Extracting DDosia targets from process memory

viuleeenz.github.io/posts/2023/05/extracting-ddosia-targets-from-process-memory/

May 8, 2023

6 minutes

Introduction

This post is part of an analysis that I have carried out during my spare time, motivated by a friend that asked me to have a look at the DDosia project related to the NoName057(16) group. The reason behind this request was caused by DDosia client changes for performing the DDos attacks. Because of that, all procedures used so far for monitoring NoName057(16) activities did not work anymore.

Before starting to reverse DDosia Windows sample, I preferred to gather as much information as possible about NoName057(16) TTPs and a few references to their samples.

<u>Avast</u> wrote a very detailed article about that project and described thoroughly all changes observed in the last few months. Because of that, before proceeding with this post, If you feel you are missing something, I strongly recommend that you read their article.

Client Setup

According to the information retrieved from the Telegram channel of DDosia Project, there are a couple of requirements before executing the client. The very first action is to create your id through a dedicated bot that will be used later on for authentication purposes. After that, it's necessary to put the client_id.txt file (generated from DDosia bot) and the executable file in the same folder. If everything has been done properly, it should be possible to observe that authentication process will be done correctly and the client is going to download targets from its server:

📮 uid	5/2/2023 6:44 AN	/ File folder	
client_id.txt	4/26/2023 2:47 A	M Text Docume	ent 1 KB
d_windows_amd64.exe	4/19/2023 5:28 A	M Application	6,046 KB
■ C:\Users\Desktop\DDosia\d_wi Go-Stresser версия 1.0 PID 328 © NoName057(16)	ndows_amd64.exe		
Авторизация пройдена успешно Получено целей: 106	Target received	Authentication successf	ully completed

Dynamic analysis and process memory inspection

Here we are with the fun part. Because of the issues of analyzing GO binaries statically, I preferred to use a dynamic approach supported by Cape sandbox. In fact, executing the client with Cape it was possible to gather behavioral information to speed up our analysis (<u>ref</u>). Since the executable is going to be used for DDoS attacks, it's easy to expect that most of the functions are related to network routines. One of the most interesting WindowsAPI refers to <u>WSAStartup</u>. This is interesting for us, because according to Microsoft documentation, it must be the first function to be used in order to retrieve socket implementation for further network operations:

The WSAStartup function must be the first Windows Sockets function called by an application or DLL. It allows an application or DLL to specify the version of Windows Sockets required and retrieve details of the specific Windows Sockets implementation. The application or DLL can only issue further Windows Sockets functions after successfully calling WSAStartup.

Moreover, starting to monitor network requests with Wireshark, give us additional information about client-server interactions and targets retrieving procedure:

```
1682503566629297883GET /client/get_targets HTTP/1.1
Host: 94.140.114.239
User-Agent: Go-http-client/1.1
Client-Hash: 🔤
                                                                            2:7452
Content-Type: application/json
Time: 1682503566629297898
User-Hash: 🗂
Accept-Encoding: gzip
HTTP/1.1 200 OK
Server: nginx/1.18.0 (Ubuntu)
Date: Wed, 26 Apr 2023 10:06:11 GMT
Content-Type: text/plain; charset=utf-8
Access-Control-Allow-Origin:
Access-Control-Allow-Credentials: true
Access-Control-Expose-Headers: Link
Content-Length: 25631
Connection: keep-alive
Vary: Origin
{"token":
1682503571664790096, "data": "PRshcSQ+d3AxZbOtwyDJaGDcbglN5y559rBY+9gi8CSXRw8Xf5uE3/oly/
wrY0jrxtr0RauLqSJCaLoQQOY/
vcA7jhxdOClihbVeKUxNMjWhz1Y7zFsDwNLMeLRI6aOMIXtNg6mFHbPeScjhFeUUD7j1vbgzX3NiJuzDVBd/
```

Figure 2 - Request for target list

As already mentioned on Avast blogspot, the target list is encrypted and retrieved after the authentication process. However, performing DDoS attacks requires a decryption routine to make targets in cleartext and forward them to a proper procedure. With this insight, it's possible to open up a debugger and set a breakpoint of WSAStartup and start exploring the process flow from that point.

-	326 59.776854 192.168.	.1.9 94.140.114.239 1.2 255.255.255.255	HTTP 383 GET /client/get_targe	ets HTTP/1.1	
	K d_windows_amd64.exe - PID: 6000 - Modul File View Debug Tracing Plugins Fav	ule: ws2_32.dll - Thread: Main Thread: 1292 - x64dbg vourites Options Help Apr 17 2021 (C:\Users	[Elevated] Desktop:\DDosia\d_windows_amd64.exe	×	
>	🗁 🕤 🖬 🌩 🔢 🈤 😽 🐋	points Memory Map 🔂 Call Stac	ег версия 1,6 PID 6000 57(16)	^	278675301}, id e
>	RIP RAX 00007F9AFIEEB10 00007F9AFIEEB15 00007FF9AFIEEB1A 00007F9AFIEEB1A 00007FF9AFIEEB1F	vs2_32.wSAStartup> Авторизаци Получено и	ия пройдена успешно целей: 80	·	
>	 00007FF9AF1EEB20 00007FF9AF1EEB22 00007FF9AF1EEB24 00007FF9AF1EEB26 				
> >	 L000/FF9Ar2IEEB3A 0000/FF9Ar2IEEB1A 0000/FF9Ar2IEEB1A 0000/FF9Ar2IEEB2A 0000/FF9Ar2IEEB2A 0000/FF9Ar2IEEB2A 0000/FF9Ar2IEEB2A 0000/FF9Ar2IEEB2A 0000/FF9Ar2IEEB2A 0000/FF9Ar2IEEB2A 	Авторизаци Получено ц	ия пройдена успешно целей: 80		

Figure 3 - Exploring DDosia executable control flow

Exploring the process execution, it's possible to observe that **WSAStartup API is called two times before starting the attack.** The first one has been used from the main thread to perform the authentication process on the server side, instead the second call will be done right after retrieving the target file and it will be used from another thread to start the attack phase. Since that information we are looking for has been already downloaded and hopefully decrypted (at the time of the second call) we could explore the process memory trying to identify our target list.

🎇 d_windows_amd64.exe - PID: 8116 - Module: ws2_32.dll - Thread: 6248 (switched from 1BD8) - x64dbg [Elevated]											_		×									
File View D	Debug T	racing	Plugins	Favo	urites	Options	Help	Apr 17	2021 (T	itanEngi	ne)											
🖻 🧿 🔳	🔶 II	1 🕈 🕯	≫ ≉	2	-	⇒& [8		8 🖉	2 🥒	fx ;	# 4	A2 📕		9								
CPU	🌛 Log	1 п	otes	• 8	reakpoin	ts 🔳	Mem	ory Map	Í	Call Sta	k	🧠 SEH	Į.	Script	•	Symb	ools	<> Sour	rce 🖌	Reference	s 🔰	
RIP RAX	Image: Wight of the state in the											p. ~				FPU						
		007FFB 007FFB 007FFB 007FFB 007FFB 007FFB 007FFB 007FFB 007FFB	7508E 7508E 7508E 7508E 7508E 7508E 7508E 7508E 7508E 7508E 7508E	B1A B1F B20 B22 B24 B26 B28 B28 B2F B36 B39				66: 57 41: 41: 41: 41: 48: 48: 48: 48:	894C2 54 55 56 57 81EC 8805 33C4 89842	4 08 B0000 1A350 4 A00	0000 0400	00	mov pus pus pus sub mov	/ word h rdi h r12 h r13 h r14 h r15 o rsp,B / rax,q rax,q / qword	0 wo sp p. ~	RA RB RC RD RB RS RS RD		00007FF 0000000 0000007 0000007 000001c 0000007 0000007	87508E 000000 000000 9333FD 514B1D 9333FD 000000 9333FE	B10 001 202 700 F2C 6B8 001 3C0	<ws2_ L'Â' L"loc "GXàr</ws2_ 	32.v ∧ alhc û\x; ∽
aword ptr s	s · [rsn	+18]=[00000	07933	350600)]=1										<						>
rbx=1 .text:00007FFB7508EB10 ws2_32.dll:\$EB10 #DF10 <wsastartup></wsastartup>											1: 2: 3: 4: 5:	rcx rdx r8 0 r9 0 [rsp	0000000 0000000 0000000 0000079 +28] 00	0000000 79333FD 0000000 9333FE3 0000000	202 700 01 00 000000F5		^					
🛄 Dump 1	🛄 Du	mp 2	, , c	0ump 3		Dump 4		Dump 5	G.	Watch	1	[x=] Loca	ls	Struct		(00000	0793331 0793331	D760 D768	00000	00000	000(3FD: ^
Address 00000c0002	50D97 50DA7 50DB7 50DC7 50DD7 50DE7 50E07 50E17 50E27 50E37 50E47	Hex 74 61 65 33 36 33 64 22 35 62 6F 73 62 73 22 31 22 74 65 74 72 74 22 34	72 39 65 3A 33 74 2E 38 79 68 22 74	67 65 38 33 32 22 22 36 63 38 22 3A 67 6F 35 2E 70 65 6F 64 3A 34 72 75	74 51 31 32 2C 22 33 33 34 33 22 70 76 21 31 32 22 35 22 35 34 3 65 20	- 69 6 3 61 3 4 7 36 3 7 36 3 2 62 3 61 2 3 2 62 3 61 2 3 2 6 1 3 2 62 3 3 2 6 1 2 1 3 1 2 4 22 6 4 2 4 4 3 2 2 4 3 2 2 4 3 2 2 7	4 22 9 35 5 71 0 32 0 64 E 65 B 22 E 31 8 74 7 45 2 75 0 61	3A 22 65 33 75 65 37 61 61 73 2C 22 33 30 74 70 54 22 73 65 74 68	36 3 39 3 73 7 33 3 22 2 74 6 69 7 2E 3 22 2 2C 2 5F 7 22 3	4 32 1 34 4 5F 3 61 0 22 3 61 0 22 2 70 3 73 4 22	66 1 32 6 69 6 38 0 68 5 6D 0 3A b 2C '' 6D '' 6F 6 6C r 2E ''	SCII arget 39831 322", ":"63 553c84 5555c84 55555c84 55555c84 55555c84 55555c84 55555c84 55555c84 55555c84 55555c84	_id": 3a95e "requ 7602e 2b0d7 pa.ea .uk", 11.13 :"htt :"GET 3,"us "nat	"642f 39142 est_i 53368 a","h stcam "ip": 0.3", p","m ","po e_ss1 b""'/			00000 00000 00000 00000 00000 00000 0000	0793331 0793331 0793331 0793331 0793331 0793331 0793331 0793331 0793331 0793331	-D770 -D778 -D788 -D790 -D798 -D798 -D798 -D748 -D788 -D788 -D788 -D788 -D700 -D768	00007 00000 00000 00000 00000 00000 00000 0000	7FB72 07900 000000 000000 7FB76 000000 7FFB76 7FB72 01C514 000000 7FFB76	E4CI 0000 0000 29DI 0000 22E1 E4CI B1A 0000 29DI 22E1 29DI 0000 29DI 22E1 29DI 29DI 29DI 29DI 29DI 29DI 29DI 29DI
Command: Commands are comma separated (like assembly instructions): mov eax, ebx Default																						

Figure 4 - Target stored in cleartext within process memory

As we expected, information is actually decrypted right before being used from threads that are in charge to flood the targets. From the cleartext sample, it's also possible to reconstruct the original json file structure that follow this format:

```
{"target_id":"435te3af574b95e395847362","request_id":"23cer8c5mmp4434dlad53f2s","host
":"www.tartuhly.ee","ip":"90.190.99.85","type":"http","method":"GET","port":443,"use_
ssl":true,"path":"/otsi/$_1","body":{"type":"","value":""},"headers":null}
```

At this point I have shown all procedures to quickly follow the execution flow until the decryption routine is called. From now on, it's just a matter of looking for those data within process memory and extracting them for your own purpose. It's worth noting that information won't be stored decrypted forever, in fact, as the executable keeps running, the json file is actually mangled in a way that is not easy to resemble it properly.

A little bit of automation

Even if the analysis has been completed and targets are correctly retrieved, I thought that giving a little tool to extract that information would be useful. Instead of doing complex stuff, I wrote two simple scripts called targets.js and recover.py. The purpose of these two files is to allow analysts from different backgrounds to extract those targets, even performing a simple memory dump. Probably there are easier and smarter techniques out there, but it was also a good chance to put in practice DBI, which I have already covered in a previous <u>post</u>.

- <u>target.js</u>: Frida script that aims to get a memory dump after the WSAStartup has been called for the second time (when payloads are in cleartext in memory).
- <u>recover.py</u>: it's a simple python script that retrieves structured information from the files dumped. It's worth noting that I limited my script to look for structured information, retrieving IP and Hostname (additional improvements are left to user's needs).

Script Testing

In order to run the mentioned scripts there are two requirements to fulfill:

- Installing frida-tool (pip install frida-tools).
- Create a folder named "dumps" in the same place where you run the target.js file.

If all requirements are satisfied it's just a matter of running those scripts and getting the results. The first step is to run frida.exe, using the targets.js file that contains all the information to dump the process memory:

frida.exe <ddosia_client.exe> -l targets.js

If everything has been done correctly (please keep in mind the requirements), you should be able to see a message "[END] Memory dumped correctly" in your console.



Figure 5 - Dumping process Memory with Frida

Now you can navigate in dumps folder and run the python script using the following command line that is going to forward all dumped file from the current directory to the script that is going to print the result in your console:

python.exe recover.py (Get-Item .*dump)

PS C:\Users_______sktop\dumps > python .\recover.py (Get-Item .*dump)
[+] Structured data discovered in C:\Users\Alessandro\Desktop\dumps\0xc00000000_dump file
{'ulc.gov.pl : 91.228.11.93', 'www.pekao.com.pl : 193.111.166.166', 'sb.lt : 185.189.155.16', 'www.skm.pkp.pl : 213.192.
75.70', 'partneriusavitarna.sb.lt : 185.189.155.35', 'pis.org.pl : 89.161.255.58', 'zamowienia.metro.waw.pl : 195.205.14
8.130', 'metro.waw.pl : 54.38.54.236', 'fx.sydbank.dk : 131.164.253.236', 'www.vilnius-airport.lt : 88.119.246.80', 'dan
skebank.com : 212.93.59.102', 'www.siauliai-airport.com : 194.135.87.142', 'www.sydbank.dk : 176.21.158.52', 'fm.dk : 18
8.64.157.250', 'sbip.sb.lt : 185.189.155.16'}

Figure 6 - Extracting DDosia targets from dump files

Final Notes

Before concluding, It's worth mentioning that updates on these scripts and new techniques to dealing with further improvements of DDosia project are not going to be shown, because it represents a topic that I'm not following personally and I'm sure that more authoritative voices will keep track of this threat and its evolution.

References:

Binary analyzed: <u>d_windows_amd64.exe</u> | 726c2c2b35cb1adbe59039193030f23e552a28226ecf0b175ec5eba9dbcd336e (sha256) | 19/04/2023

1123 Words

2023-05-08 02:00