“PortDoor” Malware

By Ilan Duhin

Published: 2023-07-20 · Archived: 2026-04-06 00:45:24 UTC

7 min read

Apr 5, 2023

Researched by: Ilan Duhin

Executive Summary:

“PortDoor” is a Chinese Backdoor that targeted ministry and public organizations such as ministry agencies, and industrial plants in East Europe countries (Russia, Belarus and Ukraine). It spreads via spear-phishing emails that contain malicious RTF file as an attachment. once executed, opened a Microsoft add-in file (.wll file) that is actually a DLL file.

The dropped DLL has the capabilities of gathering reconnaissance and profiling of the victim’s machine, communicating with the C2 server, and privilege escalation.

Behind “PortDoor” is the group dubbed “TA428” (which belongs to the Chinese government) according to the TTPs we analyze on the DLL backdoor.

The DLL gains accessibility to the network via the Microsoft vulnerability called: Equation Editor (CVE-2017–11882) which enables to execution of arbitrary code without any additional user activity.

To communicate, the malware establishes an HTTP connection on the victim’s endpoint, it uses the connection to exfiltrate the victim’s information such as username, and network configuration.

This report contains in-depth research of the Backdoor and the RTF attachment, including an analysis of its behavior.

Banker Attack Analysis:

The Backdoor’s attack steps were as follows:

1. “PortDoor” encrypts its own configuration with XOR key. The XOR loop appears as part of the file run.

2. After launching, the malware collects general information about the infected system via API calls, such as computer names, IP addresses, and sends the information collected to the C2 server.

3. After decrypting the configuration information, “PortDoor” check it is not running in a debugger (it uses the IsDebuggerPresent call).

4. “PortDoor” establishes a connection with the C2 server via HTTP protocol. — 45.63.27[.]162

5. The data that is received/sent is encrypted, using the AES key.

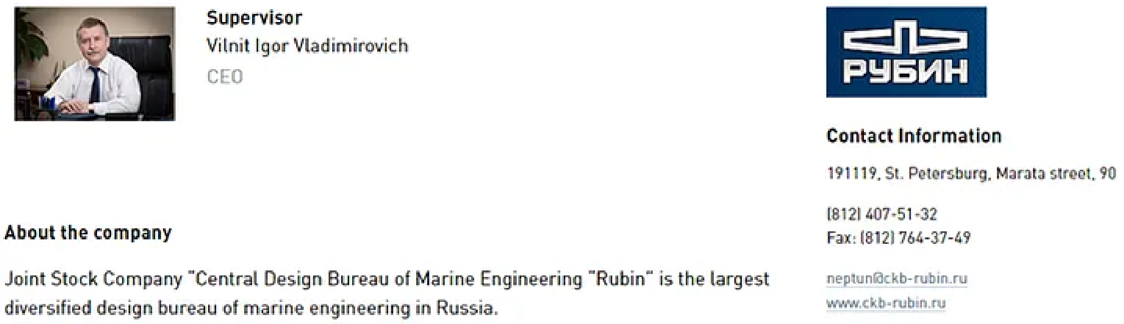
Technical Analysis:

Headers of spear phishing: we can see who sends and which target mail.

· The sender was “Gidropribor[1]” — the research center of the Russian Federation.

· The receiver was Igor Vladimirovich[2], CEO of Marine Engineering “Rubin”.

Press enter or click to view image in full size



Press enter or click to view image in full size

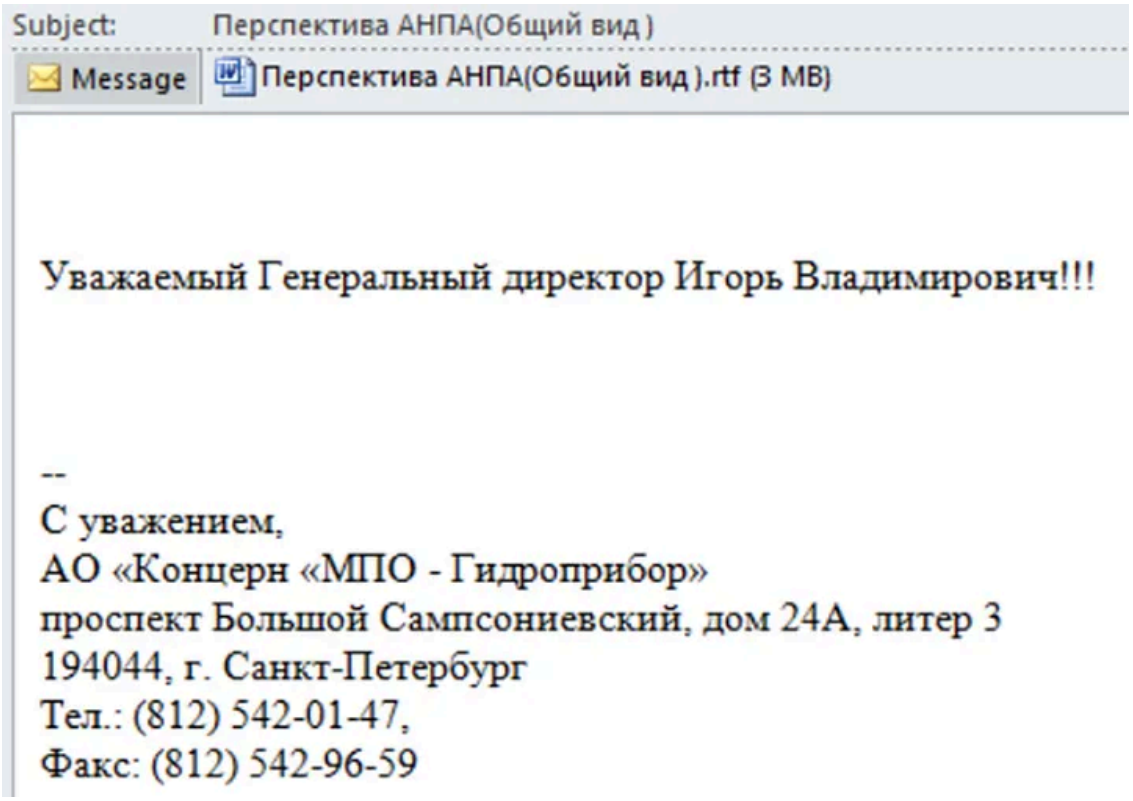
```
Return-Path: <mpo.gidropribor@mail.ru>
Delivered-To: neptun@ckb-rubin.ru
Received: from localhost (localhost [127.0.0.1])
    by relay.ckb-rubin.ru (Postfix) with ESMTP id 8BD2E14057A;
    Thu,  1 Apr 2021 10:38:27 +0300 (MSK)
X-Virus-Scanned: amavisd-new at ckb-rubin.ru
X-Spam-Flag: NO
X-Spam-Score: -1.99
X-Spam-Level:
X-Spam-Status: No, score=-1.99 tagged_above=-999 required=6.2
    tests=[BAYES_00=-1.9, DKIM_SIGNED=0.1, DKIM_VALID=-0.1,
    DKIM_VALID_AU=-0.1, FREEMAIL_FROM=0.001, HTML_MESSAGE=0.001,
    RCVD_IN_DNSWL_NONE=-0.0001, SPF_HELO_PASS=-0.001, SPF_PASS=-0.001,
    T_FREEMAIL_DOC_PDF=0.01] autolearn=ham
Authentication-Results: relay.ckb-rubin.ru (amavisd-new);
    dkim=pass (1024-bit key) header.d=mail.ru header.b=dcFYwsec;
    dkim=pass (1024-bit key) header.d=mail.ru header.b=oo8Ma5t4
Received: from relay.ckb-rubin.ru ([127.0.0.1])
    by localhost (relay.ckb-rubin.ru [127.0.0.1]) (amavisd-new, port 10024)
    with LMTP id D0Y6hVXegUNL; Thu,  1 Apr 2021 10:38:25 +0300 (MSK)
Received: from fallback19.mail.ru (fallback19.mail.ru [185.5.136.251])
    by relay.ckb-rubin.ru (Postfix) with ESMTPS id 8B202140454
    for <neptun@ckb-rubin.ru>; Thu,  1 Apr 2021 10:38:19 +0300 (MSK)
DKIM-Signature: v=1; a=rsa-sha256; q=dns/txt; c=relaxed/relaxed; d=mail.ru; s=mail3;
    h=Content-Type:Message-ID:Reply-To:Date:MIME-Version:Subject:To:From; bh=Lp4BsMFQADYx+57FLp4Hq7LB8sdaKBvXy3880AIH7XI=;
    b=dcFYwsec1e/L44c31omAle3wIhOZEK0HNQ1LFZ3hDHOiZOFRtH7y1OfJOmEQQ3WnJnpDXrCa+J7QFqGRz4enol1HGLa8etbPrmD58YMsOWACUFL/vTeYIx2dirhb8nd7K6U2agfivhnJI9B+zL/4D76znxZWe5d0OrbqnX+UOMI=;
Received: from [10.161.1.10] (port=48350 helo=f330.i.mail.ru)
    by fallback19.m.smailru.net with esmtp (envelope-from <mpo.gidropribor@mail.ru>)
    id 1lRrqa-0003oK-Sc
    for neptun@ckb-rubin.ru; Thu, 01 Apr 2021 10:34:34 +0300
DKIM-Signature: v=1; a=rsa-sha256; q=dns/txt; c=relaxed/relaxed; d=mail.ru; s=mail3;
    h=Content-Type:Message-ID:Reply-To:Date:MIME-Version:Subject:To:From:From:Subject:Content-Type:Content-Transfer-Encoding:To:Cc; bh=Lp4BsMFQADYx+57FLp4Hq7LB8sdaKBvXy3880AIH7XI=;
    b=oo8Ma5t4di6tVf+N+HNG9AU2CQ4s7FKT3OL1MiwL/i87nIR5Y60DAAAnRvGgTPi0nnICfyk1OzLUblyEntBIGQRtjNDyEW2H5gNZCCndEgZ1KmjHF4TtIkx3DFHJW6eZoA3E04y7Ft9Xh85zE73OqlHpgZwE/AQNN985nC9sZiQ=;
Received: by f330.i.mail.ru with local (envelope-from <mpo.gidropribor@mail.ru>)
    id 1lRrqY-00011I-Qj
    for neptun@ckb-rubin.ru; Thu, 01 Apr 2021 10:34:32 +0300
Received: by e.mail.ru with HTTP;
    Thu, 01 Apr 2021 10:34:30 +0300
From: =?UTF-8?B?0JzQn9CeINCT0LjQtNGA0L7Qv9GA0LjQsdC+0YA=?= <mpo.gidropribor@mail.ru>
To: neptun@ckb-rubin.ru
Subject: =?UTF-8?B?0J/QtdGA0YHQv9C10LzRgtC40LLQsCDQkNCd0J/QkCjQntCm0YnQuNC5INCy?=
 =?UTF-8?B?0LjQtCAp?=
MIME-Version: 1.0
X-Mailer: Mail.Ru Mailer 1.0
Date: Thu, 01 Apr 2021 10:34:30 +0300
Reply-To: =?UTF-8?B?0JzQn9CeINCT0LjQtNGA0L7Qv9GA0LjQsdC+0YA=?= <mpo.gidropribor@mail.ru>
X-Priority: 3 (Normal)
Message-ID: <1617262470.434654086@f330.i.mail.ru>
```

Mail Headers

[1] https://gidropribor.ru/en/

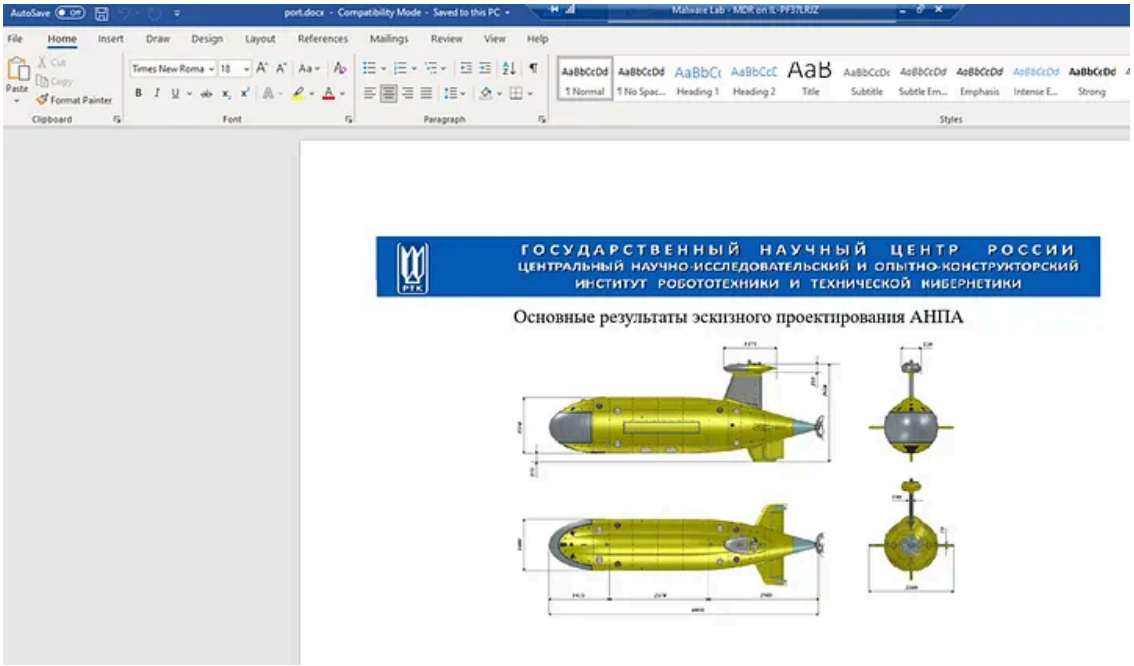
[2] https://www.aoosk.ru/companies/oao-tsentralnoe-konstruktorskoe-byuro-morskoy-tekhniki-rubin/

Press enter or click to view image in full size



When opening the rtf file.

Press enter or click to view image in full size



From looking at HxD and searching the header we can see that this is an RTF File.

The fact that this is an RTF file rather than a typical Word Doc should raise some red flags regarding the attachment. RTF (Rich Text Format) allows other files to be embedded in the file itself and are often used by attackers to embed malware.

port.docx.doc

Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Decoded text
00000000	7B	5C	72	74	66	31	5C	61	64	65	66	6C	61	6E	67	31	{\rtf1\adeflangl
00000010	30	32	35	5C	61	6E	73	69	5C	61	6E	73	69	63	70	67	025\ansi\ansicpg
00000020	39	33	36	5C	75	63	32	5C	61	64	65	66	66	30	5C	64	936\uc2\adeff0\d
00000030	65	66	66	30	5C	73	74	73	68	66	64	62	63	68	33	31	eff0\stshfdbch31
00000040	35	30	35	5C	73	74	73	68	66	6C	6F	63	68	33	31	35	505\stshfloch315
00000050	30	36	5C	73	74	73	68	66	68	69	63	68	33	31	35	30	06\stshfhich3150
00000060	36	5C	73	74	73	68	66	62	69	30	5C	64	65	66	6C	61	6\stshfbi0\defla
00000070	6E	67	31	30	33	33	5C	64	65	66	6C	61	6E	67	66	65	ng1033\deflangfe
00000080	32	30	35	32	5C	74	68	65	6D	65	6C	61	6E	67	31	30	2052\themelang10
00000090	33	33	5C	74	68	65	6D	65	6C	61	6E	67	66	65	32	30	33\themelangfe20
000000A0	35	32	5C	74	68	65	6D	65	6C	61	6E	67	63	73	30	7B	52\themelangcs0{
000000B0	5C	66	6F	6E	74	74	62	6C	7B	5C	66	30	5C	66	62	69	\fonttbl{\f0\fbi
000000C0	64	69	20	5C	66	72	6F	6D	61	6E	5C	66	63	68	61	72	di \froman\fchar
000000D0	73	65	74	30	5C	66	70	72	71	32	7B	5C	2A	5C	70	61	set0\fprq2{*\pa
000000E0	6E	6F	73	65	20	30	32	30	32	30	36	30	33	30	35	30	nose 02020603050
000000F0	34	30	35	30	32	30	33	30	34	7D	54	69	6D	65	73	20	405020304}Times
00000100	4E	65	77	20	52	6F	6D	61	6E	3B	7D	0D	0A	7B	5C	66	New Roman;}..{\f
00000110	31	33	5C	66	62	69	64	69	20	5C	66	6E	69	6C	5C	66	13\fbidi \fnil\f
00000120	63	68	61	72	73	65	74	31	33	34	5C	66	70	72	71	32	charset134\fprq2
00000130	7B	5C	2A	5C	70	61	6E	6F	73	65	20	30	32	30	31	30	{*\panose 02010
00000140	36	30	30	30	33	30	31	30	31	30	31	30	31	30	31	7D	600030101010101}
00000150	5C	27	63	62	5C	27	63	65	5C	27	63	63	5C	27	65	35	\'cb\'ce\'cc\'e5
00000160	7B	5C	2A	5C	66	61	6C	74	20	53	69	6D	53	75	6E	7D	{*\falt SimSun}
00000170	3B	7D	7B	5C	66	31	33	5C	66	62	69	64	69	20	5C	66	;}{\f13\fbidi \f
00000180	6E	69	6C	5C	66	63	68	61	72	73	65	74	31	33	34	5C	nil\fcharset134\
00000190	66	70	72	71	32	7B	5C	2A	5C	70	61	6E	6F	73	65	20	fprq2{*\panose
000001A0	30	32	30	31	30	36	30	30	30	33	30	31	30	31	30	31	02010600030101
000001B0	30	31	30	31	7D	5C	27	63	62	5C	27	63	65	5C	27	63	0101}\'cb\'ce\'c
000001C0	63	5C	27	65	35	7B	5C	2A	5C	66	61	6C	74	20	53	69	c\'e5{*\falt Si
000001D0	6D	53	75	6E	7D	3B	7D	0D	0A	7B	5C	66	33	37	5C	66	mSun};}..{\f37\f
000001E0	62	69	64	69	20	5C	66	73	77	69	73	73	5C	66	63	68	bidi \fswiss\fch
000001F0	61	72	73	65	74	30	5C	66	70	72	71	32	7B	5C	2A	5C	arset0\fprq2{*\
00000200	70	61	6E	6F	73	65	20	30	32	30	66	30	35	30	32	30	panose 020f05020

Searching the hash in VT and see that the creation time of the RTF file was in 2007.

Too long time will indicate a technique that adversaries used to try goes under the radar of the security products (especially when we see the long difference between the creation time and the last analysis section).

History

Creation Time	2007-05-25
First Submission	2021-04-09
Last Submission	2021-05-03
Last Analysis	2022-12-27

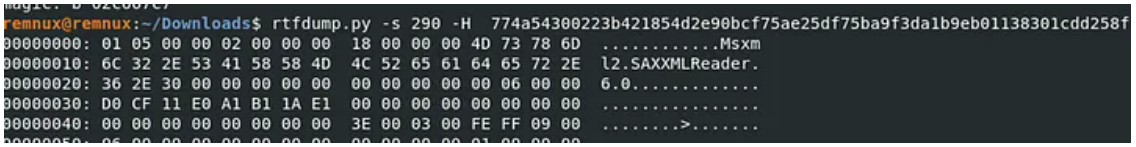
OLE Tools:

After extracting the file objects with rtfdump (The tool extracts file objects)

I started analyzing them deeply.

An object that seems suspicious is 290.

Press enter or click to view image in full size



When I input the argument “i” to gather more information it indicates that we have a reference to the **embedded file** within this object. The magic number for this object is called: “**d0cfile0**”.

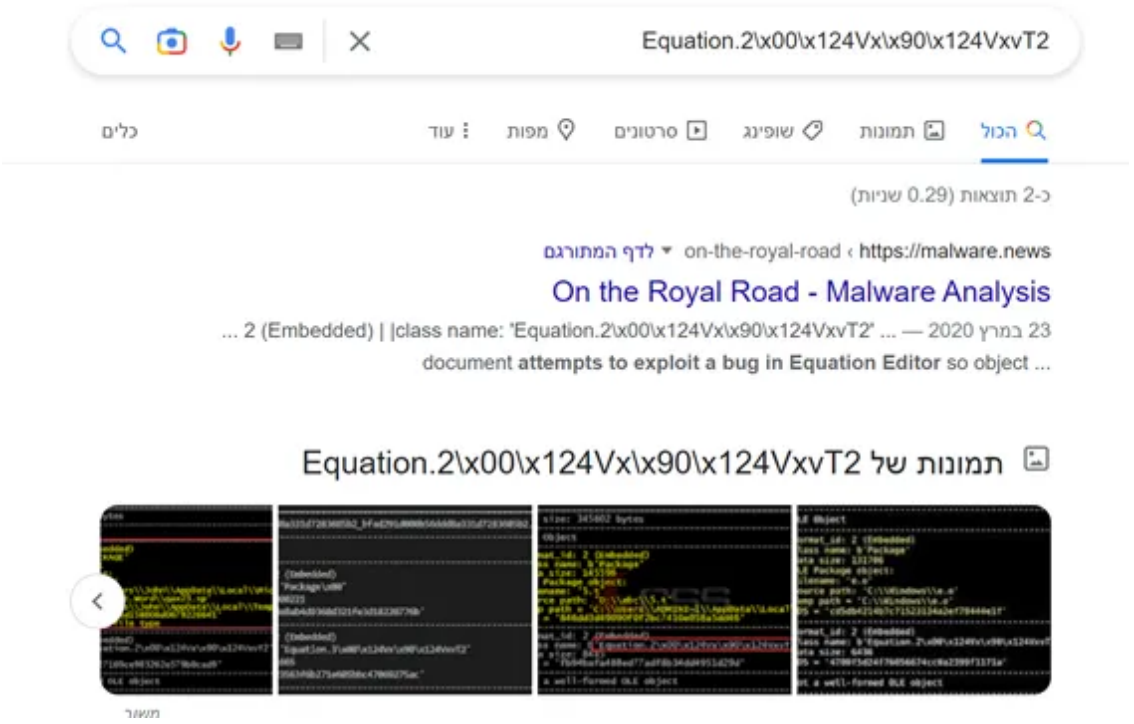
Press enter or click to view image in full size

The object that connect us to Equation Editor class name.

Press enter or click to view image in full size

From a quick look, we can see that the file contains the Equation class name. it is evidence that an Equation Editor vulnerability will be exploit.

- When I searched the Equation pattern in Google, I found that it connects to **Royal Road** malware or its second name, weaponizer, that use the same technique in previous attacks of several Chinese threat actors such TA428, the same APT that spread the “PortDoor”.

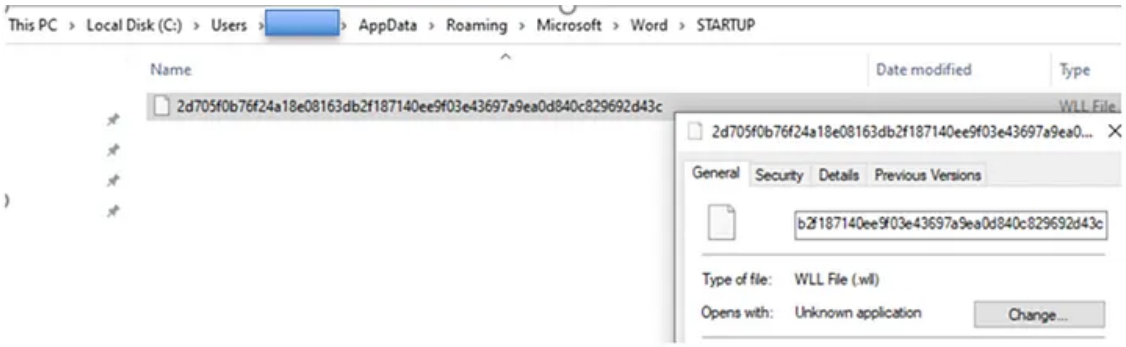


Once the RTF document is executed, a .wll (word add-in file) is dropped to Microsoft startup folder.

This is largely done as an evasion technique as when a Word document is launched, an add-in file is loaded along with it and security products recognize it as legitimate behavior.

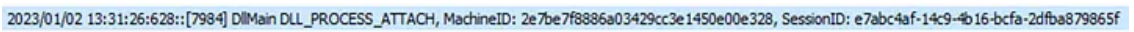
Press enter or click to view image in full size

Press enter or click to view image in full size



after running the .doc, I checked the memory strings and see that the file below attached a DLL file. In other words, our sample doing a backdoor to the DLL.

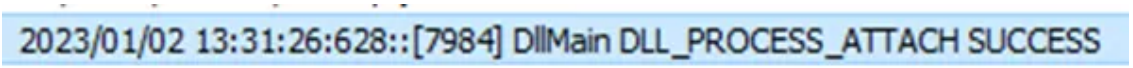
Press enter or click to view image in full size



Attach success:

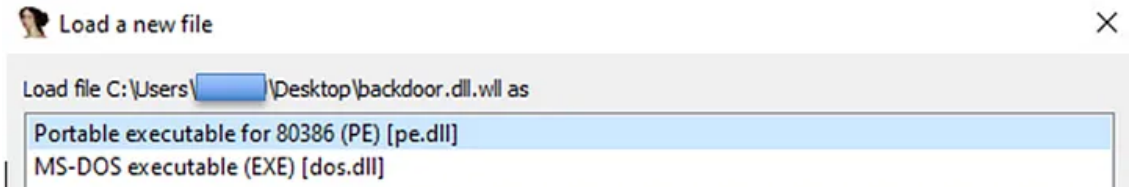
Get Ilan Duhin's stories in your inbox

In other words, the file probably was exist when we run the RTF file earlier (our backdoor). The strings just verify the information.

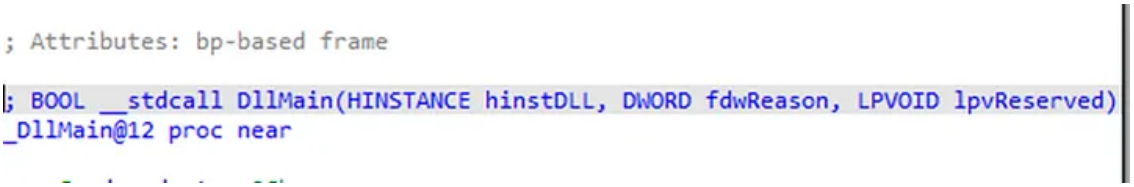


When dropping the .wll file into IDA, it describe it as a DLL file.

Press enter or click to view image in full size



Press enter or click to view image in full size



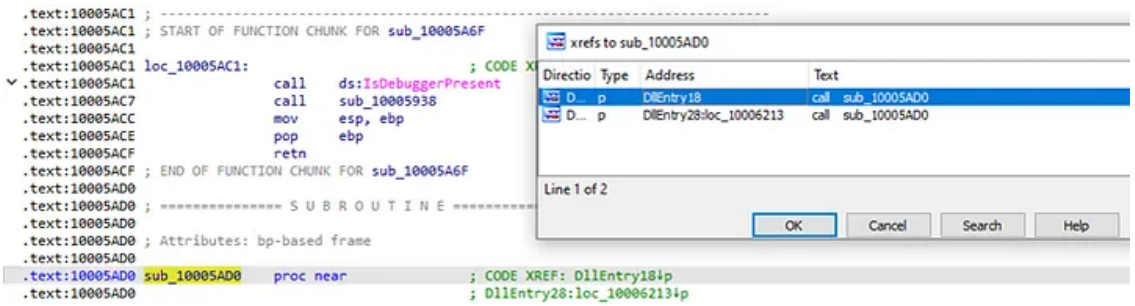
The DLL has 33 export functions.

functions 18 and 28 are the ones that "PortDoor" used to make his activities. The rest of them enters to **sleep loop** as part of the anti-analysis technique.

```
push    3E8h            ; DllEntry00
                        ; DllEntry01
                        ; DllEntry02
                        ; DllEntry03
                        ; DllEntry04
                        ; DllEntry05
                        ; DllEntry06
                        ; DllEntry07
                        ; DllEntry08
                        ; DllEntry09
                        ; DllEntry10
                        ; DllEntry11
                        ; DllEntry12
                        ; DllEntry13
                        ; DllEntry14
                        ; DllEntry15
                        ; DllEntry16
                        ; DllEntry17
                        ; DllEntry19
                        ; DllEntry20
                        ; DllEntry21
                        ; DllEntry22
                        ; DllEntry23
                        ; DllEntry24
                        ; DllEntry25
                        ; DllEntry26
                        ; DllEntry27
                        ; DllEntry29
                        ; DllEntry30
                        ; DllEntry31
                        ; DllEntry32
call    ds:Sleep
jmp     short DllEntry33
DllEntry33 endp
```

when investigating the two missing functions, they contains xor loop if one of them will called.

Press enter or click to view image in full size

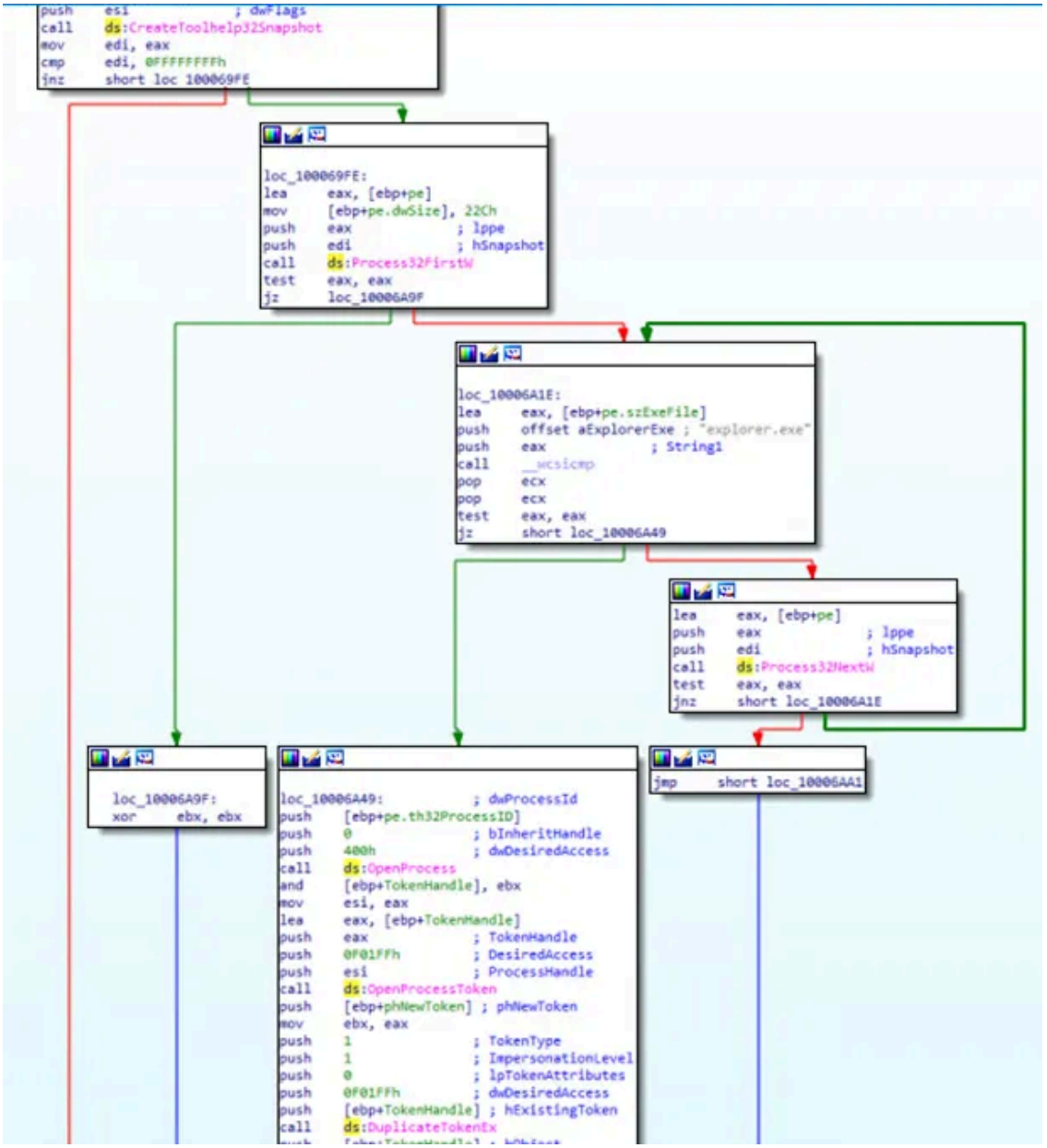


Trying to achieve **privilege escalation** by applying the **Access Token Theft Technique**[1] to steal **explorer.exe** tokens and run under a privileged security context.

[1] https://attack.mitre.org/techniques/T1134/

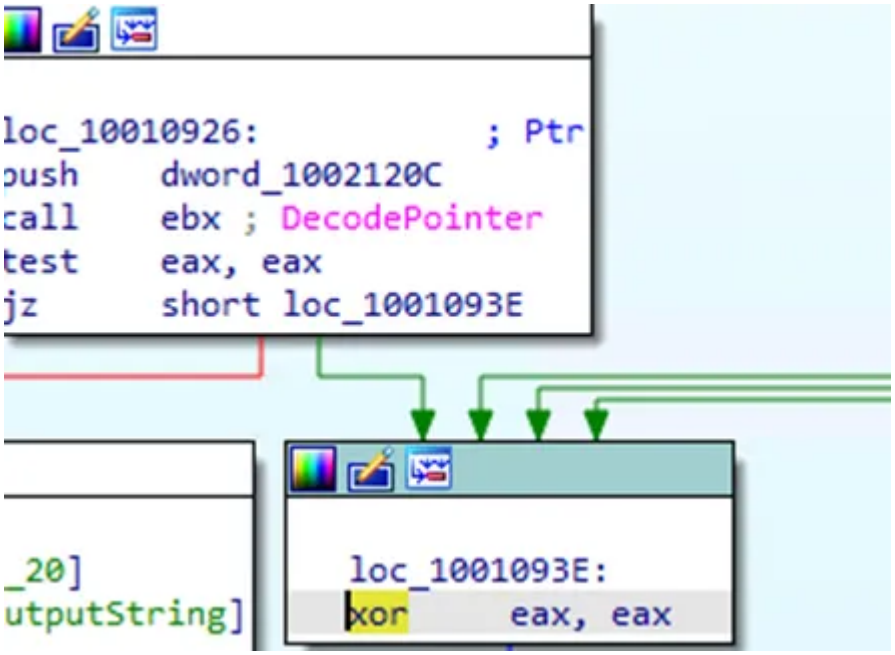
```
pe= PROCESSENTRY32W ptr -244h
TokenHandle= dword ptr -18h
NewState= _TOKEN_PRIVILEGES ptr -14h
var_4= dword ptr -4
arg_0= dword ptr  8

push    ebp
mov     ebp, esp
sub     esp, 248h
mov     eax, ___security_cookie
xor     eax, ebp
mov     [ebp+var_4], eax
mov     eax, [ebp+arg_0]
push    ebx
push    esi
push    edi
mov     [ebp+phNewToken], eax
lea     eax, [ebp+TokenHandle]
push    eax             ; TokenHandle
push    28h ; '('       ; DesiredAccess
call    ds:GetCurrentProcess
push    eax             ; ProcessHandle
call    ds:OpenProcessToken
push    2
pop     esi
lea     eax, [ebp+NewState.Privileges]
mov     [ebp+NewState.PrivilegeCount], 1
push    eax             ; lpLuid
push    offset Name     ; "SeShutdownPrivilege"
xor     edi, edi
mov     [ebp+NewState.Privileges.Attributes], esi
push    edi             ; lpSystemName
call    ds:LookupPrivilegeValueW
push    edi             ; ReturnLength
push    edi             ; PreviousState
push    10h             ; BufferLength
lea     eax, [ebp+NewState]
push    eax             ; NewState
push    edi             ; DisableAllPrivileges
push    [ebp+TokenHandle] ; TokenHandle
call    ds:AdjustTokenPrivileges
call    ds:GetLastError
push    [ebp+TokenHandle] ; hObject
call    ds:CloseHandle
push    228h            ; Size
lea     eax, [ebp+pe.cntUsage]
mov     [ebp+pe.dwSize], edi
push    edi             ; Val
push    eax             ; void *
mov     ebx, edi
call    _memset
```



Another suspicious API call is **IsDebuggerPresent.** here, the malware is used to realize if it is located under debug mode. **if not** it **decodes** the pointer that points to the victim's process that is encrypted with the XOR key.

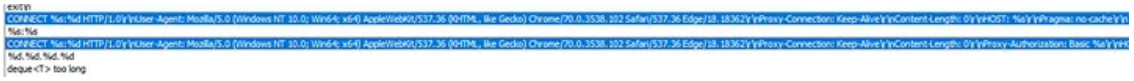

push offset aGetprocesswind ; "GetProcessWindowStation"
push edi ; hModule
call ds:GetProcAddress
push eax ; Ptr
call ebx ; EncodePointer
mov dword_10021218, eax
loc_1001085B:
mov edi, [ebp+var_18]
loc_1001085E:
call ds:IsDebuggerPresent
test eax, eax
jz short loc_10010883
+lpOutputString]
_10010876
; lpOutputString
gStringW



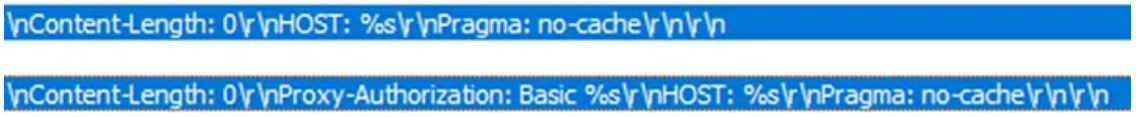
In addition, I find two suspicious strings that describe HTTP connection **via proxy server** to a specific host.

the backdoor appears to be distinguishing between two HTTP response types ‘200’ response and ‘407’.

Press enter or click to view image in full size



Press enter or click to view image in full size



The full code from IDA:

Press enter or click to view image in full size

```
.rdata:1001D660 ; const char aConnectSDHttp1[]
.rdata:1001D660 aConnectSDHttp1 db 'CONNECT %s:%d HTTP/1.0',0Dh,0Ah
.rdata:1001D660                                         ; DATA XREF: sub_1000644C+4F↑o
.rdata:1001D660                 db 'User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit'
.rdata:1001D660                 db '/537.36 (KHTML, like Gecko) Chrome/70.0.3538.102 Safari/537.36 Ed'
.rdata:1001D660                 db 'ge/18.18362',0Dh,0Ah
.rdata:1001D660                 db 'Proxy-Connection: Keep-Alive',0Dh,0Ah
.rdata:1001D660                 db 'Content-Length: 0',0Dh,0Ah
.rdata:1001D660                 db 'HOST: %s',0Dh,0Ah
.rdata:1001D660                 db 'Pragma: no-cache',0Dh,0Ah
.rdata:1001D660                 db 0Dh,0Ah,0
.rdata:1001D757                 align 4
.rdata:1001D758 word_1001D758   dw 0A0Dh                ; DATA XREF: sub_1000644C+AE↑r
.rdata:1001D758                                         ; sub_10006562+118↑r
.rdata:1001D75A byte_1001D75A   db 0                    ; DATA XREF: sub_1000644C+BB↑r
.rdata:1001D75A                                         ; sub_10006562+125↑r
.rdata:1001D75B                 align 4
.rdata:1001D75C ; const char SubStr[4]
.rdata:1001D75C SubStr          db '200',0              ; DATA XREF: sub_1000644C+E1↑o
.rdata:1001D75C                                         ; sub_10006562+14B↑o
.rdata:1001D760 ; const char a407[4]
.rdata:1001D760 a407            db '407',0              ; DATA XREF: sub_1000644C:loc_10006542↑o
.rdata:1001D760                                         ; sub_10006562:loc_100066C2↑o
.rdata:1001D764 ; const char Format[]
.rdata:1001D764 Format          db '%s:%s',0            ; DATA XREF: sub_10006562+40↑o
.rdata:1001D76A                 align 10h
.rdata:1001D770 ; const char aConnectSDHttp1_0[]
.rdata:1001D770 aConnectSDHttp1_0 db 'CONNECT %s:%d HTTP/1.0',0Dh,0Ah
.rdata:1001D770                                         ; DATA XREF: sub_10006562+A2↑o
.rdata:1001D770                 db 'User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit'
.rdata:1001D770                 db '/537.36 (KHTML, like Gecko) Chrome/70.0.3538.102 Safari/537.36 Ed'
.rdata:1001D770                 db 'ge/18.18362',0Dh,0Ah
.rdata:1001D770                 db 'Proxy-Connection: Keep-Alive',0Dh,0Ah
.rdata:1001D770                 db 'Content-Length: 0',0Dh,0Ah
.rdata:1001D770                 db 'Proxy-Authorization: Basic %s',0Dh,0Ah
.rdata:1001D770                 db 'HOST: %s',0Dh,0Ah
.rdata:1001D770                 db 'Pragma: no-cache',0Dh,0Ah
.rdata:1001D770                 db 0Dh,0Ah,0
```

One of the suspicious IP addresses that I saw is: 45.63.27[.]162[1][2]

This IP address is the **only one** that shows up in the source and in the destination.

From searching the IP address in attack reports, it seems that it uses “PortDoor” C2.

[1] https://kcm.trellix.com/corporate/index?page=content&id=KB94757&locale=en_US

[2] https://vuldb.com/?actor.portdoor

Press enter or click to view image in full size

Source	Destination	Protocol	Length	Info
192.168.5.111	45.63.27.162	TCP	66	58337 → 443 [SYN] Seq=0 Win=64240 Len=
45.63.27.162	192.168.5.111	TCP	66	443 → 58337 [SYN, ACK] Seq=0 Ack=1 Wir
192.168.5.111	45.63.27.162	TCP	54	58337 → 443 [ACK] Seq=1 Ack=1 Win=2631
192.168.5.111	45.63.27.162	TCP	58	58337 → 443 [PSH, ACK] Seq=1 Ack=1 Wir
45.63.27.162	192.168.5.111	TCP	66	[TCP Retransmission] 443 → 58337 [SYN,
192.168.5.111	45.63.27.162	TCP	66	[TCP Dup ACK 43#1] 58337 → 443 [ACK] S
45.63.27.162	192.168.5.111	TCP	66	[TCP Retransmission] 443 → 58337 [SYN,
192.168.5.111	45.63.27.162	TCP	66	[TCP Dup ACK 43#2] 58337 → 443 [ACK] S
192.168.5.111	45.63.27.162	TCP	58	[TCP Retransmission] 58337 → 443 [PSH,
45.63.27.162	192.168.5.111	TCP	66	[TCP Retransmission] 443 → 58337 [SYN,
192.168.5.111	45.63.27.162	TCP	66	[TCP Dup ACK 43#3] 58337 → 443 [ACK] S
45.63.27.162	192.168.5.111	TCP	66	[TCP Retransmission] 443 → 58337 [SYN,
192.168.5.111	45.63.27.162	TCP	66	[TCP Dup ACK 43#4] 58337 → 443 [ACK] S
192.168.5.111	45.63.27.162	TCP	58	[TCP Retransmission] 58337 → 443 [PSH,
45.63.27.162	192.168.5.111	TCP	66	[TCP Retransmission] 443 → 58337 [SYN,
192.168.5.111	45.63.27.162	TCP	66	[TCP Dup ACK 43#5] 58337 → 443 [ACK] S
45.63.27.162	192.168.5.111	TCP	60	443 → 58337 [ACK] Seq=1 Ack=5 Win=6425
45.63.27.162	192.168.5.111	HTTP	349	HTTP/1.1 400 Bad Request (text/html)
45.63.27.162	192.168.5.111	TCP	60	443 → 58337 [FIN, ACK] Seq=296 Ack=5 W
192.168.5.111	45.63.27.162	TCP	54	58337 → 443 [ACK] Seq=5 Ack=297 Win=26
45.63.27.162	192.168.5.111	TCP	66	[TCP Dup ACK 178#1] 443 → 58337 [ACK]
45.63.27.162	192.168.5.111	TCP	66	[TCP Dup ACK 178#2] 443 → 58337 [ACK]

After the malware establish a connection with the C2 server, the data that received/sent is encrypted using AES key.

Press enter or click to view image in full size

```
.rdata:100181E4                 align 10h
.rdata:100181F0 aAesPartOfOpens db 'AES part of OpenSSL 1.0.2l  2
```

AES encrypted information.

Press enter or click to view image in full size

```
7784F030 75 28 BC 4C 2A 51 7D 49 A8 9C 1F B2 8C 68 D0 AA u(¼L*Q}I¨..².hÐª
7784F040 E0 DB FF DA 3B 5C 27 5C 53 81 48 3B B1 27 E7 73 àÛÿÚ;\'\S.H;±'çs
7784F050 1B C6 A8 53 06 13 10 85 35 34 26 F8 A4 D0 67 2C .Æ S....54&øÐg,
7784F060 7D 73 C6 90 13 E4 90 DA CE 29 71 71 99 9F 50 F5 }sÆ..ä.ÚÎ)qq..Põ
7784F070 86 6E 91 19 2E AB A7 03 F9 5E FB 7E 8F 3E 66 93 .n...«§.ù^û~.>f.
7784F080 B1 19 1B 16 38 0A 91 65 02 43 AC F7 B2 71 51 4A ±...8..e.C¬÷²qQJ
7784F090 4A 04 4C 9F 05 45 A6 BC 64 F6 C2 34 A7 86 D1 15 J.L..E¦¼döÂ4§.Ñ.
7784F0A0 2C B1 22 5C 10 B2 26 E3 9F EB 95 BD 9A C9 E6 CC ,±"\.²&ã.ë.½.ÉæÌ
7784F0B0 D7 AC 75 D5 2D FD 11 3A 00 00 00 00 00 00 00 00 ×¬uÕ-ý.:........
7784F0C0 5D 33 12 A5 3D C1 B0 71 BA 66 24 4A 7B 82 61 E3 ]3.¥=Á°qºf$J{.aã
7784F0D0 E7 55 36 EF 46 43 D1 92 1F 5E DF CC A5 22 1A F2 çU6ïFCÑ..^ßÌ¥".ò
```

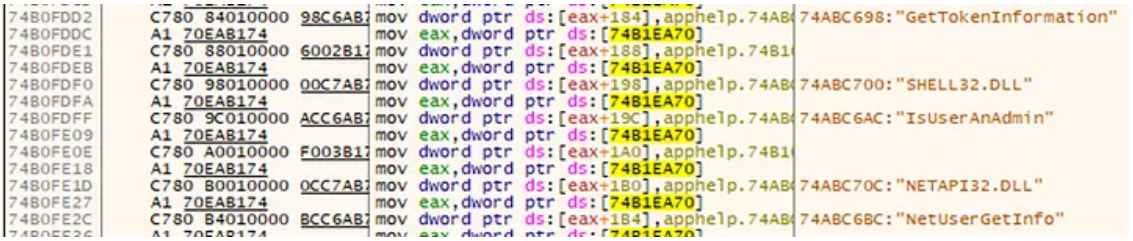
Backdoor commands (Symbols tab). It reinforces the fact that there is C2 server.

Press enter or click to view image in full size

713E816C	Import	ws2_32.Ordinal#4		connect
713E8170	Import	ws2_32.Ordinal#115		WSAStartup
713E8174	Import	ws2_32.Ordinal#11		inet_addr
713E8178	Import	ws2_32.Ordinal#3		closesocket
713E817C	Import	ws2_32.Ordinal#16		recv
713E8180	Import	ws2_32.Ordinal#23		socket
713E8184	Import	ws2_32.Ordinal#52		gethostbyname
713E8188	Import	ws2_32.Ordinal#19		send
713E818C	Import	ws2_32.Ordinal#21		setsockopt
713E8190	Import	ws2_32.Ordinal#9		htons

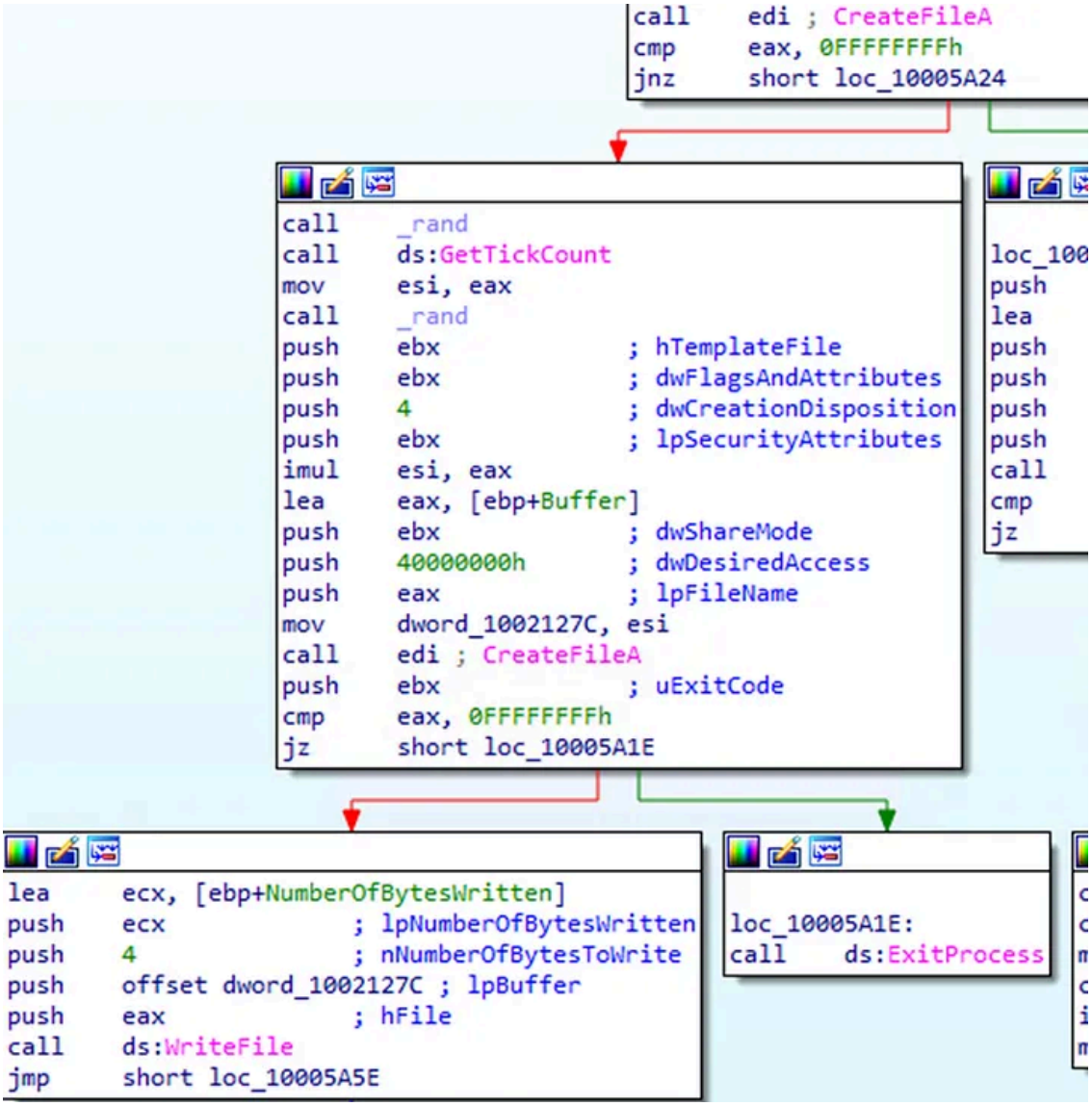
The malware checks if the current user is a member of the Administrator's group. If **yes**, use Netapi32.dll (querying and managing network interfaces) library to get information about the user.

Press enter or click to view image in full size



“PortDoor,” writes the GetTickCount value (to see how long the hostname is awake since started).

Press enter or click to view image in full size



The decryption of string by the backdoor. the register of “mov” and “add” to the “eax” register, then it repeats it with the “ecx” register and “mov” all the parameters to him.

Press enter or click to view image in full size

```
75445AA9    8B45 F8     mov eax,dword ptr ss:[ebp-8]     [ebp-8]:EntryPoint
75445AAC    0345 FC     add eax,dword ptr ss:[ebp-4]
75445AAF    0FB600      movzx eax,byte ptr ds:[eax]
75445AB2    35 FE000000 xor eax,FE
75445AB7    8B4D F8     mov ecx,dword ptr ss:[ebp-8]     ecx:EntryPoint, [ebp-8]:EntryPoint
75445ABA    034D FC     add ecx,dword ptr ss:[ebp-4]     ecx:EntryPoint
75445ABD    8801        mov byte ptr ds:[ecx],al         ecx:EntryPoint
```

To verify if it is the backdoor activity, we checked in the Memory map to see the permissions. Can see clearly that it the Executable code of our backdoor.

Press enter or click to view image in full size

```
74B71000 00017000  ".text"    Executable code    IMG    ER---    ERWC
```

Check what cause to debugger send us an “Exception” message when we try to run until the BP on xor loop. As you can see the xor loop is located between **two** IsDebuggerPresent functions that check if the malware in the debugger or not.

```
.text:10005A74                  mov     [ebp+var_8], offset unk_100201E8
.text:10005A7B                  call    ds:IsDebuggerPresent
.text:10005A81                  and     [ebp+var_4], 0
.text:10005A85                  jmp     short loc_10005A8E
.text:10005A87 ; ---------------------------------------------------------------------------
.text:10005A87
.text:10005A87 loc_10005A87:                            ; CODE XREF: sub_10005AA9+16↓j
.text:10005A87                  mov     eax, [ebp+var_4]
.text:10005A8A                  inc     eax
.text:10005A8B                  mov     [ebp+var_4], eax
.text:10005A8E
.text:10005A8E loc_10005A8E:                            ; CODE XREF: sub_10005A6F+16↑j
.text:10005A8E                  cmp     [ebp+var_4], 0BDh
.text:10005A95                  jnb     short loc_10005AC1
.text:10005A95 sub_10005A6F     endp
.text:10005A95
.text:10005A97
.text:10005A97 ; =============== S U B R O U T I N E =======================================
.text:10005A97
.text:10005A97
.text:10005A97 sub_10005A97     proc near               ; CODE XREF: sub_10005A97+5↓p
.text:10005A97                  jz      short sub_10005AA9
.text:10005A99                  jnz     short sub_10005AA9
.text:10005A9B                  pusha
.text:10005A9C                  call    sub_10005A97
.text:10005AA1                  call    sub_10005AA9
.text:10005AA6                  retn    80h
.text:10005AA6 sub_10005A97     endp ; sp-analysis failed
.text:10005AA6
.text:10005AA9
.text:10005AA9 ; =============== S U B R O U T I N E =======================================
.text:10005AA9
.text:10005AA9
.text:10005AA9 sub_10005AA9     proc near               ; CODE XREF: sub_10005A97↑j
.text:10005AA9                                          ; sub_10005A97+2↑j ...
.text:10005AA9                  mov     eax, [ebp-8]
.text:10005AAC                  add     eax, [ebp-4]
.text:10005AAF                  movzx   eax, byte ptr [eax]
.text:10005AB2                  xor     eax, 0FEh
.text:10005AB7                  mov     ecx, [ebp-8]
.text:10005ABA                  add     ecx, [ebp-4]
.text:10005ABD                  mov     [ecx], al
.text:10005ABF                  jmp     short loc_10005A87
.text:10005ABF sub_10005AA9     endp
.text:10005ABF
.text:10005AC1 ; ---------------------------------------------------------------------------
.text:10005AC1 ; START OF FUNCTION CHUNK FOR sub_10005A6F
.text:10005AC1
.text:10005AC1 loc_10005AC1:                            ; CODE XREF: sub_10005A6F+26↑j
.text:10005AC1                  call    ds:IsDebuggerPresent
.text:10005AC7                  call    sub_10005938
```

The malware uses the Anti-Debugging technique before it runs the xor loop.

- **Reference**: https://www.cybereason.com/blog/research/portdoor-new-chinese-apt-backdoor-attack-targets-russian-defense-sector

Indicators of compromise:

Email- Phishing:

48a312bfbcd1674501a633fbdcaa99a487e6260414a6e450a19982578b128a52

RTF:

774a54300223b421854d2e90bcf75ae25df75ba9f3da1b9eb01138301cdd258f

DLL:

2d705f0b76f24a18e08163db2f187140ee9f03e43697a9ea0d840c829692d43c

---