Neshta Infected Zoom Installers

May 2020

Cyberint

TABLE OF CONTENTS

SUMMARY

Given the surge in popularity of the Zoom Video Conferencing platform, for both businesses and individuals to stay connected during these testing times, Cyberint Research have observed, since 30 March 2020, instances of the Zoom Installer (for Windows) as being infected by the file infector threat known as 'Neshta'.

First detected in 2003, Neshta has previously been associated with BlackPOS, a threat used to scrape payment card data from point-of-sale (POS) systems, and was prevalent during attacks against the consumer goods, energy, finance and manufacturing industries in 2018. Once the Neshta payload is executed on a victim machine, it searches for other executable files, across both local and network drives, and infects them by prepending it's own malicious code. Having infected a victim machine, the threat then attempts to collect system information and exfiltrate this to a remote server using HTTP POST requests.

Whilst the original infection vector for these recent discoveries hasn't been confirmed, it is feasible that threat actors would seek to distribute weaponized versions of popular software, such as the Zoom video conferencing application, to gain access to new victims. Typically, Neshta campaigns utilize email lures to encourage victims to execute the payload from either an attachment or a malicious website, given this, and the fact that the threat infects multiple executables on victim machines by design, it is highly unlikely that this particular threat originated from, or was distributed via, the legitimate Zoom Video Communications website.

KEY FINDINGS

- Zoom Installer (for Windows) executables discovered as infected by the Neshta file infector although there is no evidence to suggest these infected files originated from, or were distributed via, the legitimate Zoom Video Communications website
- Aside from infected files increasing in size by some 40.5 kilobytes, the executable should appear as before and will execute, albeit with Neshta as the parent process.
- Neshta campaigns typically commence with the delivery of email lures containing malicious attachments or links
- Indicators of compromise include hard-coded directory names, mutex and registry modifications.

TECHNICAL ANALYSIS

First observed in 2003, Neshta (a transliteration of the Belarusian word for "something") is a file infector developed in Delphi that, based on strings visible within the binary (Figure 1), was created by a Belarusian threat actor.

```
Delphi-the best. F[redacted]k off all the rest. Neshta 1.0 Made in Belarus.
 :)
...
! Best regards 2 Tommy Salo. [Nov-2005] yours [Dziadulja Apanas]
```

Figure 1 - Threat actor message/identifiers

As observed in multiple malicious samples since 30 March 2020, this recent Neshta threat appears to include a legitimate Zoom for Windows installer that has been infected with the malicious code.

INITIAL EXECUTION

Once executed, the infected executable restores the original legitimate executable to the Windows temporary directory, in this case the Zoom installer, as well as dropping a separate Neshta malicious executable. This method allows the legitimate file to execute 'as normal' whilst allowing Neshta to spread and infect other executable files and, ultimately, act on its malicious objective.

Having obtained details of infected executable using the `GetModuleFileNameA` and `CreateFileA` functions, the `ReadFile` function is used to read the first 41,472 bytes (`0xa200`) of the file into a buffer, `0x40a698` in this case (Figure 2).

```
GetModuleFileNameA (in: hModule=0x0, lpFilename=0x19fe2b, nSize=0x105 | out: lpFilename="C:\\
Users\\<USERNAME>\\Desktop\\sample.exe" (normalized: "c:\\users\\<USERNAME>\\desktop\\
sample.exe")) returned 0x24;
CreateFileA (lpFileName="C:\\Users\\<USERNAME>\\Desktop\\sample.exe" (normalized: "c:\\users\\
<USERNAME>\\desktop\\sample.exe"), dwDesiredAccess=0x80000000, dwShareMode=0x1,
lpSecurityAttributes=0x0, dwCreationDisposition=0x3, dwFlagsAndAttributes=0x80,
hTemplateFile=0x0) returned 0x170;
ReadFile (in: hFile=0x170, lpBuffer=0x40a698, nNumberOfBytesToRead=0xa200,
lpNumberOfBytesRead=0x19ff08, lpOverlapped=0x0 | out: lpBuffer=0x40a698*, lpNumberOfBytesRead=
0x19ff08*=0xa200, lpOverlapped=0x0) returned 1;
```

Figure 2 - Extraction of the Neshta malicious code from the infected executable

These bytes contain the Neshta malicious code and are referenced after the original legitimate executable file has been restored.

ORIGINAL LEGITIMATE EXECUTABLE RESTORATION

To allow restoration of the original legitimate executable, a temporary directory is created by first determining the location of the Windows temporary directory, using the `GetTempPathA` function, and then determining if the Neshta directory `\3582-490` already exists. If the directory is not already present, it will be created via the `CreateDirectoryA` function (Figure 3).

```
GetTempPathA (in: nBufferLength=0x105, lpBuffer=0x195bd7 | out: lpBuffer="C:\\Users\\<USERNAME
>\\AppData\\Local\\Temp\\") returned 0x25;
GetFileAttributesA (lpFileName="C:\\Users\\<USERNAME>\\AppData\\Local\\Temp\\3582-490" (
normalized: "c:\\users\\<USERNAME>\\appdata\\local\\temp\\3582-490")) returned 0xffffffff;
```

Figure 3 - Creation of a temporary directory for the restored original legitimate executable

Note: The `%TEMP%\3582-490` directory appears to be hard-coded within Neshta and as such it's presence serves as a useful indicator of compromise (IOC).

Once the directory structure has been created, a new file is created using the same name as the infected executable (Figure 4), this will be used for the restoration of the original legitimate executable.

```
GetModuleFileNameA (in: hModule=0x0, lpFilename=0x195bbf, nSize=0x105 | out: lpFilename="C:\\
Users\\<USERNAME>\\Desktop\\sample.exe" (normalized: "c:\\users\\<USERNAME>\\desktop\\
sample.exe")) returned 0x24;
CreateFileA (lpFileName="C:\\Users\\<USERNAME>\\AppData\\Local\\Temp\\3582-490\\sample.exe" (
normalized: "c:\\users\\<USERNAME>\\appdata\\local\\temp\\3582-490\\sample.exe"),
dwDesiredAccess=0x40000000, dwShareMode=0x3, lpSecurityAttributes=0x0,
dwCreationDisposition=0x1, dwFlagsAndAttributes=0x80, hTemplateFile=0x0) returned 0x174;
```

Figure 4 - Creation of the file to be used for the restored original legitimate executable

During the infection process (detailed later in this report) Neshta copies the first 41,472 bytes (`0xa200`) of the original legitimate executable to the end of the infected executable before overwriting them with the Neshta malicious code. As such, the restoration process determines the location of the original legitimate code by determining the infected file size, using `GetFileSize`, and then setting a pointer to '*file size minus 41,472 bytes*' (figure 5).

```
GetFileSize (in: hFile=0x170, lpFileSizeHigh=0x0 | out: lpFileSizeHigh=0x0) returned 0x4cd58;
SetFilePointer (in: hFile=0x170, lDistanceToMove=273240, lpDistanceToMoveHigh=0x0,
dwMoveMethod=0x0 | out: lpDistanceToMoveHigh=0x0) returned 0x42b58;
```

Figure 5 - Determining the location of the original legitimate code

Subsequently, 41,472 bytes (`0xa200`) of the original legitimate code are read from the end of the infected file and written to the file in the temporary directory (Figure 6), effectively restoring the first part of the original legitimate executable file.

```
ReadFile (in: hFile=0x170, lpBuffer=0x195d40, nNumberOfBytesToRead=0xa200,
lpNumberOfBytesRead=0x195cb8, lpOverlapped=0x0 | out: lpBuffer=0x195d40*, lpNumberOfBytesRead=
0x195cb8*=0xa200, lpOverlapped=0x0) returned 1;
WriteFile (in: hFile=0x174, lpBuffer=0x195d40*, nNumberOfBytesToWrite=0xa200,
lpNumberOfBytesWritten=0x195cb8, lpOverlapped=0x0 | out: lpBuffer=0x195d40*,
lpNumberOfBytesWritten=0x195cb8*=0xa200, lpOverlapped=0x0) returned 1;
```

Figure 6 - Initial restoration of the original legitimate executable (first 41,472 bytes)

To complete the restoration of the original legitimate executable, an offset of 41,472 bytes (`0xa200`) is set using `SetFilePointer` at the beginning of the infected executable, skipping the Neshta malicious code, and the remainder of original legitimate executable is read from this pointer to the end of the infected file *minus* 41,472 bytes (as the end of the infected executable contains the already restored first bytes of the original legitimate executable). This data is then written, after

the already restored first bytes, to the temporary directory and completes the restoration of the legitimate file (Figure 7).

```
SetFilePointer (in: hFile=0x170, lDistanceToMove=41472, lpDistanceToMoveHigh=0x0,
dwMoveMethod=0x0 | out: lpDistanceToMoveHigh=0x0) returned 0xa200;
GetFileSize (in: hFile=0x170, lpFileSizeHigh=0x0 | out: lpFileSizeHigh=0x0) returned 0x4cd58;

ReadFile (in: hFile=0x170, lpBuffer=0x1e402fc, nNumberOfBytesToRead=0x38958,
lpNumberOfBytesRead=0x195ca4, lpOverlapped=0x0 | out: lpBuffer=0x1e402fc*,
lpNumberOfBytesRead=0x195ca4*=0x38958, lpOverlapped=0x0) returned 1;
WriteFile (in: hFile=0x174, lpBuffer=0x1e402fc*, nNumberOfBytesToWrite=0x38958,
lpNumberOfBytesWritten=0x195ca4, lpOverlapped=0x0 | out: lpBuffer=0x1e402fc*,
lpNumberOfBytesWritten=0x195ca4*=0x38958, lpOverlapped=0x0) returned 1;
```

Figure 7 - Final restoration of the original legitimate executable

DECOY EXECUTION

To ensure that the victim is unaware of any untoward activity, the original legitimate executable is launched using the `ShellExecuteA` function and should proceed as normal (Figure 8).

```
ShellExecuteA (hwnd=0x400000, lpOperation="open", lpFile="C:\\Users\\<USERNAME>\\AppData\\
Local\\Temp\\3582-490\\sample.exe", lpParameters="", lpDirectory=0x0, nShowCmd=1) returned
0x2a
```

Figure 8 - Launching the original legitimate executable

In the infected Zoom installers observed, those executed with the corresponding installation files functioned as normal and communicate with the legitimate Zoom Video Communications website (Figure 9).

Figure 9 - Zoom installer success

Those with missing installation files fail with an installation error, albeit this is likely expected behavior rather than therefore the Neshta infection (Figure 10).

Figure 10 - Zoom installer failure

NESHTA PAYLOAD

Having restored the original legitimate executable, Neshta will create a new executable file named svchost.com` that contains the Neshta malicious code and is referenced by the malware's persistence method. In preparation for this process, Neshta obtains the path of the Windows directory using the GetWindowsDirectoryA` function and then, using the `GetFileAttributesA` function attempts to determine if the Neshta dropped payload already exists (Figure 11), if so, the existing file would be deleted.

```
GetWindowsDirectoryA (in: lpBuffer=0x19fe13, uSize=0x105 | out: lpBuffer="C:\\Windows")
returned 0xa;
GetFileAttributesA (lpFileName="C:\\Windows\\svchost.com" (normalized: "c:\\windows\\
svchost.com")) returned 0xffffffff;
```

Figure 11 - Determining if the Neshta dropped payload already exists

The return value of `0xffffffff`, as seen above, indicates that an existing `svchost.com` file was not found. Using the previously buffered data, referenced as `0x40a698` and containing the first 41,472 bytes of the infected executable, the Neshta malicious payload file is created as `svchost.com` and the buffer content written to it (Figure 12).

```
CreateFileA (lpFileName="C:\\Windows\\svchost.com" (normalized: "c:\\windows\\svchost.com"),
dwDesiredAccess=0x40000000, dwShareMode=0x0, lpSecurityAttributes=0x0,
dwCreationDisposition=0x4, dwFlagsAndAttributes=0x80, hTemplateFile=0x0) returned 0x300;
WriteFile (in: hFile=0x300, lpBuffer=0x40a698*, nNumberOfBytesToWrite=0xa200,
lpNumberOfBytesWritten=0x19fef0, lpOverlapped=0x0 | out: lpBuffer=0x40a698*,
lpNumberOfBytesWritten=0x19fef0*=0xa200, lpOverlapped=0x0) returned 1;
```

Figure 12 - Creation of the Neshta malicious executable

PERSISTENCE

After the Neshta malicious executable has been created, the 'shell open command' registry key for executable files is modified from its default value, "%1" %*, to C:\Windows\svchost.com "%1" %* that results in any executable file, when launched, being passed as a parameter to Neshta (Figure 13).

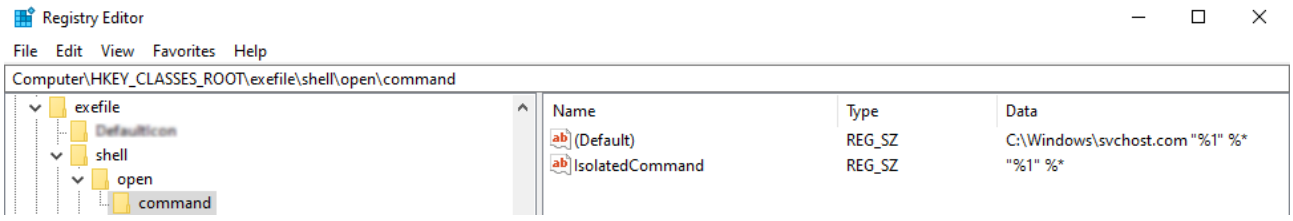

Figure 13 - Modification of the default executable file shell registry key

In addition to ensuring Neshta executes as the parent process of any legitimate executable when launched by the victim, this method provides persistence across reboots and may evade detection of typical persistence keys such as (HKCU|HKLM)\Software\Microsoft\Windows\CurrentVersion\Run.

MUTUAL EXCLUSION OBJECT CREATION

To prevent multiple instances of the malware from executing simultaneously, a hard-coded mutual exclusion object (mutex) is created (Figure 14) that includes details of the persistence registry key as well as another seemingly Belarusian transliteration, "PolesskayaGlush" (potentially "*woodland*").

```
CreateMutexA (lpMutexAttributes=0x0, bInitialOwner=0, lpName="MutexPolesskayaGlush*.*\x90
svchost.com\x90exefile\\shell\\open\\command\x8bÀ \"%1\" %*\x9c\x91@") returned 0x0
```

Figure 14 - Mutex creation

DISK/FILE DISCOVERY

The discovery phase sees Neshta determine which logical disks are available on the compromised host, determine their type, such as removable, fixed or network, and then search for files across these drives and in each directory (Figure 15).

```
GetLogicalDriveStringsA (in: nBufferLength=0x97, lpBuffer=0x19fe8d | out: lpBuffer="C:\\")
returned 0x4;
GetDriveTypeA (lpRootPathName="C:\\") returned 0x3;
FindFirstFileA (in: lpFileName="C:\\*.*", lpFindFileData=0x19fd88 | out: lpFindFileData=0x
19fd88*(dwFileAttributes=0x16, ftCreationTime.dwLowDateTime=0xbaec25,
ftCreationTime.dwHighDateTime=0x1d112e4, ftLastAccessTime.dwLowDateTime=0x98a4cabc,
ftLastAccessTime.dwHighDateTime=0x1d31d78, ftLastWriteTime.dwLowDateTime=0x98a4cabc,
ftLastWriteTime.dwHighDateTime=0x1d31d78, nFileSizeHigh=0x0, nFileSizeLow=0x0, dwReserved0=
0x0, dwReserved1=0x2080060, cFileName="$Recycle.Bin", cAlternateFileName="")) returned
0x45fd40;
CharLowerBuffA (in: lpsz="*", cchLength=0x1 | out: lpsz="*") returned 0x1;
CharLowerBuffA (in: lpsz="$Recycle.Bin", cchLength=0xc | out: lpsz="$recycle.bin")
returned 0xc;
FindNextFileA (in: hFindFile=0x45fd40, lpFindFileData=0x19fd88 | out: lpFindFileData=0x
19fd88*(dwFileAttributes=0x16, ftCreationTime.dwLowDateTime=0xb203fd16,
ftCreationTime.dwHighDateTime=0x1d30bd2, ftLastAccessTime.dwLowDateTime=0xb260f943,
ftLastAccessTime.dwHighDateTime=0x1d30bd2, ftLastWriteTime.dwLowDateTime=0xb260f943,
ftLastWriteTime.dwHighDateTime=0x1d30bd2, nFileSizeHigh=0x0, nFileSizeLow=0x0, dwReserved0=
0x520024, dwReserved1=0x630065, cFileName="Boot", cAlternateFileName="")) returned 1;
```

Figure 15 - Disk and file discovery

FILE INFECTION

Having found an executable file in the discovery phase, the Neshta infection process goes through several steps to determine if it should be infected. In short, the target file must be a valid executable that is not located in the 'Program Files', 'Temporary' or 'Windows' directories and have a file size between 41,472 bytes and 10,000,000 bytes.

- Using the `GetShortPathNameA` function (Figure 16), the short filename and path (conforming to the 8.3 convention) of a target file is retrieved.

```
GetShortPathNameA (in: lpszLongPath="C:\\Users\\<USERNAME>\\AppData\\Roaming\\Zoom\\bin\\
Installer.exe", lpszShortPath=0x195a3c, cchBuffer=0x104 | out: lpszShortPath="C:\\Users\\
Nd9E1FYi\\AppData\\Roaming\\Zoom\\bin\\INSTAL~1.EXE") returned 0x37;
```

Figure 16 - Get short filename/path of the infection target executable

- Using the `GetWindowsDirectoryA` and `GetWindowsDirectoryA` functions (Figure 17), the paths of both the Windows (`%WINDIR%`) and Temporary (`%TEMP%`) directories are determined.

```
GetWindowsDirectoryA (in: lpBuffer=0x195a3f, uSize=0x105 | out: lpBuffer="C:\\Windows")
returned 0xa;
GetTempPathA (in: nBufferLength=0x105, lpBuffer=0x195a3f | out: lpBuffer="C:\\Users\\<USERNAME>
\\AppData\\Local\\Temp\\") returned 0x25;
```

Figure 17 - Windows and Temporary directory enumeration

Any executable discovered in these locations, along with those located in the Program Files (`%PROGRAMFILES%`) directory, are excluded from the infection process, presumably to maintain system stability and avoid detection.

- Having obtained a target file's size using the `FindFirstFileA` function (Figure 18), Neshta can check to determine if it is within the acceptable range of greater than, or equal to, 41,472 bytes and less than 10,000,000 bytes. The lower bound is presumably set to this byte value as this is the size of the malicious code itself, the upper bound is likely set to prevent performance or stability issues when making changes to large files.

```
FindFirstFileA (in: lpFileName="C:\\Users\\<USERNAME>\\AppData\\Roaming\\Zoom\\bin\\
INSTAL~1.EXE", lpFindFileData=0x195a08 | out: lpFindFileData=0x195a08*(dwFileAttributes=0x20,
ftCreationTime.dwLowDateTime=0x3eb968a8, ftCreationTime.dwHighDateTime=0x1d61894,
ftLastAccessTime.dwLowDateTime=0x3eb968a8, ftLastAccessTime.dwHighDateTime=0x1d61894,
ftLastWriteTime.dwLowDateTime=0x3c0edfbf, ftLastWriteTime.dwHighDateTime=0x1d61894,
nFileSizeHigh=0x0, nFileSizeLow=0x98d58, dwReserved0=0x777aeec0, dwReserved1=0xbb6bff0,
cFileName="Installer.exe", cAlternateFileName="INSTAL~1.EXE")) returned 0x45ffc0;
```

Figure 18 - Obtain target file size for comparison

- Assuming the target file passes the initial tests, it is opened using the `CreateFileA` function and, based on the `dwDesiredAccess` value being set to `0x80000000` (`FILE_FLAG_WRITE_THROUGH`), any write operations will be made directly to disk and not cached in memory (Figure 19).

```
CreateFileA (lpFileName="C:\\Users\\<USERNAME>\\AppData\\Roaming\\Zoom\\bin\\INSTAL~1.EXE" (
normalized: "c:\\users\\<USERNAME>\\appdata\\roaming\\zoom\\bin\\instal~1.exe"),
dwDesiredAccess=0x80000000, dwShareMode=0x3, lpSecurityAttributes=0x0,
dwCreationDisposition=0x3, dwFlagsAndAttributes=0x80, hTemplateFile=0x0) returned 0x300;
```

Figure 19 - Open the target file for read/write operations

- To determine if the target file has previously been infected, the `SetFilePointer` function configures a 1,000 byte (`0x3e8`) offset from the beginning of the file after which the `ReadFile` function reads 256 bytes (`0x100`) (Figure 20).

```
SetFilePointer (in: hFile=0x300, lDistanceToMove=1000, lpDistanceToMoveHigh=0x0, dwMoveMethod=0
x0 | out: lpDistanceToMoveHigh=0x0) returned 0x3e8;
ReadFile (in: hFile=0x300, lpBuffer=0x195a40, nNumberOfBytesToRead=0x100, lpNumberOfBytesRead=0
x195a24, lpOverlapped=0x0 | out: lpBuffer=0x195a40*, lpNumberOfBytesRead=0x195a24*=0x100,
lpOverlapped=0x0) returned 1;
```

Figure 20 - Target file read offset to determine infection status

- The result of this read operation is compared against the Neshta code to ensure that an already infected file is not re-infected. Additionally, to ensure the target file can be modified, Neshta also checks for, and removes, any 'read only' file attribute using the `GetFileAttributesA` and `SetFileAttributesA` functions.

- To ensure the resulting infected executable file appears visually like the original legitimate executable, the file icon is located and copied so it can be restored and used (Figure 21).

```
CopyImage (h=0x3d050853, type=0x0, cx=32, cy=32, flags=0x2000) returned 0xb8050846;
```

Figure 21 - Target file icon (32x32 pixels) copied

- The target file is once again opened, this time with the `dwDesiredAccess` value set to `0xc0000000`, generic read and generic write access, and subsequently two bytes are read to confirm that the file is still a valid executable by checking for `MZ` in the file header (Figure 22).

```
CreateFileA (lpFileName="C:\\Users\\<USERNAME>\\AppData\\Roaming\\Zoom\\bin\\INSTAL~1.EXE" (
normalized: "c:\\users\\<USERNAME>\\appdata\\roaming\\zoom\\bin\\instal~1.exe"),
dwDesiredAccess=0xc0000000, dwShareMode=0x3, lpSecurityAttributes=0x0,
dwCreationDisposition=0x3, dwFlagsAndAttributes=0x80, hTemplateFile=0x0) returned 0x300;
ReadFile (in: hFile=0x300, lpBuffer=0x195b74, nNumberOfBytesToRead=0x2, lpNumberOfBytesRead=0x
195b40, lpOverlapped=0x0 | out: lpBuffer=0x195b74*, lpNumberOfBytesRead=0x195b40*=0x2,
lpOverlapped=0x0) returned 1;
```

Figure 22 - Target file opened for generic read/write operations

- Assuming the target file is confirmed as being a valid executable file, the first 41,472 bytes (`0xa200`) of the target file are read into a buffer, `0x195b74` in this case (Figure 23).

```
ReadFile (in: hFile=0x300, lpBuffer=0x195b74, nNumberOfBytesToRead=0xa200, lpNumberOfBytesRead=
0x195b40, lpOverlapped=0x0 | out: lpBuffer=0x195b74*, lpNumberOfBytesRead=0x195b40*=0xa200,
lpOverlapped=0x0) returned 1;
```

Figure 23 - Read the first (legitimate) 41,472 bytes of the target file into a buffer

- Having saved the legitimate target file content, Neshta's malicious code overwrites the first 41,472 bytes (`0xa200`) of the target file (Figure 24), infecting the file and ensuring that the malicious code will be executed whenever the file is launched.

```
WriteFile (in: hFile=0x300, lpBuffer=0x40a698*, nNumberOfBytesToWrite=0xa200,
lpNumberOfBytesWritten=0x195b40, lpOverlapped=0x0 | out: lpBuffer=0x40a698*,
lpNumberOfBytesWritten=0x195b40*=0xa200, lpOverlapped=0x0) returned 1;
```

Figure 24 - Neshta overwrites the beginning of the original legitimate executable

- Calling the `SetFilePointer` function with a `dwMoveMethod` of `0x2` (`FILE_END`) ensures that the next `WriteFile` operation appends data to the end of the target file. In this case, the previously buffered 41,472 bytes in `0x195b74`, obtained from the beginning of the legitimate target file, are written before the now infected file is closed (Figure 25).

```
SetFilePointer (in: hFile=0x300, lDistanceToMove=0, lpDistanceToMoveHigh=0x0, dwMoveMethod=0x2
| out: lpDistanceToMoveHigh=0x0) returned 0x98d58;
WriteFile (in: hFile=0x300, lpBuffer=0x195b74*, nNumberOfBytesToWrite=0xa200,
lpNumberOfBytesWritten=0x195b40, lpOverlapped=0x0 | out: lpBuffer=0x195b74*,
lpNumberOfBytesWritten=0x195b40*=0xa200, lpOverlapped=0x0) returned 1;
CloseHandle (hObject=0x300) returned 1;
```

Figure 25 - Appending the original legitimate code to the end of the infected executable.

RECOMMENDATIONS

Whilst the original infection vector hasn't been confirmed, users should be reminded to only install software from known trusted sources, in this case, the official Zoom Video Communications website [x]. Furthermore, given the use of COVID-19 themes and the active targeting of employees working remotely, users should be cautious of unsolicited communications and suspicious attachments, especially those that encourage the opening or installation of attachments or links.

[x] https://zoom.us

INDICATORS OF COMPROMISE

Whilst antivirus vendors will detect this threat as 'Win32.Neshta', 'Win32.Neshuta', 'Win32.Apanas' or variations of these names, the following indicators of compromise (IOC) will enable defenders to detect Neshta campaigns within their environments.

INITIAL NESHTA INFECTED ZOOM INSTALLER

MD5	SHA1	SHA256
d8aae1634936b71f02a53a9134fe651b	a5a8f4f8d5e2f81326641226c4370e555ba9c432	f25ce30e66372235855828a37212c41d1ad9b4b5b58b2d9773598e37eb8e021c
d0b8a0435b3c649a44bc9cd3e81628ae	d8eeece1ebbc8f1f83c95fef2a98fb159713513c	4ee8b7ed03b0903b683e47fbd78078f86de989e67f5399fb343e5f7e3b0fd0e3
86428556e0d06331b2a88a482ddc6e23	c9fda9208a85e956a562333a6c45382866f72d90	36e6d2f30c3798a2ea14b0bb2e68e7887a1c2b26310aa99a301ed8e77ccd5e78

DROPPED NESHTA PAYLOAD

MD5	SHA1	SHA256

44c3157c9d4f2e619f26da54cb9241d3	e48aa6ff4af90bba48bc5b86478a1fe57fdf909d	29482926cea5bdd91a24364d5e8eb54c165e89a00c30bcd24ffd9fa2fc35139d
fef59b651bab9b278cf2fca0564ed8ed	91b68bb2d9fe1358ceb18da5a1a48e6625ee4735	4a1f540669838f193fba145e79481bd3dbceab9c7142b8e252a96196a06c7e7b

REGISTRY KEYS

KEY	TYPE	DATA
HKEY_CLASSES_ROOT\exefile\shell\open\command	REG_SZ	C:\Windows\svchost.com "%1" %*

MUTEXES

The following hard-coded mutual exclusion object (mutex) is created during the execution of Neshta:

- `MutexPolesskayaGlush*.*\\x90svchost.com\\x90exefile\\\\shell\\\\open\\\\command\\x8bÃ \\"%1\\" %*\\x9c\\x91@`

SUSPICIOUS PATHS/FILENAMES

The following hard-coded paths and filenames are referenced during the execution of Neshta:

- svchost.com (Located in %WINDIR%)
- \\3582-490 (Located in %Temp% and used to store the original legitimate executable)

MITRE ATT&CK MAPPING

TECHNIQUE	TACTIC
T1192 - Spearphishing Link	Initial Access
T1193 - Spearphishing Attachment	Initial Access
T1204 - User Execution	Execution
T1060 - Registry Run Keys / Startup Folder	Persistence
T1055 - Process Injection	Defense Evasion, Privilege Escalation
T1027 - Obfuscated Files or Information	Defense Evasion
T1036 - Masquerading	Defense Evasion
T1045 - Software Packing	Defense Evasion
T1112 - Modify Registry	Defense Evasion
T1012 - Query Registry	Discovery
T1057 - Process Discovery	Discovery
T1083 - File and Directory Discovery	Discovery
T1080 - Taint Shared Content	Lateral Movement
T1091 - Replication Through Removable Media	Initial Access, Lateral Movement
T1071 - Standard Application Layer Protocol	Command and Control

CONTACT US

www.cyberint.com | sales@cyberint.com | blog.cyberint.com

USA

Tel: +1-646-568-7813
214 W 29th St, 2nd Floor New York, NY 10001

ISRAEL

Tel: +972-3-7286-777
17 Ha-Mefalsim St 4951447 Petah Tikva

UNITED KINGDOM

Tel: +44-203-514-1515
Fox Court 14 Grays Inn Rd, Holborn, WC1X 8HN, Suite 2068 London

SINGAPORE

Tel: +65-3163-5760
135 Cecil St. #10-01 MYP PLAZA 069536

LATAM

Tel: +507-395-1553
Panama City