New Conversation Hijacking Campaign Delivering IcedID

Written by Joakim Kennedy and Ryan Robinson - 28 March 2022

This post describes the technical analysis of a new campaign detected by Intezer's research team, which initiates attacks with a phishing email that uses conversation hijacking to deliver IcedID.

The underground economy is constantly evolving with threat actors specializing in specific fields. One field that has bloomed in the last few years is initial access brokers. Initial access brokers specialize in gaining an initial beachhead access to organizations and once achieved, sell the access to other threat actors that monetize it further.

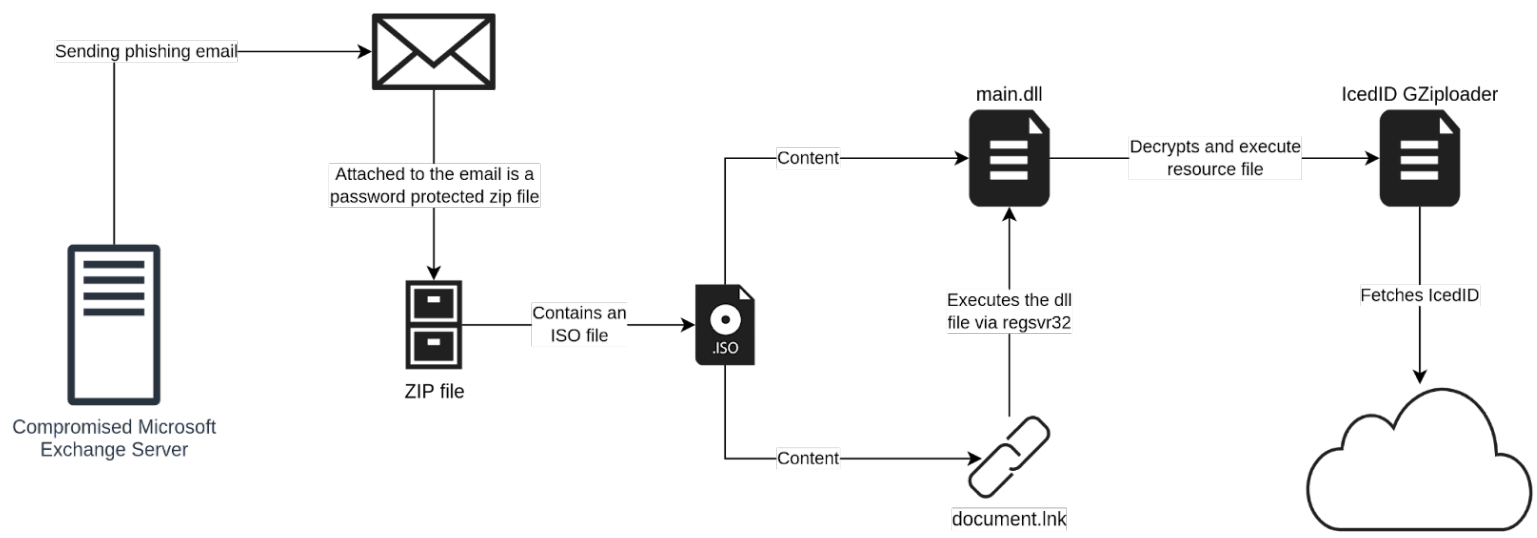
Some of the customers to initial access brokers buy the access to deploy ransomware. Proofpoint has identified ten access brokers that sell access to ransomware groups. These access brokers largely infect their victims with banking trojans that are later used to deploy another malware at the "purchaser's request."

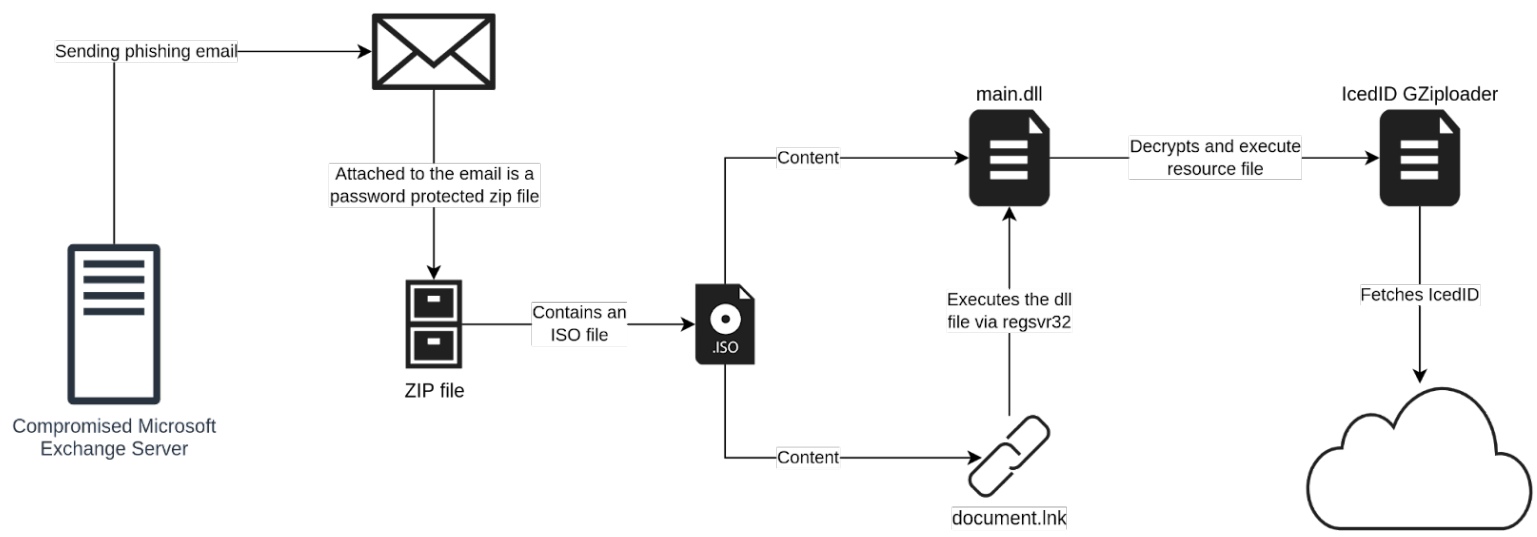
One of these banking trojans that have been used to deploy ransomware is IcedID (BokBot). IcedID was first reported on by IBM X-Force in November 2017 and the malware shared some code with Pony. While initially designed to steal banking credentials, like many other banking trojans, the malware has been repurposed for deploying other malware on the infected machines.

One way IcedID infects machines is via phishing emails. The infection chain that commonly has been used is an email with an attached password protected "zip" archive. Inside the archive is a macro enabled office document that executes the IcedID installer. Some phishing emails reuse previously stolen emails to make the lure more convincing.

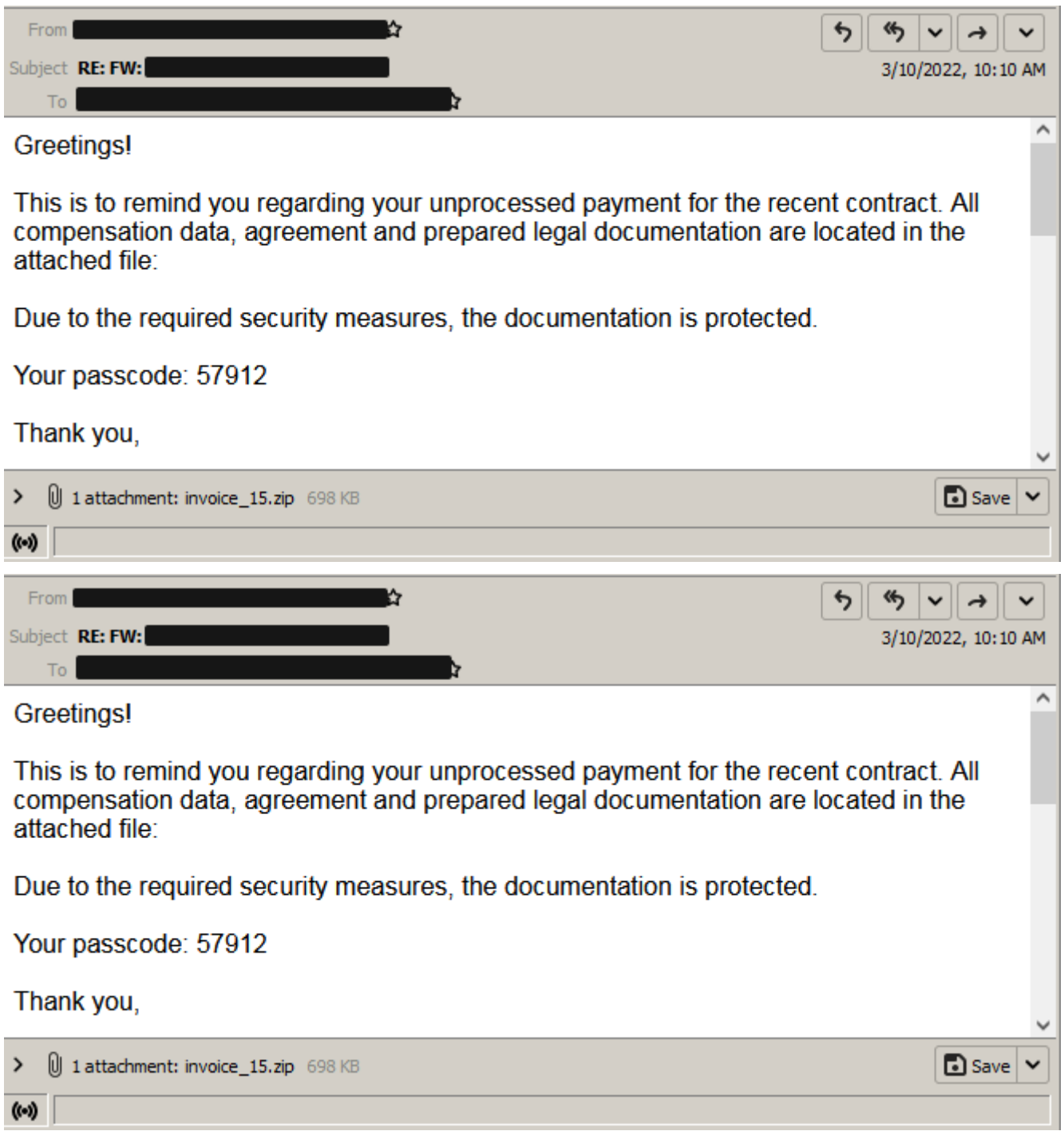
In the new IcedID campaign we have discovered a further evolution of the threat actors' technique. The threat actor now uses compromised Microsoft Exchange servers to send the phishing emails from the account that they stole from. The payload has also moved away from using office documents to the use of ISO files with a Windows LNK file and a DLL file. The use of ISO files allows the threat actor to bypass the Mark-of-the-Web controls, resulting in execution of the malware without warning to the user. With regards to targeting, we have seen organizations within energy, healthcare, law, and pharmaceutical sectors.

Infection Chain





The attack-chain starts with a phishing email. The email includes a message about some important document and has a password protected "zip" archive file attached. The password to the archive is given in the email body, as can be seen in the screenshot below. What makes the phishing email more convincing is that it's using conversation hijacking (thread hijacking). A forged reply to a previous stolen email is being used. Additionally, the email has also been sent from the email account from whom the email was stolen from.



The content of the zip archive is shown in the screenshot below. It includes a single “ISO” file with the same filename as the zip archive. It can also be seen that the file was created not that long before the email was sent.

Name	Size	Packed Size	Modified	Created	Accessed
invoice_15.iso	1 271 808	714 748	2019-11-11 21:00	2022-03-10 09:24	2022-03-10 09:24

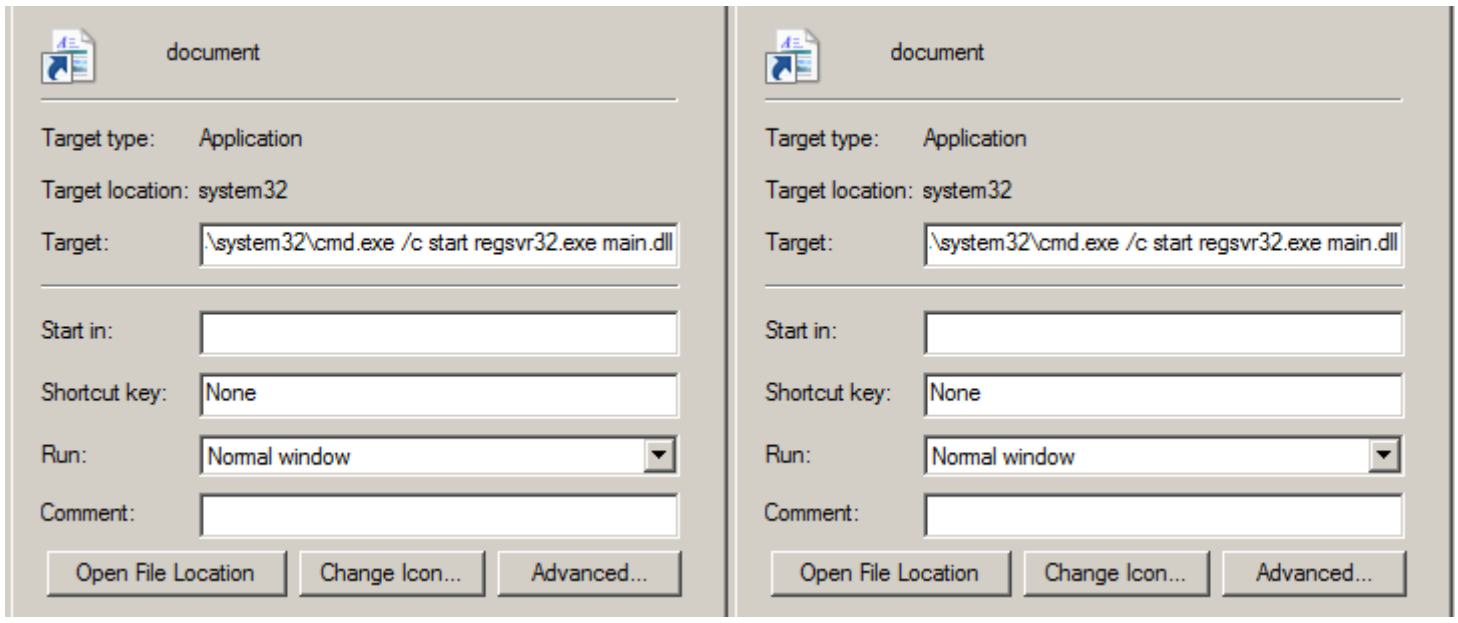
Name	Size	Packed Size	Modified	Created	Accessed
invoice_15.iso	1 271 808	714 748	2019-11-11 21:00	2022-03-10 09:24	2022-03-10 09:24

The ISO file includes two files, a LNK file named “document” and a DLL file named “main.” From the timestamps it can be concluded that the DLL file was prepared the day before while the LNK file was prepared about a week before. It is possible that the LNK file has been used in earlier phishing emails.

Name	Size	Packed Size	Modified
document.lnk	2 798	2 798	2022-03-03 18:35
main.dll	1 204 736	1 204 736	2022-03-09 16:26

Name	Size	Packed Size	Modified
document.lnk	2 798	2 798	2022-03-03 18:35
main.dll	1 204 736	1 204 736	2022-03-09 16:26

The LNK file has been made to look like a document file via its embedded icon file. As can be seen in the screenshot below, when a user double clicks the link file, it uses “regsvr32” to execute the DLL file.



The use of regsvr32 allows for proxy execution of malicious code in main.dll for defense evasion. The DLL file is a loader for the IcedID payload. It contains a number of exports, most of which consist of junk code.

Name	Address	Ordinal
DllGetClassObject	**0000000180056770**	**1**
DllRegisterServer	**00000001800568C0**	**2**
PluginInit	0000000180056CA0	3
aoflzkfwvdmcyxdl	0000000180056FF0	4
bttdeeedabgnsezg	0000000180056FB0	5
cttsnnarqxwd	0000000180056FD0	6
eygomnkcpqpilfqsr	0000000180056F10	7
hdeylqseigrra	0000000180056DF0	8
hkjehmypbmo	0000000180056E10	9
htallgyzd	0000000180056E90	10
hwvcazraantyz	0000000180057030	11
ifsunhfoggxojmvka	0000000180056F70	12
ijwxmfjmec	0000000180056E70	13
iokvvqtxkqgivps	0000000180057010	14
joynovxqivdfapbc	0000000180056EF0	15
kuwodho	0000000180056F50	16
mhbhlfcoqwltakbr	0000000180056A10	17
mjbisvugvmsu	0000000180056F30	18
nftsuscyjsxmn	0000000180056E30	19
nmykguaw	0000000180056F90	20
qjurwnmbegpln	0000000180056EB0	21
rfhibhk	0000000180056ED0	22
wniqeandiev	0000000180056B50	23
wrzfrhgsqoidw	0000000180056E50	24
DllEntryPoint	**000000018006A444**	**[main entry]**

Name	Address	Ordinal
DllGetClassObject	**0000000180056770**	**1**
DllRegisterServer	**00000001800568C0**	**2**
PluginInit	0000000180056CA0	3
aoflzkfwvdmcyxdl	0000000180056FF0	4
bttdeeedabgnsezg	0000000180056FB0	5
cttsnnarqxwd	0000000180056FD0	6
eygomnkcpqpilfqsr	0000000180056F10	7
hdeylqseigrra	0000000180056DF0	8
hkjehmypbmo	0000000180056E10	9
htallgyzd	0000000180056E90	10
hwvcazraantyz	0000000180057030	11
ifsunhfoggxojmvka	0000000180056F70	12
ijwxmfjmec	0000000180056E70	13
iokvvqtxkqgivps	0000000180057010	14
joynovxqivdfapbc	0000000180056EF0	15
kuwodho	0000000180056F50	16
mhbhlfcoqwltakbr	0000000180056A10	17
mjbisvugvmsu	0000000180056F30	18
nftsuscyjsxmn	0000000180056E30	19
nmykguaw	0000000180056F90	20
qjurwnmbegpln	0000000180056EB0	21
rfhibhk	0000000180056ED0	22
wniqeandiev	0000000180056B50	23
wrzfrhgsqoidw	0000000180056E50	24
DllEntryPoint	**000000018006A444**	**[main entry]**

The loader will locate the encrypted payload, stored in the resource section of the binary. It does this through the technique API hashing. A decompilation of the simple hashing function is shown below.

```
uint32_t fcn.180059640(int64_t arg4, int64_t arg1)

{
    int32_t iVar1;
    char *pcStackX8;
    uint32_t uStack24;

    uStack24 = 0;
    pcStackX8 = (char *)arg4;
    do {
        iVar1 = fcn.180059630((uint64_t)uStack24);
        uStack24 = iVar1 + *pcStackX8;
        pcStackX8 = pcStackX8 + 1;
    } while (*pcStackX8 != '\0');
    return uStack24;
}
:>
```

```
uint32_t fcn.180059630(int64_t arg4)

{
    return (uint32_t)arg4 >> 0xd | (uint32_t)arg4 << 0x13;
}
:>
```

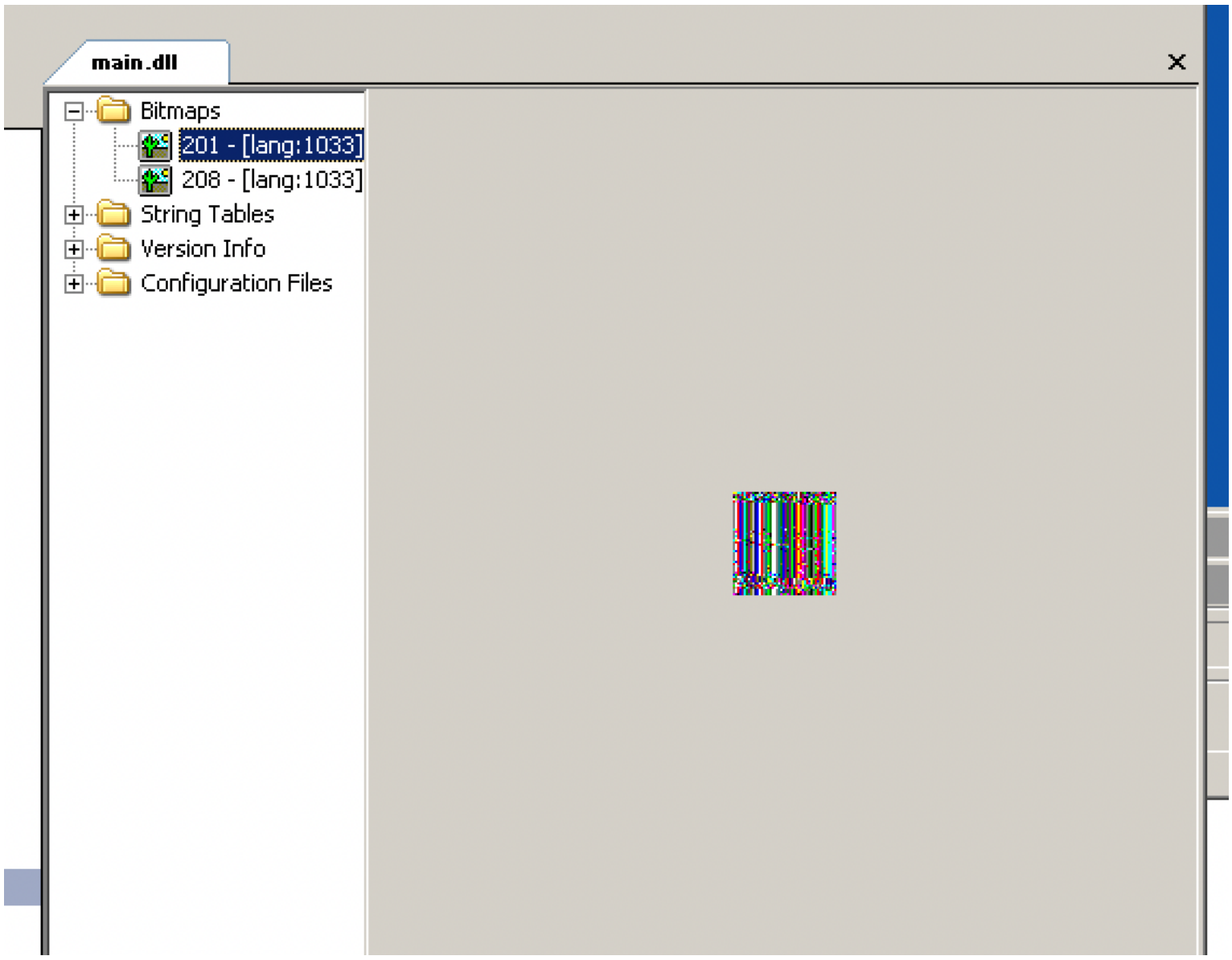

```
uint32_t fcn.180059640(int64_t arg4, int64_t arg1)

{
    int32_t iVar1;
    char *pcStackX8;
    uint32_t uStack24;

    uStack24 = 0;
    pcStackX8 = (char *)arg4;
    do {
        iVar1 = fcn.180059630((uint64_t)uStack24);
        uStack24 = iVar1 + *pcStackX8;
        pcStackX8 = pcStackX8 + 1;
    } while (*pcStackX8 != '\0');
    return uStack24;
}
:>
```

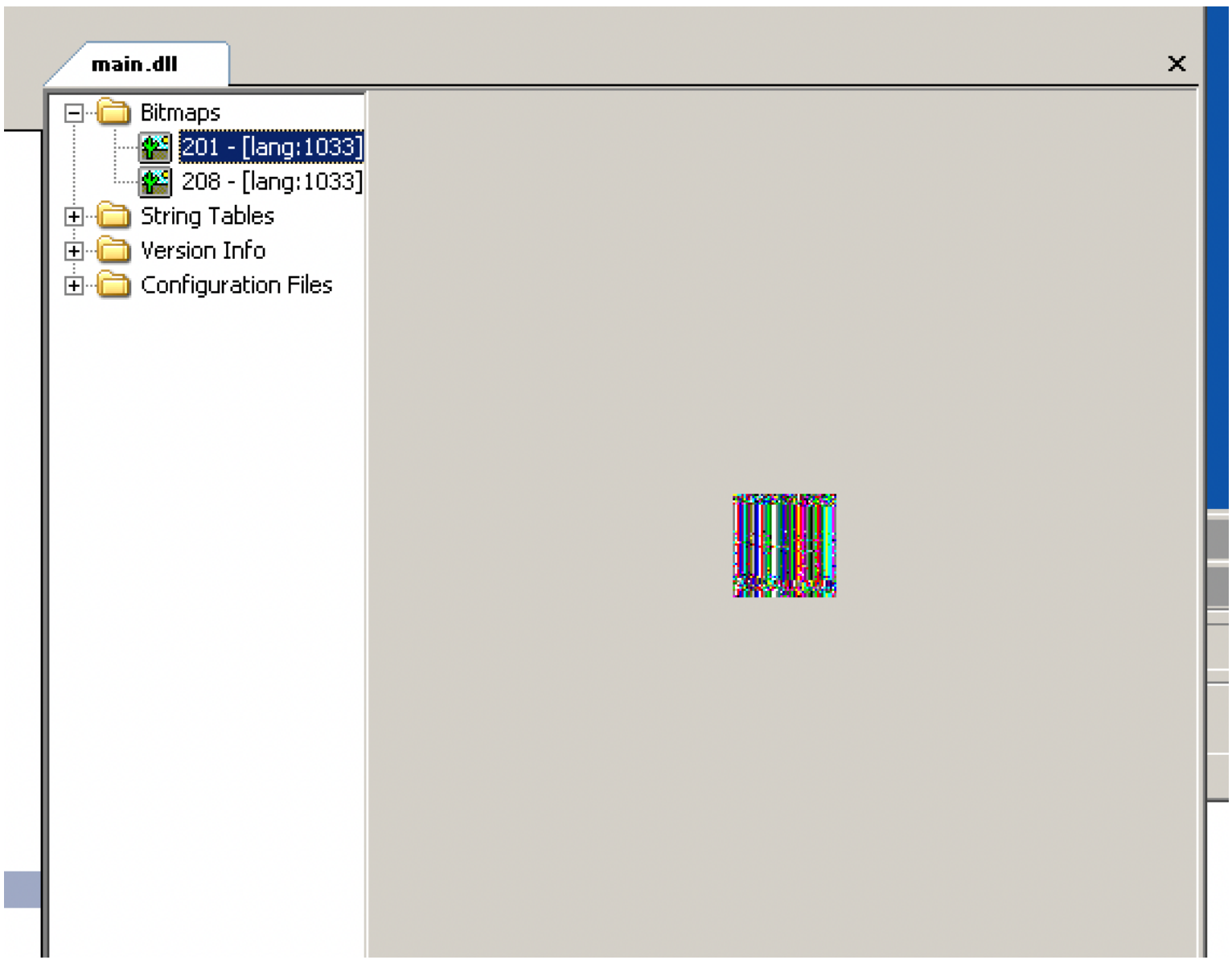
```
uint32_t fcn.180059630(int64_t arg4)

{
    return (uint32_t)arg4 >> 0xd | (uint32_t)arg4 << 0x13;
}
:>
```


The resulting hash is then compared with a hardcoded hash, locating the call for FindResourceA. The function is dynamically called to fetch the payload.

```
000007FEF1A673E5   48:83EC 68          sub rsp,68
000007FEF1A673E9   48:8B4424 70        mov rax,qword ptr ss:[rsp+70]      [rsp+70]:"MZ"
000007FEF1A673EE   48:894424 38        mov qword ptr ss:[rsp+38],rax      [rsp+38]:"MZ"
000007FEF1A673F3   BA 559AD03B         mov edx,3BD09A55
000007FEF1A673F8   B9 5BBC4A6A         mov ecx,6A4ABC5B
000007FEF1A673FD   E8 0E730000         call <main.get_proc_address_by_hash>
000007FEF1A67402   48:894424 40        mov qword ptr ss:[rsp+40],rax
000007FEF1A67407   41:B8 02000000      mov r8d,2
000007FEF1A6740D   BA C9000000         mov edx,C9
000007FEF1A67412   48:8B4C24 38        mov rcx,qword ptr ss:[rsp+38]      [rsp+38]:"MZ"
000007FEF1A67417   FF5424 40           call qword ptr ss:[rsp+40]
000007FEF1A6741B   48:894424 30        mov qword ptr ss:[rsp+30],rax
000007FEF1A67420   48:8B4424 30        mov rax,qword ptr ss:[rsp+30]
000007FEF1A67425   8B40 04             mov eax,dword ptr ds:[rax+4]
000007FEF1A67428   894424 20           mov dword ptr ss:[rsp+20],eax
000007FEF1A6742C   48:8B4424 30        mov rax,qword ptr ss:[rsp+30]

word ptr ss:[rsp+40]=[000000000010E0A0 <&FindResourceA>]=<kernel32.FindResourceA>

text:000007FEF1A67417 main.dll:$57417 #56817

Dump 1 | Dump 2 | Dump 3 | Dump 4 | Dump 5 | Watch 1 | [x=] Locals | Struct
```


```
000007FEF1A673E5   48:83EC 68          sub rsp,68
000007FEF1A673E9   48:8B4424 70        mov rax,qword ptr ss:[rsp+70]      [rsp+70]:"MZ"
000007FEF1A673EE   48:894424 38        mov qword ptr ss:[rsp+38],rax      [rsp+38]:"MZ"
000007FEF1A673F3   BA 559AD03B         mov edx,3BD09A55
000007FEF1A673F8   B9 5BBC4A6A         mov ecx,6A4ABC5B
000007FEF1A673FD   E8 0E730000         call <main.get_proc_address_by_hash>
000007FEF1A67402   48:894424 40        mov qword ptr ss:[rsp+40],rax
000007FEF1A67407   41:B8 02000000      mov r8d,2
000007FEF1A6740D   BA C9000000         mov edx,C9
000007FEF1A67412   48:8B4C24 38        mov rcx,qword ptr ss:[rsp+38]      [rsp+38]:"MZ"
000007FEF1A67417   FF5424 40           call qword ptr ss:[rsp+40]
000007FEF1A6741B   48:894424 30        mov qword ptr ss:[rsp+30],rax
000007FEF1A67420   48:8B4424 30        mov rax,qword ptr ss:[rsp+30]
000007FEF1A67425   8B40 04             mov eax,dword ptr ds:[rax+4]
000007FEF1A67428   894424 20           mov dword ptr ss:[rsp+20],eax
000007FEF1A6742C   48:8B4424 30        mov rax,qword ptr ss:[rsp+30]

word ptr ss:[rsp+40]=[000000000010E0A0 <&FindResourceA>]=<kernel32.FindResourceA>

text:000007FEF1A67417 main.dll:$57417 #56817

Dump 1 | Dump 2 | Dump 3 | Dump 4 | Dump 5 | Watch 1 | [x=] Locals | Struct
```





Memory is allocated using VirtualAlloc to hold the decrypted payload.

```
var_638], xmm0

mov     rax, [rsp+658h+var_578]
mov     eax, [rax+50h]
mov     r9d, 40h ; '@'   ; flProtect
mov     r8d, 3000h       ; flAllocationType
mov     edx, eax         ; dwSize
xor     ecx, ecx         ; lpAddress
call    cs:VirtualAlloc
mov     [rsp+658h+var_5F8], rax
```


```
var_638], xmm0

mov     rax, [rsp+658h+var_578]
mov     eax, [rax+50h]
mov     r9d, 40h ; '@'   ; flProtect
mov     r8d, 3000h       ; flAllocationType
mov     edx, eax         ; dwSize
xor     ecx, ecx         ; lpAddress
call    cs:VirtualAlloc
mov     [rsp+658h+var_5F8], rax
```


The IcedID “Gziploader” payload is decoded and placed in memory and then executed. GZiploader fingerprints the machine and sends a beacon to the command and control server with information about the infected host. The information is smuggled through the cookies header via an HTTP GET request.

```
UNICODE
.......................................Cookie: __gads=3993579037
:1:54251:38; _gat=6.1.7601.64; _ga=1.591594.2020557398.138; _u=5
7494E2D3838373638414D4F325230:61646D696E:3544313038363838443042 3
645304236; __io=21_148699988_1048996948_4080777056; _gid=4008498
8BA9A.....................................................
.........................................................
```


```
UNICODE
.......................................Cookie: __gads=3993579037
:1:54251:38; _gat=6.1.7601.64; _ga=1.591594.2020557398.138; _u=5
7494E2D3838373638414D4F325230:61646D696E:3544313038363838443042 3
645304236; __io=21_148699988_1048996948_4080777056; _gid=4008498
8BA9A.....................................................
.........................................................
```


The C2 is located at yourgroceries[.]top. The C2 can respond with a further stage to be dropped and executed. The C2 did not respond with a payload during our analysis.

```
00000000001E1156   85C0                  test eax,eax
00000000001E1158   74 15                 je 1E116F
00000000001E115A   66:0F6F05 DE7F0000    movdqa xmm0,xmmword ptr ds:[1E9140]     00000000001E9140:"c:\\ProgramData\\"
00000000001E1162   48:8D5424 30          lea rdx,qword ptr ss:[rsp+30]
00000000001E1167   F3:0F7F4424 30        movdqu xmmword ptr ss:[rsp+30],xmm0
00000000001E116D   EB 0A                 jmp 1E1179
00000000001E116F   66:C745 70 5C00       mov word ptr ss:[rbp+70],5C
00000000001E1175   48:8D55 70            lea rdx,qword ptr ss:[rbp+70]
00000000001E1179   FF15 C15F0000         call qword ptr ds:[<&lstrcatA>]
00000000001E117F   48:8D53 0A            lea rdx,qword ptr ds:[rbx+A]
00000000001E1183   48:8D4C24 40          lea rcx,qword ptr ss:[rsp+40]
00000000001E1188   FF15 B25F0000         call qword ptr ds:[<&lstrcatA>]
00000000001E118E   33D2                  xor edx,edx
00000000001E1190   48:8D4C24 40          lea rcx,qword ptr ss:[rsp+40]
00000000001E1195   FF15 B55F0000         call qword ptr ds:[<&CreateDirectoryA>]
00000000001E119B   48:8D53 2A            lea rdx,qword ptr ds:[rbx+2A]
00000000001E119F   48:8D4C24 40          lea rcx,qword ptr ss:[rsp+40]
00000000001E11A4   FF15 965F0000         call qword ptr ds:[<&lstrcatA>]
00000000001E11AA   48:8D53 0A            lea rdx,qword ptr ds:[rbx+A]
00000000001E11AE   48:8BCE               mov rcx,rsi
00000000001E11B1   FF15 915F0000         call qword ptr ds:[<&lstrcpy>]
00000000001E11B7   48:8D53 2A            lea rdx,qword ptr ds:[rbx+2A]
00000000001E11BB   48:8BCE               mov rcx,rsi
00000000001E11BE   FF15 7C5F0000         call qword ptr ds:[<&lstrcatA>]
00000000001E11C4   4D:8BC7               mov r8,r15
00000000001E11C7   48:8D93 C6020000      lea rdx,qword ptr ds:[rbx+2C6]
00000000001E11CE   48:8D4C24 40          lea rcx,qword ptr ss:[rsp+40]
00000000001E11D3   E8 20000000           call <write_to_file>
00000000001E11D8   4C:8D9C24 50010000    lea r11,qword ptr ss:[rsp+150]
00000000001E11E0   49:8B5B 28            mov rbx,qword ptr ds:[r11+28]
00000000001E11E4   49:8B73 30            mov rsi,qword ptr ds:[r11+30]
00000000001E11E8   49:8B7B 38            mov rdi,qword ptr ds:[r11+38]
00000000001E11EC   49:8BE3               mov rsp,r11
00000000001E11EF   41:5F                 pop r15
00000000001E11F1   41:5E                 pop r14
00000000001E11F3   5D                    pop rbp
00000000001E11F4   C3                    ret
00000000001E11F5   CC                    int3
00000000001E11F6   CC                    int3
```


```
00000000001E1156   85C0                  test eax,eax
00000000001E1158   74 15                 je 1E116F
00000000001E115A   66:0F6F05 DE7F0000    movdqa xmm0,xmmword ptr ds:[1E9140]     00000000001E9140:"c:\\ProgramData\\"
00000000001E1162   48:8D5424 30          lea rdx,qword ptr ss:[rsp+30]
00000000001E1167   F3:0F7F4424 30        movdqu xmmword ptr ss:[rsp+30],xmm0
00000000001E116D   EB 0A                 jmp 1E1179
00000000001E116F   66:C745 70 5C00       mov word ptr ss:[rbp+70],5C
00000000001E1175   48:8D55 70            lea rdx,qword ptr ss:[rbp+70]
00000000001E1179   FF15 C15F0000         call qword ptr ds:[<&lstrcatA>]
00000000001E117F   48:8D53 0A            lea rdx,qword ptr ds:[rbx+A]
00000000001E1183   48:8D4C24 40          lea rcx,qword ptr ss:[rsp+40]
00000000001E1188   FF15 B25F0000         call qword ptr ds:[<&lstrcatA>]
00000000001E118E   33D2                  xor edx,edx
00000000001E1190   48:8D4C24 40          lea rcx,qword ptr ss:[rsp+40]
00000000001E1195   FF15 B55F0000         call qword ptr ds:[<&CreateDirectoryA>]
00000000001E119B   48:8D53 2A            lea rdx,qword ptr ds:[rbx+2A]
00000000001E119F   48:8D4C24 40          lea rcx,qword ptr ss:[rsp+40]
00000000001E11A4   FF15 965F0000         call qword ptr ds:[<&lstrcatA>]
00000000001E11AA   48:8D53 0A            lea rdx,qword ptr ds:[rbx+A]
00000000001E11AE   48:8BCE               mov rcx,rsi
00000000001E11B1   FF15 915F0000         call qword ptr ds:[<&lstrcpy>]
00000000001E11B7   48:8D53 2A            lea rdx,qword ptr ds:[rbx+2A]
00000000001E11BB   48:8BCE               mov rcx,rsi
00000000001E11BE   FF15 7C5F0000         call qword ptr ds:[<&lstrcatA>]
00000000001E11C4   4D:8BC7               mov r8,r15
00000000001E11C7   48:8D93 C6020000      lea rdx,qword ptr ds:[rbx+2C6]
00000000001E11CE   48:8D4C24 40          lea rcx,qword ptr ss:[rsp+40]
00000000001E11D3   E8 20000000           call <write_to_file>
00000000001E11D8   4C:8D9C24 50010000    lea r11,qword ptr ss:[rsp+150]
00000000001E11E0   49:8B5B 28            mov rbx,qword ptr ds:[r11+28]
00000000001E11E4   49:8B73 30            mov rsi,qword ptr ds:[r11+30]
00000000001E11E8   49:8B7B 38            mov rdi,qword ptr ds:[r11+38]
00000000001E11EC   49:8BE3               mov rsp,r11
00000000001E11EF   41:5F                 pop r15
00000000001E11F1   41:5E                 pop r14
00000000001E11F3   5D                    pop rbp
00000000001E11F4   C3                    ret
00000000001E11F5   CC                    int3
00000000001E11F6   CC                    int3
```


Conversation Hijacking as a Phishing Technique

The technique of hijacking an already existing conversation over email to spread malware is something threat actors have been using for a while. Normally email messages are stolen during an infection and used in future attacks to make the phishing email appear more legitimate. In the last six months, threat actors have evolved the technique further to make it even more convincing. Instead of sending the stolen conversation to the victim with a "spoofed" email address, threat actors are now using the email address of the victim that they stole the original email from to make the phishing email even more convincing.

Kevin Beaumont reported on this conversation hijacking technique back in November 2021 being used to distribute Qakbot. Through the investigation, he confirmed that the Microsoft Exchange servers where the emails originated from had evidence of being exploited by ProxyShell.

New Campaign Discovered in March 2022

In the current mid-March campaign, we have discovered reuse of the same stolen conversation now being sent from the email address that received the latest email. Back in January when this conversation was also used, the "FROM" address was "webmaster@[REDACTED].com" with the name of the recipient of the last email in the conversation. By using this approach, the email appears more legitimate and is transported through the normal channels which can also include security products.

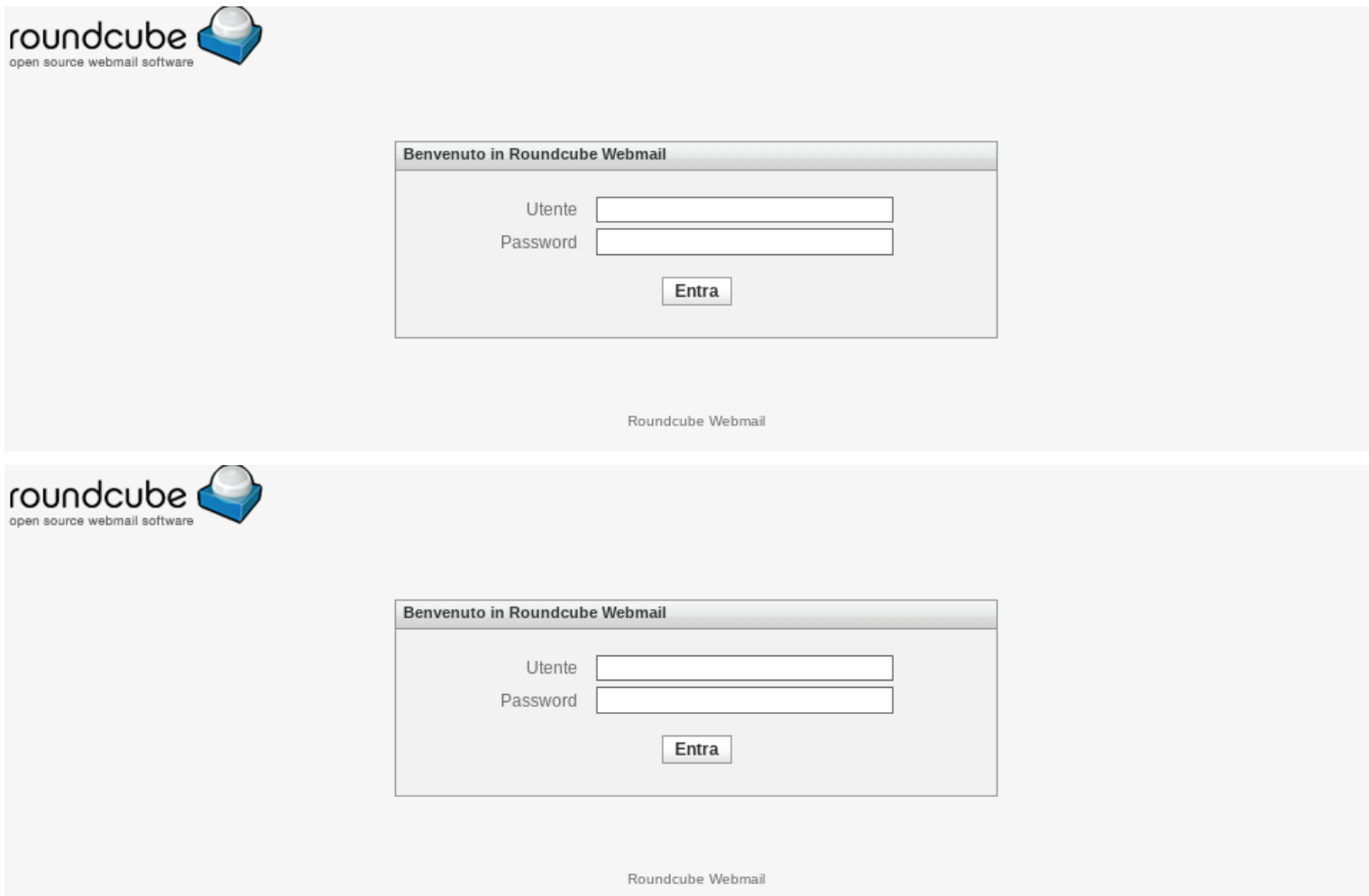
The majority of the originating Exchange servers we have observed appear to also be unpatched and publicly exposed, making the ProxyShell vector a good theory. While the majority of the Exchange servers used to send the phishing emails can be accessed by anyone over the Internet, we have also seen a phishing email sent internally on what appears to be an "internal" Exchange server.

The code snippet below shows a small part of the email header. The IP address of the Exchange server is a local IP address (172.29.0.12) with a top-level domain name of "local". We can also see a header added by Exchange marking it as an internal email. The exchange server also has added a header of the original client (172.29.5.131 which also is a local IP address) that connected to the Exchange server over MAPI.

```
Received: from ExchSrv01.[REDACTED].local (172.29.0.12) by ExchSrv01.[REDACTED].local (172.29.0.12) with
Microsoft SMTP Server (version=TLS1_2, cipher=TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384) id 15.2.464.5 via
```

```
Mailbox Transport; Thu, 10 Mar 2022 14:34:29 +0100 Received: from ExchSrv01.[REDACTED].local (172.29.0.12) by ExchSrv01.[REDACTED].local (172.29.0.12) with Microsoft SMTP Server (version=TLS1_2, cipher=TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384) id 15.2.464.5; Thu, 10 Mar 2022 14:34:29 +0100 Received: from ExchSrv01.[REDACTED].local ([fe80::b148:8e7:61f8:61b4]) by ExchSrv01.[REDACTED].local ([fe80::b148:8e7:61f8:61b4%6]) with mapi id 15.02.0464.005; Thu, 10 Mar 2022 14:34:29 +0100 … X-MS-Exchange-Organization-AuthAs: Internal X-MS-Exchange-Organization-AuthMechanism: 04 X-MS-Exchange-Organization-AuthSource: ExchSrv01.[REDACTED].local X-MS-Has-Attach: yes X-MS-Exchange-Organization-SCL: -1 X-MS-Exchange-Organization-RecordReviewCfmType: 0 x-ms-exchange-organization-originalclientipaddress: 172.29.5.131 x-ms-exchange-organization-originalserveripaddress: fe80::b148:8e7:61f8:61b4%6
```

We didn't manage to find a corresponding public IP address for this Exchange server and it is not known to us how it was accessed by the threat actor. The only thing we managed to find was a roundcube webmail instance. The login page is shown in the screenshot below.



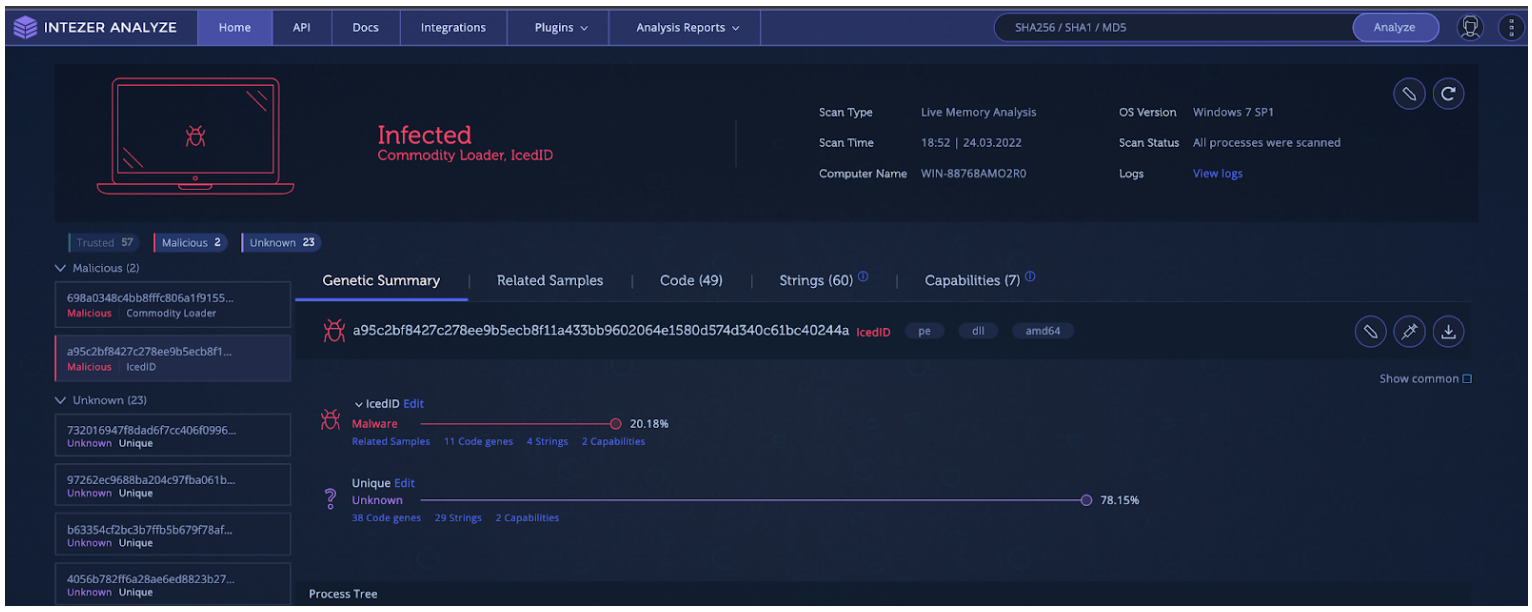
One of the headers in the snippet above reported that the client connected to the server via MAPI. MAPI is a protocol used (for example, by Outlook) to access the mailbox on an Exchange server. This suggests that the threat actor used an Exchange client instead of using SMTP to send the email. We have also seen the header "X-Mailer: Microsoft Outlook 16.0" in multiple phishing emails. In other phishing emails a "X-Originating-IP" header can be found. This is a header added by the Exchange server when the web interface is used. The IP address in the header is that of the client that connected to the server. We have observed both hosting providers and non-commercial IP addresses for the client IP.

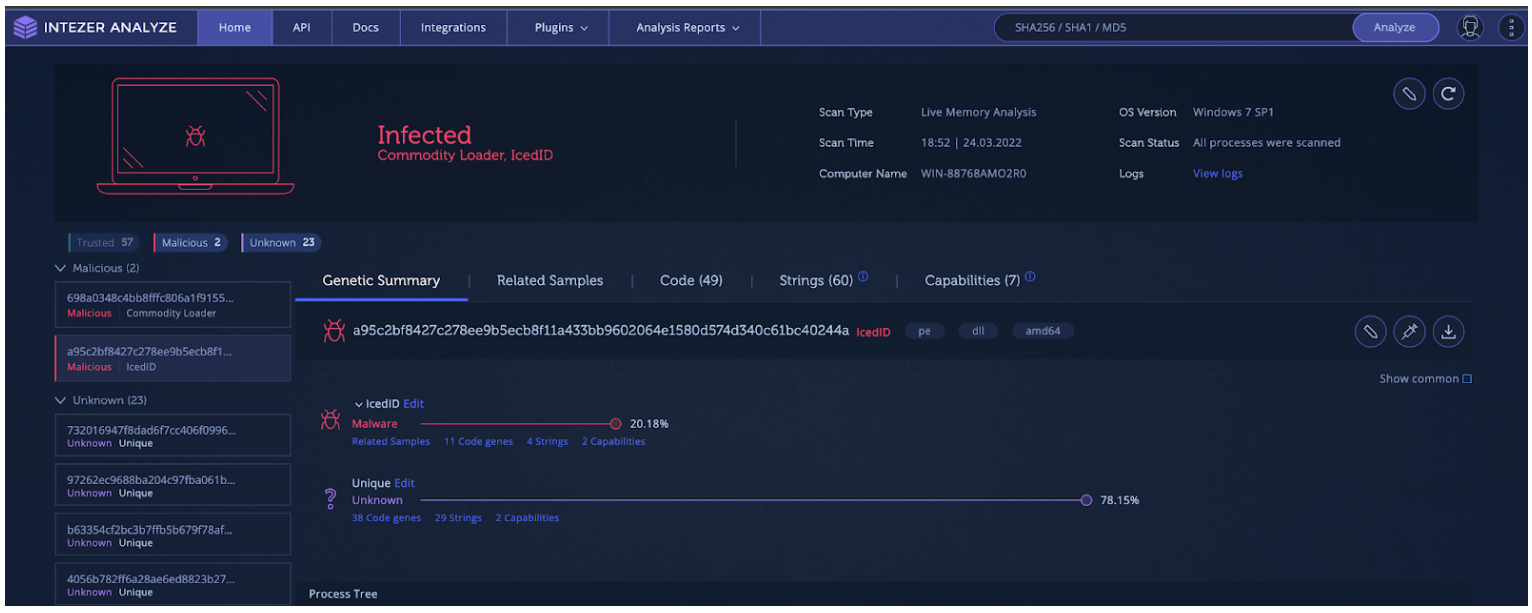
Attribution

In June 2021, Proofpoint released a report on different access brokers that facilitates access for ransomware groups. Of the different threat actors, according to Proofpoint, two of them (TA577 and TA551) used IcedID as their malware. The techniques used by TA551 include conversation hijacking and password protected zip files. The group is also known to use regsvr32.exe for signed binary proxy execution for malicious DLLs.

Summary

The use of conversation hijacking is a powerful social engineering technique that can increase the rate of a successful phishing attempt. The payload has been moved away from office documents to the use of ISO files, employing the use of commodity packers and multiple stages to hide activity. It is important to be able to detect malicious files in memory to detect this type of attack. We recommend you use an endpoint scanner.





IoCs

`ISO File: 3542d5179100a7644e0a747139d775dbc8d914245292209bc9038ad2413b3213 Loader DLL: 698a0348c4bb8fffc806a1f915592b20193229568647807e88a39d2ab81cb4c2 LNK File: a17e32b43f96c8db69c979865a8732f3784c7c42714197091866473bcfac8250 IcedID GZiploader Network: yourgroceries[.]top`

Joakim Kennedy

Dr. Joakim Kennedy is a Security Researcher analyzing malware and tracking threat actors on a daily basis. For the last few years, Joakim has been researching malware written in Go. To make the analysis easier he has written the Go Reverse Engineering Toolkit (github.com/goretk), an open-source toolkit for analysis of Go binaries.

Ryan Robinson

Ryan is a security researcher analyzing malware and scripts. Formerly, he was a researcher on Anomali's Threat Research Team.

conversation hijacking DLL file icedID IoCs LNK file Microsoft Exchange Server Phishing ransomware