

Collecting Malicious Particles from Neutrino Botnets

Jakub Souček¹, Jakub Tomanek¹ and Peter Kálnai¹

¹ESET, Czech Republic

It is published in the Journal on Cybercrime & Digital Investigations by CECyF, <https://journal.cecyp.fr/ojs>
© It is shared under the CC BY license <http://creativecommons.org/licenses/by/4.0/>.

Abstract

Neutrino Bot (also known and detected as Win/Kasidet) is a rapidly changing threat. It first became known around December 2013 [1]. It has been actively developed ever since resulting in version 5.4 at the very beginning of 2018. It is being sold for an attractive price to a large variety of cybercriminals.

This paper shows an extensive summary of the history of the bot while focusing on the most recent versions. It presents methods how to analyse Neutrino botnets and provides key findings that have been discovered during the year 2018.

Keywords: Neutrino Bot, Kasidet, bot, botnet, reverse engineering.

1 Introduction

Neutrino Bot started as a worm being able to spread to removable drives and RAR archives. At that time, its capabilities were mainly DDoS attacks, yet even then the bot could utilize keylogger activities, perform simple redirection or execute files from the Internet. Since then, it obtained some new tricks and got rid of a part of the old ones. The main change over the years has been the ability to manipulate network traffic, acquiring it the status of a banking trojan. We provide a brief summary of the historical progress in the bot while focusing on the most recent versions to give a good understanding of how the bot works today.

Neutrino Bot is being sold to a large variety of cybercriminals. Because of its affordability, there are many independent actors using the bot, each in a very different way. Therefore, it is useful to be able to separate the acting groups from one another and track each one separately. We present what information the bot leaks that is valuable to distinguishing different botnets. We also introduce the most interesting Neutrino

botnets we have discovered during 2018. For each one, we present what is typical for it, how do the actors use the bot and what have they achieved.

2 History

The code of Neutrino Bot has changed almost completely over the years. The author(s) have modified its structure, stealth techniques, control flow, functionality and even persistence methods.

The bot uses versioning but it is necessary to say that it is not very strict. That results in situations where substantial changes have been made, yet the version number did not change. Still, it gives an analyst a good overview of how Neutrino Bot looked throughout the years. We have concluded all the important milestones in this bot's history in Appendix 1. We have also provided the hashes of the collected samples here [2].

3 More recent history

The long history of Neutrino Bot brought many changes. Let's focus on the more recent history in detail. Before we go any further, it is necessary to say a few things about some fundamental basics Neutrino Bot stands on.

3.1 Neutrino Bot fundamental basics

Each sample contains four main pieces of information: a version, a bot name, a list of C&C servers and a build id. We explain what a build id is later, for now, just think of it as an alphanumeric character string that identifies the one specific build.

Value name	Data
R	rate at which the bot requests new commands
D	date when the bot has been executed last
I	webinjects configuration
Dns	DNS redirection configuration
%CMD_ID%	indication that a command with that ID is being processed

Table 1: Neutrino Bot Windows registry hive configuration.

Neutrino Bot also creates a Windows registry hive for itself inside `HKCU\Software\%BUILD_ID%`. All configuration that should stay persistent is stored there. Table 1 summarizes the possible values that can be stored there.

The network protocol has been described many times before [3] and did not change substantially.

3.2 Version 5.0 - Starting point

Around June 2016 version 5.0 has appeared and as the numbering suggests, it brought some substantial changes. Let's take it as a starting point and observe how it works.

This version is the first one using a modular structure we have discovered. It splits the malicious actions into two parts (a *dropper* and a *module*), resulting in a control flow like Fig. 1 shows.

This flow graph is, with some modifications, applicable to all further versions. Table 2 describes the basic behaviour of all the phases.

Now, let's look at what these stages actually do for this version. The Anti-Emulation phase is very robust and will probably only grow in the future. It scans the system for widely used virtualization software (VMWare, Virtual Box), debuggers (WinDbg, OllyDbg), tools (Process Monitor, Process Explorer) and others. It uses an extensive set of methods how to identify these targets, namely

- Open device tests
- Windows Registry fragments check
- GetProcAddress

¹POS = Point of Sale

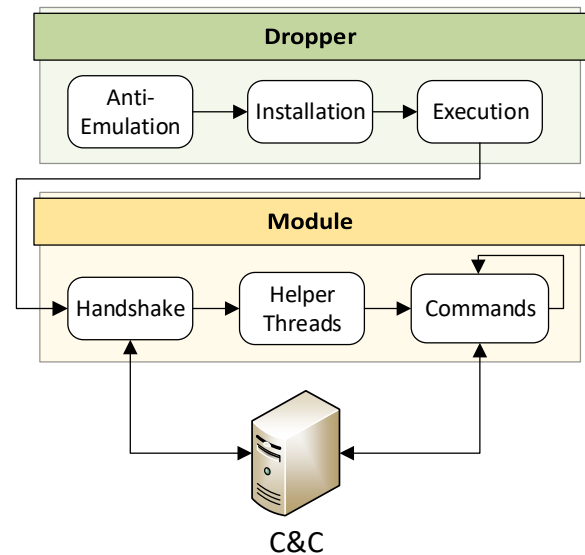


Figure 1: Control flow in version 5.0.

- IsDebuggerPresent
- CheckRemoteDebuggerPresent
- Timing check
- Modules check
- Window names check
- File name check

The Installation phase installs the bot to `%APPDATA%\%BUILD_ID%\%NAME%`. The algorithm to create the install filename is quite interesting and we provide a pseudocode in Appendix 2. Persistence is done by utilizing the 'Run' key. To ensure the entry won't be easily removed, it uses the `RegNotifyChangeKey` API.

The Execution phase then decompresses the *module* and decodes it using Base64 (the *module* is part of the *dropper's* data). It then launches a new process from the *dropper* executable and injects the *module* to it.

The Handshake phase verifies the C&C server by sending the "enter" command and receiving a "success" response. The addresses of C&C servers are stored in the binary encoded using Base64.

The Helper Threads are a very interesting part of the bot. In the case of this version we recognize two. The first one is a *Network Data Stealer* that hooks web browsers in order to exfiltrate outgoing traffic. The second one is a *Credit Card Scraper*; a thread that tries to extract credit card numbers from memory. This can be a powerful functionality if Neutrino Bot were to infect a POS¹ system [4] [5]. It utilizes the Luhn algorithm [6] to verify the validity of an extracted credit card.

Phase	Description
Anti-Emulation	Performs checks to discover a virtual machine, sandbox, debugger or other kind of emulation and analysis techniques.
Installation	Chooses the installation path and filename. Sets up persistence.
Execution	Obtains the <i>module</i> , decrypts it if necessary and executes it.
Handshake	Verifies the availability and validity of a C&C server.
Helper Threads	Creates a set of threads that either help other parts of the bot or offer supportive functionality.
Commands	Obtains commands and executes them. After a specific time interval, requests new ones.

Table 2: Description of different phases in Neutrino Bot version 5.0.

Command name	Meaning
rate	Change the rate of the following commands requests.
FINDPROC	Check if a process with a name matching provided substring(s) is running.
PLUGIN	Download a file and execute it via injection.
LOADER	Download a file and execute it.
screenshot	Take a screenshot and upload it to the C&C.
CMD	Execute a command via cmd.exe.
DNS	Obtain an entry in the form of (source, destination). Spoof the DNS resolving process so that the source domain is resolved to the destination IP.
UPDATE	Download an update file and update the bot.
FIND	Find any file(s) with names matching provided substring(s) and upload them to the C&C.
PROXY	Set up connection to a proxy server.

Table 3: Command capabilities of version 5.0.

As for the commands the bot recognizes, Table 3 summarizes the capabilities of the 5.0 version. The case sensitivity of the commands is important. Neutrino Bot calculates a checksum of the command name, compares it to a set of known checksums and decides what command to execute based on that.

The bot also utilizes randomized sleep intervals between actions. It does so in order to fool automated analysis. The technique can be found especially in the *dropper* and is more elaborated here [7].

3.3 Version 5.1 - Hardening the analysis

This version brought one major update – it extended the character string comparison by checksum to all the strings in the Anti-Emulation phase, making it very hard to determine any new targets. We checked that the majority of the targets match the ones in previous versions but new ones have occurred as well.

The Installation phase changed the installation filename generation (see Appendix 2). It also added some

new tricks to make the bot stealthier. Namely

- adds a firewall exception for itself
- disables showing of hidden files in Windows Explorer
- disables Windows SmartScreen

As for the commands, it added the CMD-Result command which does the same as CMD but reports the command result to the C&C server. It also changed the PLUGIN command to download the Ammy Remote Admin plugin. The connection with Ammy is very interesting because, as you will see shortly, Ammy has been used in one of the observed infection chains.

Finally, it is crucial to say that it is well known that this version has been cracked and a builder for it is available online. It is not an official one, just a tool that patches the URLs in the binary. This is the main reason why we still see version 5.1 builds active and even used in campaigns.

Phase	Description
Modules	Downloads a list of URLs to download <i>modules</i> from. Downloads a <i>module</i> from each of them.
Webinjects	Downloads a JSON webinjects configuration and stores it in the registry hive.

Table 4: Description of new phases in version 5.3.

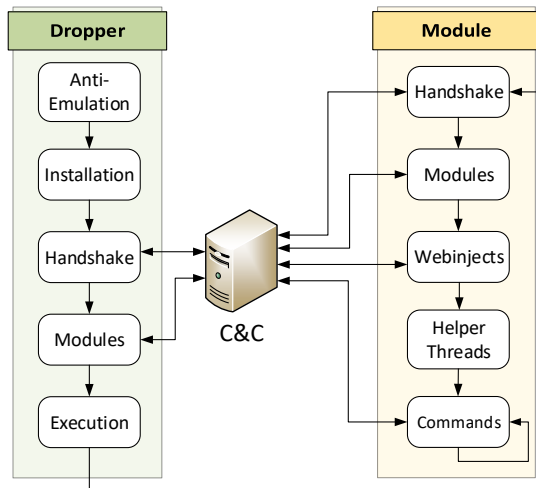


Figure 2: New control flow in version 5.3.

```

key_RC4 = Mem::Load(KeyRC4_enc, 0x20u);
for ( i = 0; i < 0x20; ++i )
    key_RC4[i] ^= 7u;
CnC_b64 = Mem::Load(CnC_enc, 0x150u);
for ( j = 0; j < 0x150; ++j )
    CnC_b64[j] ^= 7u;
decDataLen = 0;
decData = Crypto::FromBase64W(
    CnC_b64,
    0x150u,
    &decDataLen);

```

Figure 3: C&C addresses and an RC4 key stored inside the binary (version 5.3). Both values are additionally protected by a single byte XOR key 0x07.

3.4 Version 5.2 - Minor update

This version is very similar to version 5.1 and adds mostly minor changes. It introduces more stealth tricks:

- Changes the timeouts of `HKLM\SOFTWARE\Policies\Microsoft\Windows NT\Terminal Services` to 0.
- Tries to cripple Windows Defender and adds itself to its exclusion paths.

As for persistence, it prefers a scheduled task and writes to the 'Run' key only if the task creation fails.

The *module* introduces one new command – botkiller. This command is used to delete files and kill processes of executables that might be related to other malware.

3.5 Version 5.3 - Where things got interesting

This version is a major update from the previous one. Mainly, it modifies the control flow. It is now more complicated and new phases have been added (see Fig. 2). The new phases are described in Table 4.

The main change, as you can see, is that the *dropper* received more responsibility. Now it performed the

handshake and downloaded the *modules* instead of having the *module* stored inside its own data. This version also brought the support of 64-bit systems (therefore the phase is named "Modules" instead of "Module").

The installation process changed once again. The filename is now just a random string (see Appendix 2), the main persistence method is creating a startup link and some of the previous stealth methods were dropped.

You can spot the duplicity in the Handshake and Modules phases. The *module* indeed performs these duplicate actions upon its execution. They do not differ from the ones the *dropper* performed.

The *Network Data Stealer Helper Thread* has been replaced by *Injector*. Since now the bot supports webinjects, it has become necessary to hook additional browser functions. Therefore, this new Helper Thread injects the *module* to all other processes (with a few exceptions) and the *module* acts differently when inside a web browser. For more information refer to Section 4.1.

Three other Helper Threads have been added. The first one is *Pipe Operator*. Outputs of some of the commands and hooked functions are now sent to a pipe and this single thread handles the reporting of results back to the C&C.

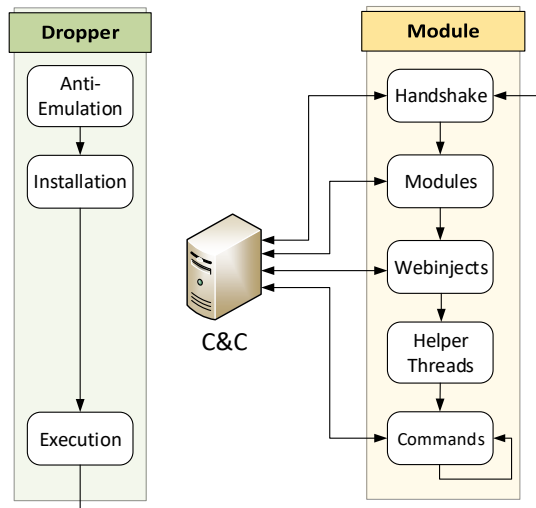


Figure 4: Control flow in early version 5.4.

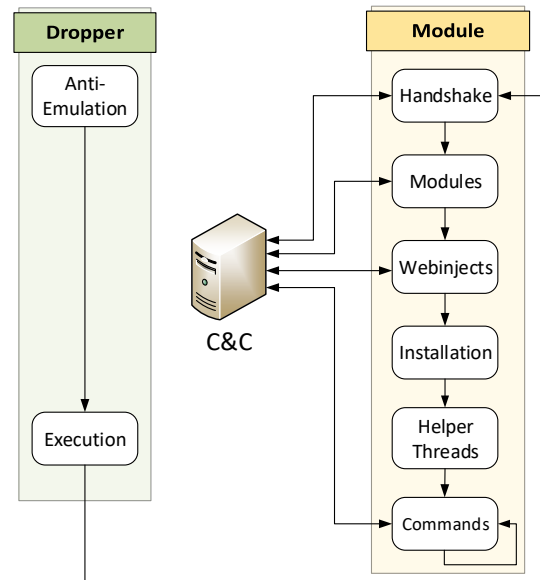


Figure 5: Control flow in later version 5.4.

The second one is *Chrome Link Modifier* which searches for a shortcut to the Google Chrome browser. If it finds such link, it modifies it so that it will disable HTTP2 and SPDY when launched to ease the network traffic exfiltration and modification.

The final new Helper Thread is *Parent Protector*. This thread ensures that the *dropper* survives file deletion. It reads *dropper's* data from the disk right after execution and periodically checks its existence. If the file is not found, it creates it again.

This version also introduces RC4 encryption. An RC4 key is stored inside the data section of the binary. It is used to decrypt C&C server responses and protect some data inside the registry hive. Both the stored C&Cs and the RC4 key can be additionally protected using a simple XOR cipher with one byte key specific for each sample (see Fig. 3).

No new command has been added, but the behaviour of two commands have changed. The PLUGIN command has been changed to download a file and inject it into *svchost.exe*. The botkiller has been changed to do nothing at all. Since this version, the command is useless, yet it is still supported at the time of writing.

As for minor changes, it introduced a new anti-emulation technique by checking the `cpuid` assembly instruction and the *module* is injected into *svchost.exe* instead of a copy of the *dropper's* process.

4 Version 5.4 - The current state

At the very beginning of 2018, version 5.4 has been released and it is the most recent one at the time of writing. It changed the execution flow yet again. The new flow is clearer and more straightforward (see Fig. 4).

Probably the most important change is the addition of encryption of *modules*. We did not mention that the downloaded *modules* are encrypted because they were really downloaded as raw binary executable files. Since this version, encryption has been introduced and the *modules* are protected using RC4 and compression. The RC4 key used is the same as the one used to decrypt the URLs.

The *module* is yet again part of the *dropper's* data, now stored encrypted using RC4 and compressed.

One interesting change has been added to the *dropper* - it no longer runs when the infected machine is located in Russia, Belarus or Kazakhstan. Code is available in Appendix 3.

Completely new persistence methods were introduced. First, the bot tries to add itself to the `HKCU\Software\Microsoft NT\CurrentVersion\Winlogon\Shell` registry. If that fails, it tries to set itself as a screensaver setting the `HKCU\Control Panel\Desktop` registry key as follows:

- `ScreenSaveActive = 1`
- `ScreenSaverIsSecure = 0`
- `ScreenSaveTimeOut = 60`
- `SCRNSAVE.EXE = %THIS%`

The *module* remained almost unchanged from the functionality point of view. It only dropped the support for the `CMD-Result` command and introduced one new command, `wbj`, being able to update the *webinjects* configuration. This change is significant, because in version 5.3, *webinjects* were only obtained once when the *module* has been executed and there was no way to propagate new ones.

```
v30 = sprintf_s(
    cmdRequest,
    cmdRequestMaxSize,
    cmdRequestFormat,
    StatusInfo->machineGUID,
    StatusInfo->pcname,
    StatusInfo->version.dwMajorVersion,
    StatusInfo->version.dwMinorVersion,
    StatusInfo->version.wProductType,
    bitness,
    is_admin,
    AV_info,
    L"5.4",
    date,
    L"Sochost32");
```

Figure 6: Commands request creation.

```
if ( !_snwprintf(
    mutexName,
    size + 1,
    L"%ls_%ls_DL",
    L"emFIZXIXqRphYmJlciSubwrr",
    L"Sochost32") > 0 )
{
    hHandle = CreateMutexW(
        0,
        1,
        mutexName);
    if ( GetLastError() == ERROR_ALREADY_EXISTS
        && WaitForSingleObject(
            hHandle,
            30000u) == WAIT_TIMEOUT )
    {
        result = 1;
    }
}
```

Figure 7: Synchronization mutex creation.

As mentioned in the beginning, modifying the bot's functionality without changing the version number is common in Neutrino Bot. We noticed a change at the very beginning of the *module* that tries to cripple Mozilla Firefox security mechanisms. It does so by patching the code of *mozglue.dll*. It aims at an exported function *DllBlocklist_Initialize*, specifically at the hook of *BaseThreadInitThunk* it installs. The bot ensures that no blocking will be done (see Appendix 4 for examples of code).

More importantly, the control flow has been changed as well. Quite interestingly, both installation and persistence is now done from the *module*, but it still installs the *dropper*. This is done by an upgraded version of the *Parent Protector* Helper Thread mentioned earlier.

We also discovered the return of the previously dropped feature of credentials stealing in September 2018 – Neutrino Bot now successfully steals credentials for Microsoft Outlook, Mozilla Thunderbird and Windows Live.

4.1 Man-in-the-browser attacks

This feature is becoming more and more important for Neutrino Bot, so we decided to provide a more detailed overview of how it achieves its goal.

As mentioned in the previous sections, this conventional attack consists of injecting the *module* into the browser processes (Internet Explorer, Mozilla Firefox, Google Chrome). The injected *module* then hooks specific important functions of the browser in order to read and alter the browsing data. A detailed survey on how the approaches among malware authors differ in fulfilling the task could be found here [8]. The methods of locating the important functions require regular maintenance only for Chromium-based projects and the principles behind them in the mid of 2018 are described here [9].

Generally, Neutrino Bot tries to hook four functions in *chrome.dll*: *SSL_new*, *SSL_free*, *SSL_read* and *SSL_write*, all of which are basically wrappers of low-

level functions from the table of SSL protocol methods (*kTLSProtocolMethods*, aka the SSL VMT table). The function look up is based on a pattern-based search in the corresponding part of the memory of *chrome.exe* process where the *chrome.dll* module is mapped. A proof of Neutrino Bot's active development can be seen in the fact that a build dated 19th June 2018 is successful against all 4 functions in Chrome versions 66 and 67, but fails to find *SSL_new* and *SSL_free* in the newer ones. However, an updated bot built on 2nd October 2018 fixes the attacks completely by adding new search patterns. Examples that illustrate the whole process can be found in Appendix 5.

5 Collecting the fragments

If you try to track Neutrino Bot and you have access to enough data, soon, you will get lost in the chaos that you will see. One build may try to attack banks in one country, another one steal passwords all over the world and another one may try for example to distribute ransomware. All this information will be mixed together and it will be very hard to keep track of what is actually happening. That happens because a lot of different malware operators use this bot.

To make any sense of the collected data, one has to start sorting the samples into groups. Ideally, we would like to know what samples belong together and are probably operated by the same actor. But how? Neutrino Bot is kind enough to provide us with information that help us to understand the situation better.

As mentioned before, every sample of the bot you encounter in the wild consists of four valuable pieces of information:

- Version
- Bot name
- List of C&C servers
- Build id

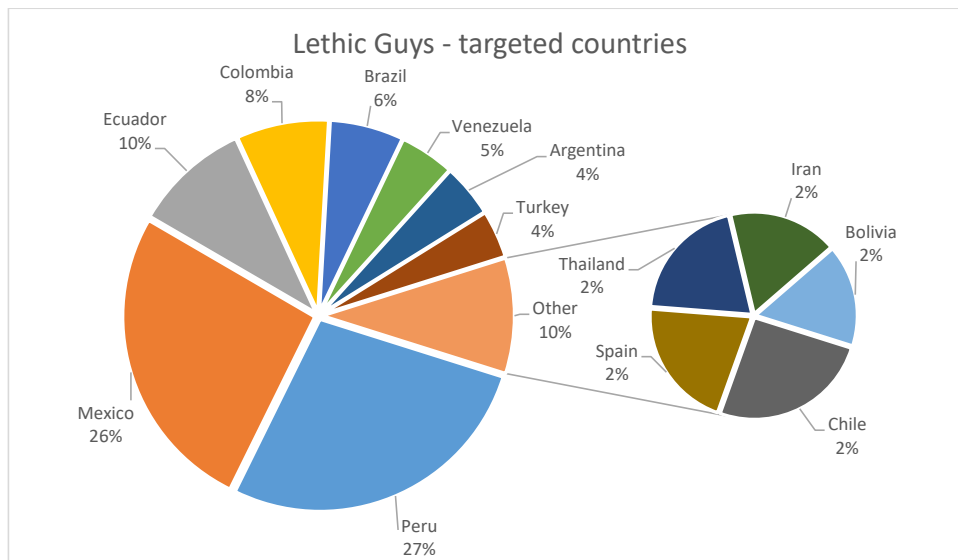


Figure 8: Countries targeted by the Lethic Guys.

All this information is stored in the *module*, the *dropper* offers none. The C&C addresses are the only item on that list that is stored encrypted. All the three remaining ones can be retrieved quite easily.

The best way to find the version and the bot name is to locate a function responsible to process commands. This function has to first form a commands request and the code looks like in Fig. 6. You can immediately see the version ("5.4") and the bot name ("Sochost32").

The build id can be found in function responsible for creating a mutex to ensure only one instance is running (see Fig. 7) – "emFiZXlxQrphYmJlci5ubwrr" in this case. When you get familiar with the format of these strings, checking the Strings window in IDA Pro is enough.

The version is great, but it does not help much with sorting. The bot name, however, seems like a much better choice. Sadly, the main issue is that about 95% of the samples we have encountered share the same bot name – "NONE". That kinda breaks this strategy. The list of C&C servers could be useful should the authors use some pattern – and some do indeed. Therefore, it can help in some cases, but definitely does not work for all of the groups.

Which brings us to the fourth fragment – the build id. If you collect enough data, you will find that there are different configurations using the same build id. There are also ones that share an "almost identical" build id differing only in the last one or two characters. This is the main method you can use to discover new botnets or new builds for existing ones – if their build ids (almost) match, both builds most likely belong to the same botnet.

We should mention that these conclusions are not supported by a verifiable method, but rather come from a year long observation. However, the data we collected strongly support this theory.

6 A journey with Neutrino Bot through 2018

Now that all the basics have been told, it remains to unravel the results of our one year long observation. We cover the specific period of exactly one year starting from 1st October 2017. Let's begin with some numbers.

During this period, we have discovered 120 different builds. Using the method described, we were able to sort them into 41 unique botnets. Of these botnets

- 12 showed no activity that would differ from the default configuration
- 18 were significantly active, but are not any more at the time of writing
- 8 were active and still are at the time of writing
- 3 were evaluated as a special case

There is no guarantee that a connection between two different groups does not exist or that they share a common operator, but none of the mentioned botnets provided proof of that.

Each botnet has been given a nickname. We dubbed them so to represent some strong characteristics of their behaviour. This nickname is not an official mark, merely a way to help us distinguish them from one another. Let's look at the most interesting ones.

6.1 The Lethic Guys

This botnet is the most persistent and stable one we have encountered. Its operation consists mainly of distributing the Win32/Lethic malware. Besides that, it occasionally distributes Win32/Zurgop and a password stealer.

```

if (this.Titles != null && this.Redirs != null)
{
    this.Debug("\r\n\r\nStarting redir timer func");
    string activeWindowTitle = this.GetActiveWindowTitle();
    for (int i = 0; i < this.Redirs.GetLength(0); i++)
    {
        this.Debug("Checking title: " + this.Titles[i] + " -> " + this.Redirs[i]);
        if (activeWindowTitle.IndexOf(this.Titles[i]) > -1)
        {
            string text = Clipboard.GetText(TextDataFormat.Text);
            Clipboard.SetData(DataFormats.Text, this.Redirs[i]);
            Thread.Sleep(100);
            SendKeys.SendWait("{F6}");
            Thread.Sleep(50);
            SendKeys.SendWait("^v");
            Thread.Sleep(100);
            SendKeys.SendWait("{ENTER}");
            Clipboard.SetData(DataFormats.Text, text);
            Thread.Sleep(7000);
        }
    }
}

```

Figure 9: Redirection using keystroke simulation.

We have collected 28 different builds for this group. Their distribution method is not completely known, but we have discovered they distribute the bot through a fake installer, most likely for Adobe Flash. An interesting thing about the files is that the filenames have some letters 'i' (I) replaced by 'l' (L) and otherwise. The victims are probably persuaded to install the fake update to view some content.

They have been active from the beginning of the evaluation period and remain active at the time of writing. See the graph in Fig. 8 for information about affected countries. For campaign information, refer to Appendix 6.

6.2 The Redirectors

This is one of the most iconic Neutrino botnets we know of. They focus exclusively on Mexico and target Mexican banks. However, they never distributed any webinjects. Instead, they utilize the redirection technique.

They evolved greatly throughout the year and we can spot three specific periods of their activities. First, they used to rely solely on Neutrino Bot's DNS command. After a while, they started to distribute a simple program that modified the `hosts` file in order to perform the redirection. A final stage came when they released malware that performs the redirection in a completely different way. It communicates with a server to retrieve configuration and by simulating the F6, CTRL+V and ENTER keystrokes to replace the address bar of a web browser redirects the victim (see an example in Fig. 9). We believe that the operators created this payload themselves.

We have discovered 8 different builds of this group. The authors helped us themselves with discovering the distribution method. They pushed a spam tool through the bot that revealed the method as well as

the name of one of the malicious attachments. All the campaigns shared a common characteristic – the malicious file had a `.scr` extension and was zipped twice.




Name	Size
 CFE_Factura.zip	216 KB
 CFE_Factura 07-03-2018.zip	216 KB
 CFE_Factura 07-03-2018.scr	358 KB

Figure 10: Typical infection chain for the Redirectors.

They have been active through the whole examination period. Besides the already mentioned payloads, the operators utilized a tool to steal email credentials and contacts, a malware setting up a proxy connection and once even a banking trojan targeting South America.

6.3 The Mining RATs

This group could not be more different from the previous two. They specialize in distributing two types of malware - cryptocurrency miners and Remote Access Trojans (RAT). Specifically, they utilized the Win32/Remcos, MSIL/Immirat and MSIL/NanoCore RATs. Besides that, a mixture of other malware has been seen being distributed by them – Win32/Zurgop, Win32/Formbook, Win32/Neurevt.

We have discovered 13 different builds for this group. The distribution method has not been discovered and they do not seem to target a specific part of the world. They have been active throughout the whole year with a pause from April to September. An interesting thing about them is that they still use old 5.1 and 5.2 builds actively, even though they possess the newest builds.

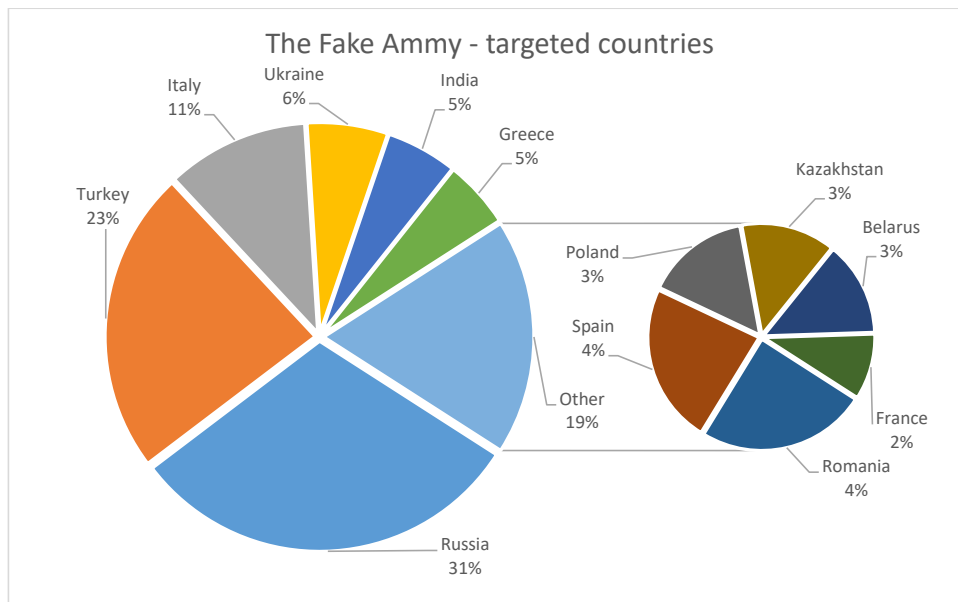


Figure 11: Countries targeted by the Fake Ammy group.

6.4 The Arsonists

This group appeared at the beginning of May 2018 and remains active till the time of writing. They started with a simple coin miner, but almost immediately moved to a different target – stealing bank information from French victims.

They achieve so in two ways. The first one is trying to attack French banks using webinjects. The second one is utilizing the FIND command to exfiltrate data, specifically files that could be connected to the identity of the victim or bank-related sensitive information. Additionally, they try to retrieve files that could be connected to a quite large variety of cryptocurrencies.

We have discovered only three different builds for this group and the distribution method remains unknown. However, the targeted attack on victims from only one country and the way the authors are able to utilize the bot's capabilities prove that they belong to the more experienced ones.

6.5 Tinuebot & Fareit

This group emerged at the beginning of February 2018. As far as payloads go, the operators focus purely on distributing the Win32/Tinuebot and Win32/PSW.Fareit. They are active in waves and often change both the URL and hash of their payloads. Besides that, they utilize heavily the FIND command to exfiltrate crypto wallets, passwords and private keys.

During September 2018, they made a breakthrough and started distributing webinjects as well. They try to aim at two targets - an Italian Post Office and Facebook. We have discovered 8 different builds for this group, but we were not able to discover the distribution method.

6.6 The Fake ExpressVPN

This group represents one that is specific mainly in one thing – it uses the old cracked Neutrino Bot 5.1 version. They were active in May 2018.

What makes it especially interesting is that the operators distributed the bot via a fake ExpressVPN download page. They took a legitimate installer for the VPN software and bundled it with Neutrino Bot. That way, they successfully infected the victim while still installing the expected software.

They took an approach of try-it-all. They first distributed a credentials stealer, then moved to a cryptocurrency miner and finally pushed a Win32/ClipBanker – a type of banker that monitors the clipboard and replaces strings that look like crypto wallets with the attacker's wallet.

6.7 The Fake Ammy

Now this group has definitely been the most advanced one. They were able to push their build of Neutrino Bot via an official Ammy Remote Admin website. They used the same approach as the Fake ExpressVPN and bundled a legitimate Ammy installer with the bot. The infection happened the 13th of June 2018 and lasted for one day.

The operators did not try to distribute any malicious payload through the bot. Instead, they tried to scan the victimized machines for cryptocurrency wallets and a large variety of remote access software such as PuTTY or WinSCP. Whether they then chose to further attack specific victims separately remains unknown. For more information, you can refer to an article about the infection [10].

6.8 The Dridex Distributors

This group is a proof that Neutrino Bot's buyers can have access to much more advanced malware. Even though their lifespan was very short (only the first half of September 2018), they pushed two infamous malware families - Win/Dridex and Win/Ursnif.

They also represent something that is not uncommon in Neutrino botnets - distributing webinjects in an invalid format. The bot's webinjects JSON configuration has its rules of course. These operators distributed webinjects for 9 famous bank institutions in the world, but formatted it for Win32/Tinukebot [11]. Therefore, Neutrino Bot could not do anything but ignore it.

6.9 The Filecoder Guys

The last on our list is this group that has been distributed as a fake Adobe Flash update. Their lifespan was a short one during April 2018. Its operators distributed a cryptoransomware and tried to steal cryptocurrency related files.

Even though it may not seem like an important group, it represents a type of Neutrino botnet that, despite its short lifespan, focuses purely on one thing and remains with the same configuration for the whole infection.

6.10 Others

In this subsection, we mention only briefly the remaining interesting botnets discovered.

One botnet we have encountered focused only on setting up a malicious proxy connection using the Neutrino Bot's PROXY command. Another one used solely the FIND command to steal files that may store credit card information.

We have discovered one botnet that seemed to focus on web servers. It used to distribute a cryptocurrency miner, but then disappeared in March

2018. It reappeared again in October and delivered Win32/Filecoder.GandCrab instead.

From the webinjects point of view, we have discovered a botnet that tried to target a New Zealand bank and one targeting an online payment solution.

7 Conclusion

We have mapped the history of Neutrino Bot marking the most important milestones. We focused on the more recent history in detail providing an overview of how the bot works today.

We have explained that Neutrino Bot is being sold to a large variety of malware operators. Each of those can act differently and use the bot in his own way. Therefore, it is crucial to be able to distinguish one botnet from another. We have shown how to extract the important data from the bot which enables an analyst to do just that.

We have provided an overview of the most important Neutrino botnets discovered in 2018 and described them more closely. We also provided the numbers that prove the popularity of this bot among cybercriminals.

Some of the characteristics such as cryptocurrency stealing seem to be an attractive target for most of the botnets. Payloads, however, differ almost completely. We have described botnets that distributed other banking trojans, cryptoransomware, credential stealers, cryptocurrency miners or RATs. We have seen the operators distributing their bots through supply chain attacks, fake product websites, malvertising or exploits. The lifespan of the botnets ranged from one day to a whole year.

We hope that we brought Neutrino Bot a little closer to the spotlight, since it has been in the shadows for a long time. We believe it deserves more attention due to the large number of different occasions we have encountered it throughout the year 2018 and the fact that its evolution is anything but over.

Appendices

Appendix 1: Changelog

- 3.2.1 (11/2014)
 - Absolutely no obfuscation - both imports and character strings are immediately visible
 - DDoS commands, remote file and command execution, keylogging, simple redirection
 - Spreading through removable drives and RAR archives
- 3.5 (02/2015)
 - The concept of build id introduced - a unique character string identifying one specific build
 - Spreading mechanism removed
 - Dropped the DNS spoofing support (until 3.9)
- 3.6 (03/2015)
 - New feature: Login data Stealer (IMAP, POP3, HTTP, SMTP)

- New feature: Credit card scraping
- New feature: Capturing the outgoing traffic by installing the corresponding function hooks
- 3.9.4 (07/2015)
 - First wave of obfuscation - process and command names now compared by computing a checksum of the name
- 4.4 (10/2015)
 - Lost all the support for DDoS attacks
 - New feature: Exfiltration of data
 - New feature: Scanning running processes
 - New feature: Ammy Remote Admin plugin
- 5.0 (06/2016)
 - Second wave of obfuscation – Windows APIs called by hash
 - Modular structure introduced – a *dropper* that performs anti-emulation and installation and a *module* responsible for the core functionality
 - Two new commands: Proxy and screenshot
- 5.1 (09/2016)
 - Privilege escalation introduced
 - Firewall exception added
- 5.2 (04/2017)
 - Windows Defender evasion
- 5.3 (11/2017)
 - 64-bit builds of the *module* available
 - New feature: Support for webinjects
 - RC4 encryption introduced
- 5.4 (01/2018)
 - New protective *dropper* not running in Russia, Belarus and Kazakhstan
 - Mozilla Firefox security undermining mechanism
 - Resurrection of the credential stealing

Appendix 2: Install filename generation algorithms

```
def create_inst_filename():
    src_filenames = get_all_files_on_path(%WINDOWS%)
    if len(src_filenames) == 0:
        src_filenames = get_all_files_on_path(%SYSTEM%)

    # remove all entries that contain a forbidden keyword
    src_filenames.remove_all_that_contains(["install", "setup", "update", "patch"])

    filename = src_filenames[randint(0, len(src_filenames))]
    filename.swap_all('i', 'l') # replace all 'i' characters to 'l' and otherwise
    filename.swap_all('y', 'u')
    filename.swap_all('0', 'o')
    filename.swap_all('3', 'e')
    filename.swap_all('6', 'b')
    filename.swap_all('q', 'g')
    return filename
```

Listing 1: Filename generation algorithm in version 5.0.

```

def create_inst_filename():
    src_filenames = get_all_files_on_path(%WINDOWS%, "*.exe")
    if len(src_filenames) == 0:
        src_filenames = get_all_files_on_path(%SYSTEM%, "*.exe")
    filename = src_filenames[randint(0, len(src_filenames))]

    for i, c in enumerate(filename):
        if (c >= 'a' and c <= 'm') or (c >= 'A' and c <= 'M'):
            filename[i] = c + 0xD
        else:
            filename[i] = c - 0xD

    return filename

```

Listing 2: Filename generation algorithm in versions 5.1 and 5.2.

```

def create_inst_filename():
    return random_string(random(8,10))

```

Listing 3: Filename generation algorithm in version 5.3.

```

def create_inst_filename():
    return random_guid()

```

Listing 4: Filename generation algorithm in version 5.4.

Appendix 3: Country check inside version 5.4 dropper

```

bResult = 0;
data = 0;
geoId = GetUserGeoID(GEOCLASS_NATION);
Location = geoId;
if ( geoId == -1 )
{
    data = Func::VirtualAlloc(0x208);
    if ( !GetLocaleInfow(LOCALE_SYSTEM_DEFAULT, 0x5Au, data, 0x104) )
        Func::VirtualFree(data);
}
else
{
    dataSize = GetGeoInfow(geoId, GEO_IS02, 0, 0, 0);
    if ( dataSize > 0 )
    {
        data = Func::VirtualAlloc(2 * dataSize);
        GetGeoInfow(Location, GEO_IS02, data, dataSize, 0);
    }
}
if ( Func::VirtualQuery(data) )
{
    exceptCountries[0] = 'B';           // Belarus
    exceptCountries[1] = 'Y';
    exceptCountries[2] = '*';
    exceptCountries[3] = 'R';           // Russia
    exceptCountries[4] = 'U';
    exceptCountries[5] = '*';
    exceptCountries[6] = 'K';           // Kazakhstan
    exceptCountries[7] = 'Z';
    exceptCountries[8] = '\0';
    if ( String::ContainsW(exceptCountries, data) )
        bResult = 1;
    Func::VirtualFree(data);
}
return bResult != 0;

```

Figure 12: Checking the country the victim is in.

```

if ( gInfo->isInExcludedCountry )
{
    OutputString[0] = 'R';
    OutputString[1] = 'C';
    v20 = 0;
    OutputDebugStringW(OutputString);
    ExitProcess_1 = GetProcByHash(1, 0xE80FB45);
    (ExitProcess_1)(0);
}

```

Figure 13: Process exit if an excluded country is detected.

Appendix 4: Mozilla Firefox security undermining mechanism

```

hModule = GetModuleHandleW(L"mozglue.dll");
if ( hModule )
{
    LODWORD(moduleSize) = getModuleSize(hModule);
    HIDWORD(moduleSize) = v3;
    if ( v3 || moduleSize )
    {
        patternAddr = PatternSearch(&FF_Pattern, hModule, moduleSize, 21, 0);
        if ( patternAddr )
            v5 = WriteProcessMemory(0xFFFFFFFF, patternAddr, &patch, 1u, 0); // patch = 0xEB
    }
}
CreateThread(0, 0, MainThread, 0, 0, 0);

```

Figure 14: The code that searches for a pattern inside mozglue.dll. If it finds it, it patches the first byte, making the conditional jump an unconditional one.

```

.data:1002BA00 74 13          jz     short near ptr unk_1002BA15
.data:1002BA02 81 7D F4 00 10 00 00  cmp    dword ptr [ebp-0Ch], 1000h
.data:1002BA09 0F 95 3F       setnz  byte ptr [edi]
.data:1002BA0C 83 7D F8 20    cmp    dword ptr [ebp-8], 20h ; ' '
.data:1002BA10 0F 95 3F       setnz  byte ptr [edi]
.data:1002BA13 0A D9         or     bl, cl
.data:1002BA13          ; -----
.data:1002BA15 00          unk_1002BA15 db  0

```

Figure 15: The pattern that the malware looks for.

```

.data:1002BA00 EB 13          jmp    short near ptr unk_1002BA15
.data:1002BA02 81 7D F4 00 10 00 00  cmp    dword ptr [ebp-0Ch], 1000h
.data:1002BA09 0F 95 3F       setnz  byte ptr [edi]
.data:1002BA0C 83 7D F8 20    cmp    dword ptr [ebp-8], 20h ; ' '
.data:1002BA10 0F 95 3F       setnz  byte ptr [edi]
.data:1002BA13 0A D9         or     bl, cl
.data:1002BA13          ; -----
.data:1002BA15 00          unk_1002BA15 db  0

```

Figure 16: How the code would look after patching the first byte with 0xEB.

```

realStartAddress = StartAddress;
isInitial = IsInitialThread;
if ( ShouldBlockThread(StartAddress) )
    realStartAddress = DoNothing;
return Old::BaseThreadInitThunk(
    isInitial,
    realStartAddress,
    ThreadParam);

```

Figure 17: The original code of a hook that mozglue.dll installs on BaseThreadInitThunk. The malware changes the ShouldBlockThread function so that it always returns false.

Appendix 5: Man-in-the-browser attacks

```

.text:1000533F          loc_1000533F:                ; CODE XREF: HookChrome+217fj
.text:1000533F 6A 00          push     0
.text:10005341 6A 21          push     21h ; '!'
.text:10005343 8B 55 FC      mov     edx, [ebp+var_4]
.text:10005346 52           push     edx
.text:10005347 8B 45 F8      mov     eax, [ebp+var_8]
.text:1000534A 50           push     eax
.text:10005348 8B 45 DC      mov     eax, [ebp+hModule]
.text:1000534E 99           cdq
.text:1000534F 52           push     edx
.text:10005350 50           push     eax
.text:10005351 68 28 C9 02 10 push   offset fn_SSL_write
.text:10005356 E8 65 B8 00 00 call   PatternSearch
.text:10005358 83 C4 1C      add     esp, 1Ch
.text:1000535E 89 45 C8      mov     [ebp+var_38], eax
.text:10005361 89 55 CC      mov     [ebp+var_34], edx
.text:10005364 8B 4D C8      mov     ecx, [ebp+var_38]
.text:10005367 0B 4D CC      or     ecx, [ebp+var_34]
.text:1000536A 74 16        jz     short loc_10005382
.text:1000536C 68 88 13 03 10 push   offset pTrampoline_SSL_write ; int
.text:10005371 68 D0 60 00 10 push   offset pCallback_SSL_write ; LPCVOID
.text:10005376 8B 55 C8      mov     edx, [ebp+var_38]
.text:10005379 52           push     edx ; lpAddress
.text:1000537A E8 A1 2A 01 00 call   CreateTrampoline

```

Figure 18: Using the pattern search approach to locate the SSL_write function in chrome.dll.

```

.text:104298AB          SSL_write proc near          ; CODE XREF: DoPayloadWrite+4Ffj
.text:104298AB 55           push     ebp
.text:104298AC 89 E5        mov     ebp, esp
.text:104298AE 53           push     ebx
.text:104298AF 57           push     edi
.text:104298B0 56           push     esi
.text:104298B1 83 EC 08     sub     esp, 8
.text:104298B4 A1 44 60 CB 12 mov     eax, ___security_cookie
.text:104298B9 8B 7D 08     mov     edi, [ebp+arg_0]
.text:104298BC 31 E8        xor     eax, ebp
.text:104298BE 89 45 F0     mov     [ebp+var_10], eax
.text:104298C1 8B 47 18     mov     eax, [edi+18h]
.text:104298C4 C7 80 98 00 00 01+mov   dword ptr [eax+98h], SSL_NOTHING
.text:104298CE E8 54 39 C8 FF call   ERR_clear_error
.text:104298D3 E8 AD D5 FC FF call   ERR_clear_system_error

```

Figure 19: How SSL_write looks inside chrome.dll.

```

.data:1002C914          fn_SSL_read:                ; DATA XREF: HookChrome+1FEfj
.data:1002C914 55           push     ebp
.data:1002C915 89 E5        mov     ebp, esp
.data:1002C917 57           push     edi
.data:1002C918 56           push     esi
.data:1002C919 8B 7D 08     mov     edi, [ebp+8]
.data:1002C91C FF 75 10     push   dword ptr [ebp+10h]
.data:1002C91F FF 75 0C     push   dword ptr [ebp+0Ch]
.data:1002C922 57           push     edi
.data:1002C923 E8 35 00 00 00 call   near ptr loc_1002C95B+
.data:1002C928
.data:1002C928          fn_SSL_write:              ; DATA XREF: HookChrome+241fj
.data:1002C928 55           push     ebp
.data:1002C929 89 E5        mov     ebp, esp
.data:1002C92B 53           push     ebx
.data:1002C92C 57           push     edi
.data:1002C92D 56           push     esi
.data:1002C92E 83 EC 08     sub     esp, 8
.data:1002C931 A1 3F 3F 3F 3F mov     eax, ds:3F3F3F3Fh
.data:1002C936 8B 7D 3F     mov     edi, [ebp+3Fh]
.data:1002C936
.data:1002C939 3F 3F 3F 3F 3F 3F+xmmword_1002C939 xmmword 3F003F003F3F3F3F3F3F3F3F3F3Fh
.data:1002C949 00          db     0

```

Figure 20: The patterns for SSL_read and SSL_write the malware uses.

Appendix 6: Infection campaigns information

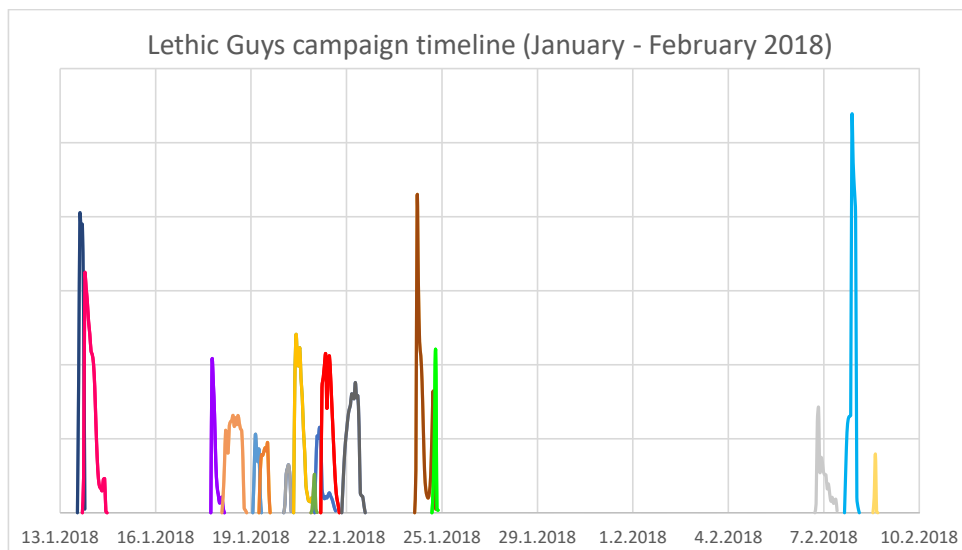


Figure 21: Campaigns of the Lethic Guys during January and February 2018.

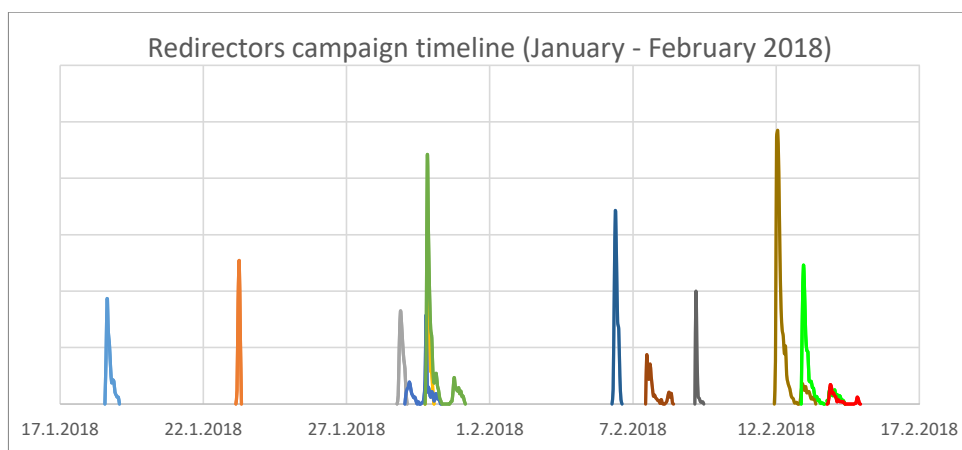


Figure 22: Campaigns of the Redirectors during January and February 2018.

Acknowledgment: The authors would like to thank their employer, ESET, for allowing them to perform this research and for making it possible to complete it in such a detail.

Author details

Jakub Souček

ESET Research Czech Republic s.r.o.
U Přehradý 3204/61, 466 02 Jablonec nad Nisou-Mšeno nad Nisou, Czech Republic
jakub.soucek@eset.cz

Jakub Tomanek

ESET Research Czech Republic s.r.o.
U Přehradý 3204/61, 466 02 Jablonec nad Nisou-Mšeno nad Nisou, Czech Republic
jakub.tomanek@eset.cz

Peter Kálnai

ESET Research Czech Republic s.r.o.

U Přehradý 3204/61, 466 02 Jablonec nad Nisou-Mšeno nad Nisou, Czech Republic

peter.kalnai@eset.cz

References

- [1] Malware don't need coffee, "Neutrino Bot (aka MS:Win32/Kasidet)," June 2014. <https://malware.dontneedcoffee.com/2014/06/neutrino-bot-aka-kasidet.html>.
- [2] "ESET GitHub, SHA-256 hashes of Neutrino Bot files." <https://github.com/eset/malware-ioc/tree/master/kasidet>.
- [3] S. Yunakovsky, "Jimmy Nukebot: from Neutrino with love," tech. rep., Kaspersky lab, August 2017. <https://securelist.com/jimmy-ukebot-from-neutrino-with-love/81667/>.
- [4] V. Tom, "Kasidet POS malware spread through fake security update," tech. rep., ThreatSTOP, June 2017. <https://blog.threatstop.com/kasidet-pos-malware-spread-through-fake-security-update>.
- [5] S. Yunakovsky, "Neutrino modification for POS-terminals," tech. rep., Kaspersky lab, June 2017. <https://securelist.com/neutrino-modification-for-pos-terminals/78839/>.
- [6] Wikipedia. https://en.wikipedia.org/wiki/Luhn_algorithm.
- [7] Y. Oyama, "Investigation of the Diverse Sleep Behavior of Malware," *Journal of Information Processing*, vol. 26, pp. 461–476, June 2018. https://www.jstage.jst.go.jp/article/ipsjip/26/0/26_461/_pdf/char/en.
- [8] P. Kálnai and M. Poslušný, "Browser Attack Points Still Abused by Banking Trojans," tech. rep., Virus Bulletin, 2017. <https://www.virusbulletin.com/uploads/pdf/magazine/2018/VB2017-Kalnai-Poslusny.pdf>.
- [9] P. Kálnai and M. Poslušný, "Browser Attack Points Still Abused by Banking Trojans - 2018 update," tech. rep., Virus Bulletin, 2018. <https://www.virusbulletin.com/blog/2018/07/vb2017-paper-and-update-browser-attack-points-still-abused-banking-trojans/>.
- [10] O. Kubovič, "Ammy Admin compromised with malware again; World Cup used as cover," tech. rep., ESET, July 2018. <https://www.welivesecurity.com/2018/07/11/ammy-admin-compromised-malware-world-cup-cover/>.
- [11] "TinyNuke." <https://github.com/rossja/TinyNuke/blob/master/Bot/WebInjects.cpp>.