Mylobot: Investigating a proxy botnet

Archived: 2026-04-05 15:08:25 UTC

Mylobot is a malware that targets Windows systems, it first appeared in 2017 and until now hasn’t received much attention over the years. In this article, we'll focus on its main capability, which is transforming the infected system into a proxy. We'll also see how it's distributed and the capabilities of its downloader. We'll try to make a connection between Mylobot and BHProxies (a residential proxy service), and finally we'll present the telemetry we were able to collect since we started tracking it in 2018.

The first Mylobot sample we found has a size of 106496 bytes and a compilation timestamp of October 20, 2017. At that time, the malware had three different stages, with the third stage being the actual Mylobot proxy bot payload and the one responsible for performing the network communications.

Before going into details, Figure 1 details the execution of Mylobot's samples.

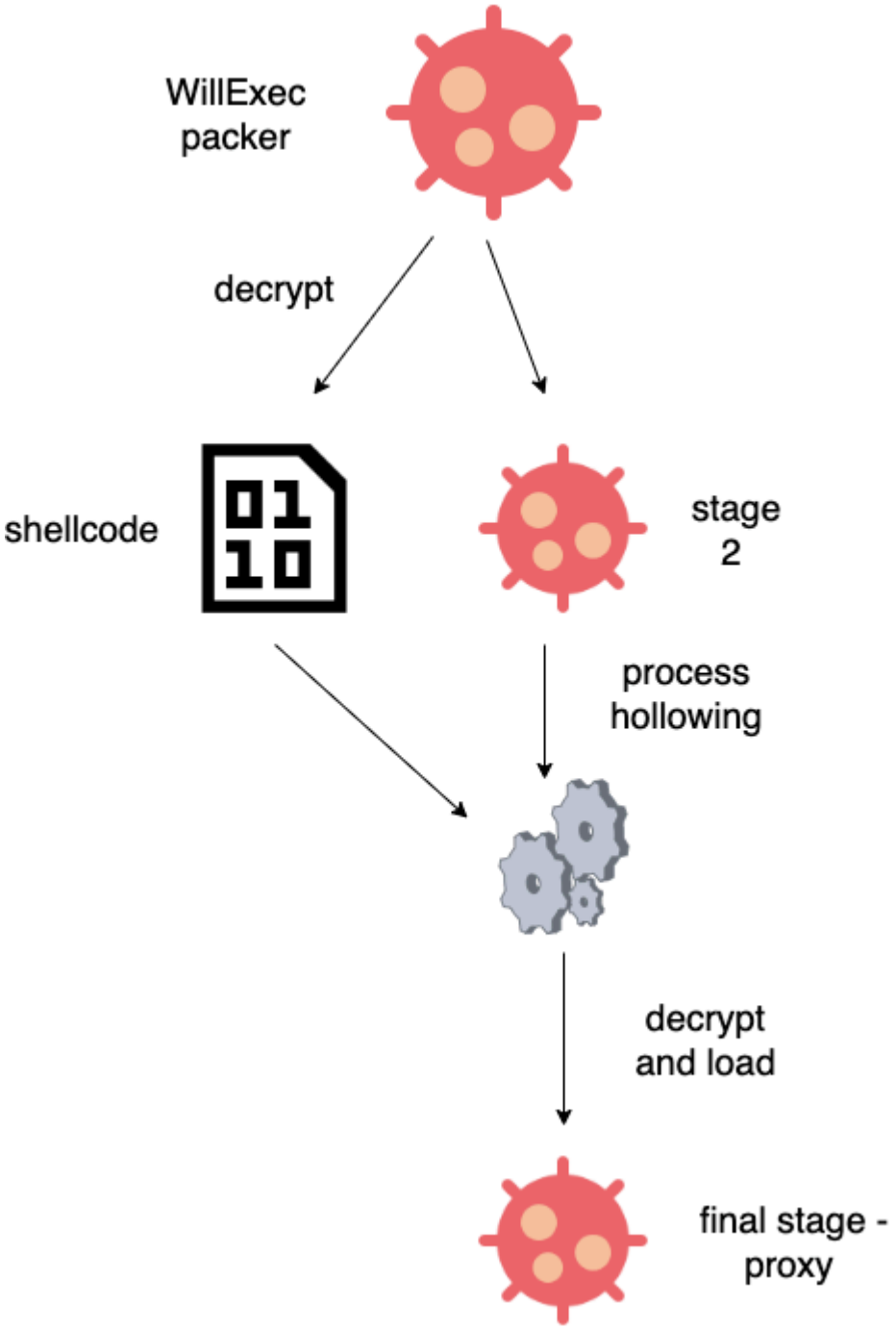


Figure 1 - Execution of Mylobot's samples

First stage - WillExec dropper

The first stage embeds an encrypted resource and performs some anti-debug checks using windows API CreateTimerQueueTimer and SetUnhandledExceptionFilter that are well-described on Minerva blog post from 2022. After those checks, the sample fetches from its resource a very long base64 encoded string and decodes it. The resulting buffer has the following structure:

```
struct decoded_res
{
  uint32_t sz_next_stage;
  uint32_t sz_shellcode;
  char rc4_key[5];
  char padding[19];
  char encrypted_blob[sz_shellcode + sz_next_stage];
}
```

In all samples we found, sz_shellcode was equal to 0x820. The function that decodes and decrypts the resource (figure 2a) and ends with the instructions listed in figure 2b, resulting in the execution of the decoded shellcode.

```
handle_exe = GetModuleHandleA((LPCSTR)0x0);
fn_load_resource(handle_exe);
ptr_VirtualAlloc = (PROC *)fn_get_api_from_hash(*(uint *)&mem->hash_VirtualAlloc);
mem->ptr_VirtualAlloc = ptr_VirtualAlloc;
iVar2 = (*mem->ptr_VirtualAlloc)(0,mem->resource_size,0x3000,4);
mem->resource_alloc = iVar2;
_memset((void *)mem->resource_alloc,0,mem->resource_size);
decoded_size = fn_b64_decode(mem->resource_ptr,mem->resource_size,mem->resource_alloc,
                             mem->resource_size);
mem->b64_decoded_size = decoded_size;
_memcpy(&mem->header,(void *)mem->resource_alloc,0x20);
iVar2 = fn_get_api_from_hash(mem->hash_InterlockedDecrement);
mem->ptr_InterlockedDecrement = iVar2;
for (local_c = 0; local_c < mem->b64_decoded_size; local_c = local_c + 1) {
}
fn_decrypt(mem->resource_alloc + 0x20,mem->b64_decoded_size - 0x20,(int)(mem->header).rc4_key,5);
mem->resource_alloc = mem->resource_alloc + 0x20;
pvVar3 = (void *)(*mem->ptr_VirtualAlloc)(0,(mem->header).sz_shellcode,0x3000,0x40);
mem->shellcode = pvVar3;
_memcpy(mem->shellcode,(void *)mem->resource_alloc,(mem->header).sz_shellcode);
mem->resource_alloc = mem->resource_alloc + (mem->header).sz_shellcode;
return;
```

Figure 2a - Decryption code

```
00401824 52          PUSH    EDX             ptr_decoded_resource
00401825 51          PUSH    ECX             ptr_shellcode
00401826 51          PUSH    ECX             ptr_shellcode
00401827 c3          RET                     int (*ptr_shellcode)(char* ptr_shellcode, char* ptr_decoded_resource)
```

Figure 2b - Epilogue of the decrypting function

The shellcode is short and straightforward and has the purpose of running the decrypted PE file following it in the decrypted blob. The shellcode creates a new process and does a process hollowing on it, in order to run the decrypted PE. The concept of process hollowing is the way of replacing an executable image at runtime with another executable. All the hollowing procedure is detailed in figure 3.

```
uVar1 = (*apis->GetCommandLineW)();
iVar2 = (*apis->CreateProcessW)
                  (local_73c,uVar1,uVar3,uVar4,uVar5,uVar6,uVar7,uVar8,puVar9,puVar10);
if (iVar2 != 0) {
  ctx.ContextFlags = 0x10007;
  iVar2 = (*apis->GetThreadContext)(process_info.hThread,&ctx);
  if (iVar2 != 0) {
    (*apis->ReadProcessMemory)(process_info.hProcess,ctx.Ebx + 8,&target_base_addr,4,0);
    if (target_base_addr == (ptr_nt_headers->OptionalHeader).ImageBase) {
      (*apis->NtUnmapViewOfSection)(process_info.hProcess,target_base_addr);
    }
    local_c = (*apis->VirtualAllocEx)
                        (process_info.hProcess,(ptr_nt_headers->OptionalHeader).ImageBase,
                         (ptr_nt_headers->OptionalHeader).SizeOfImage,0x3000,0x40);
    if (local_c != 0) {
      (*apis->WriteProcessMemory)
                (process_info.hProcess,local_c,ptr_pe,
                 (ptr_nt_headers->OptionalHeader).SizeOfHeaders,0);
      for (cpt_section = 0;
          cpt_section < (int)(uint)(ptr_nt_headers->FileHeader).NumberOfSections;
          cpt_section = cpt_section + 1) {
        section = (IMAGE_SECTION_HEADER *)
                  (ptr_pe + cpt_section * 0x28 + _ptr_pe->e_lfanew + 0xf8);
        (*apis->WriteProcessMemory)
                  (process_info.hProcess,local_c + section->VirtualAddress,
                   ptr_pe + section->PointerToRawData,section->SizeOfRawData,0);
      }
      (*apis->WriteProcessMemory)
                (process_info.hProcess,ctx.Ebx + 8,&(ptr_nt_headers->OptionalHeader).ImageBase,4
                 ,0);
      ctx.Eax = local_c + (ptr_nt_headers->OptionalHeader).AddressOfEntryPoint;
      ctx.Eip = local_c + (ptr_nt_headers->OptionalHeader).AddressOfEntryPoint;
      (*(code *)apis->SetThreadContext)(process_info.hThread,&ctx);
      (*(code *)apis->ResumeThread)(process_info.hThread);
    }
  }
}
return process_info.hProcess;
```

Figure 3 - Process hollowing

One noticeable thing is that this stage is executing multiple times the following API call

MessageBoxA(0xffffa481, "Will exec", 0, 0);

This call is failing because of the unknown HWND value 0xffffa481. We'll refer to this dropper as WillExec in the rest of this document.

Second stage

The second stage is quite straightforward as well. It contains 2 resources:

- an encrypted resource, this time unencoded (with resource ID equal to 101)
- A very small 4 bytes resource (with resource ID equal to 102)

The 4 bytes resource is a RC4 key that is used by the program to decrypt resource 101. The decrypted resource is a PE file.

Once the program has mapped the decrypted PE file in memory, it locates one of its exported functions named _ep@4, and executes it.

One interesting thing to notice in this stage is the comparison of the command line argument with the string wusaupdate:

```
pbVar1 = (byte *)GetCommandLineA();
pbVar1 = fn_strcmp(pbVar1,(byte *)"wusaupdate");
if (pbVar1 != (byte *)0x0) {
  fn_disable_windefender();
  fn_add_fw_rules();
```

Figure 4 - Command line check

If it matches, the program disables Windefender (Figure 5) and adds a firewall rule that blocks all outgoing TCP connections trying to connect to the following ports:

- 2900
- 1100
- 2200
- 3300
- 4400
- 5500
- 6600
- 7700
- 8800
- 9900

The function responsible for disabling Windefender executes a series of shell commands and changes the registry:

```
bool fn_disable_windefender(void)

{
  LSTATUS LVar1;
  bool bVar2;
  DWORD local_10;
  undefined4 local_c;
  HKEY local_8;

  ShellExecuteA((HWND)0x0,"open","cmd.exe","/C sc stop wuauserv",(LPCSTR)0x0,0);
  ShellExecuteA((HWND)0x0,"open","cmd.exe","/C sc config wuauserv start= disabled",(LPCSTR)0x0,0);
  ShellExecuteA((HWND)0x0,"open","cmd.exe",
                "/C Reg add \"HKEY_LOCAL_MACHINE\\SOFTWARE\\Policies\\Microsoft\\Windows Defender\"
                /v DisableAntiSpyware /t REG_DWORD /d 1 /f"
                ,(LPCSTR)0x0,0);
  LVar1 = RegCreateKeyExA((HKEY)0x80000002,"SOFTWARE\\Policies\\Microsoft\\Windows Defender",0,
                          (LPSTR)0x0,0,2,(LPSECURITY_ATTRIBUTES)0x0,&local_8,&local_10);
  if (LVar1 == 0) {
    local_c = 1;
    LVar1 = RegSetValueExA(local_8,"DisableAntiSpyware",0,4,(BYTE *)&local_c,4);
    bVar2 = LVar1 == 0;
    RegCloseKey(local_8);
  }
  else {
    bVar2 = false;
  }
  return bVar2;
}
```

Figure 5 - Function disabling windefender

Third stage

The third stage is the most interesting one as it's the one turning the infected computer into a proxy. Before starting its communication with the remote command and control server, the third stage writes itself on disk, then runs cmd.exe with the attribute PROCESSINFORMATION.wShowWindow equal to 0 (the program's window won't appear to the user)

Then, it will inject itself into the newly created process using the APIs WriteProcessMemory and CreateRemoteThread. More specifically, the program injects:

- itself as a raw file
- an array containing functions pointer to useful ntdll.dll and kernel32.dll apis
- a small binary blob that can be seen as a Portable Executable manual mapper.

Once the manual mapper has mapped the raw file in memory, the original process will run the exported function _re@4 in the newly created process, then terminate itself.

The binary will achieve persistence by writing itself to the following registry key:

Software\\Microsoft\\Windows\\CurrentVersion\\Run

The value stored in this registry key is the path on disk of the second stage.

The malware will remove potential other malware presence by changing the filename of executable files in multiple directories. It checks for the presence of files ending with .exe in:

- %APPDATA%
- %APPDATA%\WindowsAudio
- %APPDATA%\Windows Live
- %APPDATA%\Update
- %APPDATA%\Adobe
- %APPDATA%\WindowsUpdate
- %APPDATA%\Identities
- %APPDATA%\Microsoft
- %APPDATA%\Microsoft\Windows
- %APPDATA%\Microsoft\Windows\Themes

If an executable file is found, it will replace the .exe extension with .local.backup.Moreover, the binary will create a new process of its second stage with the command line argument wusaupdate to disable Windows defender and create the firewall rules detailed above.

Finally the binary will store an encoded FILETIME on the filesystem in %TEMP%\dd.te if the file does not exist. The malware will start communicating with the command and control server only if 12 days have passed since the date written down in this file.

Communication protocol

The first version of Mylobot had a very unique network fingerprint. The sample usually embeds more than 1000 hard-coded domains, mostly ending with top level domain (TLD) .ru or .com. All domains look like they have been generated by some sort of domain generation algorithm (DGA). An overview of some of the hardcoded domains:

- zdrussle.ru:2173
- pseyumd.ru:5492
- stydodo.ru:2619
- tqzknrx.com:1123
- mdcqrxw.com:4984
- tpwtgyw.com:9631
- cnoyucn.com:9426
- qhloury.com:4759
- fnjxpwy.com:3863
- csxpzlz.com:5778
- wlkjopy.com:8778
- mynfwwk.com:8427
- uuitwxg.com:6656
- agnxomu.com:8881
- wcagsib.com:3547

- fmniltb.com:9582
- oapwxiu.com:3922

For each of these domains, the sample tries to connect to many of its subdomains. Most subdomains will start with the letter x, w, or m, followed by a number. In this sample, the first hardcoded domain is fywkuzp[.]ru:7432, and we could observe a infected machine trying to connect to the following domains:

- m1.fywkuzp[.]ru:7432
- m2.fywkuzp[.]ru:7432
- …
- m42.fywkuzp[.]ru:7432

In the end, Mylobot produces thousands of DNS requests, which makes it quite noisy. If the sample successfully connects to one of those domains, it keeps the connection open and waits for an instruction from the command and control server (C2).

When Mylobot receives an instruction from the C2, it transforms the infected computer into a proxy. The infected machine will be able to handle many connections and relay traffic sent through the command and control server.

Here's a list of the different instructions supported by Mylobot:

Message ID (msg_id)	**Description**
1	Connect to an IP:port
2	Close connection (specified by its ID (data)
3	Send data to a connected IP/domain:port (specified by its ID (data)
4	Restart the client networking stuff
5	End all active connections
6	Echo
7	Download a binary using HTTP

8	Download multiple binaries using HTTP + delay (8 hours)
17	Connect to an domain:port
19	Force re-read from socket (specified by its ID)

A typical message exchanged between the C2 and the client has the following structure:

```
struct msg
{
  uint32_t conn_id;
  uint8_t msg_id;
  char* msg_data;
}
```

The figure below shows an example of a C2 instruction telling the infected machine to connect to google.com on port 443.

Figure 5a - A message from the command and control

It has been reported by Lumen that, at that time, infected computers were receiving samples of Khalesi or Zusy malwares using the msg_id 7 and 8. We only observed self binary updates through those commands, as well as new versions of its downloader (see section below).

Since the samples we found contain many hardcoded domain names, we started to monitor them to get infection telemetry and get an idea about the botnet size.

In the end, Mylobot is nothing more than a proxy bot, with some ability to download and run other samples. The sample that is implementing the proxy functionality will be referred to as "Mylobot's proxy bot" in the rest of this

post.

In 2018, we started seeing Mylobot's proxy bot being distributed by a new malware sample.
This new binary presents a lot of similarities with Mylobot, but the last stage acts as a downloader. Let's begin by exposing the similarities between this new sample and Mylobot's proxy bot sample, and describe how it is downloading Mylobot on an infected system.

The first thing to note is that the new sample uses WillExec, the same dropper used for Mylobot samples. The dropped file performs anti virtual machine checks, and tries to remove other malware running on the system as well, after that, it connects to its command and control, and downloads the next stages. This sample has been well described by Minerva, so we won't go into too much detail.

The downloader has a huge list of hard-coded encrypted command and control domains (more than 1000).

```
AFE38E8A2F7B385FBD2B3A74CFDCFF68289562F65B78909FB065D6D26ECFE378
DB5100020F8BE37724E02A19ACDCFD4F289562F65B78909FB065D6D26ECFE378
0216E8A02258AE24F3C33449083DAC90289562F65B78909FB065D6D26ECFE378
B96CD43A92AADED9B36DA8C0A4B08117289562F65B78909FB065D6D26ECFE378
1136E2358547702FFDE67DBACD73BAD3289562F65B78909FB065D6D26ECFE378
7733E9373CFDDC407CBF55E3B7E6E47A289562F65B78909FB065D6D26ECFE378
89283A92DB79E124EF9FBCAB67CD4713289562F65B78909FB065D6D26ECFE378
8DE63CBFF893137911D064C2035551F8289562F65B78909FB065D6D26ECFE378
097FCD2588B57E92AD7B483F2FA7E505289562F65B78909FB065D6D26ECFE378
A05A265913DD68E1A08B6CC43576463A289562F65B78909FB065D6D26ECFE378
2C2D00DF7EEFB14F2ED871314BEEE36F289562F65B78909FB065D6D26ECFE378
BB97D194655B00B8587D8CFC4AEA5897289562F65B78909FB065D6D26ECFE378
90E125FB21E6748593633DBDFECCA8FC289562F65B78909FB065D6D26ECFE378
```

Figure 6 - Hardcoded encrypted domain names

Those domains are AES-ECB encrypted with the key GD!brWJJBeTgTGSgEFB/quRcfCkBHWgl, and have probably been generated using the same DGA mentioned in the first part of this post. Indeed, there's a strong similarity between Mylobot downloader's domain names and Mylobot proxy's ones.

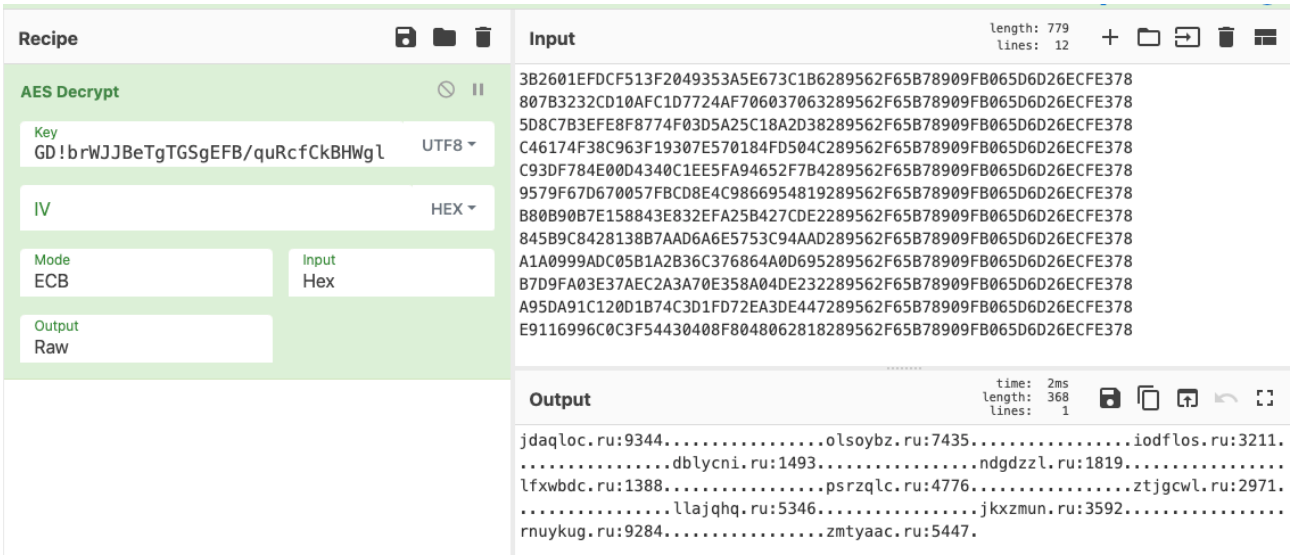


Figure 7 - Decryption of domain names

The downloader decrypts the domains at runtime, and tries to connect to the subdomain buy1, v1 or up1 (depending on the sample) of those domains. In the end, it tries to connect to the following domains:

- v1.flkpuod[.]ru:5796
- v1.iqaagar[.]ru:2919
- v1.fchbwme[.]ru:7533
- …

The command and control server responds with an AES encrypted message that, when decrypted, contains a link to download the next stage.

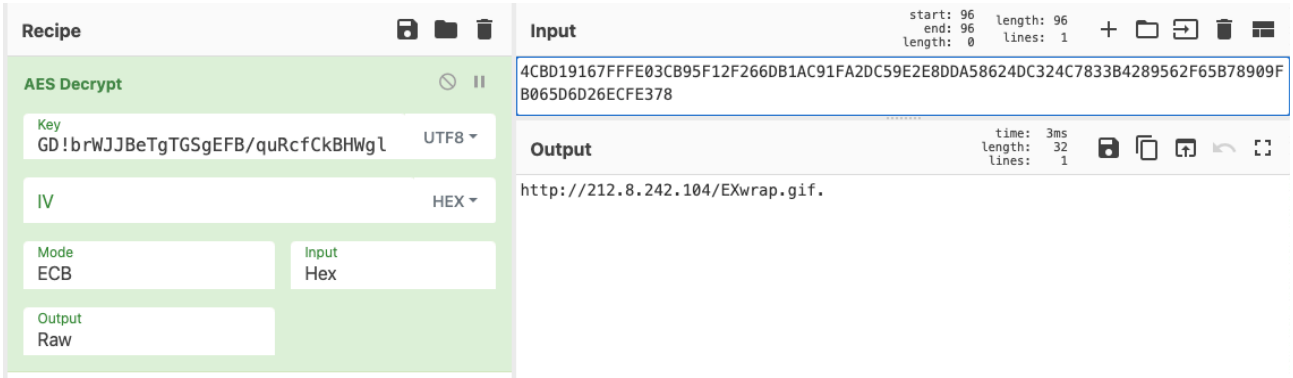


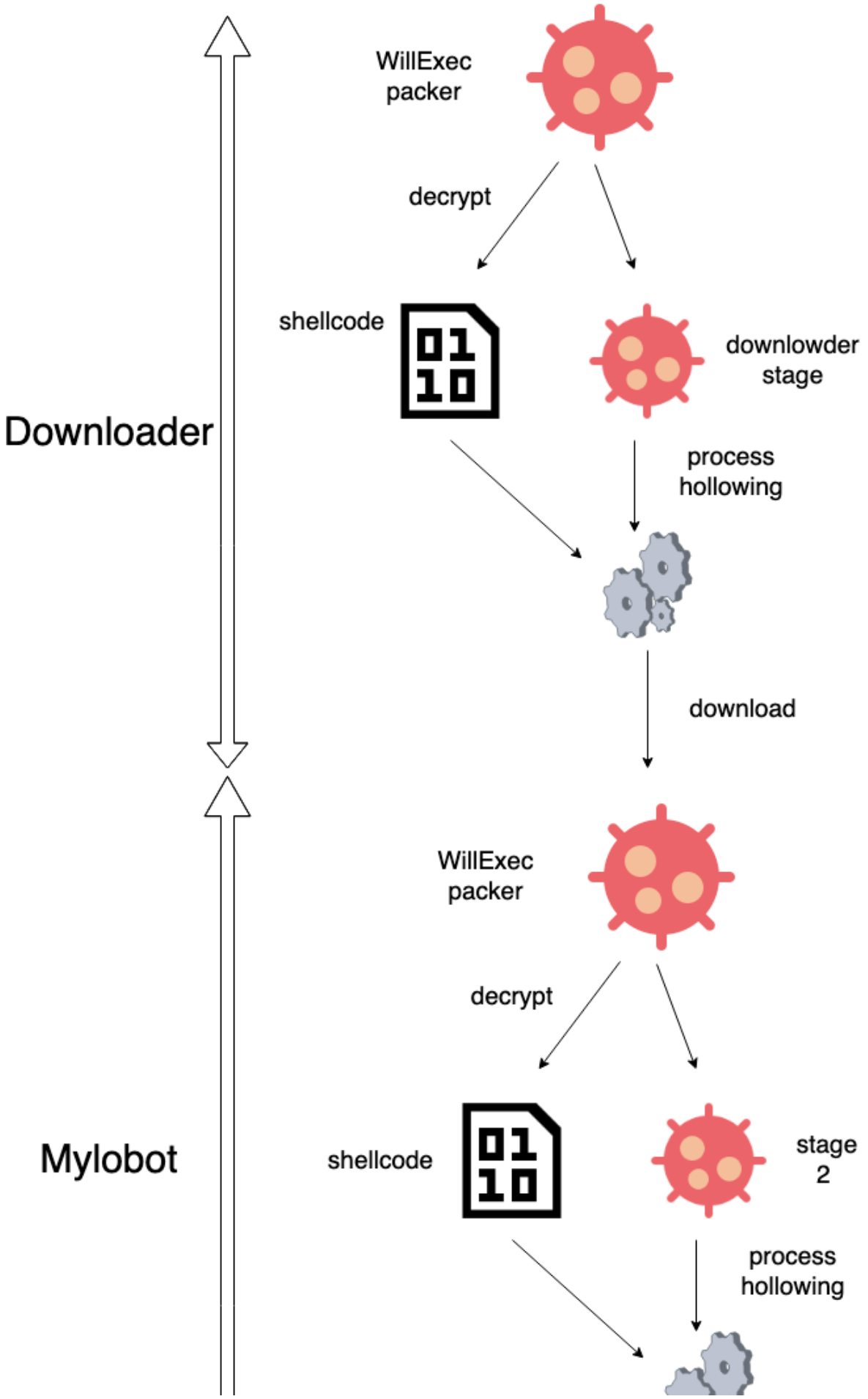
Figure 8 - Decryption of the response from the command and control

Once again, as the sample we found contains many hardcoded domain names, we started to monitor some of them to get an overview of the botnet.

The downloaded payload is a Mylobot sample, embedded in a WillExec dropper, the same way it was distributed in 2017. We've seen Mylobot's downloader distributing other samples than Mylobot's proxy bot (Minerva is

showing an example), but it was quite rare.

The distribution has evolved in the way described in figure 9.


Downloader
WillExec packer
decrypt
shellcode
01 10
downlowder stage
process hollowing
download
Mylobot
WillExec packer
decrypt
shellcode
01 10
stage 2
process hollowing

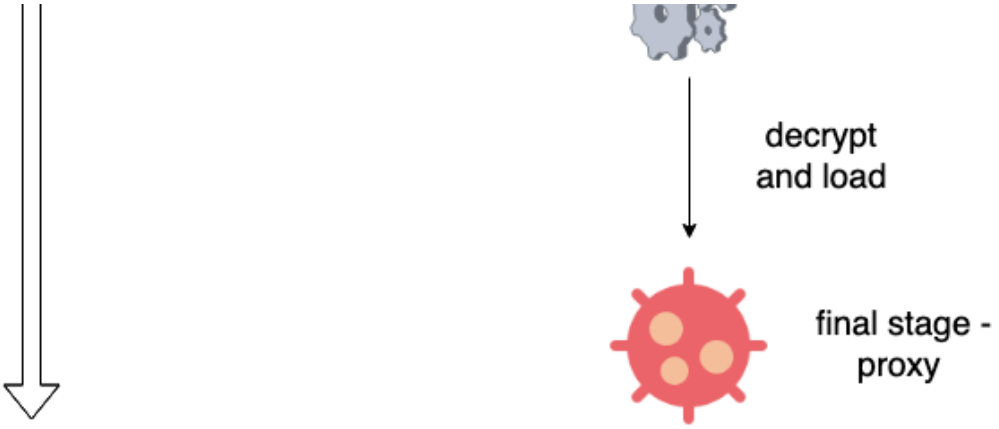


Figure 9 - Execution of the downloader sample downloading Mylobot

Regarding the Mylobot sample that is downloaded, it hasn't evolved much over the years. The only major change is the number of command and control domains hardcoded in the binary, that evolved from ~1000 in the first versions, to only 3 since the beginning of 2022:

- fywkuzp[.]ru:7432
- dealpatu[.]ru:8737
- rooftop7[.]ru:8848

We decided to have an overview of Mylobot's infrastructure. We started by looking at the 3 domain names used in the last version of the proxy. Starting with the domain fywkuzp[.]ru, we looked at the IP addresses that were pointed by the subdomains from m0.fywkuzp[.]ru to m42.fywkuzp[.]ru. We were able to identify 25 IP addresses that were used between 2017 and 2022, all associated with cloud providers from Netherlands (worldstream.nl), Lithuania (cherryservers.com) and Latvia (bite.lv).

We looked for other domain names pointing to those IP addresses. Unsurprisingly, we found other fywkuzp[.]ru subdomains that were using other prefixes (w5.fywkuzp[.]ru, x6.fywkuzp[.]ru,...), and other Mylobot domains (pseyumd[.]ru, stydodo[.]ru, zdrussle[.]ru) as well.

One domain caught our attention in our research because of its name: clients.bhproxies[.]com. From June 22, 2016 to September 17, 2017, this domain resolved to the IP address 46.166.173.180. The next day (September 18th 2017), clients.bhproxies[.]com started to resolve to 109.236.80.135, and up1.pseyumd[.]ru (which is a domain used by Mylobot) started to resolve to 46.166.173.180. During the following months, many Mylobot domains will resolve to this IP address.

2017-12-27 09:52:28	2019-04-04 20:57:33	788211	m24.fywkuzp.ru.	A	46.166.173.180
2017-11-05 09:46:29	2017-12-27 10:25:42	142008	x40.fywkuzp.ru.	A	46.166.173.180
2017-10-12 08:07:01	2017-11-07 08:54:46	25673	up0.pseyumd.ru.	A	46.166.173.180
2017-10-07 09:34:39	2017-10-14 10:04:13	1324	w0.fywkuzp.ru.	A	46.166.173.180
2017-09-18 13:41:39	2017-09-24 17:50:25	22609	up1.pseyumd.ru.	A	46.166.173.180
2016-06-22 22:18:11	2017-09-17 17:05:03	726	clients.bhproxies.com.	A	46.166.173.180

Figure 10 - History of reverse DNS lookup for the ip 46.166.173.180

The website bhproxies[.]com is pretty explicit: it provides a service of "Backconnect residential proxies", with IP addresses from all over the world. They mention that they could provide custom packages to clients, with up to 150,000 unique IP addresses.

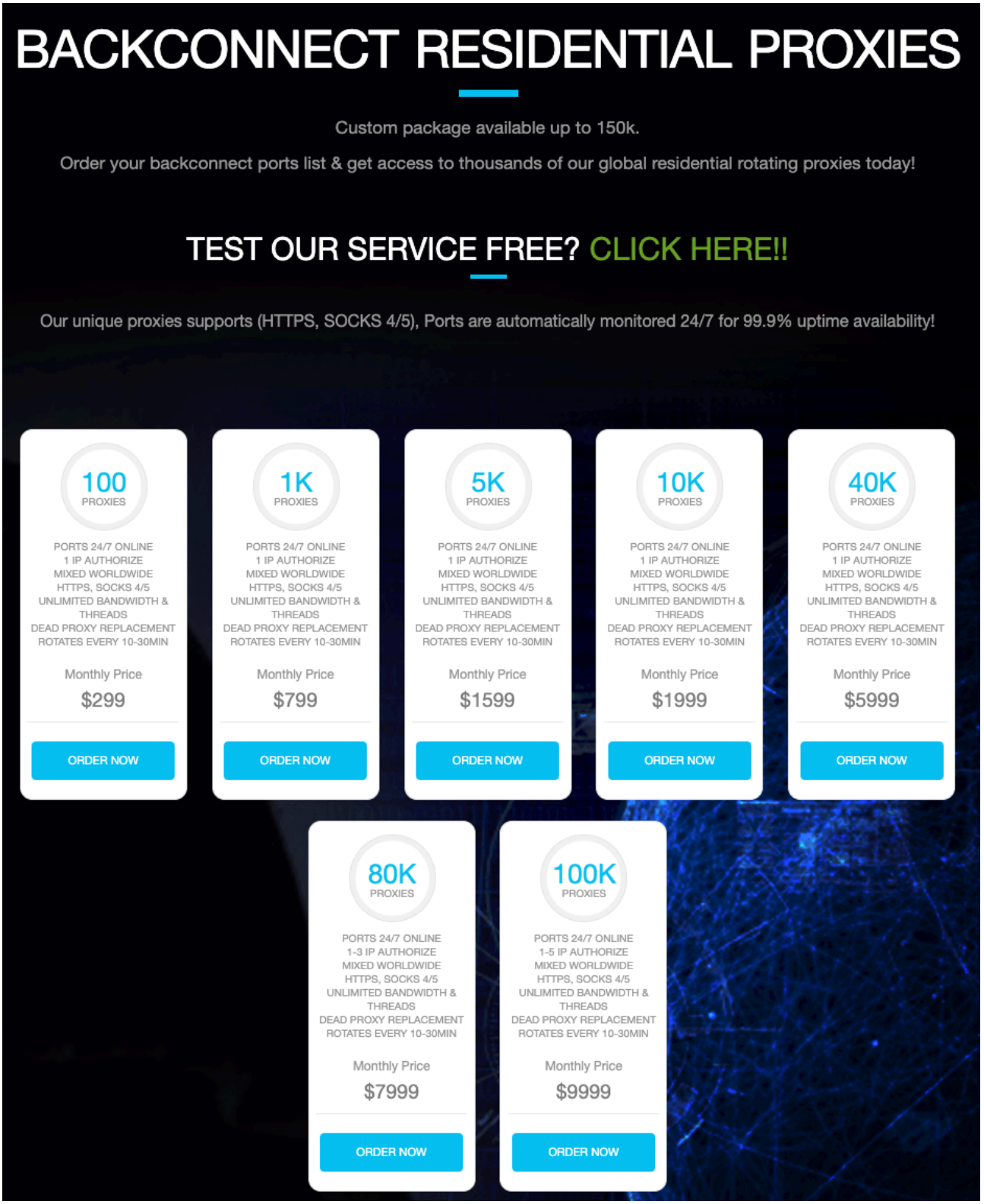


Figure 11 - BHProxies website

When looking for BHProxies on search engines, a post from BlackHatWorld forum shows up. This post was written on May 12, 2014. The post author using the alias BHProxies, is still very active on this post, as his last messages were posted in December 2022. The author promotes bhproxies[.]com, and provides a Telegram channel and a Skype account to discuss with the potential customers. He also provides a free trial to convince users to buy his service. The free trial is available at the address http://clients.bhproxies.com/panel/trial.php

At this point, we cannot prove that BHProxies is linked to Mylobot, but we have a strong suspicion, since Mylobot and BHProxies used the exact same IP 46.166.173.180 on an interval of 24 hours.

To confirm our hypothesis, we tested the trial version of BHProxies. Once you enter your public IP address on the service, you receive an IP address and a list of ports to connect to.

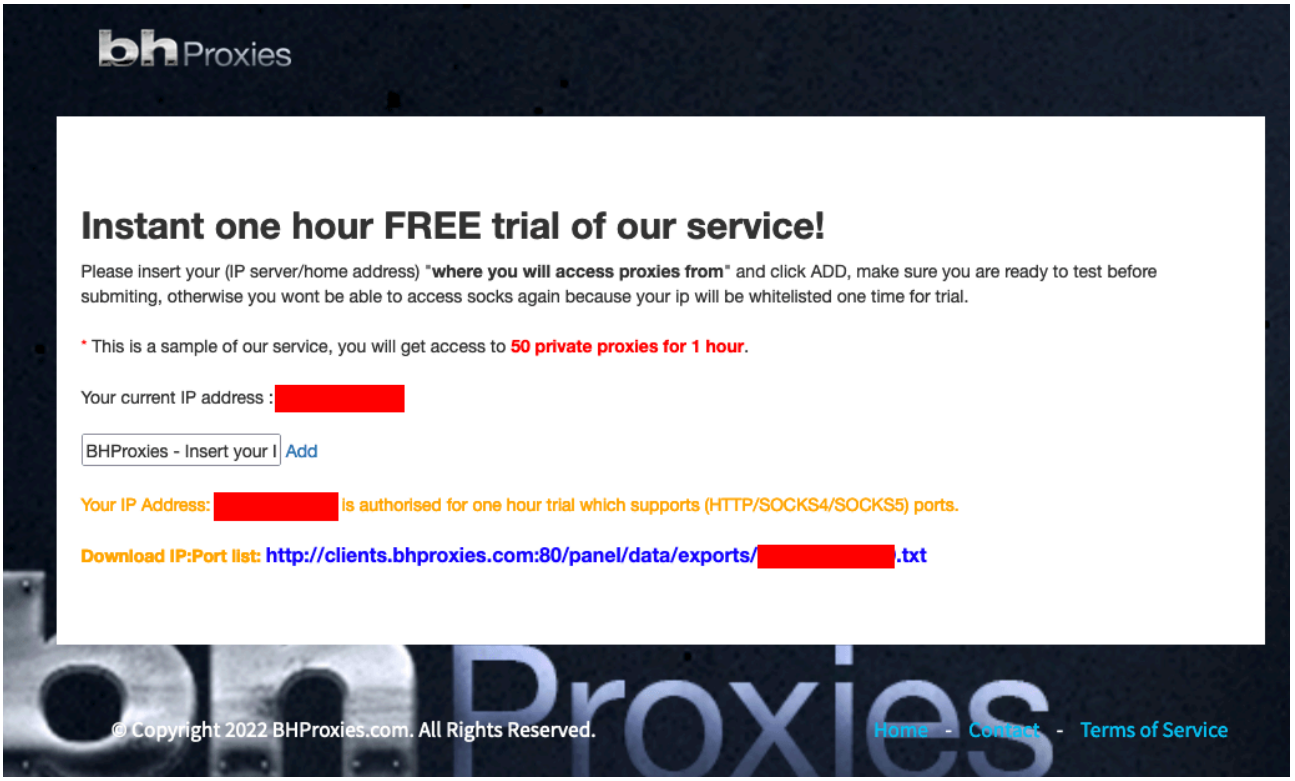


Figure 12 - BHProxies free trial

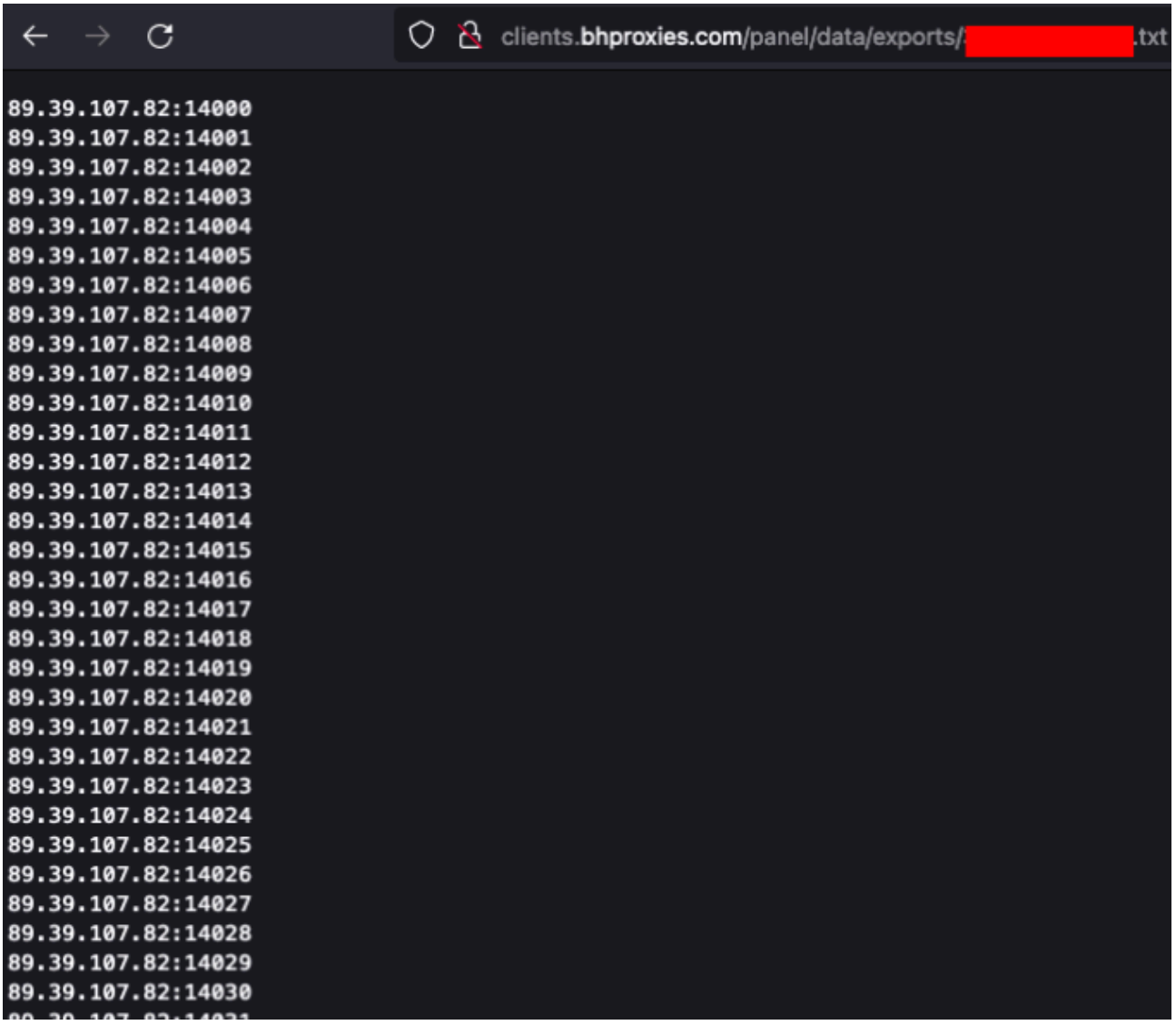


Figure 13 - BHProxies trial proxies

Each of those pairs of IP and port is a frontend for the residential proxies of BHProxies. We recovered the residential proxies IP addresses by performing an HTTP request to a server we control.

On the 50 frontend proxies provided, we were able to perform a HTTP request for 48 of those. Among these 48 recovered residential proxies IP addresses, 28 (58.3%) of those were already present in our sinkhole systems, associated with the Mylobot malware family. This number is probably higher, but we don't have a full visibility of the botnet. This gave us clear evidence that Mylobot infected computers are used by the BHProxies service.

The trial proxies list is another indicator of the strong ties between BHProxies and Mylobot, since m41.fywkuzp[.]ru, one of Mylobot proxy C2, resolves to IP 89.39.107[.]82:

Time First Seen ⇄	Time Last Seen ⇄	Count	Bailiwick	RRName ↔	RRType	RData
2020-09-24 10:41:26	2023-02-08 08:09:59	2183921	fywkuzp.ru.	m41.fywkuzp.ru.	A	89.39.107.82

We started sinkholing Mylobot in November 2018. At that time, Mylobot's proxy sample contained a lot of hardcoded DGA domains, so we were able to observe the majority of the botnet. It had led us to a maximum of 250,000 unique daily infected machines in the beginning of 2020.

Since the beginning of 2022, we're not able to get infection telemetry from the latest Mylobot version as the sample doesn't contain unregistered DGA domains anymore. Instead, we started monitoring Mylobot downloader's domains and continue to see the evolution of Mylobot's botnet.
We are currently seeing more than 50,000 unique infected systems every day, but we believe we are only seeing part of the full botnet, which may lead to more than 150,000 infected computers as advertised by BHProxies' operators.

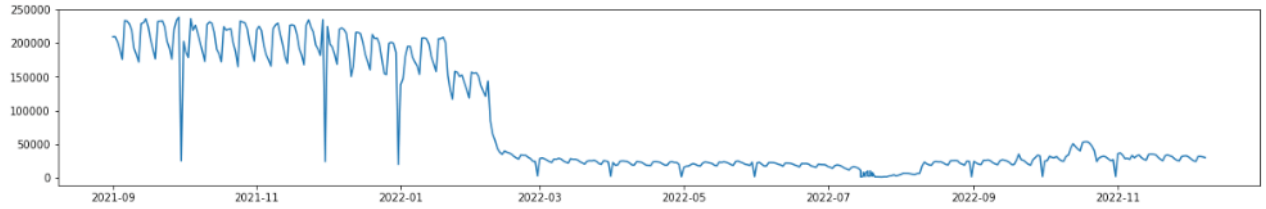


Figure 14 - Unique mylobot's infected system per day

Figure 15 shows the countries where the most computers infected with Mylobot are found. India appears to be the most targeted country, followed by the US, Indonesia, then Iran.

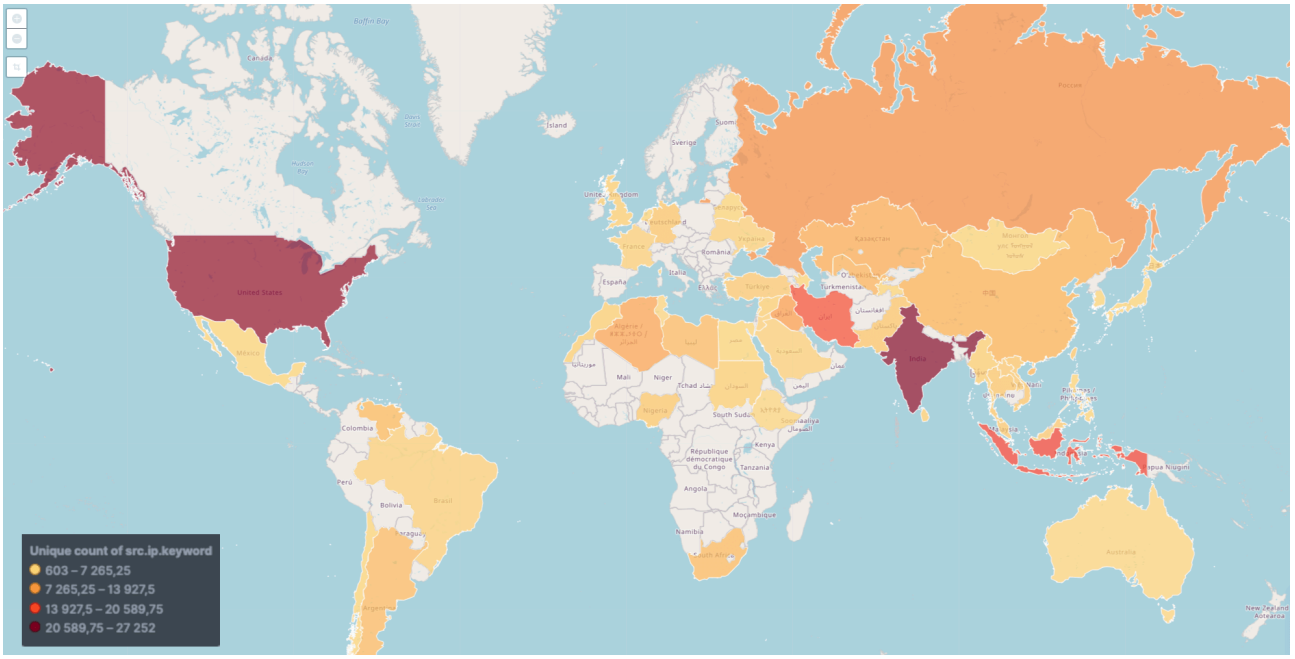


Figure 15 - Heat map of infected computers

Mylobot proxy

84733af3b60b966042d5cd17e12fd8d90650e0731297d203bd913dc5c663b91c
11fc02dd825c8e67d58cc40a47e3f4c572097bd58c6aae80591a5fb73b9167f2
392f1054815c5f805d50b60ea261210012bdda386158a1da92d992a929eb77c2
03b2164da6318fff63b6cad2fc613c3d885bd65432a7b8744c2b1709f2f9a479
69a36e6f12b4e9b9cd15528a068385f2311b0c540336c142aabdd73c2a2e2015
a63a5639d0cb6a10f7af5bd0dd30ca1800958a0f5bb47f358b6d37f51d0f0a31
2ae61c8c2a8e83cde33f38b89599032a6fb455256aa414a15f2724c94d3460d2
40cfb7b7fad1602276ebf3fa63514ba91be6186d5d3bd190f593bdec0b6d8d64

Mylobot downloader

cfde42903367d77ab7d5f7c2a8cfc1780872d6f1bfac42e9c2577dfd4b6cdeb2
fcdb7247aa6e41ff23dc1747517a3682e5a89b41bfd0f37666d496a1d3faa4ba
ad53ad1d3e4ac4cc762f596af8855fd368331d9da78f35d738ae026dd778eb9f

Mylobot proxy C2 IPs

89.39.105.47
89.38.96.140
89.38.96.14
217.23.12.80
178.132.3.12
168.119.15.229
89.38.98.48
49.12.128.181
37.48.112.111
109.236.82.28
49.12.128.180
144.76.8.93
194.88.106.18
95.211.203.197
89.39.104.201
95.168.169.43
95.211.198.102
91.229.23.112
217.23.13.104
95.211.140.149
62.112.11.245
178.132.2.82
116.202.114.236
217.23.12.50
89.39.104.58

89.38.98.47
194.88.105.108
109.236.83.166
109.236.91.239
89.39.107.92
190.2.134.165
217.23.8.12
89.39.104.62
89.39.107.82
