New OpcJacker Malware Distributed via Fake VPN Malvertising

trendmicro.com/en_us/research/23/c/new-opcjacker-malware-distributed-via-fake-vpn-malvertising.html

March 29, 2023

Malware

We discovered a new malware, which we named “OpcJacker” (due to its opcode configuration design and its cryptocurrency hijacking ability), that has been distributed in the wild since the second half of 2022.

By: Jaromir Horejsi, Joseph C Chen March 29, 2023 Read time: (words)

We discovered a new malware, which we named “OpcJacker” (due to its opcode configuration design and its cryptocurrency hijacking ability), that has been distributed in the wild since the second half of 2022. OpcJacker is an interesting piece of malware, since its configuration file uses a custom file format to define the stealer’s behavior. Specifically, the format resembles custom virtual machine code, where numeric hexadecimal identifiers present in the configuration file make the stealer run desired functions. The purpose of using such a design is likely to make understanding and analyzing the malware’s code flow more difficult for researchers.

OpcJacker’s main functions include keylogging, taking screenshots, stealing sensitive data from browsers, loading additional modules, and replacing cryptocurrency addresses in the clipboard for hijacking purposes.

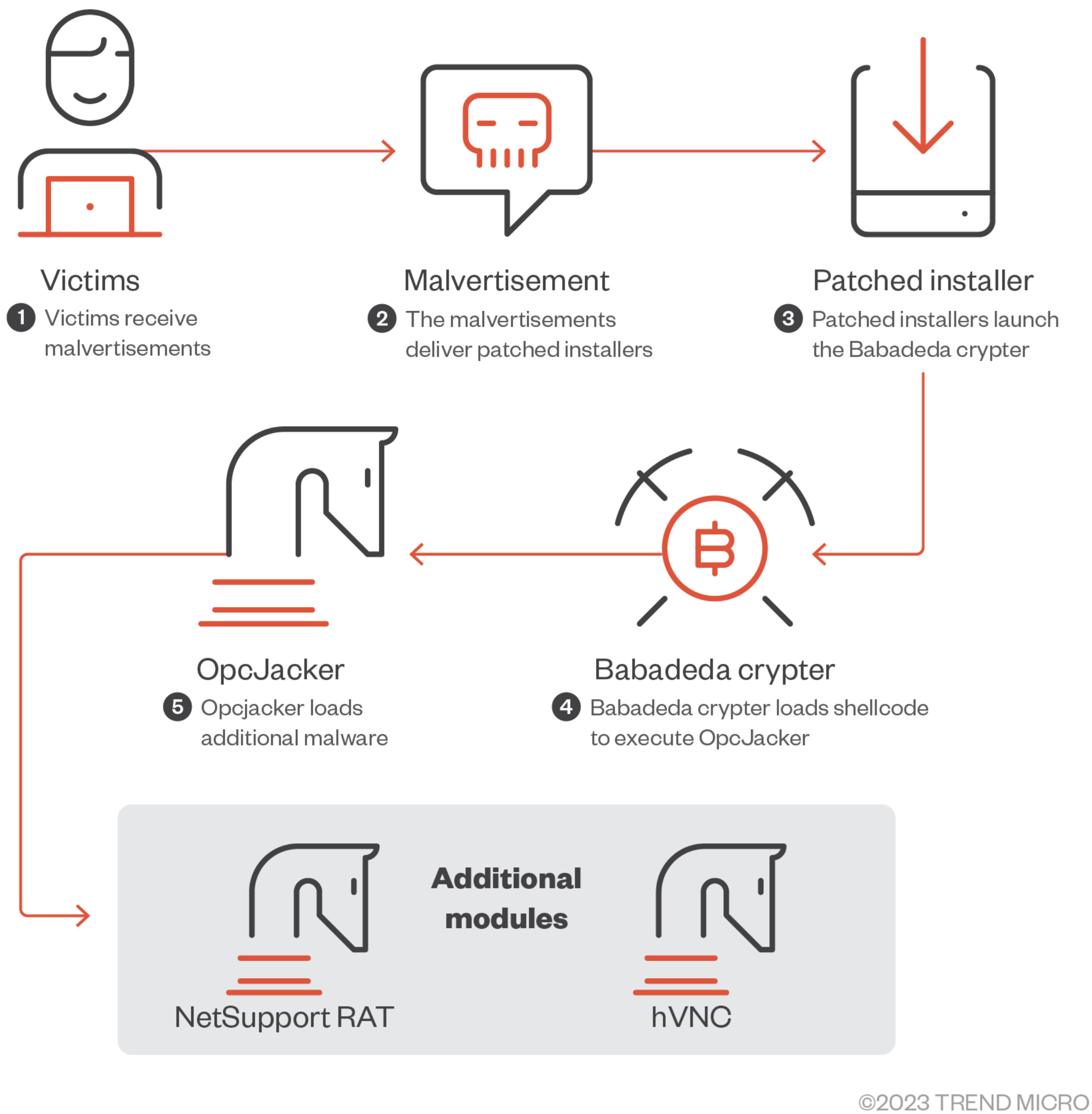

Figure 1. The OpcJacker infection chain

We've observed OpcJacker being distributed via different campaigns that involve the malware being disguised as cryptocurrency-related applications and other legitimate software, which the threat actors distribute through fake websites. In the latest (February 2023) campaign involving OpcJacker, the infection chain began with malvertisements that were geofenced to users in Iran. The malvertisements were disguised as a legitimate VPN service that tricked its victims into downloading an archive file containing OpcJacker.

The malware is loaded by patching a legitimate DLL library within an installed application, which loads another malicious DLL library. This DLL library then assembles and runs shellcode — the loader and runner of another malicious executable — and OpcJacker from chunks of data stored in data files of various formats, such as Waveform Audio File Format (WAV) and Microsoft Compiled HTML Help (CHM). This loader has been in use for over a year since it was previously described and named as the Babadeda crypter. The threat actor behind the campaign implemented a few changes in the cryptor itself, then added a completely new payload (a stealer/clipper/keylogger).

We noticed that OpcJacker mostly drops (or downloads) and runs additional modules, which are remote access tools — either the NetSupport RAT or a hidden virtual network computing (hVNC) variant. We also found a report sharing information on a loader called "Phobos Crypter" (which is actually the same malware as OpcJacker) being used to load the Phobos ransomware.

Delivery

As mentioned in the introduction, we observed OpcJacker being distributed through several different campaigns that usually involve fake websites advertising seemingly legitimate software and cryptocurrency-related applications, but are actually hosting malware. As these campaigns deliver a few other different malware in addition to OpcJacker, we believe that they are most likely to be different pay-per-install services leveraged by OpcJacker's operators.

In the latest campaign from February 2023, we noticed OpcJacker being distributed via malvertisements geotargeting Iran. These malvertisements were linked to a malicious website disguised as a website for a legitimate VPN software. The site's content was copied from the website of a legitimate commercial VPN service — however, the links were modified to link to a compromised website hosting malicious content.

The malicious website checks the client's IP address to determine whether the victim uses a VPN service. If the IP address is not from a VPN service, it then redirects the victim to the second compromised website to lure them into downloading an archive file containing OpcJacker. Note that the attack will not proceed if the intended victim is using a VPN service.

Figure 2. An example of a malvertisement designed to deliver OpcJacker

Furthermore, we also found a bunch of ISO images and RAR/ZIP archives containing modified installers of various pieces of software that all lead to the loading of OpcJacker. These installers, which were previously used by other campaigns, were hosted on various hacked WordPress-powered websites or software development platforms like GitHub. A possible reason why threat actors favor the use of ISO files is to bypass Mark-of-the-Web warnings.

The following are some file name examples we found:

- CLF_security.iso
- Cloudflare_security_setup.iso
- GoldenDict-1.5.0-RC2-372-gc3ff15f-Install.zip
- MSI_Afterburner.iso
- tigervnc64-winvnc-1.12.0.rar
- TradingViewDesktop.zip
- XDag.x64.rar

Babadeda crypter

Note that the file names mentioned in this section often change between different installers. However, their overall functions remain the same.

After the installer drops all the necessary files, it then loads the main executable file (*RawDigger.exe*), which is a clean legitimate file.

qe	msi
settings	dat
librawf	dll
libpushpp	dll
hm	
unins000	dat
unins000	msg
rawspeed	dll
RawDigger	exe
jpeg8	dll
libxml2	dll
README	txt
Copyrights	txt
Changelog	txt
avutil-56	dll
swresample-3	dll
avcodec-58	dll
avformat-58	dll
libmap	dll
vccorlib140	dll
msvcp140_codecvt_ids	dll
msvcp140_atomic_wait	dll
msvcp140_2	dll
msvcp140_1	dll
concrt140	dll
API-MS-Win-core-xstate-l2-1-0	dll
api-ms-win-core-console-l1-2-0	dll
client32	ini

_Figure 3. A list of files dropped by the installer; while most of them are clean legitimate files, some are patched or malicious files

The executable file loads a DLL library that includes patched imports (*librawf.dll*).

DllName	OriginalFirstThunk	TimeDateStamp	ForwarderChain	Name	FirstThunk
jpeg8.dll	0011CF10	00000000	00000000	0011CFA4	000C8B08
librawf.dll	0011CF34	00000000	00000000	0011D386	000C8B2C
QtGui4.dll	0011BD50	00000000	00000000	0012A7A2	000C7948
QtNetwork4.dll	0011CE7C	00000000	00000000	0012AD8E	000C8A74
QtCore4.dll	0011B5BC	00000000	00000000	0012F8BA	000C71B4

_Figure 4. A list of imported DLL libraries; the highlighted library was patched to load another malicious DLL library

The patched DLL's (*librawf.dll*, which is connected to the legitimate app RawDigger, a raw image analyzer) import address table was further patched to include two additional DLL libraries. In the figure below, notice how the FirstThunk addresses (of the newly added libraries) start with 001Dxxxx instead of the 0012xxxx used in the FirstThunk addresses from the original libraries.

DllName	OriginalFirstThunk	TimeDateStamp	ForwarderChain	Name	FirstThunk
MSVCP100.dll	00166250	00000000	00000000	0016779A	00122098
WS2_32.dll	001664F4	00000000	00000000	001677A8	0012233C
MSVCR100.dll	00166260	00000000	00000000	00167B80	001220A8
libpushpp.dll	001D2101	00000000	00000000	001D20DC	001D20F9
libmap.dll	001D212B	00000000	00000000	001D2109	001D2123

_Figure 5. A patched import address table

The highlighted library in Figure 5 (*libpushpp.dll*) is then loaded and executed. Its main task is to open one of the data files (*hm*) and load the first stage shellcode stored inside.

```
FileA = CreateFileA("hm", 0xC0000000, 1u, 0, 3u, 0x80u, 0);
v137 = 4i64;
hFile = FileA;
```

_Figure 6. Malicious library opening a data file

The offset and size of the first stage shellcode is hardcoded into the DLL library.

```
hFile = (char *)VirtualAlloc(0, 0x4F0000u, 0x3000u, 0x40u) + 5120;
qmemcpy(hFile, pBufferRead + 0x37D50, 0x75Au);
```

_Figure 7. Malicious library copying the first stage shellcode from offset 0x37D50; the size of the shellcode is 0x75A bytes

In newer versions of the Babadeda crypter, another DLL library (*mdb.dll*, from the fake VPN installer) is loaded into memory, after which a hardcoded, randomly selected block of memory is overwritten with the first stage shellcode. Note that this change is just a small detail and has no influence on the first stage shellcode's overall function.

```
hmdb_dll = LoadLibraryA("mdb.dll");
if ( hmdb_dll )
  hmdb_dll += 0x400;
hmdb_dll_plus_0x2a00 = (char *)(hmdb_dll + 0x680);
LODWORD(lpEnumFunc_shellcode_address) = hmdb_dll + 0x680;
memset(hmdb_dll + 0x680, 0, 0x7B5u);
qmemcpy(hmdb_dll_plus_0x2a00, pBuffer_gp_chm + 0x4551C, 0x7B5u);
```

_Figure 8. The legitimate library (mdb.dll) is loaded into memory, after which the first stage shellcode (0x7B5 bytes) is copied into the library's memory space

There is a configuration table containing offsets of encrypted chunks followed by their respective sizes at the end of the first stage shellcode. The first stage shellcode then decrypts and combines all chunks to form the second stage shellcode (a loader) and the main malware (OpcJacker with the ability to load additional malicious modules).

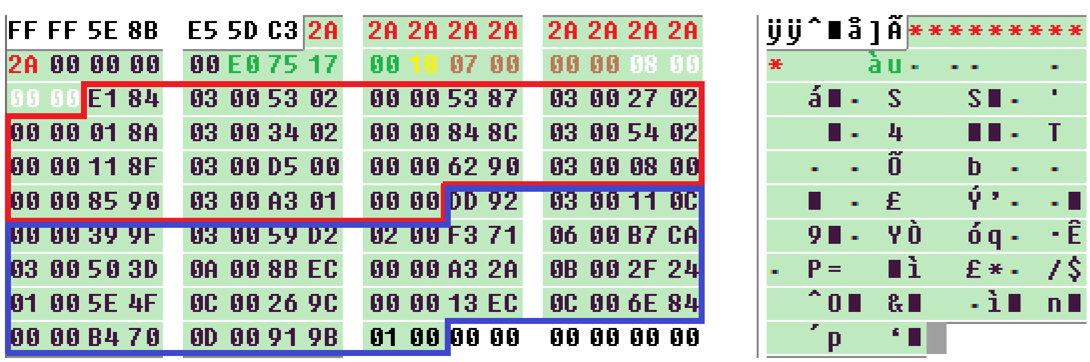

_Figure 9. Configuration table of the first stage shellcode

The configuration table starts with at least eight of the same characters (the red colored "*" in Figure 9, but different characters may be used in other samples), followed by the total length of the data file (green color; length of *hm* = 0x1775e0 = 1537504 bytes), the encryption key (yellow color; 0x18), the number of chunks in the second stage of the shellcode (brown color;

0x07), and finally, by the number of chunks in the main malware (white color; 0x08). The list of 0x07 (red bracket) and 0x08 (blue bracket) is equivalent to fifteen addresses and sizes of each chunk.

At the beginning of the data file (*hm*), we can see the (WAV) file header as it tries to mimic a WAVE file format. Note that the data file can be a different file format, since we also observed CHM being used.

```
00000000: 52 49 46 46 D8 75 17 00|57 41 56 45 66 6D 74 20  | RIFFØu· WAVEfmt
00000010: 28 00 00 00 FE FF 08 00|80 BB 00 00 00 B8 0B 00  | (   þÿ· €»   ¸·
00000020: 10 00 10 00 16 00 10 00|3F 06 00 00 01 00 00 00  | · · · · ?·  .
00000030: 00 00 10 00 80 00 00 AA|00 38 9B 71 64 61 74 61  |   · €  ª 8qdata
00000040: F0 6F 17 00 00 00 00 00|27 00 00 00 00 00 00 00  | ðo·     '
```

Figure 10. Data file that starts with a WAV header

Main stealer component (OpcJacker)

The main malware component (OpcJacker) is an interesting stealer that first decrypts and loads its configuration file. The configuration file format resembles a bytecode written in a custom machine language, where each instruction is parsed, individual opcodes are obtained, and then the specific handler is executed.

When analyzing the custom bytecode, we noticed the following patterns:

ASCII strings were encoded as 01 xx xx xx xx <string bytes>; where xx xx xx xx is the length of the string.

```
00 00 01 09  00 00 00 5C  44 61 74 61  42 61 73 65  |   \DataBase
```

Figure 11. Encoded ASCII string inside the configuration file

Similarly, wide character strings started with byte 02, while binary arrays started with byte 03.

```
00 00 00 09  00 00 00 A5  06 00 00 02  30 00 00 00  |        ¥·   0
25 00 70 00  72 00 6F 00  67 00 72 00  61 00 6D 00  | % p r o g r a m
64 00 61 00  74 00 61 00  25 00 5C 00  75 00 73 00  | d a t a % \ u s
6E 00 65 00  74 00 5C 00  73 00 61 00  76 00 65 00  | n e t \ s a v e
```

Figure 12. Encoded UNICODE string inside the configuration file

```
65 30 00 00  00 2E 00 00  00 FA 00 00  00 03 00 32  | e0   .   ú   · 2
5F 00 4D 5A  90 00 03 00  00 00 04 00  00 00 FF FF  | _ MZ  ·   ·   ÿÿ
```

Figure 13. Encoded binary array inside the configuration file

The configuration file format is a sequence of instructions where instruction starts with three 4-byte little-endian (DWORD) numbers. The first number is the virtual program counter, the second is likely the parent instruction's virtual program counter, while the third is the handler ID (code to be executed in the virtual machine), followed by data bytes or additional handler IDs.

Based on these observations, we wrote an instruction parser, from which we were presented with the following output. Although our observations and understanding of the virtual machine's internal implementation was incomplete, the parser gave us a good understanding of what behavior was defined in the configuration file.

The decrypted and decoded configuration file starts with the initialization of certain system variables, with “*test*” and “*rik*” likely being campaign IDs. The configuration file dropped by SHA256 c5b499e886d8e86d0d85d0f73bc760516e7476442d3def2feeade417926f04a5 contains different keywords “*test*” and “*ilk*” as campaign IDs. Meanwhile, the configuration file dropped by the latest campaign from February 2023 (SHA256 565EA7469F9769DD05C925A3F3EF9A2F9756FF1F35FD154107786BFC63703B52) contains the keywords “*test_installs*” and “*yorik*.”

```
00000001: 00000000: 000005dc: 00 00 00 00 00
00000002: 00000001: 000005e7: b'\\DataBase'
00000003: 00000001: 000005e8: b'test'
00000004: 00000001: 000005e9: b'rik'
00000005: 00000001: 000005e6: data value: 01
00000006: 00000000: 000001f4: 00 00 00 00 00
00000007: 00000000: 0000012c: 00 00 00 00 00
00000008: 00000000: 00000640: 00 00 00 00 00
00000009: 00000000: 000006a4: 00 00 00 00 00
0000000a: 00000009: 000006a5: %programdata%\usnet\save
0000000b: 00000009: 000006a6: %programdata%\usnet\task
```

_Figure 14. Initialization commands

Then initialization of clipboard replacement functionality (clipping) follows.

```
0000000c: 00000000: 00000578: 00 00 00 00 00                    clipper
0000000d: 0000000c: 00000597: b'automatic'
0000000e: 0000000c: 00000598: b'ucustom'
0000000f: 0000000c: 00000599: b'15ktoNbCXGZZebqDc5tLV6cxrPB7JjYF7S'
00000010: 0000000c: 00000599: b'bc1q20uw3sphsjz47174wcz7tchw7ddq5tulapcj4m'
00000011: 0000000c: 0000059a: b'0xe82Bcb6d75Ec304D2447B587Dee01A0D5aB25785'
00000012: 0000000c: 0000059b: b'TRpsovtxWSWYkSFFBY5u8ZiqKEphQzWmiv'
00000013: 0000000c: 0000059c: b'bnb1kz5temu9dpuwlwx87wn85akefm8c066s857wc8'
00000014: 0000000c: 0000059d: b'LPpSV1hsvD28peed5T42bBH12iBH8iYcRc'
00000015: 0000000c: 0000059d: b'ltc1qmwys30pszs0nfxcg9ggzxn05zxqunghsa4l88s'
00000016: 0000000c: 0000059e: b'4BBrGNnn9CYebvSTV3VZ4K6odRhFwrQxGhDaUwV2tbQ5WqnsW8ojtoP5CEc3Q7Lax4Lmdfy4FXMSpDtSroL3jQ1r7dVb5Ao'
00000017: 0000000c: 0000059f: b'GAR76RTZD7PL4PL2POS7WYTAKVWZKUIFGZ3RZR3LONWSAUDOZWMO6FQ2'
00000018: 0000000c: 000005a0: b'rPJAjQ4AE5j7sz63V1SWV4WL2P9dXZYvJN'
00000019: 0000000c: 000005a1: b'addr1q8gcf6cunxc8wjke6jd55r4anj94hq5xs4v3rs5lzrxjgwpgzeff4j830ee7fepyhqgw9p6ryr7cap73x2ksl9ecgweqg047u3'
0000001a: 0000000c: 000005a2: b'DT86c9XE8fsdnEftdMeXAkxW3Wx5KGmPdd'
0000001b: 0000000c: 000005a3: b'Xuh6VJVhVfQxonbyjBbdRkDxz4b4KFMrr5'
0000001c: 0000000c: 000005a4: b'tz1KiqtqcA8driWrFeqpJGSMVaktFfmxnodb'
0000001d: 0000000c: 000005a5: b'57wmmCSdhsY1ZN9cv4kUepiGMrqKqS12wbpBGDq2artr'
0000001e: 0000000c: 000005a6: b'cosmos1pjngjkq52u9fa7ynlg9k7addhsu8hh5y62ytgu'
0000001f: 0000000c: 000005a7: b't1f6LRR7fUrLCPqa94i6S9Zast2wzv82XSy'
00000020: 0000000c: 000005a8: b'qr909zt20ntnf5w7mzrpjh29vmhptxxt8g7uz2zpuv'
00000021: 0000000c: 000005a9: b'12Kmtbte2attx8m2nmMmV5S25Wmc4h2FDzLwRM5q9e3xkn1E'
00000022: 0000000c: 000005aa: b'terra1yfykxet8f8cyy3n7rm5nqlenggnr21d967evk4'
```

Figure 15. Clipboard replacer (clipper) initialization

Later, the variable “*exe”* is initialized with executable file bytes (see the 4d 5a 90 = MZ marker). This executable is a remote access tool.

```
0000002f: 0000002e: 000000f9: b'exe'
00000030: 0000002e: 000000fa: binary blob: len 00017000 bytes: data: 4d 5a 90
```

Figure 16. The embedded module (PE EXE format)

The malware sets up persistence via registry run and task scheduler methods. Note the *$itself_exe* variable used for holding the file name of the current process.

```
0000008b: 0000008a: 000000ef: 00 00 00 00 00
0000008c: 0000008b: 000000f0: b'$hk_autorun'                    (maybe h = HKML)
0000008d: 0000008b: 000000e5: 00 00 00 00 00
0000008e: 0000008d: 000000e6: empty string
0000008f: 0000008b: 000000eb: 00 00 00 00 00
00000090: 0000008f: 000000ec: command: 000003e8; data: 00000000
00000091: 0000008f: 000000ed: command: 000003ea; data: 00000000    persistence HKLM
00000092: 0000008f: 000000ee: b'$itself_exe'
00000093: 0000008f: 000000ee: b'RawDig Inspector'
00000094: 0000008a: 000000ef: 00 00 00 00 00
00000095: 00000094: 000000f0: b'$sh_autorun'                    (maybe s = scheduler)
00000096: 00000094: 000000e5: 00 00 00 00 00
00000097: 00000096: 000000e6: empty string
00000098: 00000094: 000000eb: 00 00 00 00 00
00000099: 00000098: 000000ec: command: 000003e8; data: 00000000
0000009a: 00000098: 000000ed: command: 000003ec; data: 00000000    persistence task scheduler; at log on
0000009b: 00000098: 000000ee: b'Common'
0000009c: 00000098: 000000ee: b'RawDig Inspector'
0000009d: 00000098: 000000ee: b'RawDig Inspector'
0000009e: 00000098: 000000ee: b'$itself_exe'
```

Figure 17. Method for setting persistence

The malware then starts the clipper function, that is, it monitors the clipboard for cryptocurrency addresses and replaces them with its own cryptocurrency addresses controlled by the attackers.

```
000000c2: 000000c1: 000000e8: b'$invm'
000000c3: 000000c1: 000000e9: b'no'
000000c4: 000000c1: 000000ea: b'is'
000000c5: 000000bd: 000000eb: 00 00 00 00 00
000000c6: 000000c5: 000000ec: command: 00000578; data: 00000000
000000c7: 000000c5: 000000ed: command: 00000579; data: 00000000    clipper start
000000c8: 000000b0: 000000f1: 00 00 00 00 00
000000c9: 000000c8: 000000f3: b'$evid_onclip'                    was clip event received?
000000ca: 000000c8: 000000e5: 00 00 00 00 00
000000cb: 000000ca: 000000e6: empty string
000000cc: 000000c8: 000000eb: 00 00 00 00 00
000000cd: 000000cc: 000000ec: command: 00000578; data: 00000000
000000ce: 000000cc: 000000f4: command: 0000057b; data: 00000000    clipper op
000000cf: 000000c8: 000000f2: 00 00 00 00 00
000000d0: 000000cf: 000000ef: 00 00 00 00 00
000000d1: 000000d0: 000000f0: empty string
000000d2: 000000d0: 000000e5: 00 00 00 00 00
000000d3: 000000d2: 000000e6: empty string
000000d4: 000000d0: 000000eb: 00 00 00 00 00
000000d5: 000000d4: 000000ec: command: 00000578; data: 00000000
000000d6: 000000d4: 000000ed: command: 0000057f; data: 00000000    clipper op
```

Figure 18. The clipper function

Finally, the *virtual_launch_exe* function runs the previously embedded executable, which we observed to be RATs, either the NetSupport RAT, the NetSupport RAT downloader, or hVNC.

```
000000ec: 00000000: 000000e0: 00 00 00 00 00
000000ed: 000000ec: 000000e1: b'virtual_launch_exe'              function name
000000ee: 000000ec: 000000e2: data value: 01
000000ef: 000000ec: 000000e3: data value: 01
000000f0: 000000ec: 000000e4: data value: 00
000000f1: 000000ec: 000000ef: 00 00 00 00 00
000000f2: 000000f1: 000000f0: empty string
000000f3: 000000f1: 000000e5: 00 00 00 00 00
000000f4: 000000f3: 000000e6: empty string
000000f5: 000000f1: 000000eb: 00 00 00 00 00
000000f6: 000000f5: 000000ec: command: 00000384; data: 00000000
000000f7: 000000f5: 000000ed: command: 00000388; data: 00000000    runPE
000000f8: 000000f5: 000000ee: b'exe'
```

Figure 19. Function to run the embedded executable file

Handler IDs in custom virtual machines

As can be observed in the third column (or decoded "command" variable) in a few of the previous screenshots, the virtual machine implements numerous internal handlers. Most of these are related to various data manipulations. We list a few of the notable handlers that have specific high-level functionalities in Table 1. The functions the stealer implements include the following: clipping (clipboard content replacement), keylogging, file execution and listing, killing processes, stealing chromium credentials, detecting idleness, and detecting virtual machines. However, during our testing scenarios, we observed the stealer mostly just sets the persistence and delivers additional modules (remote access tools).

Handler ID	Function
0x3E9	Used for persistence (registry; HKCU)
0x3EA	Used for persistence (registry; HKLM)
0x3EB	Used for persistence (startup folder)
0x3EC;0x3ED	Used for persistence (task scheduler)
0x7d1	Lists files
0x579	Starts clipper
0x57A	Stops clipper
0x12d	Puts the machine into sleep mode
0x385	Terminates process
0x387	Exits process
0x388; 0x38B	Runs PE executable
0x389	Runs shellcode
0x38A	Runs PE executable export routine
0x76D	Gets current committed memory limit (ullTotalPageFile)
0x76E	Gets the amount of actual physical memory (ullTotalPhys)
0x641	Steals sensitive data from Chromium
0x259	Checks if the machine is idle and if the cursor is not moving
0x25B	Checks if the machine is idle and if no new process is being created
0x25D	Checks if the machine idle and if no new window is being created

0x835	Starts keylogger
0x836	Starts keylogger for a certain period
0x837	Stops keylogger
0x839	Copies data (likely logs) then return 0x83a (klogs)
0x1F5	Retrieves VMWare via CPUID
0x1f7	Searches for 'virtual' in SYSTEM\\ControlSet001\\Services\\disk\\Enum
0x83A	Writes file(s) to klogs//
0x89a	Writes file(s) to screenshots\\
0x596	Writes to clp\clp_log.txt
0xf6	Writes file(s) to chromium_creds\\
0xCE	Copies files to filesystem\\
0x321	Creates messagemonitor window, which needed for the clipper
0x322	Destroys messagemonitor window, which is needed for the clipper
0x5DC	Gets environment ID
0x5E0	Runs GetModuleFileNameW, which is needed for resolving $itself_exe

Table 1. Virtual machine command IDs

```
 switch ( a2 )
 {
   case 0x835:
     started = start_keylogger_ex();
ABEL_8:
     *(_BYTE *)(a1 + 12) = started;
     return a1;
   case 0x837:
     v4 = UnhookWindowsHookEx(hhk);
     hhk = 0;
     started = v4;
     goto LABEL_8;
   case 0x836:
     if ( *a3 != a3[1] )
     {
       started = start_keylogger_ex2(*(
       goto LABEL_8;
     }
     break;
   case 0x839:
```

Figure 20. Keylogger-related commands implemented within the stealer's binary; Command IDs can also be observed in the screenshot (0x835; 0x837; 0x836; 0x839)

Embedded modules

NetSupport RAT module

Some embedded modules contain the *client32.exe* (SHA256 *18DF68D1581C11130C139FA52ABB74DFD098A9AF698A250645D6A4A65EFCBF2D* or SHA256 *49A568F8AC11173E3A0D76CFF6BC1D4B9BDF2C35C6D8570177422F142DCFDBE3*) file from the NetSupport RAT. This single file is not enough, however, as the NetSupport tool needs additional DLL libraries and a configuration file. Note that these missing files have already been dropped by the modified installer into the installation directory.

For researchers, the most important file is called *client32.ini*, which contains important settings such as gateway addresses, gateway keys (GSK), and ports.

```
[HTTP]
CMPI=60
GatewayAddress=uzurtela42.com:3961
GSK=FM;I@GED9M>LBPGP<NAFEM:D
Port=3961
SecondaryGateway=uzurtela1.com:3961
SecondaryPort=3961
```

Figure 21. The configuration file of NetSupport RAT

NetSupport RAT downloader module

Some embedded modules contain the NetSupport RAT downloader (SHA256 *C68096EB0A655924CA840EA1C71F9372AC055F299B52335AD10DDFA835F3633D*). This downloader decrypts the URL payload, then downloads and executes it.


```
þô.X7Þ ·· U D P _ H e l p e r .F h t t p : / / n e s u p c l i . c o m / n
e s u p / n e s k 2 . z i p .D h t t p : / / n e s u p c l i . c o m / g l
u / n e s k 2 g . z i p x·· ■·ÊÙ·■Âi}±Þ■X¯.x·· ■·ÊÙ·■Âi}±Þ■X¯.x·· ■·ÊÙ·■Âi}
±Þ■X¯.x·· ■·ÊÙ·■Âi}±Þ■X¯.x·· ■·ÊÙ·■Âi}±Þ■X¯.x·· ■·ÊÙ·■Âi}±Þ■X¯.x·· ■·ÊÙ·■Âi
}±Þ■X¯.x·· ■·ÊÙ·■Âi}±Þ■X¯.x·· ■·ÊÙ·■Âi}±Þ■X¯.x·· ■·ÊÙ·■Âi}±Þ■X¯.x·· ■·ÊÙ·■Â
i}±Þ■X¯.x·· ■·ÊÙ·■Âi}±Þ■X¯.x·· ■·ÊÙ·■Âi}±Þ■X¯.x·· ■·ÊÙ·■Âi}±Þ■X¯.x·· ■·ÊÙ·■
Âi}±Þ■X¯.x·· ■·ÊÙ·■Âi}±Þ■X¯.x·· ■·ÊÙ·■Âi}±Þ■X¯.x·· ■·ÊÙ·■Âi}±Þ
```


Figure 22. Decrypted downloader’s configuration file, with additional URLs being visible in clear text

The decrypted configuration contains two URLs, one leading to an archive containing the NetSupport RAT, like the previous module, while the second contains a few batch scripts, which display messages such as the one seen in Figure 23. Later, one of these batch scripts downloads additional stealers.

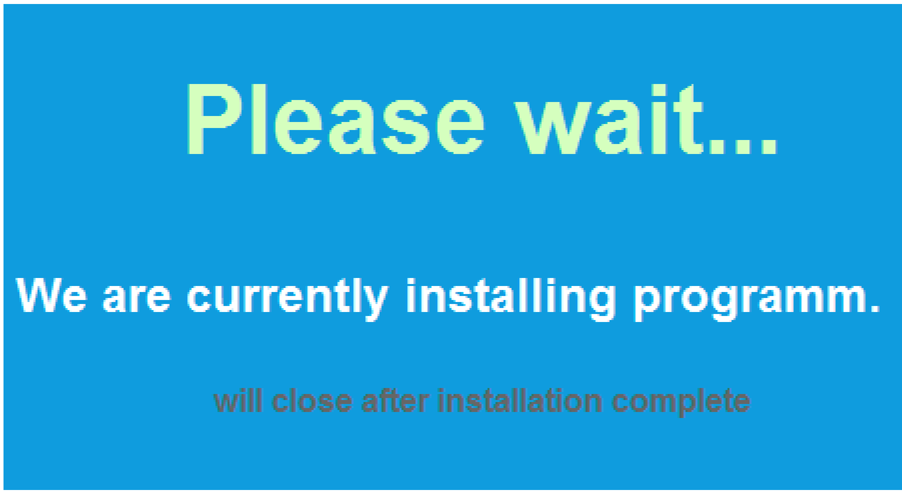

_Figure 23. Decoy message telling the victim to wait for the program to be installed

hVNC module

Some embedded modules contain a modified hVNC module *F772B652176A6E40012969E05D1C75E3C51A8DB4471245754975678F04DEDAAA*. This module, in addition to standard remote desktop functionality, also contains routines to search for the existence of the following cryptocurrency related Google Chrome, Microsoft Edge, and Mozilla Firefox extensions (wallets):

Google Chrome extension ID	Extension name
ffnbelfdoeiohenkjibnmadjiehjhajb	Yoroi
ibnejdfjmmkpcnlpebklmnkoeoihofec	TronLink
jbdaocneiiinmjbjlgalhcelgbejmnid	Nifty Wallet
nkbihfbeogaeaoehlefnkodbefgpgknn	MetaMask
afbcbjpbpfadlkmhmclhkeeodmamcflc	Math Wallet
hnfanknocfeofbddgcijnmhnfnkdnaad	Coinbase Wallet
fhbohimaelbohpjbbldcngcnapndodjp	Binance Wallet
odbfpeeihdkbihmopkbjmoonfanlbfcl	Brave Wallet
hpglfhgfnhbgpjdenjgmdgoeiappafln	Guarda Wallet
blnieiiffboillknjnepogjhkgnoapac	Equall Wallet
cjelfplplebdjjenllpjcblmjkfcffne	Jaxx Liberty
fihkakfobkmkjojpchpfgcmhfjnmnfpi	BitApp Wallet
kncchdigobghenbbaddojjnnaogfppfj	iWallet

amkmjjmmflddogmhpjloimipbofnfjih	Wombat
fhilaheimglignddkjgofkcbgekhenbh	Oxygen
nlbmnnijcnlegkjjpcfjclmcfggfefdm	MyEtherWallet
nanjmdknhkinifnkgdcggcfnhdaammmj	GuildWallet
nkddgncdjgjfcddamfgcmfnlhccnimig	Saturn Wallet
fnjhmkhhmkbjkkabndcnnogagogbneec	Ronin Wallet
aiifbnbfobpmeekipheeijimdpnlpgpp	Station Wallet
fnnegphlobjdpkhecapkijjdkgcjhkib	Harmony
aeachknmefphepccionboohckonoeemg	Coin98
cgeeodpfagjceefieflmdfphplkenlfk	EVER Wallet
pdadjkfkgcafgbceimcpbkalnfnepbnk	KardiaChain
bfnaelmomeimhlpmgjnjophhpkkoljpa	Phantom
fhilaheimglignddkjgofkcbgekhenbh	Oxygen
mgffkfbidihjpoaomajlbgchddlicgpn	Pali
aodkkagnadcbobfpggfnjeongemjbjca	BoltX
kpfopkelmapcoipemfendmdcghnegimn	Liquality
hmeobnfnfcmdkdcmlblgagmfpfboieaf	XDEFI
lpfcbjknijpeeillifnkikgncikgfhdo	Nami
dngmlblcodfobpdpecaadgfbcggfjfnm	MultiversX DeFi

Table 2. Targeted Chrome extensions

Microsoft Edge extension ID	Extension name
akoiaibnepcedcplijmiamnaigbepmcb	Yoroi
ejbalbakoplchlghecdalmeeeajnimhm	MetaMask
dfeccadlilpndjjohbjdblepmjeahlmm	Math Wallet
kjmoohlgokccodicjjfebfomlbljgfhk	Ronin Wallet
ajkhoeiiokighlmdnlakpjfoobnjinie	Terra Station

fplfipmamcjaknpgnipjeaeeidnjooao	BDLT wallet
niihfokdlimbddhfmngnplgfcgpmlido	Glow
obffkkagpmohennipjokmpllocnlndac	OneKey
kfocnlddfahihoalinnfbnfmopjokmhl	MetaWallet

Table 3. Targeted Edge extensions

Mozilla Firefox extension ID	Extension name
{530f7c6c-6077-4703-8f71-cb368c663e35}.xpi	Yoroi
ronin-wallet@axieinfinity.com.xpi	Ronin Wallet
webextension@metamask.io.xpi	MetaMask
{5799d9b6-8343-4c26-9ab6-5d2ad39884ce}.xpi	TronLink
{aa812bee-9e92-48ba-9570-5faf0cfe2578}.xpi	
{59ea5f29-6ea9-40b5-83cd-937249b001e1}.xpi	
{d8ddfc2a-97d9-4c60-8b53-5edd299b6674}.xpi	
{7c42eea1-b3e4-4be4-a56f-82a5852b12dc}.xpi	Phantom
{b3e96b5f-b5bf-8b48-846b-52f430365e80}.xpi	
{eb1fb57b-ca3d-4624-a841-728fdb28455f}.xpi	
{76596e30-ecdb-477a-91fd-c08f2018df1a}.xpi	

Table 4. Targeted Firefox extensions

In our analyzed sample, command-and-control (C&C) communication starts with the following magic:

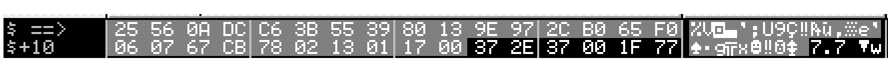

Figure 24. HVNC network communication magic

The snippet below shows that some values are hardcoded into the executable, others are generated from MachineGuid or randomly generated. Note the string “7.7” seen in Figure 25, which is likely the modified hVNC version.

```
dwRandomValue_ = dwRandomValue;
*(_WORD *)(pBufferOut + 0x10) = 0x706;
*(_DWORD *)(pBufferOut + 0x12) = dwMachineGuid;
*(_DWORD *)pBufferOut = 0xDC0A5625;
*(_DWORD *)(pBufferOut + 4) = 0x39553BC6;
*(_DWORD *)(pBufferOut + 8) = 0x979E1380;
lstrcpyA((LPSTR)(pBufferOut + 0x1A), "7.7");
```

Figure 25. Code generating hVNC packet magic

Conclusion

It seems that OpcJacker's operator is motivated by financial gain, since the malware's primary purpose is stealing cryptocurrency funds from wallets. However, its versatile functions also allow OpcJacker to act as an information stealer or a malware loader, meaning it can be used beyond its initial intended use.

The campaign IDs we found in the samples, such as "test" and "test_installs", indicate that OpcJacker could still be under development and testing stages. Given its unique design combined with a variety of VM-like functionalities, it's possible that the malware could prove to be popular with threat actors, and therefore could see use in future threat campaigns.

Indicators of Compromise

The indicators for this blog entry can be found here.