathieu 'p01' Henri that contains several very beautiful JavaScript animations. From our source we have three names:

- Raytraced Checkboard
- p01 256b Starfield
- Wolfensteiny

These will give us some plain text. Also since it is likely that the returned data starts with `<html>`, we can pad this to 13 bytes (see Figure 5):

0	1	2	3	4	5	6	7	8	9	A	B	C	0123456789ABC
3C	68	74	6D	6C	3E	00	00	00	00	00	00	00	<html>.......

Figure 5: Initial key for <html> plaintext

Applying this XOR key to the first 13 bytes yields `t_rsaa` (see Figure 6):

	0	1	2	3	4	5	6	7	8	9	A	B	C	0123456789ABC
0000h:	74	5F	72	73	61	61	79	55	08	12	1B	1B	33	t rsaayU....3

Figure 6: Deriving the first 6 bytes of the key

The blob, decrypted with `74 5F 72 73 61 61 00 00 00 00 00 00 00` yields Figure 7:

	0	1	2	3	4	5	6	7	8	9	A	B	C	0123456789ABC
0000h:	3C	68	74	6D	6C	3E	79	55	08	12	1B	1B	0F	<html>yU.....
000Dh:	65	3E	52	61	79	74	06	3E	57	03	16	4F	20	e>Rayt.>W..O
001Ah:	68	65	63	6B	62	6F	15	2D	50	5A	5D	1B	0A	heckbo.-PZ]..
0027h:	74	6C	65	3E	0D	0A	48	7E	19	4B	52	22	02	tle>..H~.KR".
0034h:	74	68	69	65	75	20	53	2F	04	57	55	4F	2B	thieu S/.WUO+
0041h:	65	6E	72	69	20	2D	59	61	39	6C	4E	0D	0C	enri -Ya9lN..
004Eh:	64	79	20	69	64	3D	36	7F	40	03	0A	1B	5E	dy id=6.@...^
005Bh:	73	6E	6F	77	20	62	13	1C	5B	0A	1D	1D	5E	snow b..[...^
0068h:	30	20	66	6C	61	67	2B	32	5B	02	42	52	41	0 flag+2[.BRA
0075h:	74	5F	72	73	61	61	00	00	00	00	00	00	00	t_rsaa.......
0082h:	22	3E	0D	0A	3C	70	06	3A	14	12	1B	1B	0F	">..<p.:.....
008Fh:	65	3D	22	42	79	20	39	3E	40	0E	1B	0A	16	e="By 9>@....
009Ch:	20	27	70	30	31	27	54	17	51	08	00	06	41	'p01'T.Q...A
00A9h:	20	69	64	3D	70	3E	48	2C	57	14	1B	1F	17	id=p>H,W....
00B6h:	3E	6E	3D	73	65	74	3D	31	40	03	00	19	02	>n=set=1@....
00C3h:	6C	28	22	66	6F	72	5C	31	1F	5B	45	43	0A	l("for\1.[EC.
00D0h:	3D	6B	2C	50	3D	27	32	71	68	3A	1C	48	58	=k,P='2qh:.HX
00DDh:	69	2D	3D	31	2F	6B	4F	0F	1F	5B	22	34	0A	i-=1/kO..["4.
00EAh:	25	32	3F	28	69	25	46	75	5E	4B	18	44	0D	%2?(i%Fu^K.D.
00F7h:	2F	6B	5E	6A	29	26	45	65	06	3B	5B	05	5E	/k^j)&Ee.;[.^
0104h:	6B	2F	69	3B	70	2E	1D	31	5A	03	00	27	37	k/i;p..1Z..'7
0111h:	4D	4C	3D	50	2E	73	18	36	57	03	5A	5B	4A	ML=P.s.6W.Z[J
011Eh:	22	2C	6B	3D	36	34	5D	63	1B	15	11	1D	0A	",k=64]c.....
012Bh:	70	74	3E	0D	0A	3C	5B	3D	5B	02	0B	51	6E	pt>..<[=[..Qn
0138h:	0A	3C	2F	68	74	6D	18	61						.</htm.a

Figure 7: 6/12 bytes applied

We see a string **`Raytra`** that will likely be **`Raytraced Checkboard`**

This allows us to derive the key with the same steps as above, yielding `t_rsaat_4froc` and a decrypted `blob 1` shown in Figure 8:

```
       0  1  2  3  4  5  6  7  8  9  A  B  C   0123456789ABC
0000h: 3C 68 74 6D 6C 3E 0D 0A 3C 74 69 74 6C  <html>..<titl
000Dh: 65 3E 52 61 79 74 72 61 63 65 64 20 43  e>Raytraced C
001Ah: 68 65 63 6B 62 6F 61 72 64 3C 2F 74 69  heckboard</ti
0027h: 74 6C 65 3E 0D 0A 3C 21 2D 2D 20 4D 61  tle>..<!-- Ma
0034h: 74 68 69 65 75 20 27 70 30 31 27 20 48  thieu 'p01' H
0041h: 65 6E 72 69 20 2D 2D 3E 0D 0A 3C 62 6F  enri -->..<bo
004Eh: 64 79 20 69 64 3D 42 20 74 65 78 74 3D  dy id=B text=
005Bh: 73 6E 6F 77 20 62 67 43 6F 6C 6F 72 3D  snow bgColor=
0068h: 30 20 66 6C 61 67 5F 6D 6F 64 30 3D 22  0 flag_mod0="
0075h: 74 5F 72 73 61 61 74 5F 34 66 72 6F 63  t_rsaat_4froc
0082h: 22 3E 0D 0A 3C 70 72 65 20 74 69 74 6C  ">..<pre titl
008Fh: 65 3D 22 42 79 20 4D 61 74 68 69 65 75  e="By Mathieu
009Ch: 20 27 70 30 31 27 20 48 65 6E 72 69 22   'p01' Henri"
00A9h: 20 69 64 3D 70 3E 3C 73 63 72 69 70 74   id=p><script
00B6h: 3E 6E 3D 73 65 74 49 6E 74 65 72 76 61  >n=setInterva
00C3h: 6C 28 22 66 6F 72 28 6E 2B 3D 37 2C 69  l("for(n+=7,i
00D0h: 3D 6B 2C 50 3D 27 46 2E 5C 5C 6E 27 3B  =k,P='F.\\n';
00DDh: 69 2D 3D 31 2F 6B 3B 50 2B 3D 50 5B 69  i-=1/k;P+=P[i
00EAh: 25 32 3F 28 69 25 32 2A 6A 2D 6A 2B 6E  %2?(i%2*j-j+n
00F7h: 2F 6B 5E 6A 29 26 31 3A 32 5D 29 6A 3D  /k^j)&1:2])j=
0104h: 6B 2F 69 3B 70 2E 69 6E 6E 65 72 48 54  k/i;p.innerHT
0111h: 4D 4C 3D 50 2E 73 6C 69 63 65 28 34 29  ML=P.slice(4)
011Eh: 22 2C 6B 3D 36 34 29 3C 2F 73 63 72 69  ",k=64)</scri
012Bh: 70 74 3E 0D 0A 3C 2F 62 6F 64 79 3E 0D  pt>..</body>.
0138h: 0A 3C 2F 68 74 6D 6C 3E                 .</html>
```

Figure 8: Decrypted Blob 1

After doing this to the other two `blobs`, we see that they contain the strings:

```
flag_mod0="t_rsaat_4froc"
flagmod1="hx__ayowkleno"
flag_mod_2="3Oiwa_o3@a-.m"
```

These are also the decryption keys for the blobs and when read top to bottom, and left to right, yields the flag: `th3_xOr_is_waaaay_too_w34k@flare-on.com`

Make sure to check out the three cool JavaScript animations from Mathieu 'p01' Henri. He kindly allowed my use of them.