ESET RESEARCH

Sednit: What's going on with Zebrocy?

In August 2018, Sednit's operators deployed two new Zebrocy components, and since then we have seen an uptick in Zebrocy deployments, with targets in Central Asia, as well as countries in Central and Eastern Europe, notably embassies, ministries of foreign affairs, and diplomats

ESET Research
20 Nov 2018 , 16 min. read

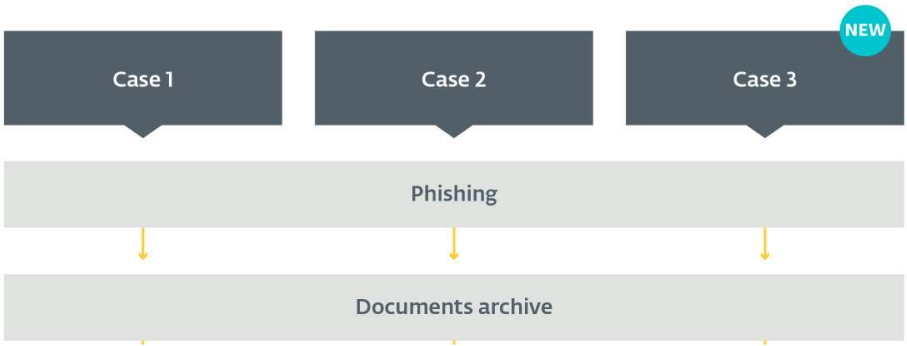
The Sednit group has been operating since at least 2004, and has made headlines frequently in past years: it is believed to be behind major, high profile attacks. For instance, the US Department of Justice named the group as being responsible for the Democratic National Committee (DNC) hack just before the US 2016 elections. The group is also presumed to be behind the hacking of global television network TV5Monde, the World Anti-Doping Agency (WADA) email leak, and many others. This group has a diversified set of malware tools in its arsenal, several examples of which we have documented previously in our Sednit white paper from 2016.

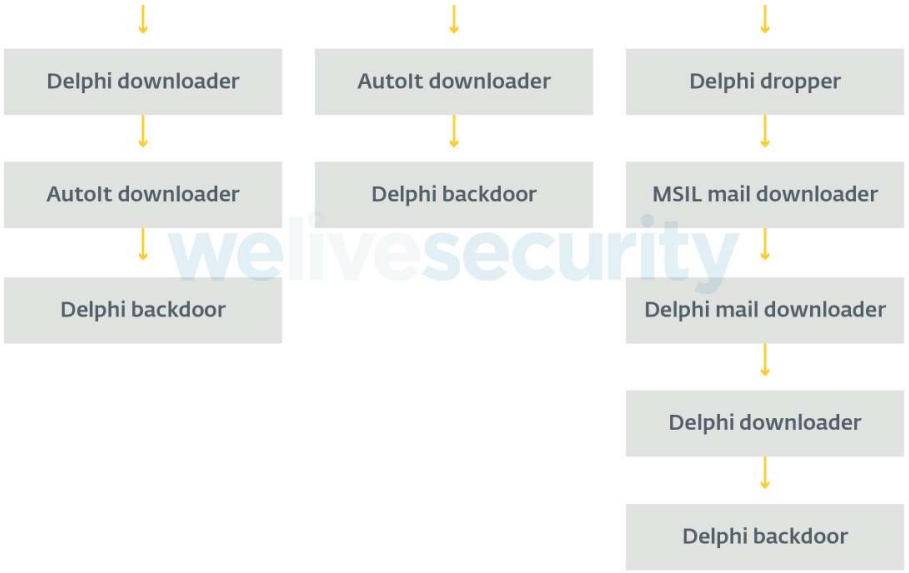
Meanwhile, ESET researchers released a whitepaper on LoJax, a UEFI rootkit we attribute to Sednit, used against organizations in the Balkans, and Central and Eastern Europe.

In August 2018, Sednit's operators deployed two new Zebrocy components, and since then we have seen an uptick in Zebrocy deployments. Zebrocy is a set of downloaders, droppers and backdoors; while downloaders and droppers are doing reconnaissance, backdoors implement persistence and spying activities against the target. These new components use a an unusual way to exfiltrate gathered information by using protocols related to mail services such as SMTP and POP3.

The victims targeted by these new components are similar to victims mentioned in our previous Zebrocy post and by Kaspersky. The targets of such attacks are located in Central Asia, as well as countries in Central and Eastern Europe, notably embassies, ministries of foreign affairs, and diplomats.

The big picture





For two years now, the Sednit group has primarily used phishing emails as the infection vector for Zebrocy campaigns (Case 1 and Case 2). Once the targets have been compromised, they use different first stage downloaders to gather information about the victims and, should the victims be interesting enough, after a delay of several hours – or even days – they deploy one of their second-level backdoors.

The classic modus operandi for a Zebrocy campaign is for the victim to receive an archive attached to an email. This archive contains two files, one a benign document and one an executable. The operator tries to fool the victim by naming the executable with an apparent document or image file name by incorporating the "double extension" trick..

This new campaign, depicted as Case 3 in Figure 1, uses a more involved procedure. We dissect this process below.

Delphi dropper

The first binary is a Delphi dropper, which is kind of unusual for a Zebrocy campaign. Most of the time it's a downloader rather than a dropper that is installed on the victim system as the first stage.

This dropper contains some tricks to make it more difficult to reverse-engineer. It uses a keyword – ***liver*** in the samples we

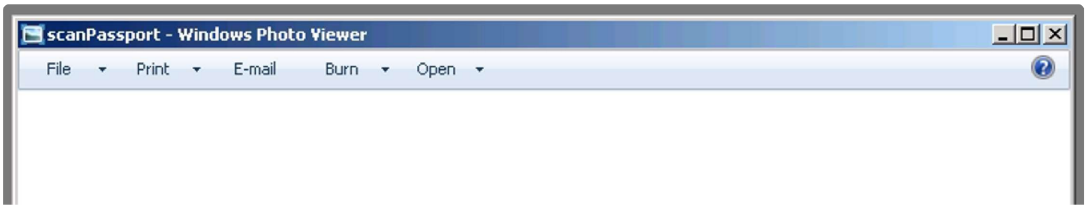
reverse-engineer. It uses a keyword – ***liver*** in the samples we described here – to mark the start and end of key elements as shown below.

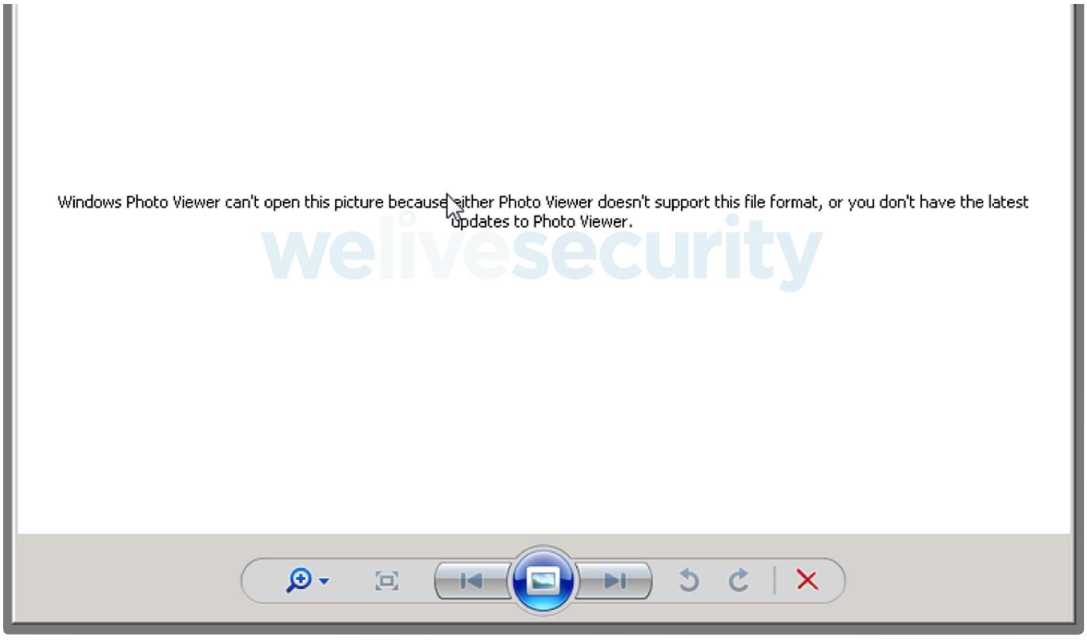
```
$ yara -s tag_yara.yar  SCANPASS_QXWEGRFGCVT_32380348890
find_tag SCANPASS_QXWEGRFGCVT_323803488900X_jpeg.exe
0x4c260:$tag: l\x00i\x00v\x00e\x00r\x00
0x6f000:$tag: liver
0x6f020:$tag: liver
0x13ab0c:$tag: liver
```

The YARA rule above looks for the string ***liver***. The first ***liver*** is the one used in the code and it doesn't separate anything from anything, while the others separate the key descriptor, the image (hexdump below) and the encrypted payload in the dropper.

```
$ hexdump -Cn 48 -s 0x6f000  SCANPASS_QXWEGRFGCVT_323803
0006f000  6c 69 76 65 72 4f 70 65  6e 41 69 72 33 39 30
0006f010  35 5f 42 61 79 72 65 6e  5f 4d 75 6e 63 68 65
0006f020  6c 69 76 65 72 ff d8 ff  e0 00 10 4a 46 49 46
```

Starting with the image, this is dropped as ***C:\Users\public\Pictures\scanPassport.jpg*** if a file of that name does not already exist. Interestingly, the dropper's filename, ***SCANPASS_QXWEGRFGCVT_323803488900X_jpeg.exe***, also hints at a phishing scheme revolving around traveling or passport information. This might indicate that the operator knew the phishing message's target. The dropper opens the image: if the file exists, it stops executing; otherwise, it drops the image, opens it, and retrieves the key descriptor ***OpenAir39045_Bayren_Munchen***. The image doesn't seem to display anything while the file format is valid; see Figure 2.





The key descriptor's string contains ***Bayren_Munchen*** which seems likely to refer to the German soccer team FC Bayern Munich. Regardless, it is not the content of the key descriptor – but its length – that matters, with that length used to retrieve the XOR key used to encrypt the payload.

To get the XOR key, the dropper looks for the last ***liver*** keyword and adds the offset of the key descriptor. The length of the XOR key – 27 (0x1b) bytes – is the same as that of the key descriptor.

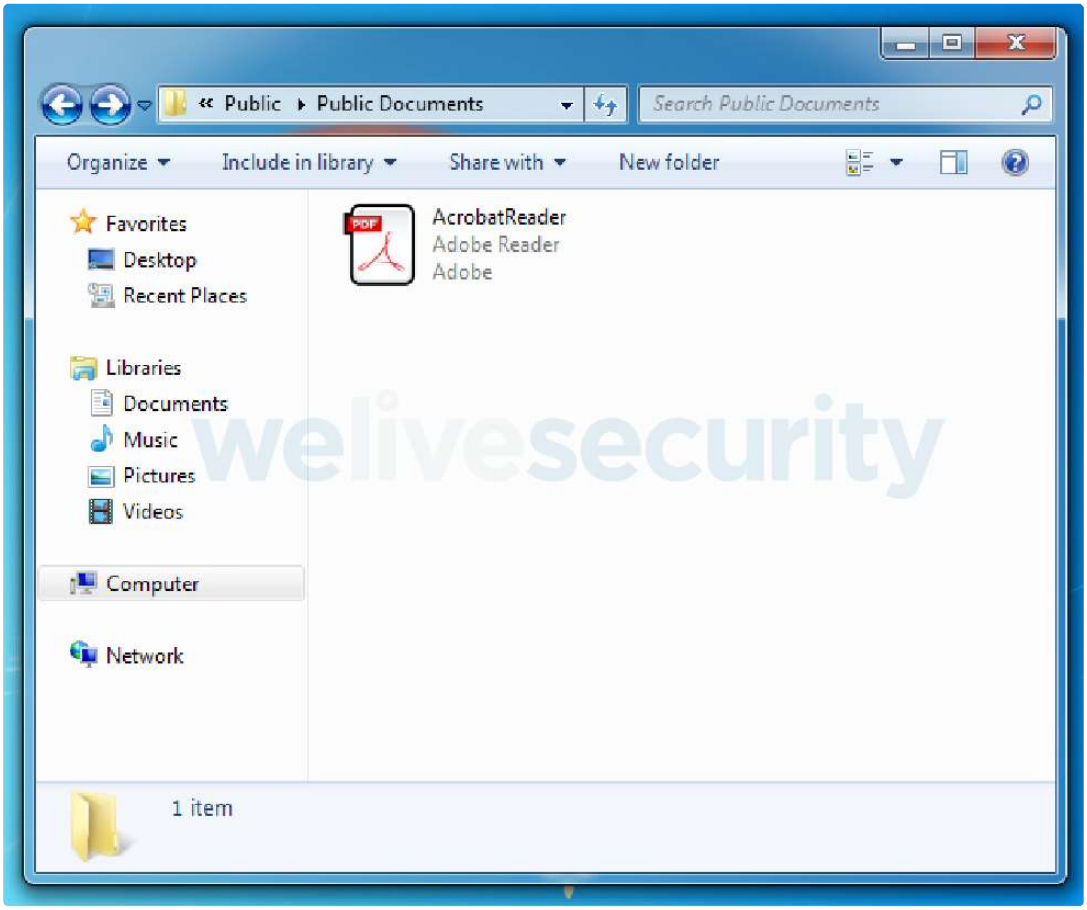
Using the XOR key and a simple XOR loop, the dropper decrypts the last part – which is the encrypted payload – right after the last tag until the end of the file. Notice that the executable payload's MZ header starts right after the keyword ***liver*** and the XOR key retrieved from a part of the PE header that is normally a sequence of


```
$ diff -W200 -y <(xxd -s 0x13ab08 SCANPASS_QXWEGRFGCVT_323803488900X_jpeg.exe) <(xxd -s 0x13ab08 decrypted)
0013ab08: a571 ffd9 6c69 7665 7219 31ee 7559 7269  .q..liver.1.uYri     |    0013ab08: a571 ffd9 6c69 7665 724d 5a90 0003 0000  .q..liverMZ.....
0013ab18: 2826 2b2f 2ebb a67a 62d1 7e75 4456 6e75  (&+/...zb.~uDVnu     |    0013ab18: 0004 0000 00ff ff00 00b8 0000 0000 0000  ................
0013ab28: 7833 7e75 546b 7e75 5a72 6928 222b 2f2e  x3~uTk~uZri("+/.     |    0013ab28: 0040 0000 0000 0000 0000 0000 0000 0000  .@..............
0013ab38: 4459 7a62 697e 7544 566e 7578 737e 7554  DYzbi~uDVnuxs~uT     |    0013ab38: 0000 0000 0000 0000 0000 0000 0000 0000  ................
0013ab48: 6b7e 755a 72e9 2822 2b21 31fe 577a d660  k~uZr.("+!1.Wz.      |    0013ab48: 0000 0000 0080 0000 000e 1fba 0e00 b409  ................
0013ab58: b354 fc57 22b8 5927 161c 274b 0e07 3515  .T.W".Y'..'K..5.     |    0013ab58: cd21 b801 4ccd 2154 6869 7320 7072 6f67  .!..L.!This prog
0013ab68: 1b49 4f0b 4c4f 2a37 1516 491c 1064 241b  .IO.LO*7..I..d$.     |    0013ab68: 7261 6d20 6361 6e6e 6f74 2062 6520 7275  ram cannot be ru
0013ab78: 1b58 1a10 5510 242d 5537 1d0d 4d0c 2622  .X..U.$-U7..M.&"     |    0013ab78: 6e20 696e 2044 4f53 206d 6f64 652e 0d0d  n in DOS mode...
```

It drops the payload as ***C:\Users\Public\Documents\AcrobatReader.txt*** and moves the file to ***C:\Users\Public\Documents\AcrobatReader.exe***

Perhaps this is an attempt to avoid endpoint protection systems triggering an alert based on a binary dropping a file with a .exe extension.

Once again, the operator tries to fool victims in the event that they take a look at the directory, in which case they'll see the file displayed as in Figure 4:



By default, Windows hides the extension, and this is leveraged by the operator to drop an executable in a Documents directory and it makes it look like a PDF file.

Finally, the dropper executes its freshly-dropped payload, and exits.

MSIL mail downloader

The payload of the previous dropper is a UPX-packed MSIL downloader. To make the process easier to understand, the main logic is described below, followed by source code, then an overview

of the dissected control flow.

The Main method calls ***Run*** to start the application, which then creates the form ***Form1***.

```
{
    Application.EnableVisualStyles();
    Application.SetCompatibleTextRenderingDefault(false)
    Application.Run((Form) new Form1());
}
```

Form1 initiates a lot of variables, including a new Timer for seven of them.

```
this.start = new Timer(this.components);
this.inf = new Timer(this.components);
this.txt = new Timer(this.components);
this.subject = new Timer(this.components);
this.run = new Timer(this.components);
this.load = new Timer(this.components);
this.screen = new Timer(this.components);
```

A Timer object has three important fields:

- Enabled: indicates if the timer is active.
- Interval: the time, in milliseconds, between elapsed events.
- Tick: the callback executed when the timer interval has elapsed and the timer is enabled.

Here these fields are initialized as:

```
this.start.Enabled = true;
 this.start.Interval = 120000;
 this.start.Tick += new EventHandler(this.start_Tick)
 this.inf.Interval = 10000;
 this.inf.Tick += new EventHandler(this.inf_Tick);
```

```
this.txt.Interval = 120000;
this.txt.Tick += new EventHandler(this.txt_Tick);
this.subject.Interval = 120000;
this.subject.Tick += new EventHandler(this.subject_T
this.run.Interval = 60000;
this.run.Tick += new EventHandler(this.run_Tick);
this.load.Interval = 120000;
this.load.Tick += new EventHandler(this.load_Tick);
this.screen.Interval = 8000;
this.screen.Tick += new EventHandler(this.screen_Tic
```

For each object, it sets an ***Interval*** which is from 8 seconds to 2 minutes. A callback is added to the event handler. Notice that ***start*** is the only one that sets ***Enabled*** as true, meaning that after 2 minutes (12000 milliseconds = 120 seconds) ***start_Tick*** will be called by the event handler.

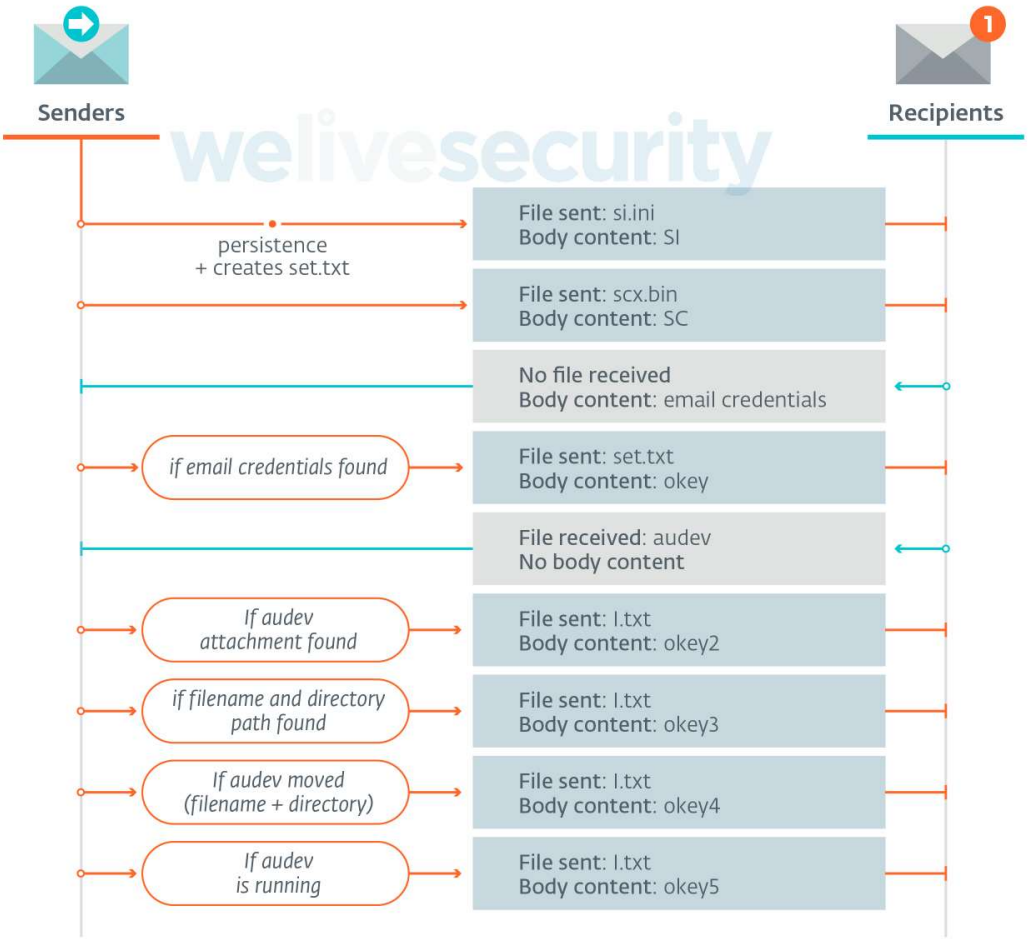
```
private void start_Tick(object sender, EventArgs e)
{
    try
    {
        this.start.Enabled = false;
        Lenor lenor = new Lenor();
        this.dir = !Directory.Exists(this.label15.Te
        this.att = this.dir + "audev.txt";
        this._id = lenor.id(this.dir);
        this.inf.Enabled = true;
    }
```

Thereafter each method has the same behavior: it sets the ***Enabled*** variable to ***false*** at the beginning of the method. The method occurs, and afterwards sets the ***Enabled*** variable of the next object to ***true***, which will activate the next timer. The ***Enabled*** variable is used by the operator to put in place a kind of state machine: if the functions fail, this is a mechanism to repeat failed functions until they succeed. The time between the execution of two functions might be an attempt to evade endpoint protection systems by

adding a delay.

Now the structure of each method is defined; the following part will focus on the control flow of the malware.

As the exchanges happen between different email inboxes, here is an overview of the different steps.



One of the early checks made by the malware is for the existence of a specific path used to drop every file used during its execution. If possible, it uses ***C:\Users\Public\Videos*** -- otherwise it will fall back to ***C:\Documents and Settings\All Users\Documents*** as its default directory. Notice that the latter path is specific to Windows XP while the former is for Vista and above.

A 16-byte ***id*** is generated by concatenating the ***C:*** volume serial number and the UserName, and stored in the file ***audev.txt***.

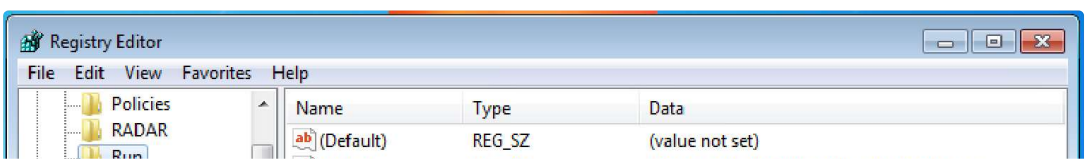
The downloader gathers the following information:

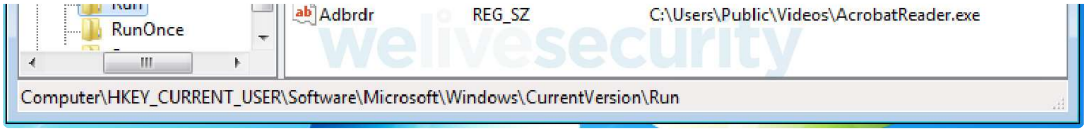
- current path of the application
- operating system version
- system directory
- user domain
- machine name
- UserName
- current time zone
- current date
- logical drive list and information about each of them (model, serial number, etc...)
- directory listing of ***C:\Program Files*** and ***C:\Program Files (x86)***
- process list

All this information is stored in the ***C:\Users\Public\Videos\si.ini*** file and sent in an email message, as an attachment, via SMTPS, using the default port 465. The email body contains the string ***SI*** (which probably stands for System Information), the recipient is ***sym777.g@post.cz***. For all email exchange, the message's Subject: set to the ***id***.

The operator chooses to have multiple fallback addresses and sends the same email to two other, different recipients, presumably in case the main one is not working. Once the email has been sent, the downloader deletes the ***si.ini*** file.

For the first execution of the malware, it creates the file ***set.txt*** with ***{System_Parametrs = 10}*** inside and creates the Windows registry entry:





One screenshot of the victim's computer is taken under the name ***scx.bin*** and sent as an email attachment with ***SC*** (which probably stands for Screenshot) in the email's body.

After dispatch, this malware connects to the ***kae.mezhnosh@post.cz*** mailbox via POP3 over SSL (port 995) and looks for messages with a Subject: that corresponds to its own ***id***. If there is such a message and the body is not empty, the malware hex decodes it and then sends a message with ***okey*** in the body to ***sym777.g@post.cz***. The content of the email previously retrieved is cleaned and parsed as below:

```
string[] strArray = this._adr.Replace("B&", "").
string str1 = strArray[0];
string str2 = strArray[1];
```

Two strings are obtained: the first one is a password and the second is a username for an email address.

These new credentials are used to connect to the specified inbox freshly collected, and to also look there for a message with a subject that matches the malware's ***id*** as well as an attachment with the string ***audev*** in its filename. If both conditions are met, the malware saves the attachment and deletes the message from the server.

All logging messages are sent to ***sym777.g@post.cz*** while messages retrieved via POP3 come from credentials recently obtained.

These decisions from the operators make forensics more difficult. First, if you have the downloader with emails, you can't connect to the mailbox that contains the next stage.

Second, if you retrieve the email credentials, you still can't get the next payload because it was deleted after retrieval.

Once the downloader successfully writes the attachment to disk, it sends an email with ***okey2*** in the body and an attachment, named ***l.txt,*** containing ***090***. The same file is overwritten with ***000*** and the malware tries to retrieve another message. Again – if it works – the ***l.txt*** file is sent with body text of ***okey3***. The content of the attachment is a directory and a filename. The malware moves the ***audev*** file to this filepath. Finally, the malware sends an email with a body message of ***okey4*** and ***l.txt*** as attachment. It starts the executable — ***audev.exe*** and checks in the list of processes to see if one of them contains the string ***audev***.

```
Process.Start(this.rn);
 foreach (Process process in Process.GetProcesses())
 {
     if (process.ProcessName.Contains("audev"))
 }
```

If a process with such a name is found, it sends a last email with ***okey5*** as the body message and ***l.txt*** as the attachment. Finally, it deletes ***l.txt*** and ***set.txt***, deletes the Windows registry key it created, and exits.

Delphi mail downloader

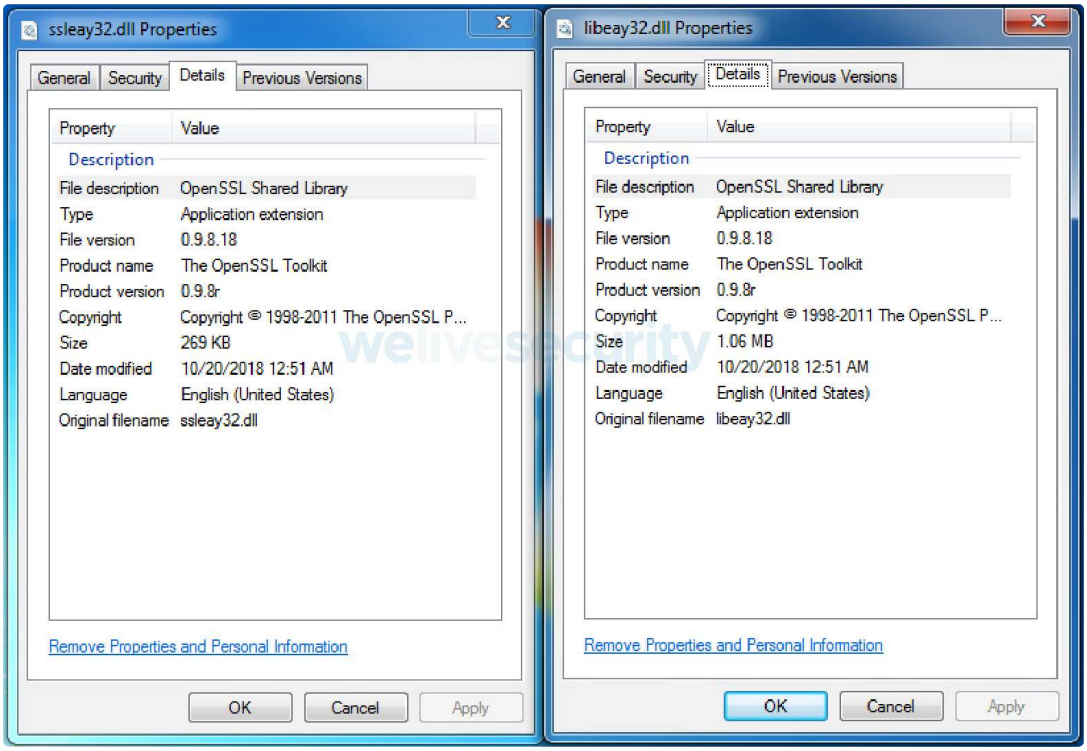
The main role of this downloader is to assess the importance of the compromised system and, if it is deemed important enough, to download and execute Zebrocy's last downloader.

The binary is written in Delphi and packed with UPX. The complete definition of the ***TForm1*** object can be found in its resource section and contains some configuration parameters used by the malware. The following sections focus on the initialization, capabilities, and network protocol of the downloader.

Initialization

At the beginning, it decrypts a bunch of strings that are email addresses and passwords. The operator uses the AES ECB encryption algorithm. Each string is hex-encoded, with the first four bytes corresponding to the final size of the decrypted string (the decrypted strings may contain some padding at the end). There are two AES keys in the ***TForm1*** object; the first one is used to encrypt data while the second is used to decrypt.

Emails and passwords are used by the operator to send commands to the malware and also to retrieve information harvested from the victim's computer. The communication protocols are SMTP and POP3 – both of them over SSL. To use OpenSSL, the malware drops and uses two OpenSSL dynamic link libraries (DLLs): ***libeay32.dll*** (***98c348cab0f835d6cf17c3a31cd5811f86c0388b***) and ***ssleay32.dll*** (***6d981d71895581dfb103170486b8614f7f203bdc***).



Notice that all files are dropped in the malware's working directory, ***C:\Users\Public***

The persistence is done during the first execution of the malware using a well-known technique, the "Logon scripts". It creates a script file ***registration.bat*** and writes several strings from the ***TForm1*** object. The final script is:

```
reg add HKCU\Environment /v "UserInitMprLogonScript" /t
del C:\Users\Public\Videos\registr.bat
exit
```

Last but not least, the malware creates an ***id***, in the same way as seen in previous Zebrocy binaries. It retrieves the UserName via the ***GetUserNameW*** Windows API and prepends the volume serial number of the C:\ drive.

Capabilities

While there are some conditions and order in the execution flow to collect information about the victim, the following section describes different gathering capabilities. The scan configuration is stored in the ***TForm1*** object, grouped under seven different possibilities for retrieving information from the victim's computer.

Starting with a simple scan, the first information that the malware can collect is related to files with the following extensions: ***.docx, .xlsx, .pdf, .pptx, .rar, .zip, .jpg, .bmp, .tiff***. For each file found on the disk, it retrieves the full path and the last modified date of the file. That information is encrypted using the AES key mentioned earlier and stored in the file ***0.txt***. Another scan targets the extensions ***.dat, .json, .db*** and like the previous scan it retrieves the full path and last modified date of the file. Then it encrypts them and it stores it under the file ***57.txt***.

Listing running processes is also one of the capabilities of this malware and it stores that information in the ***08.txt*** file, and this looks like the listing below:

```
======Listing_of_processes=======
[System Process]
System
smss.exe
csrss.exe
```

```
wininit.exe
csrss.exe
winlogon.exe
services.exe
lsass.exe
[…]
```

In the file ***i.txt*** the malware gathers general information regarding the victim's computer as well as some information about the malware, like the version number and the path where it's executed, as shown below:

```
v7.00
C:\Users\Public\Videos\audev.txt
==============================================
Log_Drivers:
 C: fixed; size= 102297 Mb, free=83927 Mb S/N: [redacted
 ================================================
 OSV: Windows 7
WinType: 32
WinDir: C:\Windows
Lang: English (United States)
TZ: UTC1:0 Romance Standard Time
HostN: [redacted]-PC
User: [redacted]
===============S_LIST=====================
C:\Program Files\Common Files
C:\Program Files\desktop.ini
C:\Program Files\DVD Maker
C:\Program Files\Internet Explorer
C:\Program Files\Microsoft.NET
C:\Program Files\MSBuild
C:\Program Files\Reference Assemblies
C:\Program Files\Uninstall Information
C:\Program Files\Windows Defender
[…]
```

The malware is capable of taking screenshots, which are stored as **2\[YYYY.mm.dd HH.MM.SS] Image_001.jpg**, and generates another

[YYYY-mm-dd HH-MM-SS]-image_001.jpg, and generates another file ***2\sa.bin,*** populated with the path listing of all screenshots taken. Its last capability is network enumeration; the output is found as ***4.txt***.

Network Protocol

The Delphi Mail Downloader is quite a new addition to Zebrocy's toolset and it features a new way to exfiltrate data and retrieve commands from the operator. The exfiltration is quite simple but very noisy on the network, as all the encrypted files gathered previously are sent over SMTPS and each version of the file is sent three times.

Sender	Recipient
kevin30@ambcomission.com	rishit333@ambcomission.com
salah444@ambcomission.com	rishit333@ambcomission.com
karakos3232@seznam.cz	antony.miloshevich128@seznam.cz

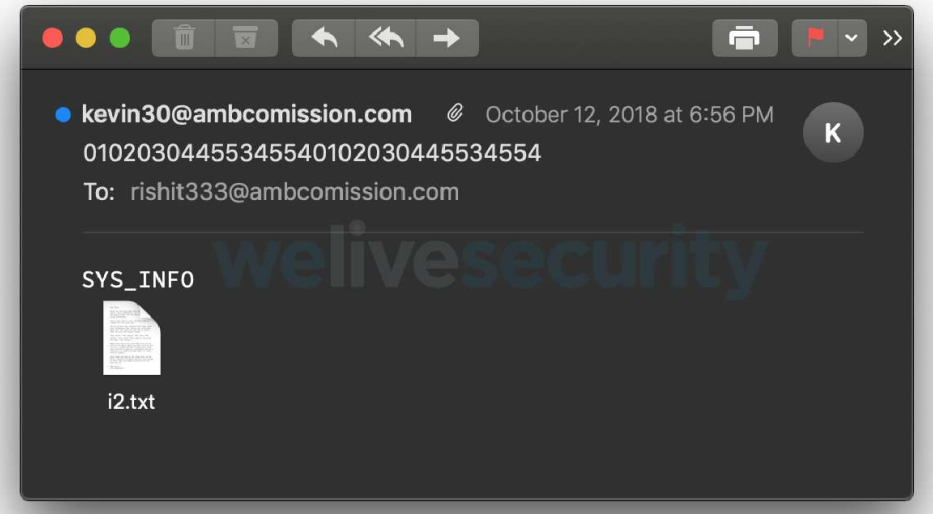
The Subject: of the mail is the ***id*** of the victim and the file is sent as an attachment with a keyword corresponding to the file content. Notice that for each file there is an encrypted version of the file sent.

files	files encrypted	keywords
-	0.txt	SCAN
57.txt	58.txt	ACC
08.txt	082.txt	PrL

i.txt	i2.txt	SYS_INFO
4.txt	42.txt	GET_NETWORK

Screenshots taken and files matching both scans are sent as well but with different keywords.

Content	Keywords
screenshots	SC
.docx, .xlsx, .pdf, .pptx, .rar, .zip, .jpg, .bmp, .tiff	FILEs
.dat, .json, .db	D_ACC



While the exfiltration uses SMTP, the binary connects to the email address ***tomasso25@ambcomission.com*** via POP3 and parses emails. The body of the email contains different keywords that are

interpreted as commands by the malware.

Keywords	Purpose	Log
scan	scan	
ldfile	scan	
edit34	execute and delete	
pKL90	register	isreg
prlist	process listing	
Start23	execute	isr
net40	enumerating network resources	
dele5	delete file	isd
dele6	delete directory	isd
cd25	create directory	isc
autodel	delete itself	
Co55	copy file	is_cp
Mo00	move file	is_m

Once executed, a debug log and the result of the command, if any, are sent back to the operator. For example, for a scan command, the operator receives a file that contains the list of files matching the

operator receives a file that contains the list of files matching the scan extensions along with each matching file.

While this downloader has some backdoor features, it drops a Delphi downloader already associated with the group, and described in our previous Zebrocy article.

Summary

In the past, we identified an overlap between Zebrocy and other traditional Sednit malware. We caught Zebrocy dropping XAgent, the Sednit flagship backdoor. Thus, we attribute Zebrocy to the Sednit group with high confidence.

However, the analysis of these binaries shows some mistakes at the language level as well as development decisions that indicate a different maturity in the development of the toolset. Both downloaders are using mail protocols to exfiltrate information and share common mechanisms to gather the same information. Both are also very noisy on the network and on the system, as they create a lot of files and send a lot of them over the network. While analyzing the Delphi mail downloader, some features seem to have disappeared but some strings still remain the binary. Thus, while this toolset is being operated by the Sednit group, we are very confident that it is being developed by a different and less experienced team, as compared to those who develop the traditional Sednit components.

Zebrocy components are fresh add-ons to the Sednit toolset, and the recent events might explain the increasing use of Zebrocy's binaries rather than the good old Sednit main malware.

Indicators of Compromise (IoCs)

Filename	SHA-1
SCANPASS_QXWEGRFGCVT_323803488900X_jpeg.exe	7768fd2812ceff05db8

C:\Users\public\Pictures\scanPassport.jpg	da70c54a8b9fd2367
C:\Users\Public\Documents\AcrobatReader.{exe,txt}	a225d457c3396e647f
C:\Users\Public\Videos\audev.txt	a659a765536d2099e
%TMP%\Indy0037C632.tmp	20954fe36388ae8b11
%TMP%\Indy01863A21.tmp	e0d8829d2e76e9bb0

List of emails

Emails
carl.dolzhek17@post.cz
shinina.lezh@post.cz
P0tr4h4s7a@post.cz
carl.dolzhek17@post.cz
sym777.g@post.cz
kae.mezhnosh@post.cz
tomasso25@ambcomission.com
kevin30@ambcomission.com

salah444@ambcomission.com

karakos3232@seznam.cz

rishit333@ambcomission.com

antony.miloshevich128@seznam.cz

Related Articles

ESET RESEARCH, THREAT REPORTS

ESET Threat Report H2 2023

ESET RESEARCH

ESET Research Podcast: Neanderthals, Mammoths and Telekopye

ESET RESEARCH

OilRig's persistent attacks using cloud service-powered downloaders

Discussion

What do you think?

0 Responses

 Upvote

 Funny

 Love

 Surprised

 Angry

 Sad

0 Comments Login

Start the discussion...

LOG IN WITH OR SIGN UP WITH DISQUS

Name

welivesecurity BY ESET

Award-winning news, views, and insight from the ESET security community

About us ESET

Contact us Privacy Policy

Legal Information Manage Cookies

RSS Feed