T9000: Advanced Modular Backdoor Uses Complex Anti-Analysis Techniques - Palo Alto Networks Blog

Most custom backdoors used by advanced attackers have limited functionality. They evade detection by keeping their code simple and flying under the radar. But during a recent investigation we found a backdoor that takes a very different approach. We refer to this backdoor as T9000, which is a newer variant of the T5000 malware family, also known as Plat1.

In addition to the basic functionality all backdoors provide, T9000 allows the attacker to capture encrypted data, take screenshots of specific applications and specifically target Skype users. The malware goes to great lengths to identify a total of 24 potential security products that may be running on a system and customizes its installation mechanism to specifically evade those that are installed. It uses a multi-stage installation process with specific checks at each point to identify if it is undergoing analysis by a security researcher.

The primary functionality of this tool is to gather information about the victim. In fact, the author chose to store critical files dropped by the Trojan in a directory named “Intel.” T9000 is pre-configured to automatically capture data about the infected system and steal files of specific types stored on removable media.

We have observed T9000 used in multiple targeted attacks against organizations based in the United States. However, the malware’s functionality indicates that the tool is intended for use against a broad range of users. In this report, we share an analysis of each stage in T9000’s execution flow. Stay tuned for a future report in which we will provide more detail on how this tool has been used and the infrastructure we have identified as part of our analysis.

T9000 Backdoor Analysis

The entire execution flow of the malware is represented in the following diagram:

As this malware uses a multistage execution flow, we’ll discuss each stage individually.

Initial Exploitation

The sample of T9000 used in this analysis was originally dropped via a RTF file that contained exploits for both CVE-2012-1856 and CVE-2015-1641. When triggered, an initial shellcode stage is run, which is responsible for locating and executing a secondary shellcode stub. The second stage shellcode reads the initial RTF document and seeks to the end of the file, using the last four bytes as the size of the embedded payload.

With the payload size confirmed, the shellcode will create a file in the %TEMP% folder using a temporary filename. The shellcode will decrypt and subsequently load the embedded payload in the RTF file. The decrypted payload is written to the temporary file and executed using WinExec. The shellcode then attempts to decrypt an embedded decoy document with the same algorithm used to decrypt the payload, which it will save to %TEMP%\~tmp.doc path. This file is opened using the following command:

cmd /C %TEMP%\~tmp.doc
However, this particular sample did not contain a decoy document.

Stage 1

When this temporary file is initially executed, it will begin by creating the following mutex to ensure only one instance of the malware is running at a given time:

820C90CxxA1B084495866C6D95B2595xx1C3
It continues to perform a number of checks for installed security products on the victim machine. The following security platforms are queried by checking entries within the HKLM\Software\ registry path:

- Sophos
- INCAInternet
- DoctorWeb
- Baidu
- Comodo
- TrustPortAntivirus
- GData
- AVG
- BitDefender
- VirusChaser
- McAfee
- Panda
- Trend Micro
- Kingsoft
- Norton
- Micropoint
- Filseclab
- AhnLab

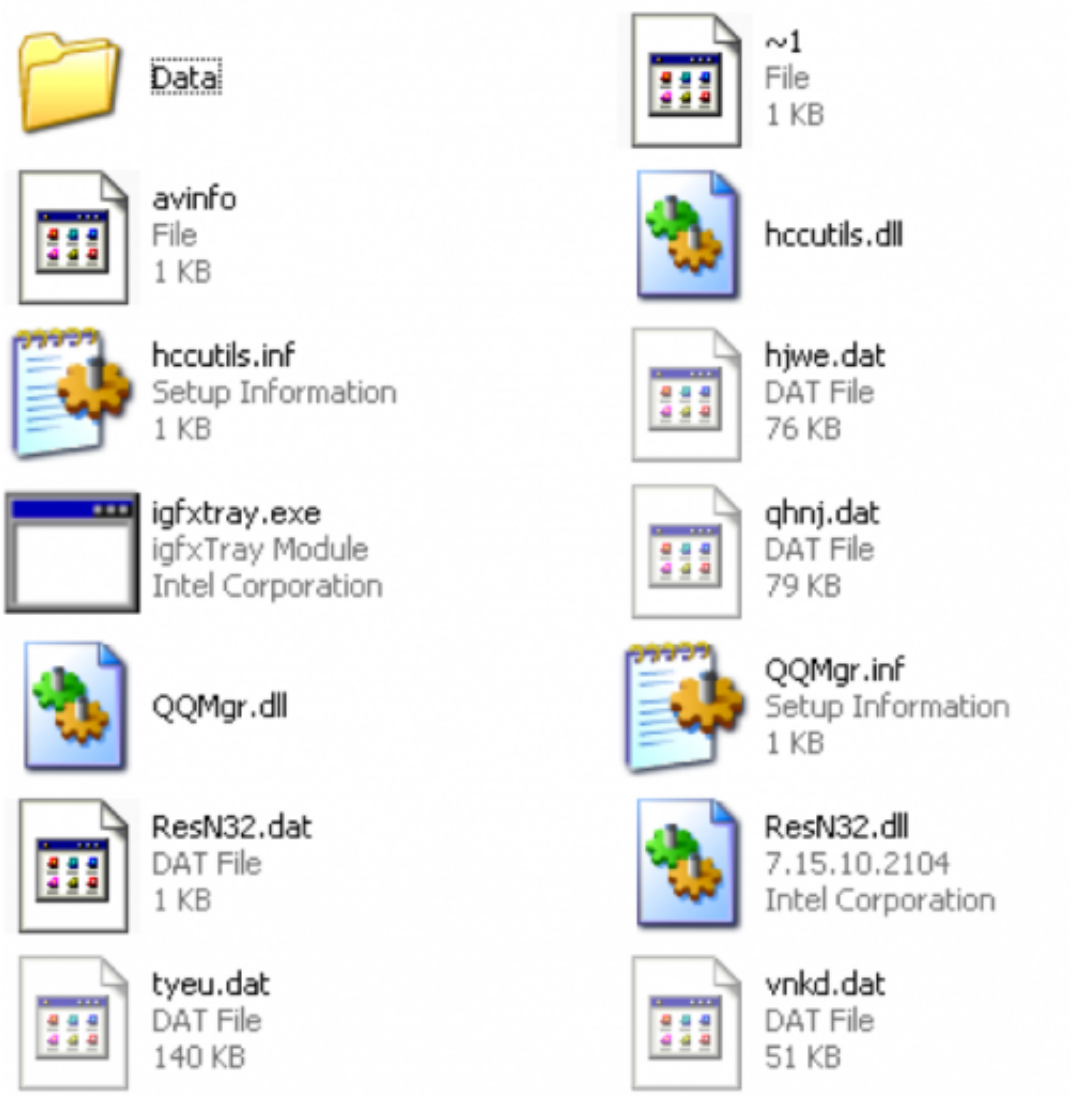
- JiangMin
- Tencent
- Avira
- Kaspersky
- Rising
- 360

These security products are represented by a value that is binary AND-ed with any other products found. The following numbers represent each respective security product.

0x08000000 : Sophos
0x02000000 : INCAInternet
0x04000000 : DoctorWeb
0x00200000 : Baidu
0x00100000 : Comodo
0x00080000 : TrustPortAntivirus
0x00040000 : GData
0x00020000 : AVG
0x00010000 : BitDefender
0x00008000 : VirusChaser
0x00002000 : McAfee
0x00001000 : Panda
0x00000800 : Trend Micro
0x00000400 : Kingsoft
0x00000200 : Norton
0x00000100 : Micropoint
0x00000080 : Filseclab
0x00000040 : AhnLab
0x00000020 : JiangMin
0x00000010 : Tencent
0x00000004 : Avira
0x00000008 : Kaspersky
0x00000002 : Rising
0x00000001 : 360

So, for example, if both Trend Micro and Sophos were discovered on a victim machine, the resulting value would be 0x08000800. This numerical value is written to the following file:

%APPDATA%\Intel\avinfo
The malware proceeds to drop the following files to the %APPDATA%\Intel directory:



Additionally, the following two files are written to the Data directory:

The following table provides a description of each file dropped:

File Name	Description
~1	Debug information about files used by malware.
avinfo	Installed security products on victim.
hccutils.dll	Malicious DLL. Loads ResN32.dll.
hccutils.inf	Malicious INF file. Points to hccutils.dll.
hjwe.dat	Encrypted core of malware family.

igfxtray.exe	Legitimate Microsoft executable. Loads hccutils.dll.
qhnj.dat	Encrypted plugin. Hooks a number of functions and logs results.
QQMgr.dll	Malicious DLL. Sets persistence via Run registry key.
QQMgr.inf	Malicious INF file. Points to QQMgr.dll
ResN32.dat	String pointing to path of encrypted core of malware.
ResN32.dll	Malicious DLL. Decrypts, decompresses, and loads core malware.
tyeu.dat	Encrypted plugin. Takes screenshots and collects Skype information.
vnkd.dat	Encrypted plugin. Finds files on removable drives on victim machine.
dtl.dat	Encrypted configuration information.
glp.uin	Plugin configuration information.

You'll notice that QQMgr* files are not listed in the original malware execution flow diagram. In the event the victim is running any of the following operating system versions, as well as either Kingsoft, Filseclab, or Tencent security products, the malware will be installed using an alternative method.

- Windows 2008 R2
- Windows 7
- Windows 2012
- Windows 8

In such a situation, the malware will find and run the built-in Microsoft Windows InfDefaultInstall.exe program, which will install a DLL via an INF file. Should Tencent be installed, the malware will execute the InfDefaultInstall.exe program with an argument of 'QQMgr.inf'. Otherwise, it will use 'hccutils.inf' as an argument.

QQMgr.inf will install the QQMgr.dll, while hccutils.inf will install the hccutils.dll library. QQMgr.dll will set the following registry key:

HKLM\Software\Microsoft\Windows\CurrentVersion\Run\Eupdate – %APPDATA%\Intel\ResN32.dll
The QQMgr.dll file has the following debug string found within it:

*H:\WORK\PROJECT\InfInstallBypassUAC\Release\BypassUAC.pdb*The hccutils.dll file is described later within this post.
After the malware drops the required files, by default the malware will spawn

%APPDATA%\Intel\igfxtray.exe in a new process, which begins the second stage of the malware's execution.

Stage 2

The igfxtray.exe is a legitimate Microsoft Windows executable that sideloads the malicious hccutils.dll DLL file. This DLL has the following debug string embedded within it:

D:\WORK\T9000\hccutils_M4\Release\hccutils.pdb
Upon loading this malicious DLL, the malware will initially perform the same queries for security products that were witnessed in stage 1.

Three separate techniques for starting stage 3 are used depending on the properties of the victim.

The first technique is used if the victim meets the following criteria:

- Microsoft Windows 8 / Windows Server 2012 R2
- DoctorWeb security product installed

For this situation, the following registry key is set:

HKLM\Software\Microsoft\Windows\CurrentVersion\Run\update – %SYSTEM%\rundll32.exe %APPDATA\Intel\ResN32.dll Run
This ensures that the ResN32.dll library will be run using the 'Run' exported function whenever the machine is rebooted.

The second technique is used if the victim meets any of the following sets of criteria:

- Microsoft Windows 8 / Windows Server 2012 R2
- Not running Kingsoft, Tencent, or DoctorWeb security products

- Microsoft Windows XP or lower
- No security products installed, or running any of the following:
  - Sophos
  - GData
  - TrendMicro
  - AhnLab
  - Kaspersky

In these situations, the following persistence technique is used.

HKLM\Software\Microsoft\Windows NT\CurrentVersion\Windows\AppInit_DLLs – %APPDATA%\Intel\ResN32.dllHKLM\Software\Microsoft\Windows NT\CurrentVersion\Windows\LoadAppInit_DLLs – 0x1
Setting these registry keys both enables the AppInit_DLL functionality, and ensures that every user mode process that is spawned will load the ResN32.dll library. More information about this can be

found here.

The third technique is used in any other situation. When this occurs, the malware will first identify the explorer.exe process identifier. It proceeds to inject the ResN32.dll library into this process.

At this point, the third stage of the malware family is loaded.

Stage 3

The third stage begins when the ResN32.dll file begins operating. This file contains the following debug string:

D:\WORK\T9000\ResN_M2\Release\ResN32.pdb
The ResN32.dll library begins by spawning a new thread that is responsible for the majority of the capabilities built into this sample. This thread begins by checking the operating system version, and once again runs a query on the various security products installed on the victim machine.

Under certain conditions, the following registry key is set, ensuring persistence across reboots:

HKLM\Software\Microsoft\Windows\CurrentVersion\Run\update – c:\windows\system32\rundll32.exe %APPDATA\Intel\ResN32.dll Run
Following this, a new thread is created that is responsible for deleting previously written files. This thread creates the following mutex:

Global\\deletethread
It proceeds to attempt to delete the following files in an infinite loop until said files have been deleted:

- %STARTUP%\hccutils.dll
- %STARTUP%\hccutil.dll
- %STARTUP%\igfxtray.exe

The ResN32.dll malware proceeds to read in the ResN32.dat file that was previously written to disk. This file contains a path to the hjwe.dat file, which is subsequently read in.

The data within the hjwe.dat file is decrypted using the RC4 algorithm, and subsequently decompressed using the LZMA algorithm. The following script can be used to decrypt the hjwe.dat file, along with the plugins that will be discussed later.

1
2
3
4
5
6
7
8

```
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
import sys, pylzma
from base64 import *
from binascii import *
from struct import *
def rc4( data , key ):
  S = range(256)
  j = 0
  out = []
  for i in range(256):
    j = (j + S[i] + ord( key[i % len(key)] )) % 256
    S[i] , S[j] = S[j] , S[i]
  i = j = 0
```

```
  for char in data:
    i = ( i + 1 ) % 256
    j = ( j + S[i] ) % 256
    S[i] , S[j] = S[j] , S[i]
    out.append(chr(ord(char) ^ S[(S[i] + S[j]) % 256]))
  return ''.join(out)
f = open(sys.argv[1], 'rb')
fd = f.read()
f.close()
bytes_0_4, bytes_4_8, bytes_8_12, bytes_12_16 = unpack("<IIII", fd[0:16])
if bytes_0_4 == 0xf7e4aa65:
  length = bytes_8_12
  if len(fd)-16 != length:
    print "[*] Possible error reading in length of data."
  key_size = 260
  key = fd[16:16+key_size]
  data = fd[16+key_size:]
  decrypted = rc4(data, key)
  decompressed = pylzma.decompress_compat(decrypted)
  f1 = open(sys.argv[1]+".decompressed", 'wb')
  f1.write(decompressed)
  f1.close
  print "[+] Wrote %s" % (sys.argv[1]+".decompressed")
```

After this file has been decrypted and decompressed, it is written to a file in the %TEMP% directory with a file prefix of '____RES'. This file, which contains a Windows DLL, is then loaded into the current process. After the malicious library has been loaded, the previously written temporary file is deleted. This begins the last stage of the malware, which will load the core of the malware family.

Stage 4

Once the decrypted and decompressed hjwe.dat file is loaded, it begins by checking its parent process against the following list. If the parent process matches the following blacklist, the malicious DLL will exit without performing any malicious activities.

- winlogon.exe
- csrss.exe
- logonui.exe
- ctfmon.exe
- drwtsn32.exe
- logonui.exe
- explore.exe
- System
- Dbgview.exe

- userinit.exe
- lsass.exe
- wmiprvse.exe
- services.exe
- inetinfo.exe
- avp.exe
- Rtvscan.exe

The malware proceeds to collect the username of the victim, as well as the operating system version. It then compares its parent process against the following list of executables:

- winlogon.exe
- csrss.exe
- logonui.exe
- ctfmon.exe
- drwtsn32.exe
- logonui.exe
- System
- Dwm.exe
- QQPCRTP.exe
- Tasking.exe
- Taskhost.exe
- Taskmgr.exe
- Dbgview.exe
- suerinit.exe
- lsass.exe
- wmiprvse.exe
- services.exe
- inetinfo.exe
- avp.exe
- Rtvscan.exe

Notice the repeated check for the ‘logonui.exe’, as well as the overlap with the previous parent executable check, which implies sloppiness by the malware author.

After these checks are performed, the following mutex is created.

Global\\{A59CF429-D0DD-4207-88A1-04090680F714}
The following folders are then created:

- utd_CE31
- XOLOADER
- Update

The path of these folders is determined by the version of Microsoft Windows running. The following possibilities exist:

- %ALLUSERSPROFILE%\Documents\My Document\
- %PUBLIC%\Downloads\Update\

At this point, the malware will read in the dtl.dat file, which contains configuration data. Data contained with this file starting at offset 0x20 is xor-encrypted using a single-byte key of 0x5F. The following script can be used to extract the IP address and port for the C2 server from this file.

```
from struct import *
import sys, socket
def int2ip(addr):
  return socket.inet_ntoa(pack("!I", addr))
config_file = sys.argv[1]
f = open(config_file, 'rb')
fd = f.read()
f.close()
decrypted = ""
for x in fd[32:]:
  decrypted += chr(ord(x) ^ 0x5f)
port = unpack("<I", decrypted[4:8])[0]
ip = int2ip(unpack(">I", decrypted[8:12])[0])
```

print "IP Address : %s" % ip
print "Port : %d" % port
The malware will then read in and parse the included plugin configuration information, which is found within the glp.uin file that was previously dropped. These included plugins are encrypted and compressed using the same method witnessed by the hjwe.dat file previously. The previously included script can be used to decrypt and decompress the following three plugin files:

- tyeu.dat
- vnkd.dat
- qhnj.dat

These three plugins are subsequently loaded after being decrypted and decompressed. An overview of these plugins can be found later in this post.

The malware proceeds to create the following event:

Global\\{34748A26-4EAD-4331-B039-673612E8A5FC}
Additionally, the following three mutexes are created:

Global\\{3C6FB3CA-69B1-454f-8B2F-BD157762810E}
Global\\{43EE34A9-9063-4d2c-AACD-F5C62B849089}
Global\\{A8859547-C62D-4e8b-A82D-BE1479C684C9}

The malware will spawn a new thread to handle network communication. The following event is created prior to this communication occurring:

Global\\{EED5CA6C-9958-4611-B7A7-1238F2E1B17E}
The malware includes proxy support in the event that the victim is behind a web proxy. Network traffic occurs over a binary protocol on the port specified within the configuration. Traffic is xor-encrypted with a single-byte key of 0x55 in an attempt to bypass any network security products that may be in place. Once decrypted, the following traffic is sent by the malware.

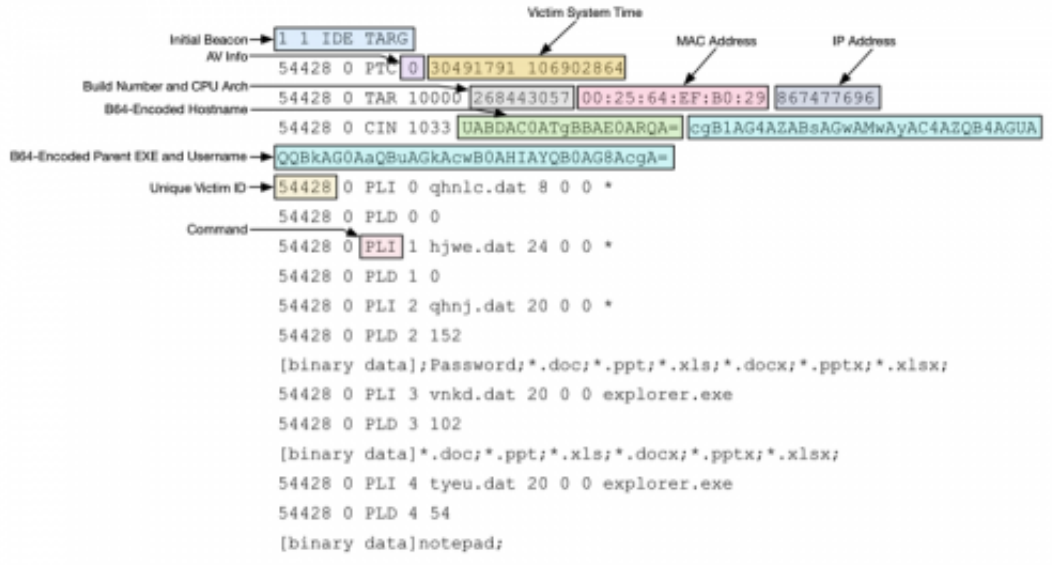


Figure 1: Decrypted data sent by malware

As we can see from the above image, the malware will send out an initial beacon, followed by various collected information from the victim machine. The following information is exfiltrated:

- Installed security products
- System time
- Build Number
- CPU Architecture (32-bit/64-bit)
- MAC Address
- IP Address
- Hostname
- Username
- Parent executable name
- Plugin configuration information

The malware is configured to receive a number of commands. The following command functionalities have been identified.

Command	**Description**
DIR	Directory listing
LIR	Drive listing
RUN	Execute command (Either interactively or not)
CIT	Send command to interactively spawned command
CFI	Kill interactively spawned process
DOW	Download file
UPL	Upload file
DEL	Delete file
DTK	Retrieve statistics for file
ERR	Null command

Additionally, the following commands have been identified, however, their functionalities have yet to be fully discovered.

- PNG
- PLI
- PLD

- FDL
- OSC
- OSF
- SDA
- QDA
- TFD
- SDS
- SCP
- FMT
- STK
- CRP

Plugin #1 – tyeu.dat

When this plugin is called with the default exported function, it will create the following mutex:

{CE2100CF-3418-4f9a-9D5D-CC7B58C5AC62}
When called with the SetCallbackInterface function export, the malicious capabilities of the plugin begin. The plugin begins by collecting the username of the running process, and determining if it is running under the SYSTEM account. If running as SYSTEM, the plugin will associate the active desktop with the plugin's thread.

The plugin proceeds to create the following named event:

Global\\{EED5CA6C-9958-4611-B7A7-1238F2E1B17E}
Multiple threads are then spawned to handle various actions. The first thread is responsible for taking a screenshot of the desktop of the victim machine. This screenshot data is both compressed and encrypted using a single-byte xor key of 0x5F. This data is written to one of the following files:

%PUBLIC%\Downloads\Update\S[random].dat
%ALLUSERSPROFILE%\Documents\My Document\S[random].dat

The random data is generated via the current system time. Additionally, when a screenshot is written, one of the following log files has data appended to it:

%PUBLIC%\Downloads\Update\Log.txt
%ALLUSERSPROFILE%\Documents\My Document\Log.txt

```
15:14:27 2000 [4] PrintFullScreen: ===>> Process ID : 2000
15:14:34 2000 [4] PrintFullScreen: ===>> Process ID : 2000
15:14:42 2000 [4] PrintFullScreen: ===>> Process ID : 2000
15:14:43 2000 [4] PrintFullScreen: ===>> Process ID : 2000
15:14:44 2000 [4] PrintFullScreen: ===>> Process ID : 2000
15:14:44 2000 [4] PrintFullScreen: ===>> Process ID : 2000
15:14:45 2000 [4] PrintFullScreen: ===>> Process ID : 2000
15:14:45 2000 [4] PrintFullScreen: ===>> Process ID : 2000
15:29:14 2000 [4] PrintFullScreen: ===>> Process ID : 2000
15:41:40 2000 [4] PrintFullScreen: ===>> Process ID : 2000
```

Figure 2: Example data found within Log.txt file

A second thread is responsible for monitoring the foreground window every 20 seconds. The thread will target the window names set within the plugin configuration. In this particular instance, the malware will target the ‘notepad’ process.

When this process is discovered to be running in the foreground window, the malware will take a screenshot of this window. The data is compressed and encrypted using a single-byte xor key of 0x5F. This data is written to one of the following files:

%PUBLIC%\Downloads\Update\W[random].dat
%ALLUSERSPROFILE%\Documents\My Document\W[random].dat

Like the previous thread, this one attempts to write another log file to the disk. However, due to a bug within the code of this plugin, the malware author attempts to append the ‘C:\\Windows\\Temp\\Log.txt’ string to the path, resulting in an inaccessible file path. In the event this bug did not exist, the following example data would be written:

08:37:49 2000 [4] PrintKeyTitleWnd: ===>> Process ID : 2000
The third and final thread spawned by this plugin is responsible for collecting information from the Skype program. The malware will use the built-in Skype API to accomplish this. This only takes places if both Skype is running and the victim is logged into Skype. It makes calls to the following functions:

- SkypeControlAPIDiscover
- SkypeControlAPIAttach

When hooking into the Skype API, the victim is presented with the following dialog:

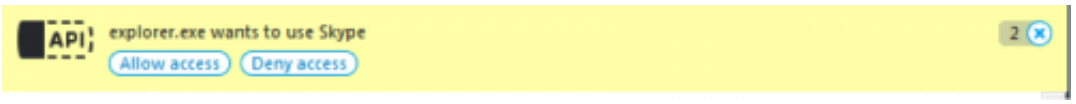
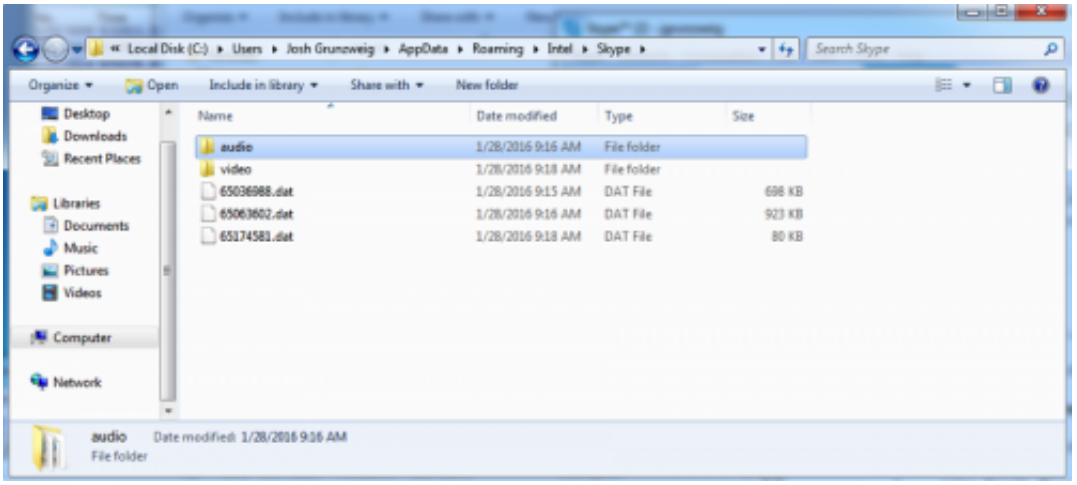


Figure 3: Skype API access request

The victim must explicitly allow the malware to access Skype for this particular functionality to work. However, since a legitimate process is requesting access, the user may find him- or herself allowing this access without realizing what is actually happening.

Once enabled, the malware will record video calls, audio calls, and chat messages. Audio and video files are stored in the following folder:

%APPDATA%\Intel\Skype
Temporary audio and video files are stored within the audio and video sub-folders respectively. After a call is finished, this data is compressed and encrypted using the same techniques previously witnessed. These files are stored in randomly named .dat files within the Skype folder.



When decrypted, we can see that the malware periodically takes images of the video calls. Audio calls are stored as .wav files.

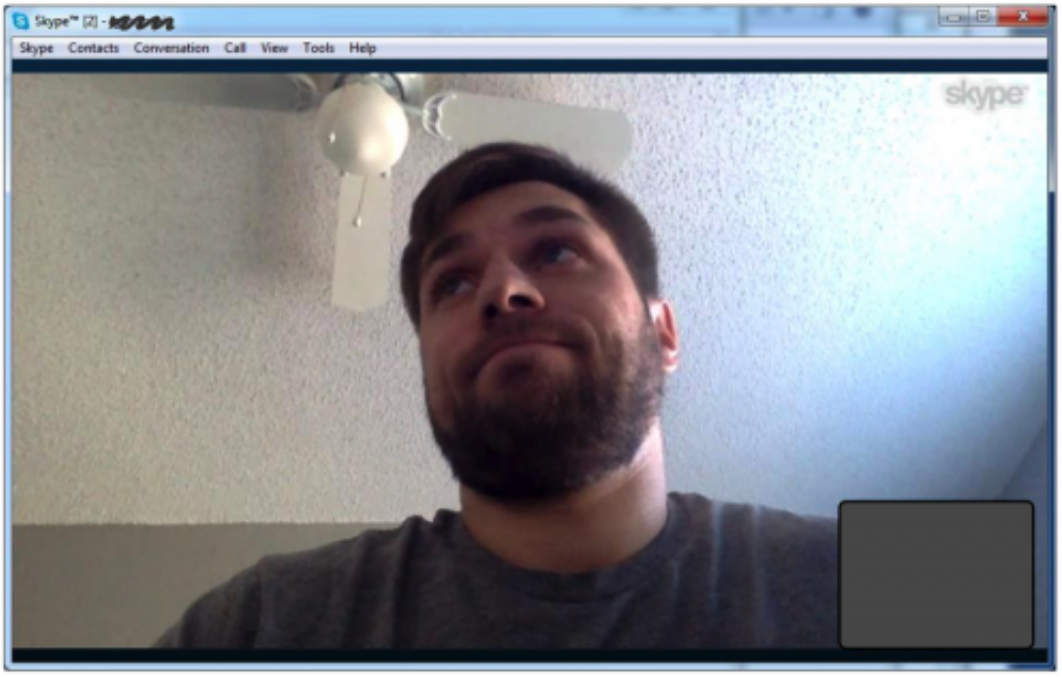


Figure 4: A lonely malware reverser is captured on video by the malicious plugin

The original name for this plugin is ‘CaptureDLL.dll’. This is aptly named, as we see that this plugin has the following functionality:

- Capture full desktop screenshots
- Capture window screenshots of targeted processes
- Capture Skype audio, video, and chat messages

Plugin #2 – vnkd.dat

The vnkd.dat plugin has the following debug path, leading us to believe that the original name for this plugin is ‘FlashDiskThief’:

e:\WORK\Project\T9000\Windows\Target\FlashDiskThief.pdb
When loaded with the default DllEntryPoint exported function, it will create the following mutex:

Global\\{6BB1120C-16E9-4c91-96D5-04B42D1611B4}

Like the other plugins associated with this malware, the majority of the functionality for this malware resides within the SetCallbackInterface exported function. This function spawns a new thread that begins by registering a new window with a class name and window name of 'xx'.

The plugin proceeds to iterate through all connected drives on the system, looking for removable drives.

```
.text:10005ABC loc_10005ABC:                           ; CODE XREF: iterating_through_drives+41'j
.text:10005ABC                 cmp     [ebp+var_1C], 5Ah
.text:10005AC0                 jge     short loc_10005AED
.text:10005AC2                 mov     cx, word ptr [ebp+var_1C]
.text:10005AC6                 mov     [ebp+RootPathName], cx
.text:10005ACA                 mov     esi, esp
.text:10005ACC                 lea     edx, [ebp+RootPathName]
.text:10005ACF                 push    edx             ; lpRootPathName
.text:10005AD0                 call    ds:GetDriveTypeW
.text:10005AD6                 cmp     esi, esp
.text:10005AD8                 call    __RTC_CheckEsp
.text:10005ADD                 cmp     eax, DRIVE_REMOVABLE
.text:10005AE0                 jnz     short loc_10005AEB
.text:10005AE2                 lea     eax, [ebp+RootPathName]
```

Figure 5. Plugin check for removable drives

Should a removable drive be discovered, the plugin will seek any files residing on this device based on the plugin's configured list. In this particular instance, the malware will seek out the following file types:

- *.doc
- *.ppt
- *.xls
- *.docx
- *.pptx
- *.xlsx

If one of these file types is found, the malware will create a copy of the file in one of the following paths:

%PUBLIC%\Downloads\Update\D[random].tmp

%ALLUSERSPROFILE%\Documents\My Document\D[random].tmp

The data found within this file is encrypted using a single-byte xor key of 0x41. The file header structure, with the underlying data still encrypted, can be seen below.

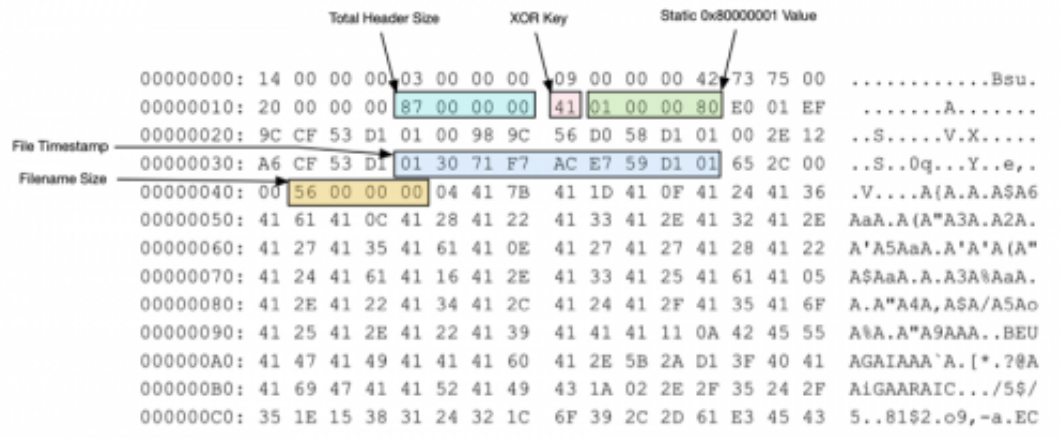


Figure 6: File structure prior to decryption

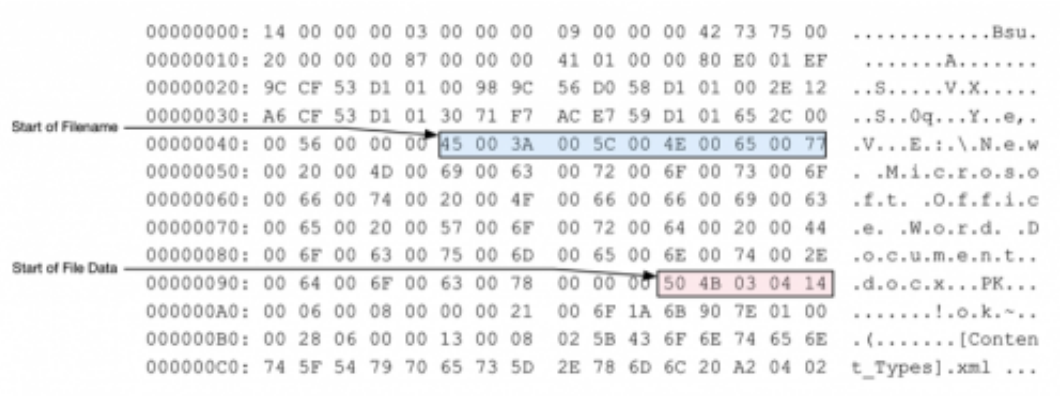


Figure 7: File structure post decryption

This concludes the functionality of the vnkd.dat plugin, or FlaskDiskThief as it's known by the malware's author. While specific in nature, this plugin allows attackers to collect files being passed around from one machine to another via removable drives.

Plugin #3 – qhnj.dat

This particular plugin appears to have an original filename of 'kplugin.dll' due to debugging information found within the file. The qhnj.dat plugin is responsible for hooking a number of common Microsoft Windows API calls, and logging the results.

The following functions are hooked by this plugin:

- ImmGetCompositionStringA
- ImmGetCompositionStringW
- CreateFileW
- DeleteFileW
- CopyFileExW
- MoveFileWithProgressW
- CreateDirectoryW
- CreateDirectoryExW
- RemoveDirectoryW
- GetClipboardData
- CryptEncrypt
- CryptDecrypt

The plugin is most likely hooking the ImmGetCompositionString* functions in order to collect information about Unicode characters on the victim machine, such as Chinese, Japanese, and Korean.

Hooking the various file and directory operations allows the malware to log what file changes are occurring on the system. When a file is created, copied, moved, or deleted on the system, the malware will check the directory of said file against the following blacklist:

- \\\\.\\
- :\\program files\\
- \\AppData\\
- \\temporary internet files\\
- \\application data\\
- \\Local Settings\\
- \\cookies\\
- \\temp\\
- \\history\\

Additionally, the filename is compared against the '.tmp' extension to ensure a temporary file is ignored.

Should the file meet the required criteria, this data is logged. Additionally, all folder modifications and clipboard data are logged as well.

The Crypt* functions allow the malware to collect sensitive encrypted data sent to and from the victim machine. This is especially useful when viewing network traffic, allowing the attackers to potentially gain access to remote systems used by the victim.

All of the data logged by the qhnj.dat plugin file is stored in one of the following file paths. Data is encrypted using a single-byte XOR key of 0x79.

%PUBLIC%\Downloads\Update\uai[random].tmp
%ALLUSERSPROFILE%\Documents\My Document\uai[random].tmp

This last plugin allows the attackers to record important actions taken by the victim, which in turn may allow them to gain additional access as well as insight into the victim's actions.

Conclusion

T9000 appears to be the latest version of this Trojan, which has been partially exposed in previous reports. In 2013, Cylance published a report on a group they named "Grand Theft Auto Panda", which includes some details on the T5000 version of this Trojan. FireEye researchers also noted that the malware was used in an attack in 2014 using a lure related to the disappearance of Malaysian flight MH370.

The author of this backdoor has gone to great lengths to avoid being detected and to evade the scrutiny of the malware analysis community. We hope that sharing the details of how this tool works as well as the indicators in the section below will help others defend themselves against attacks using this tool.

In a future report, we will detail the infrastructure used by the variants of the malware we have identified and discuss the methods attackers use to infect systems with it.

Palo Alto Networks customers are protected from T9000/T5000 attacks through our next-generation security platform, including the following.

- **Threat Prevention** signatures for the software vulnerabilities listed in this report are available to detect the exploit files during delivery.
- **Traps** is capable of preventing exploitation of the vulnerabilities exploited to install T9000.
- **WildFire** classifies all of the malware described in this report as malicious.
- **Anti-malware signatures** for the files listed in this report.
- **AutoFocus** users can identify the malware discussed in this report with the T5000 tag

Indicators of Compromise

Hashes

RTF File, d5fa43be20aa94baf1737289c5034e2235f1393890fb6f4e8d4104565be52d8c
QQMGr.dll, bf1b00b7430899d33795ef3405142e880ef8dcbda8aab0b19d80875a14ed852f
QQMGR.inf, ace7e3535f2f1fe32e693920a9f411eea21682c87a8e6661d3b67330cd221a2a
ResN32.dat, aa28db689f73d77babd1c763c53b3e63950f6a15b7c1a974c7481a216dda9afd
ResN32.dll, 1cea4e49bd785378d8beb863bb8eb662042dffd18c85b8c14c74a0367071d9a7
hqwe.dat, bb73261072d2ef220b8f87c6bb7488ad2da736790898d61f33a5fb7747abf48b
hqwe.dat.decrypted, 7daf3c3dbecb60bee3d5eb3320b20f2648cf26bd9203564ce162c97dcb132569
hccutils.dll, 3dfc94605daf51ebd7bbccbb3a9049999f8d555db0999a6a7e6265a7e458cab9
hccutils.inf, f05cd0353817bf6c2cab396181464c31c352d6dea07e2d688def261dd6542b27
igfxtray.exe, 21a5818822a0b2d52a068d1e3339ed4c767f4d83b081bf17b837e9b6e112ee61
qhnj.dat, c61dbc7b51caab1d0353cbba9a8f51f65ef167459277c1c16f15eb6c7025cfe3
qhnj.dat.decrypted, 2b973adbb2addf62cf36cef9975cb0193a7ff0b960e2cff2c80560126bee6f37
tyeu.dat, e52b5ed63719a2798314a9c49c42c0ed4eb22a1ac4a2ad30e8bfc899edcea926
tyeu.dat.decrypted, 5fc3dc25276b01d6cb2fb821b83aa596f1d64ae8430c5576b953e3220a01d9aa
vnkd.dat, c22b40db7f9f8ebdbde4e5fc3a44e15449f75c40830c88932f9abd541cc78465
vnkd.dat.decrypted, 157e0a9323eaaa911b3847d64ca0d08be8cd26b2573687be461627e410cb1b3f
dtl.dat, 00add5c817f89b9ec490885be39398f878fa64a5c3564eaca679226cf73d929e
glp.uin, 3fa05f2f73a0c44a5f51f28319c4dc5b8198fb25e1cfcbea5327c9f1b3a871d4

Mutexes

820C90CxxA1B084495866C6D95B2595xx1C3
Global\\deletethread
Global\\{A59CF429-D0DD-4207-88A1-04090680F714}
Global\\{3C6FB3CA-69B1-454f-8B2F-BD157762810E}
Global\\{43EE34A9-9063-4d2c-AACD-F5C62B849089}
Global\\{A8859547-C62D-4e8b-A82D-BE1479C684C9}
{CE2100CF-3418-4f9a-9D5D-CC7B58C5AC62}
Global\\{6BB1120C-16E9-4c91-96D5-04B42D1611B4}

Named Events

Global\\{34748A26-4EAD-4331-B039-673612E8A5FC}

Global\\{EED5CA6C-9958-4611-B7A7-1238F2E1B17E}

File Modifications

%TEMP%\~tmp.doc
%APPDATA%\Intel\avinfo
%APPDATA%\Intel\Data\dtl.dat
%APPDATA%\Intel\Data\glp.uin
%APPDATA%\Intel\Data\
%APPDATA%\Intel\~1
%APPDATA%\Intel\hccutils.dll
%APPDATA%\Intel\hccutils.inf
%APPDATA%\Intel\hjwe.dat
%APPDATA%\Intel\igfxtray.exe
%APPDATA%\Intel\qhnj.dat
%APPDATA%\Intel\QQMgr.dll
%APPDATA%\Intel\QQMgr.inf
%APPDATA%\Intel\ResN32.dll
%APPDATA%\Intel\ResN32.dat
%APPDATA%\Intel\tyeu.dat
%APPDATA%\Intel\vnkd.dat
%STARTUP%\hccutils.dll
%STARTUP%\hccutil.dll
%STARTUP%\igfxtray.exe
%ALLUSERSPROFILE%\Documents\My Document\utd_CE31
%ALLUSERSPROFILE%\Documents\My Document\XOLOADER
%ALLUSERSPROFILE%\Documents\My Document\update
%ALLUSERSPROFILE%\Documents\My Document\Log.txt
%PUBLIC%\Downloads\Update\utd_CE31
%PUBLIC%\Downloads\Update\XOLOADER
%PUBLIC%\Downloads\Update\update
%PUBLIC%\Downloads\Update\Log.txt
%APPDATA%\Intel\Skype

Registry Modifications

HKLM\Software\Microsoft\Windows\CurrentVersion\Run\Eupdate – %APPDATA%\Intel\ResN32.dll
HKLM\Software\Microsoft\Windows\CurrentVersion\Run\update – %SYSTEM%\rundll32.exe
%APPDATA\Intel\ResN32.dll Run
HKLM\Software\Microsoft\Windows NT\CurrentVersion\Windows\AppInit_DLLs –
%APPDATA%\Intel\ResN32.dll
HKLM\Software\Microsoft\Windows NT\CurrentVersion\Windows\LoadAppInit_DLLs – 0x1
HKLM\Software\Microsoft\Windows\CurrentVersion\Run\update – c:\windows\system32\rundll32.exe
%APPDATA\Intel\ResN32.dll Run

Command and Control

198.55.120[.]143:8080