A Detailed Analysis of an Advanced Persistent Threat Malware

Frankie Fu Kay Li
A Detailed Analysis of an Advanced Persistent Threat Malware

GIAC (GREM) Gold Certification

Author: Frankie Li, ran2@vxrl.org
Advisor: Antonios Atlasis

Accepted: October 13, 2011

Abstract
Spear-phishing emails were sent to a political figure at my place of residence. An email together with the attached sample was provided for forensics analysis. It appears to be an Advanced Persistent Threat type malware. By performing behavioral and code analysis in an alternatively way, most of its important functions were identified. The aim of this technical paper is to illustrate the detailed procedures of how this malware was dissected.
1. Introduction

Spear-phishing emails were sent to a political figure at my place of residence. An email, including the attached sample was provided for forensics analysis. This email contained obviously well crafted message to lure the recipient to open the malicious attachment. It was predicted as an Advanced Persistent Threat attack (APT-attack).

By performing the detailed behavioral and code analysis in a Spiral way (Brand, Valli & Woodward, 2010, p 6), most of the important functions of the malware were identified. The aim of this technical paper is to demonstrate the step-by-step procedures on how this malware was dissected.

Advance Persistent Threat (APT) is a hot and controversial term used amongst security professionals, including Bejtlich (2011), Cloppert (2010), Lee (2011) and Hoglund (2011), especially after McAfee published their white paper called: “Revealed: Operation Shady RAT” (Alperovitch, 2011). Kaspersky criticized this report and flagged “the report as alarmist due to its deliberately spreading misrepresented information” (Frye, 2011).

The term APT is frequently used as a replacement term to describe cyber warfare between countries (Cloppert, 2009). Albert Hui, an IT professional in Hong Kong viewed it as “an entire threat class (analogous to other threat classes like insider fraud, industrial espionage, and hactivism), whereas things like RATs, drive-by malwares, rootkits, DDoS are threat vehicles” (Hui, 2011). Referring to the traditional meaning of APT, the malware is only one of the threat components of APT. However, most anti-virus vendors, including McAfee and Kaspersky continue to fame APT as a malware. One side of them even treats it as a “bot” or “back door”, which contains sophisticated techniques that persistently hiding itself in the victim’s system for collection of intelligence. However, comparing with other threat tools, because malware is easier to prepare and deploy, the term APT will be continuously be described as a malware. A fine-tuned definition proclaimed by Mandiant (2011, “What is M-Trends?”) is accepted. In this technical paper, APT is considered as a cyber attack launched by a group of sophisticated, determined and coordinated attackers that have been systematically

Frankie Li, ran2@vxrl.org
compromising a specific target’s machine or entity’s networks for prolonged period. The meaning of “Persistence” is also expanded to the acts of the attackers of persistently launching spear-phishing attacks against the targets. The findings indicate the main functions of the APT-type malware are usually placed heavily on spying instead of for the purpose of financial gain.

Other than the purpose of collecting of national secrets or political espionage, based on the functions discovered, it is believed that this threat can also apply to the cases in business or industrial espionages, spying acts or even un-ethical detective investigations.

2. The Behavioral and Code Analysis

2.1. Setting Up the Lab Environment

The analysis was conducted inside a host-only network VMware machines.

a) The infection box was installed with a Windows XP SP2 system. The IP address was configured as 192.168.80.125 with default gateway and DNS pointed to 192.168.80.130 (i.e. IP address of the responsive box described below).

b) A lightweight Ubuntu distribution, REMnux 2.0 was used for interacting with the malware. It was configured with IP address of 192.168.80.130 as a responsive box.

The analysis methodology proposed by Zeltser (2007) was adopted. It is a way of molding of analysis environment that alternately uses behavioral and code analysis techniques to identify the functionality of the executable (Valli & Brand, 2008, p.2).

The table shows the tools that were used to perform the behavioral and code analysis (Table 1).
<table>
<thead>
<tr>
<th>Analysis Technique</th>
<th>Tool name</th>
<th>Reference</th>
<th>Functional usage of the tool</th>
</tr>
</thead>
<tbody>
<tr>
<td>Behavioral</td>
<td>ListDLLs (v3.1)</td>
<td>Russinovich, M. (2011d)</td>
<td>Display DLLs loaded on the system.</td>
</tr>
<tr>
<td>Behavioral</td>
<td>Winobj (v.2.21)</td>
<td>Russinovich, M. (2011g)</td>
<td>Display Windows’ Object Manager namespace.</td>
</tr>
<tr>
<td>Behavioral</td>
<td>HandleDiff (v.0.2)</td>
<td>Ligh, M., Adair, S., Hartstein, B., &amp; Richard, M. (2011b)</td>
<td>Detect changes to the handle tables of process.</td>
</tr>
<tr>
<td>Code</td>
<td>FileInsight (v2.1)</td>
<td>McAfee (2009)</td>
<td>A file reader that displays the document in either textual or hexadecimal format.</td>
</tr>
<tr>
<td>Code</td>
<td>PEiD (v0.94)</td>
<td>Snaker, Qwerton &amp; Jibz (2008)</td>
<td>PE File identification tool.</td>
</tr>
</tbody>
</table>

*Table 1.* Tools used to perform behavioral and code analysis
2.2. The Observed Behavior

The sample was contained inside a well-crafted phishing email, which described the news of a riot incident happened at a county near the Guangzhou City in China. The malicious attachment named: “XinTang Event.rar”. It was not password protected and after decompressed, a file called “XinTang Event.chm” was created. If double-clicked, the following message was displayed (Figure 1).

![Figure 1. The displayed chm help file if double clicked](image)

To clear up the possible infection and assuming this malware did not exploit any known or 0-day VMware vulnerability, the snapshot of the infection box was rolled-back and uploaded the decompressed chm file to the responsive box for checking with the “file” utility. It was identified as a “MS Windows HtmlHelp Data file”. After Googling, an instruction on how to create Trojan horse using chm file was found from a Chinese website (Chen Yi-Tian, 2005)

The chm file was decompiled by using a tool call “Malcode Analysis Pack”. A folder (Figure 2) called “chm_src” was generated containing some jpeg files and an executable called “dg003_improve_8080_V132.exe” (hereafter also called as dg003.exe).

Frankie Li, ran2@vxrl.org
The file “dg003.exe” was checked with PEiD v0.94 and found UPX-packed (Figure 3). After unpacked using the UPX utility, the file size of “dg003.exe” was expanded from 84,992 bytes to 196,608 bytes generating hash value of 4EC0027BEF4D7E1786A04D021FA8A67F.

Various monitoring tools, including: Autoruns, TCPView, Process Explorer, Regshot, Process Monitor, CaputreBAT and HandleDiff were activated in an order to capture maximum information during execution of the malicious “dg003.exe”.

The CaptureBAT log was imported into Microsoft Excel for quick review (Li, 2011). Using the smart-filtering feature, all modifications of the registry entries and file system were identified (Figure 4).
CaptureBAT log was first checked because of its distinctive advantages in behavioral analysis. It can (a) identify which process is responsible for the file or registry modifications; (b) backup the modified or deleted files into a folder during execution; and (c) create a .pcap file to log the network activities during the capture. However, if files were in accessed during the capture, CaptureBAT would miss backing up of these files. As a supplement, Regshot was also activated to capture similar changes.

The file system and registry changes are summarized as below (Table 2).

<table>
<thead>
<tr>
<th>Tool</th>
<th>Process</th>
<th>Location</th>
<th>Event</th>
<th>Event Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CaptureBAT</td>
<td>dg003.exe</td>
<td>C:\Documents and Settings&lt;user&gt;\Local Settings\Application\Data</td>
<td>File added</td>
<td>ws2help.PNF</td>
</tr>
<tr>
<td>CaptureBAT</td>
<td>dg003.exe</td>
<td>C:\Documents and Settings&lt;user&gt;\Local Settings\Application\Data</td>
<td>File added</td>
<td>msvcr.dll</td>
</tr>
<tr>
<td>CaptureBAT</td>
<td>dg003.exe</td>
<td>C:\Windows\system32</td>
<td>File altered</td>
<td>netstat.exe</td>
</tr>
<tr>
<td>CaptureBAT</td>
<td>dg003.exe</td>
<td>C:\Windows\inf</td>
<td>File added &amp; removed</td>
<td>1.txt</td>
</tr>
<tr>
<td>CaptureBAT</td>
<td>explorer.exe</td>
<td>C:\Documents and Settings&lt;user&gt;\Local Settings\Application\Data</td>
<td>File added</td>
<td>IECheck.exe</td>
</tr>
<tr>
<td>CaptureBAT</td>
<td>explorer.exe</td>
<td>C:\Windows\system32</td>
<td>File added</td>
<td>Internet Explorer Security Check.Ink</td>
</tr>
<tr>
<td>Regshot</td>
<td></td>
<td>C:\Windows\system32</td>
<td>File added</td>
<td>13605</td>
</tr>
<tr>
<td>HandleDiff</td>
<td>explorer.exe</td>
<td>\BaseNamedObjects</td>
<td>Mutant added</td>
<td>VistaDLLPro RUNNING</td>
</tr>
</tbody>
</table>

Table 2. Changes on file system and registry
Using the tool – Vmmmap and ListDLLs, the malicious dynamic-link library (DLL) - “msvcr.dll” was found as an injected thread in the running process of “explorer.exe” at the base address of 0x10000000 with image size of 0x43000 (Figure 5).

![Figure 5](image.png)

*Figure 5. Injected DLL was found by using Vmmmap*

Checking the Thread information of the injected process - “explorer.exe” with Process Explorer, three additional threads of “msvcr.dll” were discovered and they were started at relative virtual address (RVA) of 0xAEE0, 0xABB0 and 0xA940 (Figure 6).

Wireshark was used to inspect a file named “192.168.80.125.pcap”, which was generated by CaptureBAT. The infection box tried to resolve the non-existent hostnames of “test.3322.org.cn”, “1.test.3322.org.cn”, “2.test.3322.org.cn”, “3.test.3322.org.cn” and “4.test.3322.org.cn”. Immediate after the unsuccessful retrials, it tried to connect to IP addresses 172.16.0.61 and 115.x.x.249 by using TCP port number 8080.
Figure 6. Three additional threads was indentied by using Process Explorer

In order to allow “dg003.exe” to establish connections with these IP addresses inside a controlled virtual environment, the honeyd and farpd services on the Remnux responsive box were activated. The farpd was configured to reply to any ARP request for an IP address matching with the MAC address of the Remnux network adaptor and the honeyd was configured to run http service at TCP port 8080 on the Remnux responsive box (Appendix I). After these services started, some encrypted http communicated was captured and the detail discussion of this communication can be found at section 3.2 of this paper.

3. Technical details discovered by Code Analysis

The malware was identified as a kind of “multi-staged” malware (i.e. a malware using form of multipartite infection strategy) (Szor, 2005, p. 76). During the analysis, which will be described below, the malware or the file “dg003.exe” was found as a dropper, which dropped the file “msvcr.dll” as the droppee. If the Internet access is enabled, the injected “msvcr.dll” will download some more Trojan-Spies.

Frankie Li, ran2@vxrll.org
IDA Pro was used for disassembling the binaries. The graphical view was used along with the “Chart of X-ref to” and “Chart of X-ref from” functions to identify the program flows between each important subroutine.

If possible, every subroutine was dynamically studied by stepping through under a debugger, OllyDbg. Whenever a key function was identified, the subroutine was renamed with a meaningful name both in OllyDbg and IDA Pro. If variables or pass through arguments were found during debugging with OllyDbg, appropriate comments were added to OllyDbg and IDA Pro. By switching back and forth, pseudo-code flowcharts for all investigating binary could be generated.

3.1. The Code Analysis of “dg003.exe”

For quick identification, the file “dg003.exe” was uploaded to VirusTotal. The report indicated only 15 out of 44 anti-virus engines could detect it as malicious, but not a single one could clearly identify its virus family or signature.

The following diagram shows a board overview of the order of executions and key functions of the file “dg003.exe” (Figure 7).

![Figure 7. The key functions of “dg003.exe”](image)

The file “dg003.exe” starts by checking if it is called from a command prompt with passing argument of “Update” or “Special” (Figure 8). These parameters were passed to OllyDbg as argument during different debugging sessions. However, no obvious function was triggered.

Frankie Li, ran2@vxrl.org
Argument checking was found under IDA Pro

Then, it checks if the victim is installed with some anti-virus programs of "Kaspersky", "ESET", "BitDefender", "AVG", "NOD32", "Rising" or "360+" by enumerating the registry key at "SOFTWARE\Microsoft\windows\CurrentVersion\Uninstall"

After checking with the passing arguments, “dg003.exe” tries to create a mutex named “VistaDLLPro RUNNING” (Figure 9) to prevent double installation of itself on the system. This malware uses the similar method like, ZeuS bot to mark its presence on the system (Ligh, Adair, Hartstein & Richard, 2011a. p.301).

Figure 9. CreateMutex was found under IDA Pro

Frankie Li, ran2@vxrl.org
Then, “dg003.exe” writes a duplicate as C:\Documents and Settings\<user>\Local Settings\Application Data\w2help.PNF at 0x00403DA2 (Figure 10).

![Figure 10. The file “w2help.PNF” was written](image)

Then it calls into 0x00402BB9 to append string of “C:\Documents and Settings\<user>\Local Settings\Application Data\msvcr.dll” at the end of the file “w2help.PNF” to prevent detection of checksum based detection. At 0x00402BC9, “dg003.exe” creates a DLL in the memory from the code stored at the resource section with ID node name “VISTADLL” (Figure 11).

![Figure 11. ID node “VISTALDLL” was found under IDA Pro](image)

At 0x0040367F, it copies 0x21000 (135,158) bytes from the resource section to the memory address at 0x0040F1F0. After executing the decoding routine at 0x0043691 to 0x004036AB, a DLL is decoded at memory location of 0x0040F1F0 (Figure 12).

Frankie Li, ran2@vxrl.org
Subsequent the decoding, the DLL is packed with a proprietary packing stub in the memory and “dg003.exe” writes the packed DLL in name of “msvcr.dll” at 0x00402C71 (Figure 13).

Figure 12. The Decoding Routine

Figure 13. The “write_pack_dll” function was found under IDA Pro

Frankie Li, ran2@vxrl.org
After some clean ups, “dg003.exe” changes the MAC time (i.e. the Modification time, Access time and Change time) of the newly created file “msvcr.dll” at 0x00402D15 and hides “msvcr.dll” as system file at 0x00402D41. At 0x004017A2, “dg003.exe” copies two Windows system files of “netstat.exe” to C:\Windows\System32\13605 and “SFC_OS.dll” to C:\Windows\inf\1.txt. The “SFC_OS.dll” is the executable portion of Windows File Protection mechanism (WFP), which protects system files from being modified or deleted. The malware calls to ordinal 5 function of “1.txt” in order to bypass the WFP (Collake, 2006) during patching "netstat.exe" for hiding network connection of IP address of 115.x.x.249 (Figure 14).

![Image](image.png)

**Figure 14. Copy and patch “SFC_OS.dll”**

During the behavioral analysis, it was found that “dg003.exe” would inject “msvcr.dll” into the running “explorer.exe” process and because the process is the GUI shell of Windows system, simply attaching this process to OllyDbg will freeze the system. To execute two copies of “explorer.exe”, a Dword of DesktopProcess=1 was added to the registry of HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer. Then, the second copy of C:\WINDOWS\explorer.exe was attached under OllyDbg (hereafter it is called as debugging “explorer.exe”) (Figure 15).

Frankie Li, ran2@vxrl.org
To continue the debugging process, the file “dg003.exe” was patched for injecting the “msvcr.dll” into the debugging “explorer.exe”. A break point was set at 0x00403FBF for monitoring the scan of each running processes’ PID at 0x0012EBE4. After stopping at this break point a few times, the process name of “explorer.exe” was displayed at the ASCII pane near the dumped address at 0x0012EBE4. The PID value of the system “explorer.exe” was replaced with the PID value of the debugging “explorer.exe”. As an example, the system running “explorer.exe” PID of 864 (0x360) was patched with the debugging “explorer.exe” PID of 3768 (0xEB8). Because the memory address is represented in little endian under Intel CPU, to redirect the injection to the debugging “explorer.exe”, the value of “60 03” was replaced with “B8 0E”. (Figure 16)
After the patching, the malware injects “msvcr.dll” to debugging “explorer.exe” by calling CreateRemoteThread API at 0x00404094. It also generates a debugger message of “🗑️️ ₿️️ 🌐” (Literally “Injection Success!”) and then terminates itself.

3.2. Analysis of The Injected “msvcr.dll”

Because the “msvcr.dll” is packed by a proprietary packing routine, it has to be manually un-packed before further code analysis. The file “msvcr.dll” is loaded into OllyDbg and a hardware break point is set at 4 bytes about the ESP (Figure 17).

Figure 16. Patching with the PID value to inject into the 2nd “explorer.exe”

Figure 17. Setting Hardware Break Point at ESP-4

Frankie Li, ran2@vxrl.org
After pressing F9 to let it run, it stops at an instruction at a long jump to the address contains in EAX. (Figure 18)

![CPU - main thread, module msvcr]

**Figure 18.** EAX shows the possible Original Entry Point

Pressing F7 to step into EAX at 0x100166CA, it looks like “msvcr.dll” is unpacked. The OllyDump is called to create an unpacked version of “msvcr.dll”, which is loaded into IDA Pro for static code analysis.

Recalling the behavioral findings after the DLL injection (Figure 6), there are three running threads of “msvcr.dll”, which are set to start at RVAs of 0xAEE0, 0xABB0 and 0xA940. After checking with the “Function View” of IDA Pro by adding base address of 0x10000000 to these RVAs, three functions at 0x10009F40, 0x1000ABB0 and 0x1000AEE0 are identified. It is clearly a good choice for setting break points at OllyDbg on the debugging “explorer.exe” (Figure 19).

![IDA - C:Documents and Settings\Administrator\Desktop\Giangy\msvcrt.dll (dp005_dumped.exe) [functions window]]

**Figure 19.** The three entry functions identified under IDA Pro

The following diagram shows a board overview of the order of execution and key functions of the injected “msvcr.dll” (Figure 20). A description of the performed analysis that provided these results will follow.
Figure 20. The key functions of the injected “msvcr.dll”

At address 0x1000AF12, the injected “msvcr.dll” creates a mutex “VistaDLLPro IS RUNNING” and generates a debugging message of “MainThread Start!” at 0x1000AF37. Then it generates a unique machine ID by referring to the network adaptor information and the hard disk volume serial number at 0x1000B013 (Figure 21).

Figure 21. The hard disk volume serial number was used for generating machine ID

Frankie Li, ran2@vxrll.org
A string of “STAPPro” and “dg003” is added at the end of this machine ID, it is encoded at 0x1000B0F1 and 0x1000B16D (Figure 22). A newly created file is created at C:\WINDOWS\system32\ipsecstap.dat.

Figure 22. Encoding routine was found under IDA Pro

Then the injected “msvcr.dll” makes an API call, GetPrivateProfileString at 0x1000BD8D to retrieve strings from a non-exist file at C:\Windows\msip.ini. At 0x1000B6DA, it creates an “Assiant Thread” (The misspelled name which was found at 0x1000AB89 is used) to execute a function at 0x1000A940 (Figure 23). This thread will create a mutex of “VistaDLLPro Want Wood To Exit?” and runs a loop to wait for other instructions.

Figure 23. An assistant thread was created


Frankie Li, ran2@vxrl.org
tries to connect to a non-routable IP address of 172.16.0.61. It is believed that these are un-removed junk codes that have been previously used for program testing.

Because the injected “msvcr.dll” thread is analyzed inside a controlled environment, it runs into a loop and waits for further instructions from the C&C.

To trigger additional response, the honeyd and farpd services on the Remnux responsive box are turned on to handle the network request. Under the same behavioral studies, the injected “msvcr.dll” starts connecting to the IP address of 115.x.x.249 with TCP port number 8080. If the socket is created, it sends out some encrypted network traffic (Figure 24).

![Figure 24. Encrypted traffic captured by Remnux responsive box](image)

### 3.3. Behavioral and Code Analysis with Internet Access

In order to gather more responses from the actual C&C, Internet access was enabled and another session of behavioral analysis was performed.

After connected with the C&C, the injected “msvcr.dll” jumps into the function at 0x1000BB10 to 0x10001E9A. It calls various APIs including: GetVersionExA, GetComputerNameA, GetUserNameA, GetLogicalDrives, GetDriveTypeA, GetDiskFreeSpaceExA and GetACP to collect some hard disk information from the victim’s system. Before calling the encryption function, the information is temporary kept in memory (Figure 25). It also enumerates registry key of “System\CurrentControlSet\Control\ProductOption” and “Hardware\Descriptions\System\CentralProcessor\0” to gather more machine information from the infected machine.

Frankie Li, ran2@vxrl.org
Figure 25. Gathering hard disk information under IDA Pro

The information is encrypted at 0x1000FB10. The encryption algorithm block is identified at 0x1000FB2A to 0x1000FB54.

Figure 26. The encryption algorithm was found under IDA Pro
Based on this encryption algorithm, a script (Appendix II) is written for decryption part of the communication (Figure 27). It is found that the injected “msvcr.dll” first sends standard HTTP request together with the machine ID and receives a standard HTTP response from the connected C&C. Then it sends all collected information through encryption HTTP traffic to the C&C.

![Image of decrypted communication](image)

At the same moment, another session of behavioral analysis is performed and the changes on the file system and registry changes are summarized below (Table 3).

<table>
<thead>
<tr>
<th>Tool</th>
<th>Process</th>
<th>Location</th>
<th>Event</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CaptureBAT</td>
<td>explorer.exe</td>
<td>C:\WINDOWS\Debug\</td>
<td>File added</td>
<td>fvcwin32.exe</td>
</tr>
<tr>
<td>CaptureBAT</td>
<td>explorer.exe</td>
<td>C:\WINDOWS\Debug\</td>
<td>File added</td>
<td>avcwin32.exe</td>
</tr>
<tr>
<td>CaptureBAT</td>
<td>explorer.exe</td>
<td>C:\WINDOWS\Debug\</td>
<td>File added</td>
<td>avcwin32.exe</td>
</tr>
<tr>
<td>CaptureBAT</td>
<td>fvcwin32.exe</td>
<td>C:\WINDOWS\Debug\Data</td>
<td>File</td>
<td>drive</td>
</tr>
<tr>
<td>CaptureBAT</td>
<td>avcwin32.exe</td>
<td>C:\WINDOWS\Debug\Data</td>
<td>File added</td>
<td>20110704145735.bmp</td>
</tr>
<tr>
<td>CaptureBAT</td>
<td>avcwin32.exe</td>
<td>C:\Documents and Settings&lt;user&gt;\Application Data</td>
<td>File added</td>
<td>自动列表.txt (Lite. “AutoList.txt”)</td>
</tr>
<tr>
<td>CaptureBAT</td>
<td>avcwin32.exe</td>
<td>C:\WINDOWS\Debug\Data</td>
<td>File added</td>
<td>SAM.dll</td>
</tr>
<tr>
<td>CaptureBAT</td>
<td>avcwin32.exe</td>
<td>C:\WINDOWS\Debug\Data</td>
<td>File added</td>
<td>system.dll</td>
</tr>
<tr>
<td>CaptureBAT</td>
<td>explorer.exe</td>
<td>C:\WINDOWS\Debug\Data</td>
<td>File added</td>
<td>drive</td>
</tr>
<tr>
<td>CaptureBAT</td>
<td>explorer.exe</td>
<td>C:\WINDOWS\Debug\Data</td>
<td>File added</td>
<td>iestorage.dll</td>
</tr>
<tr>
<td>CaptureBAT</td>
<td>explorer.exe</td>
<td>C:\WINDOWS\Debug\Data</td>
<td>File added</td>
<td>SAM.dll</td>
</tr>
<tr>
<td>CaptureBAT</td>
<td>explorer.exe</td>
<td>C:\WINDOWS\Debug\Data</td>
<td>File added</td>
<td>system.dll</td>
</tr>
<tr>
<td>CaptureBAT</td>
<td>explorer.exe</td>
<td>C:\WINDOWS\Debug\Data</td>
<td>File added</td>
<td>iestorage.dll</td>
</tr>
<tr>
<td>CaptureBAT</td>
<td>explorer.exe</td>
<td>C:\WINDOWS\Debug\Data</td>
<td>File added</td>
<td>drive.cab</td>
</tr>
</tbody>
</table>

**Figure 27. Example of decrypted communication**

**Table 3. Further changes on the file system and registry**

Frankie Li, ran2@vxrl.org
Applying the filtering function on Process Monitor, three new processes can be identified. (Figure 28)

Three binaries (fvcwin32.exe, acvcwin32.exe and avcwin32.exe) were downloaded and called by the injected “msvcr.dll”. During the behavioral analysis, some files (drive.cab, iestorge.dll, SAM.dll, system.dll, 20110704145735.bmp) were created at C:\WINDOWS\Debug\Data folder. Furthermore, all files with extension of *.dll, *.v2 were uploaded to the C&C through the encrypted HTTP traffic. These files were removed by the injected “msvcr.dll” immediate after upload. In order to test the selective ability of the upload function, some files named as “Secret.v1”, “Secret.v2” and “Secret.dll” were created under C:\WINDOWS\Debug\Data folder. It was found that only the file “Secret.v2” and “Secret.dll” were removed and uploaded to the C&C (Figure 29).

Figure 28. The debugging “Explorer.exe” creates these processes

Figure 29. Only files named with extension of *.dll and *.v2 were uploaded
3.4. Trojan-Spy for Collection of Email Passwords

Before entering the main routine, the binary “avcwin32.exe” tries to create two mutex of “my lovely wood” and “SPI64 RUNNING” (Figure 30).

![Figure 30. Mutux named “my lovely wood” was found under IDA Pro](image)

It extracts information from the SAM file and generates a temporary file with prefix of “SAM” at C:\Documents and Settings\<user>\Application Data. It collects all passwords from “Foxmail”, “Outlook express”, “Outlook”, “Protected Storage”, “IE Form Storage”, “MSN”, “Passport DotNet” and from the system (Figure 31).

![Figure 31. Malicious functions are identified under IDA Pro](image)

Frankie Li, ran2@vxrl.org
All collected passwords are written to temporary files with prefix of “自动表单.txt” (Lit. “AutoList.txt”). These files are subsequently compressed in cab format and renamed as “iestorage.dll” to C:\Windows\Debug\Data folder (Figure 32).

![Contents of iestoreage.dll](image)

Before termination, the process renamed “avicwin32.exe” as “svcwin32.exe”.

3.5. **Trojan-Spy for Collection of File System Details**

Similar to “avicwin32.exe”, the “fvcwin32.exe” process tries to create the same mutex of “my lovely wood” and “SPI64 RUNNING”.

It scans all hard disk, CDROW and floppy diskette to collect all directories and file names and the respective MAC time information from these storage devices (Figure 33).

Frankie Li, ran2@vxrl.org
Figure 33. Directory and file names were gathered

All collected information is kept inside a file called “drive”, which is then compressed into cab format and put under C:\WINDOWS\Debug\Data. The injected “msvcr.dll” remove the un-compressed file after the upload.

3.6. Trojan-Spy for Capturing Screens

The “acvcwin32.exe” process is responsible for creation of screen captures in bitmap format for every 1000 milliseconds. All these screen captures are compressed and renamed with extension of *.v2 and put under C:\WINDOWS\Debug\Data (Figure 34).

Figure 34. Bitmap is created for every 1000 milliseconds
4. Conclusion

The malware is multi-partite in nature, which includes, a dropper (“dg003.exe”), a droppee (“msver.dll”) and at least three Trojan-Spies (“fvcwin32.exe”, “acvcwin32.exe” and “avcwin32.exe”). The malware uses large amount of Windows API calls to reduce its size.

The malware uses encrypted HTTP traffic to transmit collected intelligence back to C&C, which makes it difficult to be discovered.

Indicated by the well-crafted email contents, the intruder has performed intensive prior reconnaissance on the targeted victim or has well studied the culture of the attacking organization before sending the spear-phishing email. The email contains materials of an updated political event and it was sent promptly response to the incidents.

Based on the result of the analysis, the key functions of the malware are non-profit driven (aka not motivated by financial gains), but places high emphasis on spying functions, including: generating screen captures, gathering email and messaging passwords and every directories and file names information from the victim’s machine.

It bears the similar attributes as mentioned in the GhostNet report (TheSecDevGroup, 2009, p. 18-22, 24-26) and in the Operation Shady RAT (Alperovitch, 2011, p. 2-3) which indicates that it is an APT-type malware.

Frankie Li, ran2@vxrl.org
5. References


Frankie Li, ran2@vxrl.org


Frankie Li, ran2@vxrl.org


Snaker, Qwerton & Jibz (2008) PEiD (Version 0.95) [computing software]. Available from http://www.peid.info/


Frankie Li, ran2@vxrl.org
Appendix I

On the Remnux responsive box, the honeyd configuration file was found at /etc/honeypot/honeyd.conf (Figure 35) and the configuration file of farpd was found at /etc/default/farpd (Figure 36).

Figure 35. Configuration of honeyd

```plaintext
create default
#set default default tcp action block
set default default udp action block
#set default default icmp action block
add default tcp port 80 "sh /usr/share/honeyd/scripts/web.sh"
add default tcp port 6666 proxy 127.0.0.1:6666
add default tcp port 3127 proxy 127.0.0.1:3127
add default tcp port 8080 proxy 127.0.0.1:8080
```

Figure 36. Configuration of farpd

```plaintext
# Options for the Fake arp daemon
#
# Interface where farpd will listen in
#INTERFACE="eth0"
#
# Network for which fake arp daemon will answer to
# You can leave this empty but then the daemon will try to take over
# all the network addresses in that interface
#
# Note: Default 'unconfigured' means that farpd will not startup. you
# have to setup a valid value here (can be empty)
#NETWORK=""
#
# Time to wait for the daemon to die before restarting it
# (in seconds)
# DOOTIME=5
```

Frankie Li, ran2@vxrl.org
Appendix II

import ps, sys, glob, pcap, dpkt
from dpkt import ip, tcp
from socket import inet_ntoa
import binascii

def prepare_key(key):
    or_key = '5272059a603057123454b5b0d72'  # or key
    newkey = '
    newkey = []
    for x in or_key:
        step = ord(x)
        newkey.append(step)
        newkey.append(hex(step))
    return newkey

def decode(data, key):
    lenkey = len(key)
    lendata = len(data)
    newdata = []
    newjsondata = []
    for i in range(len(data)):
        if 1 <= i < lenkey:
            j = i + 1
            if ord(j) < 0:
                j = j + 256
            key = j & lenkey
            sk = key[key]
            sv = sv+r
            newdata.append(hex(sv))

    if data[1] + 1:
        j = i + 1
        if (ord(data) & 1) == 1:
            if 1 <= i <= lenkey:
                j = i + 1
                if ord(j) < 0:
                    j = j + 256
                key = j & lenkey
                sk = key[key]
                sv = sv+r
                newdata.append(hex(sv))
        continue

    elif i == 0:
        for i in range(len(data)):
            j = i + 1
            if ord(data) & 1:
                if 1 <= i <= lenkey:
                    j = i + 1
                    if ord(j) < 0:
                        j = j + 256
                    key = j & lenkey
                    sk = key[key]
                    sv = sv+r
                    newdata.append(hex(sv))
            continue

filename = 'file.pcap'
capture = DLPCapturer('file')

for f in files:
    key = prepare_key()
    for f in files:
        outfilename = f.replace('.pcap', '.out.bin')
        outfile = open(outfilename, 'w')
        outfile.close()
        outfile = open(outfilename, 'r')
        outfile.close()
        outfile = open(outfilename, 'w')
        outfile.close()
        outfile = open(outfilename, 'r')
        outfile.close()
        outfile = open(outfilename, 'w')
        outfile.close()
        outfile = open(outfilename, 'r')
        outfile.close()
        outfile = open(outfilename, 'w')
        outfile.close()
        outfile = open(outfilename, 'r')
        outfile.close()
        outfile = open(outfilename, 'w')
        outfile.close()
        outfile = open(outfilename, 'r')
        outfile.close()
        outfile = open(outfilename, 'w')
        outfile.close()
        outfile = open(outfilename, 'r')
        outfile.close()
        outfile = open(outfilename, 'w')
        outfile.close()
        outfile = open(outfilename, 'r')
        outfile.close()
        outfile = open(outfilename, 'w')
        outfile.close()
        outfile = open(outfilename, 'r')
        outfile.close()
        outfile = open(outfilename, 'w')
        outfile.close()
        outfile = open(outfilename, 'r')
        outfile.close()
        outfile = open(outfilename, 'w')
        outfile.close()
        outfile = open(outfilename, 'r')
        outfile.close()
        outfile = open(outfilename, 'w')
        outfile.close()
        outfile = open(outfilename, 'r')
        outfile.close()
        outfile = open(outfilename, 'w')
        outfile.close()
        outfile = open(outfilename, 'r')
        outfile.close()
        outfile = open(outfilename, 'w')
        outfile.close()
        outfile = open(outfilename, 'r')
        outfile.close()
        outfile = open(outfilename, 'w')
        outfile.close()
        outfile = open(outfilename, 'r')
        outfile.close()
        outfile = open(outfilename, 'w')
        outfile.close()
        outfile = open(outfilename, 'r')
        outfile.close()
        outfile = open(outfilename, 'w')
        outfile.close()
        outfile = open(outfilename, 'r')
        outfile.close()
        outfile = open(outfilename, 'w')
        outfile.close()
        outfile = open(outfilename, 'r')
        outfile.close()
        outfile = open(outfilename, 'w')
        outfile.close()
        outfile = open(outfilename, 'r')
        outfile.close()
        outfile = open(outfilename, 'w')
        outfile.close()
        outfile = open(outfilename, 'r')
        outfile.close()
        outfile = open(outfilename, 'w')
        outfile.close()
        outfile = open(outfilename, 'r')
        outfile.close()
        outfile = open(outfilename, 'w')
        outfile.close()
        outfile = open(outfilename, 'r')
        outfile.close()
        outfile = open(outfilename, 'w')
        outfile.close()
        outfile = open(outfilename, 'r')
        outfile.close()