Zeus Sphinx Back in Business: Some Core Modifications Arise

securityintelligence.com/posts/zeus-sphinx-back-in-business-some-core-modifications-arise/

Malware May 11, 2020

By Nir Shwarts co-authored by Limor Kessem 8 min read

The Zeus Sphinx banking Trojan is financial malware that was built upon the existing and leaked codebase of the forefather of many other Trojans in this class: Zeus v2.0.8.9. Over the years, Sphinx has been in different hands, initially offered as a commodity in underground forums and then suspected to be operated by various closed gangs.

After a lengthy hiatus, this malware began stepping up attack campaigns starting in late 2019 and increased its spreading power in the first quarter of 2020 via malspam featuring coronavirus relief payment updates.

With Sphinx back in the financial cybercrime arena, IBM X-Force wrote the following technical analysis of the Sphinx Trojan's current version, which was first released into the wild in late 2019. We will be covering the following components, shedding light on parts of the malware that were modified in this version, as other parts likely remained the same:

- Persistence mechanism
- Injection tactics
- Bot configuration
- Hidden configuration nuggets
- Bot identification method
- Sphinx's naming algorithms

Let's dive in.

Establishing Persistence

Almost any malware nowadays seeks to establish persistence on infected devices, both desktop and mobile, with the goal of surviving system reboots. Sphinx establishes persistence using a very common method: adding a Run key to the Windows Registry. This tactic has been used by Sphinx since its earliest versions, released in 2015.

HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run

Byychad REG_SZ C:\Users\[redacted]\AppData\Roaming\Guubd\uhhod.exe

Figure 1: Run key set for Sphinx's executable payload

Since Sphinx's malicious payload can come in two different formats, an executable file or a dynamic link library (DLL), it also sets the Registry Run key according to the format being installed. For the DLL format, we would see the following string type created:

Ciuhifa REG_SZ rundll32.exe C:\Users\[redacted]\AppData\Roaming\Afbe\yffyafu.dll,DllRegisterServer

Figure 2: Run key set for a Sphinx DLL

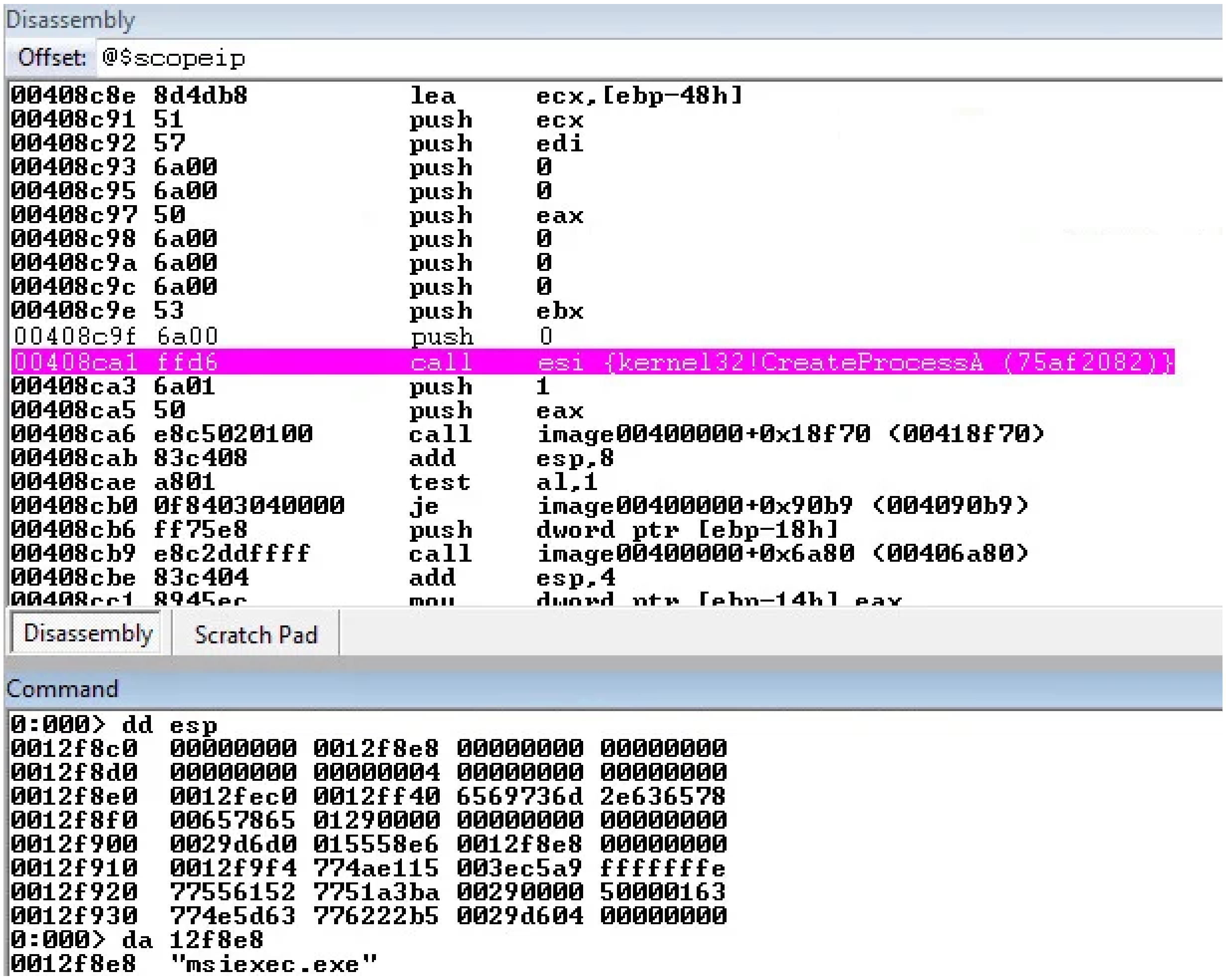
The malicious DLL’s entry point is named *DllRegisterServer*, which is usually an entry point for a COM module. When malware elects to use generic system names for its resources, it is done to blend in with the other benign elements in the operating system (OS).

Sphinx’s Code Injection Choice: Process Injection

Since its main function is to grab user credentials and other personal information from online banking sessions, Zeus Sphinx is designed with the ability to hook browser functions. Before gaining the ability to hook these types of functions, Sphinx has to ensure its stealthy ongoing operations on the OS. It does this by injecting malicious code into other processes first.

The tactic Sphinx uses is a process injection technique:

1. Sphinx calls on the *CreateProcessA* function, which creates a new process and its primary thread. The function’s parameters are *msiexec.exe* for the new process name and the *suspend* flag applied as the process state. This is another part of the malware’s stealth mechanism, as *msiexec.exe* usually stands for the name of a legitimate Windows Installer process that is responsible for installation, storage and removal of programs.



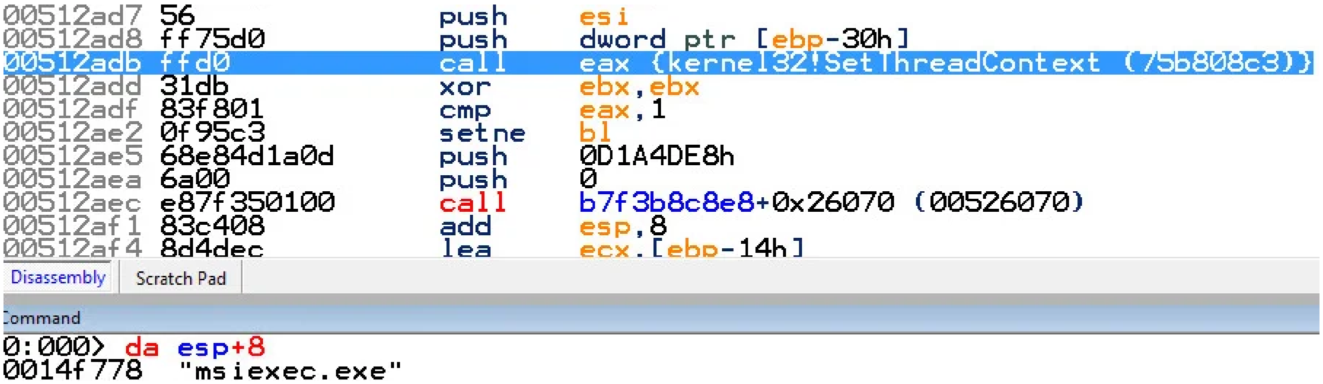
```
00408c8e 8d4db8          lea     ecx,[ebp-48h]
00408c91 51              push    ecx
00408c92 57              push    edi
00408c93 6a00            push    0
00408c95 6a00            push    0
00408c97 50              push    eax
00408c98 6a00            push    0
00408c9a 6a00            push    0
00408c9c 6a00            push    0
00408c9e 53              push    ebx
00408c9f 6a00            push    0
00408ca1 ffd6            call    esi {kernel32!CreateProcessA (75af2082)}
00408ca3 6a01            push    1
00408ca5 50              push    eax
00408ca6 e8c5020100      call    image00400000+0x18f70 (00418f70)
00408cab 83c408          add     esp,8
00408cae a801            test    al,1
00408cb0 0f8403040000    je      image00400000+0x90b9 (004090b9)
00408cb6 ff75e8          push    dword ptr [ebp-18h]
00408cb9 e8c2ddffff      call    image00400000+0x6a80 (00406a80)
00408cbe 83c404          add     esp,4
00408cc1 8945ec          mov     dword ptr [ebp-14h],eax
```

```
0:000> dd esp
0012f8c0  00000000 0012f8e8 00000000 00000000
0012f8d0  00000000 00000004 00000000 00000000
0012f8e0  0012fec0 0012ff40 6569736d 2e636578
0012f8f0  00657865 01290000 00000000 00000000
0012f900  0029d6d0 015558e6 0012f8e8 00000000
0012f910  0012f9f4 774ae115 003ec5a9 fffffffe
0012f920  77556152 7751a3ba 00290000 50000163
0012f930  774e5d63 776222b5 0029d604 00000000
0:000> da 12f8e8
0012f8e8  "msiexec.exe"
```

Figure 3: Sphinx's process injection

1. It calls the *WriteProcessMemory* function to inject a payload into the *msiexec.exe* process.
2. Next, Sphinx changes the execution point of the targeted process to start from the injected payload, using *GetThreadContext* and *SetThreadContex*t functions. *GetThreadContext* is used to get the current extended instruction pointer of the remote process. *SetThreadContext* is used to set the current extended instruction pointer of the remote process.

The extended instruction pointer holds the address of the next instruction.



```
00512ad7 56              push    esi
00512ad8 ff75d0          push    dword ptr [ebp-30h]
00512adb ffd0            call    eax {kernel32!SetThreadContext (75b808c3)}
00512add 31db            xor     ebx,ebx
00512adf 83f801          cmp     eax,1
00512ae2 0f95c3          setne   bl
00512ae5 68e84d1a0d      push    0D1A4DE8h
00512aea 6a00            push    0
00512aec e87f350100      call    b7f3b8c8e8+0x26070 (00526070)
00512af1 83c408          add     esp,8
00512af4 8d4dec          lea     ecx,[ebp-14h]
```

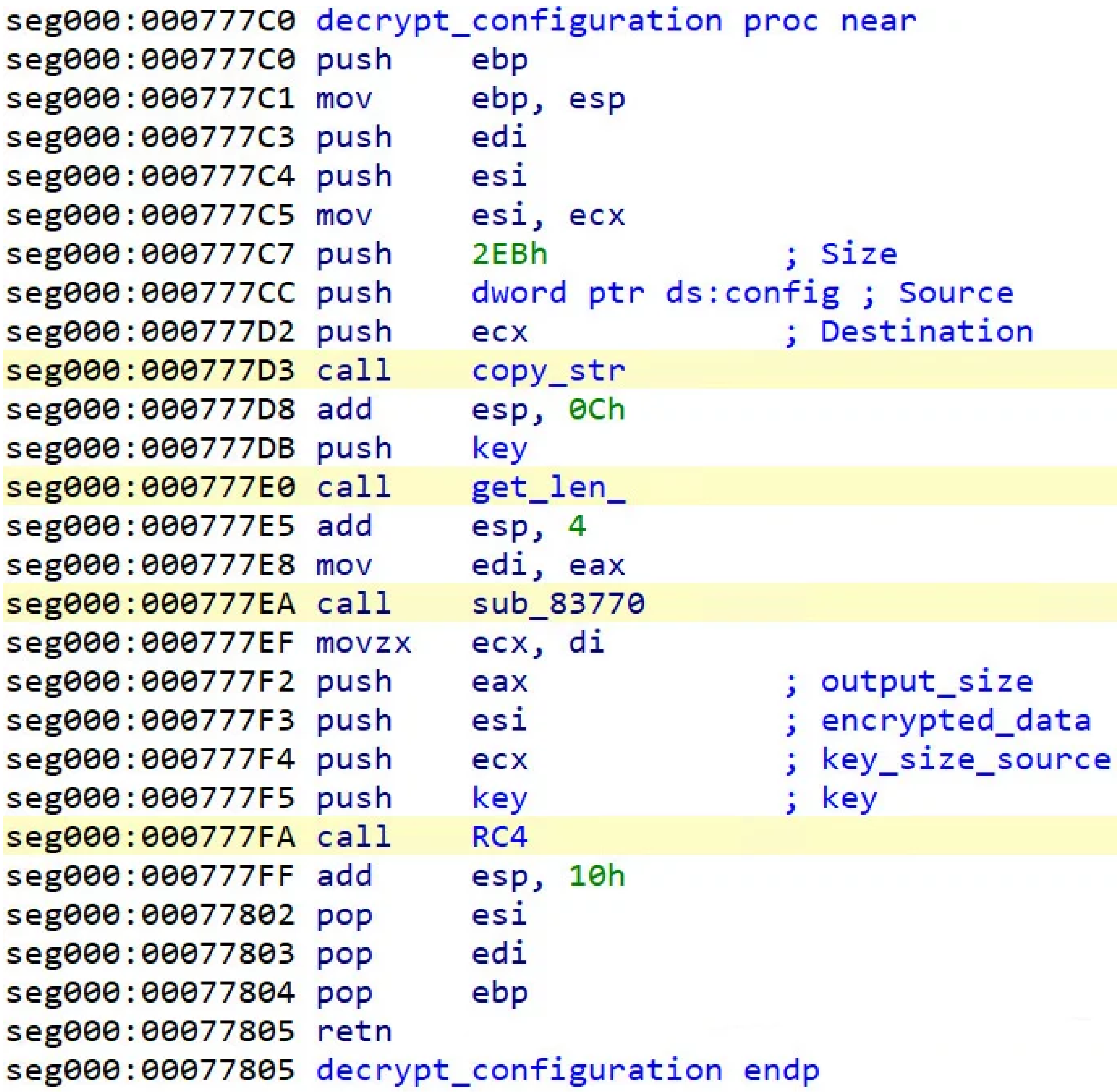
```
0:000> da esp+8
0014f778  "msiexec.exe"
```

Figure 4: Sphinx's process injection

Bot Configuration

The bot's encrypted configuration is embedded within the injected executable in *msiexec.exe*.

What makes it rather easy to decrypt is that both the configuration and its decryption key reside near each other in a hardcoded address location. The following function decrypts the configuration using the hardcoded addresses of each component:



```
seg000:000777C0 decrypt_configuration proc near
seg000:000777C0 push    ebp
seg000:000777C1 mov     ebp, esp
seg000:000777C3 push    edi
seg000:000777C4 push    esi
seg000:000777C5 mov     esi, ecx
seg000:000777C7 push    2EBh               ; Size
seg000:000777CC push    dword ptr ds:config ; Source
seg000:000777D2 push    ecx                ; Destination
seg000:000777D3 call    copy_str
seg000:000777D8 add     esp, 0Ch
seg000:000777DB push    key
seg000:000777E0 call    get_len_
seg000:000777E5 add     esp, 4
seg000:000777E8 mov     edi, eax
seg000:000777EA call    sub_83770
seg000:000777EF movzx   ecx, di
seg000:000777F2 push    eax                ; output_size
seg000:000777F3 push    esi                ; encrypted_data
seg000:000777F4 push    ecx                ; key_size_source
seg000:000777F5 push    key                ; key
seg000:000777FA call    RC4
seg000:000777FF add     esp, 10h
seg000:00077802 pop     esi
seg000:00077803 pop     edi
seg000:00077804 pop     ebp
seg000:00077805 retn
seg000:00077805 decrypt_configuration endp
```

Figure 5: Bot configuration decryption process

Let’s take a look at the main components of a January 2020 campaign configuration. It began with noting the malware’s variant ID by using Russian language words that translate to “2020 Upgrade” (obnovlenie2020).

Next, we see the attacker’s command-and-control (C&C) server domain list. Sphinx does not use a domain generation algorithm (DGA).

These elements can help defenders better protect networks against Sphinx infections by monitoring or blocking any communications to the listed C&C servers. The RC4 key itself is an important element to those looking to analyze the malware since it is the same key that Sphinx uses to encrypt and decrypt most of its data.

Please note that the key inside the configuration is different from the key used to decrypt the configuration itself.

In the following image, we can see an example of two different Sphinx configurations.

```
000ef73c obnovlenie2020.......obno
000ef755 va20.............https://
000ef76e dasifosafjasfhasf.com/gat
000ef787 e.php....................
000ef7a0 .......https://kasfajfsaf
000ef7b9 hasfhaf.com/gate.php.....
000ef7d2 ......................htt
000ef7eb ps://fdsjfjdsfjdsjfdjsfh.
000ef804 com/gate.php.............
000ef81d ............https://fdsjf
000ef836 jdsfjdsdsjajjs.com/gate.p
000ef84f hp.......................
000ef868 ..https://idisaudhasdhasd
000ef881 j.com/gate.php...........
000ef89a .................https://
000ef8b3 dsjdjsjdsadhasdas.com/gat
000ef8cc e.php....................
000ef8e5 .......https://dsdjfhdsuf
000ef8fe udhjas.com/gate.php......
000ef917
000ef930 .........................
000ef949 .........................
000ef962 .........................
000ef97b .........................
000ef994 .........................
000ef9ad .........................
000ef9c6
000ef9df .................03d5ae30
000ef9f8 a0bd934a23b6a7f0756aa504.
000efa11 .........................
```

```
0022f434 DLLobnova.......
0022f444 .....dllobnovano
0022f454 win.......https:
0022f464 //fdsjfjdsfjdsjf
0022f474 djsfh.com/gate.p
0022f484 hp..............
0022f494 ...........https
0022f4a4 ://fdsjfjdsfjdsd
0022f4b4 sjajjs.com/gate.
0022f4c4 php.............
0022f4d4 ............http
0022f4e4 s://idisaudhasdh
0022f4f4 asdj.com/gate.ph
0022f504 p...............
0022f514 .............htt
0022f524 ps://dsjdjsjdsad
0022f534 hasdas.com/gate.
0022f544 php.............
0022f554 ..............ht
0022f564 tps://dsdjfhdsuf
0022f574 udhjas.com/gate.
0022f584 php.............
0022f594 ...............h
0022f5a4 ttps://dsdjfhdsu
0022f5b4 fudhjas.info/gat
0022f5c4 e.php...........
0022f5d4 ................
0022f5e4 https://fdsjfjds
0022f5f4 fjdsdsjajjs.info
0022f604 /gate.php.......
0022f614 ................
0022f624 .https://idisaud
0022f634 hasdhasdj.info/g
0022f644 ate.php.........
0022f654 ................
0022f664 ................
0022f674 ................
0022f684 ................
0022f694 ................
0022f6a4 ................
0022f6b4 ................
0022f6c4 ................
0022f6d4 ................
0022f6e4 ....03d5ae30a0bd
0022f6f4 934a23b6a7f0756a
0022f704 a504............
0022f714 ................
```

Figure 6: Sphinx configuration excerpts from January 2020 Campaign

Taking into consideration the date they first appeared in the wild, similar C&C domains and the same RC4 key they contain, we can conclude that both configurations are related to the same campaign. On the left, a configuration fetched by an executable-type payload and on the right, one fetched by a DLL-type payload — both are from a January 2020 campaign.

Sphinx configurations are modified as campaigns are launched, changing the C&C addresses and the RC4 keys. In the following image, we can see a newer configuration fetched from an April 2020 campaign.

```
miguel...............17/04......
..........https://lgepubbf.icu/w
p-config.php....................
...........https://ajvwdjtebb.pw
/wp-config.php..................
................................
................................
................................
................................
................................
................................
................................
................................
................................
................................
................................
................................
................................
................................
................................
................................
....................41997b4a729e
1a0175208305170752dd............
```

Figure 7: A different Sphinx configuration excerpt from an April 2020 campaign

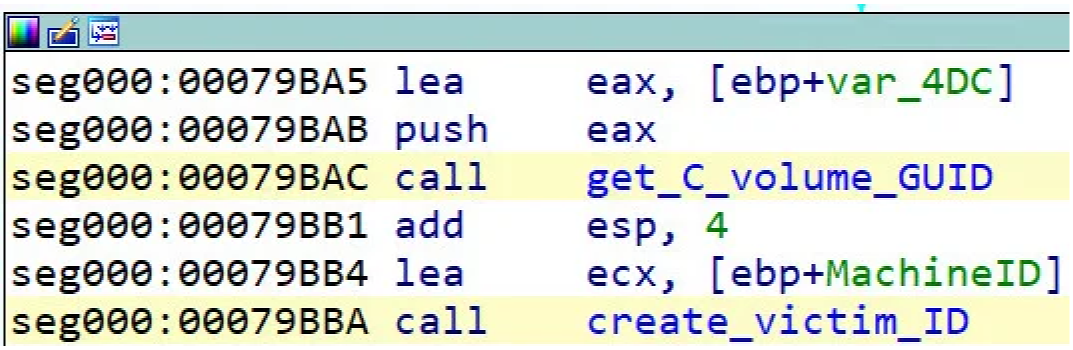
Please note the main differences from the January 2020 configurations: a different RC4 key, a smaller and different set of C&C domains, a changed variant ID and the precise date of publish were added.

Bot Identification

Once infected by Sphinx, every device sends information home and is defined in the botnet by a bot ID to ensure control and updates through the attacker’s server. To do that, Sphinx uses an algorithm that includes the following elements from the infected device:

- Volume C GUID
- Computer Name
- Windows Version
- Windows Install Date
- Digital Product ID

We can see the generated string when we run Sphinx dynamically as well:



```
seg000:00079BA5 lea     eax, [ebp+var_4DC]
seg000:00079BAB push    eax
seg000:00079BAC call    get_C_volume_GUID
seg000:00079BB1 add     esp, 4
seg000:00079BB4 lea     ecx, [ebp+MachineID]
seg000:00079BBA call    create_victim_ID
```

Figure 8: Sphinx creating Bot ID string

After creating the bot ID, it's encrypted with an RC4 stream cipher using the key derived from the bot's configuration and then stored in the Registry with other binary data.

For example, a key created for storing this information:

HKCU\Software\Microsoft\bmqhcn\gwehhxf

The name of the key depends on the variant of the malware and is produced by encoding some constants.

Looking at the function's output before the result is encrypted reveals Sphinx's bot ID layout:

- [VOLUME_C_GUID][COMPUTER_NAME][2EBFF1F4][0ADE2A62]
- [VOLUME_C_GUID] – Bot's volume C GUID
- [COMPUTER_NAME] – Bot's computer name
- [2EBFF1F4] – A hash of the operating system version.
- [0ADE2A62] – A hash of *InstallDate* and *DigitalProductID* registry values.
- Both hashes mentioned above are computed using a Sphinx internal hash function.

Sphinx's Naming Algorithms

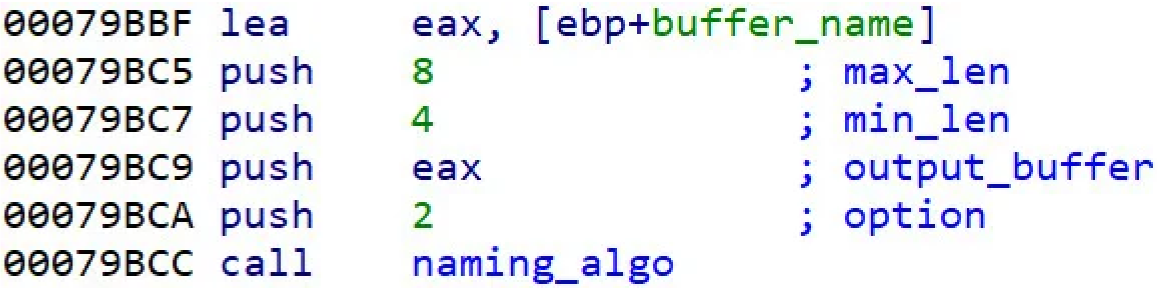
Malware codes often use a naming algorithm to create different names for files and resources on each infected device. They do this to evade static detection that might search for a certain file name as an indicator of compromise (IoC).

In Sphinx's case, one naming algorithm is used to create files and resource names and a different one is used to create a unique mutex object name.

File/Resource Name Generator

Beginning with the algorithm used to create file and resource names, to create what would appear to be random names, Sphinx uses a pseudo-random number generator (PRNG) named MT19937 (also known as the Mersenne Twister). Let's look at how Zeus Sphinx implements this PRNG to create names for its resources.

The Sphinx naming algorithm function takes four parameters to create its names: maximum length, minimum length, output buffer pointer and a binary option to upper or not the first character. As shown in the next example, these parameters are hardcoded, which can help write more regular expressions (RegEx) for detecting such names.

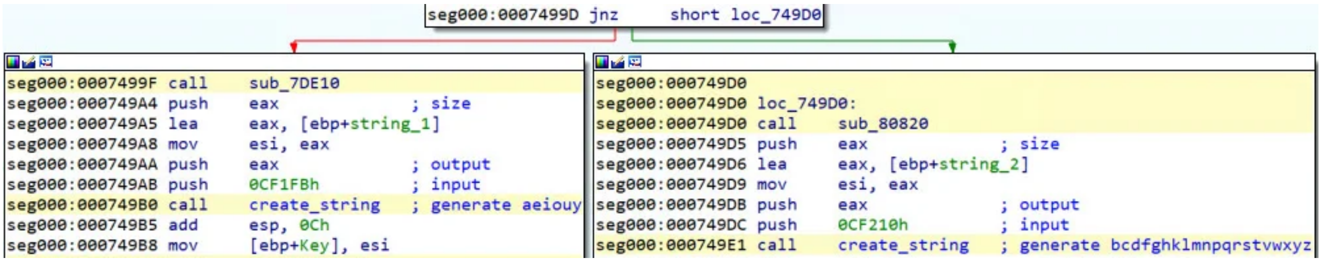


```
00079BBF lea     eax, [ebp+buffer_name]
00079BC5 push    8                  ; max_len
00079BC7 push    4                  ; min_len
00079BC9 push    eax                ; output_buffer
00079BCA push    2                  ; option
00079BCC call    naming_algo
```

Figure 9: Sphinx's naming algorithm

Let's look at the *naming_algo* function:

- Sphinx starts the process by decoding two hardcoded strings, which amount to 25 of the 26 English language characters:
    - Aeiouy
    - bcdfghklmnpqrstvwxz
- It uses randomization for choosing the output (name) size and loops through additional steps to build it.
- It randomly selects one of the two initial strings.
- It randomly chooses one character from the selected string.
- It appends the character to what's going to eventually compose the generated name.
- If the name has not yet met the selected length requirement, it loops back and repeats the process.



```
seg000:0007499D jnz     short loc_749D0
```

```
seg000:0007499F call    sub_7DE10
seg000:000749A4 push    eax             ; size
seg000:000749A5 lea     eax, [ebp+string_1]
seg000:000749A8 mov     esi, eax
seg000:000749AA push    eax             ; output
seg000:000749AB push    0CF1FBh         ; input
seg000:000749B0 call    create_string   ; generate aeiouy
seg000:000749B5 add     esp, 0Ch
seg000:000749B8 mov     [ebp+Key], esi
```

```
seg000:000749D0
seg000:000749D0 loc_749D0:
seg000:000749D0 call    sub_80820
seg000:000749D5 push    eax             ; size
seg000:000749D6 lea     eax, [ebp+string_2]
seg000:000749D9 mov     esi, eax
seg000:000749DB push    eax             ; output
seg000:000749DC push    0CF210h         ; input
seg000:000749E1 call    create_string   ; generate bcdfghklmnpqrstvwxyz
```

Figure 10: Choosing between the "aeiouy" or "bcdfghklmnpqrstvwxz" strings

Mutex Name Generator

Like other malware, Sphinx generates mutex names upon execution. Mutex names are often searched by security tools and researchers as a way to gather IoCs. Therefore, it can be a better way for malware to hide on infected devices if its mutex name is harder to find. The use of a unique mutex name also helps prevent the malware from infecting the same machine twice.

In Sphinx's case, the mutex name is always a unique string created per machine, and the algorithm used to create it is relatively complicated.

To start, Sphinx uses two system data components for building its mutex names:

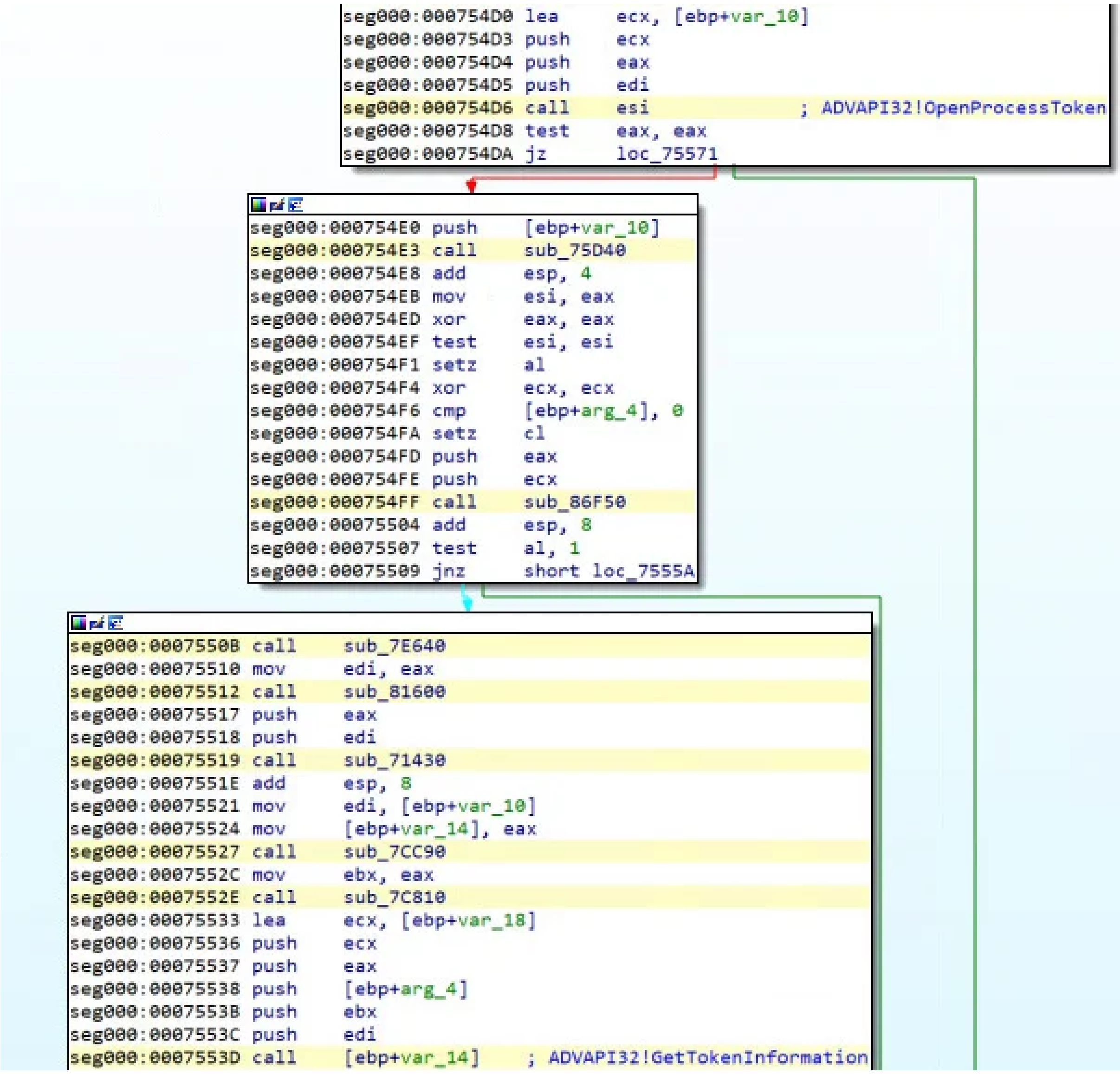
- The device's volume C globally unique identifier (GUID)
- The current user's security identifier (SID)

To fetch the first value, it uses the Windows function *GetVolumeNameForVolumeMountPointW*.

```
seg000:00074DDB push    104h
seg000:00074DE0 push    edi
seg000:00074DE1 push    ebx
seg000:00074DE2 call    eax             ; GetVolumeNameForVolumeMountPointW
```

Figure 11: Sphinx composing its unique mutex name

To fetch the second value, it uses two functions: *OpenProcessToken* and *GetTokenInformation*.



```
seg000:000754D0 lea     ecx, [ebp+var_10]
seg000:000754D3 push    ecx
seg000:000754D4 push    eax
seg000:000754D5 push    edi
seg000:000754D6 call    esi             ; ADVAPI32!OpenProcessToken
seg000:000754D8 test    eax, eax
seg000:000754DA jz      loc_75571
```

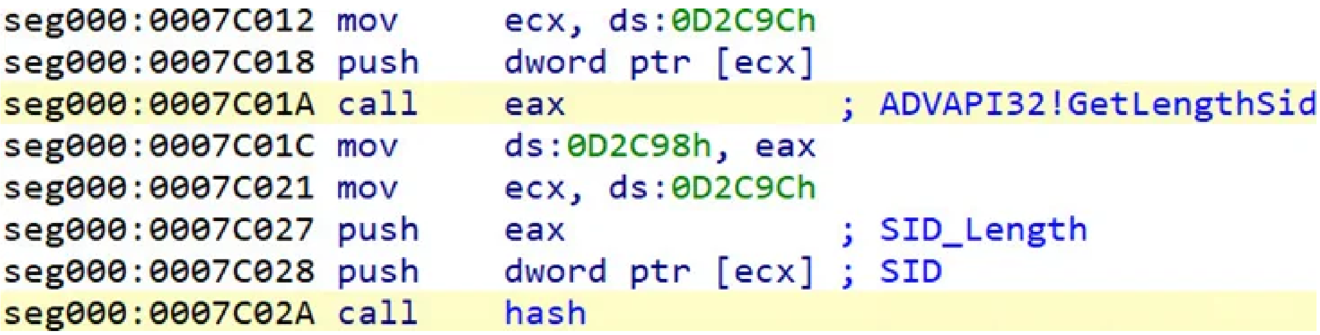
```
seg000:000754E0 push    [ebp+var_10]
seg000:000754E3 call    sub_75D40
seg000:000754E8 add     esp, 4
seg000:000754EB mov     esi, eax
seg000:000754ED xor     eax, eax
seg000:000754EF test    esi, esi
seg000:000754F1 setz    al
seg000:000754F4 xor     ecx, ecx
seg000:000754F6 cmp     [ebp+arg_4], 0
seg000:000754FA setz    cl
seg000:000754FD push    eax
seg000:000754FE push    ecx
seg000:000754FF call    sub_86F50
seg000:00075504 add     esp, 8
seg000:00075507 test    al, 1
seg000:00075509 jnz     short loc_7555A
```

```
seg000:0007550B call    sub_7E640
seg000:00075510 mov     edi, eax
seg000:00075512 call    sub_81600
seg000:00075517 push    eax
seg000:00075518 push    edi
seg000:00075519 call    sub_71430
seg000:0007551E add     esp, 8
seg000:00075521 mov     edi, [ebp+var_10]
seg000:00075524 mov     [ebp+var_14], eax
seg000:00075527 call    sub_7CC90
seg000:0007552C mov     ebx, eax
seg000:0007552E call    sub_7C810
seg000:00075533 lea     ecx, [ebp+var_18]
seg000:00075536 push    ecx
seg000:00075537 push    eax
seg000:00075538 push    [ebp+arg_4]
seg000:0007553B push    ebx
seg000:0007553C push    edi
seg000:0007553D call    [ebp+var_14]    ; ADVAPI32!GetTokenInformation
```

Figure 12: Sphinx composing its unique mutex name

Next, to generate a unique name, Sphinx takes the following steps:

1. It creates a hash of the infected device’s SID value that it obtained earlier on.

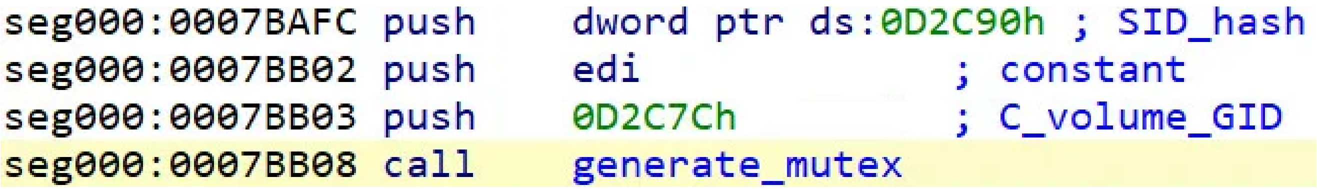


```
seg000:0007C012 mov     ecx, ds:0D2C9Ch
seg000:0007C018 push    dword ptr [ecx]
seg000:0007C01A call    eax             ; ADVAPI32!GetLengthSid
seg000:0007C01C mov     ds:0D2C98h, eax
seg000:0007C021 mov     ecx, ds:0D2C9Ch
seg000:0007C027 push    eax             ; SID_Length
seg000:0007C028 push    dword ptr [ecx] ; SID
seg000:0007C02A call    hash
```

Figure 13: Sphinx composing its unique mutex name

1. It uses the GUID to encode the SID’s hash.

This function is called seven times, with varying constants being used. Then, some of the names are randomly selected to become mutex names.

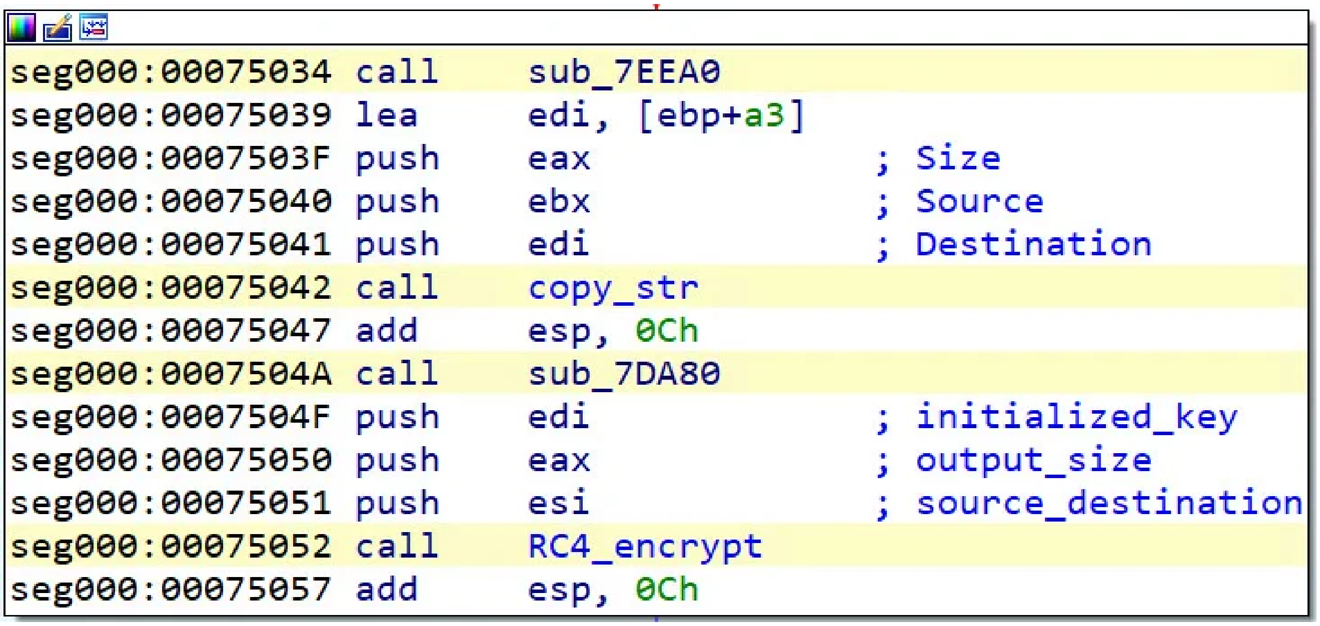


```
seg000:0007BAFC push    dword ptr ds:0D2C90h ; SID_hash
seg000:0007BB02 push    edi                 ; constant
seg000:0007BB03 push    0D2C7Ch             ; C_volume_GID
seg000:0007BB08 call    generate_mutex
```

Figure 14: Sphinx generates seven different mutex names on each device

Technically, there could be seven different mutex names created on each device, which Sphinx checks for to ensure that the device is not already running the malware.

1. Next, using the key derived from the bot's configuration, the mutex is encrypted with an RC4 cipher.

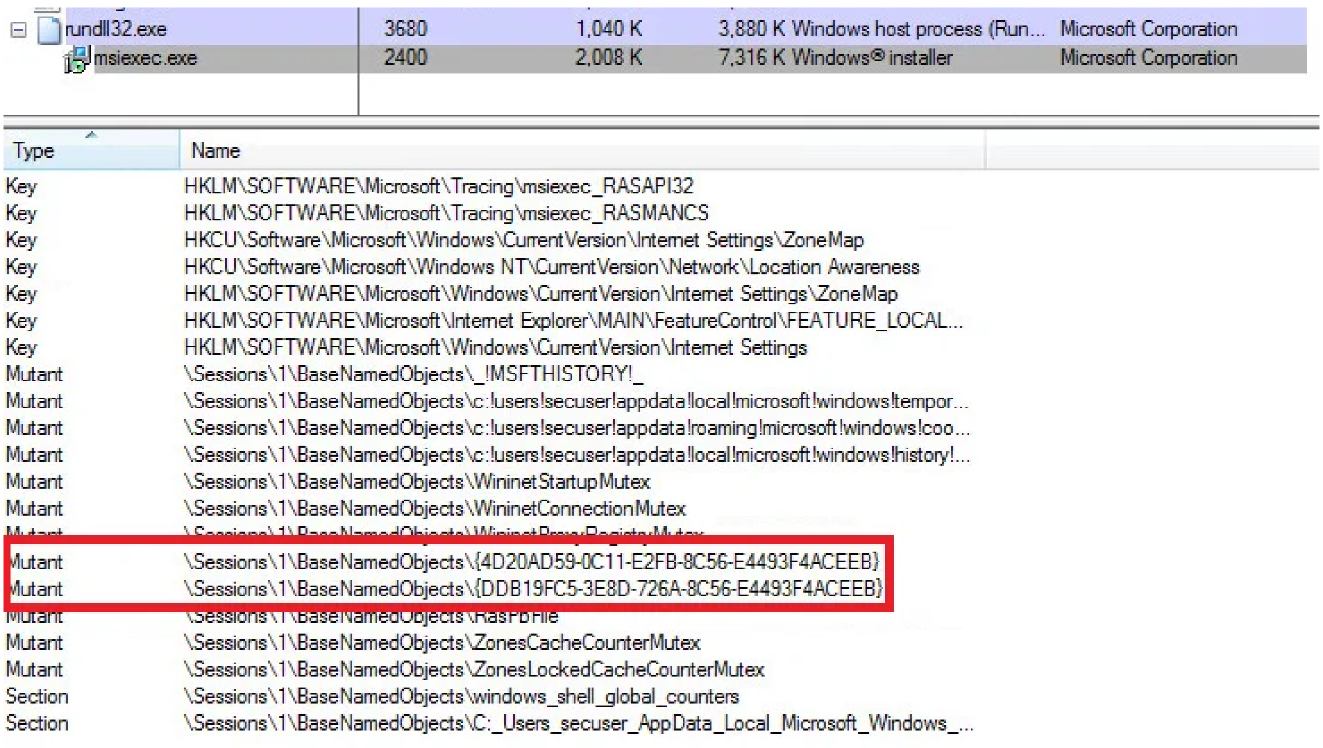


```
seg000:00075034 call    sub_7EEA0
seg000:00075039 lea     edi, [ebp+a3]
seg000:0007503F push    eax             ; Size
seg000:00075040 push    ebx             ; Source
seg000:00075041 push    edi             ; Destination
seg000:00075042 call    copy_str
seg000:00075047 add     esp, 0Ch
seg000:0007504A call    sub_7DA80
seg000:0007504F push    edi             ; initialized_key
seg000:00075050 push    eax             ; output_size
seg000:00075051 push    esi             ; source_destination
seg000:00075052 call    RC4_encrypt
seg000:00075057 add     esp, 0Ch
```

Figure 15: Sphinx encrypts mutex name

1. To make its mutex names blend in with other system elements, it calls on the function *ole32!StringFromGUID2*, making the names look like GUIDs.

Below is an example of two mutex names created within *msiexec.exe*:



rundll32.exe	3680	1,040 K	3,880 K	Windows host process (Run...	Microsoft Corporation
msiexec.exe	2400	2,008 K	7,316 K	Windows® installer	Microsoft Corporation

Type	Name
Key	HKLM\SOFTWARE\Microsoft\Tracing\msiexec_RASAPI32
Key	HKLM\SOFTWARE\Microsoft\Tracing\msiexec_RASMANCS
Key	HKCU\Software\Microsoft\Windows\CurrentVersion\Internet Settings\ZoneMap
Key	HKCU\Software\Microsoft\Windows NT\CurrentVersion\Network\Location Awareness
Key	HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Internet Settings\ZoneMap
Key	HKLM\SOFTWARE\Microsoft\Internet Explorer\MAIN\FeatureControl\FEATURE_LOCAL...
Key	HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Internet Settings
Mutant	\Sessions\1\BaseNamedObjects_!MSFTHISTORY!_
Mutant	\Sessions\1\BaseNamedObjects\c:!users!secuser!appdata!local!microsoft!windows!tempor...
Mutant	\Sessions\1\BaseNamedObjects\c:!users!secuser!appdata!roaming!microsoft!windows!coo...
Mutant	\Sessions\1\BaseNamedObjects\c:!users!secuser!appdata!local!microsoft!windows!history!...
Mutant	\Sessions\1\BaseNamedObjects\WininetStartupMutex
Mutant	\Sessions\1\BaseNamedObjects\WininetConnectionMutex
Mutant	\Sessions\1\BaseNamedObjects\WininetProxyRegistryMutex
Mutant	\Sessions\1\BaseNamedObjects\{4D20AD59-0C11-E2FB-8C56-E4493F4ACEEB}
Mutant	\Sessions\1\BaseNamedObjects\{DDB19FC5-3E8D-726A-8C56-E4493F4ACEEB}
Mutant	\Sessions\1\BaseNamedObjects\RasPbFile
Mutant	\Sessions\1\BaseNamedObjects\ZonesCacheCounterMutex
Mutant	\Sessions\1\BaseNamedObjects\ZonesLockedCacheCounterMutex
Section	\Sessions\1\BaseNamedObjects\windows_shell_global_counters
Section	\Sessions\1\BaseNamedObjects\C:_Users_secuser_AppData_Local_Microsoft_Windows_...

Figure 16: Examples of Sphinx mutex names generated through its process

Sphinx Is Back in Business

The Sphinx Trojan emerged in 2015, at which point its main focus was banks in North America. Over the years, different operators of this malware launched it into campaigns in other parts of the world, such as the U.K., then Brazil, then Canada and Australia. Most recently, Sphinx was implemented in infection campaigns targeting users in Japan.

While Sphinx has been an on-and-off type of operation over the years, it appears it is now on-again, with version updates and new infection campaigns that are back to targeting North American banks.

While less common in the wild than Trojans like TrickBot, for example, Sphinx's underlying Zeus DNA has been an undying enabler of online banking fraud. Financial institutions must reckon with its return and spread to new victims amid the current pandemic.

Sphinx is just one more threat we regularly cover. To learn more about emerging threats and campaigns, please join us on X-Force Exchange. Our research team also regularly releases blogs on Security Intelligence to keep you up to date on what we see in the wild.

Nir Shwarts
Malware Research-Reverse Engineering, IBM Security

Nir Shwarts is a contributor for SecurityIntelligence.

Cost of a Data Breach Report 2022

Prevent, detect and respond to cybersecurity threats faster

Get the report →