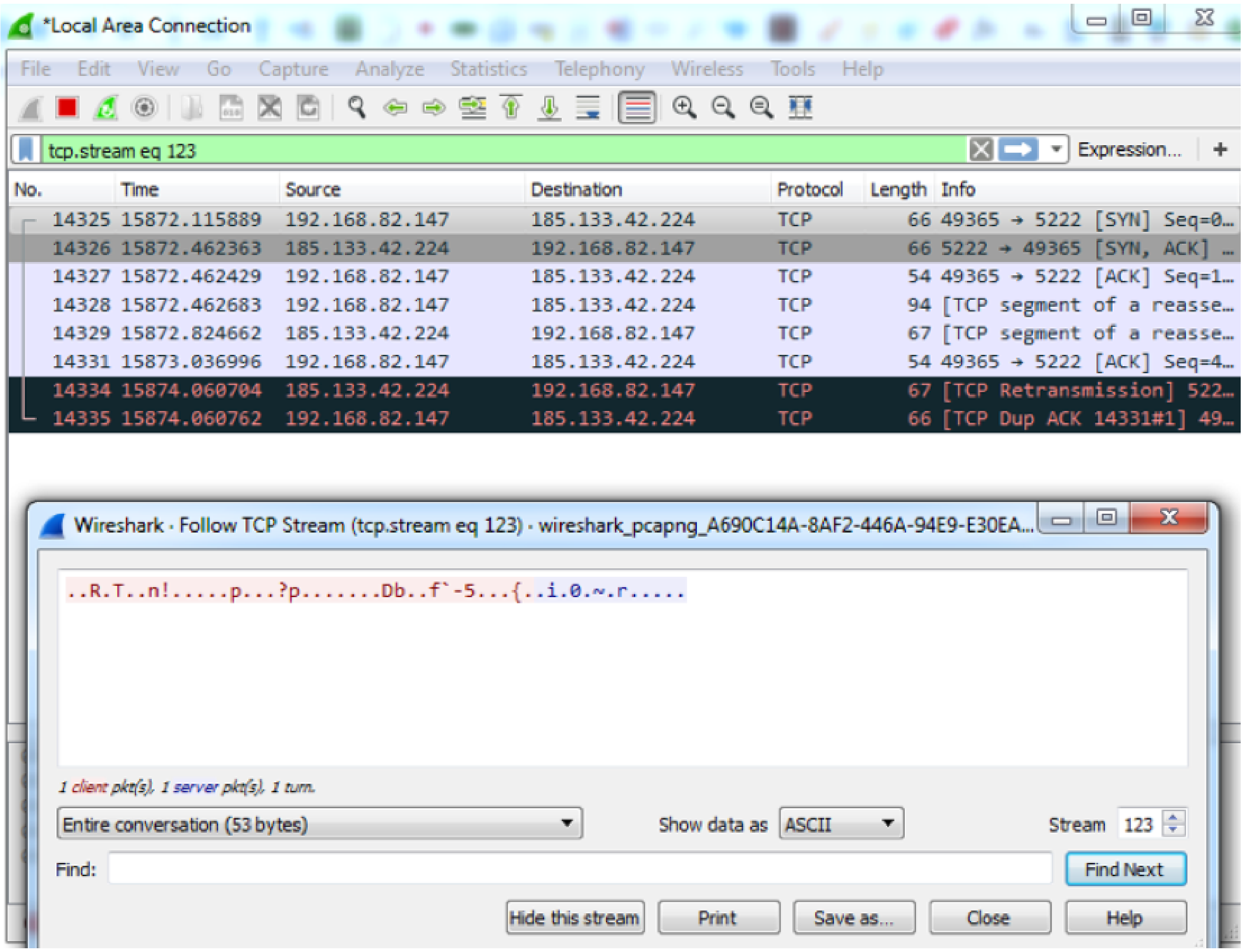
Necurs Proxy Module With DDOS Features

Written by Sofia Luis February 24, 2017 Share Facebook Twitter LinkedIn

Necurs is a malware that is mainly known for sending large spam campaigns, most notably the Locky ransomware. However, Necurs is not only a spambot, it is a modular piece of malware that is composed of a main bot module, a userland rootkit and it can dynamically load additional modules.

Having been around for a few years, this malware has been subject to a lot of great research that has covered many of its aspects including the rootkit, the DGA, the communication protocol and the spam module. However, there has not been much public information on any modules loaded by Necurs besides the spam module.

About six months ago we noticed that besides the usual port 80 communications, a Necurs infected system was communicating with a set of IPs on a different port using, what appeared to be, a different protocol. The following image shows an example of this network traffic.


*Local Area Connection
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
tcp.stream eq 123
Expression…
+

No.	Time	Source	Destination	Protocol	Length	Info
14325	15872.115889	192.168.82.147	185.133.42.224	TCP	66	49365 → 5222 [SYN] Seq=0…
14326	15872.462363	185.133.42.224	192.168.82.147	TCP	66	5222 → 49365 [SYN, ACK] …
14327	15872.462429	192.168.82.147	185.133.42.224	TCP	54	49365 → 5222 [ACK] Seq=1…
14328	15872.462683	192.168.82.147	185.133.42.224	TCP	94	[TCP segment of a reasse…
14329	15872.824662	185.133.42.224	192.168.82.147	TCP	67	[TCP segment of a reasse…
14331	15873.036996	192.168.82.147	185.133.42.224	TCP	54	49365 → 5222 [ACK] Seq=4…
14334	15874.060704	185.133.42.224	192.168.82.147	TCP	67	[TCP Retransmission] 522…
14335	15874.060762	192.168.82.147	185.133.42.224	TCP	66	[TCP Dup ACK 14331#1] 49…

Wireshark · Follow TCP Stream (tcp.stream eq 123) · wireshark_pcapng_A690C14A-8AF2-446A-94E9-E30EA…

```
..R.T..n!.....p...?p.......Db..f`-5...{..i.0.~.r.....
```

1 client pkt(s), 1 server pkt(s), 1 turn.

Entire conversation (53 bytes)   Show data as ASCII   Stream 123

Recently, while decrypting the C2 communication of the a Necurs bot, we observed a request to load two different modules, each with a different parameter list. The following is the decrypted translation of the modules section of the C2 response:

```
"module_id: EB76FF144CB59D0E4727F15970DC7E1EFFF52F49 params: http://158.255.6.204/forum/module.php
  \nhttp://178.159.43.78/forum/module.php\nhttp://185.127.26.236/forum/module.php\n
  http://185.133.42.224/forum/module.php\nhttp://185.22.172.209/forum/module.php\n
  http://185.31.162.135/forum/module.php\nhttp://185.39.148.28/forum/module.php\n
  http://194.63.140.115/forum/module.php\nhttp://212.109.192.231/forum/module.php\n
  http://217.12.203.33/forum/module.php\nhttp://5.39.219.216/forum/module.php\n
  http://77.222.54.239/forum/module.php\nhttp://91.234.34.187/forum/module.php\n
  http://95.215.111.172/forum/module.php"
```

```
"module_id: D6774D17970D69FFD56DC57921EE289210ADAF37 params: 158.255.6.204:5222\n
  178.159.43.78:5222\n185.127.26.236:5222\n185.133.42.224:5222\n185.22.172.209:5222\n
  185.31.162.135:5222\n185.39.148.28:5222\n194.63.140.115:5222\n212.109.192.231:5222\n
  217.12.203.33:5222\n5.39.219.216:5222\n77.222.54.239:5222\n91.234.34.187:5222\n
  95.215.111.172:5222"
```

The first one was the spam module for which Necurs is most known, and the parameters are the C2 addresses from which it can receive new spam campaigns. The second one was an unknown module and, judging by the parameter values, it was the one responsible for the communication we were seing to port 5222.

We noticed this module in September 2016 and the compilation timestamp on the module is “Aug 23 2016”, which suggests the module started being used around then. However, it is possible that another version of the same module had been deployed before, without being noticed.

We downloaded the module and reverse engineered it to try to understand exactly what it was. At first look, it seemed to be a simple SOCKS/HTTP proxy module, but as we looked at the commands the bot would accept from the C2 we realized that there was an additional command, that would cause the bot to start making HTTP or UDP requests to an arbitrary target in an endless loop, in a way that could only be explained as a DDOS attack. This is particularly interesting considering the size of the Necurs botnets (the largest one, where this module was being loaded, has over 1 million active infections each 24 hours, we blogged about it here). A botnet this big can likely produce a very powerfull DDOS attack.

Please notice that **we have not seen Necurs being used for DDOS attacks**, we simply saw that it has that capability in one of the modules that it has been loading.

The rest of this post contains the results of a technical analysis of this module, detailing its C2 protocol, the SOCKS/HTTP proxy features, and the DDOS attack features.

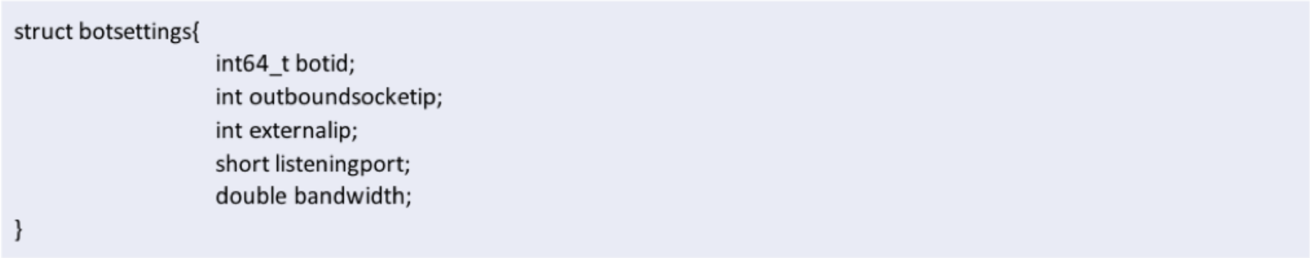
MODULE TECHNICAL ANALISYS

Module start/initialization

Once the module is loaded by the bot, it performs the following initialization actions:

- Parses the parameters and stores them in an internal **list of C2 addresses**;
- **Fills a memory structure** (see *botsettings* struct definition below) with:
    - **The BotID** - Generated through gathering unique system characteristics;
    - **The internal IP address** - Obtained by checking the outbound sockets IP address when connecting to google.com;
    - **The external IP address** - Obtained trough HTTP from ipv4.icanhazip.com or checkip.dyndns.org;
    - **The available bandwidth** - Obtained by measuring the download speed of the Windows 7 Service Pack 1 file from microsoft;
    - **The (socks/http) proxy service port** - The port of the service listening on a random port above 1024;
- **Checks if the system is behind NAT** - By checking if the outbound socket IP is not a local address and that it matches the external IP;
- If the system is not behind NAT, the bot **starts a SOCKS/HTTP proxy service** listening on a random port above 1024.

The *botsettings* struct can be defined as follows:



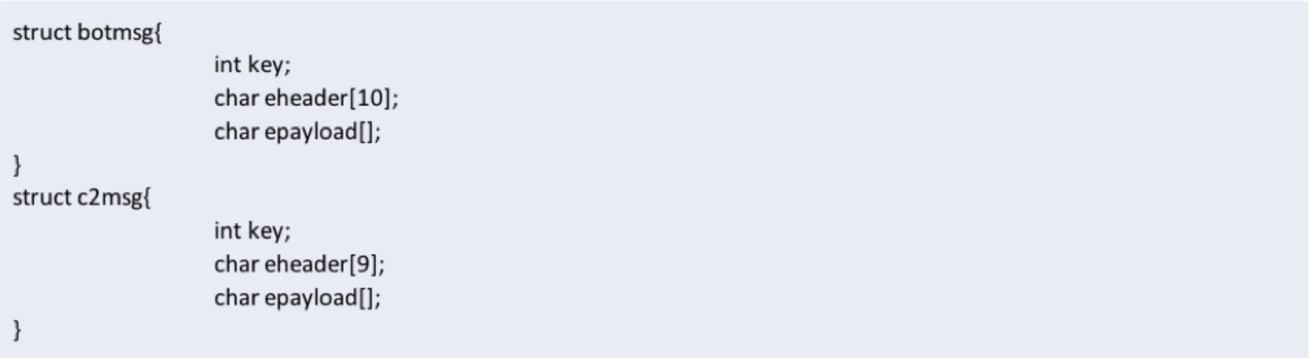
```
struct botsettings{
                    int64_t botid;
                    int outboundsocketip;
                    int externalip;
                    short listeningport;
                    double bandwidth;
}
```

C2 communication protocol

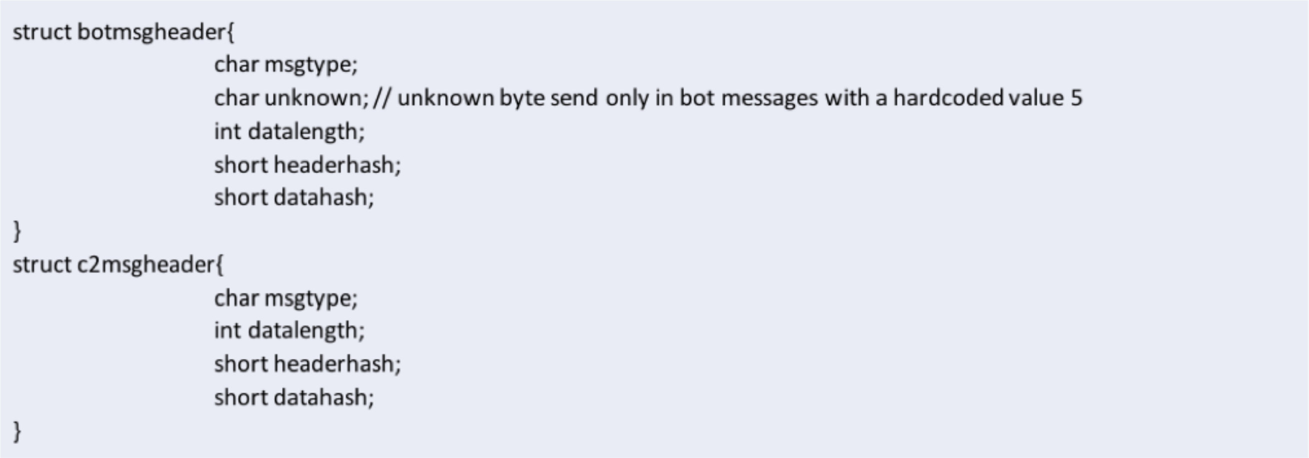
After initialization, the bot will enter the main C2 connection loop, where it will attempt to connect to the current C2 every 10 seconds, unless instructed otherwise. If the connection to the current C2 fails, it will fetch another from the address list and try again.

The communication protocol is binary and encrypted/obfuscated using a custom algorithm. Messages to, and from, the server have a very similar structure (see struct *botmsg* and *c2msg* definitions below) and contain the following data:

- **Key** - A 32 bit encryption key;
- **Encrypted header** - A header structure (see struct *botmsgheader* and *c2msgheader* definitions below), encrypted with the key and containing:
    - **Message type** - A byte that defines the type of message/command being sent;
    - **Payload length** - The length of the payload being sent;
    - **Header hash** - A hash of the first bytes of the message (key, msgtype, unknown and datalength);
    - **Data hash** - A hash of the payload, used for integrity checking;
- **Encrypted payload** - An array of data being sent, encrypted with the reverse value of the key.



```
struct botmsg{
                    int key;
                    char eheader[10];
                    char epayload[];
}
struct c2msg{
                    int key;
                    char eheader[9];
                    char epayload[];
}
```



```
struct botmsgheader{
                    char msgtype;
                    char unknown; // unknown byte send only in bot messages with a hardcoded value 5
                    int datalength;
                    short headerhash;
                    short datahash;
}
struct c2msgheader{
                    char msgtype;
                    int datalength;
                    short headerhash;
                    short datahash;
}
```

There are three types of messages sent by the bot to the C2, that can be distinguished by the *msgtype* byte in the header:

- **Beacon** (msgtype 0) - This is the main message that is sent by the bot every 10 seconds. It sends the *botsettings*struct described previously as a payload;
- **Connectivity check** (msgtype 1) - This is a simple dummy message that contains no data besides the encrypted message header. It is sent in case a timeout occurs to the current C2 to make sure it is no longer available;
- **Proxybackconnect** (msgtype 2) - This message is sent in case the bot receives the command to start a socks backconnect. This will start a connection to the C2 that will be reused for the SOCKS/HTTP proxy connection, proceeding as if it had been initiated by the proxy client.

As a response to the beacon, there are also three types of messages (or commands) sent by the C2 to the bot, that can be distinguished by the msgtype byte in the header:

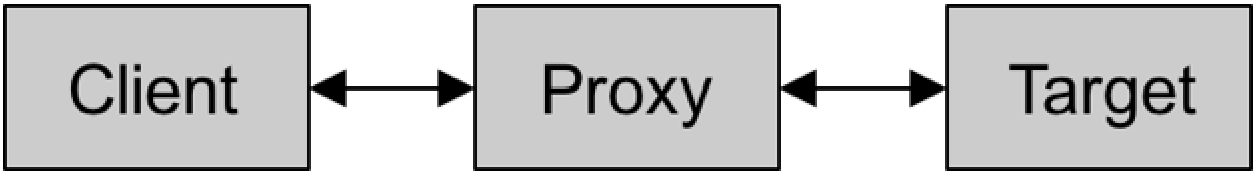
- **Start Proxybackconnect** (msgtype 1) - This command tells the bot to start a *backconnect* proxy session by sending a *proxybackconnect* message from the bot to the C2. This connection's socket will be reused and allows the bot to be used as a proxy even behind a firewall and without establishing a direct connection to it;
- **Sleep** (msgtype 2) - This will cause the bot to sleep for 5 minutes;

- **Start DDOS** (msgtype 5) - This command will start a DDOS attack against the target specified in the message payload using one of two available attack modes:
    - **HTTPFlood:** If the first bytes of the message payload are the string “http:/”, the bot will start an HTTP flood attack against the target;
    - **UDPFlood:** If the first bytes of the message payload are **not** the string “http:/”, the bot will start an UDP flood attack against the target.

Proxy features

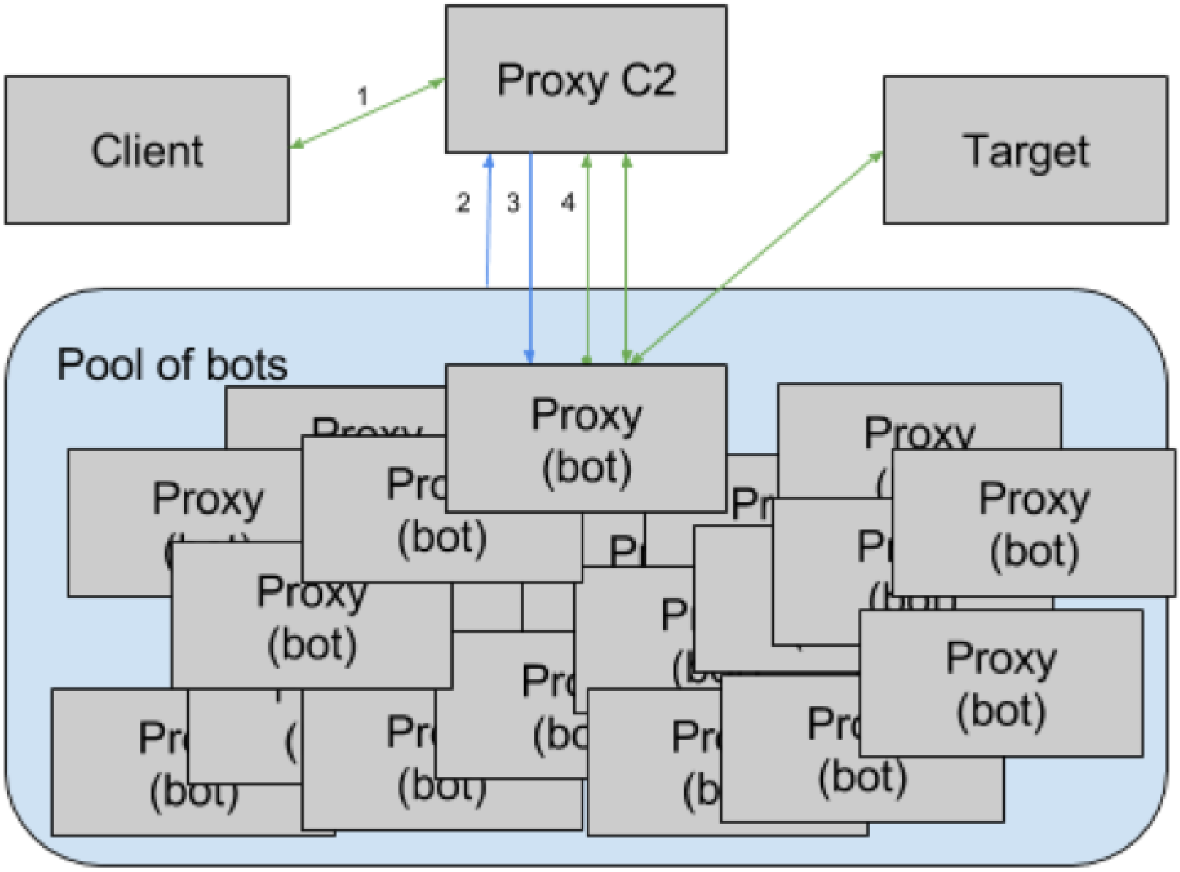
The SOCKS/HTTP proxy service and command, allows the botnet owners to use the compromised bots as proxies (HTTP, SOCKSv4 and SOCKSv5 protocols), relaying connections through them in two modes of operation (direct proxy and proxy backconnect).

In direct proxy mode, the client connects to the proxy service, which will forward the communication to the destination, as in the following image:



This is only possible if the bot is not protected behind NAT or a firewall, which is not the case in the vast majority of the botnet.

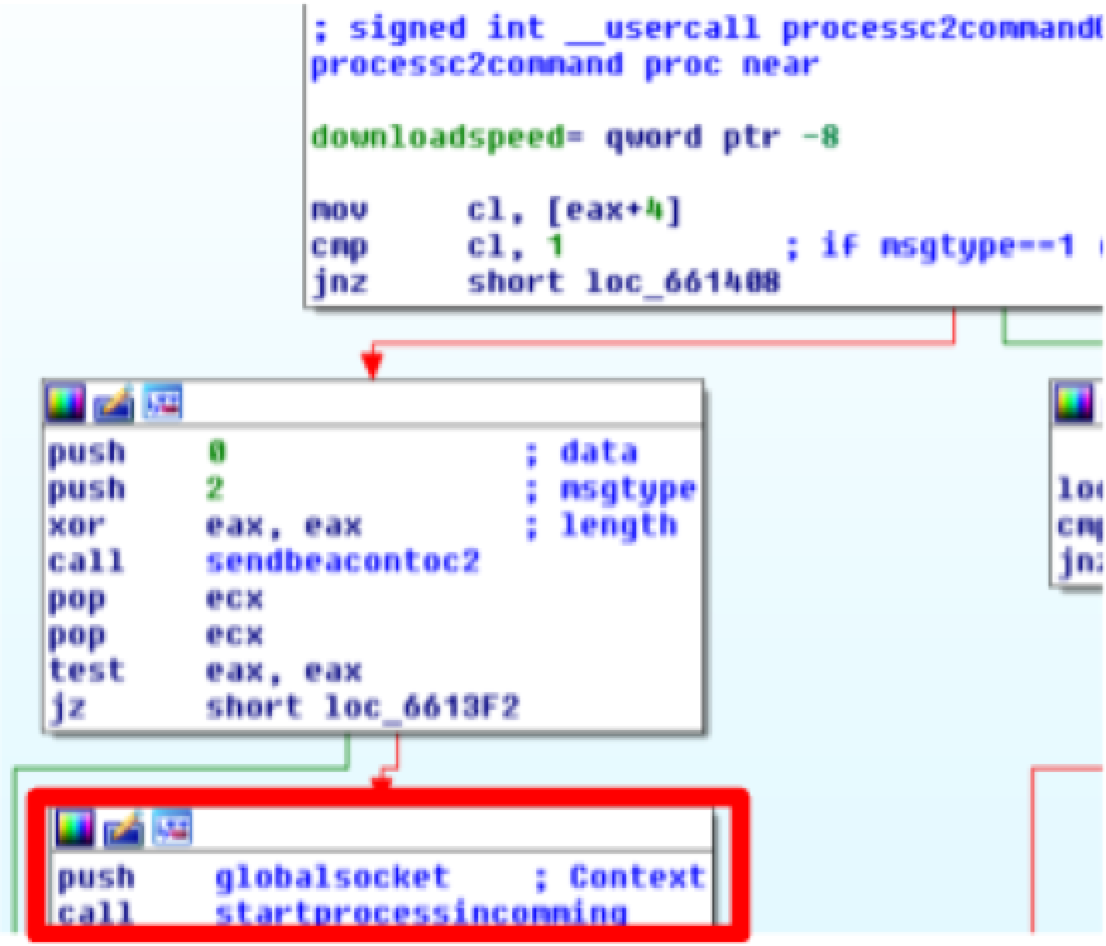
In proxy backconnect mode, the client connects to the proxy controller, that will then obtain an outbound proxy from a pool of available proxies and relay the connection through it. The following diagram illustrates the system:



1. The client connects to the proxy controller (C2) to start a connection to the target;
2. All bots in the pool periodically send a beacon message to the C2;
3. The C2 replies with the startbackconnect command to the intended bot;
4. The bot connects to the C2 using a special backconnect message. This same socket will be used from this moment on to proxy the communication between the client and the server.

There are several advantages to this operation mode. The main ones being that it works even if the infected system is behind NAT and it will allow a connection to internal network resources as if it came from the infected machine. Another common use for these kind of proxies is to have a constantly changing IP address, by constantly and automatically changing the bot serving as proxy.

Taking a brief view at how this is implemented, the following image shows part of the function that processes the C2 commands.



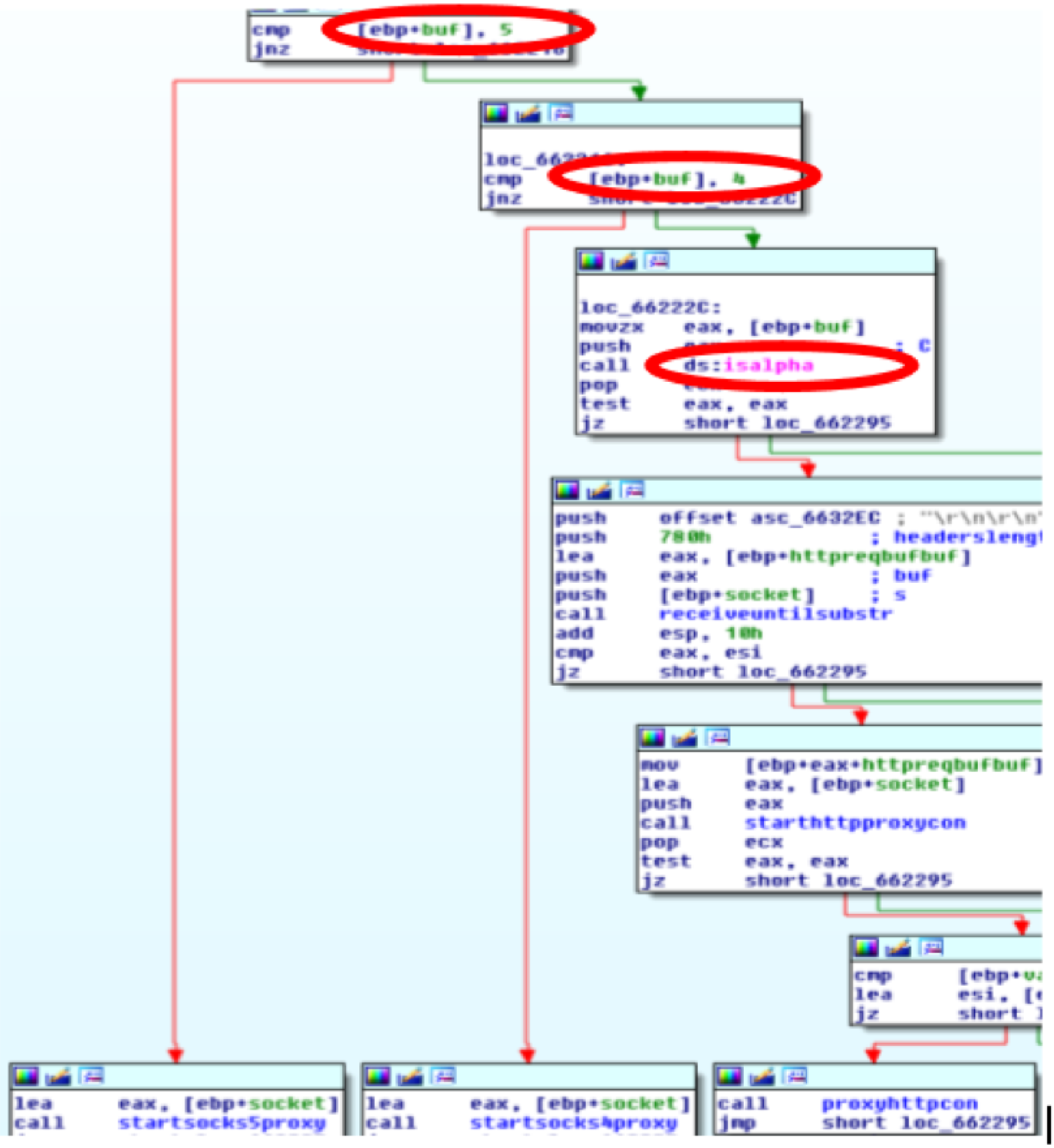
```
; signed int __usercall processc2command(
processc2command proc near

downloadspeed= qword ptr -8

mov     cl, [eax+4]
cmp     cl, 1           ; if msgtype==1
jnz     short loc_661408
```

```
push    0               ; data
push    2               ; msgtype
xor     eax, eax        ; length
call    sendbeacontoc2
pop     ecx
pop     ecx
test    eax, eax
jz      short loc_6613F2
```

```
push    globalsocket    ; Context
call    startprocessincomming
```

In case the bot receives a *startproxybackconnect* command (msgtype 1), the bot will send a *proxybackconnect* message to the C2 (msgtype 2) and then the same socket that is used in the C2 communication (*globalsocket)* to the *startprocessincoming* function that does the actual proxying work. This means that the same connection used for C2 communication will then be used to proxy the connections.

The *processincomming* function reads 2 bytes from the incoming connection (direct or through backconnect) and checks if the first has the value 5 (Socks v5 protocol), 4 (Socks v4 protocol) or if it is alphanumeric (HTTP Proxy). It then calls the appropriate function that does the actual proxying work for each of the supported protocols.



```
cmp     [ebp+buf], 5
jnz
```

```
loc_6622
cmp     [ebp+buf], 4
jnz
```

```
loc_66222C:
movzx   eax, [ebp+buf]
push                        ; C
call    ds:isalpha
pop
test    eax, eax
jz      short loc_662295
```

```
push    offset asc_6632EC ; "\r\n\r\n"
push    780h              ; headerslengt
lea     eax, [ebp+httpreqbufbuf]
push    eax               ; buf
push    [ebp+socket]      ; s
call    receiveuntilsubstr
add     esp, 10h
cmp     eax, esi
jz      short loc_662295
```

```
mov     [ebp+eax+httpreqbufbuf]
lea     eax, [ebp+socket]
push    eax
call    starthttpproxycon
pop     ecx
test    eax, eax
jz      short loc_662295
```

```
cmp     [ebp+v
lea     esi, [
jz      short 
```

```
lea     eax, [ebp+socket]
call    startsocks5proxy
```

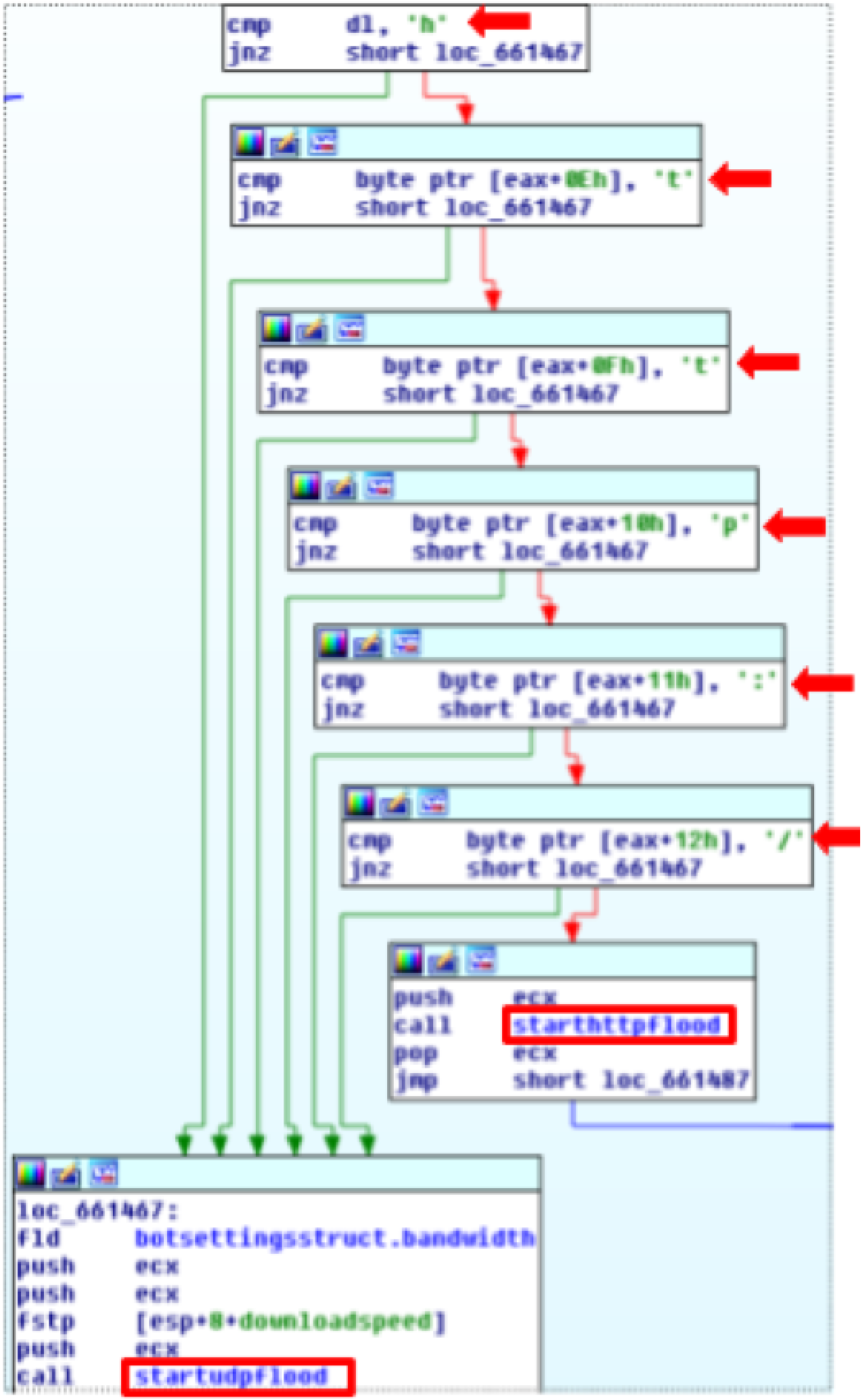
```
lea     eax, [ebp+socket]
call    startsocks4proxy
```

```
call    proxyhttpcon
jmp     short loc_662295
```

DDOS attack features

Possibly the most interesting, and unexpected, feature of this module is the DDOS attack mode. The module contains two very basic DDOS attack modes, that don’t have special features like origin IP address spoofing or amplification techniques. However, given the size of the Necurs botnets (more than 1M IP/24 hours in the largest botnet), even the most basic techniques should produce a very powerful attack.

Taking a brief look at how it is implemented, if the bot receives the *startDDOS* command (msgtype 5) the payload portion of message is parsed looking for the string “http:/”. If it is found, the *HTTPflood* function is called, if not, the *UDPFlood* function is called, as shown in the following image:



```
cmp     dl, 'h'
jnz     short loc_661467
```

```
cmp     byte ptr [eax+0Eh], 't'
jnz     short loc_661467
```

```
cmp     byte ptr [eax+0Fh], 't'
jnz     short loc_661467
```

```
cmp     byte ptr [eax+10h], 'p'
jnz     short loc_661467
```

```
cmp     byte ptr [eax+11h], ':'
jnz     short loc_661467
```

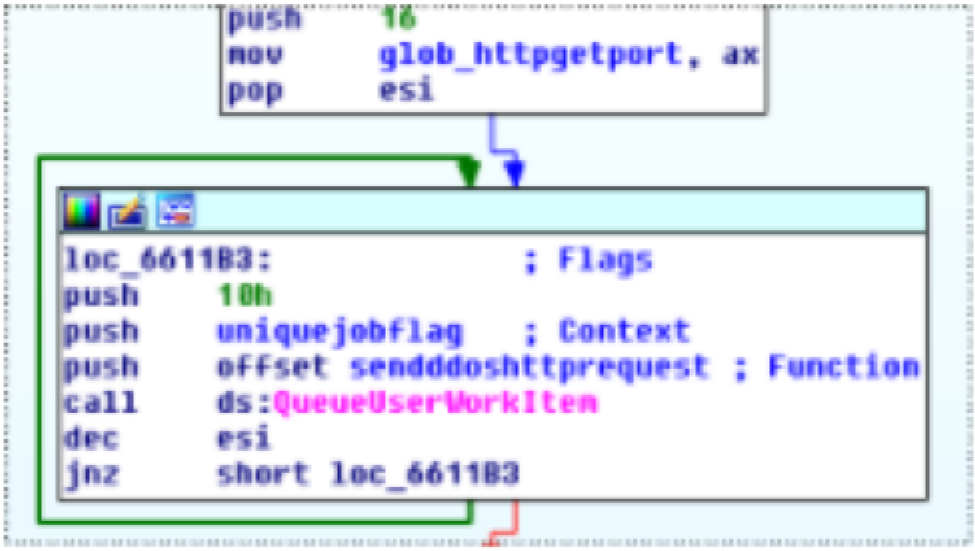
```
cmp     byte ptr [eax+12h], '/'
jnz     short loc_661467
```

```
push    ecx
call    starthttpflood
pop     ecx
jmp     short loc_661487
```

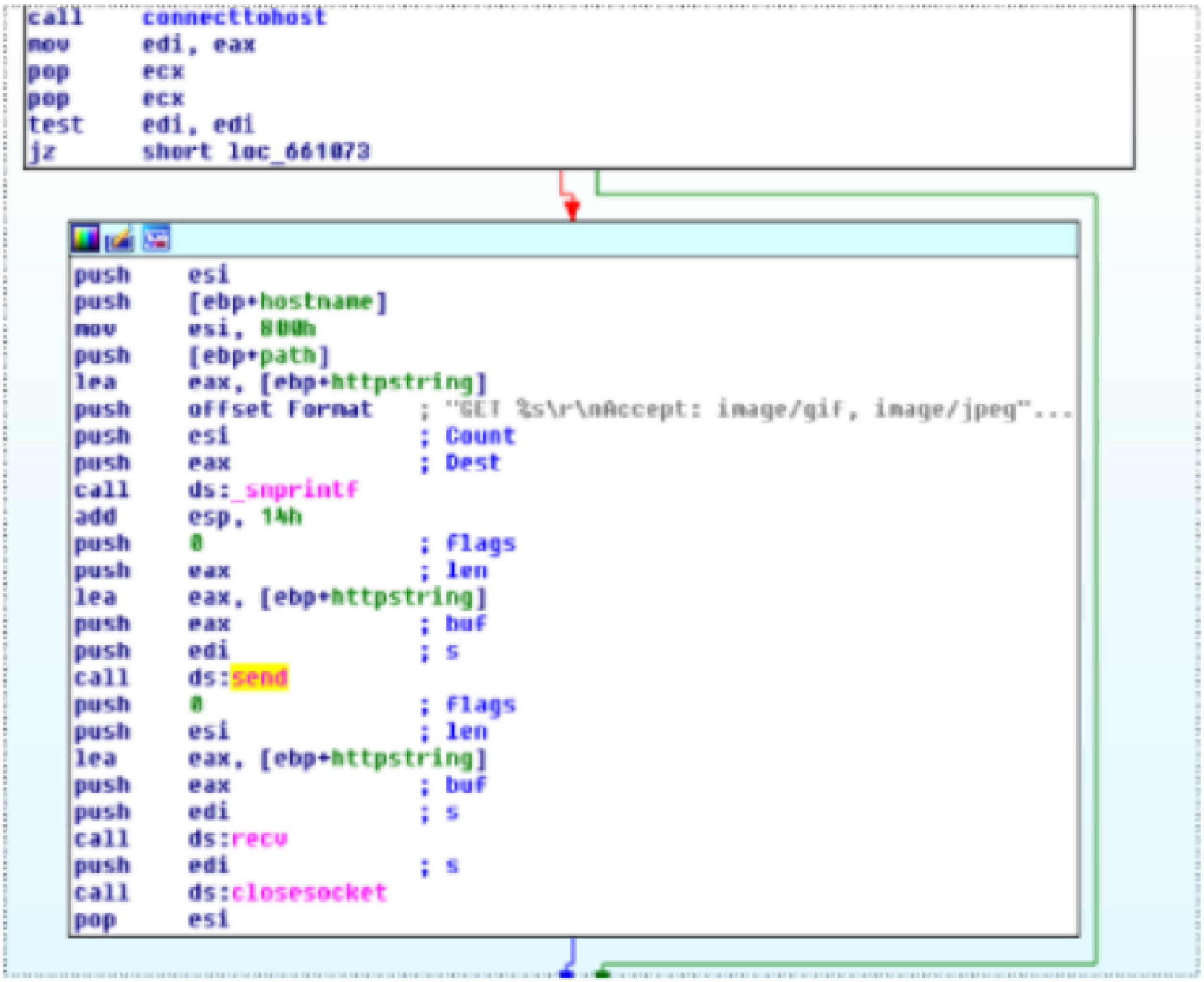
```
loc_661467:
fld     botsettingsstruct.bandwidth
push    ecx
push    ecx
fstp    [esp+8+downloadspeed]
push    ecx
call    startudpflood
```

HTTP Flood attack mode

The HTTP attack works by starting 16 threads that perform an endless loop of HTTP requests. The following images show the 16 threads being queued and the section of the code that sends the HTTP request:

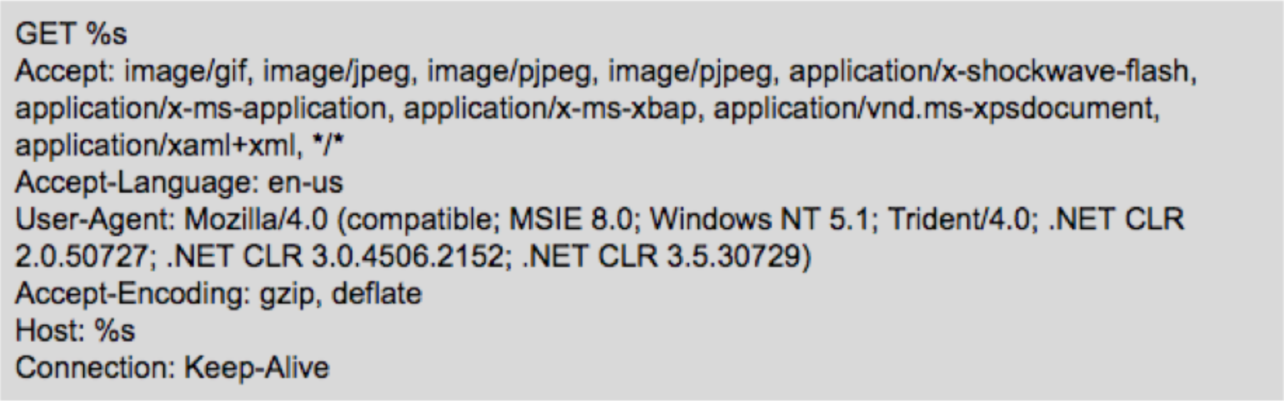


```
push    16
mov     glob_httpgetport, ax
pop     esi

loc_661183:                     ; Flags
push    10h
push    uniquejobflag   ; Context
push    offset sendddoshttprequest ; Function
call    ds:QueueUserWorkItem
dec     esi
jnz     short loc_661183
```



```
call    connecttohost
mov     edi, eax
pop     ecx
pop     ecx
test    edi, edi
jz      short loc_661073

push    esi
push    [ebp+hostname]
mov     esi, 800h
push    [ebp+path]
lea     eax, [ebp+httpstring]
push    offset Format   ; "GET %s\r\nAccept: image/gif, image/jpeg"...
push    esi             ; Count
push    eax             ; Dest
call    ds:_snprintf
add     esp, 14h
push    0               ; flags
push    eax             ; len
lea     eax, [ebp+httpstring]
push    eax             ; buf
push    edi             ; s
call    ds:send
push    0               ; flags
push    esi             ; len
lea     eax, [ebp+httpstring]
push    eax             ; buf
push    edi             ; s
call    ds:recv
push    edi             ; s
call    ds:closesocket
pop     esi
```

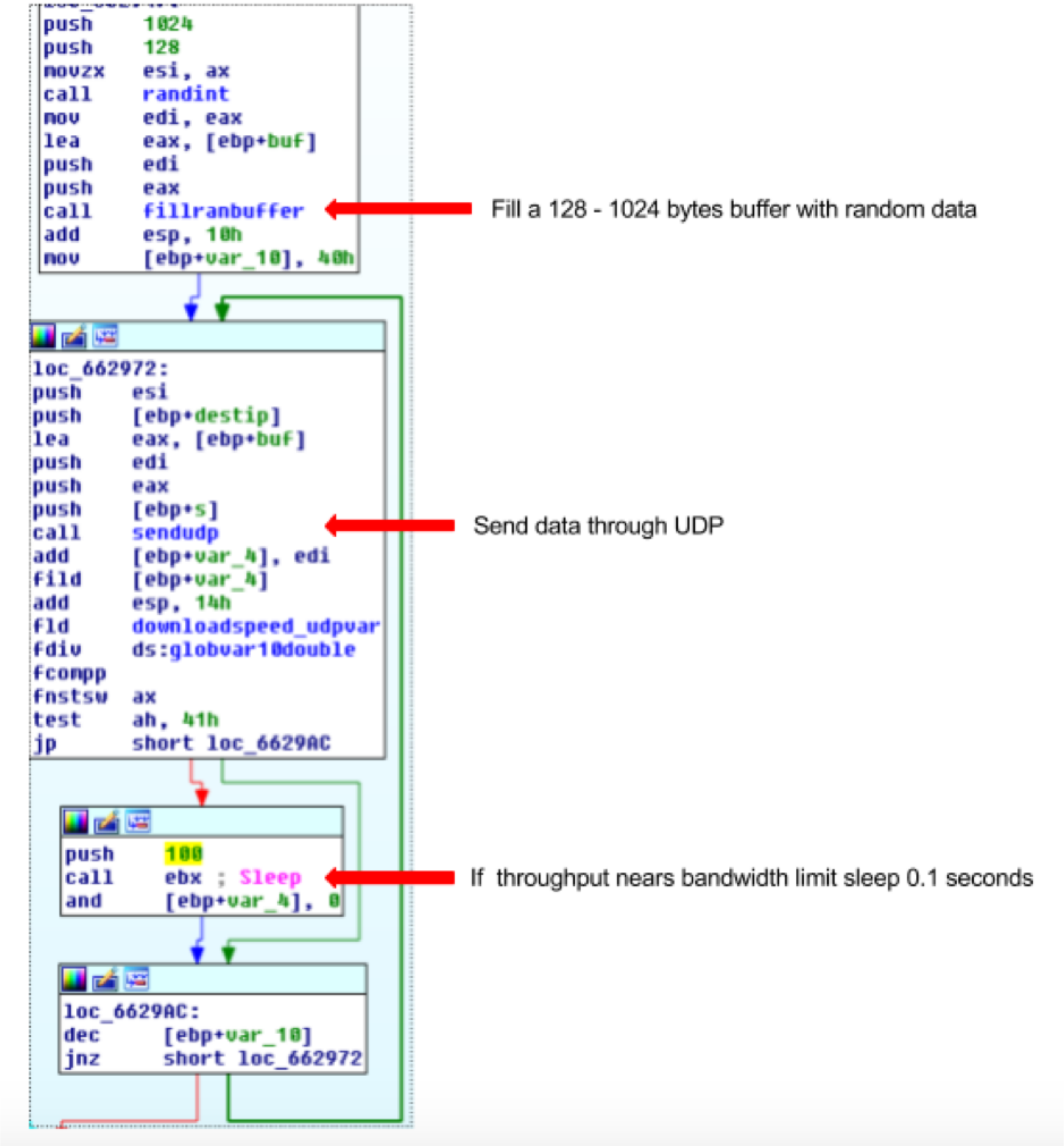
The HTTP request is built using the following format string:



```
GET %s
Accept: image/gif, image/jpeg, image/pjpeg, image/pjpeg, application/x-shockwave-flash,
application/x-ms-application, application/x-ms-xbap, application/vnd.ms-xpsdocument,
application/xaml+xml, */*
Accept-Language: en-us
User-Agent: Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 5.1; Trident/4.0; .NET CLR
2.0.50727; .NET CLR 3.0.4506.2152; .NET CLR 3.5.30729)
Accept-Encoding: gzip, deflate
Host: %s
Connection: Keep-Alive
```

UDP Flood attack mode

The UDP flood attack works by repeatedly sending a random payload with size between 128 and 1024 bytes. The function contains a 0.1 second sleep that may be triggered depending on the available bandwidth (calculated during bot initialization), possibly to avoid losing access to the bot during a DDOS attack. The following image shows the UDPFlood main loop:



```
push    1024
push    128
movzx   esi, ax
call    randint
mov     edi, eax
lea     eax, [ebp+buf]
push    edi
push    eax
call    fillranbuffer
add     esp, 10h
mov     [ebp+var_10], 40h
```

```
loc_662972:
push    esi
push    [ebp+destip]
lea     eax, [ebp+buf]
push    edi
push    eax
push    [ebp+s]
call    sendudp
add     [ebp+var_4], edi
fild    [ebp+var_4]
add     esp, 14h
fld     downloadspeed_udpvar
fdiv    ds:globvar10double
fcompp
fnstsw  ax
test    ah, 41h
jp      short loc_6629AC
```

```
push    100
call    ebx ; Sleep
and     [ebp+var_4], 0
```

```
loc_6629AC:
dec     [ebp+var_10]
jnz     short loc_662972
```

Fill a 128 - 1024 bytes buffer with random data

Send data through UDP

If throughput nears bandwidth limit sleep 0.1 seconds

Conclusion

Although known mainly for its spam module, Necurs is a modular malware that can be used for many different purposes. In this blog post we detailed a module that adds SOCKS/HTTP proxy and DDOS capabilities to this malware. Although we have not seen Necurs being used to perform DDOS attacks, this capability is currently deployed in the infected systems and taking into account the size of the botnet it could produce a powerful attack.

References

For more information about the Necurs Botnet, check out the following links:

- https://www.virusbulletin.com/virusbulletin/2014/04/curse-necurs-part-1
- https://www.virusbulletin.com/virusbulletin/2014/05/curse-necurs-part-2
- https://www.virusbulletin.com/virusbulletin/2014/06/curse-necurs-part-3
- https://www.johannesbader.ch/2015/02/the-dgas-of-necurs/
- http://www.malwaretech.com/2016/02/necursp2p-hybrid-peer-to-peer-necurs.html
- https://www.cert.pl/en/news/single/necurs-hybrid-spam-botnet/

Analysed module sample:

f3aeafe50880cb9dd584b3669800c017de561f8f9654440f62c51319fda0e970