Stealc: A new stealer emerges in 2023

By VMRay Labs

Published: 2023-05-05 · Archived: 2026-04-05 23:42:50 UTC

Overview

A new malware family called **Stealc** was released recently, which is a Spyware designed to copy files, credentials and other sensitive information from the victim's hard drive and make them available to the attacker. It also employs a variety of techniques to evade detection, including one technique based on remotely storing a hard disk-based hardware ID, which **prevents the sample from infecting the same machine twice**, and thus avoids revealing malicious behavior in any future analysis runs.

In this blog post, we want to highlight how Stealc, which has been shown to have parallels to other known malware families (such as Vidar, Raccoon, Mars and RedLine), steals sensitive data from its victims and how it tries to evade detection. We have aided our analysis with an in-depth research into how this malware and it's techniques have been implemented.

Malicious Behavior of Stealc

VMRay Platform uses a dynamic analysis method, which means that the malware is actually executed in a virtual environment, where the actions of the malware are recorded and later analyzed to detect malicious behavior.

These detection rules are known as VMRay Threat Identifiers (VTIs). In our case, they reveal plenty of malicious behavior, ranging from capturing screenshots, reading sensitive e-mail and web browser data, to searching for cryptocurrency wallets (see Figure 1).

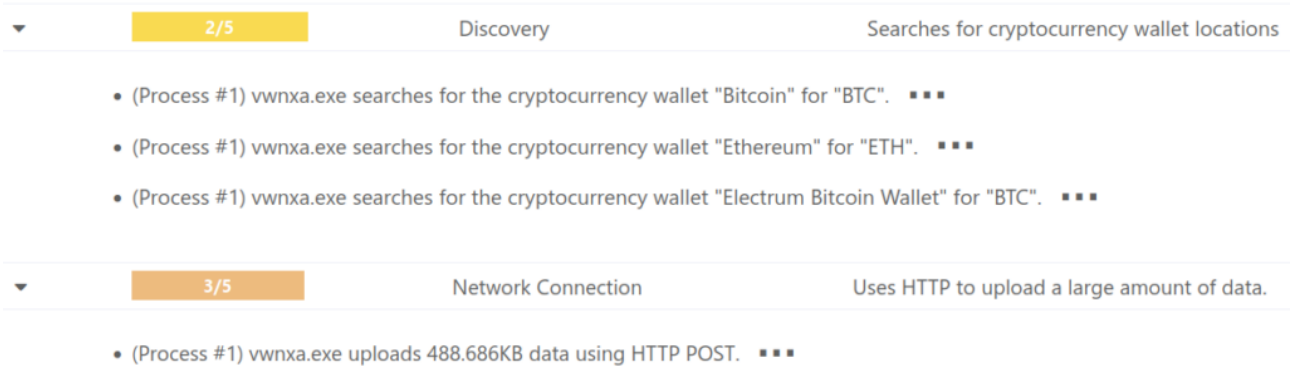
VMRay Threat Identifiers (17 rules, 38 matches)

Score	Category	Operation	Count	Classification
5/5	Extracted Configuration	Stealc configuration was extracted	1	Spyware
5/5	YARA	Malicious content matched by YARA rules	5	Spyware
5/5	Data Collection	Tries to read cached credentials of various applications	1	Spyware
5/5	Data Collection	Combination of other detections shows multiple input capture behaviors	1	Spyware
4/5	Reputation	Contacts known malicious URL	8	-
3/5	Data Collection	Takes screenshot	1	-
3/5	Network Connection	Uses HTTP to upload a large amount of data.	1	-
2/5	Data Collection	Reads sensitive browser data	2	-
2/5	Discovery	Searches for senstive browser data	2	-
2/5	Discovery	Searches for cryptocurrency wallet locations	3	-
2/5	Data Collection	Reads sensitive mail data	1	-
2/5	Discovery	Searches for senstive application data	1	-
1/5	Discovery	Enumerates running processes	1	-
1/5	Discovery	Possibly does reconnaissance	2	-
1/5	Network Connection	Downloads executable	7	Downloader
1/5	Obfuscation	Resolves API functions dynamically	1	-

The VTIs also provide additional information on what exactly the malware is trying to do if you open one of the listings (see Figure 2).

In this case, the malware tried to find cryptocurrency wallets for Bitcoin, Ethereum and Electrum Bitcoin Wallet. Additionally, one of our triggers suggests that large amounts of data are uploaded to a remote server – a good indicator that some of the information is likely copied to the attacker (see Figure 3).

All of this behavior leads VMRay Platform to correctly conclude that this must be a malicious executable, classified as Spyware.



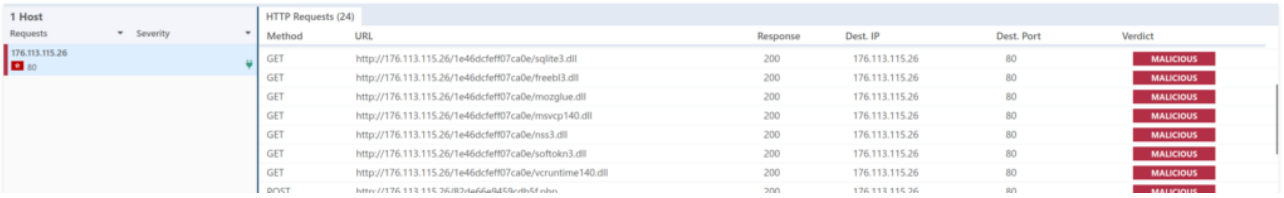
Now we have a better understanding what kind of information the stealer is collecting, but it is still unclear how the information is sent to the attacker.

Let us look into the **Network tab**, which displays all the recorded network traffic mapped to the associated Windows API calls that were used to send or receive the data. This is a powerful tool to analyze network traffic –

due to our unique and extensive monitoring approach, we can sometimes even capture communication data before it becomes encrypted.

Stealc network activity

The Network tab shows some interesting behavior (see Figure 4). We can see that the malicious sample tries to download a set of DLL files.



These libraries are used by benign popular applications such as web browsers to access and, if required, decrypt confidential information belonging to the application itself – Stealc abuses these libraries to collect the same confidential information, but with malicious intent (see Table below).

As an example, let's take "sqlite3.dll" as shown in Figure 4 above. This library allows Stealc to access a local database created in the SQLite database engine, which is also the method employed by Mozilla Firefox for storing user session cookies. When an attacker obtains a session cookie, they can potentially use it to access the victim's account without needing the second factor of authentication (e.g., a one-time password or a biometric). In essence, the attacker bypasses the 2FA mechanism by piggybacking on the authenticated session established by the victim.

Another use-case for these libraries is decrypting all saved login information, such as usernames, e-mail addresses and passwords.

Stealc does not just use the network connection to download additional DLLs, but also to communicate to a remote server about what its intended purpose is. The server then responds, for example, with specific filenames to search for. Analyzing this network behavior becomes easier thanks to one of the best features of our platform for malware researchers: the function log (or "flog" for short). This file contains all observed calls to the Windows API chronologically with human-readable function and parameter names. There is a reason why we internally see the flog as the malware analysts Swiss Army knife.

In this case, we find the base64 encoded network communication string in the function log, as well as the decoded version, which allows us analyze the communication more in-depth (see Figure 5). In one of the exchanges, for example, the C2 server asks our sample to collect information regarding the MetaMask crypto wallet and other web browser extensions, mostly related to crypto wallets and password managers.

```
[0040.946] CryptStringToBinaryA (in:
pszString="ZDYyZWE1YjM1OWY4Yjc0NGUwZGYzNzYwMmUxOGE5ZWM3MzUyZjk0NzM1MmVkZjE5YjlhYTdkMjQyZTAzNjU4MzM2NmEzZ
cchString=0x0, dwFlags=0x1, pbBinary=0x5f5d20, pcbBinary=0x17ad78, pdwSkip=0x0, pdwFlags=0x0 | out:
pbBinary=0x5f5d20, pcbBinary=0x17ad78, pdwSkip=0x0, pdwFlags=0x0) returned 1
[0040.946] lstrlenA (lpString="") returned 0
[0040.946] lstrcpyA (in: lpString1=0x1fe010, lpString2="" | out: lpString1="") returned=""
[0040.946] lstrlenA (lpString="d62ea5b359f8b744e0df37602e18a9ec7352f947352edf19b9aa7d242e036583366a3fe9
|done|jardin_rat.rtf|1|0|1|1|1|1|1|1") returned 109
```

Stealc's Encrypted Strings

Now that we have gathered all behavior-based information, we take a closer look into how the malware is implemented. A look at the code gives us a sense that this malware is likely written in C/C++. However, there are nearly no sensible human-readable strings present.

As we already know by now that this malware tries to find certain sensitive files, it needs to store file paths and search terms, but the strings associated with that process are nowhere to be found (see Figure 6).

Strings | Pseudocode-B | Segments

Address	Length	Type	String
.rdata:0062...	00000019	C	u6CEyB17NpWp8hgv9GmVVw==
.rdata:0062...	0000000D	C	kLaE+RFmP6E=
.rdata:0062...	00000015	C	u6CE23J5JYWu5CIE9HQ=
.rdata:0062...	00000019	C	u6CEyAdkNIWz4zoT+meVZR4=
.rdata:0062...	0000000D	C	kLaE+R5zKKE=
.rdata:0062...	00000011	C	ub2Z/yJkKYO45Bk=
.rdata:0062...	0000001D	C	u6mf6RN6C4Ww+BgYxnCRYhgWbrI=
.rdata:0062...	00000015	C	u6CE2AtlMoWwwwMM8A==
.rdata:0062...	0000001D	C	r7yD/xd7Eomw8j4O022cczkMRq8=
.rdata:0062...	00000011	C	naGG6gJ/ddLz8wYN
.rdata:0062...	0000000D	C	m6GZuEA4Ioyx
.rdata:0062...	00000011	C	ibaV+UEkaI5x+w==
.rdata:0062...	00000011	C	n7eJ+wYldM65+wY=
.rdata:0062...	0000000D	C	krGU5x44Ioyx
.rdata:0062...	00000011	C	u6CE3gFzNK68+g8g
.rdata:0062...	0000000D	C	v7eV6gZzAqOc
.rdata:0062...	00000015	C	u6CEzxdgL4O41AsR5g==
.rdata:0062...	0000000D	C	rqCc7hNlI65e
.rdata:0062...	0000001D	C	v7eJ+wZFMpK0+Q01+kaZeAwXUos=
.rdata:0062...	00000009	C	j7aT6hxw
.rdata:0062...	00000011	C	qoiH6nJzEK2q9hgE
.rdata:0062...	00000009	C	tIS8siZe
.rdata:0062...	0000000D	C	tqqY5TZ5Iw==
.rdata:0062...	0000000D	C	uIyj2z5XHw==
.rdata:0062...	00000011	C	2a2FpFd+M8/4/x8=
.rdata:0062...	0000001D	C	lLGE+0g5adPquVhZuzXFIUNQGQ==
.rdata:0062...	0000001D	C	0/LF6EEldoTp9VxV8DDHdwtLW6LP

Malware often uses obfuscation and encryption to try and hide important information, for example to evade static analysis tools such as antivirus signatures. In the case of Stealc, the strings are stored in an encrypted manner and are decrypted at runtime during the initialization step of the malware.

We have analyzed the sample to identify how the encryption takes place – this not only helps us to better understand the inner workings of the malware but also to develop a config extractor later on. Config extractors are tremendously helpful addition to the VMRay Platform which provides our customers with a malware family classification as well as high-quality IOCs by automatically extracting the configuration such as C2 URLs, encryption keys etc., without requiring manual reverse engineering.

We have identified the encryption algorithm to be RC4, which matches earlier reports about Stealc, however, we also found an issue involving randomly placed null bytes during decryption that all current Stealc decryptors seem to suffer from. A closer inspection reveals a key difference between how RC4 is usually implemented and how it is implemented in Stealc, namely that the ciphertext is not XORed with the keystream if it results in a null-byte, demonstrated in pseudo-code in Figure 7.

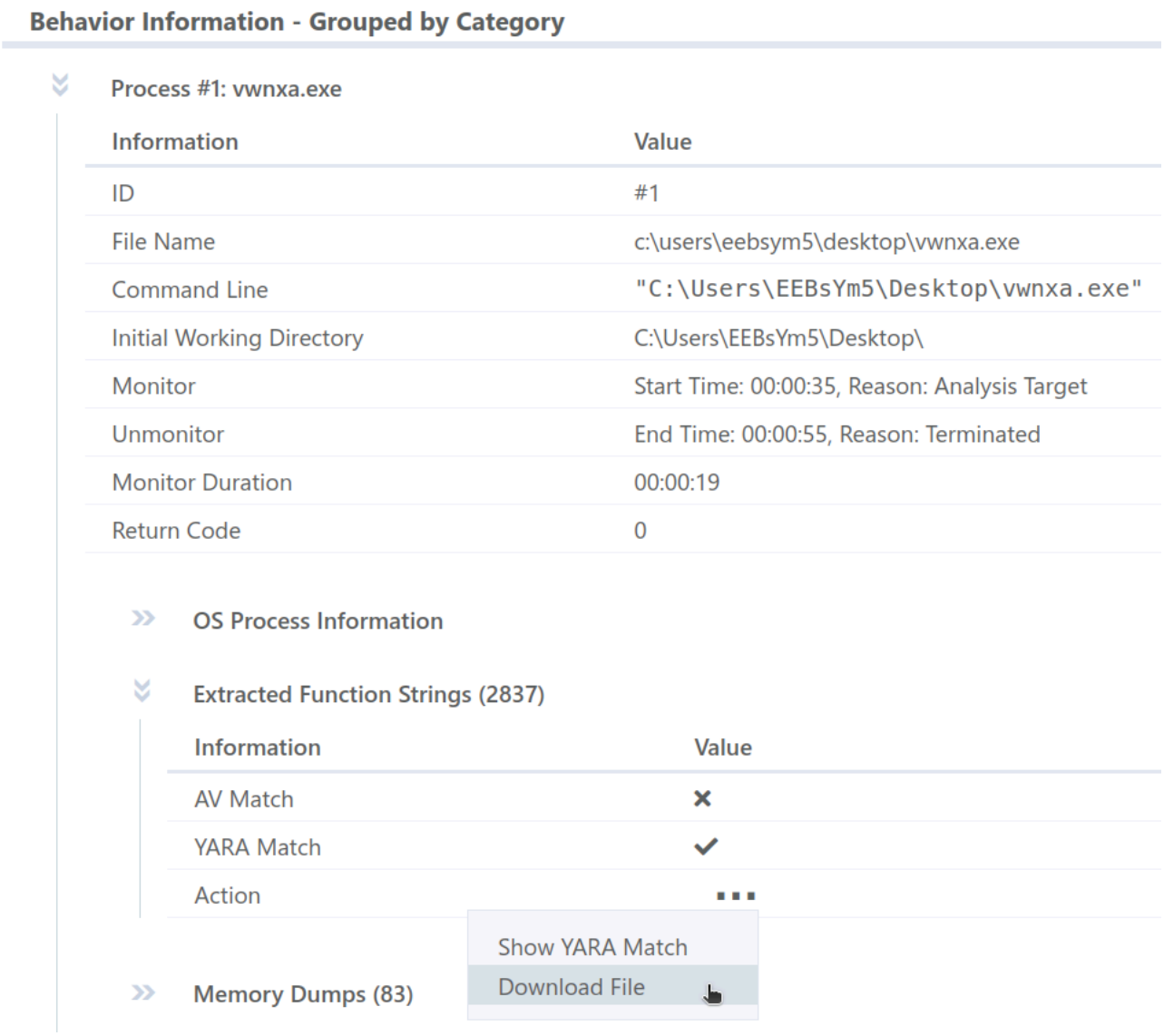
```python
# original RC4
def crypt(data):
    return [a ^ b for a, b in zip(data, keystream)]

# Stealc modified RC4
def crypt(data):
    return [a ^ b if a != b else a for a, b in zip(data, keystream)]
```

One powerful feature is VMRay's Function Strings, which is a collection of all strings that were passed as an argument to API calls during the analysis of a process. You can access this log file by going to the Behavior tab, selecting the relevant process and opening up "Extracted Function Strings" to download the file (see Figure 8).

YARA & Detection Engineering Tips

Another use of the function strings, other than helping malware researchers to extract useful information even for packed samples, is the possibility of writing YARA rules based on these runtime strings.

This opens up a robust way for detection engineers to identify certain malware families that is more resistant to code changes while threat actors continually evolve their software, often evading existing YARA rules. Basing rules on runtime function strings allows us to write more robust rules.



Additionally, obfuscated scripts are challenging to detect as they can change widely from version to version or depending on the packer, but runtime strings often reveal unique identifiers or sometimes even the unobfuscated version of the script, which allows us to write YARA rules in these difficult cases as well. For Stealc, the function strings log indeed reveals the decrypted strings, which were harder to extract before (see Figure 9). Here, we see the decrypted configuration, including the expiration date and the URL to the C2 server.

function_strings_process_1.txt

```
19/05/2023
/82de66e9459cdb5f.php
http://176.113.115.26/82de66e9459cdb5f.php
/1e46dcfeff07ca0e/
sqlite3.dll
http://176.113.115.26/1e46dcfeff07ca0e/sqlite3.dll
default
C:\Windows
%08lX%04lX%lu
```

Evasion and Obfuscation Techniques used by Stealc

We have been able to observe a number of evasion techniques that Stealc employs to avoid detection by antivirus and sandboxing technologies. One interesting technique uses a unique hardware identifier to limit a machine to just a single infection, which could be intended as a evasion techniques on platforms where the hardware ID does not change in-between analysis runs. This would limit the detonation to the very first run and avoid revealing malicious behavior in future runs.

Additionally, we have identified common evasion techniques like checking for the size of RAM or refusing to run on machines with certain language settings. In summary, we have found the following evasion and obfuscation techniques.

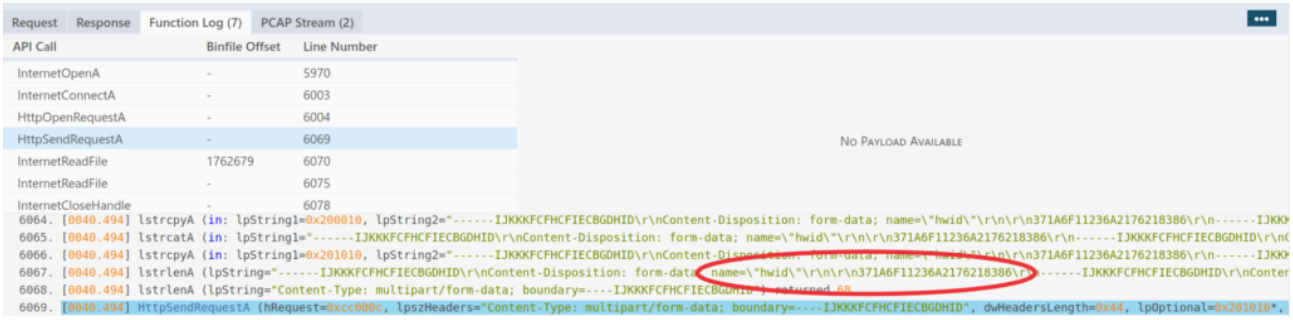
Hardware ID check

Stealc uses the serial number of the volume on the main hard disk to generate a unique identifier (see Figure 10). This hardware ID is sent to the C2 server (see Figure 11), which likely checks if this ID has ever been seen before (and thus has been infected before already), in which case the sample is terminated.

While we do not have access to the server-side code where this logic is implemented, we think there are two likely explanations for this behavior, (1) to avoid reinfecting the same machine and collecting duplicate data, and (2), as an evasion technique for dynamic, behavior based analyzers.

In the latter case, the first analysis would reveal malicious behavior while any analysis runs in the future on the same virtual machine would force the sample to terminate itself – if the volume serial number remained identical in-between runs and was thus banned.

```
GetVolumeInformationA(&RootPathName, 0, 0, (LPDWORD)&volumeSerialNumber, 0, 0, 0, 0);
hwid_part_1 = 0x14A30B * volumeSerialNumber - 0x69427551;
hwid_part_2 = 0xA30B * hwid_part_1 - 0x7551;
tmp = 0x14A30B * (0x14A30B * hwid_part_1 - 0x69427551) - 0x69427551;
for ( i = 0; i < 8; ++i )
{
  tmp = 0x14A30B * tmp - 0x69427551;
  hwid_part_3[i] = tmp;
}
volumeSerialNumber = tmp;
ProcessHeap = GetProcessHeap();
hwid = (CHAR *)HeapAlloc(ProcessHeap, 0, 0x104u);
if ( hwid )
{
  wsprintfA(hwid, (const char *)str_08lx04lxlu, hwid_part_1, hwid_part_2, _hwid_part_3);// "%08lX%04lX%lu"
```



Size of RAM

If the RAM size is smaller than 1GB, the execution is aborted as this is the case for some virtualized environments (see Figure 12).

```
if ( GlobalMemoryStatusEx(&Buffer) )
{
  v1 = Buffer.ullTotalPhys >> 20;
  result = HIDWORD(Buffer.ullTotalPhys) >> 20;
}
else
{
  v1 = 0;
  result = 0;
}
```

Limit to certain languages

Stealc does not run on machines where the user language is set to Russian, Ukrainian, Belarusian, Kazakh or Uzbek – as this is hard-coded and does not seem to be up for configuration, this is a choice made by the developers (Figure 13).

```
int check_default_language_against_blocklist()
{
  int v0; // eax
  int v1; // eax
  int v2; // eax
  int v3; // eax
  int result; // eax

  v0 = GetUserDefaultLangID() - 0x419;
  if ( !v0 || (v1 = v0 - 9) == 0 || (v2 = v1 - 1) == 0 || (v3 = v2 - 28) == 0 || (result = v3 - 4) == 0 )
    ExitProcess(0);                             // exit if Russian, Ukrainian, Belarussian, Kazakh or Uzbek
  return result;
}
```

Avoid antivirus emulators

The execution is aborted if the computer name is set to “HAL9TH” and the user name is “JohnDoe”, which is an indicator that the sample is emulated by the Windows Defender (see Figure 14).

Another check for antivirus sandboxing is implemented through a call to VirtualAllocExNuma, which is often not implemented in emulated environments..

```
[0038.891] GetUserNameA (in: lpBuffer=0x5d7898, pcbBuffer=0x17fb5c | out: lpBuffer="EEBsYm5", pcbBuffer=0x17fb5c) returned 1
[0038.906] GetProcessHeap () returned 0x5c0000
[0038.906] RtlAllocateHeap (HeapHandle=0x5c0000, Flags=0x0, Size=0x104) returned 0x5d90a0
[0038.906] GetComputerNameA (in: lpBuffer=0x5d90a0, nSize=0x17fb54 | out: lpBuffer="CRH2YWU7", nSize=0x17fb54) returned 1
[0038.906] lstrlenA (lpString="HAL9TH") returned 6
```

Indirect loading of DLL functions

Instead of importing functions statically, Stealc dynamically traverses the Process Environment Block to import DLL functions (see Figure 15), which is a well-known technique of dynamically resolving imports to avoid AV detection.

```
[0040.348] GetProcAddress (hModule=0x75c00000, lpProcName="RegEnumValueA") returned 0x75c0cf49
[0040.348] GetProcAddress (hModule=0x75590000, lpProcName="CryptBinaryToStringA") returned 0x755ca8c5
[0040.348] GetProcAddress (hModule=0x75590000, lpProcName="CryptUnprotectData") returned 0x755c5a7f
[0040.348] GetProcAddress (hModule=0x761b0000, lpProcName="SHGetFolderPathA") returned 0x762c7804
[0040.349] GetProcAddress (hModule=0x761b0000, lpProcName="ShellExecuteExA") returned 0x763f6fdd
[0040.349] GetProcAddress (hModule=0x77000000, lpProcName="InternetOpenUrlA") returned 0x770430f1
[0040.349] GetProcAddress (hModule=0x77000000, lpProcName="InternetConnectA") returned 0x770249e9
[0040.349] GetProcAddress (hModule=0x77000000, lpProcName="InternetCloseHandle") returned 0x7701ab49
[0040.349] GetProcAddress (hModule=0x77000000, lpProcName="InternetOpenA") returned 0x7702f18e
[0040.349] GetProcAddress (hModule=0x77000000, lpProcName="HttpSendRequestA") returned 0x770918f8
```

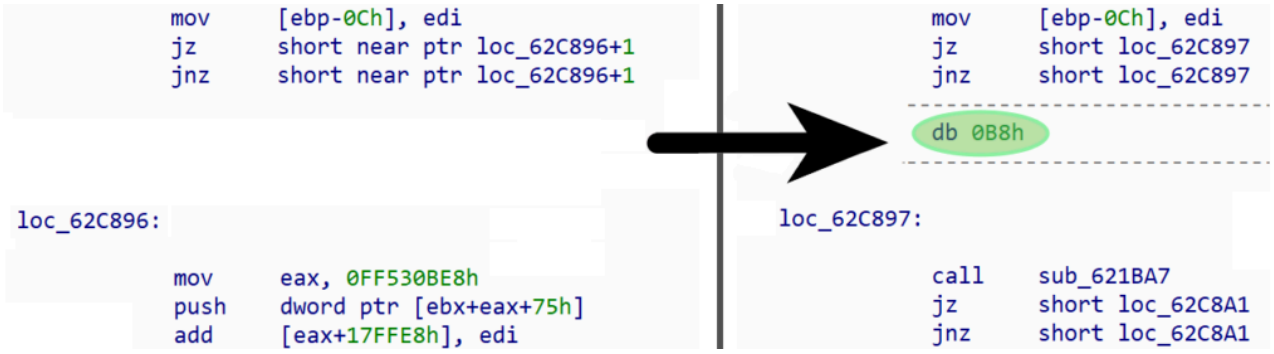
Encrypted strings / function names

As already mentioned earlier, most strings, including function names, are base64 encoded and RC4 encrypted, which are only decrypted at runtime.

Anti-disassembly

Stealc uses random bytes and jumps to confuse disassemblers and decompilers, thus impeding manual analysis by threat researchers. See the following code snippet where a random byte was placed in the middle of the code, surrounded by a jump instruction that would skip this random byte when executed.

During static analysis this would require the disassembler to have a deeper understanding of the code to avoid being fooled, which most static analysis tools do not posses and thus generate invalid code (see Figure 16). We fix these by defining the random byte as data and exclude it from being parsed as code, which reveals the correct disassembly.



Note that the behavior-based monitoring approach utilized by VMRay Platform is not confused by this as static analysis via disassembly is not necessary to detect malicious behavior on our end.

Detecting the new versions of Stealc

While researching this malware family, we found a few versions very similar to Stealc but with key differences, suggesting that these are likely updated versions. In these cases, the developers removed some of the evasion techniques, namely the check for the RAM size (see Figure 12), as well as the detections designed for antivirus emulators (see Figure 14). Notably, they have also decided to encrypt those few strings that had remained unencrypted in the original version, probably to avoid strictly YARA based detection methods.

This also demonstrates our robustness against these kinds of changes as our behavior-based VMRay Platform still identifies Stealc as malicious in the newest version.

Config Extractor

Based on this analysis, we have developed a config extractor which allows customers to peek into the configuration built into the executable by the attacker.

This enables our customers to upload a sample and get a listing of all the important configuration parameters, such as remote servers the malware communicates with, the expiration date and the encryption key (see Figure 17). In addition, the config extractor provides access to high-quality IOCs which can be used to proactively secure environments.

Malware Configurations

Stealc

Metadata	Key	Extracted Value
Encryption Key	Key Algorithm	NDYzNTI0MDA3NDU5NjY4MTQwNjA= RC4
URL	Url	http://176.113.115.26/1e46dcfeff07ca0e/82de66e9459cdb5f.php
Other: Expiration Date	Value	2023-05-19

Conclusion

Here at VMRay, we are always on the look out for malware families that have the potential to become a prominent tool among threat actors. Detecting these threats early allows us to investigate their behavior and be prepared to protect the assets of our customers.

One of the strength of VMRay Platform is it's ability to detect new attacks before malware families are known to researchers. In this regard, VMRay Platform reveals itself to be a helpful tool: our behavior-based analysis at the core of the detection engine can detect malicious behavior before static detection signatures are developed by threat researchers – in fact, threat researchers can further benefit from the detailed report on the behavior of the malware our product generates, which can be used as a strong, detailed basis to assist additional in-depth manual analysis efforts.

In malware research, one wants to quickly understand what a piece of malware does and how it is accomplished. In this post, we have seen how different features of VMRay Platform, such as the VTI's, the function log, and the function strings allow a very quick but still deep overview of what the core mechanism of the malware is in just a few minutes of analysis time.

References

https://blog.sekoia.io/stealc-a-copycat-of-vidar-and-raccoon-infostealers-gaining-in-popularity-part-1/

https://blog.sekoia.io/stealc-a-copycat-of-vidar-and-raccoon-infostealers-gaining-in-popularity-part-2/

https://www.bleepingcomputer.com/news/security/new-stealc-malware-emerges-with-a-wide-set-of-stealing-capabilities/

IOCs

Hashes:

Sample (Jan/Feb 2023)

1e09d04c793205661d88d6993cb3e0ef5e5a37a8660f504c1d36b0d8562e63a2
87f18bd70353e44aa74d3c2fda27a2ae5dd6e7d238c3d875f6240283bc909ba6
77d6f1914af6caf909fa2a246fcec05f500f79dd56e5d0d466d55924695c702d

Hashes

Sample (Mar 2023)

cde2e36ae1fef4bce98792daf14064a5e5027e5e152d653284601a698d98ef3b
464f57bb810e30c1be3765ec17bd268cfa1b4019e9ba9625329669f8385e52ab
4314a53c2c41eb8a57a933a4d1d2e3f29f9b5417074c7a12d081411418928f89

Hashes

Sample (April 2023)

660f62a2f0eb7ccae6170ec09629ade73d1874486027f22dddd92326a8e0b18e

Emre Güler
Threat Researcher
