# Inside the EquationDrug Espionage Platform

By GReAT

Published: 2015-03-11 · Archived: 2026-04-05 14:39:39 UTC

## Introduction

EquationDrug is one of the main espionage platforms used by the Equation Group, a highly sophisticated threat actor that has been engaged in multiple CNE (computer network exploitation) operations dating back to 2001, and perhaps as early as 1996. (See full report here [PDF]).

EquationDrug, which is still in use, dates back to 2003, although the more modern GrayFish platform is being pushed to new victims.

> EquationDrug represents the main espionage platform from the #EquationAPT Group
>
> Tweet

It's important to note that EquationDrug is not just a Trojan, but a full espionage ***platform,*** which includes a framework for conducting cyberespionage activities by deploying specific modules on the machines of selected victims. The concept of a cyberespionage platform is neither new nor unique. Other threat actors known to use such sophisticated platforms include Regin and Epic Turla.

The EquationDrug platform can be extended through plugins (or modules). It is pre-built with a default set of plugins supporting a number of basic cyberespionage functions. These include common features such as file collection and the making of screenshots. Sophistication is added by storing stolen data inside a custom-encrypted virtual file system before it is sent to the command and control servers.

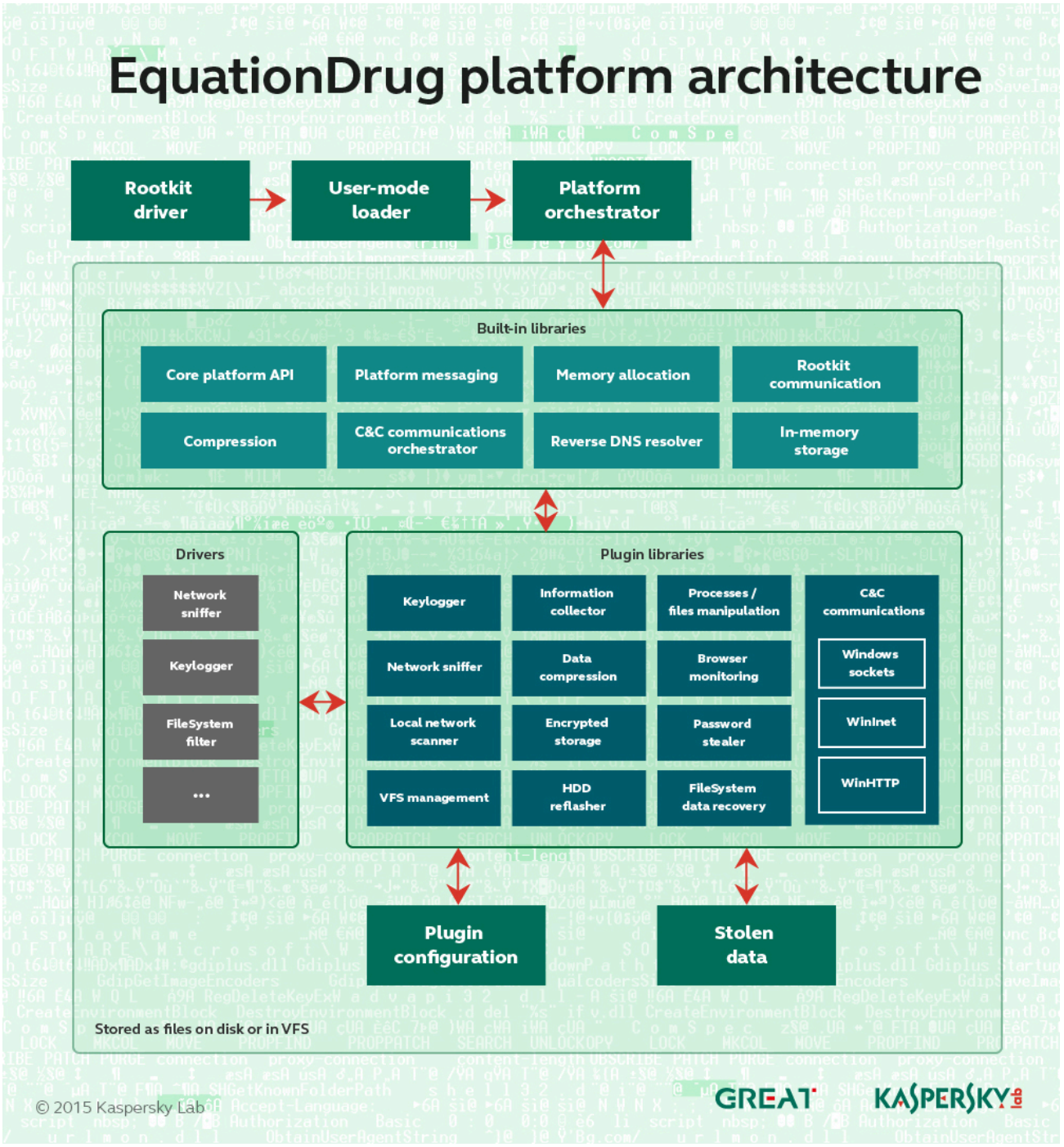
The name "EquationDrug" or "Equestre" was assigned to this framework by Kaspersky Lab researchers. The only reference left by the framework developers was a short string "**UR**", as seen in several string artifacts left in the binaries.

## Platform Architecture

The EquationDrug platform includes dozens of executables, configurations and protected storage locations. Putting all the pieces of this puzzle together in the right order may take time for those who are not familiar with the platform.

> The platform includes executables, configurations and protected storage locations #EquationAPT
>
> Tweet

The architecture of the whole framework resembles a mini-operating system with kernel-mode and user-mode components carefully interacting with each other via a custom message-passing interface. The platform includes a set of drivers, a platform core (orchestrator) and a number of plugins. Every plugin has a unique ID and version number that defines a set of functions it can provide. Some of the plugins depend on others and might not work unless dependencies are resolved.




Similar to popular OS kernel designs, such as on Unix-based systems, some of the essential modules are statically linked to the platform core, while others are loaded on demand.

> The hypothesis that these attackers have been active since the 90s seems realistic #EquationAPT
>
> Tweet

The platform is started by the kernel mode driver component (“msndsrv.sys” on Windows 2000 or above and “mssvc32.vxd” on Windows 9x). The driver then waits for the system to start and initiates execution of the user-mode loader “mscfg32.exe”. The loader then starts the platform’s central module (an orchestrator) from the “mscfg32.dll” module. Additional drivers and libraries may be loaded by different components of the platform, either built-in or auxiliary.

## Platform Components

The EquationDrug platform can be as sophisticated as a space station, but it appears to be of no use without its cyberespionage features. This function is provided by plugin modules that are part of the massive framework described above. We discovered dozens of plugins and each is a sophisticated element that can communicate with the core and become aware of the availability of other plugins.

The plugins we discovered probably represent just a fraction of the attackers' potential. Each plugin is assigned a unique plugin ID number (WORD), such as 0x8000, 0x8002, 0x8004, 0x8006, etc. All plugin IDs are even numbers and they all start from byte 0x80. The biggest plugin ID we have seen is 0x80CA. To date, we have found 30 unique plugin IDs in total. Considering the fact that the developers assigned plugin IDs incrementally, and assuming that other plugin IDs were assigned to modules that we have not yet discovered, it's not hard to calculate that **86 modules** have yet to be discovered.

> 86 modules have yet to be discovered #EquationAPT
>
> Tweet

The most interesting modules we have seen contain the following functionality:

- Network traffic interception for stealing or re-routing.
- Reverse DNS resolution (DNS PTR records).
- Computer management:
- - Start/stop processes
  - Load drivers and libraries
  - Manage files and directories
- System information gathering:
- - OS version
  - Computer name
  - User name
  - Locale
  - Keyboard layout
  - Timezone
  - Process list
- Browsing network resources and enumerating and accessing shares.
- WMI information gathering.
- Collection of cached passwords.
- Enumeration of processes and other system objects.
- Monitoring LIVE user activity in web browsers.
- Low-level NTFS filesystem access based on the popular Sleuthkit framework.
- Monitoring removable storage drives.
- Passive network backdoor (runs Equation shellcode from raw traffic).
- HDD and SSD firmware manipulation.
- Keylogging and clipboard monitoring.
- Browser history, cached passwords and form auto-fill data collection.

## Code Artifacts

During our research we paid attention to unique identifiers and codenames used by the developers in the malware. Most of this information is carefully protected with obfuscation or encryption algorithms to prevent quick recognition, but anyone who breaks through this layer of encryption may discover some interesting internal strings, as demonstrated below:


Some other interesting text strings include:

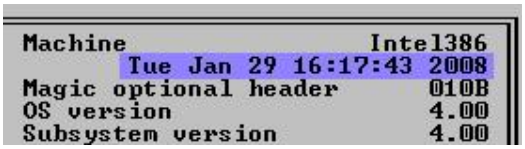
**SkyhookChow** Target
**SkyhookChow** Payload
Dissecorp
Manual/**DRINKPARSLEY**/2008-09-30/10:06:46.468-04:00
VTT/82053737/**STRAITACID**/2008-09-03/10:44:56.361-04:00
VTT/82051410/**LUTEUSOBSTOS**/2008-07-30/17:27:23.715-04:00
**STRAITSHOOTER**30.ex_
**BACKSNARF**_AB25
c:\users**rmgree5**\co**standalonegrok**_2.1.1.1\gk_driver\gk_sa_driver…
To install: run with no arguments
Attempting to drop
**SF**Criteria_Check failed!
**SF**Driver
Error detected! Uninstalling…
Timeout waiting for the “canInstallNow” event from the **implant-specific** EXE!
Trying to call **privilege lib**…
Hiding directory
Hiding plugin…
Merging plugin…
Merging old plugin key…
Couldn’t reset canInstallNowEvent!
Performing **UR-**specific pre-install…
Work complete.
Merged transport manager state.
!!**SF**Config!!

Some other names, such as kernel object and file names, abbreviations, resource code page and several generic messages, point to English-speaking developers. Due to the limited number of such text strings it’s hard to tell reliably if the developers were native English speakers.

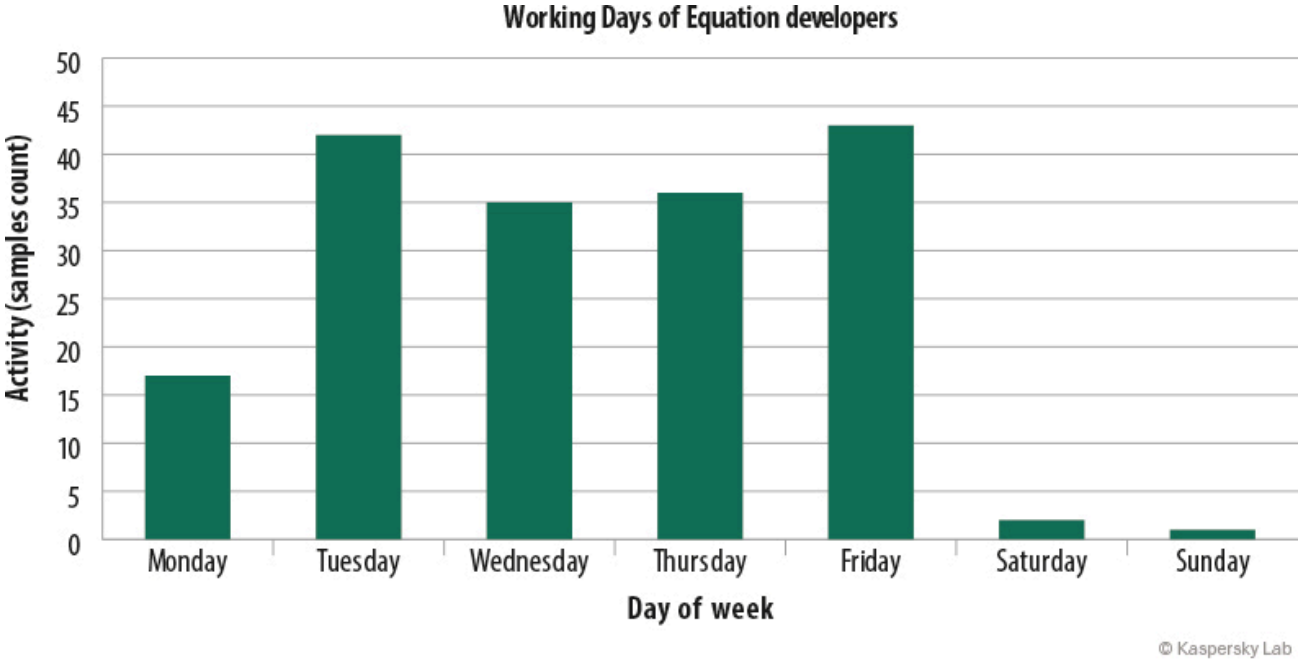
## Link Timestamp Analysis

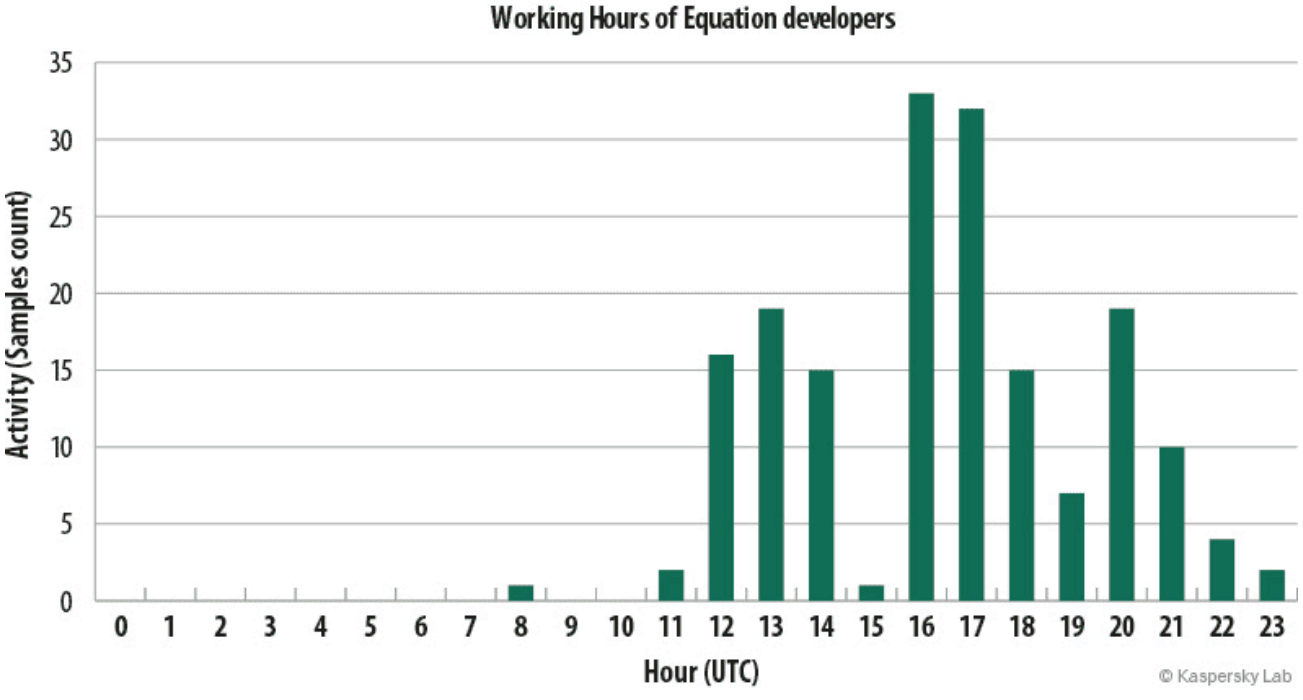
We have gathered a reasonably large number of executable samples to which we have been able to apply link timestamp analysis.

A link timestamp is a 4-bytes value stored in an executable file header. This value is automatically set by compiler software when a developer builds a new executable. The value contains a detailed timestamp including minutes and even seconds of compilation time (think of it as the file's moment of birth).




Link timestamp analysis require the collection of the timestamps of all available executables, grouping them according to certain criteria, such as the hour or day of the week, and putting them on a chart. Below are some charts built using this approach.



Can we trust this information? The answer is: not fully, because the link timestamp can be altered by the developer in a way that's not always possible to spot. However, certain indicators such as matching the year on the timestamp with the support of technology popular in that year leads us to believe that the timestamps were, at the very least, not wholly replaced. Looking at this from the other side, the easiest option for the developer is to wipe the timestamp completely, replacing it with zeroes. This was not found in the case of EquationDrug. In fact, the timestamps look very realistic and match the working days and hours of a well-organized software developer from timezone UTC-3 or UTC-4, if you assume that they come to work at 8 or 9 am.

> The timestamps match the working days of software developer from timezone UTC-3 or UTC-4 #EquationAPT
>
> Tweet

And finally, in case you are wondering if the developers work on public holidays, you can check this for yourself against the full list of their working dates:

| | | | | | |
|---|---|---|---|---|---|
| 2001.08.17 | 2007.12.11 | 2009.04.16 | 2011.10.20 | 2012.08.31 | 2013.06.11 |
| 2001.08.23 | 2007.12.17 | 2009.06.05 | 2011.10.26 | 2012.09.28 | 2013.06.26 |
| 2003.08.16 | 2008.01.01 | 2009.12.15 | 2012.03.06 | 2012.10.23 | 2013.08.09 |
| 2003.08.17 | 2008.01.23 | 2010.01.22 | 2012.03.22 | 2012.11.02 | 2013.08.28 |
| 2005.03.16 | 2008.01.24 | 2010.02.19 | 2012.04.03 | 2012.11.06 | 2013.10.16 |
| 2005.09.08 | 2008.01.29 | 2010.02.22 | 2012.04.04 | 2013.01.08 | 2013.11.04 |
| 2006.06.15 | 2008.01.30 | 2010.03.27 | 2012.04.05 | 2013.02.07 | 2013.11.26 |
| 2006.09.18 | 2008.04.24 | 2010.06.15 | 2012.04.12 | 2013.02.21 | 2013.12.04 |
| 2006.10.04 | 2008.05.07 | 2011.02.09 | 2012.07.02 | 2013.02.22 | 2013.12.05 |
| 2006.10.16 | 2008.05.09 | 2011.02.23 | 2012.07.09 | 2013.02.27 | 2013.12.13 |
| 2007.07.12 | 2008.06.17 | 2011.08.08 | 2012.07.17 | 2013.04.16 | |
| 2007.10.02 | 2008.09.17 | 2011.08.30 | 2012.08.02 | 2013.05.08 | |
| 2007.10.16 | 2008.09.24 | 2011.09.02 | 2012.08.03 | 2013.05.14 | |
| 2007.12.10 | 2008.12.05 | 2011.10.04 | 2012.08.14 | 2013.05.24 | |

## Conclusions

EquationDrug represents the main espionage platform from the Equation Group. It's been in use for over 10 years, replacing EquationLaser until it was replaced itself by the even more sophisticated GrayFish platform.

> The EquationDrug case demonstrates an interesting trend: a growth in code sophistication #EquationAPT
>
> Tweet

The EquationDrug case demonstrates an interesting trend that we have been seeing while analyzing supposedly nation-state cyberattack tools: a growth in code sophistication. It is clear that nation-state attackers are looking for better stability, invisibility, reliability and universality in their cyberespionage tools. You can make a basic browser password-stealer or a sniffer within days. However, nation-states are focused on creating frameworks for wrapping such code into something that can be customized on live systems and provide a reliable way to store all components and data in encrypted form, inaccessible to normal users. While traditional cybercriminals mass-distribute emails with malicious attachments or infect websites on a large scale, nation-states create automatic systems infecting only selected users. While traditional cybercriminals typically reuse one malicious file for all victims, nation-states prepare malware unique to each victim and even implement restrictions preventing decryption and execution outside of the target computer.

> Nation-state attackers create automatic systems infecting only selected users #EquationAPT
>
> Tweet

Sophistication of the framework is what makes this type of actor different from traditional cybercriminals, who prefer to focus on payload and malware capabilities such as implementing a long list of custom third-party software credential database parsers.

The difference in tactics between cybercriminals and nation-state attackers appears to be due to relative resource availability. It's known that cybercriminals attempt to infect as many users as possible and that they can sometimes compromise hundreds of thousands of systems. It would will take many years to check all those machines manually, analyzing who owns them, what data is stored on them, and what custom software they run.

Cybercriminals probably don't even have enough disk space to collect all the potentially interesting data from the victims hit by their large scale infections. That is why cybercriminals prefer to extract tiny chunks of the most important data (credentials, credit card numbers, etc) on the machine of the victim and transfer only few kilobytes from each compromised host. Such data, when combined from all users, normally takes up gigabytes of disk space.

Nation-state attackers have sufficient resources to store as much data as they want. They have access to virtually unlimited data storage. However, they don't need, and often try to avoid, infecting random users, for the obvious reason of avoiding attention and remaining invisible. Implementing custom data format parsers in the malware not only doesn't help them find all the valuable data on the victim's machine, but may also attract extra attention from security software running on the system. They mostly prefer to have a generic remote system management tool that can copy any information they might need even if it causes some redundancy. However, copying large volumes of information might slow down network connection and attract attention, especially in some countries with poorly developed internet infrastructure. To date, nation-state attackers have had to balance between these two poles: copying victims' entire hard drives while stealing only tiny bits of passwords and keys.

> Nation-state attackers use a remote system management tool that can copy any information they need #EquationAPT
>
> Tweet

Now, if you wonder why EquationDrug, a powerful cyberespionage platform, doesn't provide all stealing capability as standard in its malware core, the answer is that they prefer to customize the attack for each one of their victims. Only if they have chosen to actively monitor you and the security products on your machines have been disarmed, will you receive a plugin for the live tracking of your conversations or other specific functions related to your activities. We believe modularity and customization will become a unique trademark of nation-state attackers in the future.

Some code paths in EquationDrug modules lead to OS version checks including a test for Windows 95, which is accepted as one of supported platforms. While some other checks will not pass on Windows 95, the presence of this code means that this OS was supported in some earlier variants of the malware. Considering this and the existence of components designed to run on Windows 9x (such as VXD-files), as well as compilation timestamps dating back to early 2000s, the hypothesis that these attackers have been active since the 90s seems realistic. This makes the current attacker an outstanding actor operating longer than any other in the field.

---

## Technical Details

### Kernel mode stage 0 (Windows 9x) – mssvc32.vxd

| | |
|---|---|
| **MD5** | 0a5e9b15014733ee7685d8c8be81fb0d |
| **Size** | 6 710 bytes |
| **Format** | Linear Executable (LE) |

This VXD driver handles only two control messages: W32_DeviceIoControl and Dynamic_Init. The DeviceIoControl part is not completely implemented and the driver is only able to check for some known control codes. However it does nothing. This handler looks more like a code stub rather than actual payload.

On the Dynamic_Init event, the driver retrieves the location of the user-mode loader executable from the following registry value:

***[HKLM\SYSTEM\CurrentControlSet\Control\Session Manager\MemSubSys] Config***

If the value is not present in the registry, it uses the following fallback string hardcoded in the binary:

**C:\WINDOWS\SYSTEM\SVCHOST32.EXE**

Next, it installs a callback procedure using Windows function _SHELL_CallAtAppyTime. This procedure will be called when CPU is running in ring-3 mode, so that a new executable (loader process) can be started via the traditional way. This is a standard trick that was used by developers in the 90s to initiate a call to DLL export in ring-3 from ring-0 in Windows 9x OS family.

### Kernel mode stage 0 and rootkit (Windows 2000 and above) – msndsrv.sys

| | |
|---|---|
| **MD5** | c4f8671c1f00dab30f5f88d684af1927 |
| **Size** | 105 392 bytes |
| **Format** | PE32 Native |
| **Compiled** | 2008.01.23 14:12:33 (GMT) |
| **Location** | %System32%\drivers\msndsrv.sys |

This module can create log files in the following known locations:

**%systemroot%\system32\mslog32.dat**
**%systemroot%\system32\msperf32.dat** (default location)

The driver acts as the first stage of the EquationDrug platform on Windows 2000+ and implements rootkit functions for hiding the components of the platform. Additionally, it implements a NDIS driver for filtering network traffic.

When started and initialized, the driver retrieves the location of the user-mode loader executable from the registry value:

***[HKLM\System\CurrentControlSet\Services\%driver name%] Config***

The **%driver name%** is not hardcoded and is obtained dynamically from the current module name, which means that different instances may check different registry keys and this may not be a reliable way to check for infection. The sample we analyzed used "**msndsrv**" as the **%driver name%**.

Next, it crafts and injects a shellcode in "services.exe" or "winlogon.exe". The shellcode is designed to spawn the loader process from the executable called **"mscfg32.exe"**.

The rootkit code in the driver hooks several Native API functions that lets it hide or protect registry keys, files and running processes. The components of EquationDrug can modify the list of protected objects by sending DeviceIoControl messages to the driver. The driver also maintains a persistent list of protected objects that is stored in the following registry values:

***[HKLM\System\CurrentControlSet\Services\%driver name%] 1***
***[HKLM\System\CurrentControlSet\Services\%driver name%] 2***

These values are also protected by the rootkit. They can be revealed by booting Windows in Safe Mode.

The driver contains the following unused strings:

- **\\.\mailslot\dskInfo**
- **Dissecorp**

### User-mode loader – mscfg32.exe, svchost32.exe

| | |
|---|---|
| **MD5** | c3af66b9ce29efe5ee34e87b6e136e3a |
| **Size** | 22 016 bytes |
| **Format** | PE32 EXE |
| **Compiled** | 2008.01.23 14:26:05 (GMT) |
| **Location** | %System32%\mscfg32.exe |

This module opens a unique event named "**D0385CB7-B834-45d1-A501-1A1700E6C34E**". If the event exists, it waits for 10 seconds and attempts to open a file whose name can be decrypted as "**\\.\MSNDSRV**". If the device file is successfully opened, the code issues a device request with IOCTL code **0x80000194** and no parameters.

This module uses RC5 in CBC-like mode with a key length of 96-bit for string encryption.

Careful analysis reveals some bits of uninitialized memory found next to encryption key locations. This is unused but partly meaningful memory, because it seems to contain short chunks of strings resembling some local filepaths:

- **“rver\8” (probably part of “Server\8…” string)**
- **“LInj” (could be a part of “DLLInjector” or similar)**

It’s apparent that some parts of the code were designed to run on Windows 9x, for example a call to **RegisterServiceProcess** Windows API function makes sense only on Windows 9x OS family, because this API function doesn’t exist on Windows NT platform.

The module uses a unique algorithm for generating registry value names. The code contains strings, such as “SkyhookChow Target“, that are converted to GUID-like strings by calculating SHA1 hash and using its hexadecimal representation as a string. The resulting strings are used as actual registry value names in ***[HKLM\SYSTEM\CurrentControlSet\Control\Session Manager\MemSubSys]*** registry key.

Sample registry value names:

| **Original String** | **GUID-like registry value name** |
|---|---|
| SkyhookChow Target | {B6F5CD13-A74D-8B82-A6AA-6FA1BE2484C1-6832DF06} |
| SkyhookChow Payload | {F4CF0326-6DCD-EEC8-5323-01CEDB66741A-B55F6F12} |

These registry values are encrypted using an RC5 algorithm using a hardcoded 1024-bit key with 24 rounds.

The registry value:

***[HKLM\SYSTEM\CurrentControlSet\Control\Session Manager\MemSubSys] {F4CF0326-6DCD-EEC8-5323-01CEDB66741A-B55F6F12}*** (“SkyhookChow Payload”)
should contain the location of the orchestrator DLL file (“mscfg32.dll”). If the value is not present a default value **“%SYSTEM%\mscfg32.dll”** is used.

The registry value:

***[HKLM\SYSTEM\CurrentControlSet\Control\Session Manager\MemSubSys] {B6F5CD13-A74D-8B82-A6AA-6FA1BE2484C1-6832DF06}*** (“SkyhookChow Target”)
may contain the location of the executable file that will be used as a “shell” process for the orchestrator library.

The module attempts to start the “shell” process in suspended mode. If there is no “SkyhookChow Target” value or the specified executable fails to start, the module tries different failsafe locations of the programs that can be used instead:

1. 1 Default browser set in the registry ***[HKLM\SOFTWARE\Clients\StartMenuInternet\{current @default value}\shell\open\command]***
2. 2 **%SystemRoot%\System32\svchost.exe**
3. 3 **%SystemRoot%\System32\lsass.exe**
4. 4 Spoolsv service binary from the ***[HKLM\SYSTEM\CurrentControlSet\Services\Spooler] ImagePath*** registry value.
5. 5 Default html file handler from ***[HKLM\SOFTWARE\Classes\htmlfile\shell\open\command]***registry value.
6. 6 Internet Explorer path from ***[HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\App Paths\] IEXPLORE.EXE*** registry value.

Next, the module injects extra code into a newly started target process. The injected code loads the payload DLL ("mscfg32.dll") into the target process and waits for the parent process to exit. When the parent process quits, it unloads the payload DLL and exits as well. The rest of the logic relies on the loaded DLL in that new process. See the description of the **"mscfg32.dll"** module below.

The module communicates with the Stage0/Rootkit driver **"msndsrv.sys"** by sending DeviceIoControl messages to the device "**\\.\MSNDSRV".** It activates the rootkit for its own process, for the target process holding the orchestrator and for all the files involved.

### Platform orchestrator – mscfg32.dll, svchost32.dll

| | |
|---|---|
| **MD5** | 5767b9d851d0c24e13eca1bfd16ea424 |
| **Size** | 249 856 bytes |
| **Format** | PE32 DLL |
| **Compiled** | 2008.01.24 22:11:34 (GMT) |
| **Location** | %System%\mscfg32.dll |

Creates mutex: "**01C482BA-BD31-4874-A08B-A93EA5BCE511**", or terminates if one already exists.

Writes a timestamped log file to one of the following locations:

- **%SystemRoot%\temp\~yh56816.tmp**
- **C:\Windows\Temp\~yh56816.tmp**
- **%Registry_SystemRoot_Value%\temp\~yh56816.tmp**
- Value of ***[HKLM\SYSTEM\CurrentControlSet\Control\Session Manager\MemSubSys] D***

The file "~yh56816.tmp" retains the history of execution. It comprises debug records of simple structure:

**Stage: DWORD | DateTimeLow: DWORD | DateTimeHigh: DWORD**

Basically, it logs the execution of every stage of the orchestrator and the time of execution. The Stage is an integer number starting from 1.

This module spawns a new thread in the DllMain function which contains the main function body. The procedure disables application error popups shown by the default exception handler. This is probably done only in the "Release" version of the malware, because the following code generates exceptions that are reported to the user if application error popups are not disabled. We assume that the "Debug" version of the code doesn't suppress error popups when exception occurs as this helps with the debugging of the code.

The module checks the OS version and if it encounters an unsupported operating system the code generates an exception which terminates the application. The list of OS versions that pass this test:

- Windows 95/98/ME
- Windows NT 4.0 and above.

If the module runs on Win9x, it executes Win9x-specific function RegisterServiceProcess to hide from the Windows Task Manager application. If the module is NOT running on WinNT6.0+, it then attempts to open a virtual device file with one

of the following names:

- \\.**MSSVC32** on Win9x
- \\.**MSNDSRV** on WinNT

If the device file is successfully opened, the module activates a rootkit for its process and for the file location **"%SYSTEM%\unilay.dll"** local path. This is followed by finding and terminating a process named "**winproc.exe**" which is the name of another component of the platform. Note that this part of the code is executed only on platforms different from WinNT 6.x (Windows Vista and later).

The module was designed to fetch or update its main configuration data from different places. There are some default values set inside the code, such as some timeout values and the following C&Cs:

- **www.waeservices[.]com**
- **213.198.79.49**

These default values can be overwritten later.

Next, it locates a data section called "**Share2**" in the current module and verifies the starting magic number. If it is **0x63959700,** it then decrypts the rest of the data in the section and interprets it as a configuration block. However, data from the next location can override all previous settings. This is a registry value with special name.

The naming of the registry location is the same GUID-like SHA1 value as the one used in the loader ("mscfg32.exe"), and is produced from the source string "Configuration":

***[HKLM\SYSTEM\CurrentControlSet\Control\Session Manager\MemSubSys] {42E14DD3-F07A-78F1-7659-26AE141569AC-E0B3EE89}***

The configuration block stored in the registry value is encrypted using RC5 with the 1024-bit key. Both the loader and the orchestrator share the same key for encrypting and decrypting the registry values in the "MemSubSys" key.

The decrypted configuration block consists of a series of tagged configuration records in the following format:

**[RecordType:DWORD][RecordSize: DWORD][RecordValue: %RecordSize%]**

We retrieved a copy of a configuration block and decrypted and partly interpreted it. We are including the results for one of the configuration blocks:

**Time value**: 1 year 0 months 1 days 22 hours 6 mins 52 secs. The orchestrator is expected to set this field to the time of initial configuration.
**Binaries**: 3×1024-bit encryption keys
1b8e7818dad6345c53c2707a2c44648eee700d5cf34fea6a19a3fa0a6a871c72963fdded
91e2703c82b7747b8793e3063700da32cfb8d907dcce1beb36edd575418d1134ef188b
27ec3ce23711a656b0a8bf28921fbf1c39b4c90ad561e4174ed90f26ce11245bb9deb4b
4720403f47ca865ec8bbd3c1df9d93d042ff5b52ec6
05000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000
ed04953f3452068ae6439f04c7904c8be5e98e66e2cd0f267d65240aeed88bd4d3c6105

c99950dd42ccde4bc6bbaf9f6cb1b4e628d943e91f8f97f2aff705fdd25e3af6ba0bc4fd13
d67a2bcb751bb8f21f3d4b66c599f3e572802911394d142f8cf3a299d6d4558f9f0f01634
9afd1888472f4f8c729ffe913f670931f1a227
**C&C domain**: www[dot]waeservices[dot]com
**C&C IP address**: 213.198.79.49
**C&C port**: 443
**Timestamp**: 2010-12-08 11:35:57
**Tool Reference**: **VTT/82055898/STEALTHFIGHTER/ 2008-10-16/14:59:06.229-04:00**
**TimeoutA**: 25200 sec (7 hours)
**TimeoutB**: 32400 sec (9 hours)
**TimeoutC**: 3600 sec (1 hour)
**TimeoutD**: 172800 sec (48 hours)
+Several Unknown Values

Other configuration blocks we discovered contained similar information, with only some unique values:

**Timestamp**: 2009-11-23 14:10:15
**Tool Reference**: **Manual/DRINKPARSLEY/2008-09-30/10:06:46.468-04:00**
**Tool Reference**: **VTT/82053737/STRAITACID/2008-09-03/10:44:56.361-04:00**
**Tool Reference**: **STRAITSHOOTER30.ex_**
**Tool Reference:** VTT/82051410/**LUTEUSOBSTOS**/2008-07-30/17:27:23.715-04:00
**Tool Reference: BACKSNARF_AB25**

During the next step, the module obtains PE file version information from the resource section. It loads the version info using hard-coded module names, which are supposed to match the current module name:

- **SVCHOST32.DLL** for Windows 9x
- **MSCFG32.DLL** for Windows NT

If file version information is available, it gets language-specific values of the PrivateBuild block. The codepage and languages that are verified: Unicode, **LANG_NEUTRAL** and **LANG_ENGLISH_US**. When this check passes, the module gets **@default** registry value from the following location:

- ***[HKLM\SOFTWARE\Classes\CLSID\{091FD378-422D-A36E-8487-83B57ADD2109}] TypeLib***

If the key is not found, the code checks for registry value **TypeLib** in the following key:

- ***[HKLM\SOFTWARE\Classes\CLSID\{091FD378-422D-A36E-8487-83B57ADD2109}]***

If such a value is found, it is then deleted along with the **Version** value if it exists in the same key.

The string obtained from one of two possible registry values is processed as if this value is a CLSID-like string: the code takes the last 16 hexadecimal digits, splits them in two 8-chars values, converts them to binary form (two DWORDs) and reverses the order of bytes in each DWORD and XORs, the first value with **0x8ED400C0,** and the second with **0x4FC2C17B**. Next, the first DWORD value becomes second and the second becomes first. In this order, they are stored in a structure in memory. These two values seem to be very important as they override a few values in the previously known configuration. If they don’t exist, values from the current configuration replace them and are stored back in the registry following the reverse procedure:

1. 1 ***[HKLM\SOFTWARE\Classes\CLSID\{091FD378-422D-A36E-8487-83B57ADD2109}\Version]*** is created and **@default** value is set to version obtained from file version information **PrivateBuild** field (i.e. **3.04.00.0001**). This seems to be **used as kit version number**.
2. 2 ***[HKLM\SOFTWARE\Classes\CLSID\{091FD378-422D-A36E-8487-83B57ADD2109}\Version]*** is created and **@default** value is set to a CLSID like string generated from the following:
    - Fixed prefix string: **"{8C936AF9-243D-11D0-"**
    - Two important DWORD values in the format of "%04X-%04X%08X}" string.

We collected and decrypted several samples of such values. According to the code, they are initialized with values of the Microsoft filetime format. So, we decided to interpret them as filetime values:

20101C04EC2C17B: 1 year(s) 7 month(s) 21 day(s) 23 hour(s) 32 min(s) 1 sec(s)
81E01C04EC2C17B: 1 year(s) 7 month(s) 8 day(s) 12 hour(s) 13 min(s) 5 sec(s)
E0001C04EC2C17B: 1 year(s) 7 month(s) 21 day(s) 1 hour(s) 6 min(s) 15 sec(s)
77101C04EC2C17B: 1 year(s) 5 month(s) 20 day(s) 19 hour(s) 15 min(s) 4 sec(s)
30F01C04EC2C17B: 1 year(s) 8 month(s) 0 day(s) 6 hour(s) 10 min(s) 33 sec(s)
C0901C04EC2C17B: 1 year(s) 8 month(s) 2 day(s) 6 hour(s) 29 min(s) 39 sec(s)
66701C04EC2C17B: 1 year(s) 6 month(s) 9 day(s) 2 hour(s) 10 min(s) 23 sec(s)
F6501C04EC2C17B: 1 year(s) 6 month(s) 6 day(s) 19 hour(s) 53 min(s) 22 sec(s)
01401C04EC2C17B: 1 year(s) 6 month(s) 25 day(s) 23 hour(s) 34 min(s) 13 sec(s)

After that, the module stores current time values in encrypted form in the registry value:

***[HKLM\SYSTEM\CurrentControlSet\Control\Session Manager\MemSubSys] {08DAB849-0E1E-A1F0-DCF1-457081E091DB-117DB663}*** (encoded SHA1 of "StartTime")

The module contains an additional compressed Windows DLL file in the resource section, which is extracted as **"unilay.dll"** (see below). This DLL exports a number of functions that are just wrappers of the system API used to work with files and the registry, and also start processes and load additional DLL files.

The orchestrator contains several built-in plugins that form the core of the platform. These are initialized in the first place, and then additional plugins are loaded. All the plugins are indexed in a single encrypted registry value:

***[HKLM\SYSTEM\CurrentControlSet\Control\Session Manager\MemSubSys] 1***

This value has information about all the components of the current kit. It may include Unicode strings with paths to extra DLLs which serve as plugins. Each DLL exports at least four functions which are imported by ordinal numbers from 1 to 4.

The structure of the registry value "1":

**[Count:DWORD]{ [Plugin Id:WORD][Plugin Path Length:DWORD][Plugin Path String:VARIABLE] }**

Plugins interact with each other and with the orchestrator by exchanging messages of pre-defined format. The message transport is implemented as a global object that contains four communication streams. Every stream contains a pair of kernel synchronization object handles (a semaphore with fixed maximum value defaulted to 1000 and a mutex) and a message queue as an array. A dedicated thread processes messages that appear in the message queues.

A message arrives in a parcel, represented as two DWORD values that contain the size of the message and a pointer to the message data. The message data starts with a DWORD identifying a class of message (a request, reply, etc).

The orchestrator contains the following built-in plugins (listed by internal ID): 8000, 8022, 8024, 803C, 8046, 800A, 8042, 8002, 8004, 8006, 8008, 8070, 808E. Several additional built-in modules have been discovered in newer versions of the orchestrator that was shipped with the GrayFish platform.

## EquationDrug Plugins:

| Plugin ID | File name | Description |
|---|---|---|
| 8000 | *Built-in* | Core, basic API for other modules |
| 8002 | wshcom.dll | C&C communication using Windows sockets |
| 8004 | *Built-in* | Additional message queue |
| 8006 | *Built-in* | Memory allocation / storage |
| 8008 | vnetapi32.dll& | C&C communication code based on DoubleFantasy, using WinInet API |
| 800A | *Built-in* | C&C communication orchestrator |
| 800C | perfcom.dll | HTTP communication |
| 8022 | khlp680w.dll | System API: execute processes, load libraries, manipulate files and directories |
| 8024 | cmib158w.dll | Collects system information: OS version, computer name, user name, locale, keyboard layout, timezone, process lists |
| 8034 | cmib456w.dll | Management of the VFS backed by encrypted “.FON” files in the “Fonts\Extension” directory. Provides encryption using RC5 for these files |
| 803E | nls_874w.dll | Network sniffer |
| 803C | *Built-in* | Communication with the NDIS filter part of “msndsrv.sys” |
| 8040 | khlp807w.dll | Network exploration API, share enumeration and access |
| 8042 | *Built-in* | Compression library based on Nrv2d / UCL |
| 8046 | *Built-in* | Communication with the rootkit part of “msndsrv.sys” |
| 8048 | mstkpr.dll | Disk forensics and direct NTFS reader based on sources of SleuthKit |
| 8050 | khlp760w.dll | Additional encryption facilities for the file-backed VFS |
| 8058 | khlp733w.dll | Collects local system information, WMI information, cached passwords |
| 8070 | khlp747w.dll | Enumerates processes and system objects |
| 807A | mscoreep32.dll | Plugins for monitoring Internet Explorer and Mozilla browser activities |
| 808A | khlp866w.dll | Compression library based on Zlib |
| 808E | *Built-in* | Reverse (PTR record) DNS resolver |

| | | |
|---|---|---|
| 8094 | *Built-in* | In-memory storage |
| 809C | *Built-in* | In-memory storage |
| 80AA | nls933w.dll | HDD / SSD firmware manipulation |
| 80AE | wpl913h.dll | Keylogger and clipboard monitoring (aka “GROK”) |
| 80BE | vnetapi.dll | C&C communication via WinHTTP API |
| 80C6 | webmgr.dll | Extracts web history, Mozilla/Internet Explorer-saved form data and cached credentials |
| 80CA | wshapi.dll | C&C communications interface via Windows sockets |

## Additional components

### Unilay.DLL

This module provides a compatibility layer for accessing system API functions for Windows 9x. It redirects Unicode (“W”) variants of Windows API functions to corresponding ANSI variants by converting Unicode string parameters to multi-byte strings and calling the respective ANSI API.

| | |
|---|---|
| **MD5** | EF4405930E6071AE1F7F6FA7D4F3397D |
| **Size** | 9 728 bytes |
| **Compiled** | 2008.01.23 14:23:10 (GMT) |
| **Format** | PE32 DLL, linker version 6.0 (Microsoft Visual C++ 6.0) |

Exported functions (redirected to ANSI variants):

- 100017EF: CopyFileW
- 10001039: CreateDirectoryW
- 10001111: CreateFileW
- 100011B3: CreateProcessW
- 10001177: DeleteFileW
- 10001516: FindFirstChangeNotificationW
- 10001466: FindFirstFileExW
- 10001300: FindFirstFileW
- 100014C6: FindNextFileW
- 10001564: GetCurrentDirectoryW
- 1000188F: GetFileAttributesW
- 100016C6: GetStartupInfoW
- 10001602: GetSystemDirectoryW
- 10001664: GetWindowsDirectoryW
- 10001853: LoadLibraryW
- 1000178B: MoveFileExW

- 1000172D: MoveFileW
- 10001913: RegCreateKeyExW
- 100019F5: RegDeleteKeyW
- 10001DDF: RegDeleteValueW
- 10001A39: RegEnumKeyExW
- 10001BE2: RegEnumValueW
- 1000199B: RegOpenKeyExW
- 10001B23: RegQueryInfoKeyW
- 10001D57: RegSetValueExW
- 100010D5: RemoveDirectoryW
- 10001E81: SHGetFileInfoW
- 100015C6: SetCurrentDirectoryW
- 100018CB: SetFileAttributesW
- 10001E23: lstrcmpW

### Network-sniffer/patcher – atmdkdrv.sys

| | |
|---|---|
| **MD5s** | 8d87a1845122bf090b3d8656dc9d60a8<br>214f7a2c95bdc265888fbcd24e3587da |
| **Size** | 41 440, 43 840 bytes |
| **Format** | PE32 Native |
| **Compiled** | 2009.04.16 17:19:30 (GMT)<br>2008.05.07 19:55:14 (GMT) |
| **Version Info** | • FileDescription: Network Services<br>• LegalCopyright: Copyright (C) Microsoft Corp. 1981-2000<br>• InternalName: atmdkdrv.sys<br>or<br>• FileDescription: CineMaster C 1.1 WDM Main Driver<br>• LegalCopyright: Copyright 1999 RAVISENT Technologies Inc.<br>• InternalName: ATMDKDRV.SYS |

Creates a file storage "**\SystemRoot\fonts\vgafixa1.fon**". Its first word is set to 0x21 at the beginning of the DriverEntry function, and is replaced with 0x20 at the end of DriverEntry.

This driver appears to have been put together in "quick-and-dirty hack" style, using parts of the "**mstcp32.sys"** sniffer and other unknown drivers. It contains a lot of unused code which is partially broken or disabled. These include a broken "Dynamically disable/enable windows audit logging" subsystem and an incomplete "Patcher mode".

There are three algorithms used for strings encryption – RC5; alphabet encryption like the one used in "mstcp32.sys"; and XOR with a pre-seeded random number generator. Decrypted strings are immediately encrypted back until the next usage to avoid in-memory detection.

The driver's filename and device name differ across the samples. They depend on the name of the registry key that is used to start the driver.

The driver may operate in one of two independent modes – as a network sniffer or as a memory patcher. The mode of operation is selected on startup, based on the "Config2" value of the driver's registry key. By default the driver starts in "sniffer mode".

**Sniffer mode**

The sniffer code is similar to the one used in the driver's "tdip.sys" and "mstcp32.sys" and uses NT4 NDIS-4, XP NDIS-5 interfaces, targeting incoming traffic on Ethernet and VPN (ndiswanip) interfaces. It captures only directed packets (containing a destination address equal to the station address of the NIC). Packers-filtering engine rules may be set via DeviceIoControl messages. Filtered packets are stored in-memory until requested. Maximum packets storage list length is 128 items per filtering rule.

**Patcher mode**

Almost broken, it does nothing interesting except, possibly, replace the thread's ServiceTable to an unchanged, clear copy taken from the on-disk image of "ntoskrnl.exe".

**Sniffer only IOCTLs:**
44038004 – add filtering rule
44038008 – clear stored packet in specified filtering rules list
4403800C – enable specified filtering rule
44038010 – disable specified filtering rule
44038014 – get stored packet from specified filtering rules list
44038018 – process packet like the one received from the wire (filter and store)
4403801C – set maximum rules list length
44038020 – get maximum rules list length
80000004 – enablePacketsFiltering
80000008 – disablePacketsFiltering (PauseSniffer)
800024B4 – send packet to the specified network interface

**Common IOCTLs:**
80000028 – do nothing (broken/unused part)
80000038 – set external object (broken/unused part)
8000003C – get 4 dwords struct (broken/unused part)
80000040 – copy 260 bytes from the request (broken/unused part)
80000320 – set I/O port mapping (broken/unused part)
80000324 – clear I/O port mapping (broken/unused part)
80000328 – set external PnP Event (broken/unused part)
80000640 – replace specified thread's SDT (ETHREAD.ServiceTable field) to a given copy

### Backdoor driven by network sniffer – "mstcp32.sys", "fat32.sys"

| | |
|---|---|
| **MD5s** | 74DE13B5EA68B3DA24ADDC009F84BAEE<br>B2C7339E87C932C491E34CDCD99FEB07 |

| | |
|---|---|
| | 311D4923909E07D5C703235D83BF4479<br>21C278C88D8F6FAEA64250DF3BFFD7C6 |
| **Size** | 57 328 – 57 760 bytes |
| **Format** | PE32 Native |
| **Compiled** | 2007.10.02 12:42:14 (GMT)<br>2001.08.17 20:52:04 (GMT) |
| **Version Info** | • FileDescription: TCP/IP driver<br>• LegalCopyright: Copyright (C) Microsoft Corp. 1981-1999<br>• InternalName: mstcp32.sys |

This is a sniffer tool similar to “tdip.sys” and it uses NT4 NDIS-4, XP NDIS-5 interfaces.  It targets incoming traffic on Ethernet and VPN (ndiswanip) interfaces, but instead of dumb packet dumping, it uses received packets as commands for the “process injector” subsystem that is able to extract and execute code from the specially crafted network packets.

Default filtering rules are stored in the “Options” registry value of the driver’s registry key. It captures only directed packets (containing a destination address equal to the station address of the NIC).

The driver’s filename and device name differ across the samples. They depend on the name of the registry key that is used to start the driver.

**Code Patcher**

The driver patches OS code to dynamically disable or enable Windows audit logging.

It patches the function “**LsapAdtWriteLog”** in “lsasrv.dll” module of the “lsass.exe” process.

It searches for pre-defined signatures of the function “LsapAdtWriteLog” of known Windows versions – 4.0, 5.0, 5.1, 5.2 (NT4, Win2000, XP, WinSrv2003).

Then it selects a corresponding offset to replace the opcodes:

- ‘jz’ to never taken ‘jo’ in case of XP
- jmp over inner logic to procedure epilog in case of Windows Server 2003 so LsapAdtWriteLog skips logging of audit records

The module also patches “SepAdtLogAuditRecord” inside “ntoskrnl.exe” to “retn 4” instead of the first opcode of the function.

The disabled audit can be restored after a timeout or on-event by a dedicated thread.

**Expected IOCTL codes:**

- 80000004 – setFilteringRules
- 80000008 – disablePacketsFiltering (PauseSniffer)
- 80000028 – do nothing (possible broken GetDriverName)
- 80000038 – disable_audit

- 8000003C – enable_audit

**Code Injector**

The code-builder within this module facilitates exploitation by providing up to four predefined execution templates, which seem to be suitable for generating several code patterns.

Below is a list of the execution templates we found:

- locate a DLL via PEB structure and resolve exports
- call single function
- call four functions
- call six functions

Using these as a base for the templates, the code-builder inserts parameters and proper offsets to call one of the following code patterns:

- Locate and call WinExec
- Locate and call LoadLibraryW, GetProcAddress, call exported procedure, FreeLibrary
- Locate and call LoadLibraryW, GetProcAddress, call GetModuleHandle, FreeLibrary
- Locate and call OpenProcess, VirtualAllocEx, WriteProcessMemory, CreateRemoteThread, VirtualFreeEx, CloseHandle

The code injection procedure allocates memory via ZwAllocateVirtualMemory in services.exe and copies implanted code. After that it uses KeInsertQueueApc to let the code run and waits 30 seconds for APC to complete.

When the module starts, it reads registry value ***[HKLM\System\CurrentControlSet\Services\%driver name%] Processes***. This value may contain a list of process names that should be started by injected executable code but only after services.exe and winlogon.exe has been started. The injection of code into winlogon.exe and services.exe ensures that the newly started process will have SYSTEM user privileges. During the injection stage Windows Audit Logging is fully disabled to avoid leaving any suspicious records in Windows Logs.

**Magic Packet Recognition**

All incoming packets are first filtered by BPF-like rules. The filtering rules are located in ***[HKLM\System\CurrentControlSet\Services\%driver name%] Options*** registry value or passed via corresponding IOCTL. Packets that passed through the filter are added in the end of processing queue.

Packets from the queue must have valid checksum values. After checking that, the code XOR-decrypts additional data from the end of the packet. The decrypted end of the packet contains another control structure that defines which encryption algorithm is used to decipher packet body. Supported algorithms include RC5 and RSA. There is a 1024-bits RSA public key hardcoded inside the module body, while a 96-bits RC5 key is generated dynamically.

The backdoor command may arrive in a single packet or be split into pieces and come with several packets. There is a procedure for re-assembling pieces together: a multi-packet command is added to a special packet collector which puts all the pieces together before passing it further.

**Backdoor command** is stored in the first byte of the decrypted request and can be one of the following values:

| Command Byte | Short Name | Description |
|---|---|---|
| 0 | StatusCheck | Reply with encrypted DriverName, packet collector keys, “Processes” and “Params” registry values, names of processes for code injection (defaults are “services.exe” and “winlogon.exe”), list of unprocessed commands from packet collector. |
| 1 | Panic | Disable packet filtering, securely delete driver file, clear related registry keys, set ClearPageFileAtShutdown flag, unbind adapters, delete devices and prepare for unloading. However, the driver cannot unload itself and it remains in the memory until reboot, which means it can be detected via DRIVER_OBJECT structure. |
| 2 | CodeInject | Injects code implant received from packet body. |
| 3 | CallWinExec | Disable Windows Audit and start new process via standard code injection in Winlogon.exe. |
| 4 | ResetCollector | Drop all packets with specific key from packet collector. |
| 5 | Unused or disabled | |
| 6 | GenerateRC5Key | Generates RC5 session key. 96-bits RC5 key is generated by taking a SHA1 hash of value from KeTickCount, XOR-encrypt with SHA1 hash of two PerformanceFrequency values, which is followed by nine more additional SHA1 hashing. |
| 7 | SetRegProcesses | Sets new “Processes” registry value. |
| 8 | DeleteRegProcesses | Deletes current “Processes” registry value. |
| 9 | SetRegParams | Sets a universal “Params” registry value which is used for storing any information, i.e. could be a unique victim marker or exploitation stage number. |
| a | Unused or disabled | |
| b | Ping | Reply with classic ICMP Echo Response packet. |
| c | ChangeInjectTarget | Set new target injection processes (defaults are “Winlogon.exe” and “Services.exe”). Effective until reboot. |

Note: “mstcp32” is mentioned together with rootkit-like behavior in 2004 here: http://www.pcreview.co.uk/forums/mstcp32-t1445152.html

### Network Sniffer – tdip.sys

| | |
|---|---|
| **MD5s** | 20506375665a6a62f7d9dd22d1cc9870<br>60dab5bb319281747c5863b44c5ac60d |
| **Size** | 22448 – 28800 bytes |

| | |
|---|---|
| **Format** | PE32 Native |
| **Compiled** | 2006.10.16 18:42:40 (GMT)<br>2003.08.17 21:47:33 (GMT) |

Supports the following versions of Windows: NT4 using NDIS-4 and XP using NDIS-5. Doesn't use Vista and later NDIS-6 features. However, later NDIS versions are backward-compatible, so the driver is still valid for current versions of Windows.

**Version Info:**

- FileDescription: IP Transport Driver
- LegalCopyright: © Microsoft Corporation. All rights reserved.
- FileVersion: 5.1.2600.2180
- InternalName: tdip.sys

This driver is a packet sniffer for incoming-only traffic on Ethernet and VPN (ndiswanip) interfaces or any used with ms_pschedmp as an alternative connection.

It implements a BPF (Berkeley packet filter) style packet-filtering system that is configured from the driver's registry configuration values or from DeviceIoControl messages.

The captured network packets may be written to disk in libpcap format (magic 0xA1B2C3D4 version 2.4) and encrypted with one-byte XOR, key 0xE3.

The driver's configuration is stored in the registry key:
**[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\tdip]**

- **Options** – packet filtering rules in BPF format
- **Tag** – selector of filtered packet types / Defaults in case of MediumWan to NDIS_PACKET_TYPE_BROADCAST|NDIS_PACKET_TYPE_MULTICAST|NDIS_PACKET_TYPE_DIRECTED; (or NDIS_PACKET_TYPE_BROADCAST|NDIS_PACKET_TYPE_DIRECTED in any other case)
- **ImageFile** – full path name to the resulting pcap file
- **Duration** – used as Length of the original packet in dump file. (default 0xffff)
- **Backup** – max size of the pcap file

IOCTLs:

- 0x80002004 getCurrentState
- 0x80002008 setFilteringRules
- 0x8000200C getFilteringRules
- 0x80002024 getDumpFileSize
- 0x80002010/0x80002014/0x80002018/0x8000201C pause/resume
- 0x80002020 getVersion – returns 2.4.0

Driver has three logical parts, and uses an incomplete function pointer table as interface:

1. 1 Business logic: filtering rules, packet dumping, device ioctl, options
2. 2 Ndis driver skeleton

3. 3 Primitives lib: Strings, XORing, registry I/O

The code is of very good quality. It looks more complicated than Winpcap 2.3 (released 28 mar 2002), but less so than Winpcap 3.0 (released by 10 apr 2003). Interestingly, the driver identifies itself as “version 2.4” in the pcap file despite there being no Winpcap version 2.4.

### Key/clipboard logger driver – msrtvd.sys

| | |
|---|---|
| **MD5s** | 98dea1bce37bf7087360e1958400589b<br>bb8f56874189d5dfe9294f0553a49b83<br>f6bf3ed3bcd466e5fd1cbaf6ba658716 |
| **Size** | 31 488 – 36 736 bytes |
| **Format** | PE32 Native |
| **Compiled** | 2010.02.19 22:45:18 (GMT)<br>2008.09.17 16:23:54 (GMT) |
| **Version Info** | • FileDescription: MSRTvd interface driver<br>• LegalCopyright: © Microsoft Corporation. All rights reserved.<br>• InternalName: msrtvd.sys |

This is a keylogger and clipboard monitoring tool.

On startup, the driver creates a device named “**\Device\Gk0**” and a symbolic link named “**\DosDevices\Gk**“.

Then it attaches to the **csrss.exe** process and disassembles **user32.dll** and **ntdll.dll** routines to obtain **win32k.sys** and **ntoskrnl.exe** SDT services indexes and pointers of needed Nt/Zw APIs.

Then, using a built-in disassembler, it obtains pointers to NtUserPeekMessage, NtUserGetMessage, NtUserGetClipboardData and using the disassembler again selects the parts of the code that will be then hooked by splicing.

The interceptor routines are copied from a special PE section named “.msda”. These routines are able to collect key press chains and clipboard text data, add information about current Time, ProcessName, ForegroundWindowText,and UserName related to this event.

A dedicated thread (“dumper”) gathers the collected data, compresses the results with LZO appends it every 30 minutes to a file “***%system-wide TEMP%*\tm154o.da**“.

Most strings inside are encrypted by XOR with a pre-seeded random number generator.

IOCTLs:

- 0x22002C -start dumper thread
- 0x220030 – stop dumper thread
- 0x220034 – check if the driver has new data to dump
- 0x220038 – set two external events signaled on dump data availability (it references a plugin possibility)

- 0x22003C – restart dumper thread
- 0x220040 – get size of available data

### Collector plugin for Volrec – msrstd.sys

| | |
|---|---|
| **MD5s** | 69e7943f3d48233de4a39a924c59ed2c<br>15d39578460e878dd89e8911180494ff |
| **Size** | 13 696 – 17 408 bytes |
| **Format** | PE32 Native |
| **Compiled** | 2009.06.05 16:21:55 (GMT)<br>2009.12.15 16:33:52 (GMT) |
| **Version Info** | • FileDescription: msrstd driver<br>• LegalCopyright: © Microsoft Corporation. All rights reserved.<br>• InternalName: msrstd.sys |

This driver is a plugin that collects events from the "volrec.sys" driver, and delivers them by sending DeviceIoControl messages. It collects events about file and disk volume operations.

On startup the driver obtains a pointer to "**\Device\volrec**", then creates a control device "**\Device\msrstd0**" and a symbolic link to it named **"\DosDevices\msrstd"**

All strings inside the driver are encrypted by XOR with a pre-seeded random number generator.

For file events the driver collects the filenames, and caches data about read and write operations. For disk volume events it queries disk properties and reads volume labels and disk serial numbers of removable drives (USB, FireWire drives).

IOCTLs:

**0x220004** – turn on VolumeEvents collection
**0x220008** – turn off VolumeEvents collection
**0x22000C** – retrieve previously stored VolumeEvent (operationType, deviceTypeFlags, VolumeLabel, volumeSerialNumber, DosDriveLetter)
**0x220010** – turn on FileEvents collection
**0x220014** – turn off FileEvents collection
**0x220018** – retrieve previously stored FileEvent (fileName, deviceTypeFlags, VolumeLabel, volumeSerialNumber, DosDriveLetter)
**0x22001C** – connect to Volrec.sys (send ioctl 0x220004), enable plugin operation
**0x220020** – disconnect from Volrec.sys (send ioctl 0x220008), disable plugin operation

### Filesystem filter driver – volrec.sys, scsi2mgr.sys

| | |
|---|---|
| **MD5s** | a6662b8ebca61ca09ce89e1e4f43665d<br>c17e16a54916d3838f63d208ebab9879 |

| | |
|---|---|
| **Size** | 14 464-14 848 byres |
| **Format** | PE32 Native |
| **Compiled** | 2009.06.05 16:21:57 (GMT)<br>2009.12.15 16:33:57 (GMT) |
| **Version Info** | • FileDescription: Volume recognizer driver<br>• LegalCopyright: © Microsoft Corporation. All rights reserved.<br>• InternalName: volrec.sys |

This driver is a generic filesystem filter which feeds system events to user-mode plugins.

On startup the driver creates a control device named "\Device\volrec" and a symbolic link to it named "\DosDevices\volrec0". It then attaches all available filesystem devices.  It is also, able to handle removable storage devices.

All strings inside the driver are encrypted by XOR with a pre-seeded random number generator.

IOCTLs:

- 0x220004 – setup plugin interface
- 0x220008 – disable plugin calls

The driver handles the following system events:

- file opened, created or closed
- data is read or written to a file
- new volume is mounted, unmounted
- new USB or FireWire device attached

### HDD/SSD operation helper driver – WIN32M.SYS

| | |
|---|---|
| **MD5s** | 2b444ac5209a8b4140dd6b747a996653<br>b3487fdd1efd2d1ea1550fef5b749037 |
| **Size** | 19 456 – 26 631 bytes |
| **Format** | PE32 Native, PE32+ Native |
| **Compiled** | 2001.08.23 17:03:19 (GMT)<br>2013.05.14 15:58:36 (GMT) |
| **Description** | This module will be the subject of a dedicated blogpost. |

### HDD/SSD firmware operation – nls_933w.dll

| | |
|---|---|
| **MD5s** | 11fb08b9126cdb4668b3f5135cf7a6c5<br>9f3f6f46c67d3fad2479963361cf118b |

| | |
|---|---|
| **Size** | 212 480 – 310 272 bytes |
| **Format** | PE32 DLL, PE32+ DLL |
| **Compiled** | 2010.06.15 16:23:37 (GMT)<br>2013.05.14 16:12:35 (GMT) |
| **Version Info** (64bit dll only) | • FileDescription: Windows Networking Library<br>• LegalCopyright: Copyright (C) Microsoft Corp. 1981-2001<br>• FileVersion: 80AA<br>• InternalName: nls_933w.dll<br>• OriginalFilename: nls_933w.dll<br>• PrivateBuild: 4.0.1.0<br>• ProductName: Microsoft(R) Windows (R) 2000 Operating System<br>• ProductVersion: 5.0.2074.0<br>• Full Version: 1.0.0.1 |
| **Description** | This (80AA) plugin is a HDD firmware flashing tool which includes an API and the ability to read/write arbitrary information into hidden sectors on the disk.<br>The plugin will be the subject of a separate blogpost. |



---

Source: https://securelist.com/inside-the-equationdrug-espionage-platform/69203/