No summer vacations for Zebrocy

welivesecurity.com/2019/09/24/no-summer-vacations-zebrocy/

September 24, 2019

ESET researchers describe the latest components used in a recent Sednit campaign

ESET Research
24 Sep 2019 - 11:30AM

ESET researchers describe the latest components used in a recent Sednit campaign

While summer is usually synonymous with vacations, it seems that the Sednit group has been developing new components to add to the Zebrocy malware family.

The Sednit group – also known as APT28, Fancy Bear, Sofacy or STRONTIUM – has been operating since at least 2004 and has made headlines frequently in recent years.

On August 20th, 2019, a new campaign was launched by the group targeting their usual victims – embassies of, and Ministries of Foreign Affairs in, Eastern European and Central Asian countries.

This latest campaign started with a phishing email containing a malicious attachment that launches a long chain of downloaders, ending with a backdoor. An example of such an email was uploaded to VirusTotal on August 22nd, two days after the mail was delivered. An overview of the attack vector was recently published by Telsy TRT.

However, we have some further pieces of this puzzle that could help to draw a more complete picture of the campaign.

As predicted by other fellow researchers, the Sednit group added a new development language in their toolset — more precisely, for their downloader: the Nim language. However, their developers were also busy improving their Golang downloader, as well as rewriting their backdoor from Delphi into Golang.

A complicated compromise

Figure 1 depicts the different steps leading to a victim being compromised, from the malicious email initially received in the inbox to the backdoor deployed on targets deemed “interesting enough” by the operators.

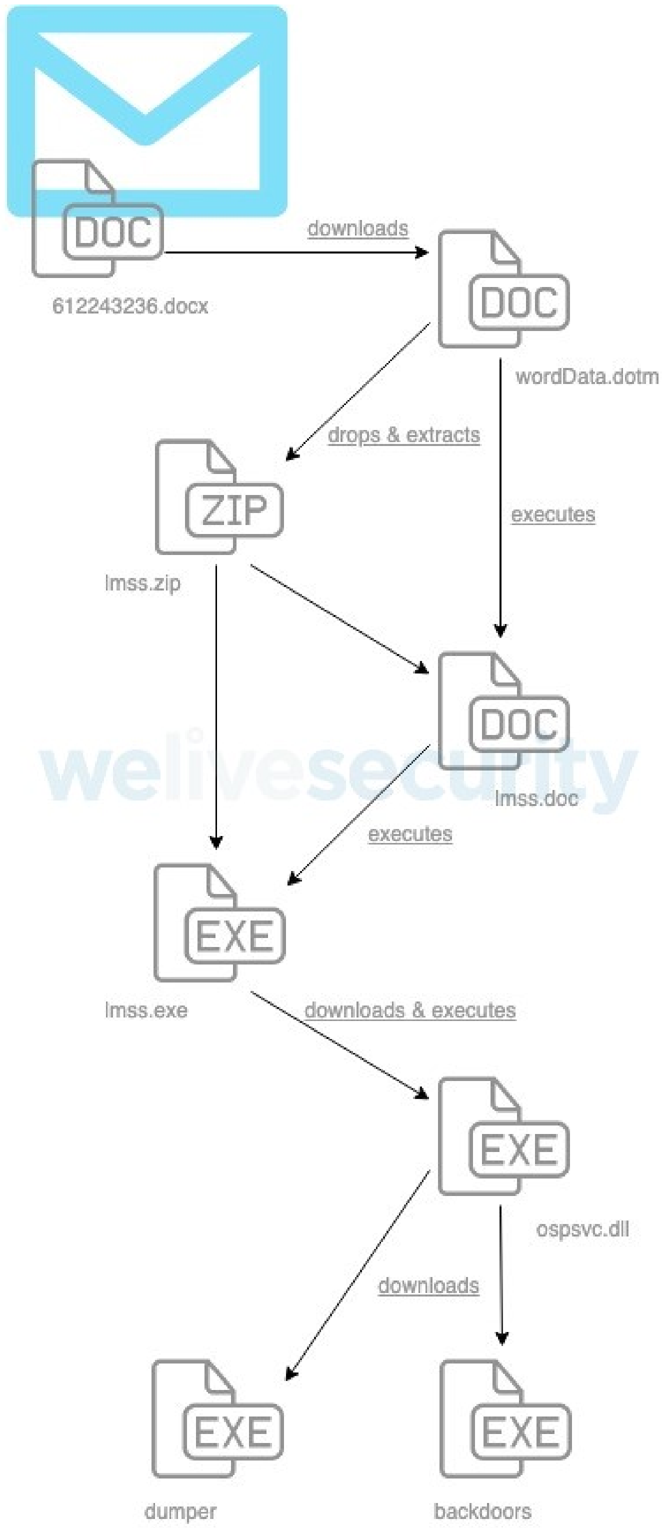

Figure 1. Chain of compromise overview

When a victim is targeted by Zebrocy's components, the chain is usually quite loud. Loud because, in this case, the victim has at least six malicious components dropped on the computer before the final payload is executed. Such activities can easily raise different types of flags for a security product.

The document attached to the phishing email is blank, but references a remote template, wordData.dotm, hosted at Dropbox. Opening this document in Word causes it to download wordData.dotm, as seen in Figure 2, and to incorporate it into the associated document's working environment, including any active content the template may contain.

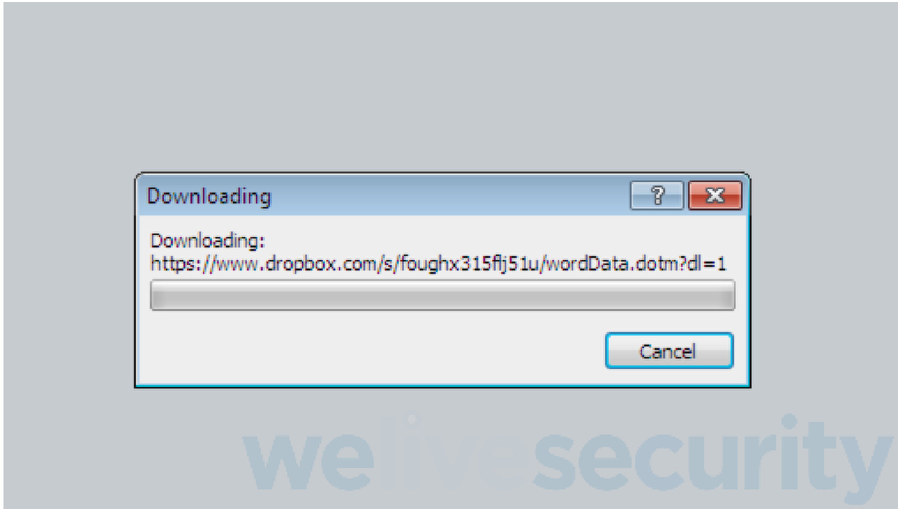

Figure 2. Empty word document downloading a remote template

The wordData.dotm file contains malicious macros that then are executed. (Depending on the Microsoft Word version, the VBA macros may be disabled by default; if so, user action is required to enable them.) It also contains an embedded ZIP archive that the macros dropped and extracted.

As shown in Figure 1, the macros in wordData.dotm open another document (lmss.doc that was unpacked from the archive extracted from wordData.dotm). Macros in lmss.doc execute lmss.exe (Zebrocy's new Nim downloader, also extracted from the archive embedded in wordData.dotm) instead of wordData.dotm executing the downloader directly.

However, it's important to notice that lmss.doc, containing the VBA code that executes the new Nim downloader, also embeds a base64-encoded executable. According to its Document Properties, lmss.doc was created in January 2019 and modified on August 20th, a few hours before the campaign started.

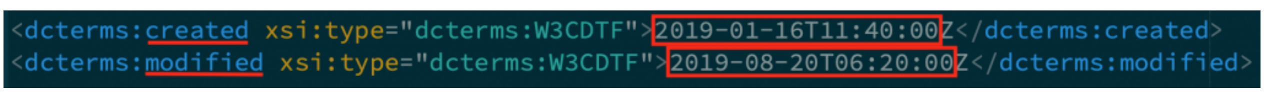

Figure 3. Creation and last modification dates of lmss.doc

The executable embedded in lmss.doc is an AutoIt downloader (SHA-1: 6b300486d17d07a02365d32b673cd6638bd384f3) used in the past for a campaign performed around the creation time of lmss.doc. Here, the AutoIt downloader is ignored and doesn't have any purpose other than making the size of the document bigger. The operator probably forgot to remove the previous embedded downloader – it would not be the first time that Sednit operators have made mistakes.

The downloaders

Sednit operators have used several downloaders written in different languages. This campaign uses the most recent extension of that list – a downloader written in the relatively new language, Nim. It's a straightforward download-and-execute binary with two small details added. The first is probably used as an anti-sandbox trick and it checks that the first letter of the executed file (letter l here or 0x6C in hex) has not changed.

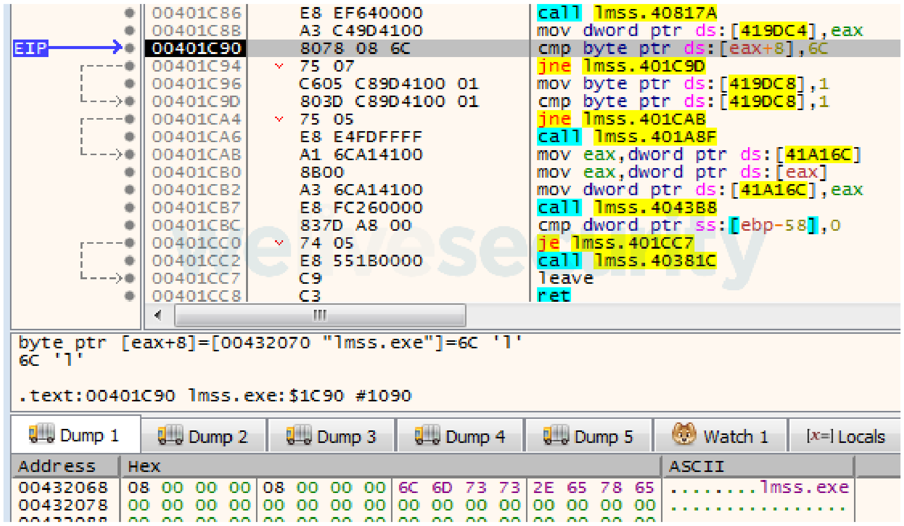

Figure 4. Filename check

The second is a kind of obfuscation where the operator replaces placeholder letters in a string with the correct ones, at defined offsets. As seen in Figure 5, the downloader reconstructs the correct download URL string with this method to avoid basic static analysis tools that could otherwise locate the URL string.

```
cnc = copyString(&::cnc);
cnc[9] = 't';
cnc[0xA] = 't';
cnc[0x12] = '.';
cnc[0x16] = '.';
v4 = cnc;
cnc[0x1D] = '/';
cnc[0x26] = '-';
cnc[0x31] = '/';
cnc[0x36] = 'p';
cnc[0x37] = 'h';
path = copyString(&::path);
filename = copyString(&filname);
v6 = 0;
v7 = filename;
*(filename + 9) = 's';
*(filename + 0xC) = 'v';
*(filename + 0xF) = 'd';
if ( path )
  v6 = *path;
v8 = rawNewString(*filename + v6 + 1);
v9 = v8;
v10 = appendString(v8, path);
v11 = appendString(v10, &separator);
appendString(v11, v7);
noscreateDir(path);
if ( nosexistsFile(v9) )
{
  v13 = nosgetFileSize(v9);
  patha = HIDWORD(v13);
  v12 = v13;
}
else
{
  v12 = 0;
  patha = 0;
}
v22 = 0;
```
welivesecurity

Figure 5. Hex-Rays output of the strings deobfuscation

For example, the string o-ps-c..ll is “patched” at three offsets by s, v and d, respectively, to give ospsvc.dll. In the case of the URL, since the beginning of the string in the downloader is h@@p://, tools looking for http:// won’t catch it.

The Nim downloader fetches its dynamic-link library (DLL) payload, named ospsvc.dll, to C:\ProgramData\Java\Oracle\, and executes it as a service via regsvr32 /s.

ospsvc.dll is a downloader written in Golang, and different from other Sednit downloaders seen in the past.

Sednit’s previous Golang downloaders have been described in detail by other researchers [1][2][3] and it seems that Sednit’s developers have rewritten their previous Delphi downloader in Golang. Those earlier downloaders gather a lot of information about the victim computer and send it to their C&C server. However, this new one is quite light in terms of its data-gathering capabilities, as described below.

Its function main_init() contains libraries that are initialized and don't need further explanations due to their names (see this article for more information).

```
syscall_init();
bytes_init();
crypto_tls_init();
encoding_base64_init();
image_jpeg_init();
io_init();
io_ioutil_init();
isudh_init();
math_rand_init();
mime_multipart_init();
net_http_init();
os_init();
os_exec_init();
os_user_init();
path_filepath_init();
regexp_init();
strings_init();
result = time_init();
```

Figure 6. Hex-Rays output of initialized functions in the main_init() *using the IDAGolangHelper plugin*

Since the DLL is run as a service, via the Nim downloader, we need to look at main_DllRegisterServer() instead of main_main(). The strings and the key are stacked and they can be decrypted using a simple XOR loop. This simple encryption is quite efficient against tools that extract strings stacked from binaries statically.

Figure 7. IDA Pro output of encrypted strings stacked

Aside from downloading the next stage of the malware, taking screenshots of the victim's desktop and executing commands received from the C&C server are the main functions of this malware.

Screenshots are taken every 35 seconds during the first few minutes of this downloader's execution, and then they are sent to the C&C server in base64-encoded form. The hostname and the %USERPROFILE% values are also sent to the C&C server encoded in base64. The reply from the C&C server is also straightforward: it's a concatenation of base64-encoded strings, separated by "|".

<spaces>|<cmd to execute>|<name of the binary to drop>|<binary to drop>

According to our telemetry, this downloader has been used to execute three different pieces of malware. The first one is the dumper that we described in our previous Zebrocy article. The second one is the usual Delphi backdoor – also run as a service via the same command used by the Nim downloader. The third one we saw is a new backdoor downloaded and executed on the victim's machine, as described in the next section.

The new backdoor

Analysis

The new Zebrocy backdoor is not written in Delphi as we are used to, but in Golang. To the best of our knowledge, this is the first time this backdoor has been seen, but it shares a lot of similarities with the Delphi one.

By looking again at the main_init() function's library initialization code (Figure 8) we can see new entries. An AES algorithm, hex encoding, and screenshot capabilities are the main entries that were added.

```
 syscall_init();
 bytes_init();
+crypto_aes_init();
+crypto_cipher_init();
 crypto_tls_init();
 encoding_base64_init();
-image_jpeg_init();
+encoding_hex_init();
+fmt_init();
+image_init();
+image_png_init();
 io_init();
 io_ioutil_init();
-isudh_init();
 math_rand_init();
 mime_multipart_init();
 net_http_init();
@@ -13,6 +17,8 @@
 os_exec_init();
 os_user_init();
 path_filepath_init();
+reflect_init();
 regexp_init();
+strconv_init();
 strings_init();
 time_init();
```

welivesecurity

Figure 8. Diff between the backdoor and the downloader functions initialized in the main_init()

Notice that image_png_init replaces image_jpeg_init for taking screenshots. Images in JPG format are usually smaller in size than the PNG format.

The backdoor is started with an argument that is a hex-encoded string. All but the last six-byte chunk of this string is XOR-encrypted with the key stored in the last six bytes of the string. The following python snippet describes the decryption logic.

```
key = arg[-6:].decode('hex')
enc = arg[:-6].decode('hex')
''.join(chr(ord(i) ^ ord(j)) for i, j in zip(itertools.cycle(key), enc))
```

It’s the address of the C&C server, which is later encrypted and stored on disk. That encryption is done using the AES-128 ECB algorithm with a key generated from the hostname. Hence, there is no possibility to obtain this C&C server just by looking at the binary. There is no persistence defined by the downloaders as we have seen in the past, nor does the backdoor have any persistence mechanism.

This new backdoor has various capabilities that were also previously seen in Zebrocy’s Delphi backdoor:

- file manipulation such as creation, modification, and deletion
- screenshot capabilities
- drive enumeration
- command execution (via cmd.exe)
- schedule a task under the following name Windows\Software\OSDebug (which the operators could use to set persistence manually)

As in the Delphi backdoor, there is a very limited set of commands – but the ability to execute arbitrary commands via cmd.exe extends possibilities like persistence or information gathering. Another similarity found is a three-digit version number (in the format x.y.z); the current major version is 4.y.z.

Network

The network protocol shares some similarities with the Delphi version of the backdoor. The first interaction with the C&C server retrieves an AES 32-bit key to encrypt future communications. The packet capture of that first request looks like this:

```
POST [REDACTED URI] HTTP/1.1
Host: [REDACTED IP]
User-Agent: Go-http-client/1.1
Content-Length: 297
Content-Type: multipart/form-data; boundary=b116f1e0a94eff1bb406531e74bb0feba65687cf90ec8a64fc409f230fbd
Accept-Encoding: gzip

–b116f1e0a94eff1bb406531e74bb0feba65687cf90ec8a64fc409f230fbd
Content-Disposition: form-data; name=”filename”; filename=”[REDACTED]”
Content-Type: application/octet-stream

1
–b116f1e0a94eff1bb406531e74bb0feba65687cf90ec8a64fc409f230fbd–
```

Those with experience with Sednit might think that the Content-Disposition and boundary keywords look familiar. They were previously used by the Delphi backdoor in its network protocol; it also uses the AES algorithm to encrypt the pseudo body (content after the Content-Type data). Notice that even if Content-Disposition and the second instance of Content-Type are real HTTP headers, here they are used inside the HTTP message body. The boundary field is randomized for every exchange and the filename field inside the pseudo Content-Disposition header can be decrypted with the following snippet of Python:

```
len_filename = len(filename)
len_key = 14
xor_key = filename[-len_key:].decode('hex')
filename = filename[:len_filename-len_key].decode('hex')
val_filename = ''.join(chr(ord(i)^ord(j)) for i,j in zip(itertools.cycle(xor_key),filename))
random_int = val_filename[-4:]
```

which results in the following string:

757365722D5043181620201908211510552O7C.inc

That string can be further understood thus:

Username: 757365722D5043
SID*: 181620
Date: 20190821151055
Random: 207C.inc

* 6 bytes comes from the current user's Security Identifiers (SID) S-1-5-21-xxxxxxxxx-yyyyyyyyy-**zzzzzz**zzzz-1000

Further interactions with the C&C server follow this pattern except that the pseudo body, which is 1 in the example above, is replaced by the output of the command requested by the C&C server. The full message body is also encrypted, using the same AES algorithm used previously, with the key provided in the first exchange.

Conclusion

New downloaders, new backdoor – the Sednit group has been active and is not letting their components get too old. New? Not really. By looking at it, it seems that the Sednit group is porting the original code to, or reimplementing it in, other languages in the hope of evading detection. It's probably easier that way and it means they do not need to change their entire TTPs. The initial compromise vector stays unchanged, but using a service like Dropbox to download a remote template is unusual for the group.

ESET recommends being attentive when users are opening attachments from suspicious emails.

We will continue to monitor new activities from the Sednit group and will publish relevant information on our blog. For any inquiries, contact us at threatintel@eset.com.

Indicators of Compromise (IoCs)

Hashes (SHA-1)	Filenames	ESET detection names
c613fcccb380f7e3ce157c4f620efca503c1bad3	- (eml file)	DOC/TrojanDownloader.Agent.AMY
6f281b30d8d6a9bc1dbe2fe73995aac382c4a543	612243236.docx	DOC/TrojanDownloader.Agent.AMY
f3f945fb22916f82cb7407cde2a80a68cd83b074	wordData.dotm	VBA/TrojanDropper.Agent.AIP
a56af5b44624e8ada60057fd7f39af5b3de10724	lmss.zip	Win32/TrojanDownloader.Sednit.BK
b8ac400e1deb6e90fa4e2adb150c511c98bafc6e	lmss.doc	VBA/TrojanDropper.Agent.AIQ
f0793e02180f3ccf48e41bd67ec1161d93f07e01	lmss.exe	Win32/TrojanDownloader.Sednit.BK
04303024ff453f918925d7160abbd199f137a442	ospsvc.dll	Win32/Sednit.DI
c96db85ece2b57a9e82ba36b5f31ca9d2051a6f0	osppsvc.exe	Win32/Sednit.DJ

Network

https://www.dropbox[.]com/s/foughx315flj51u/wordData.dotm?dl=1

185.221.202[.]35

MITRE ATT&CK techniques

Tactic	ID	Name	Description
Initial Access	T1193	Spearphishing Attachment	Zebrocy is using spearphishing emails with an attachment as method of compromise.
Execution	T1059	Command-Line Interface	The Golang backdoor uses cmd.exe to execute commands.
T1117	Regsvr32	The Nim downloader uses regsvr32.exe to launch the Golang downloader.	
T1053	Scheduled Task	The Golang backdoor can create a pre-defined scheduled task.	
T1064	Scripting	The remote template contains VBA used to execute the next stage of the malware.	
T1204	User Execution	Zebrocy attempts to get users to click on Microsoft Office attachments containing malicious macro scripts.	
Persistence	T1053	Scheduled Task	The Golang backdoor can create a pre-defined scheduled task.

Tactic	ID	Name	Description
Privilege Escalation	T1053	Scheduled Task	The Golang backdoor can create a pre-defined scheduled task.
Defense Evasion	T1107	File Deletion	The Golang backdoor can delete files.
T1117	Regsvr32	The Nim downloader uses regsvr32.exe to launch the Golang downloader.	
T1064	Scripting	The remote template contains VBA used to execute the next stage of the malware.	
Discovery	T1083	File and Directory Discovery	The Golang backdoor can list drives.
Collection	T1113	Screen Capture	HTTP is used for C&C communications.
Command and Control	T1043	Commonly Used Port	All components are using port 80 to communicate with the C&C server.
T1024	Custom Cryptographic Protocol	The Golang backdoor is using an XOR loop for its communications.	
T1132	Data Encoding	The Golang backdoor base64-encodes the data before encrypting it.	
T1071	Standard Application Layer Protocol	HTTP is used for C&C communications.	
T1032	Standard Cryptographic Protocol	The Golang backdoor encrypts communications with the C&C server with AES ECB.	
Exfiltration	T1022	Data Encrypted	The Golang backdoor encrypts the data with AES ECB before sending it over the C&C server.
T1041	Exfiltration Over Command and Control Channel	The Golang backdoor exfiltrates data to its C&C server.	

References:

[1] https://unit42.paloaltonetworks.com/sofacy-creates-new-go-variant-of-zebrocy-tool/

[2] https://securelist.com/a-zebrocy-go-downloader/89419/

[3] https://www.vkremez.com/2018/12/lets-learn-dissecting-apt28sofacy.html

24 Sep 2019 - 11:30AM

Sign up to receive an email update whenever a new article is published in our Ukraine Crisis – Digital Security Resource Center

Newsletter

Discussion