Carbon Paper: Peering into Turla's second stage backdoor

welivesecurity.com/2017/03/30/carbon-paper-peering-turlas-second-stage-backdoor/

March 30, 2017

The Turla espionage group has been targeting various institutions for many years. Recently, ESET found several new versions of Carbon.

The Turla espionage group has been targeting various institutions for many years. Recently, we found several new versions of Carbon, a second stage backdoor in the Turla group arsenal. Last year, a technical analysis of this component was made by Swiss GovCERT.ch as part of their report detailing the attack that a defense firm owned by the Swiss government, RUAG, suffered in the past.

This blog post highlights the technical innovations that we found in the latest versions of Carbon we have discovered.

Looking at the different versions numbers of Carbon we have, it is clear that it is still under active development. Through the internal versions embedded in the code, we see the new versions are pushed out regularly. The group is also known to change its tools once they are exposed. As such, we have seen that between two major versions, mutexes and file names are being changed.

Infection vectors

The Turla group is known to be painstaking and work in stages, first doing reconnaissance on their victims' systems before deploying their most sophisticated tools such as Carbon.

A classic Carbon compromise chain starts with a user receiving a spearphishing email or visiting a previously compromised website, typically one that the user visits regularly — a technique known as a watering hole attack.

After a successful attack, a first stage backdoor — such as Tavdig [1] or Skipper [2] — is installed on the user machine. Once the reconnaissance phase is over, a second stage backdoor, like Carbon, is installed on key systems.

Technical analysis

Carbon is a sophisticated backdoor used to steal sensitive information from targets of interest by the Turla group.

This malware shares some similarities with "Uroburos" [3], a rootkit used by the same group. The most relevant resemblance is the communication framework. Indeed, both of them provide communication channels between different malware components. The communication objects are implemented in the same way, the structures and vtables look identical except that there are fewer communication channels provided in Carbon. Indeed, Carbon might be a "lite" version of Uroburos (without kernel components and without exploits).

For Turla group to decide to install Carbon on a system, a (stage 1) recognition tool is usually delivered first to the target: this tool collects several pieces of information about the victim's machine and its network (through Tavdig or Skipper for example). If the target is considered interesting enough, it will receive more sophisticated malware (such as Carbon or Uroburos).

Global architecture

The Carbon framework consists of:

- a dropper that installs the carbon components and its configuration file
- a component that communicates with the C&C
- an orchestrator that handles the tasks, dispatches them to other computers on the network and injects into a legitimate process the DLL that communicates with the C&C
- a loader that executes the orchestrator

Carbon Dating

The orchestrator and the injected library have their own development branch.

Thanks to the compilation dates and the internal versions numbers hardcoded in the PE files, we might have the following timeline:

Compilation date	Orchestrator version	Injected library version
2014-02-26	3.71	3.62
2016-02-02	3.77	4.00
2016-03-17	3.79	4.01
2016-03-24	3.79	4.01
2016-04-01	3.79	4.03
2016-08-30	3.81	????
2016-10-05	3.81	????
2016-10-21	3.81	????

Table 1 – Carbon development timeline

Carbon files

The files from the Carbon framework can have different names depending on the version but they all keep the same internal name (from the metadata) regardless of the version:

- the dropper: “SERVICE.EXE”
- the loader: “SERVICE.DLL” or “KmSvc.DLL”
- the orchestrator: “MSIMGHLP.DLL”
- the injected library: “MSXIML.DLL”

Each of these files exist in 32bit and in 64bit versions.

Working directory

Several files are created by Carbon to keep logs, tasks to execute and configuration that will modify the malware’s behavior. The contents of the majority of these files are encrypted with the CAST-128 algorithm [4].

A base working directory will contain the files/folders related to Carbon. This directory is chosen randomly among the folders in %ProgramFiles% but excluding “WindowsApps”.

The filenames are hardcoded in the orchestrator. The same names are used in the 3.7x+ branch. Because the injected library accesses the same files as the orchestrator, it is another easy way to link a library version and an orchestrator.

Carbon 3.7x files tree view:

```
\%carbon_working_folder\%   // base folder
├── 0208 // tasks results and logs files
│   ├── C_56743.NLS // contains list of files to send to the C&C server, this file is neither compressed nor encrypted
├── asmcerts.rs
├── getcerts.rs
├── miniport.dat  // configuration file
├── msximl.dll    // injected library (x32)
├── Nls // contains tasks (commands to be executed or PE file) and their configuration files
│   ├── a67ncodc.ax  // tasks to be executed by the orchestrator
│   ├── b9s3coff.ax  // tasks to be executed by the injected library
├── System   // plugins folder
│   ├── bootmisc.sdi // not used
├── qavscr.dat    // error log
├── vndkrmn.dic   // log
└── ximarsh.dll   // injected library (x64)
```

Since version 3.80, all filenames have changed.

Carbon 3.8x files tree view:

```
\carbon_working_folder\%   // base folder
├── 0409  // contains tasks (commands to be executed or PE file) and their configuration files
│   ├── cifrado.xml    // tasks to be executed by the injected library
│   ├── encodebase.inf // tasks to be executed by the orchestrator
├── 1033 // tasks results and logs files
│   ├── dsntype.gif // contains list of files to send to the C&C server, this file is neither compressed nor encrypted
├── en-US  // plugins folder
│   ├── asmlang.jpg // not used
├── fsbootfail.dat  // error log
├── mkfieldsec.dll  // injected library (x32)
├── preinsta.jpg    // log
├── wkstrend.xml    // configuration file
├── xmlrts.png
└── zcerterror.png
```

File access

In the case of the majority of the files from the Carbon working folder, when one is accessed by the malware, the following steps are taken:

- a specific mutex is used to ensure its exclusive access.
- the file is decrypted (CAST-128)
- when the operations on the file are done, the file is reencrypted (CAST-128)
- the mutex is released

Mutexes

The following mutexes are created by the orchestrator in Carbon 3.7x:

- “Global\\MSCTF.Shared.MUTEX.ZRX” (used to ensure exclusive access to “vndkrmn.dic”)
- “Global\\DBWindowsBase” (used to ensure exclusive access to “C_56743.NLS”)
- “Global\\IEFrame.LockDefaultBrowser” (used to ensure exclusive access to “b9s3coss.ax”)

- “Global\\WinSta0_DesktopSessionMut” (used to ensure exclusive access to “a67ncodc.ax”)
- “Global\{5FA3BC02-920F-D42A-68BC-04F2A75BE158}” (used to ensure exclusive access to new files created in “Nls” folder)
- “Global\\SENS.LockStarterCacheResource” (used to ensure exclusive access to “miniport.dat”)
- “Global\\ShimSharedMemoryLock” (used to ensure exclusive access to “asmcerts.rs”)

In carbon 3.8x, the filenames and the mutex names have changed:

- “Global\\Stack.Trace.Multi.TOS” (used to ensure exclusive access to “preinsta.jpg”)
- “Global\\TrackFirleSystemIntegrity” (used to ensure exclusive access to “dsntype.gif”)
- “Global\\BitswapNormalOps” (used to ensure exclusive access to “cifrado.xml”)
- “Global\\VB_crypto_library_backend” (used to ensure exclusive access to “encodebase.inf”)
- “Global\{E41B9AF4-B4E1-063B-7352-4AB6E8F355C7}” (used to ensure exclusive access to new files created in “0409” folder)
- “Global\\Exchange.Properties.B” (used to ensure exclusive access to “wkstrend.xml”)
- “Global\\DatabaseTransSecurityLock” (used to ensure exclusive access to “xmlrts.png”)

These mutexes are also used in the injected dll to ensure that the orchestrator has been executed.

Configuration File

The configuration file affects the malware’s behavior. The file format is similar to “inf” files used by Windows. It contains among others:

- an “object_id” that is a unique uuid used to identify the victim, when the value is not set in the file, it is generated randomly by the malware
- a list of processes into which code is injected (iproc)
- the frequency and time for task execution / backup logs / connection to the C&C ([TIME])
- the IP addresses of other computers on the network ([CW_LOCAL])
- the C&C server addresses ([CW_INET])
- the named pipes used to communicate with the injected library and with the other computers ([TRANSPORT])

This file might be updated later. Indeed, in the communication library, some cryptographic keys are used to encrypt/decrypt data and these keys are retrieved from a section [CRYPTO] in the configuration file that does not exist when the file is dropped from the loader resources.

Carbon 3.77 configuration file:

```
1   [NAME]
2   object_id=
3   iproc = iexplore.exe,outlook.exe,msimn.exe,firefox.exe,opera.exe,chrome.exe
4   ex = #,netscape.exe,mozilla.exe,adobeupdater.exe,chrome.exe
5
6
7   [TIME]
8   user_winmin = 1800000
9   user_winmax = 3600000
10  sys_winmin = 3600000
11  sys_winmax = 3700000
12  task_min = 20000
13  task_max = 30000
14  checkmin = 60000
15  checkmax = 70000
16  logmin =  60000
17  logmax = 120000
18  lastconnect=111
19  timestop=
20  active_con = 900000
21  time2task=3600000
22
23
24  [CW_LOCAL]
25  quantity = 0
26
27  [CW_INET]
28  quantity = 3
```

```
address1 = doctorshand.org:80:/wp-content/about/
address2 = www.lasac.eu:80:/credit_payment/url/
address3 = www.shoppingexpert.it:80:/wp-content/gallery/

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


[DHCP]
server = 135


[LOG]
logperiod = 7200

[WORKDATA]
run_task=
run_task_system=
```

Logfile

The Carbon framework includes a logfile that is used to log actions performed by the malware and information on the system that can be useful to the malware operator (for example if an analysis tool such as WireShark is running on the machine).

The log's format has not changed since Carbon 3.71:

```
Date|Time|Object-Id|Source|Message
```

example

```
1 [LOG]
2 start=1
3 20/02/17|12:48:24|8hTdJtUBB57ieReZAOSgUYacts|s|OPER|New object ID generated
  '8hTdJtUBB57ieReZAOSgUYacts'|
4
  20/02/17|12:48:24|8hTdJtUBB57ieReZAOSgUYacts|s|ST|3/81|0|
5
  20/02/17|12:48:24|8hTdJtUBB57ieReZAOSgUYacts|s|START OK
```

This file is periodically backed up and sent to the C&C.

Dropper

The dropper is the only executable that is not a DLL. It is the first PE file to be executed: it is used to extract the other components from its resources.

The PE files that are used to load the main components are extracted into the Windows system directory while the orchestrator, the library used to communicate with the C&C and the configuration file are extracted into the Carbon working directory.

A new section is appended into a random “.inf” file from %SystemRoot%\INF. The section’s name is the volume serial disk number of the compromised machine and a value “root” is created with the chosen Carbon working directory.

Example:

```
1 [5049654F]
2 root="C:\Program Files\Windows Portable Devices"
```

Loader

This part of the component is used to load the orchestrator.

A service that ensures Carbon’s persistency is created. Its name can either be “srservice”, “ipvpn” or “hkmsvc” depending of the operating system version running on the compromised machine.

The Carbon working directory is retrieved by walking through the “%windir%\inf” folder and looking for the file that contains the Carbon base path.

Last but not least, the function “ModuleStart” (in Carbon 3.71) or “ModStart” (since Carbon 3.77) from the orchestrator (located in the Carbon base folder) is called.

Orchestrator

The orchestrator is the main component of the Carbon framework. It is mainly used to inject code into a process that communicates legitimately over the Internet and to dispatch the tasks received from the injected library to other computers on the same network either through named pipes or TCP.

Seven threads are created by the malware. It is easy to identify Carbon’s characteristics because each thread has a specific role:

Configuration fetching

Because the configuration file can be updated by the malware, some attributes like the C&C server addresses are monitored every 10 minutes.

Check Carbon storage folder periodically

There is a storage folder located in the Carbon working directory. This folder contains some files downloaded from the C&C server (tasks that are either commands to be executed or PE files, and their configuration files).

This thread will run continuously and check every two hours [5] whether there is still enough space available in this folder; if not, a notification is written into the logfile.

Task execution

The execution of the tasks in the context of the orchestrator process is very similar to the way in which it is performed in the communication library (cf Communication library / Tasks execution).

Unlike the communication library, it is the file “encodebase.inf” (for Carbon v3.8x) or “a67ncode.ax” that contains the list of the tasks to execute.

Each line of this file is composed in the following way:

> task_id | task_filepath | task_config_filepath | task_result_filepath | task_log_filepath | [execution_mode | username | password]

The five first fields are required, while the last three are optional. If the field “execution_mode” exists, its value will affect the way the task is executed:

- 0 or 1: normal execution

- 2: the task is executed in the security context of a specific user (credentials are provided through the username/password fields)
- 3 or 4: the task is executed in the security context of the user represented by the “explorer.exe” token

P2P

Like Uroburos/Snake, Carbon can dispatch tasks to other computers from the same network via named pipe or TCP. It is useful to be able to dispatch and execute tasks on computers that do not have Internet access.

Communication channels

Uroburos used several types of communication transports than can be categorized as follows:

- type 1: TCP
- type 2: enc, np, reliable, frag, m2b, m2d
- type 3: t2m
- type 4: UDP, doms, domc

```
.data:00079018 off_79018       dd offset aTcp          ; DATA XREF: .data:off_790BC↓o
.data:00079018                                         ; "tcp"
.data:0007901C                 dd 1
.data:00079020                 dd offset handler_tcp
.data:00079024                 dd offset aEnc          ; "enc"
.data:00079028                 dd 2
.data:0007902C                 dd offset handler_enc
.data:00079030                 dd offset aNp           ; "np"
.data:00079034                 dd 2
.data:00079038                 dd offset handler_np
.data:0007903C                 dd offset aReliable     ; "reliable"
.data:00079040                 dd 2
.data:00079044                 dd offset handler_reliable
.data:00079048                 dd offset aFrag         ; "frag"
.data:0007904C                 dd 2
.data:00079050                 dd offset handler_frag
.data:00079054                 dd offset aUdp          ; "udp"
.data:00079058                 dd 4
.data:0007905C                 dd offset handler_udp
.data:00079060                 dd offset aM2b          ; "m2b"
.data:00079064                 dd 2
.data:00079068                 dd offset handler_m2b
.data:0007906C                 dd offset aT2m          ; "t2m"
.data:00079070                 dd 3
.data:00079074                 dd offset handler_t2m
.data:00079078                 dd offset aM2d          ; "m2d"
.data:0007907C                 dd 2
.data:00079080                 dd offset handler_m2d
.data:00079084                 dd offset aDoms         ; "doms"
.data:00079088                 dd 4
.data:0007908C                 dd offset handler_doms
.data:00079090                 dd offset aDomc         ; "domc"
.data:00079094                 dd 4
.data:00079098                 dd offset handler_domc
```

Carbon uses a reduced number of communication channels:

- type 1: TCP, b2m
- type 2: np, frag, m2b

```
off_20021100    dd offset aTcp           ; DATA XREF: .data:off_20021154↓o
                                         ; "tcp"
                dd 1
                dd offset handler_tcp
                dd offset aNp            ; "np"
                dd 2
                dd offset handler_np
                dd offset aFrag          ; "frag"
                dd 2
                dd offset handler_frag
                dd offset aM2b           ; "m2b"
                dd 2
                dd offset handler_m2b
                dd offset aB2m           ; "b2m"
                dd 1
                dd offset handler_b2m
```

The data sent to peers are usually fragmented and transported either by TCP or via a named pipe. If, for example, fragmented data are sent from a computer to another one by a named pipe, an object “frag.np” is set up. In this case the mother class “frag” constructor will be called followed by a call to the constructor subclass “np”.

There is a structure composed of several handlers for each objects: initialize communication, connection (to a pipe / IP address), read data, send data etc.

How a task is forwarded to another computer

Several steps are performed to send data from one computer to another:

- a communication channel is created (frag.np or frag.tcp object) with a specific named pipe / ip address
- options are given to the object communication (for example : the fragment’s size, information about the peer etc.)
- connection to the peer
- an authentication step is performed between the host and the peer:
    - there is a handshake process where the host is sending the “magic” value “A110EAD1EAF5FA11” and expects to receive “C001DA42DEAD2DA4” from the peer
    - a command “WHO” is sent to the peer where the host sends the victim uuid and expects to receive the same uuid
- if the authentication was successful, the data are sent to the peer

All the communication between the host and the peer are encrypted with CAST-128

Note that this P2P feature is also implemented in the communication DLL.

Plugins

This malware supports additional plugins to extend its functionalities.

In the configuration file, there is a section named “PLUGINS”. It might not exist when the configuration file is dropped from the loader resources but this file can be updated by the malware. The section “PLUGINS” contains a line formed this way:

%plugin_name%=%enabled%|%mode%[:%username%:%password%]|%file_path%

%file_path% can be either the path to a PE file or to a file containing a command line to be executed. %enabled% is a string that is used to know if the plugin has to be executed. If it is the case, that string value is “enabled”.

The attribute %mode% is used to control the context in which to execute the PE file/command line. It can be either:

- 1 = execution with current user privilege in the current process context through CreateProcess().
- 2 = execution as the user specified in the configuration (:%username%:%password% attributes), the token of this specific user is retrieved through the LogonUserAs() function.
- 3 = execution in the security context of the user represented by the “explorer.exe” token (the token of the process “explorer.exe” is duplicated and passed through the CreateProcessAsUser() function.
- 4 = similar than 3 but the environment variables for the user represented by the “explorer.exe” token are retrieved and passed to the function CreateProcessAsUser()

If it is a PE file:

- the file is loaded into the malware process memory
- the module is parsed to check if it is a DLL
- if the module is a DLL and exports a function “ModStart” (since Carbon 3.77) or “ModuleStart” (for older versions of Carbon), a new thread is created to execute this function.
- if the module is not a DLL but a valid PE, it is executed from the entry point.

Injection of the communication library into remote processes

The library that is used to communicate with the C&C server is injected into remote processes. In order to know where to inject this DLL, the configuration file is parsed. The section “[NAME]” contains a field “iproc” containing a list of processes that can legitimately communicate to Internet.

Example:

```
1 [NAME]
2 iproc = iexplore.exe,outlook.exe,msimn.exe,firefox.exe,opera.exe,chrome.exe
```

For each process on the list that is running on the system, if its parent process name is either “explorer.exe” or “ieuser.exe”, the DLL will be injected into this process.
The process injection is very classical:

- the functions “CreateToolHelp32Snapshot / Module32FirstW / Module32NextW” are used to retrieve the base address of the module “kernel32.dll”
- the module EAT is parsed to get the address of the function “LoadLibraryW”
- the privilege “SeDebugPrivilege” is enabled for the current process
- memory is allocated into the remote process and the library path is written into it
- NtCreateThreadEx or CreateRemoteThread (if the address of the first function cannot be retrieved) is called to execute LoadLibraryW to load the DLL into the memory of the remote process *

Communication library

The following analysis is based on the version 4.x of msximl. This component may have changed in the latest versions.

Configuration fetching

Besides the code in the “Configuration fetching” thread from the orchestrator (which is similar), a field “sethttp1” is retrieved from the [TRANSPORT] section.

If this value is set, HTTP 1.1 will be used for future connections.

Tasks execution

The tasks are retrieved from the C&C server.

The tasks to be executed by the communication library are listed in the file “b9s3coff.ax” (for Carbon v3.7x) or “cifrado.xml” (for Carbon v3.8x).

Each line of this file is composed in the following way:

task_id | task_filepath | task_config_filepath | task_result_filepath | task_log_filepath

The task file and its config are decrypted (CAST-128) and the task executed. There are some options that are retrieved from the Carbon configuration file: “time2task” is used to set a timeout for the task execution (1 hour by default) while “task_min” and “task_max” are used as to wait a random time between the execution of the tasks on the task list (the random time will be set between “task_min” and “task_max”). These parameters come from the section [TIME].

If the task is a valid DLL file, it is loaded in the current process memory and a new thread is created to execute its “start” export. Otherwise, this is probably a command to execute. In this case, the configuration file is parsed. Like the Carbon configuration file, the task configuration file is formed as a windows “inf” file and should contain a section [CONFIG] with the following parameters:

[CONFIG]
- NAME (“cmd.exe” by default)
- ARG
- RESULT (“stdout” by default)
- COMPRESSION (“yes” by default)
- DELETE (“no” by default)

The command with its arguments is executed through CreateProcess() function and the results are stored in the %task_result_filepath% file if the RESULT option is not set to “stdout”, while error or additional info are added to the task logfile.

If the option RESULT is set to stdout:

- the result task file is deleted and the task execution output is written to the log task file
- the line “task_id | “1” | task_log_filepath | object_id” is added at the end of the file “C_56743.NLS” (Carbon 3.7x) or “dsntype.gif” (Carbon 3.8x)

Otherwise:

- the task execution is outputted to the task result file and the errors / additional info to the task logfile
- the line “task_id | “2” | task_log_filepath | task_result_filepath | object_id” is added at the end of the file “C_56743.NLS” (Carbon 3.7x) or “dsntype.gif” (Carbon 3.8x)

In both cases, the same line as written into “C_56743.NLS” is also written into the field “run_task” from the Carbon configuration file. It might be intended as a way for the malware authors to know which is the last task executed when the configuration file is sent to the C&C server (check “Configuration file backup”).

Log rotation

The logfile is backed up and sent to the C&C server periodically (by default this is done every two hours).

Like the backup of the configuration file, this action is performed only during specific times of the day.
Indeed, the log rotation will be performed only when the current local time is in the range of what is defined in the configuration file.

The fields “wh_min” and “wh_max” are retrieved from the [TIME] section of the configuration file (by default: 8 and 20). The thread will execute the code only if the current hour of the day is between %wh_min% and %wh_max%.

If there is a value for the attribute “timestop” from the [TIME] section (which looks like “wDay:wMonth:wYear:wHour:wMinute”), the thread will execute the code only after this specific datetime.

The attributes “lastsend” and “logperiod” from the [LOG] section are used to specify a delay time when to backup and send the log to the C&C server. (by default: every two hours).

A temporary file with a random name prefixed by “~D” is created in the folder “208” (for Carbon v3.7x) or “1033” (for Carbon v3.8x). The logfile content is copied into this new file. It is then compressed with Bzip2 and encrypted (CAST-128).

A new line is added at the end of the file “C_56743.NLS” (for Carbon v3.7x) or “dsntype.gif” (for Carbon v3.8x):

“10|1|%s|%s”
- 1st field: an ID to identify the file as a logfile
- 2nd field: 1 (file to be sent to the C&C server)
- 3rd field: the temp file path
- 4rd field: the victim uuid

Last but not least, the attribute “lastsend” is updated with the current time and the original logfile is deleted.

Communication with the C&C server

The code of this thread is used to retrieve new tasks from the C&C server, to send new files to the server (the files listed in the file “C_56743.NLS” / “dsntype.gif”) and to send the new tasks to the orchestrator.

First request

A random C&C server address is chosen from the ones in the section “CW_INET”. If the port and HTTP resource path are not specified, the default is to use port 80 and “/javascript/view.php”.

A user agent is set up in the following way:

- the version of Internet Explorer is retrieved through the registry key: “HKLM\Software\Microsoft\Internet Explorer\Version” and is concatenated to the string “Mozilla/4.0 (compatible; MSIE %d.0; ”
  example: “Mozilla/4.0 (compatible; MSIE 8.0.6001.18702.0;”
- concatenate the previous string with the OS major/minor version values (through GetVersionExA())
  “Mozilla/4.0 (compatible; MSIE 8.0.6001.18702.0; Windows NT 5.1; Trident/4.0”
- enumerate the values key in “HKLM\Software\Microsoft\Windows\CurrentVersion\Internet Settings\5.0\User Agent\Post Platform” and concatenate each value to the previous string and then append a closing paren.
  example: “Mozilla/4.0 (compatible; MSIE 8.0.6001.18702.0; Windows NT 5.1; Trident/4.0; .NET CLR 2.0.50727; .NET CLR 3.0.30729; .NET CLR 3.5.30729; .NET4.0C; .NET4.0E; Media Center PC 6.0; SLCC2)

The field “trans_timemax” from the section [TIME] is retrieved. It is used to set the timeout for internet requests (through InternetSetOption()). It has a value of 10 minutes by default.

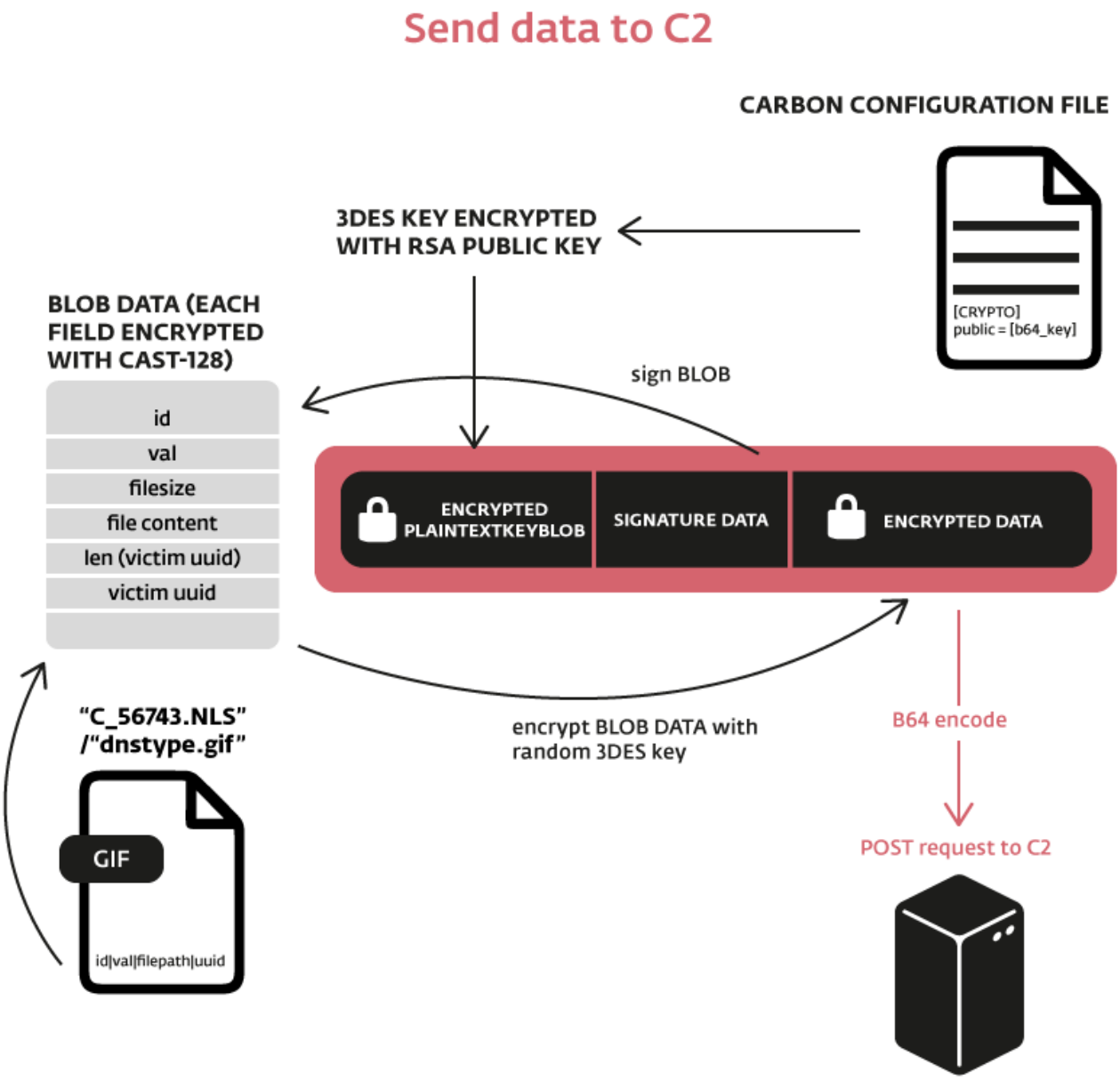
A first GET request is performed on the root page of the C&C web server to check that the host is alive. If no packet capture is running on the system, a new request is done on the C&C server to check if new tasks are available. A “PHPSESSID” cookie is added to the request with the victim uuid as its value. A header “Referer” is added as well and set to the C&C server URL.

The malware is expecting to get an answer to the GET request similar to:

<input name=”%name%” value=”%data_in_b64%”>

If the field “value” contains something, a new task is available.

Send data to the server



If the file “C_56743.NLS” / “dsntype.gif” is not empty, it means there are data to be sent the C&C server. The file is parsed and the last line is retrieved. It contains details about the data to be sent. A data blob is built and each of the following fields is encrypted with CAST-128:

id | val | tmp_filesize | tmp_content | [OPTIONAL (if val == 2) tmp2_filesize | tmp2_content] | len_object_id | object_id

- id = the type of data to send to the C&C server, it can be:
  - 10: log backup
  - 11: configuration file
  - 20: a cryptographic key
  - otherwise: an id associated to a task, it can be the result of a task or an error log in the case of task execution failure
- val = 1 if there is only one file to send, 2 if there are two files
- object_id = the victim uuid

If the field “dtc” from the section [CRYPTO] of the configuration file is set to 0, this whole blob is base64 encoded and sent to the C&C server through a POST request.

Otherwise, another layer of encryption is used. In this case, the data blob is signed and a random 3DES key is used to encrypt it. Because the 3DES key is randomly generated and the server needs it to decrypt the data, the key is encrypted with the server public key. The server key is retrieved from the field “publicc” of the section [CRYPTO] from the configuration file.

This new blob (encrypted_key | signature_data | encrypted data) is encoded in base64 and sent to the C&C server through a POST request.

In order to avoid detection based on the data size sent in a request, the blob can be fragmented into several packets. An option in the configuration file (“post_frag” in the section [TRANSPORT]) defines whether the blob will be fragmented or sent in only one POST request.

If this option is set to “yes”, the blob is divided into several fragments of a specific size. This size comes from another field in the configuration file: “post_frag_size”.

An additional header will be added to the request:

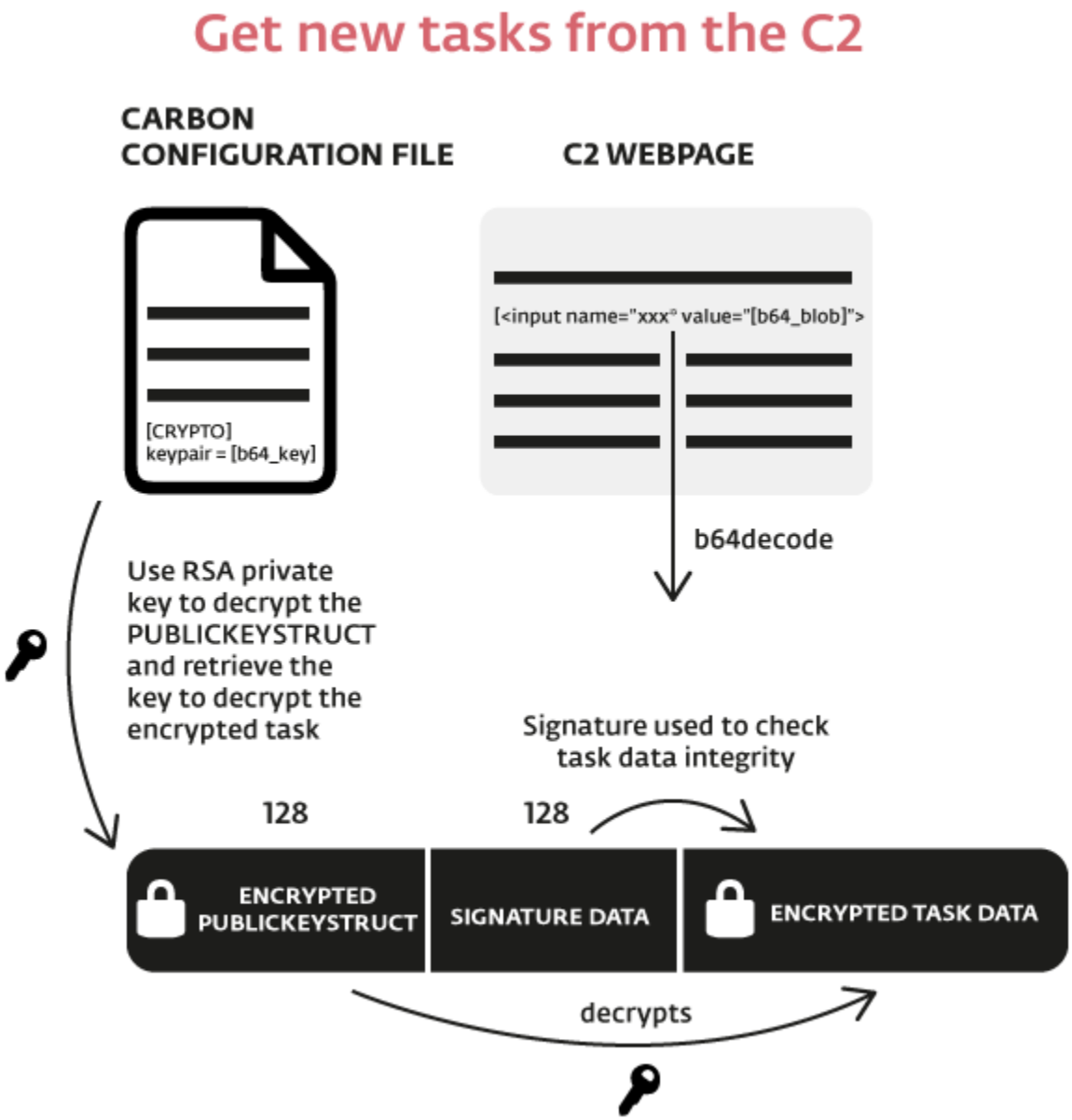
> “Content-Range: bytes %u-%u/%u; id=%u\r\n”, i, i+(fragment_size-1), data_size, task_id”

If the option http11 is set, a specific header is added as well:

> “Expect: 100-continue\r\n”

For each fragments sent, the fields “post_frag_size” and “pfslastset” from the config file (section [CW_INET_RESULTS]) are updated with the fragment size and the timestamp.

Get new tasks



New tasks are retrieved from the C&C server by parsing the html page. The malware expects to find the html tag <input> in the page with a base64 encoded blob in its “value” attribute. Once decoded, this blob contains:

- an encrypted block of 128 bytes that contains a structure “PUBLICKEYSTRUC” followed by a cryptographic key (probably a 3DES key)
- signature data (128 bytes) to verify the integrity of the next block
- a block of encrypted data that contains the task

The malware uses an RSA private key (retrieved from the field “keypair” from the section [CRYPTO] of the configuration file) to decrypt the first block and then uses the freshly decrypted key to decrypt the third block. This block of data can be either:

- a task to be executed [6]
    - the data are decrypted and stored in few temporary files, the task (a command or a PE file) and its configuration file in the “Nls” folder and the output files (file that will contains the task results and the logfile) are stored in the folder “0208” (or respectively “0409” and “1033” for Carbon v3.8x)
    - task_id | task_filepath | task_config_filepath | task_result_filepath | task_log_filepath
    - this line is appended to beginning of the file “b9s3coff.ax” (cifrado.xml on v3.8x)
- a task to be executed by the orchestrator [7]
    - the data are decrypted and stored in few temporary files (the task, its configuration etc) in the “Nls” and “0208” folder (or “0409” and “1033” for Carbon v3.8x)
    - depending of the content of the data, one of these lines will be added to the beginning of the file “a67ncode.ax” (encodebase.info on v3.8x)
        - task_id | task_filepath | task_config_filepath | task_result_filepath | task_log_filepath
        - task_id | task_filepath | task_config_filepath | task_result_filepath | task_log_filepath | execution_mode | username | password
        - task_id | task_filepath | task_config_filepath | task_result_filepath | task_log_filepath | execution_mode
- a new RSA server public key

  in this case, the configuration file is updated with the new key encoded in base64 (field publicc)
- data to be sent to an instance of Carbon running in another computer in the same network

  the data can contains a specific IP address and port, a named pipe or a named pipe with a username and password.

Check Internet availability

Each hour, the internet connection is checked. A first check is done by calling the function InternetAttemptConnect(). If it works, another test is done by sending HTTP GET requests to the following websites:

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

An event is used to notify the other threads in case of the loss of Internet access.

Configuration file backup

Similar to the logfile, the configuration file is also periodically backed up and sent to the C&C server. The thread executes the code in a specific range of time (between 8h and 20h by default) [8].

The value “configlastsend” is retrieved from the section [TIME] of the configuration file. If the config file has been sent over a month ago, the config file is copied into a temporary file with a random name prefixed by “~D” in the folder “208” (for Carbon v3.7x) or “1033” (for Carbon v3.8x). This file is then encrypted with CAST-128 algorithm.

To notify the thread that communicates with the C&C server that a new file is ready to be sent to the server, the following line is appending to the file “C_56743.NLS” (for Carbon v3.7x) or “dsntype.gif” (for Carbon v3.8x):

“11|1|%s|%s”

- 1st field: an ID to identify the file as a config file
- 2nd field: 1 (file to be sent to the C&C server)
- 3rd field: the temp filepath
- 4rd field: the victim uuid

Last but not least, the attribute “configlastsend” is updated with the current time.

Additional Notes

Calling API functions

The base address of the modules of interest are retrieved by either parsing the PEB or (if the modules are not loaded into the process memory) by loading the needed files from disk into memory and parsing their headers to get their base addresses.

Once the base addresses are retrieved, the PEB is walked again and the field “LoadCount” from the structure LDR_DATA_TABLE_ENTRY is checked. This value is used as a reference counter, to track the loading and unloading of a module.

If “LoadCount” is positive, the module EAT is parsed to get the needed function address.

Encryption

The module and function names are encrypted (at least since v3.77; it was not the case in v3.71) in a simple way, a logical shift of 1 bit being applied to each characters.

The processes’ names are encrypted as well by just XOR’ing each character with the key 0x55 (for Carbon v3.7x at least since v3.77) and with the key 0x77 for Carbon v3.8x.

With only a few the exceptions, each file from the Carbon working directory is encrypted with the CAST-128 algorithm in OFB mode. The same key and IV are used from the version 3.71 until the version 3.81:

- key = “\x12\x34\x56\x78\x9A\xBC\xDE\xF0\xFE\xFC\xBA\x98\x76\x54\x32\x10”
- IV = “\x12\x34\x56\x78\x9A\xBC\xDE\xF0”

Check if packet capture is running

Before communicating with the C&C server or with other computers, the malware ensures that none of the most common packet capture software is running on the system:

- TCPdump.exe
- windump.exe
- ethereal.exe
- wireshark.exe
- ettercap.exe
- snoop.exe
- dsniff.exe

If any of these processes are running, no communication will be done.

Carbon IoCs are also available on ESET’s GitHub repository https://github.com/eset/malware-ioc/tree/master/turla

Appendices

Yara rules

```
1   import "pe"
2   import "hash"
3
4   rule generic_carbon
5   {
6     strings:
7       $s1 = "ModStart"
8       $s2 = "STOP|OK"
9       $s3 = "STOP|KILL"
10    condition:
```

```
11      (uint16(0) == 0x5a4d) and all of them
12  }
13
14  rule carbon_metadata
15  {
16      condition:
17          (pe.version_info["InternalName"] contains "SERVICE.EXE" or
18           pe.version_info["InternalName"] contains "MSIMGHLP.DLL" or
19           pe.version_info["InternalName"] contains "MSXIML.DLL")
20           and pe.version_info["CompanyName"] contains "Microsoft Corporation"
21           and not (tags contains "signed")
22  }
23
24  rule carbon_2016_filenames
25  {
26      condition:
27          file_name contains "wkstrend.xml" or
28          file_name contains "cifrado.xml" or
29          file_name contains "fsbootfail.dat" or
30          file_name contains "encodebase.inf" or
31          file_name contains "zcerterror.png" or
32          file_name contains "mkfieldsec.dll"
33  }
```

Carbon files decryptor/encryptor

carbon_tool.py

```
1   #!/usr/bin/env python2
2
```

```
3  from Crypto.Cipher import CAST
4  import sys
5  import argparse
6
7
8  def main():
9
10 parser =
   argparse.ArgumentParser(formatter_class=argparse.RawTextHelpFormatter)
11
   parser.add_argument("-e", "--encrypt", help="encrypt carbon file", required=False)
12
   parser.add_argument("-d", "--decrypt", help="decrypt carbon file", required=False)
13
14
   try:
15
   args = parser.parse_args()
16
   except IOError as e:
17
   parser.error(e)
18
   return 0
19
20
   if len(sys.argv) != 3:
21
   parser.print_help()
22
   return 0
23
24
   key = "\x12\x34\x56\x78\x9A\xBC\xDE\xF0\xFE\xFC\xBA\x98\x76\x54\x32\x10"
25
   iv = "\x12\x34\x56\x78\x9A\xBC\xDE\xF0"
26
27
   cipher = CAST.new(key, CAST.MODE_OFB, iv)
28
29
   if args.encrypt:
30
```

```
plaintext = open(args.encrypt, "rb").read()
while len(plaintext) % 8 != 0:
plaintext += "\x00"
data = cipher.encrypt(plaintext)
open(args.encrypt + "_encrypted", "wb").write(data)
else:
ciphertext = open(args.decrypt, "rb").read()
while len(ciphertext) % 8 != 0:
ciphertext += "\x00"
data = cipher.decrypt(ciphertext)
open(args.decrypt + "_decrypted", "wb").write(data)

if __name__ == "__main__":
main()
```

Carbon footprint

Table 2 – Carbon sample hashes

SHA1 hash
7f3a60613a3bdb5f1f8616e6ca469d3b78b1b45b
a08b8371ead1919500a4759c2f46553620d5a9d9
4636dccac5acf1d95a474747bb7bcd9b1a506cc3
cbde204e7641830017bb84b89223131b2126bc46
1ad46547e3dc264f940bf62df455b26e65b0101f
a28164de29e51f154be12d163ce5818fceb69233
7c43f5df784bf50423620d8f1c96e43d8d9a9b28
7ce746bb988cb3b7e64f08174bdb02938555ea53
20393222d4eb1ba72a6536f7e67e139aadfa47fe

SHA1 hash
1dbfcb9005abb2c83ffa6a3127257a009612798c
2f7e335e092e04f3f4734b60c5345003d10aa15d
311f399c299741e80db8bec65bbf4b56109eedaf
fbc43636e3c9378162f3b9712cb6d87bd48ddbd3
554f59c1578f4ee77dbba6a23507401359a59f23
2227fd6fc9d669a9b66c59593533750477669557
87d718f2d6e46c53490c6a22de399c13f05336f0
1b233af41106d7915f6fa6fd1448b7f070b47eb3
851e538357598ed96f0123b47694e25c2d52552b
744b43d8c0fe8b217acf0494ad992df6d5191ed9
bcf52240cc7940185ce424224d39564257610340
777e2695ae408e1578a16991373144333732c3f6
56b5627debb93790fdbcc9ecbffc3260adeafbab
678d486e21b001deb58353ca0255e3e5678f9614

Table 3 – C&C server addresses (hacked websites used as 1st level of proxies

C&C server address
soheylistore.ir:80:/modules/mod_feed/feed.php
tazohor.com:80:/wp-includes/feed-rss-comments.php
jucheafrica.com:80:/wp-includes/class-wp-edit.php
61paris.fr:80:/wp-includes/ms-set.php
doctorshand.org:80:/wp-content/about/
www.lasac.eu:80:/credit_payment/url/

Notes