E CORP

Incident Response Threat Analysis

Prepared for

E Corp

Cloud Snooper – rootkit

A hallmark of the most sophisticated attackers

-Possibly Nation-State sponsored-

[28-Feb-2023]

SUMMARY

I made this report to present the reverse engineering of event-based malware (and not in response to alerts from a security software/ use case asked by a customer). This report includes findings and recommended actions (Details about the analysis given in the annex). Analysis technics have been shared by the Kaspersky GReAT team [GReAT = Global Research & Analysis Team] .

FINDINGS

Attack Vector

For the samples analyzed, the infection vector is not known. **According to the Sophos Lab researchers, one of the working theories is that the attackers broke into a server through SSH protected with password authentication**

The alert originated from the following device:

- Computer Name: {Enter Device Name}
- IP Address: {Enter IP Address}
- Assigned User: {Enter User's First name & Last name}
- Date & Time of Event: {Enter date/time event occurred}
- Last Seen Date/Time Stamp: {Enter the Last Logon Date/Time stamp}

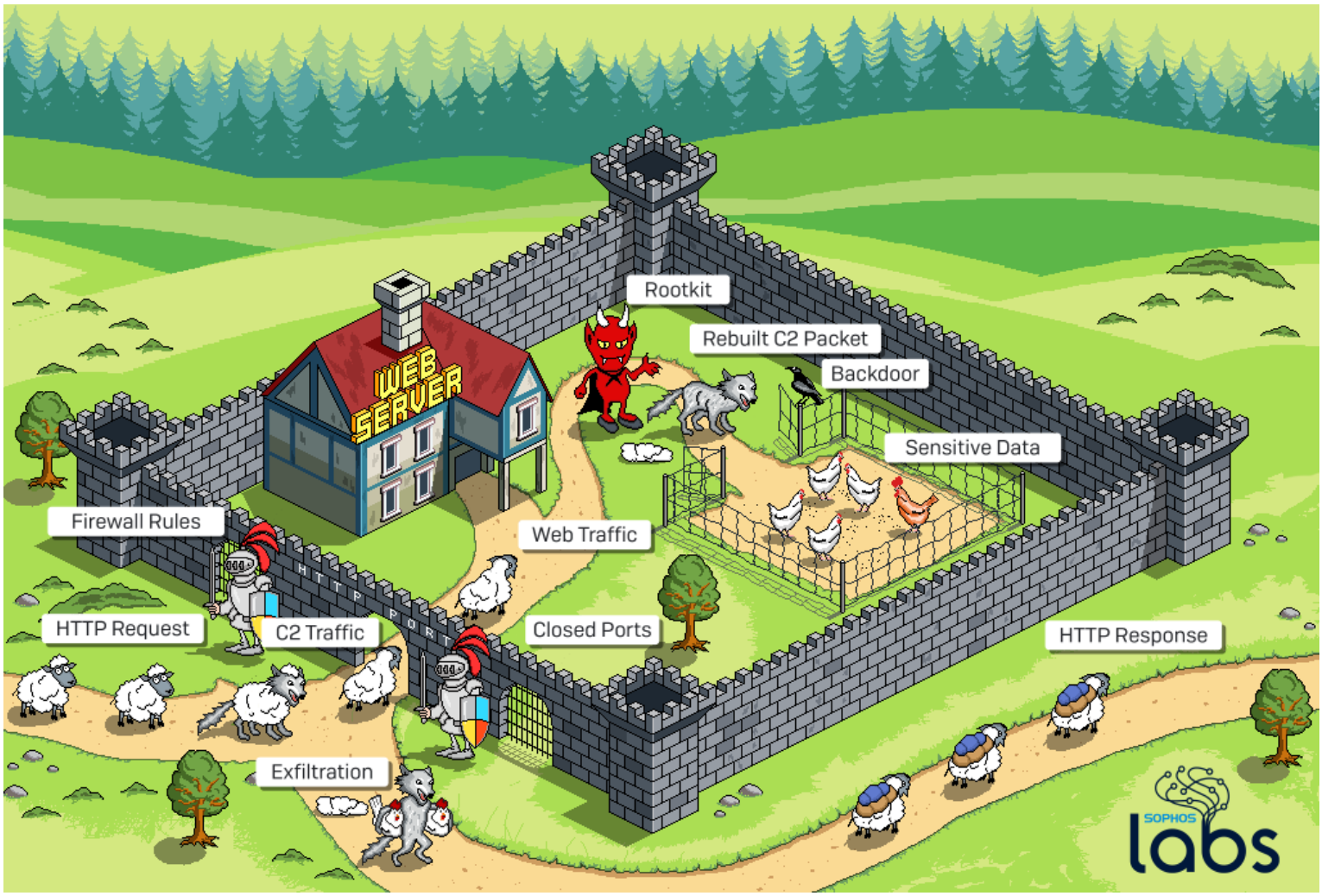
This is what happened. The following action(s) caused the device to become compromised:

Action / Infection Vector	True? Yes	Comments
Browsing the Web		
Malicious link		
Browser Exploit		
File Download		
Clicking Malicious Link(s)		
Link in e-mail	N/A	
Link in file attachment		
Link in chat application		
Downloading Malware		
From chat application		
From e-mail attachment		
From removable media or USB disk		
From website		
Opening Malicious Attachment(s)/File(s)		
From e-mail		
From removable media or USB disk		

This table shows what we know.

Indicator	Present?	Notes
MD5 Hash	bf1e5221203d595df2c2c444840fb715 b6eb82dd06b3484d7e8b95e51c4f28d6 fd4c26e8c6e1c40c6760ffae213d40da	
SHA 1 Hash	5082fb03b05318bef35544b8d47edc6c711c0e10	
SHA 256 Hash	682d6aeeaecb0233dc020430ae1af8cb01f76425f966a2947edd922a35ca895f D220ba1dec19d8cb9ac2a371b5f20ba30eb35fcffcc1f5192250d03a029fd580	
Infection Vector		
Packer		
Language		
Malware/Family		
Setup of the program		
Anti-Debug Technique		
Evasion Technique	rrootkit_net_tcp_connecting_open bind and make a call to the same socket in order to fix the source port for the socket. The source port of the TCP/UDP packet is used to encode orders	
File/DLL File	/tmp/rrtkernel.ko /usr/bin/snd_floppy	
Kernel module	snd_floppy	
URL		
cryptography	RC4	
Process/ Suspicious API	tshd	TinySHell, or 'tsh', backdoor which grants remote command execution, shell (PTY) access, command execution, and file transfer capabilities over an encrypted reverse or bind connection.
Capabilities of the program	backdoor	The server listens on the port 2080. Inbound connections from 1010, 2020, 6060, 7070, 8080, 9999
C2 communications	The messages from the c2 are processed in this order: proto-tunnel-view	

Behavior summary



The malicious packets enter through the HTTP port, then there are intercepted by a rootkit that redirect them to a backdoor installed on the system.

Process Graph

[N/A] Template paragraph

Miter Attack Matrix

[N/A] Template paragraph

RECOMMENDED ACTIONS

First, make sure all your computers are running updated security solution.

Possible Incident Response Steps

Vulnerability	Comments
Malicious file(s)	Possible Incident Response Steps:
	1. Check the Findings section for a list of infected files. 2. Check the web proxy for similar files or hashes [Yara could help to find related files] 3. Check the Next-Gen firewall for traffic to the malicious domain(s) and IP(s) address(es) 4. Check AV solution for hashes 5. Run additional AV scans on machines involved in isolated event 6. Speak with user to see if there is a reoccurring theme, such as a repeated site visited. 7. Determine if that computer has had any other triggered events from other security products in the last 48 hours leading up to isolated event. 8. According to the zero-trust model ,remote administration services use strongly encrypted protocols and only accept connections from authorized users or locations. 9. Disable Power Shell in windows 10 via security policy for classic users or/and use AppLocker to limit who can work with PowerShell. 10. Use a Micro-VM solution (such as HP Wolf Enterprise Security), to open the office documents in Micro-VM.
Malicious e-mail	Possible Incident Response Steps:
	1. Check the email gateway for possible related emails. 2. Purge additional emails from environment if necessary. 3. Proactively block senders or indicators from coming into the environment again. 4. Educate affected end users on how to handle malicious emails. 5. Run additional AV scans on machines involved in isolated event.
Malicious Software	Possible Incident Response Steps:
	1. Check the computer for additional unwanted software. 2. Check the computer for related vulnerabilities. 3. Check that the software is not installed on other computers in the environment. 4. Check the web proxy for other possible downloads. 5. Run additional AV scans on machines involved in isolated event 6. Speak with user to see if there is a reoccurring theme, such as a repeated site visited. 7. Determine if that computer has had any other triggered events from other security products in the last 48 hours leading up to isolated event.
Infected USB Drive	Possible Incident Response Steps:
	1. Check AV solution for hashes 2. Run additional AV scans on machines involved in isolated event 3. Speak with user to see if there is a reoccurring theme, plugging in the removable media to their home computer. 4. Determine if that computer has had any other triggered events from other security products in the last 48 hours leading up to isolated event. 5. Educate end user on keeping the infected device out of work computer. 6. Formatting the removable media. 7. Controlling removable media connections.

Specific test to detect the rootkit on an infected machine

- the open ports bound to localhost
- Looking at the list of loaded kernel modules
- Sending a packet with the correct source port

Yara-Rule

Element to write a Yara rule to detect this Linux Rootkit:
Assignment to this addr 0FFFF880000000000 that references +24 (0x18)
The custom Yara rule could be added here.

SIEM Rule

[N/A] Template paragraph

Other possible detection and response rules

[N/A] Template paragraph

REFERENCES

This report may contain information that is available on the Internet. For more information, please refer to the following websites:

Title	Author	URLs	Date
'Cloud Snooper' Attack Bypasses Firewall Security Measures	SophosLabs : Sergei Shevchenko, Timothy Easton	https://news.sophos.com/en-us/2020/02/25/cloud-snooper-attack-bypasses-firewall-security-measures/	25 feb 2020
Cloud Snooper Attack Bypasses AWS Security Measures	SophosLabs	https://www.sophos.com/en-us/medialibrary/PDFs/technical-papers/sophoslabs-cloud-snooper-report.pdf	
Cloud Snooper Attack Uses Innocent-Looking Requests to Bypass Firewall Rules	David Bisson	https://securityintelligence.com/news/cloud-snooper-attack-uses-innocent-looking-requests-to-bypass-firewall-rules/	
Attacks on Linux Servers in the Cloud—The Rise of SSH-Abusing Malware	Yana Blachman	https://www.venafi.com/blog/attacks-linux-servers-cloud-rise-ssh-abusing-malware	
		Useful Resources	
ELF Executable and_Linkable_Format diagram	Ange_Albertin	https://upload.wikimedia.org/wikipedia/commons/e/e4/ELF_Executable_and_Linkable_Format_diagram_by_Ange_Albertini.png	
tsh	Stanislaw Pusep creaktive	https://github.com/creaktive/tsh	
KernelMode.info – Forum Archive	N/A	https://vxug.fakedoma.in/archive/kernelmode-info/viewtopic8a2e8a2e.html?t=4128	
Understanding the Unix Stream Socket Connection	Vivekananda Holla	https://www.hitchhikersguidetolearning.com/2020/04/19/understanding-the-unix-stream-socket-connection/	
Linux Memory Cheat Sheet	Gabrio Tognozzi	https://gabrio-tognozzi.medium.com/linux-memory-cheat-sheet-2c7454aa1e29 and https://lists.linuxfoundation.org/pipermail/iommu/2019-August/038333.html [Donald Buczek] https://elixir.bootlin.com/linux/v4.5/source/include/linux/netfilter.h#L87	

CONTACT US

For additional assistance, please contact Natacha BAKIR.

- **Phone number** – On Demand
- **Email address** – alphabot42@tutanota.com
- **GitHub** – **Alphabot42**

Annexes

Analyzing an Event Based Malware

Why attackers target the Linux machines?

Rootkit are rare on windows because:

- Windows rootkits pose many challenges for attackers such as admin privileges
- Any bug or programming error inside the Windows kernel can potentially take down the whole system with it
- Linux machine are generally less monitored than Windows one
- Linux machines generally don't have security solutions installed and the Linux security market is less developed

Analyzing network protocols used by backdoors

Why rootkit deserve to be understood?

The rootkit runs on the deepest layers of the operating system, it has capabilities not available in user-land malware. So, it has the ability to hide activity from the user such as existence of files or network activity. That's the reason why it can bypass security measures

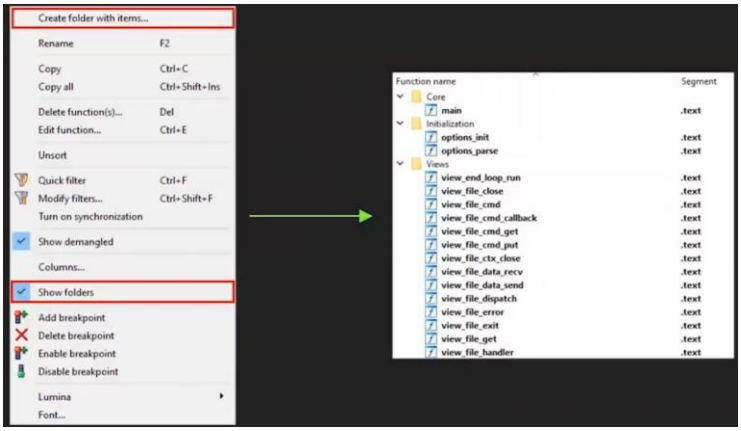
The approach

As ELF can't run on windows, we won't use the debugger, we'll used only IDA Pro
Most of trojan used http and you may encounter homebrew communication such as the Lazarus forwarding utility

Methodology / Tips

If you have to choose between PE and ELF, choose ELF, because the Linux one will contains symbols that will help to understand the code. Ida pro knows all the structures and will add symbols. Most of the time, symbols are pretty much self-explanatory.

- Try to recognize variants of open-source trojans glancing at the strings, the symbols and the functions
- Don't hesitate to Google functions
- If time permitted keep reversing the open-source program to find potential modification made by the attacker, to be able to track him.
- Analyze the network communications and try to resolve the arguments with IDA Pro
- Use IDA Pro comments and write down function names. If you put function names or addresses as comments, you can jump to them by double clicking. Use IDA's folder system for functions (create lots of folders and sort all those functions neatly inside of them)

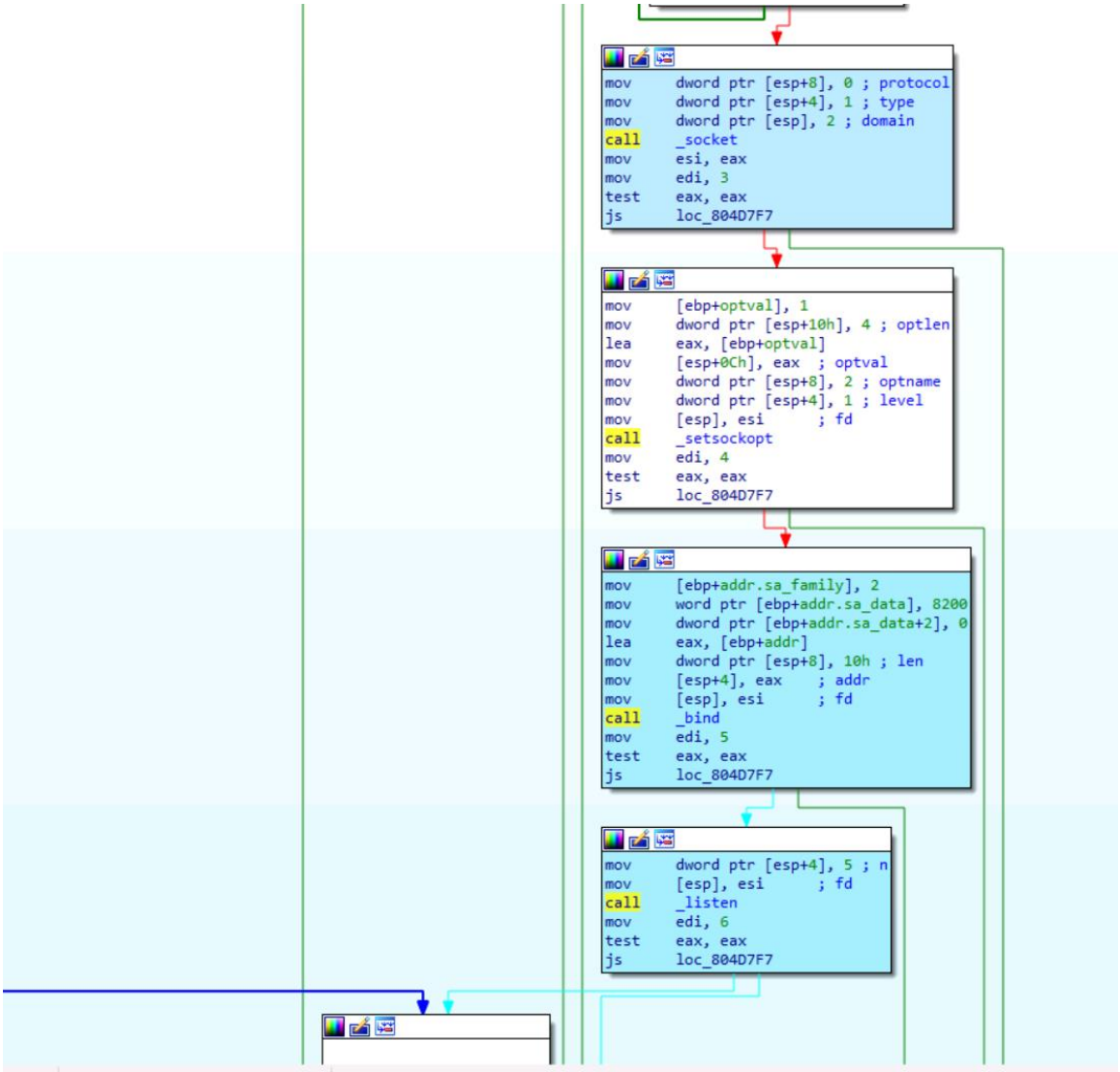


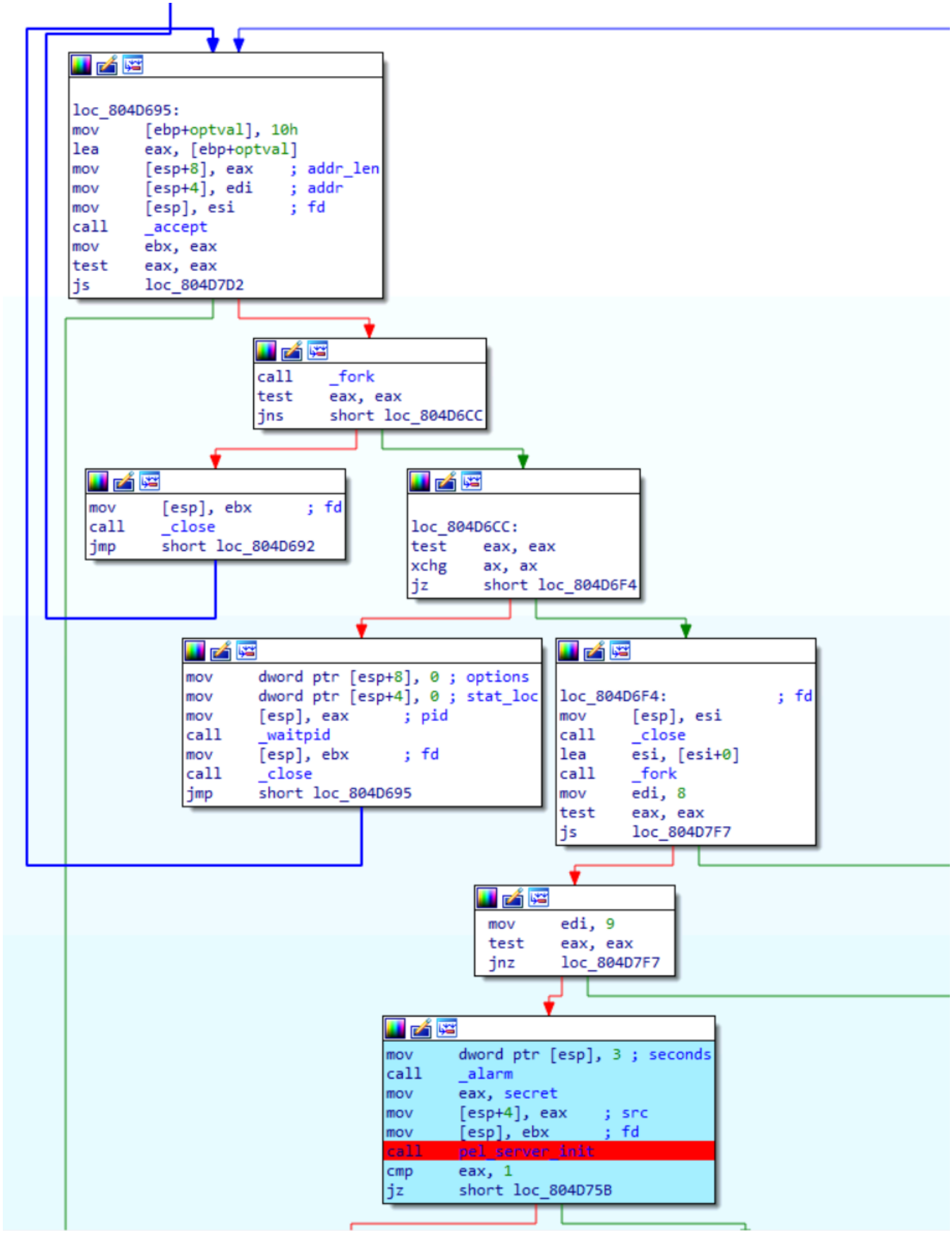
Screenshot of the analysis

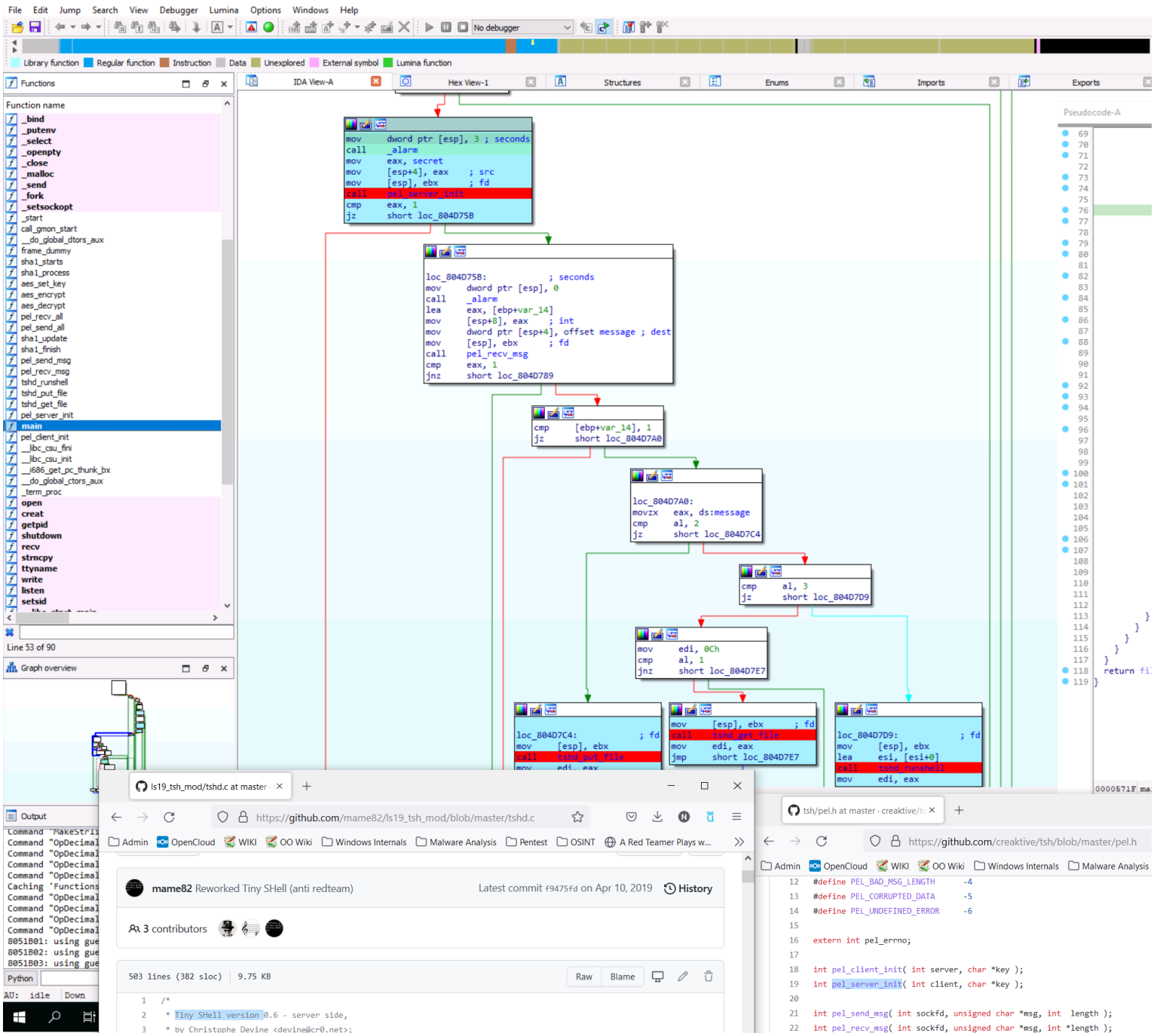
Reverse-engineer the first user-land component

Try to find which open-source malware it is derived from

- Explore the functions window, and the main function. Google functions to know more about it.



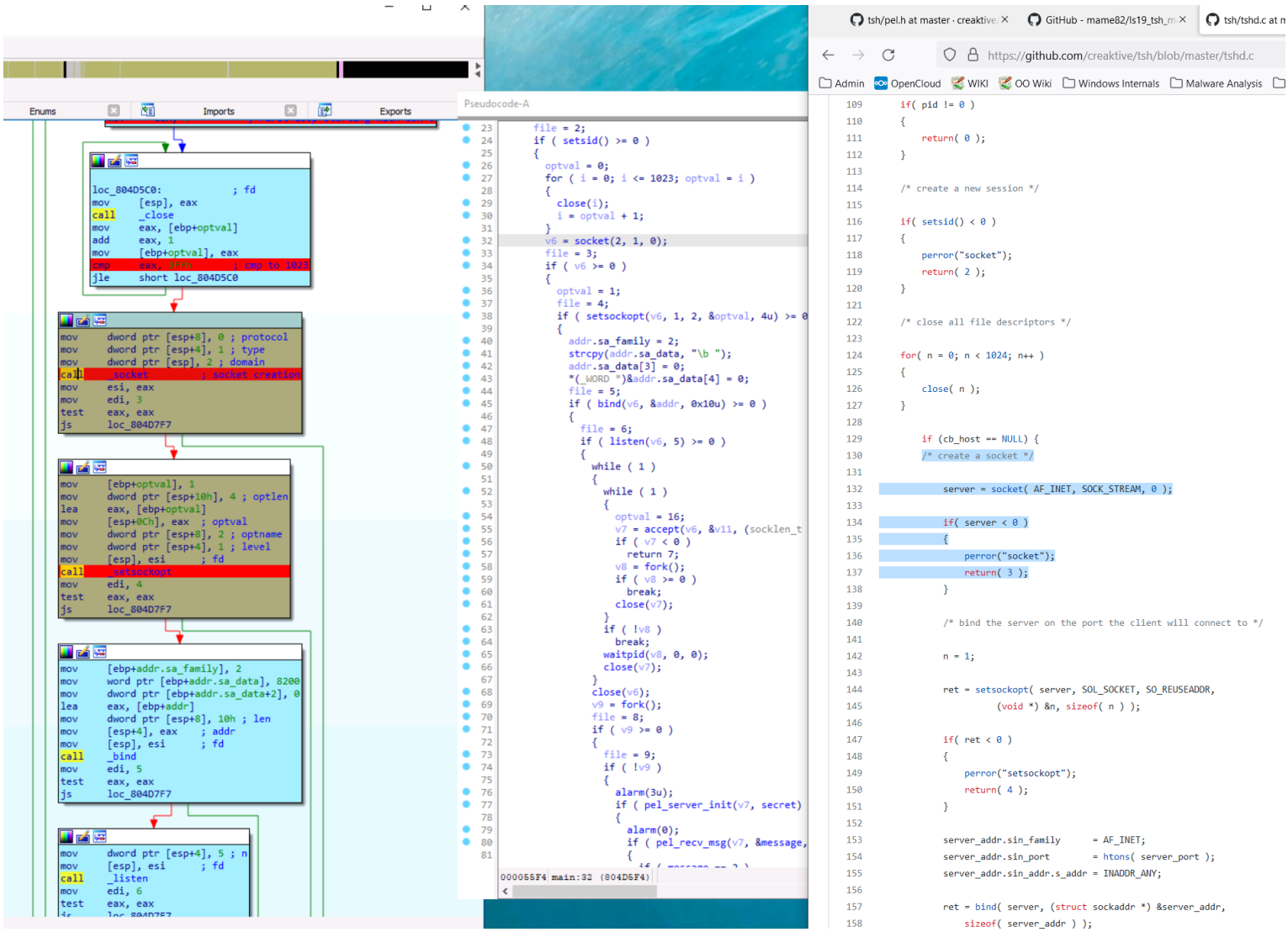

loc_804D695:
mov [ebp+optval], 10h
lea eax, [ebp+optval]
mov [esp+8], eax ; addr_len
mov [esp+4], edi ; addr
mov [esp], esi ; fd
call _accept
mov ebx, eax
test eax, eax
js loc_804D7D2
call _fork
test eax, eax
jns short loc_804D6CC
mov [esp], ebx ; fd
call _close
jmp short loc_804D692
loc_804D6CC:
test eax, eax
xchg ax, ax
jz short loc_804D6F4
mov dword ptr [esp+8], 0 ; options
mov dword ptr [esp+4], 0 ; stat_loc
mov [esp], eax ; pid
call _waitpid
mov [esp], ebx ; fd
call _close
jmp short loc_804D695
loc_804D6F4: ; fd
mov [esp], esi
call _close
lea esi, [esi+0]
call _fork
mov edi, 8
test eax, eax
js loc_804D7F7
mov edi, 9
test eax, eax
jnz loc_804D7F7
mov dword ptr [esp], 3 ; seconds
call _alarm
mov eax, secret
mov [esp+4], eax ; src
mov [esp], ebx ; fd
call pel_server_init
cmp eax, 1
jz short loc_804D75B

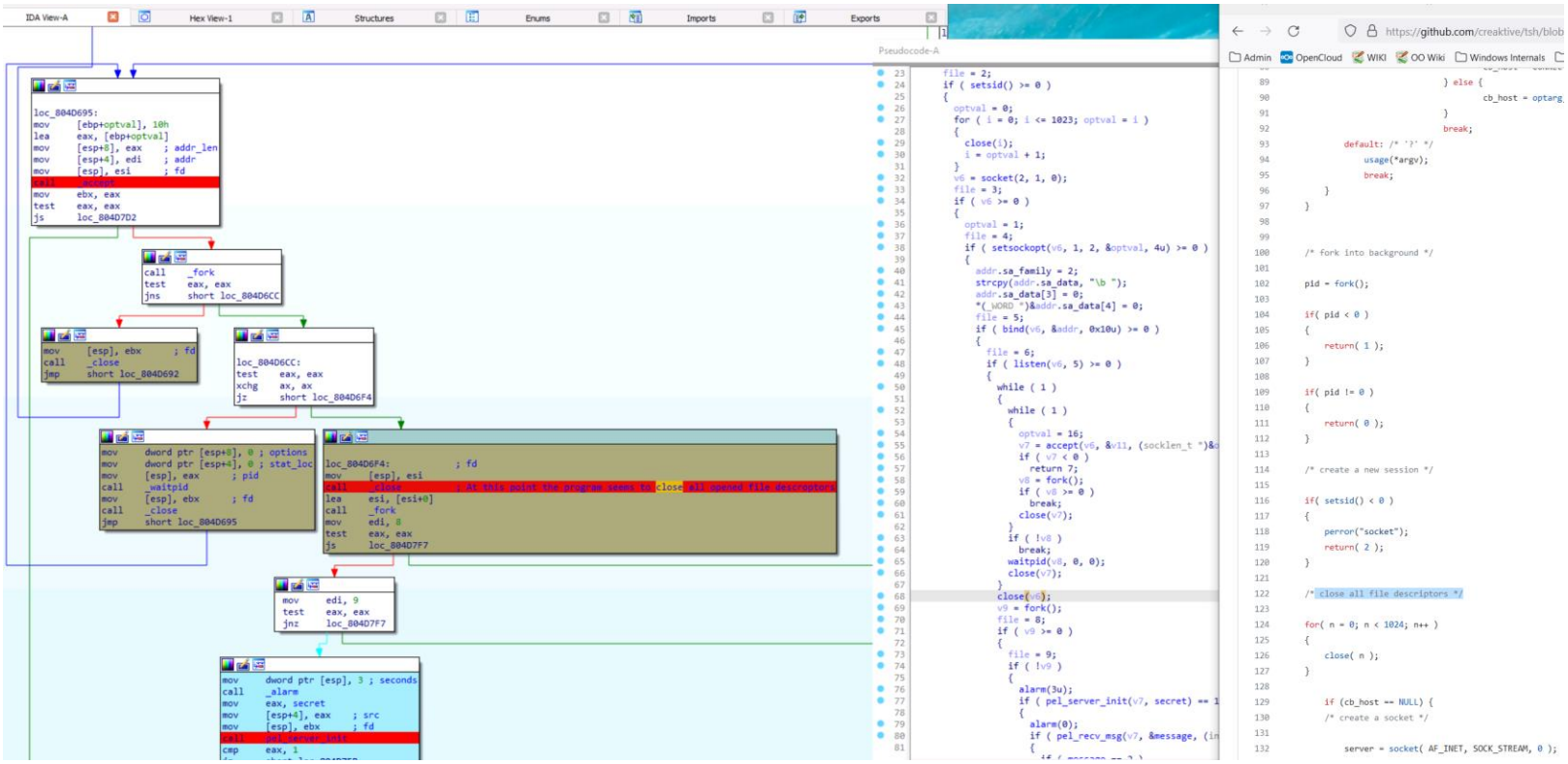


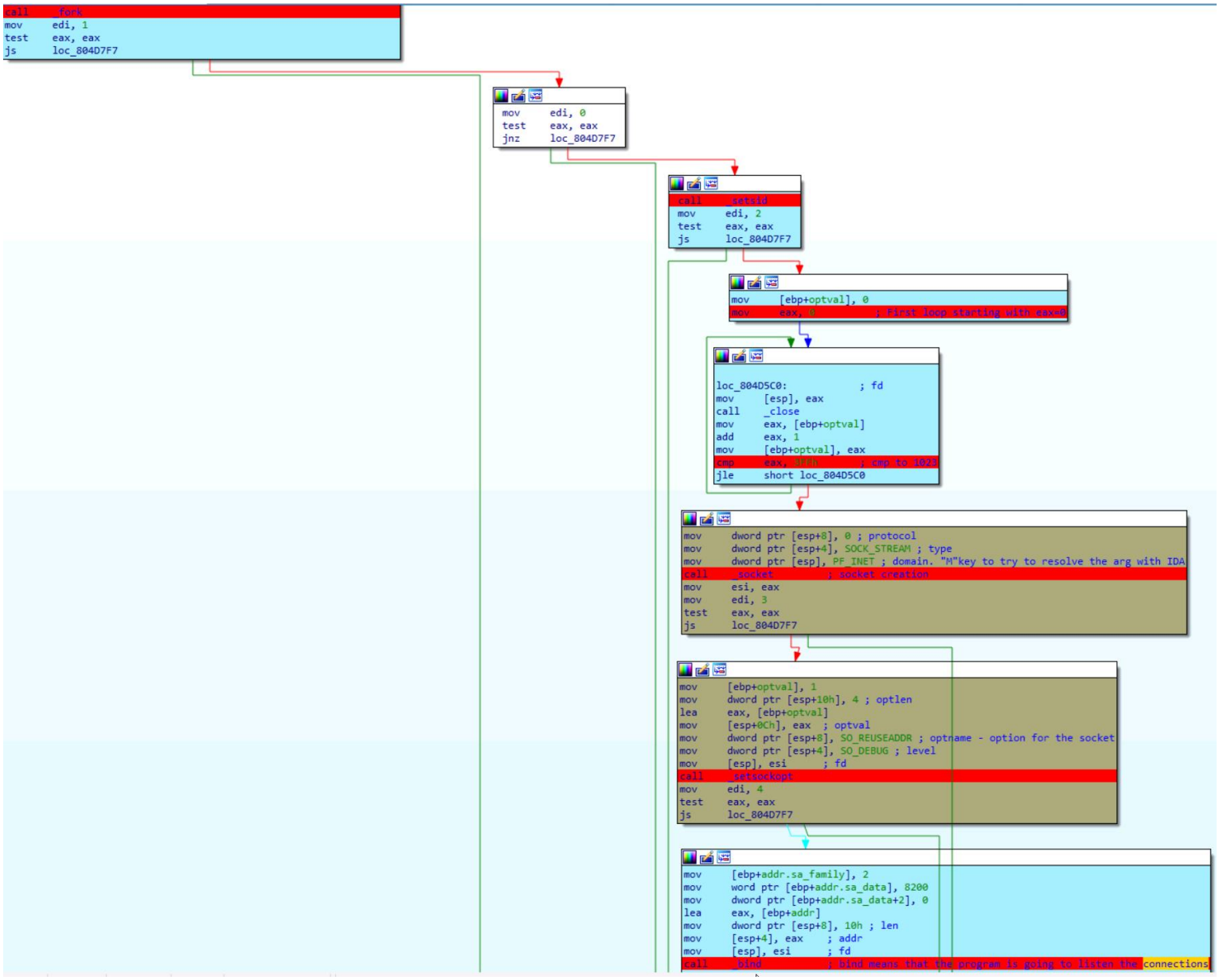
It's still interesting to keep reverse engineering the whole program, to verify if the attacker did not make any modification of this tool. The reason for this is, it's not going to be easy to cluster attacks based on usage of tsh because multiple threat actors might be using it; however, if modifications were made to it, then suddenly you have a tool that is characteristic of a single group. So, if you are able to identify exact modifications that were made to an opensource, then you will be able to still track a threat actor, even though they are using something that come from the open source. At this point, we already have identified a backdoor capability

How the communications with the attacker are established?

Glance at the network aspects.

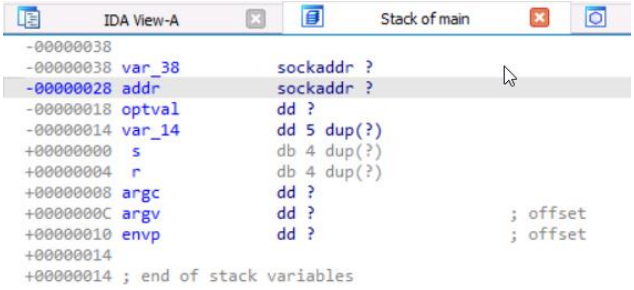


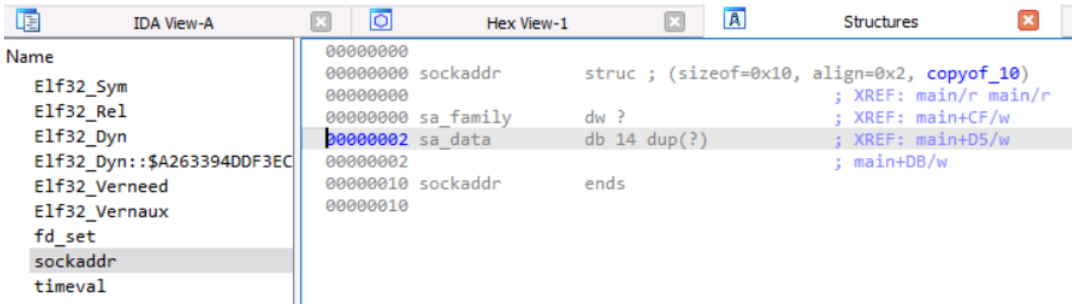

IDA View-A
Hex View-1
Structures
Enums
Imports
Exports
Pseudocode-A
https://github.com/creaktive/tsh/blob
Admin
OpenCloud
WIKI
OO Wiki
Windows Internals



"Bind" means that the program is going to listen the connections.

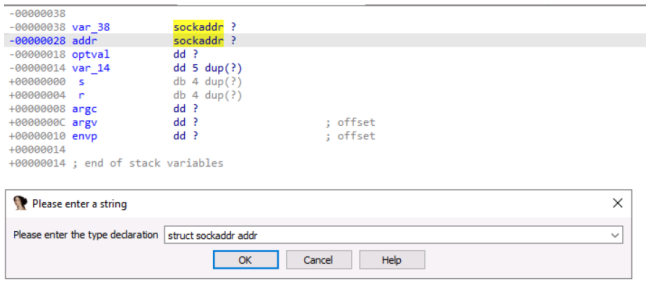
- Glance at the structure of the address

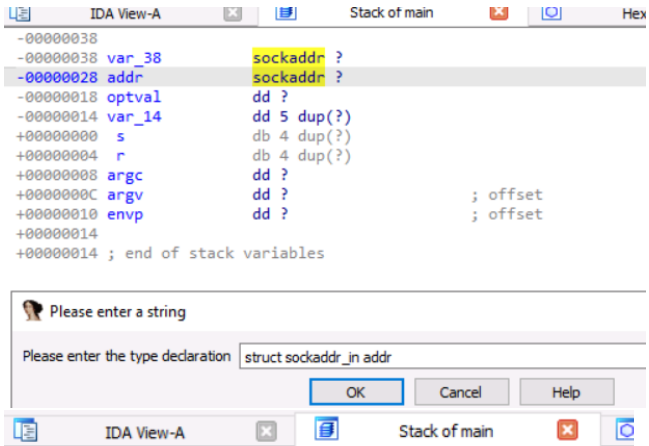


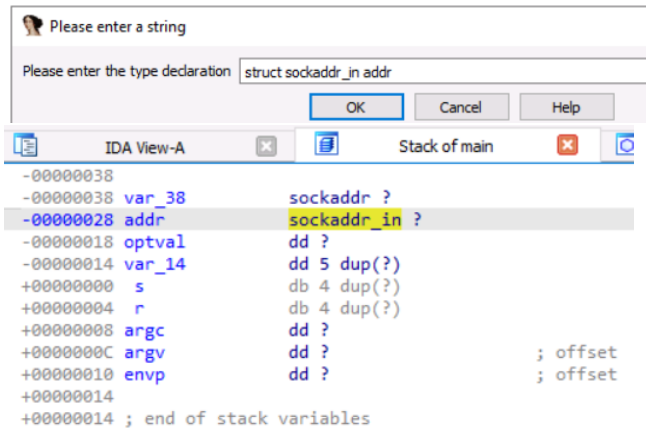


Sockaddr is a big generic structure that can handle various types of sockets

As we know that we are analyzing a stream socket, we can "update" this structure with the "Y" hotkey . Indeed, Stream sockets enable processes to communicate using TCP.

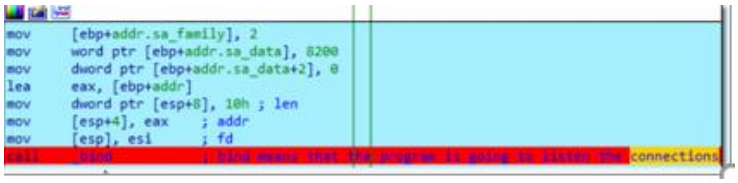




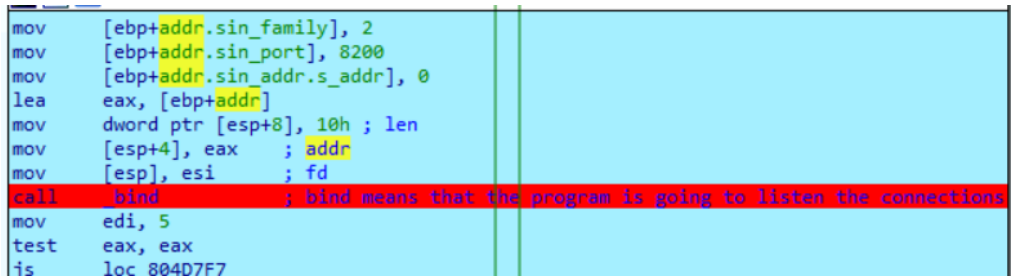


Source : https://docs.microsoft.com/fr-fr/windows/win32/api/winsock/ns-winsock-sockaddr_in

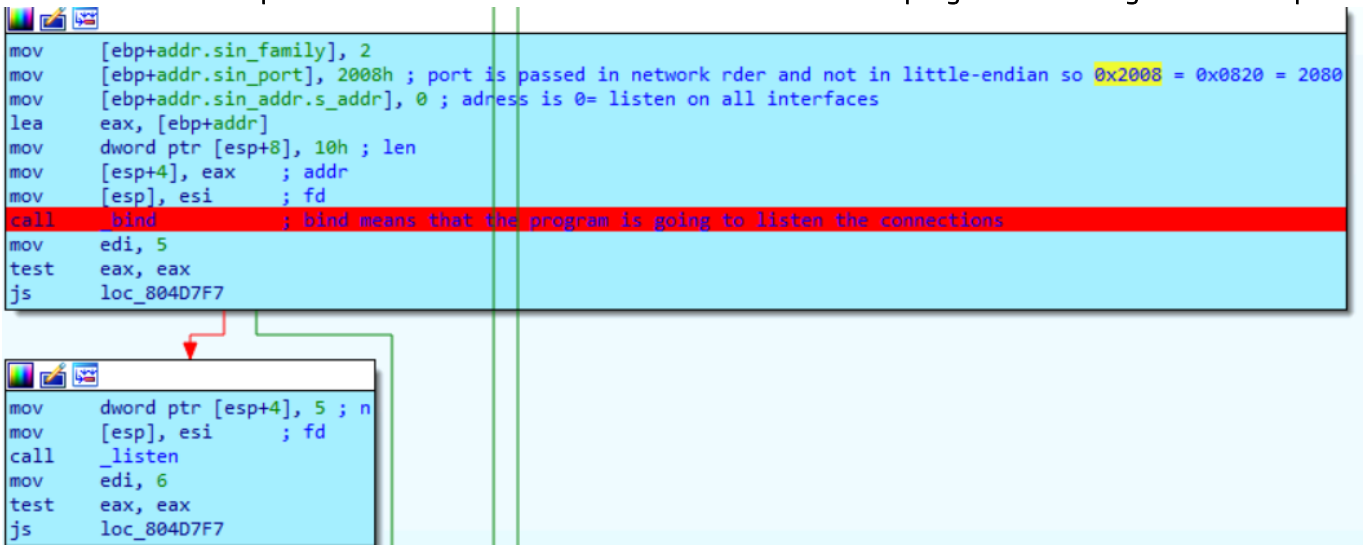
Before:



After:

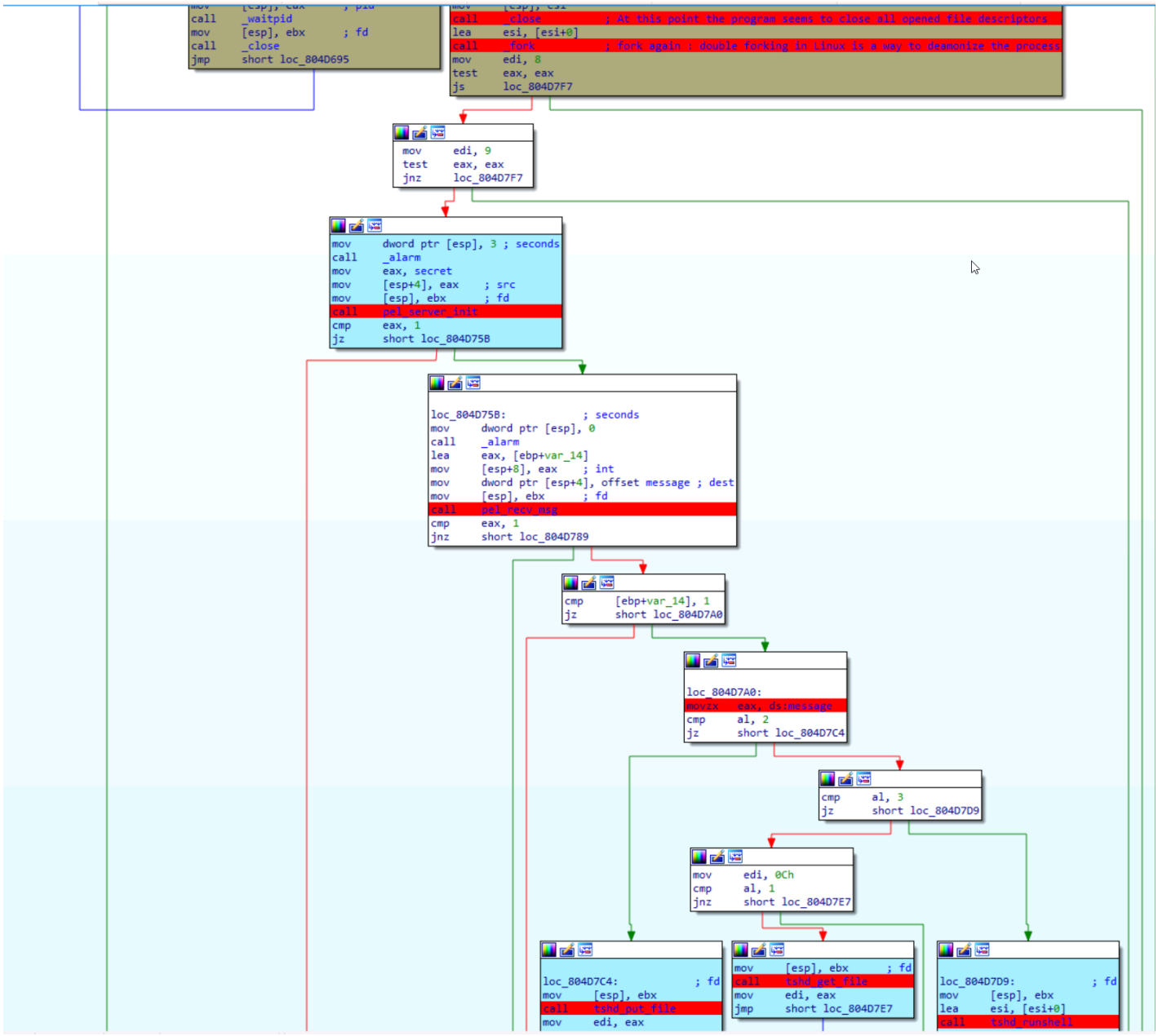


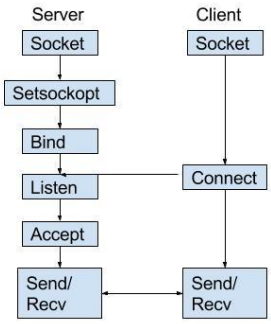
The address is passed in network order and not in little-endian so the program is listening on the 2080 port



Helpful Tips

We see below another _fork call. In Linux, double forking is a way to deamonize a process, so that, if the parent dies, then the child process will still keep running

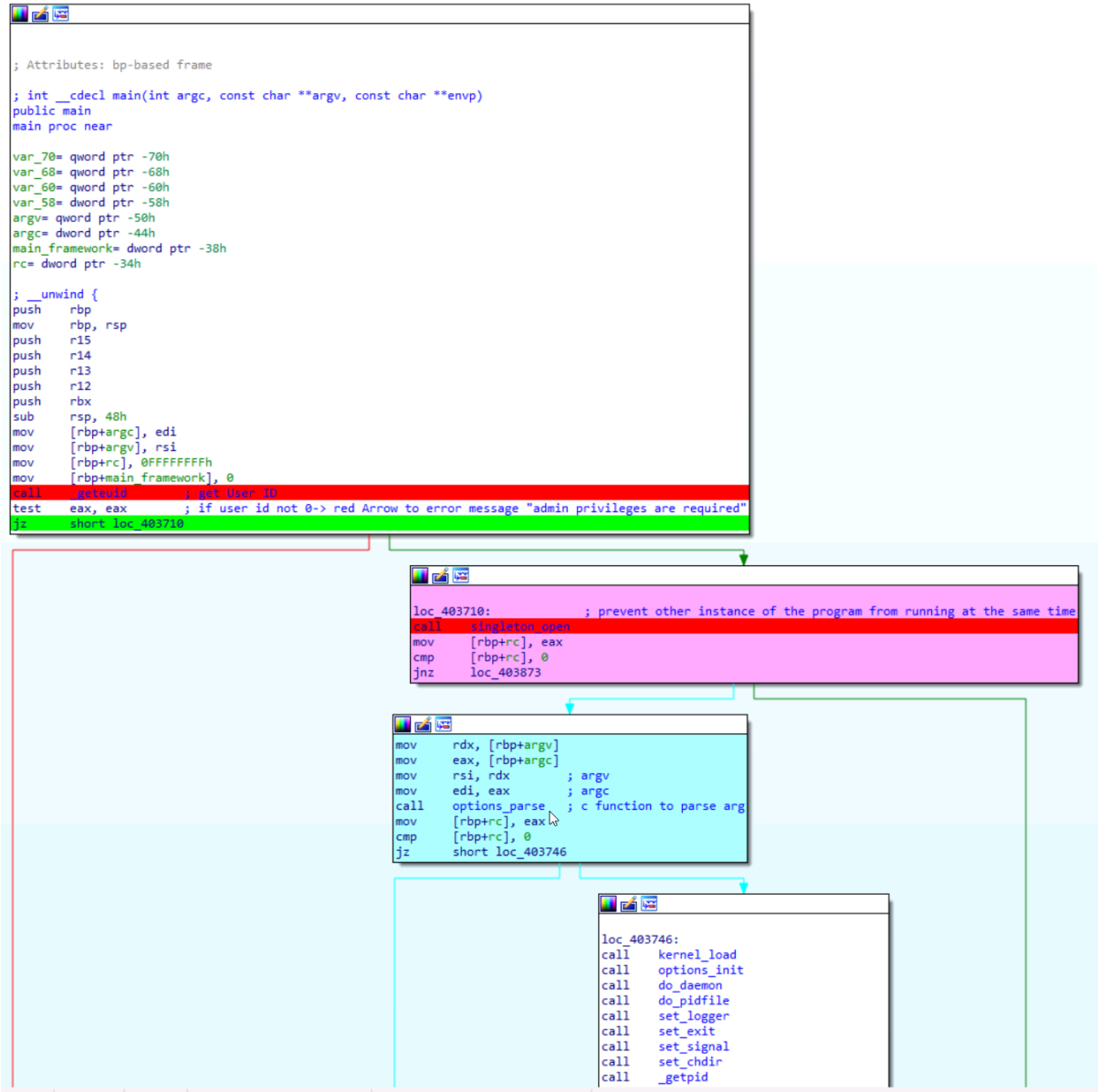

call _waitpid
mov [esp], ebx ; fd
call _close
jmp short loc_804D695
call _close ; At this point the program seems to close all opened file descriptors
lea esi, [esi+0]
call _fork ; fork again : double forking in Linux is a way to deamonize the process
mov edi, 8
test eax, eax
js loc_804D7F7
mov edi, 9
test eax, eax
jnz loc_804D7F7
mov dword ptr [esp], 3 ; seconds
call _alarm
mov eax, secret
mov [esp+4], eax ; src
mov [esp], ebx ; fd
call pel_server_init
cmp eax, 1
jz short loc_804D75B
loc_804D75B: ; seconds
mov dword ptr [esp], 0
call _alarm
lea eax, [ebp+var_14]
mov [esp+8], eax ; int
mov dword ptr [esp+4], offset message ; dest
mov [esp], ebx ; fd
call pel_recv_msg
cmp eax, 1
jnz short loc_804D789
cmp [ebp+var_14], 1
jz short loc_804D7A0
loc_804D7A0:
movzx eax, dsimessage
cmp al, 2
jz short loc_804D7C4
cmp al, 3
jz short loc_804D7D9
mov edi, 0Ch
cmp al, 1
jnz short loc_804D7E7
loc_804D7C4: ; fd
mov [esp], ebx
call tshd_put_file
mov edi, eax
mov [esp], ebx ; fd
call tshd_get_file
mov edi, eax
jmp short loc_804D7E7
loc_804D7D9: ; fd
mov [esp], ebx
lea esi, [esi+0]
call tshd_runshell

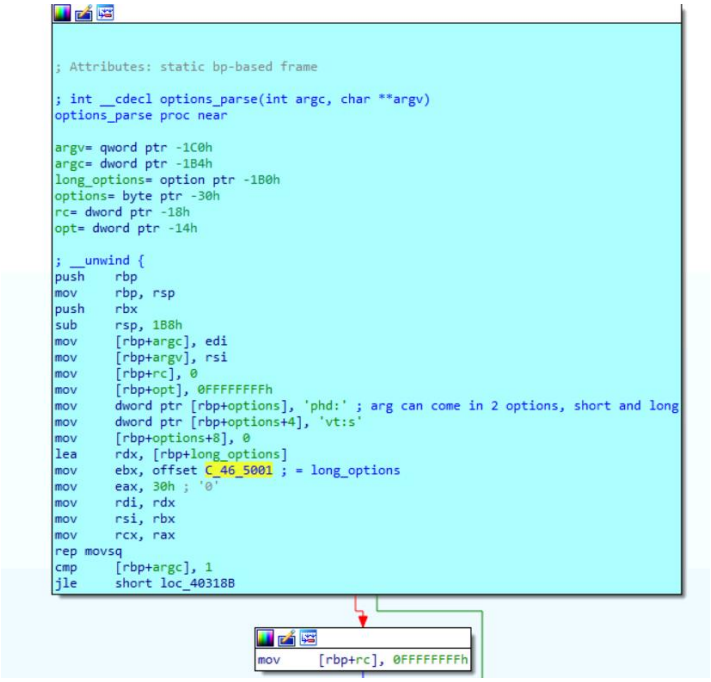

Server
Client
Socket
Socket
Setsockopt
Bind
Listen
Connect
Accept
Send/ Recv
Send/ Recv

Reverse-engineer the second user-land component (snoopy client)

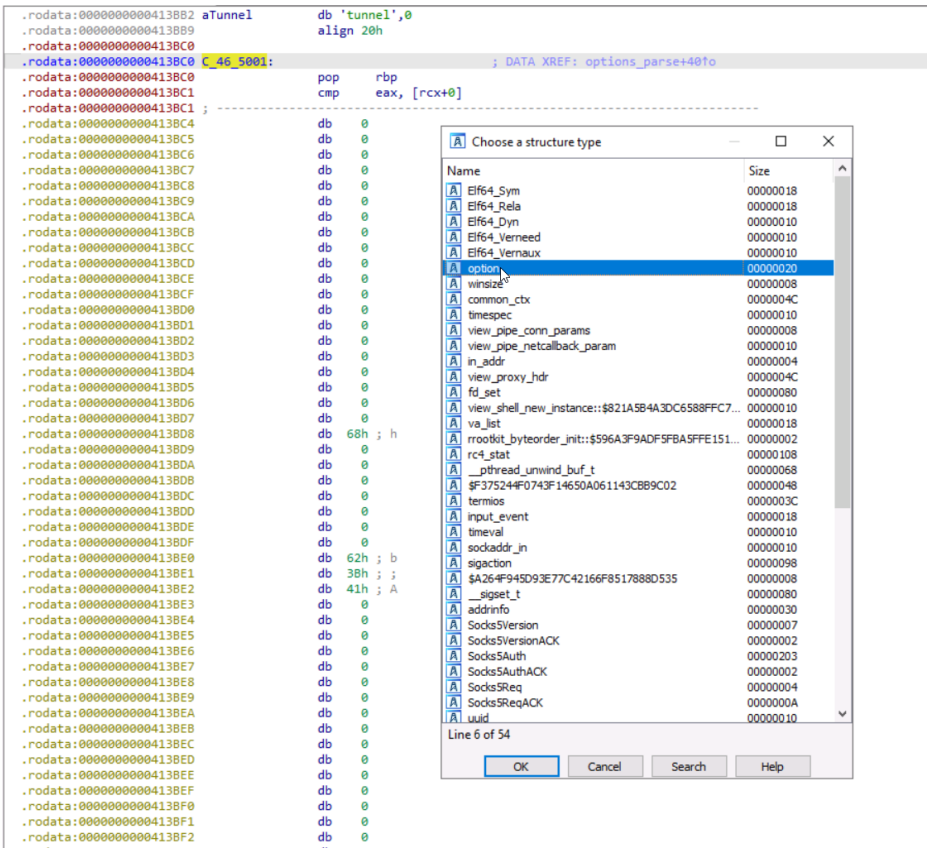
- Event-based logic malwares has non-linear execution flow based on callbacks, which makes the analysis very challenging. **xrefs just don't work** ☹
- Linux Calling convention : **RDI, RSI, RDX, RCX, r8, r9**

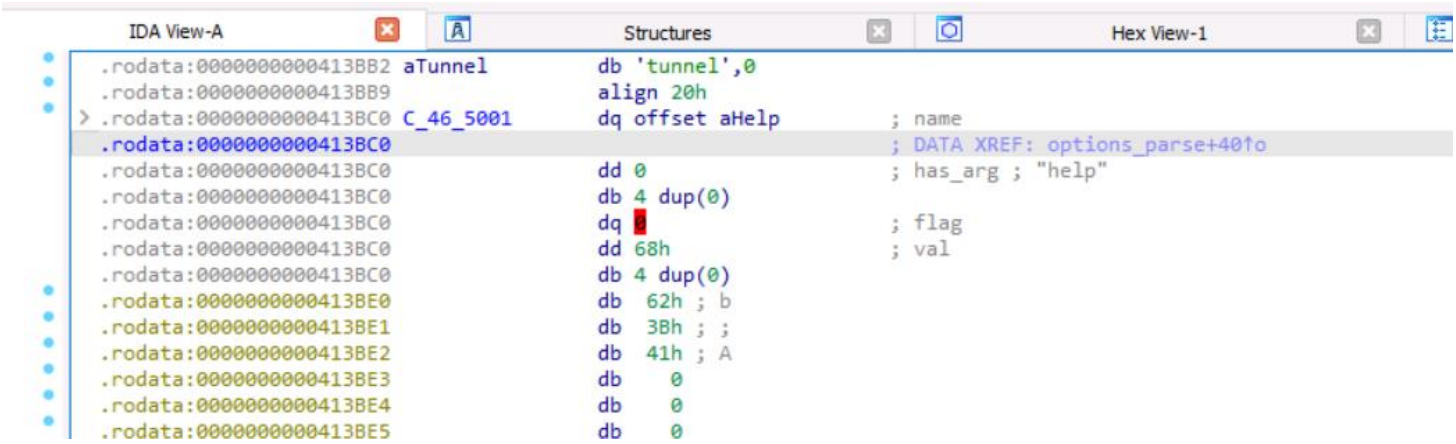
The option_parse function





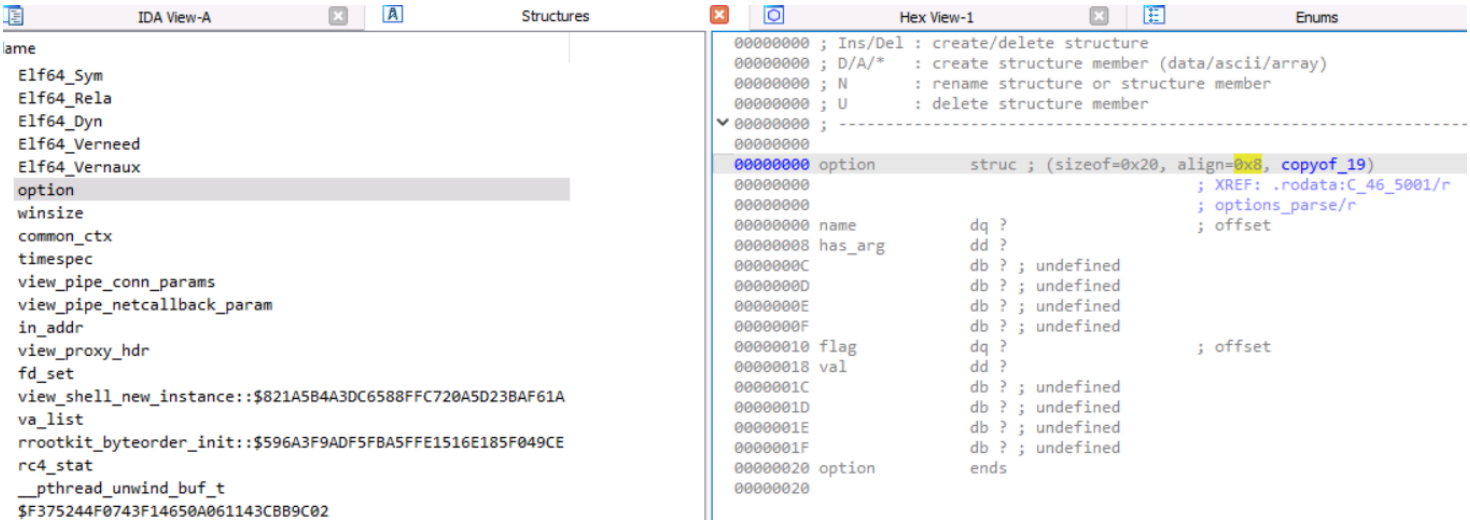
Apply the correct structure [Alt+q]





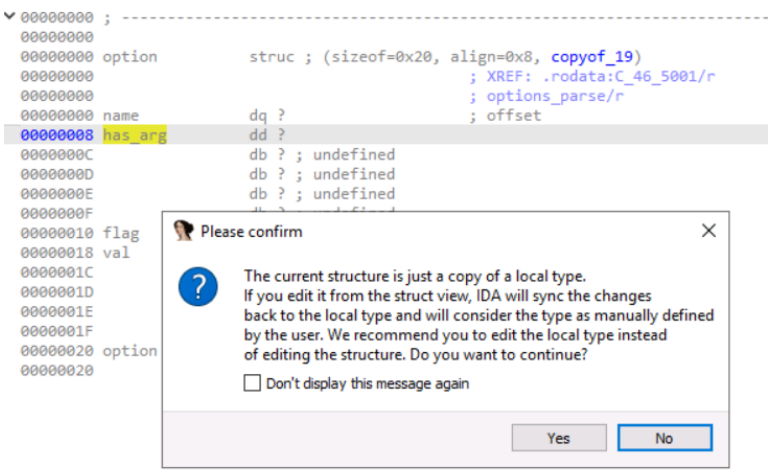
Why this 0 in red?

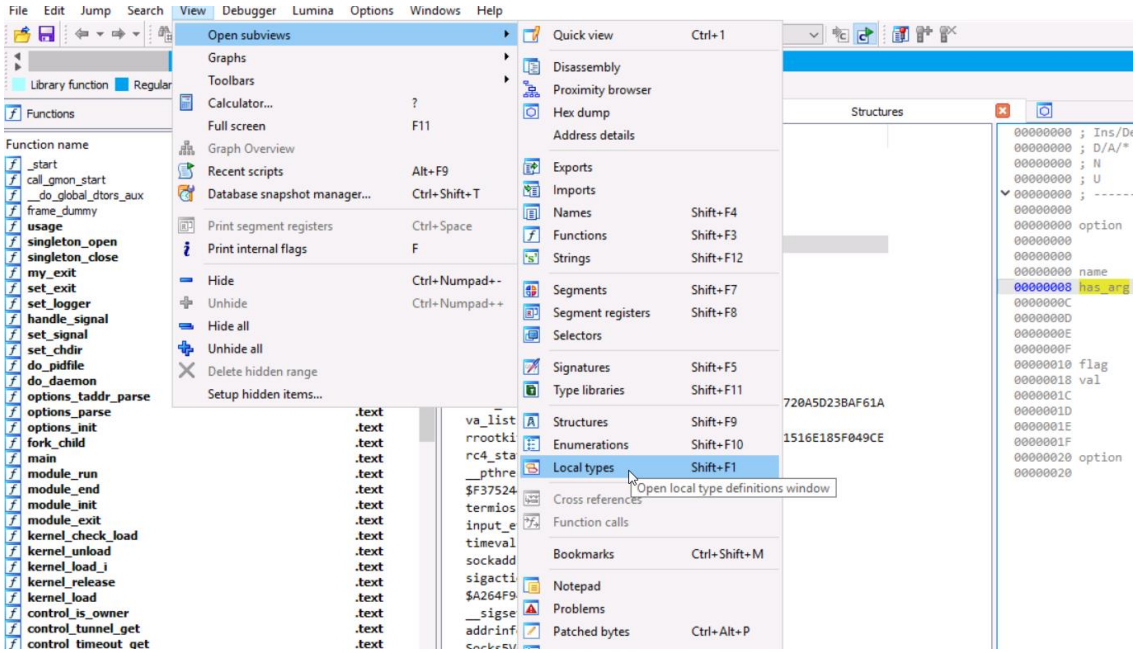
In the structure you can see Align=0x8 [alignment of 8] but in the middle you have padding bytes

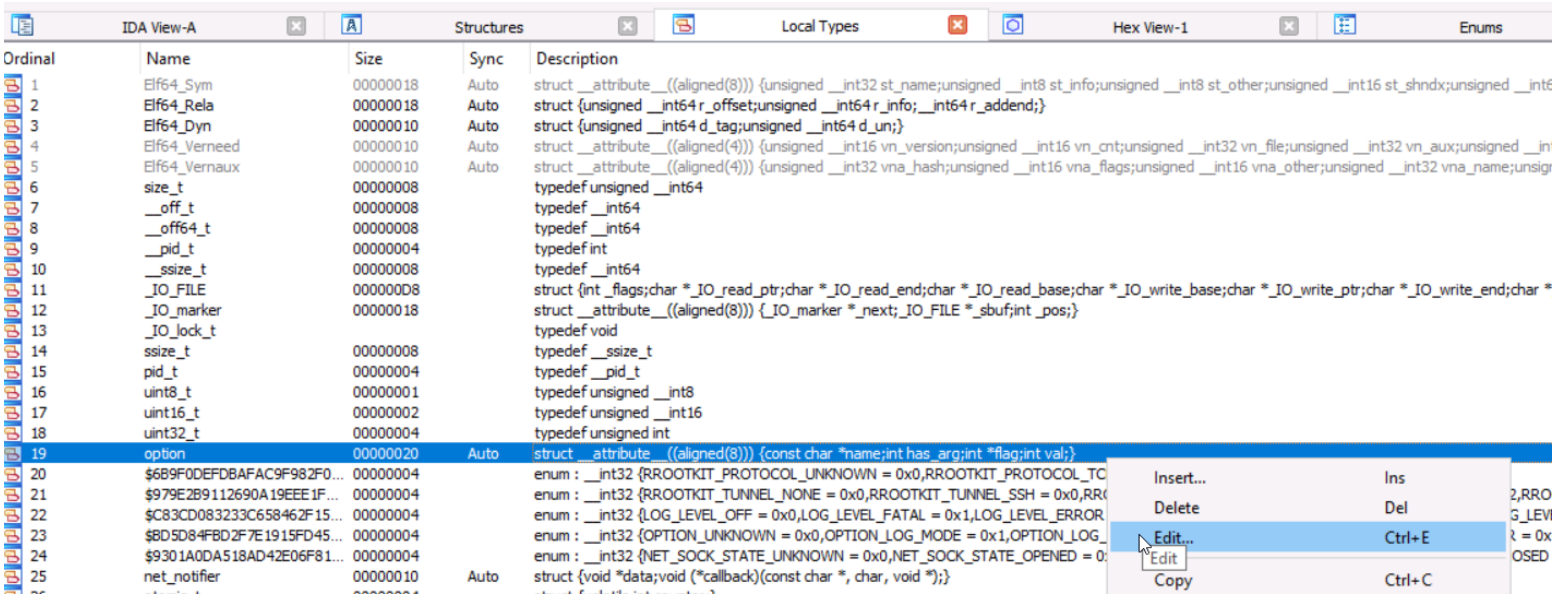


Helpful Tips

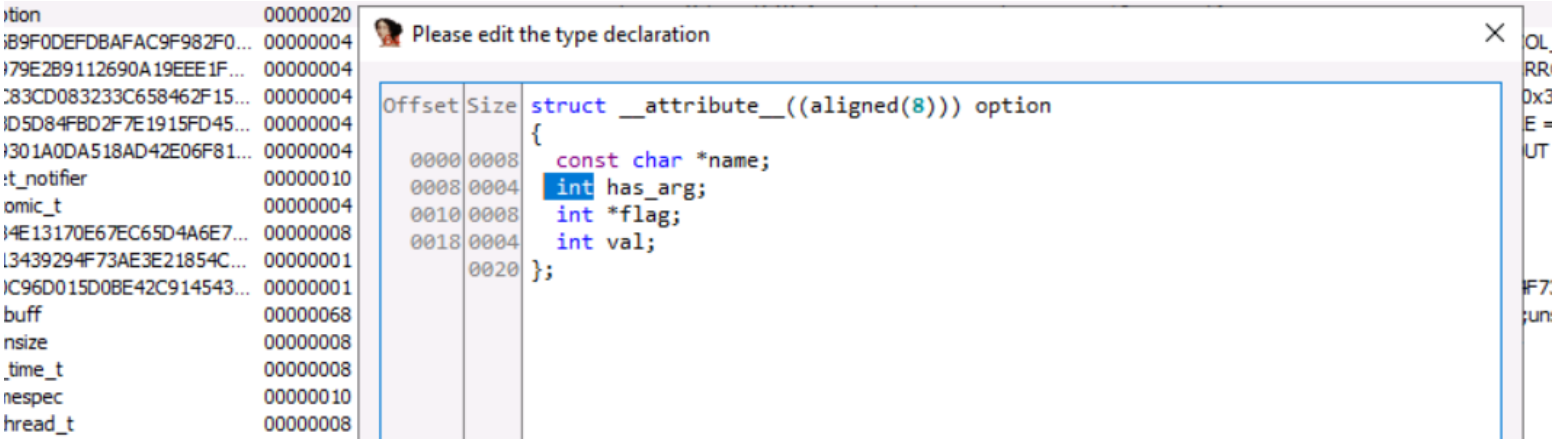
Let's improve the rendering, updating this structure. To do that, press D to change the data type


00000000 ; ---
00000000
00000000 option struc ; (sizeof=0x20, align=0x8, copyof_19)
00000000 ; XREF: .rodata:C_46_5001/r
00000000 ; options_parse/r
00000000 name dq ? ; offset
00000008 has_arg dd ?
0000000C db ? ; undefined
0000000D db ? ; undefined
0000000E db ? ; undefined
00000010 flag
00000018 val
0000001C
0000001D
0000001E
0000001F
00000020 option
00000020
Please confirm
The current structure is just a copy of a local type.
If you edit it from the struct view, IDA will sync the changes
back to the local type and will consider the type as manually defined
by the user. We recommend you to edit the local type instead
of editing the structure. Do you want to continue?
Don't display this message again
Yes
No

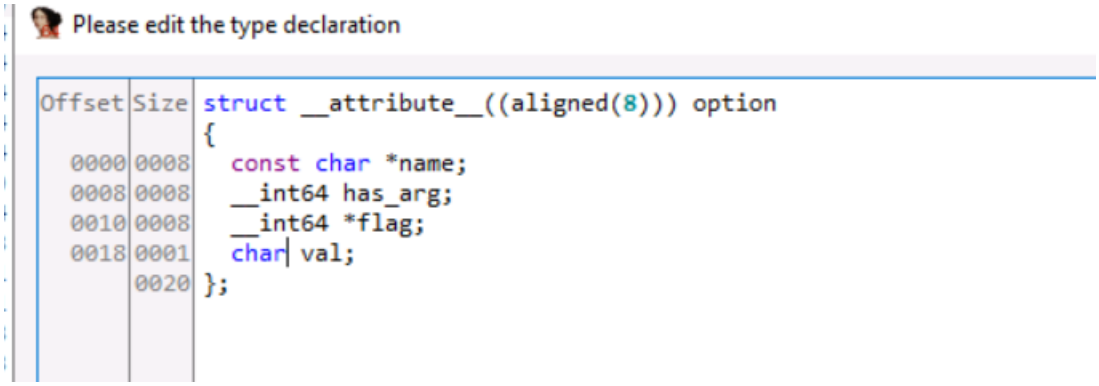

File Edit Jump Search View Debugger Lumina Options Windows Help
Open subviews
Graphs
Toolbars
Calculator... ?
Full screen F11
Graph Overview
Recent scripts Alt+F9
Database snapshot manager... Ctrl+Shift+T
Print segment registers Ctrl+Space
Print internal flags F
Hide Ctrl+Numpad+-
Unhide Ctrl+Numpad++
Hide all
Unhide all
Delete hidden range
Setup hidden items...
Quick view Ctrl+1
Disassembly
Proximity browser
Hex dump
Address details
Exports
Imports
Names Shift+F4
Functions Shift+F3
Strings Shift+F12
Segments Shift+F7
Segment registers Shift+F8
Selectors
Signatures Shift+F5
Type libraries Shift+F11
Structures Shift+F9
Enumerations Shift+F10
Local types Shift+F1
Open local type definitions window
Cross references
Function calls
Bookmarks Ctrl+Shift+M
Notepad
Problems
Patched bytes Ctrl+Alt+P
Library function
Functions
Function name
Structures



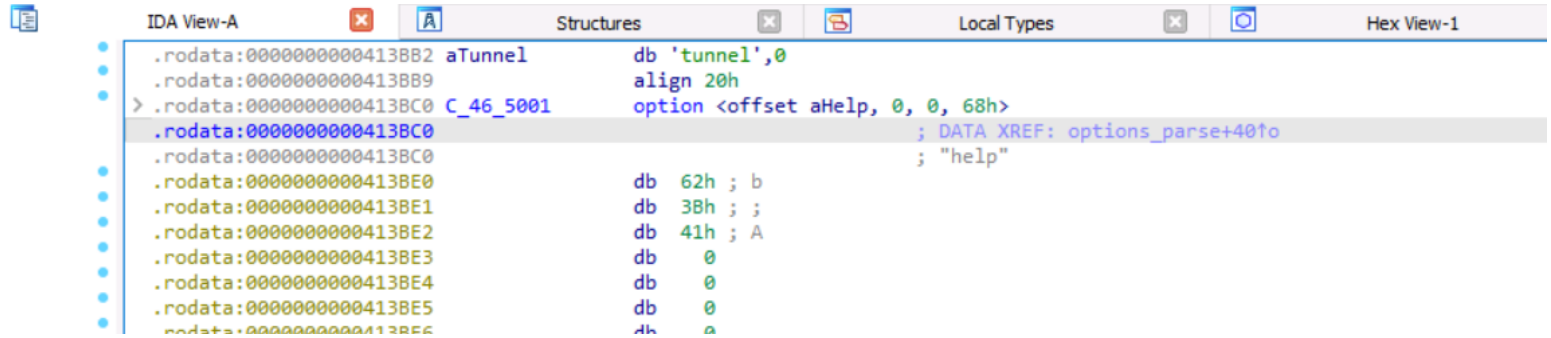
Before:



After:

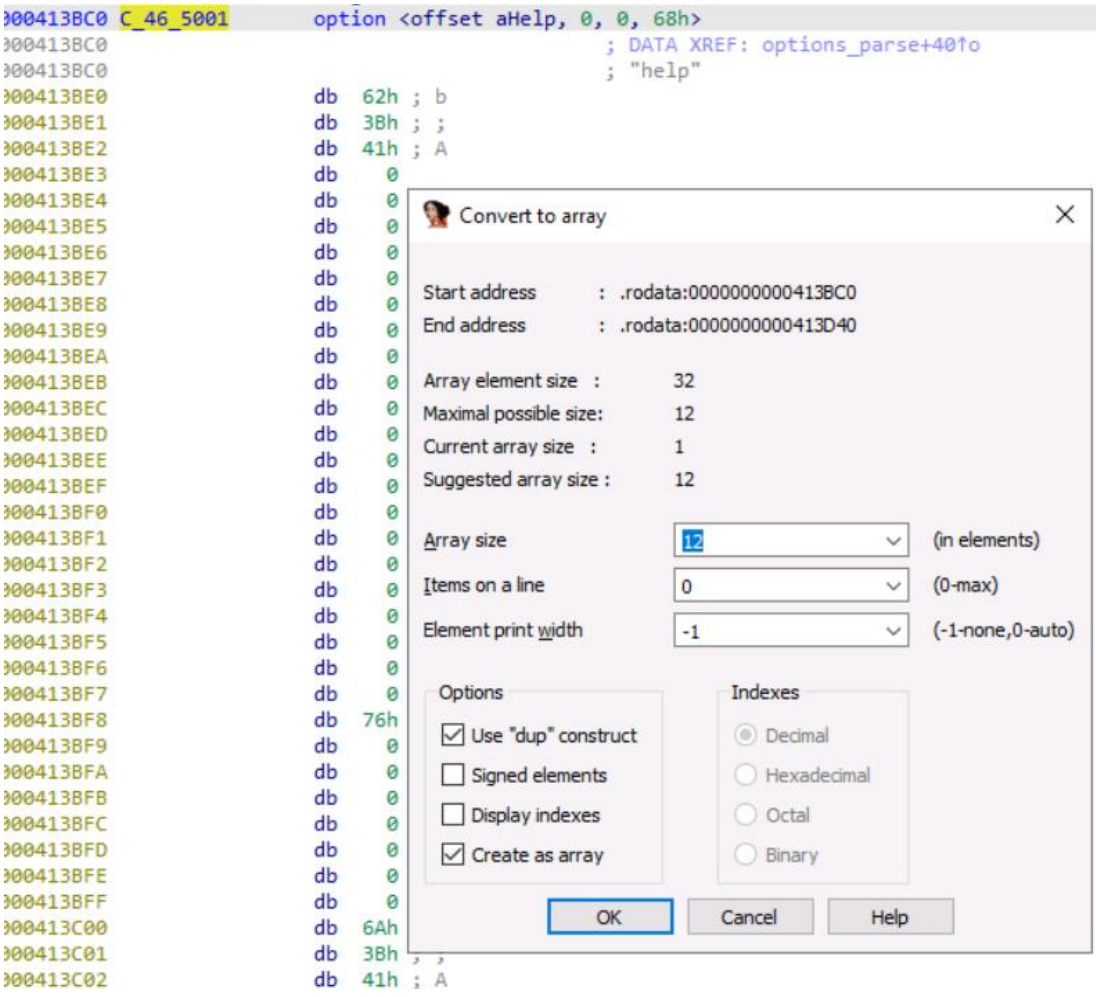


Result:

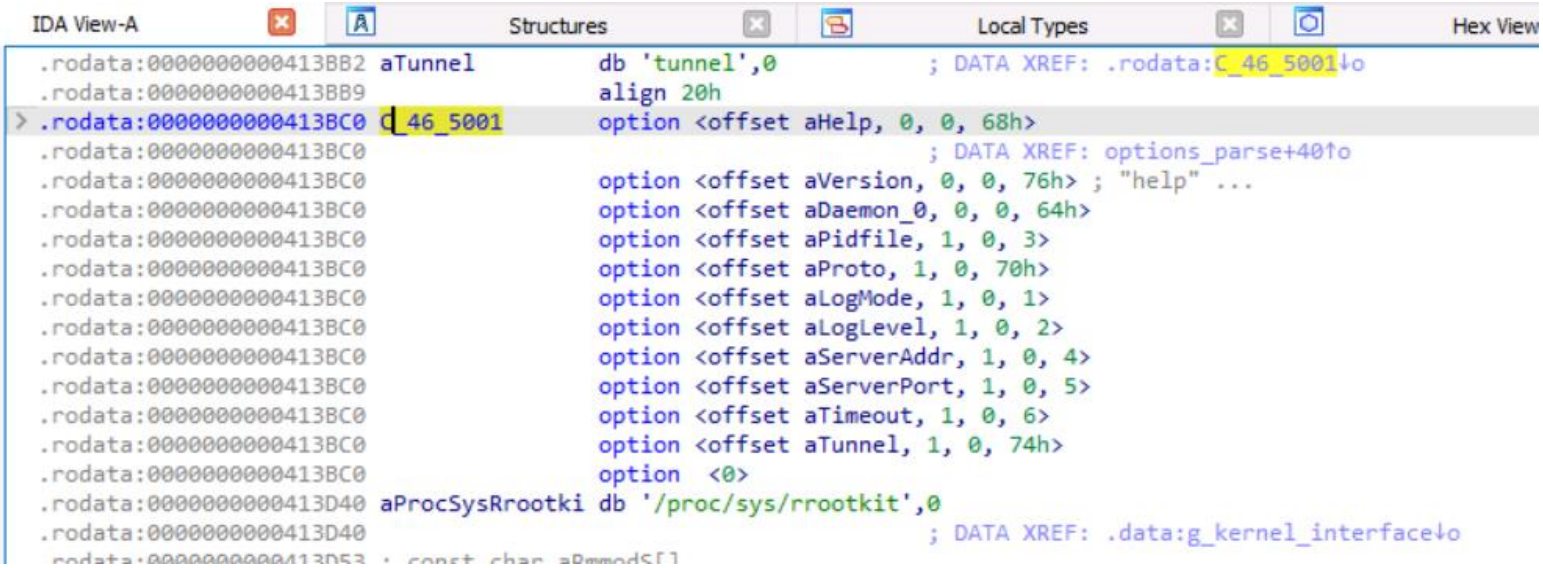


Let's do it for all the structures here. Because, actually, this long option is a series of those structures, and it stops when you reach the first null structure.

Press the *** key** to get the "convert to array" window



Note that ID knows that the length of the array is 12. Press ok



And know the arguments are displayed properly

Before:

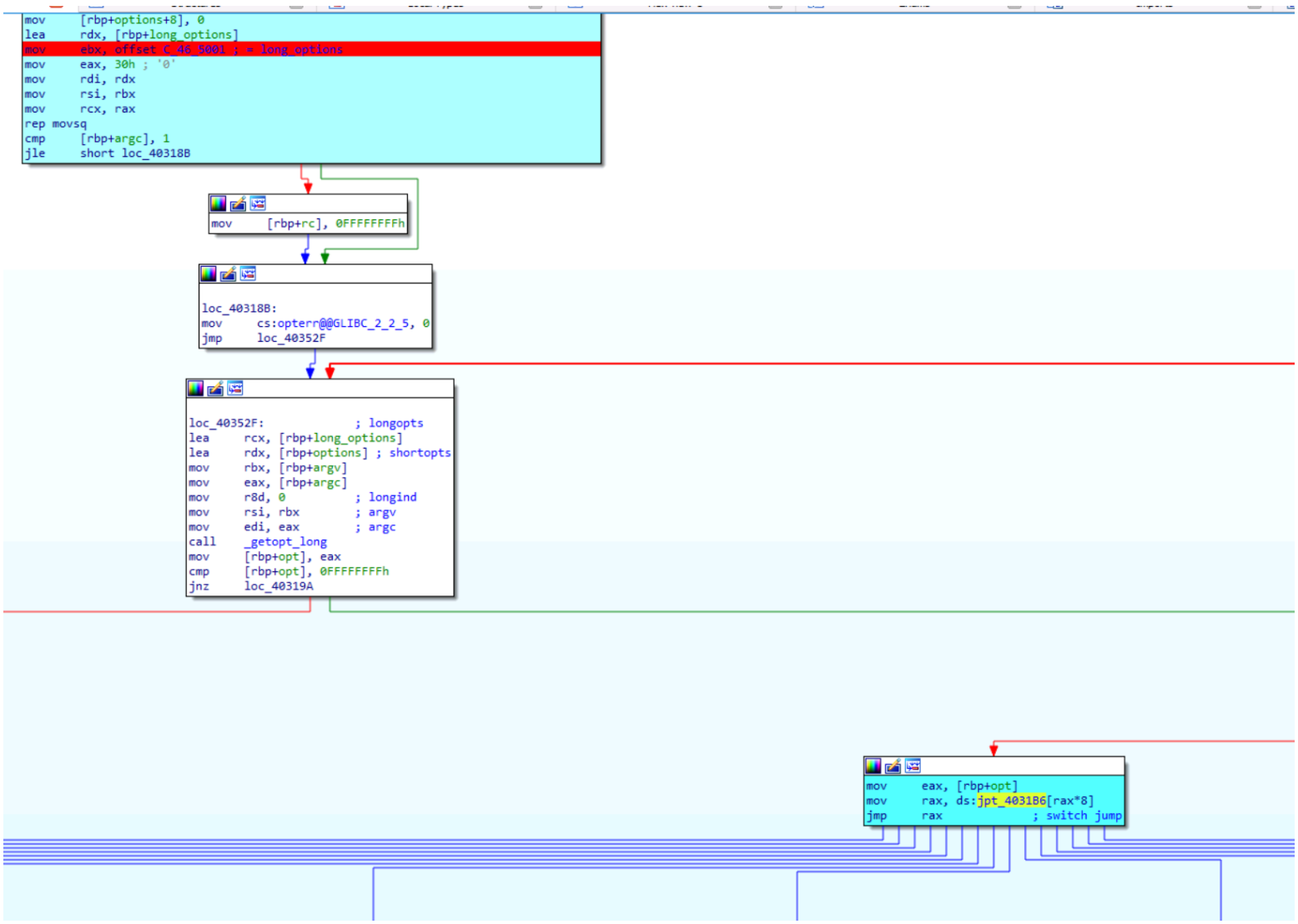
```
rodata:0000000000413BC0 C_46_5001       option <offset aHelp, 0, 0, 68h>
rodata:0000000000413BC0                                         ; DATA XREF: options_parse+40↑o
rodata:0000000000413BC0                                         ; "help"
rodata:0000000000413BE0                 db  62h ; b
rodata:0000000000413BE1                 db  38h ; ;
rodata:0000000000413BE2                 db  41h ; A
rodata:0000000000413BE3                 db    0
rodata:0000000000413BE4                 db    0
rodata:0000000000413BE5                 db    0
rodata:0000000000413BE6                 db    0
rodata:0000000000413BE7                 db    0
rodata:0000000000413BE8                 db    0
rodata:0000000000413BE9                 db    0
rodata:0000000000413BEA                 db    0
rodata:0000000000413BEB                 db    0
rodata:0000000000413BEC                 db    0
rodata:0000000000413BED                 db    0
rodata:0000000000413BEE                 db    0
rodata:0000000000413BEF                 db    0
rodata:0000000000413BF0                 db    0
rodata:0000000000413BF1                 db    0
rodata:0000000000413BF2                 db    0
rodata:0000000000413BF3                 db    0
rodata:0000000000413BF4                 db    0
rodata:0000000000413BF5                 db    0
rodata:0000000000413BF6                 db    0
rodata:0000000000413BF7                 db    0
rodata:0000000000413BF8                 db  76h ; v
rodata:0000000000413BF9                 db    0
rodata:0000000000413BFA                 db    0
rodata:0000000000413BFB                 db    0
rodata:0000000000413BFC                 db    0
rodata:0000000000413BFD                 db    0
rodata:0000000000413BFE                 db    0
rodata:0000000000413BFF                 db    0
rodata:0000000000413C00                 db  6Ah ; j
rodata:0000000000413C01                 db  3Bh ; ;
rodata:0000000000413C02                 db  41h ; A
rodata:0000000000413C03                 db    0
rodata:0000000000413C04                 db    0
rodata:0000000000413C05                 db    0
rodata:0000000000413C06                 db    0
rodata:0000000000413C07                 db    0
rodata:0000000000413C08                 db    0
rodata:0000000000413C09                 db    0
rodata:0000000000413C0A                 db    0
rodata:0000000000413C0B                 db    0
rodata:0000000000413C0C                 db    0
rodata:0000000000413C0D                 db    0
rodata:0000000000413C0E                 db    0
rodata:0000000000413C0F                 db    0
rodata:0000000000413C10                 db    0
rodata:0000000000413C11                 db    0
rodata:0000000000413C12                 db    0
rodata:0000000000413C13                 db    0
rodata:0000000000413C14                 db    0
rodata:0000000000413C15                 db    0
rodata:0000000000413C16                 db    0
rodata:0000000000413C17                 db    0
rodata:0000000000413C18                 db  64h ; d
rodata:0000000000413C19                 db    0
rodata:0000000000413C1A                 db    0
rodata:0000000000413C1B                 db    0
rodata:0000000000413C1C                 db    0
rodata:0000000000413C1D                 db    0
rodata:0000000000413C1E                 db    0
rodata:0000000000413C1F                 db    0
rodata:0000000000413C20                 db  71h ; q
rodata:0000000000413C21                 db  3Bh ; ;
```

After:

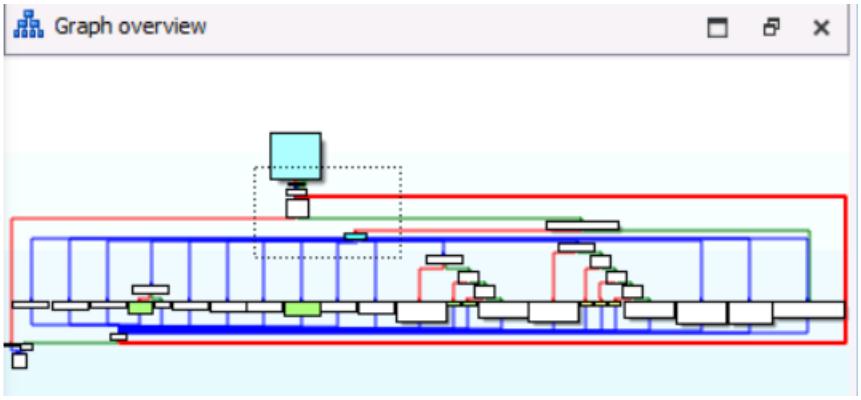
```
IDA View-A | Structures | Local Types
.rodata:0000000000413BB2 aTunnel         db 'tunnel',0          ; DATA XREF: .rodata:C_46_5001↓o
.rodata:0000000000413BB9                 align 20h
.rodata:0000000000413BC0 C_46_5001       option <offset aHelp, 0, 0, 68h>
.rodata:0000000000413BC0                                        ; DATA XREF: options_parse+40↑o
.rodata:0000000000413BC0                 option <offset aVersion, 0, 0, 76h> ; "help" ...
.rodata:0000000000413BC0                 option <offset aDaemon_0, 0, 0, 64h>
.rodata:0000000000413BC0                 option <offset aPidfile, 1, 0, 3>
.rodata:0000000000413BC0                 option <offset aProto, 1, 0, 70h>
.rodata:0000000000413BC0                 option <offset aLogMode, 1, 0, 1>
.rodata:0000000000413BC0                 option <offset aLogLevel, 1, 0, 2>
.rodata:0000000000413BC0                 option <offset aServerAddr, 1, 0, 4>
.rodata:0000000000413BC0                 option <offset aServerPort, 1, 0, 5>
.rodata:0000000000413BC0                 option <offset aTimeout, 1, 0, 6>
.rodata:0000000000413BC0                 option <offset aTunnel, 1, 0, 74h>
.rodata:0000000000413BC0                 option  <0>
.rodata:0000000000413D40 aProcSysRrootki db '/proc/sys/rrootkit',0
.rodata:0000000000413D40                                        ; DATA XREF: .data:g_kernel_interface↓o
.rodata:0000000000413D53 ; const char aRmmodS[]
.rodata:0000000000413D53 aRmmodS         db 'rmmod %s',0        ; DATA XREF: kernel_unload+30↑o
.rodata:0000000000413D5C aRrtkernelKo    db 'rrtkernel.ko',0    ; DATA XREF: kernel_unload+39↑o
.rodata:0000000000413D5C                                        ; kernel_release+133↑o
.rodata:0000000000413D69 ; const char aInsmodS2DevNul[]
.rodata:0000000000413D69 aInsmodS2DevNul db 'insmod %s 2>/dev/null',0
.rodata:0000000000413D69                                        ; DATA XREF: kernel_load_i+40↑o
.rodata:0000000000413D7F ; const char path[]
.rodata:0000000000413D7F path            db '/proc/self/exe',0  ; DATA XREF: kernel_release+6B↑o
.rodata:0000000000413D8E                 align 10h
.rodata:0000000000413D90 ; const char aSFailedToFindM[]
.rodata:0000000000413D90 aSFailedToFindM db '%s: failed to find myself. count = %d.',0
.rodata:0000000000413D90                                        ; DATA XREF: kernel_release+A3↑o
.rodata:0000000000413DB7                 align 8
.rodata:0000000000413DB8 ; const char aSFailedToOpenM[]
.rodata:0000000000413DB8 aSFailedToOpenM db '%s: failed to open myself. error = %s.',0
.rodata:0000000000413DB8                                        ; DATA XREF: kernel_release+F4↑o
.rodata:0000000000413DDF ; const char aTmpS[]
.rodata:0000000000413DDF aTmpS           db '/tmp/%s',0         ; DATA XREF: kernel_release+127↑o
.rodata:0000000000413DE7                 align 8
.rodata:0000000000413DE8 ; const char aSFailedToOpenK[]
.rodata:0000000000413DE8 aSFailedToOpenK db '%s: failed to open kernel. error = %s.',0
.rodata:0000000000413DE8                                        ; DATA XREF: kernel_release+194↑o
.rodata:0000000000413E0F                 align 10h
.rodata:0000000000413E10 ; const char aSFailedToWrite[]
.rodata:0000000000413E10 aSFailedToWrite db '%s: failed to write kernel. error = %s.',0
.rodata:0000000000413E10                                        ; DATA XREF: kernel_release+1DE↑o
.rodata:0000000000413E38 ; const char aSExitRcD[]
.rodata:0000000000413E38 aSExitRcD       db '%s: exit. rc = %d',0Ah,0
.rodata:0000000000413E38                                        ; DATA XREF: kernel_release+266↑o
.rodata:0000000000413E4B ; Function-local static variable
.rodata:0000000000413E4B ; const char _func___4133[15]
.rodata:0000000000413E4B __func___4133   db 'kernel_release',0  ; DATA XREF: kernel_release+9E↑o
.rodata:0000000000413E4B                                        ; kernel_release+EF↑o ...
.rodata:0000000000413E5A                 align 20h
.rodata:0000000000413E60 ; const char aControlTheNetw[]
.rodata:0000000000413E60 aControlTheNetw db 'control: the network node is timeout.',0
.rodata:0000000000413E60                                        ; DATA XREF: control_net_callback+21↑o
.rodata:0000000000413E86                 align 8
.rodata:0000000000413E88 ; const char aFrameworkEnter[]
.rodata:0000000000413E88 aFrameworkEnter db 'framework: enter into framework-loop.',0
.rodata:0000000000413E88                                        ; DATA XREF: framework_loop_run+F↑o
.rodata:0000000000413EAE                 align 10h
.rodata:0000000000413EB0 ; const char aFrameworkExitF[]
.rodata:0000000000413EB0 aFrameworkExitF db 'framework: exit from framework-loop.',0
.rodata:0000000000413EB0                                        ; DATA XREF: framework_loop_run+2B↑o
.rodata:0000000000413ED5 ; const char aFailedToInitNe[]
.rodata:0000000000413ED5 aFailedToInitNe db 'failed to init net.',0
.rodata:0000000000413ED5                                        ; DATA XREF: framework_init+1D↑o
.rodata:0000000000413EE9 ; const char aFailedToInitTu[]
.rodata:0000000000413EE9 aFailedToInitTu db 'failed to init tunnel.',0
.rodata:0000000000413EE9                                        ; DATA XREF: framework_init+44↑o
.rodata:0000000000413F00 ; const char aFailedToInitPr[]
.rodata:0000000000413F00 aFailedToInitPr db 'failed to init proto.',0
00013BC0 0000000000413BC0: .rodata:C_46_5001 (Synchronized with Hex View-1)
```

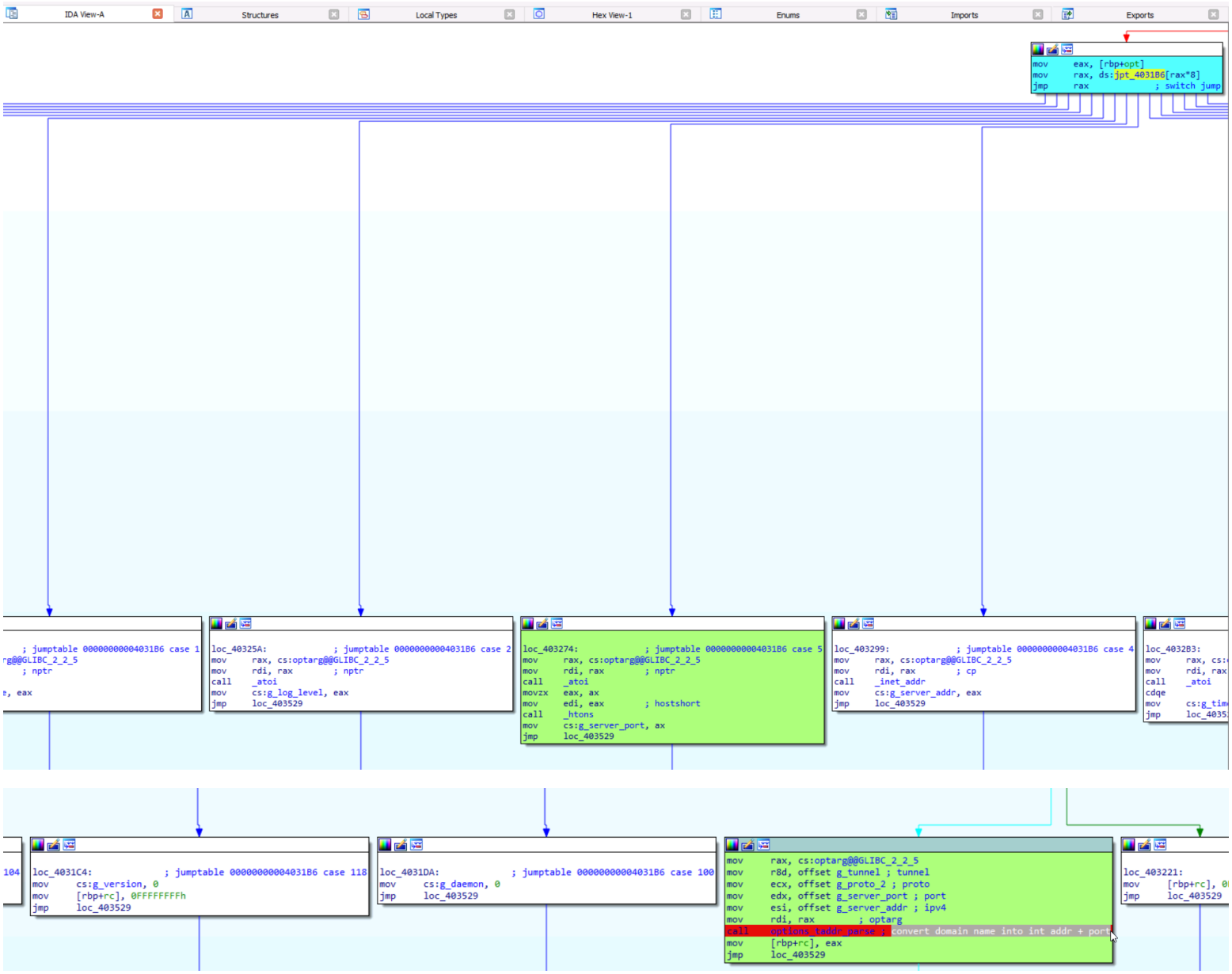
Continue to explore the option_parse

```
loc_40319A:
mov     [rbp+rc], 0
cmp     [rbp+opt], 76h  ; switch 119 cases
ja      def_4031B6      ; jumptable 00000000004031B6 default case, cases 0,7-57,59-62,64-99,101-103,105-111,113,114,117
```



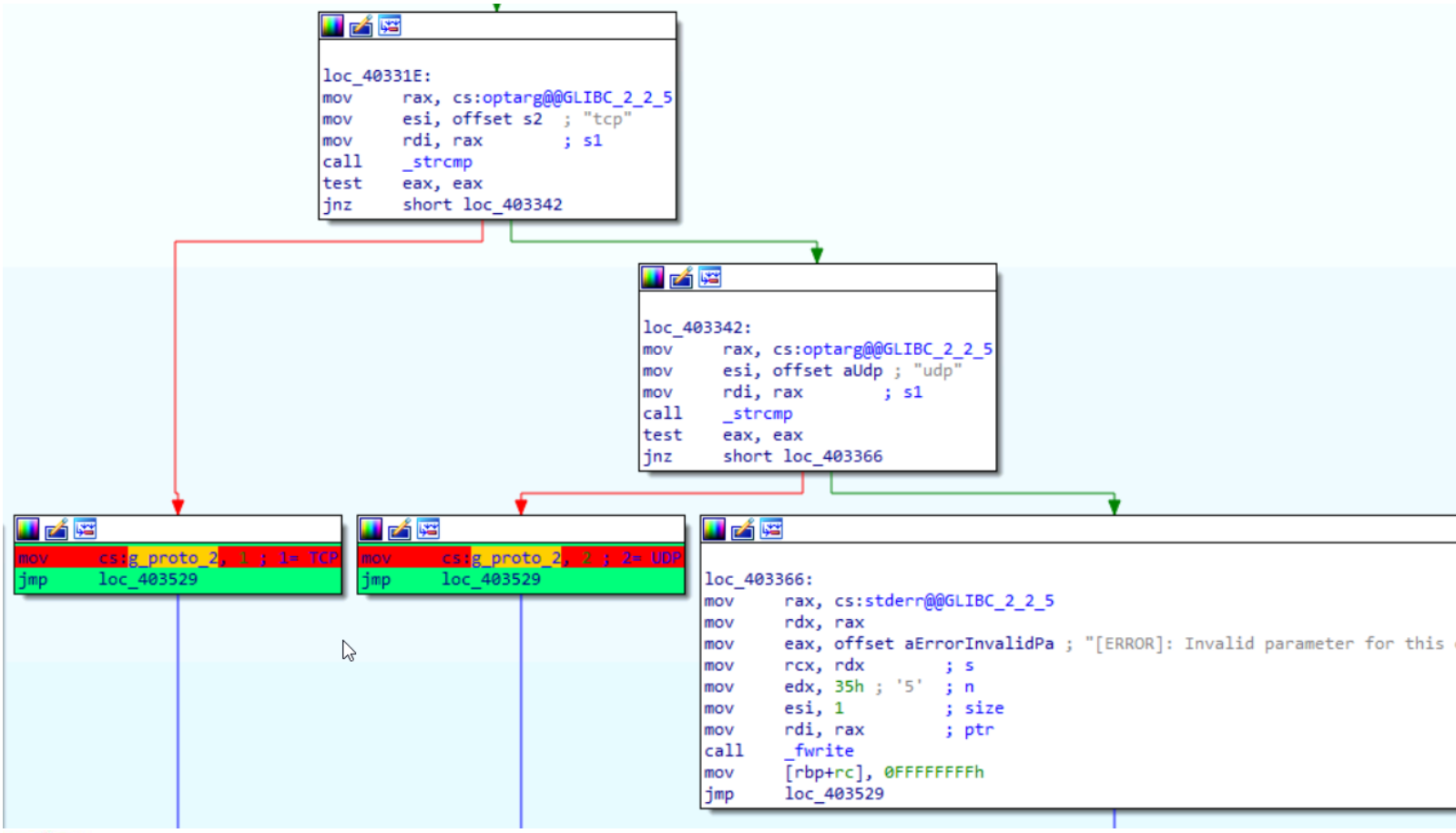
So, this jump table is the giant switch case that it was created in function of arguments which have been detected





The option_taddr_parse convert domain name into addr +port with ATOI. You can see the detail of this very simple function here.

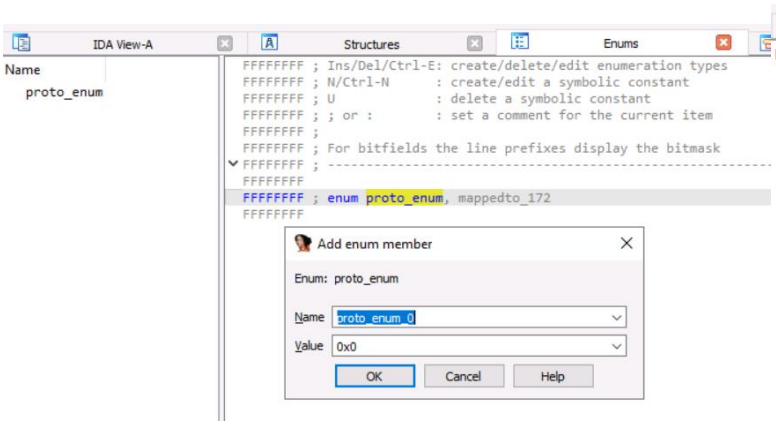

loc_40331E:
mov rax, cs:optarg@@GLIBC_2_2_5
mov esi, offset s2 ; "tcp"
mov rdi, rax ; s1
call _strcmp
test eax, eax
jnz short loc_403342
loc_403342:
mov rax, cs:optarg@@GLIBC_2_2_5
mov esi, offset aUdp ; "udp"
mov rdi, rax ; s1
call _strcmp
test eax, eax
jnz short loc_403366
mov cs:g_proto_2, 1 ; 1= TCP
jmp loc_403529
mov cs:g_proto_2, 2 ; 2= UDP
jmp loc_403529
loc_403366:
mov rax, cs:stderr@@GLIBC_2_2_5
mov rdx, rax
mov eax, offset aErrorInvalidPa ; "[ERROR]: Invalid parameter for this opt"...
mov rcx, rdx ; s
mov edx, 35h ; '5' ; n
mov esi, 1 ; size
mov rdi, rax ; ptr
call _fwrite
mov [rbp+rc], 0FFFFFFFFh
jmp loc_403529



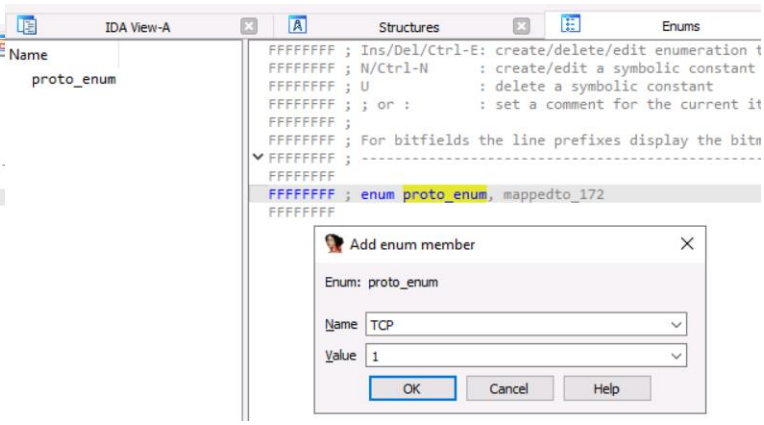
Helpful Tips

to remember tcp is 1 and udp is 2, create an enum [enum window->new enum then click the "n" key]

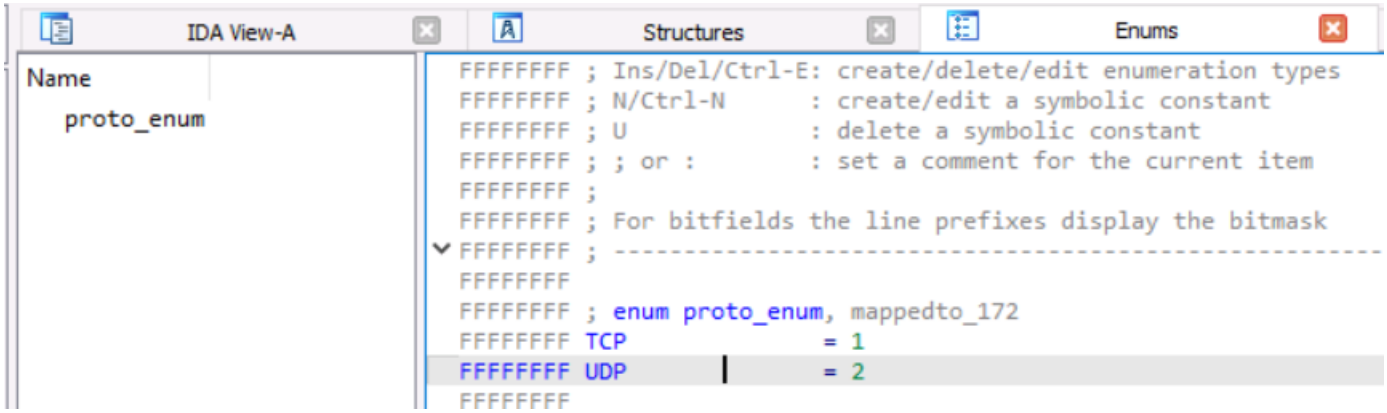
Before:



After:



Result:

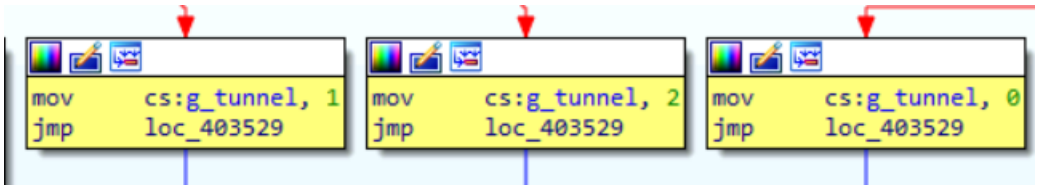


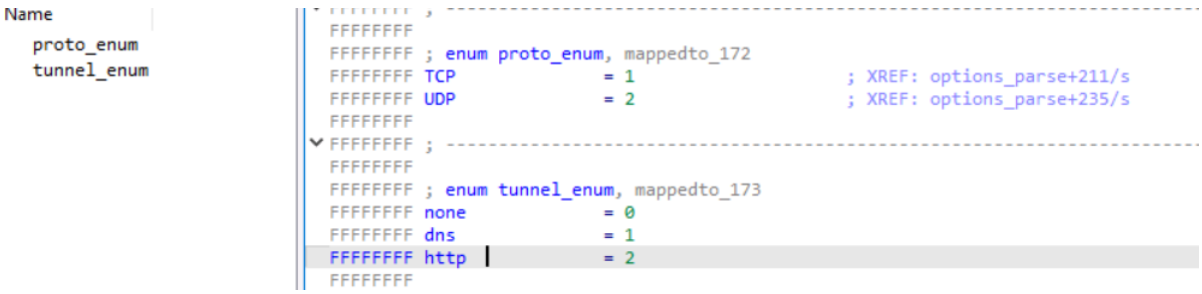
Then go back to the graph view, select the "1" representing tcp and press the "M" and choose our enum

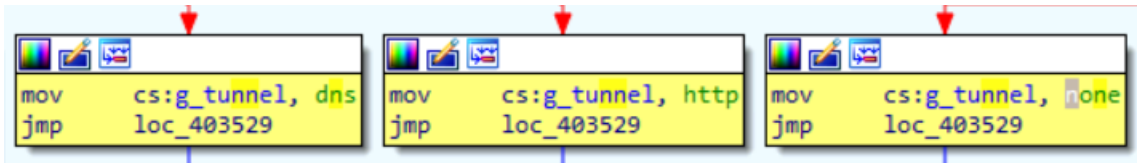
Before:

After:

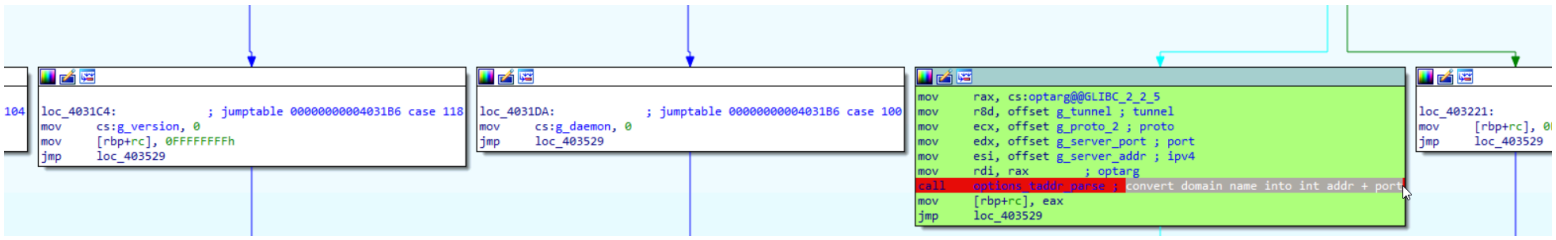
The same with g_tunnel:

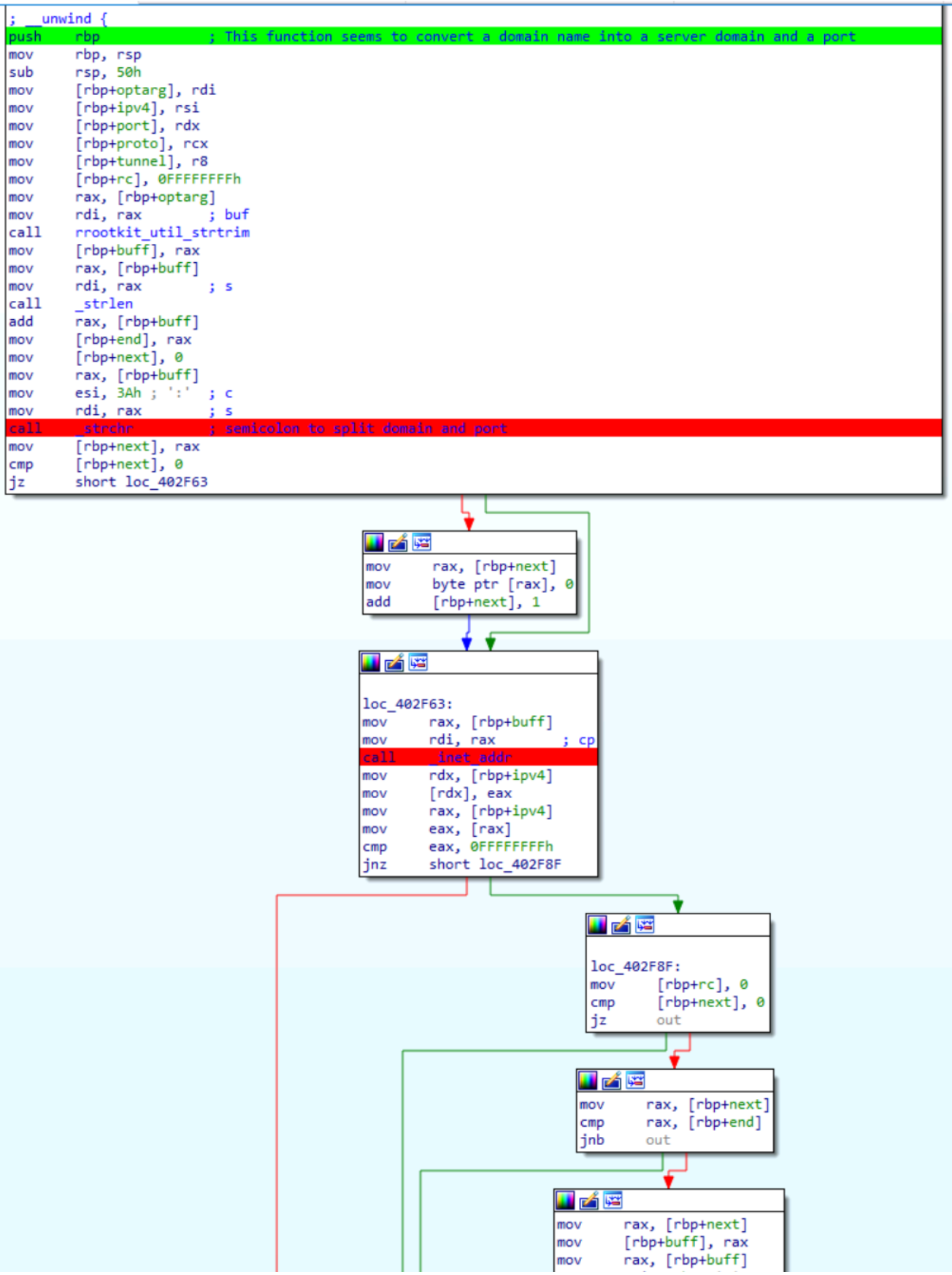


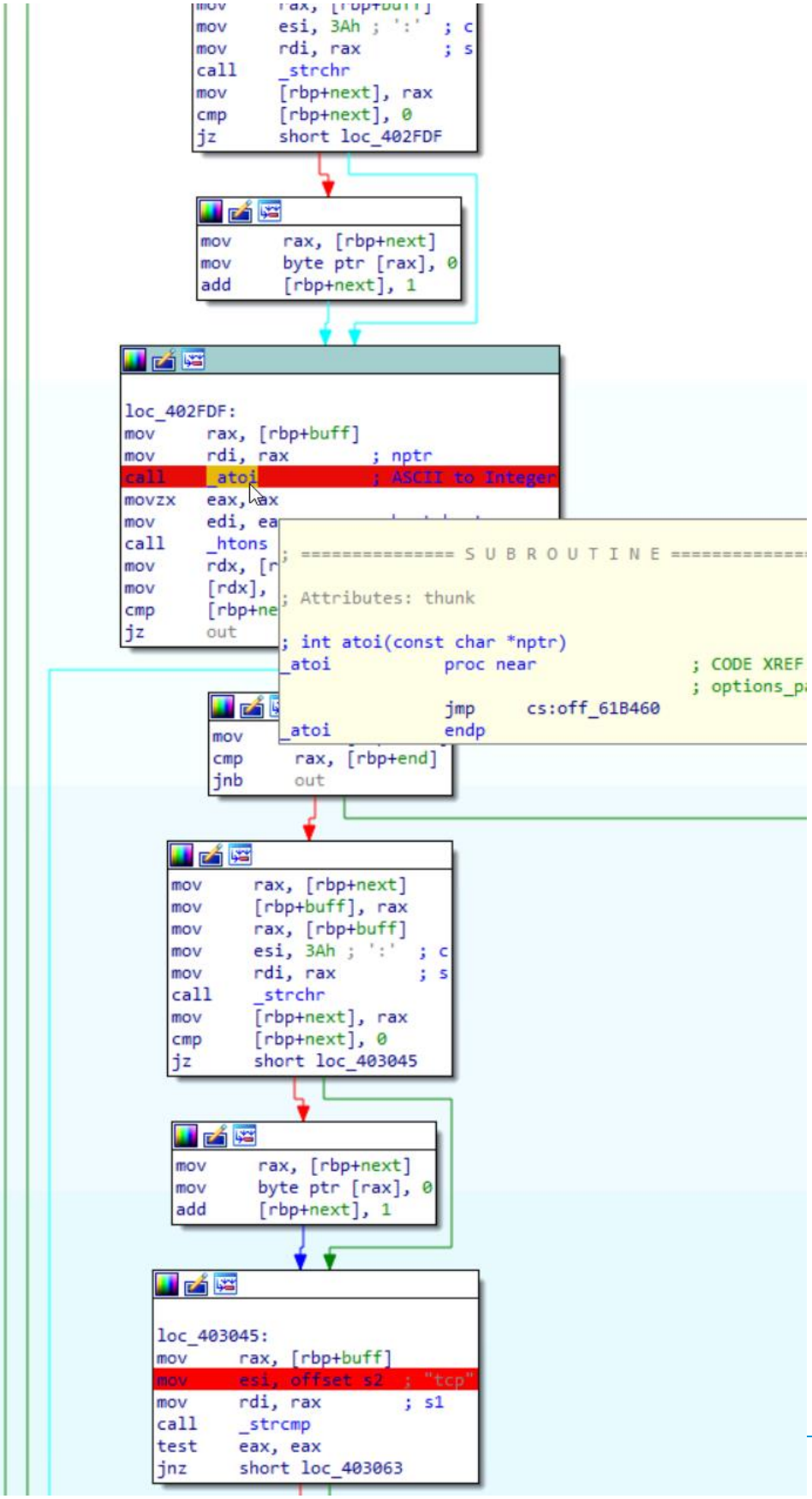


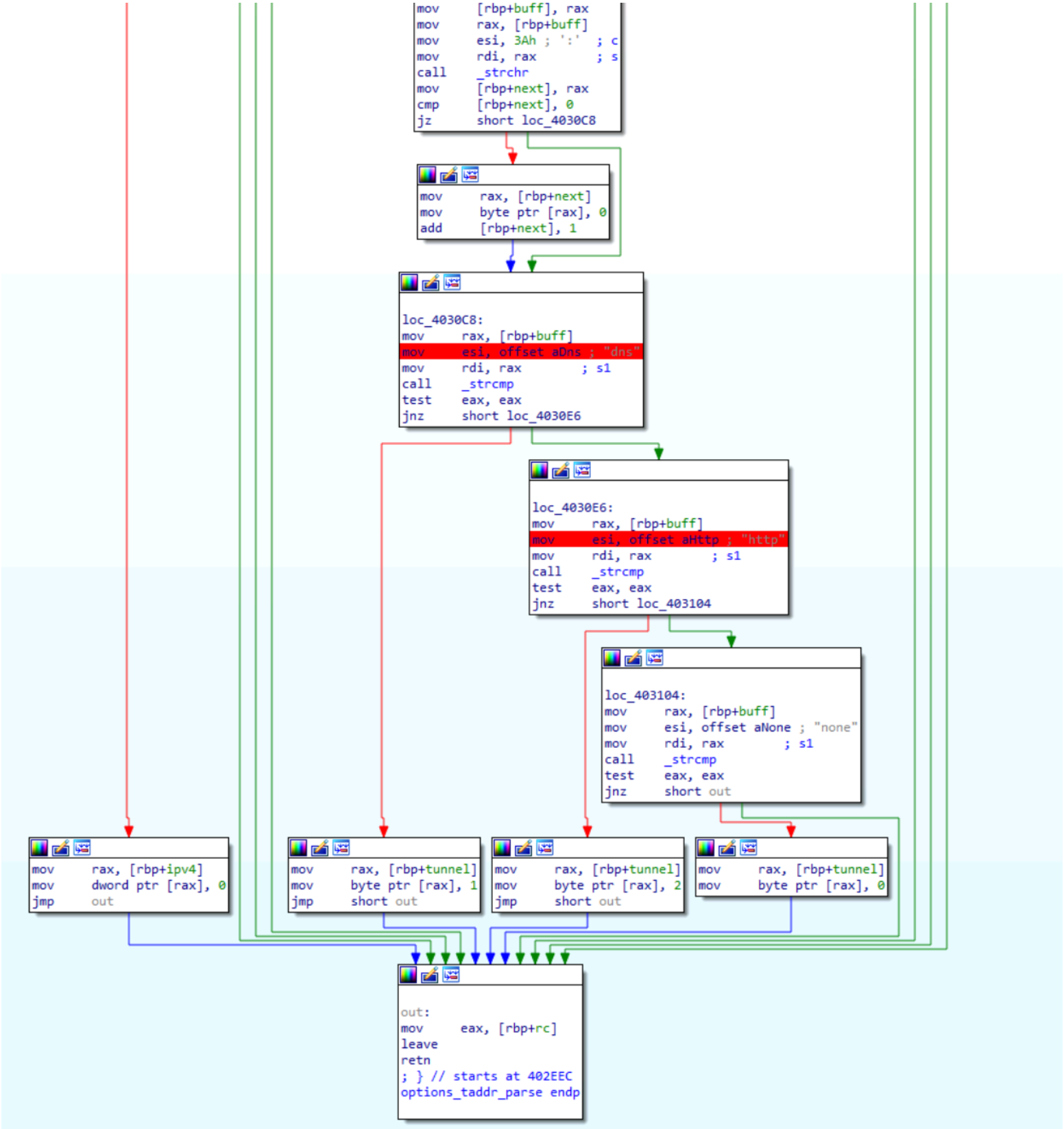


The option_taddr_parse function

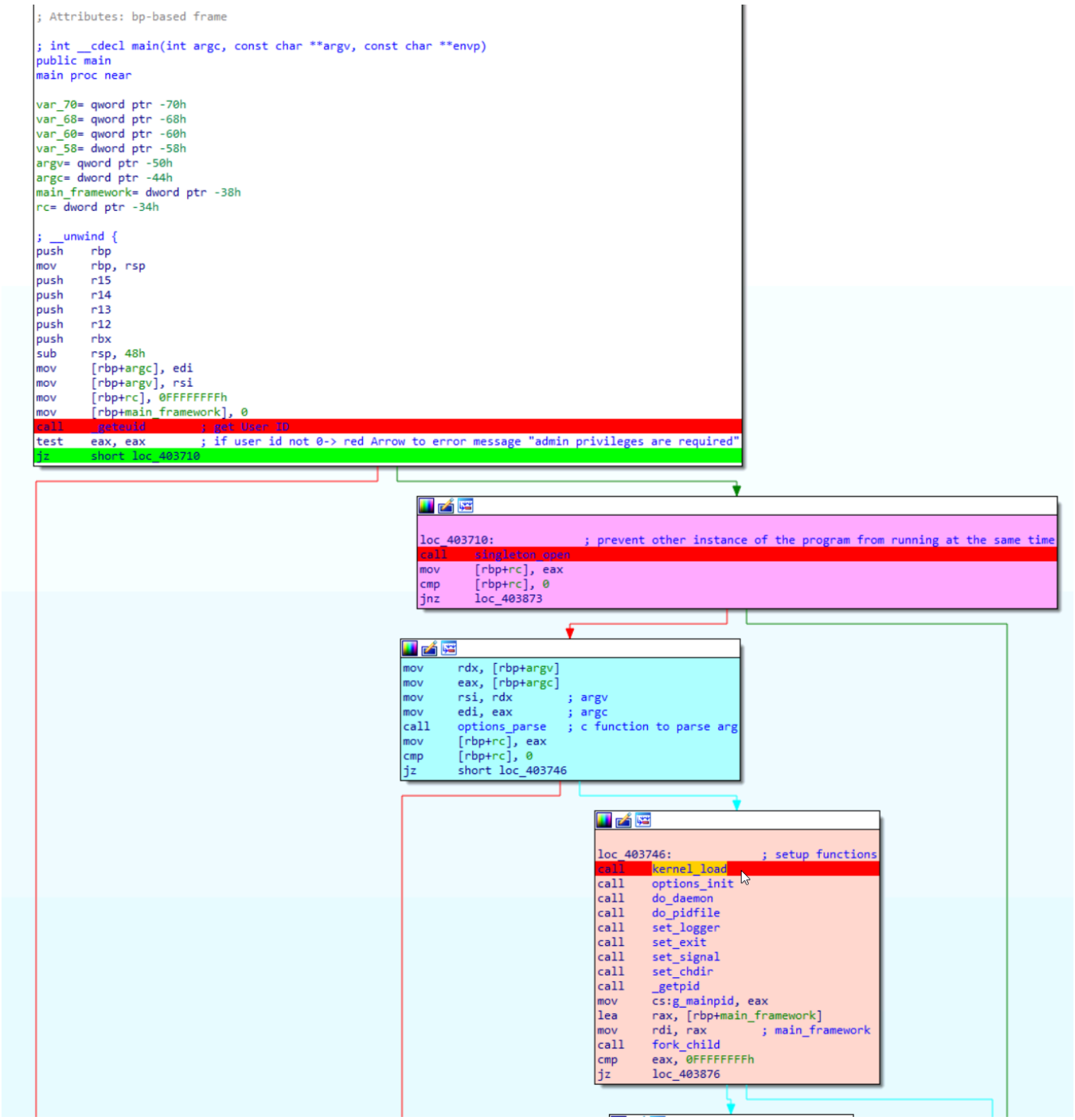


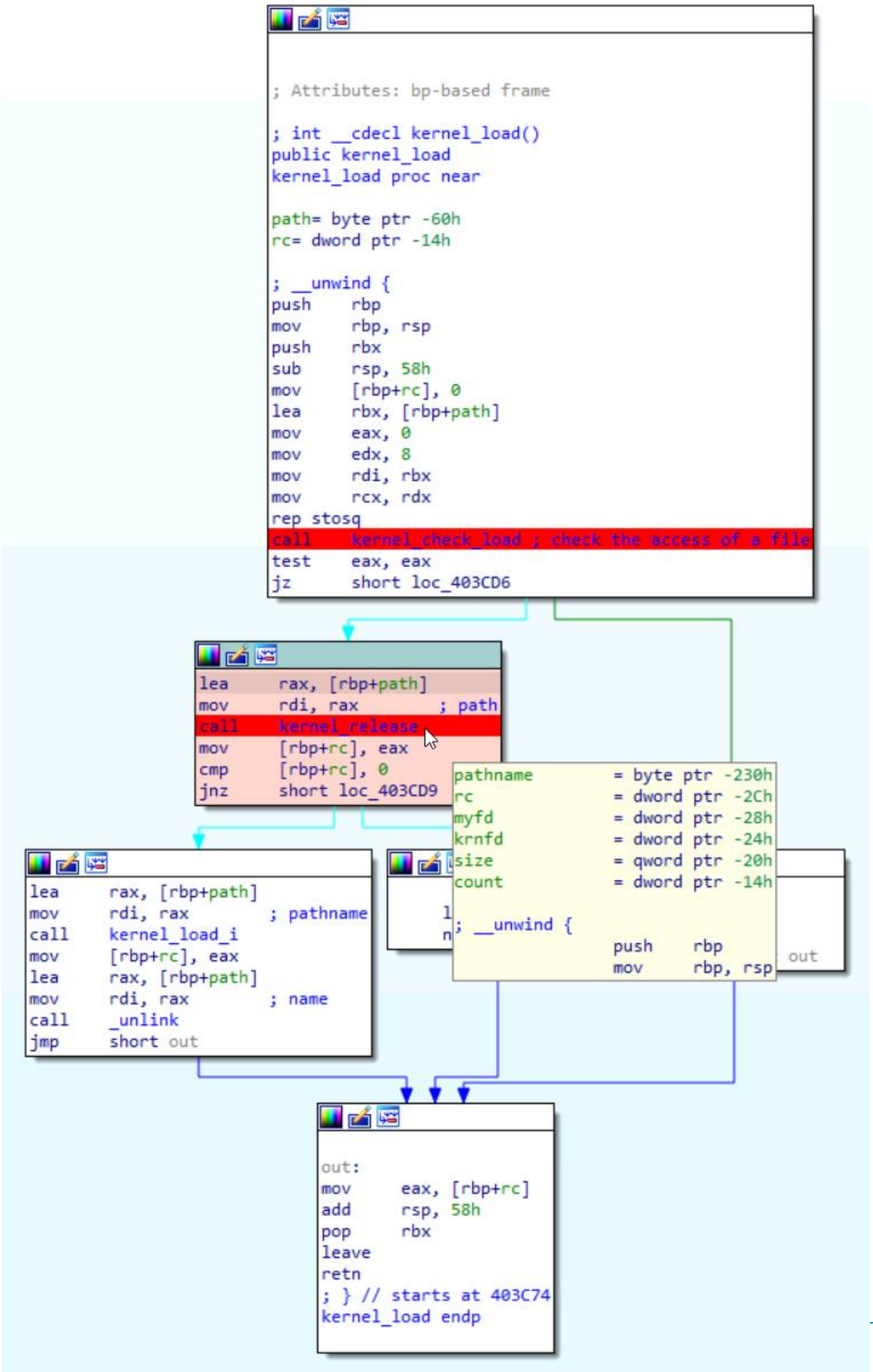

; __unwind {
push rbp ; This function seems to convert a domain name into a server domain and a port
mov rbp, rsp
sub rsp, 50h
mov [rbp+optarg], rdi
mov [rbp+ipv4], rsi
mov [rbp+port], rdx
mov [rbp+proto], rcx
mov [rbp+tunnel], r8
mov [rbp+rc], 0FFFFFFFFh
mov rax, [rbp+optarg]
mov rdi, rax ; buf
call rrootkit_util_strtrim
mov [rbp+buff], rax
mov rax, [rbp+buff]
mov rdi, rax ; s
call _strlen
add rax, [rbp+buff]
mov [rbp+end], rax
mov [rbp+next], 0
mov rax, [rbp+buff]
mov esi, 3Ah ; ':' ; c
mov rdi, rax ; s
call _strchr ; semicolon to split domain and port
mov [rbp+next], rax
cmp [rbp+next], 0
jz short loc_402F63
mov rax, [rbp+next]
mov byte ptr [rax], 0
add [rbp+next], 1
loc_402F63:
mov rax, [rbp+buff]
mov rdi, rax ; cp
call _inet_addr
mov rdx, [rbp+ipv4]
mov [rdx], eax
mov rax, [rbp+ipv4]
mov eax, [rax]
cmp eax, 0FFFFFFFFh
jnz short loc_402F8F
loc_402F8F:
mov [rbp+rc], 0
cmp [rbp+next], 0
jz out
mov rax, [rbp+next]
cmp rax, [rbp+end]
jnb out
mov rax, [rbp+next]
mov [rbp+buff], rax
mov rax, [rbp+buff]

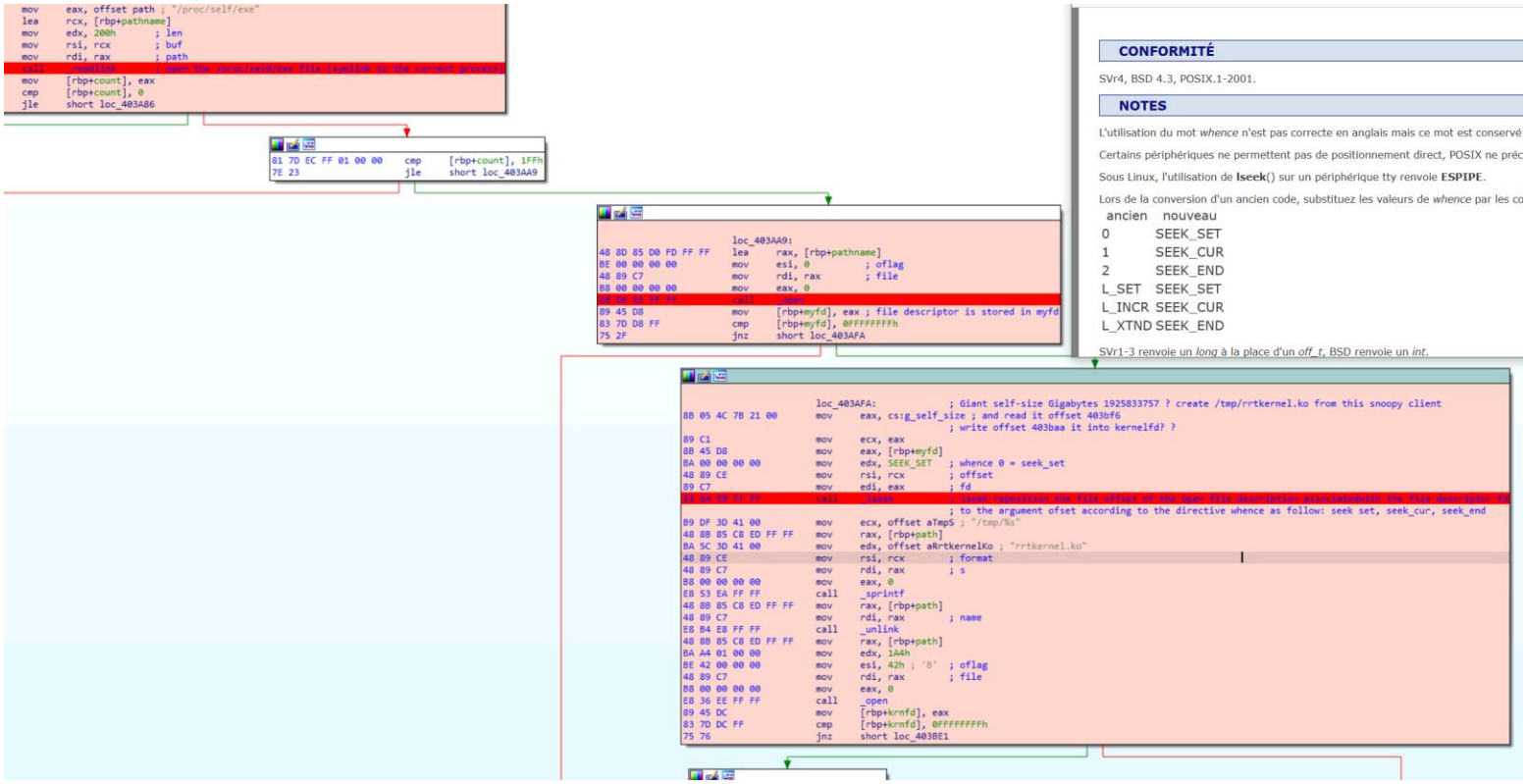

mov esi, 3Ah ; ':' ; c
mov rdi, rax ; s
call _strchr
mov [rbp+next], rax
cmp [rbp+next], 0
jz short loc_402FDF
mov rax, [rbp+next]
mov byte ptr [rax], 0
add [rbp+next], 1
loc_402FDF:
mov rax, [rbp+buff]
mov rdi, rax ; nptr
call _atoi ; ASCII to Integer
movzx eax, ax
mov edi, ea
call _htons
mov rdx, [r
mov [rdx],
cmp [rbp+ne
jz out
; ========== S U B R O U T I N E ==========
; Attributes: thunk
; int atoi(const char *nptr)
_atoi proc near ; CODE XREF
; options_p
jmp cs:off_61B460
_atoi endp
mov
cmp rax, [rbp+end]
jnb out
mov rax, [rbp+next]
mov [rbp+buff], rax
mov rax, [rbp+buff]
mov esi, 3Ah ; ':' ; c
mov rdi, rax ; s
call _strchr
mov [rbp+next], rax
cmp [rbp+next], 0
jz short loc_403045
mov rax, [rbp+next]
mov byte ptr [rax], 0
add [rbp+next], 1
loc_403045:
mov rax, [rbp+buff]
mov esi, offset s2 ; "tcp"
mov rdi, rax ; s1
call _strcmp
test eax, eax
jnz short loc_403063


mov [rbp+buff], rax
mov rax, [rbp+buff]
mov esi, 3Ah ; ':' ; c
mov rdi, rax ; s
call _strchr
mov [rbp+next], rax
cmp [rbp+next], 0
jz short loc_4030C8
mov rax, [rbp+next]
mov byte ptr [rax], 0
add [rbp+next], 1
loc_4030C8:
mov rax, [rbp+buff]
mov esi, offset aDns ; "dns"
mov rdi, rax ; s1
call _strcmp
test eax, eax
jnz short loc_4030E6
loc_4030E6:
mov rax, [rbp+buff]
mov esi, offset aHttp ; "http"
mov rdi, rax ; s1
call _strcmp
test eax, eax
jnz short loc_403104
loc_403104:
mov rax, [rbp+buff]
mov esi, offset aNone ; "none"
mov rdi, rax ; s1
call _strcmp
test eax, eax
jnz short out
mov rax, [rbp+ipv4]
mov dword ptr [rax], 0
jmp out
mov rax, [rbp+tunnel]
mov byte ptr [rax], 1
jmp short out
mov rax, [rbp+tunnel]
mov byte ptr [rax], 2
jmp short out
mov rax, [rbp+tunnel]
mov byte ptr [rax], 0
out:
mov eax, [rbp+rc]
leave
retn
; } // starts at 402EEC
options_taddr_parse endp

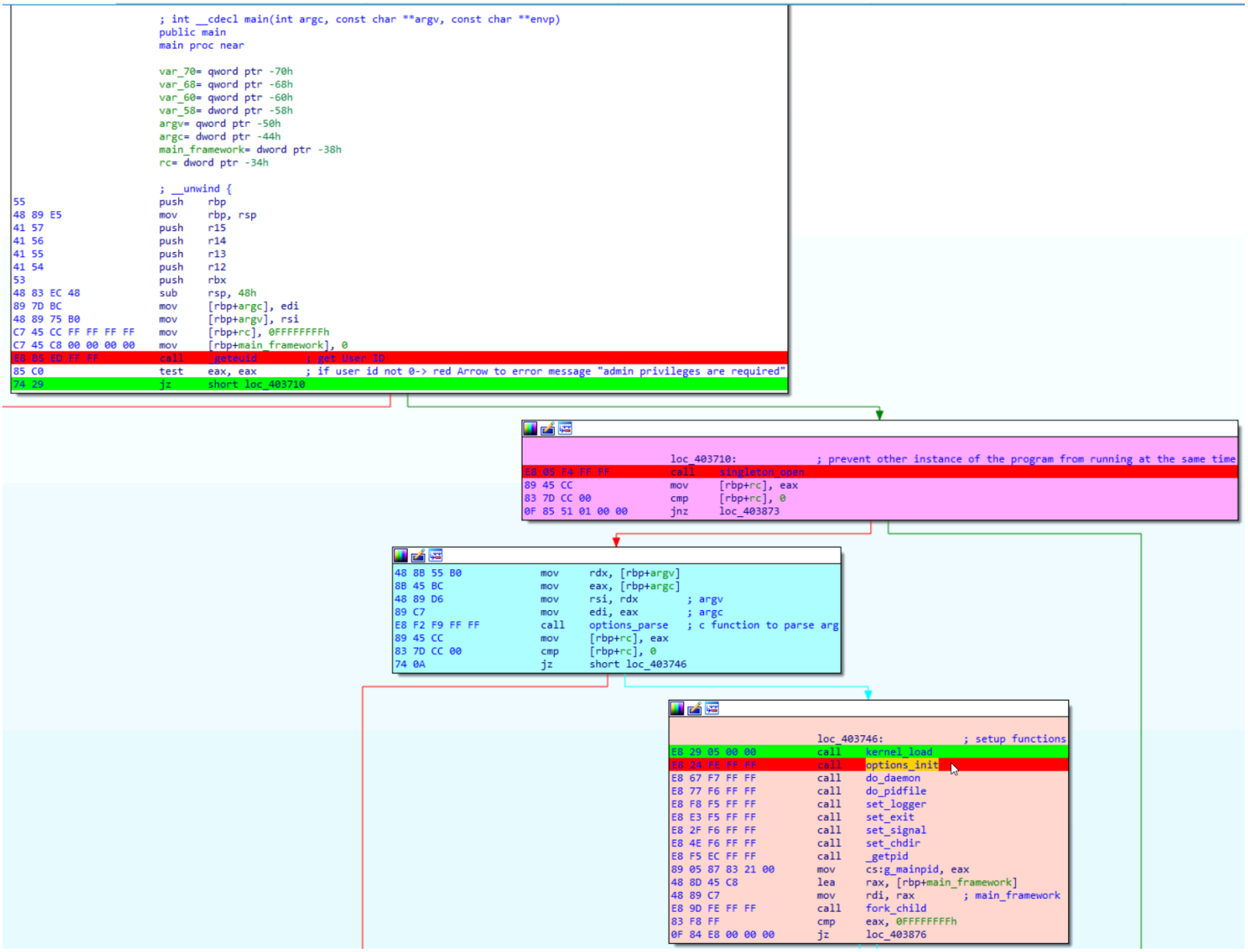
The Kernel_load function

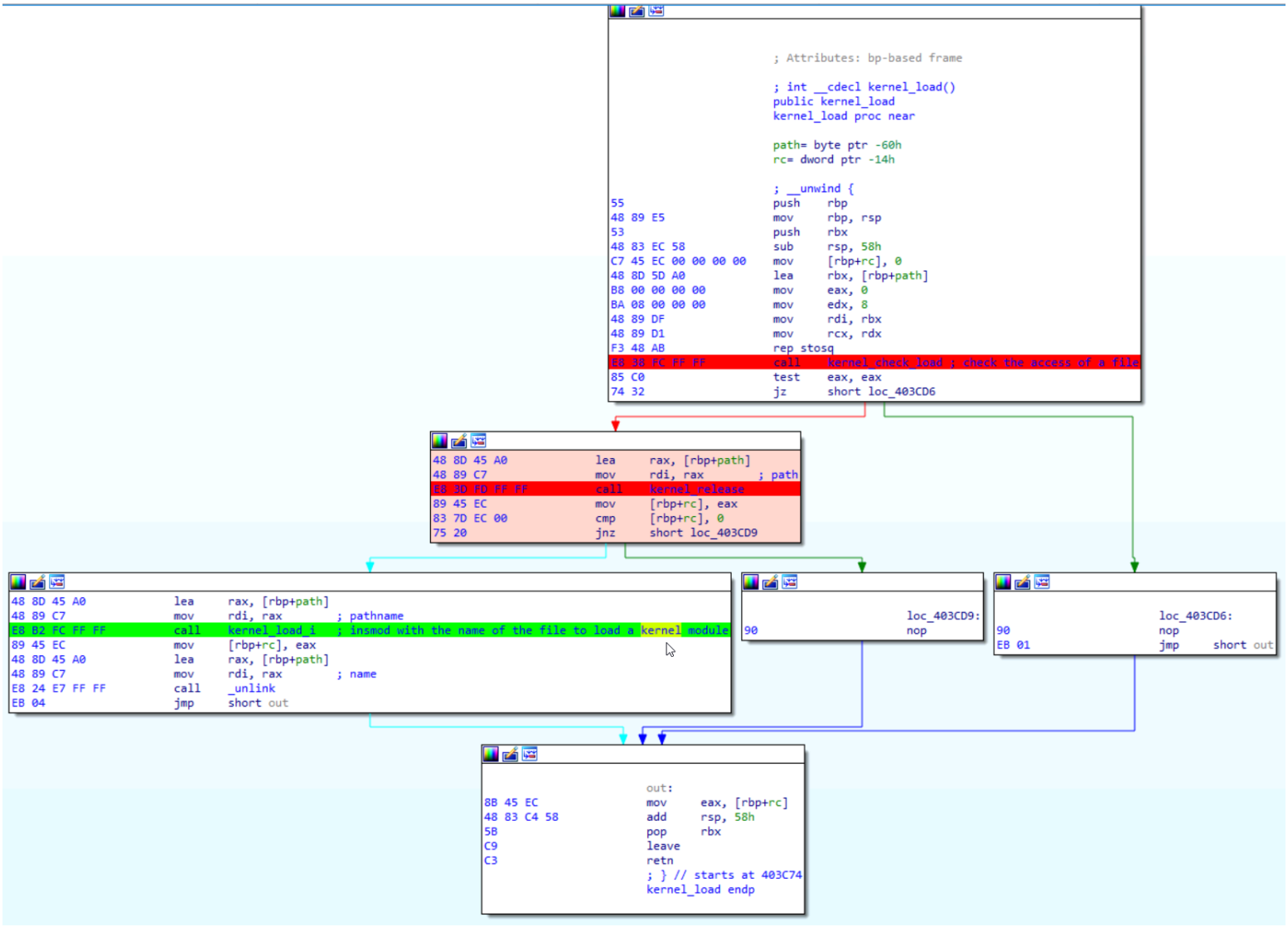


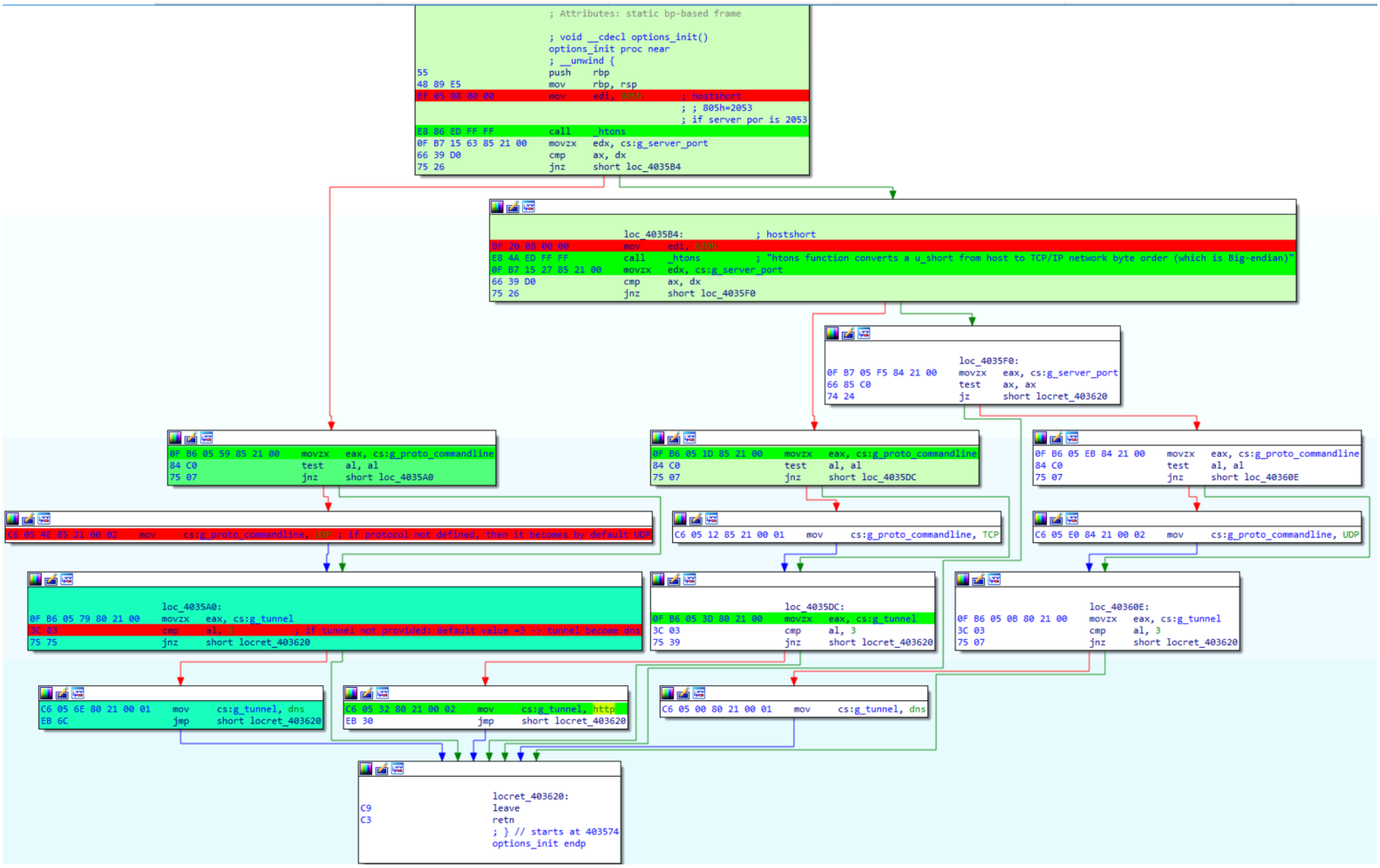

; Attributes: bp-based frame
; int __cdecl kernel_load()
public kernel_load
kernel_load proc near
path= byte ptr -60h
rc= dword ptr -14h
; __unwind {
push rbp
mov rbp, rsp
push rbx
sub rsp, 58h
mov [rbp+rc], 0
lea rbx, [rbp+path]
mov eax, 0
mov edx, 8
mov rdi, rbx
mov rcx, rdx
rep stosq
call kernel_check_load ; check the access of a file
test eax, eax
jz short loc_403CD6
lea rax, [rbp+path]
mov rdi, rax ; path
call kernel_release
mov [rbp+rc], eax
cmp [rbp+rc], 0
jnz short loc_403CD9
pathname = byte ptr -230h
rc = dword ptr -2Ch
myfd = dword ptr -28h
krnfd = dword ptr -24h
size = qword ptr -20h
count = dword ptr -14h
; __unwind {
push rbp
mov rbp, rsp
lea rax, [rbp+path]
mov rdi, rax ; pathname
call kernel_load_i
mov [rbp+rc], eax
lea rax, [rbp+path]
mov rdi, rax ; name
call _unlink
jmp short out
out:
mov eax, [rbp+rc]
add rsp, 58h
pop rbx
leave
retn
; } // starts at 403C74
kernel_load endp

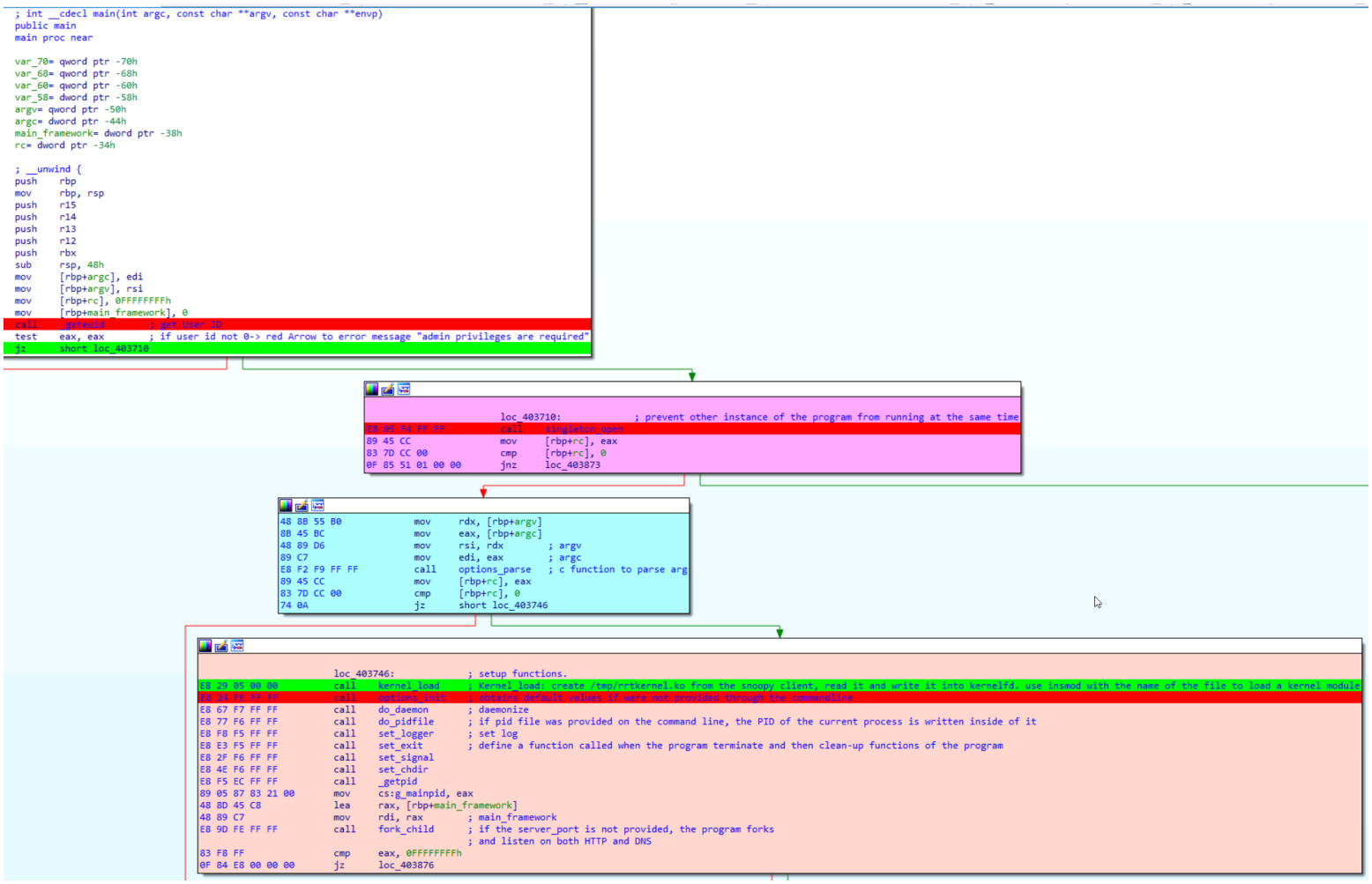

mov eax, offset path ; "/proc/self/exe"
lea rcx, [rbp+pathname]
mov edx, 200h ; len
mov rsi, rcx ; buf
mov rdi, rax ; path
call _readlink ; Open the /proc/self/exe file (symlink to the current process)
mov [rbp+count], eax
cmp [rbp+count], 0
jle short loc_403A86
81 7D EC FF 01 00 00 cmp [rbp+count], 1FFh
7E 23 jle short loc_403AA9
loc_403AA9:
48 8D 85 D0 FD FF FF lea rax, [rbp+pathname]
BE 00 00 00 00 mov esi, 0 ; oflag
48 89 C7 mov rdi, rax ; file
B8 00 00 00 00 mov eax, 0
E8 D6 E1 FF FF call _open
89 45 D8 mov [rbp+myfd], eax ; file descriptor is stored in myfd
83 7D D8 FF cmp [rbp+myfd], 0FFFFFFFFh
75 2F jnz short loc_403AFA
loc_403AFA: ; Giant self-size Gigabytes 1925833757 ? create /tmp/rrtkernel.ko from this snoopy client
8B 05 4C 7B 21 00 mov eax, cs:g_self_size ; and read it offset 403bf6
; write offset 403baa it into kernelfd? ?
89 C1 mov ecx, eax
8B 45 D8 mov eax, [rbp+myfd]
BA 00 00 00 00 mov edx, SEEK_SET ; whence 0 = seek_set
48 89 CE mov rsi, rcx ; offset
89 C7 mov edi, eax ; fd
E8 64 E8 FF FF call _lseek ; lseek repositionne the file offset of the open file description associatedwith the file descriptor fd
; to the argument ofset according to the directive whence as follow: seek set, seek_cur, seek_end
89 DF 3D 41 00 mov ecx, offset aTmpS ; "/tmp/%s"
48 8B 85 C8 ED FF FF mov rax, [rbp+path]
BA 5C 3D 41 00 mov edx, offset aRrtkernelKo ; "rrtkernel.ko"
48 89 CE mov rsi, rcx ; format
48 89 C7 mov rdi, rax ; s
B8 00 00 00 00 mov eax, 0
E8 53 EA FF FF call _sprintf
48 8B 85 C8 ED FF FF mov rax, [rbp+path]
48 89 C7 mov rdi, rax ; name
E8 B4 E8 FF FF call _unlink
48 8B 85 C8 ED FF FF mov rax, [rbp+path]
BA A4 01 00 00 mov edx, 1A4h
BE 42 00 00 00 mov esi, 42h ; 'B' ; oflag
48 89 C7 mov rdi, rax ; file
B8 00 00 00 00 mov eax, 0
E8 36 EE FF FF call _open
89 45 DC mov [rbp+krnfd], eax
83 7D DC FF cmp [rbp+krnfd], 0FFFFFFFFh
75 76 jnz short loc_403BE1
CONFORMITÉ
SVr4, BSD 4.3, POSIX.1-2001.
NOTES
L'utilisation du mot whence n'est pas correcte en anglais mais ce mot est conservé
Certains périphériques ne permettent pas de positionnement direct, POSIX ne préc
Sous Linux, l'utilisation de lseek() sur un périphérique tty renvoie ESPIPE.
Lors de la conversion d'un ancien code, substituez les valeurs de whence par les co
ancien nouveau
0 SEEK_SET
1 SEEK_CUR
2 SEEK_END
L_SET SEEK_SET
L_INCR SEEK_CUR
L_XTND SEEK_END
SVr1-3 renvoie un long à la place d'un off_t, BSD renvoie un int.

The option_init function



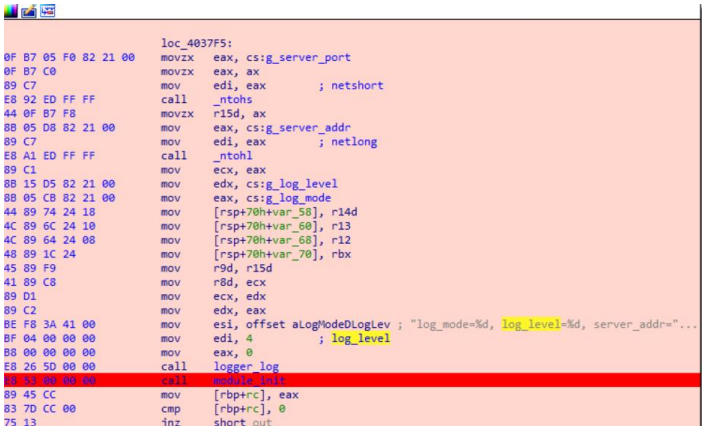

; Attributes: bp-based frame
; int __cdecl kernel_load()
public kernel_load
kernel_load proc near
path= byte ptr -60h
rc= dword ptr -14h
; __unwind {
55 push rbp
48 89 E5 mov rbp, rsp
53 push rbx
48 83 EC 58 sub rsp, 58h
C7 45 EC 00 00 00 00 mov [rbp+rc], 0
48 8D 5D A0 lea rbx, [rbp+path]
B8 00 00 00 00 mov eax, 0
BA 08 00 00 00 mov edx, 8
48 89 DF mov rdi, rbx
48 89 D1 mov rcx, rdx
F3 48 AB rep stosq
E8 38 FC FF FF call kernel_check_load ; check the access of a file
85 C0 test eax, eax
74 32 jz short loc_403CD6
48 8D 45 A0 lea rax, [rbp+path]
48 89 C7 mov rdi, rax ; path
E8 3D FD FF FF call kernel_release
89 45 EC mov [rbp+rc], eax
83 7D EC 00 cmp [rbp+rc], 0
75 20 jnz short loc_403CD9
48 8D 45 A0 lea rax, [rbp+path]
48 89 C7 mov rdi, rax ; pathname
E8 B2 FC FF FF call kernel_load_i ; insmod with the name of the file to load a kernel module
89 45 EC mov [rbp+rc], eax
48 8D 45 A0 lea rax, [rbp+path]
48 89 C7 mov rdi, rax ; name
E8 24 E7 FF FF call _unlink
EB 04 jmp short out
loc_403CD9:
90 nop
loc_403CD6:
90 nop
EB 01 jmp short out
out:
8B 45 EC mov eax, [rbp+rc]
48 83 C4 58 add rsp, 58h
5B pop rbx
C9 leave
C3 retn
; } // starts at 403C74
kernel_load endp


; Attributes: static bp-based frame
; void __cdecl options_init()
options_init proc near
; __unwind {
55 push rbp
48 89 E5 mov rbp, rsp
BF 05 08 00 00 mov edi, 805h ; hostshort
; ; 805h=2053
; if server por is 2053
E8 86 ED FF FF call _htons
0F B7 15 63 85 21 00 movzx edx, cs:g_server_port
66 39 D0 cmp ax, dx
75 26 jnz short loc_4035B4
loc_4035B4: ; hostshort
BF 20 08 00 00 mov edi, 2053
E8 4A ED FF FF call _htons ; "htons function converts a u_short from host to TCP/IP network byte order (which is Big-endian)"
0F B7 15 27 85 21 00 movzx edx, cs:g_server_port
66 39 D0 cmp ax, dx
75 26 jnz short loc_4035F0
loc_4035F0:
0F B7 05 F5 84 21 00 movzx eax, cs:g_server_port
66 85 C0 test ax, ax
74 24 jz short locret_403620
0F B6 05 59 85 21 00 movzx eax, cs:g_proto_commandline
84 C0 test al, al
75 07 jnz short loc_4035A0
0F B6 05 1D 85 21 00 movzx eax, cs:g_proto_commandline
84 C0 test al, al
75 07 jnz short loc_4035DC
0F B6 05 EB 84 21 00 movzx eax, cs:g_proto_commandline
84 C0 test al, al
75 07 jnz short loc_40360E
C6 05 4F 85 21 00 02 mov cs:g_proto_commandline, UDP ; if protocol not defined, then it becomes by default UDP
C6 05 12 85 21 00 01 mov cs:g_proto_commandline, TCP
C6 05 E0 84 21 00 02 mov cs:g_proto_commandline, UDP
loc_4035A0:
0F B6 05 79 80 21 00 movzx eax, cs:g_tunnel
3C 03 cmp al, 3 ; if tunnel not provided: default value =3 => tunnel become dns
75 75 jnz short locret_403620
loc_4035DC:
0F B6 05 3D 80 21 00 movzx eax, cs:g_tunnel
3C 03 cmp al, 3
75 39 jnz short locret_403620
loc_40360E:
0F B6 05 0B 80 21 00 movzx eax, cs:g_tunnel
3C 03 cmp al, 3
75 07 jnz short locret_403620
C6 05 6E 80 21 00 01 mov cs:g_tunnel, dns
EB 6C jmp short locret_403620
C6 05 52 80 21 00 02 mov cs:g_tunnel, http
EB 30 jmp short locret_403620
C6 05 00 80 21 00 01 mov cs:g_tunnel, dns
locret_403620:
C9 leave
C3 retn
; } // starts at 403574
options_init endp



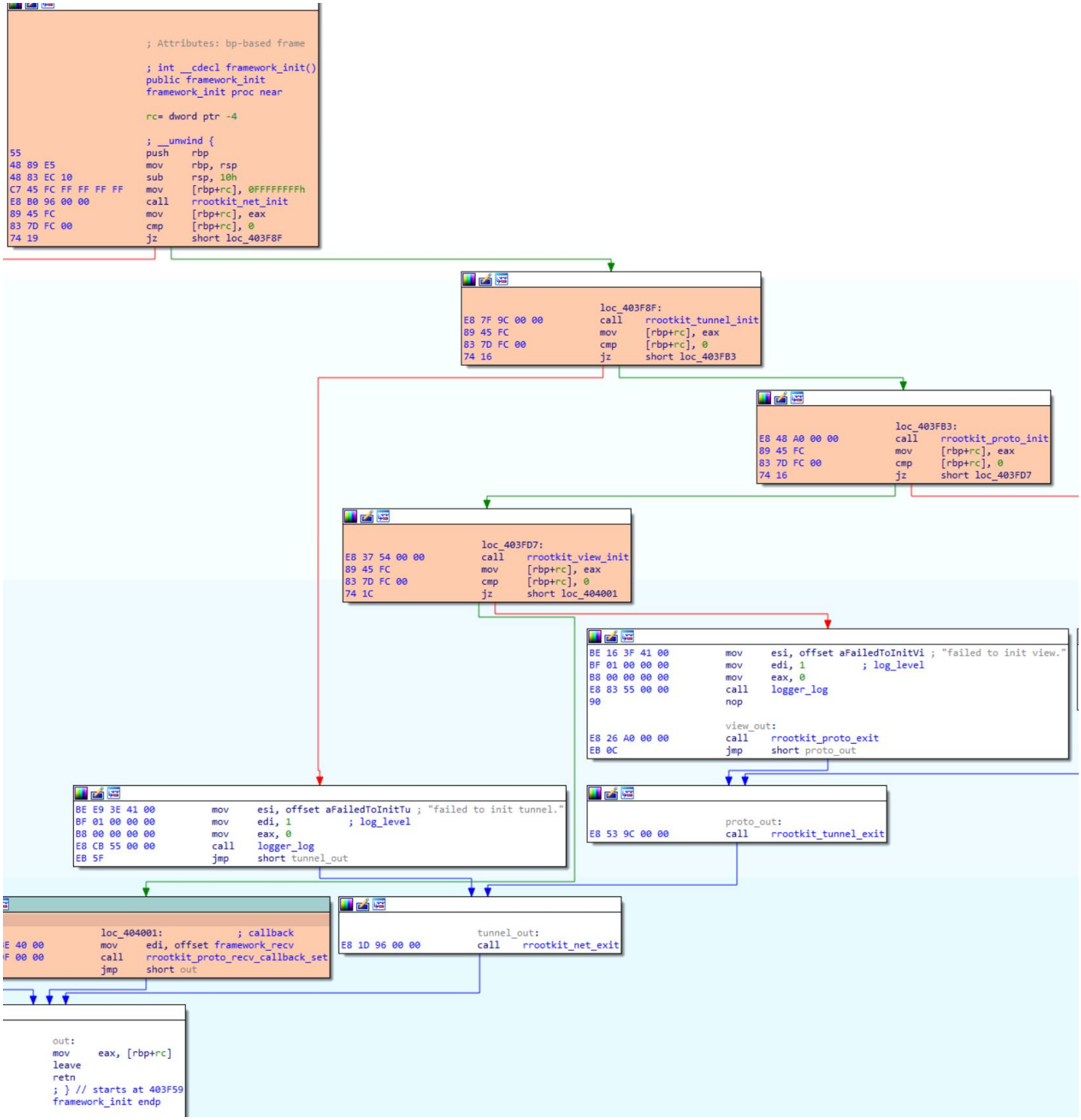
Then the program continues to log functions.

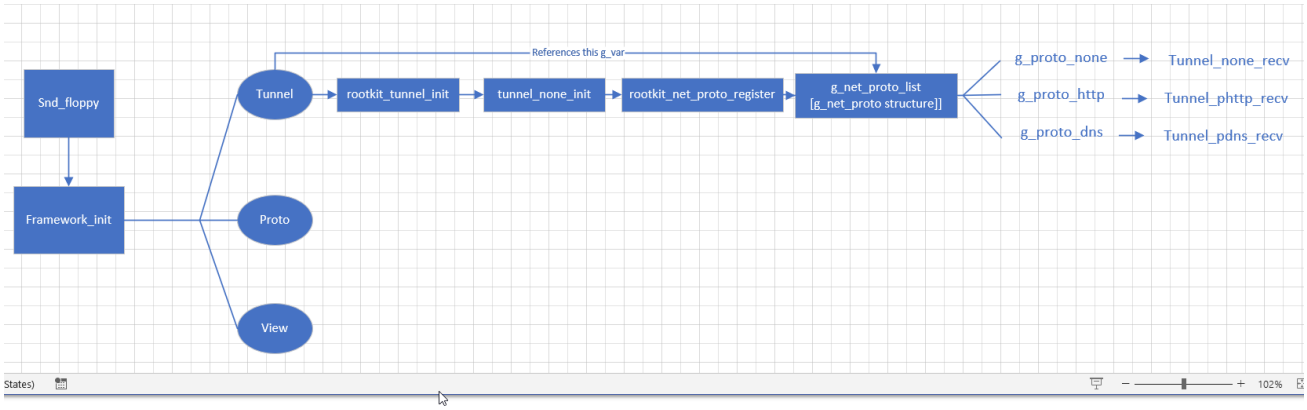
Next, we reach the module_init function

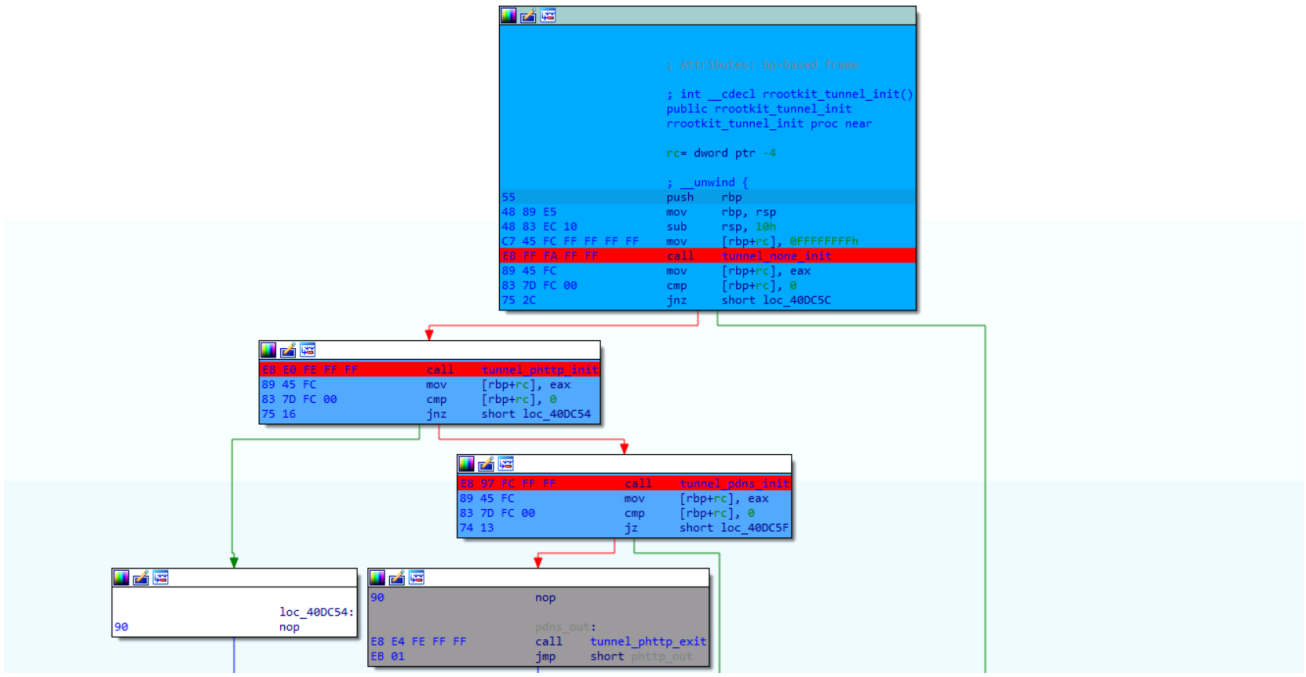


The module_init function

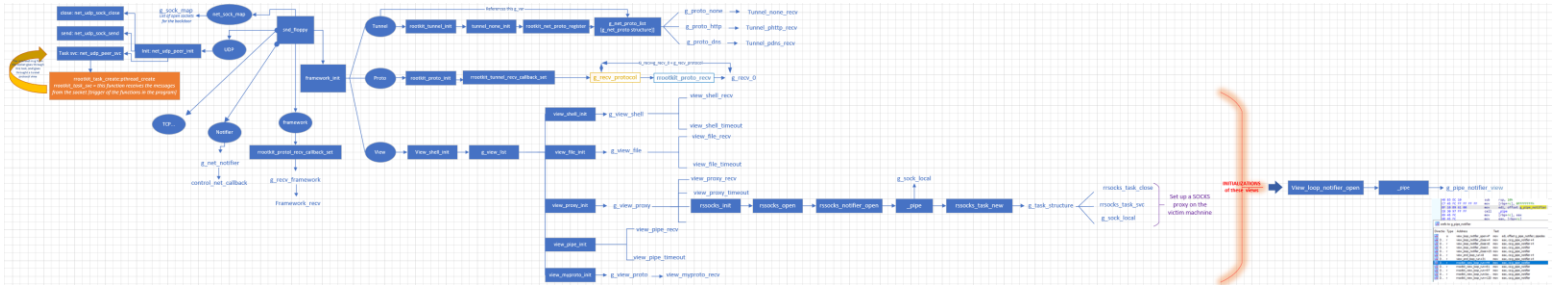
/module_init/control_init/framework_init: the setting up of the program.





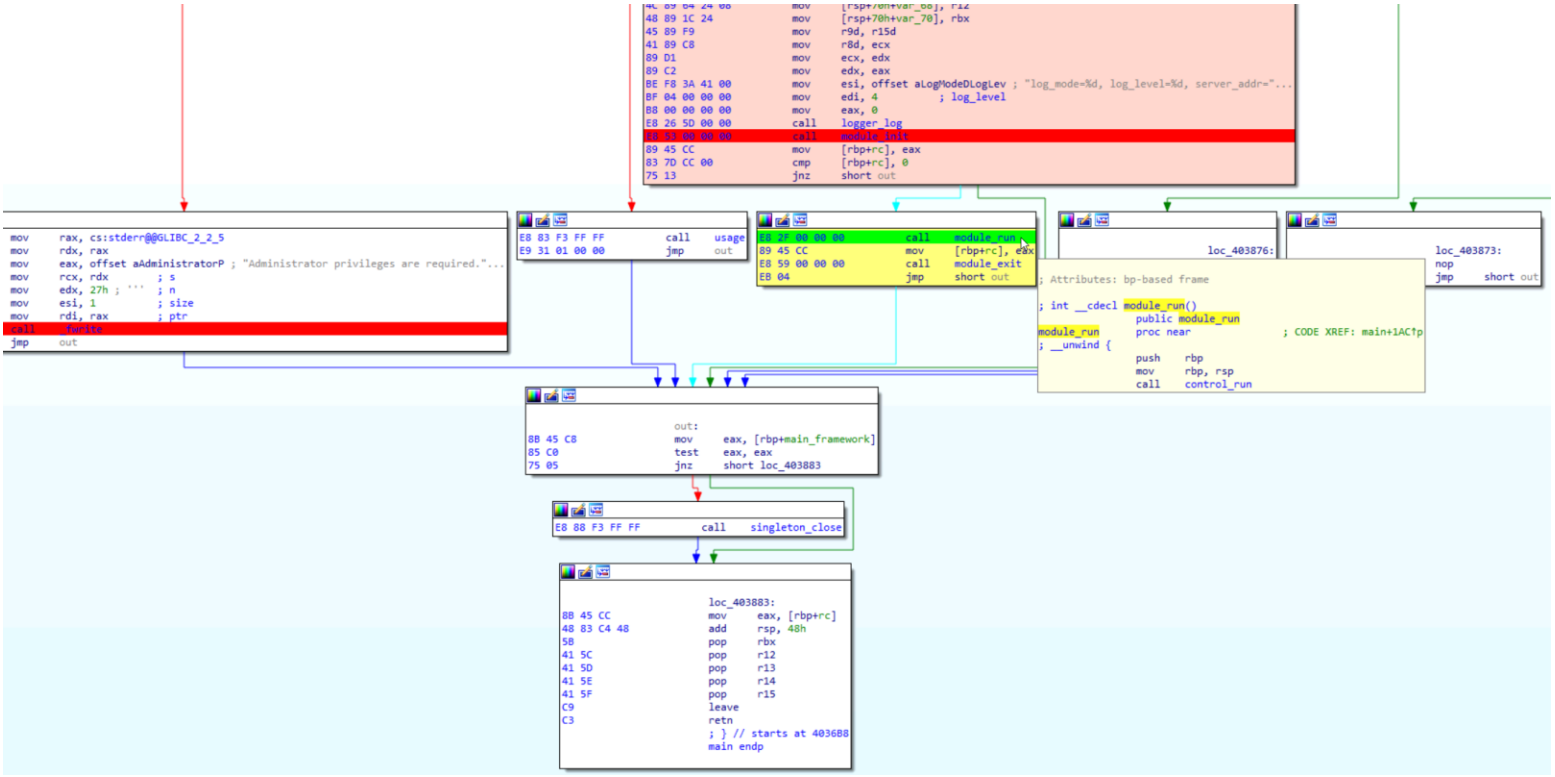


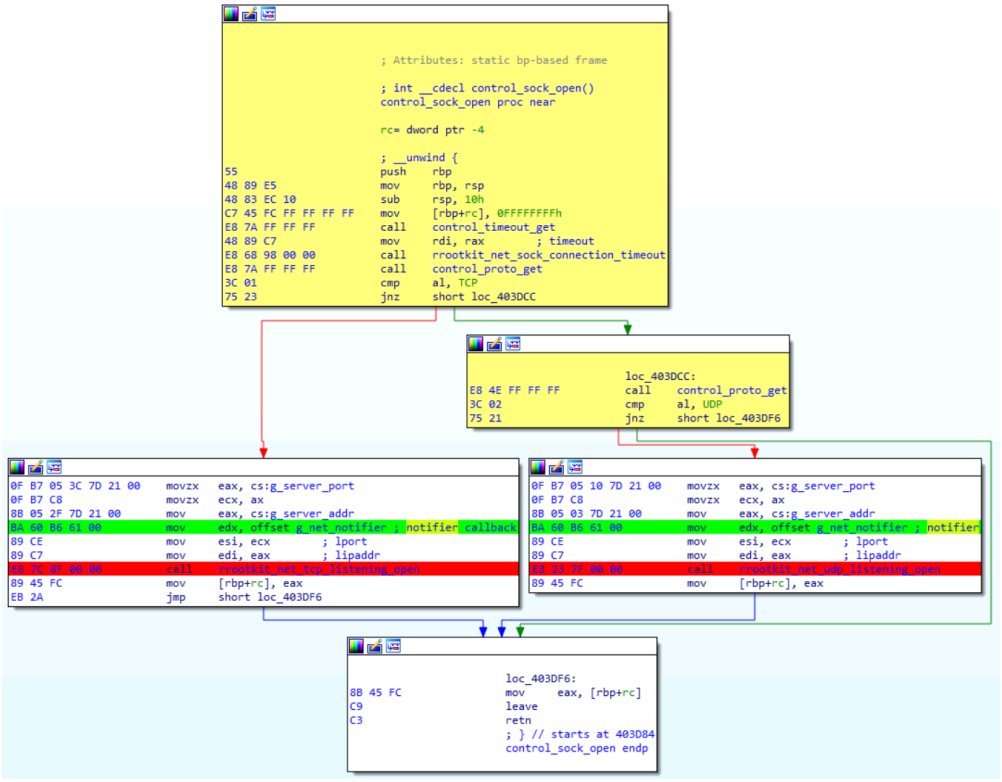
Architecture:

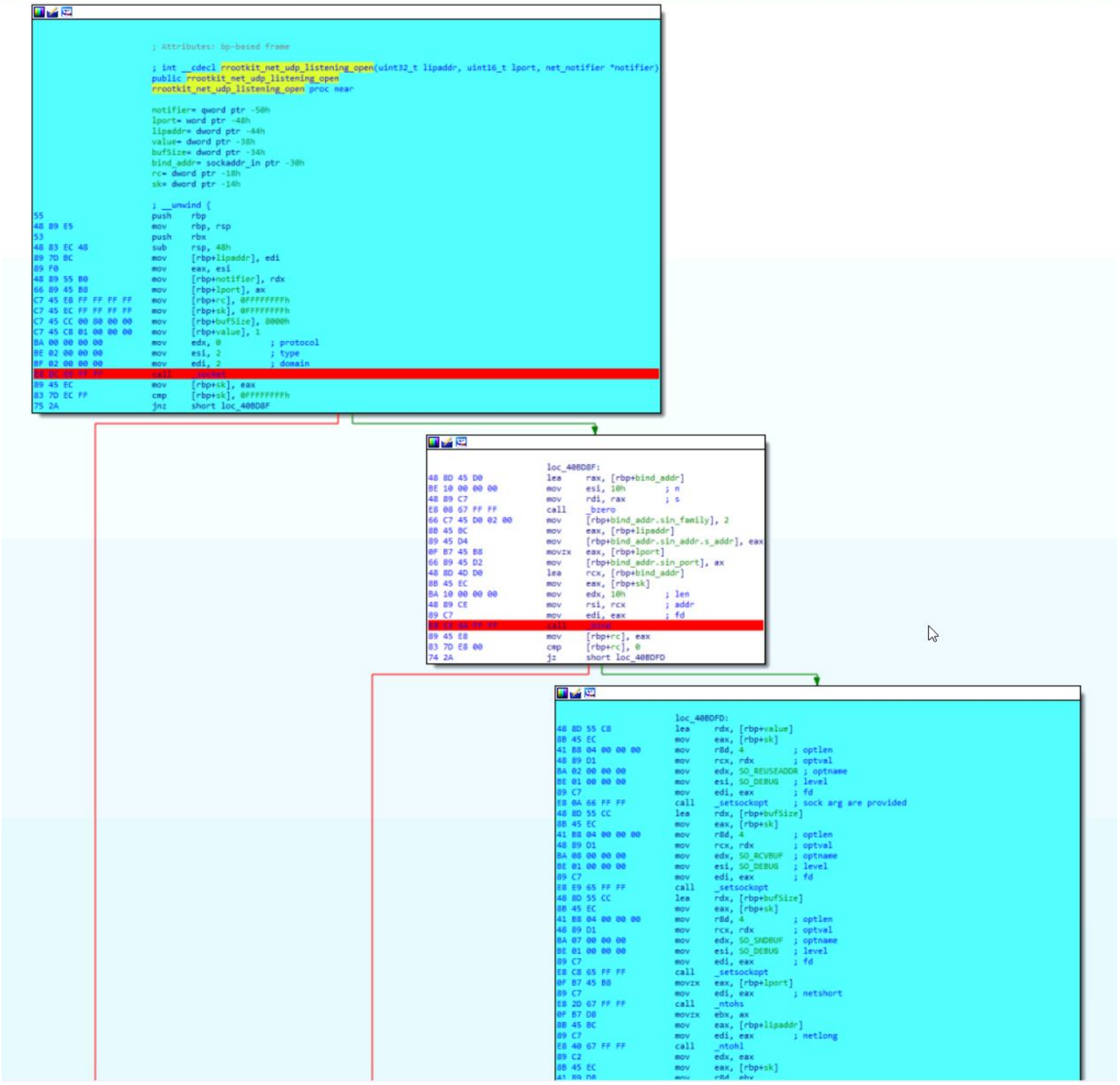


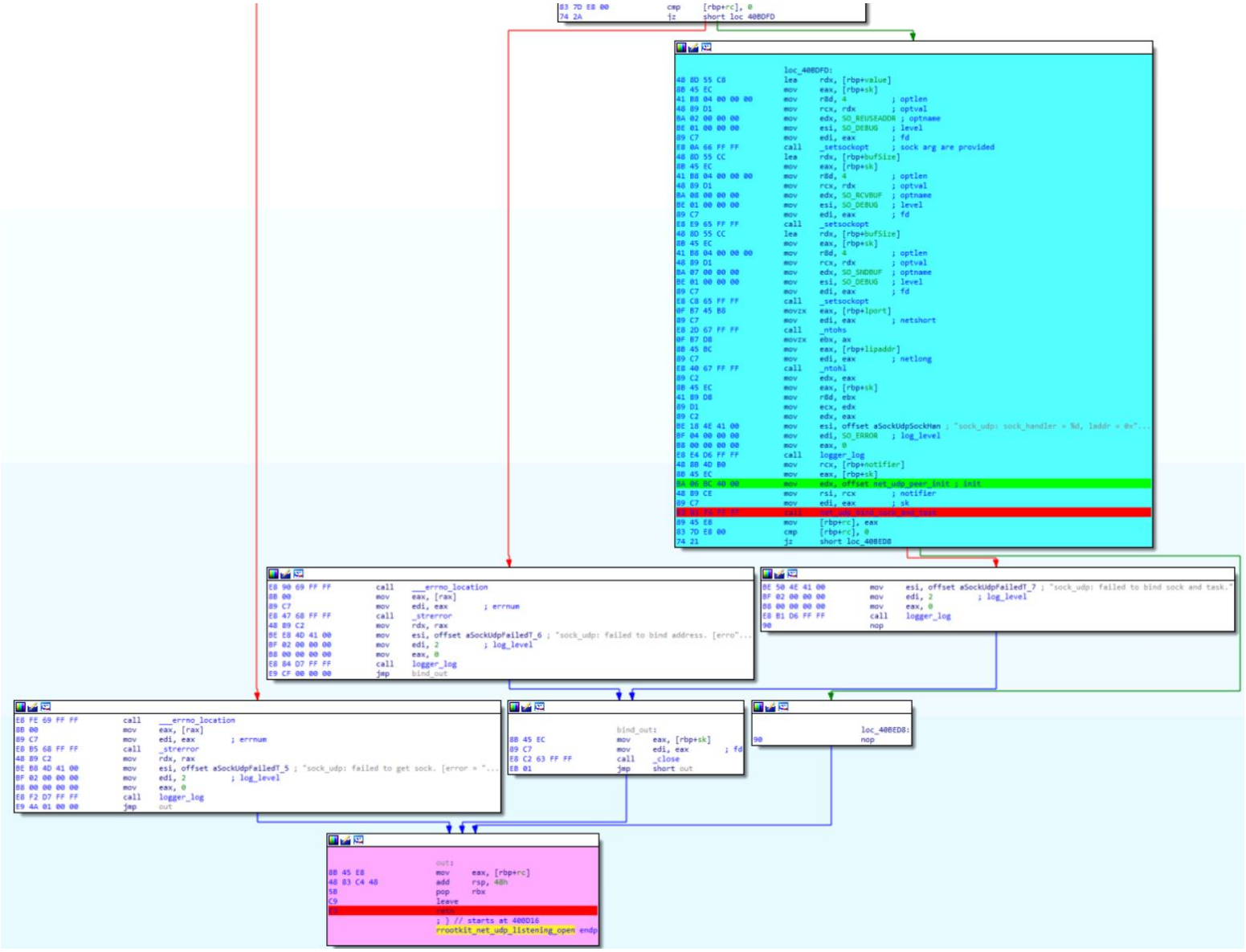
This "callbacks architecture file" is attached with this report

After the module_init function, there is a control socket open function and then the program goes to listen on UDP or TCP.


mov rax, cs:stderr@@GLIBC_2_2_5
mov rdx, rax
mov eax, offset aAdministratorP ; "Administrator privileges are required."...
mov rcx, rdx ; s
mov edx, 27h ; ''' ; n
mov esi, 1 ; size
mov rdi, rax ; ptr
call _fwrite
jmp out
call usage
jmp out
call module_run
mov [rbp+rc], eax
call module_exit
jmp short out
loc_403876:
loc_403873:
nop
jmp short out
; Attributes: bp-based frame
; int __cdecl module_run()
public module_run
module_run proc near ; CODE XREF: main+1AC↑p
; __unwind {
push rbp
mov rbp, rsp
call control_run
out:
mov eax, [rbp+main_framework]
test eax, eax
jnz short loc_403883
call singleton_close
loc_403883:
mov eax, [rbp+rc]
add rsp, 48h
pop rbx
pop r12
pop r13
pop r14
pop r15
leave
retn
; } // starts at 4036B8
main endp


; Attributes: static bp-based frame
; int __cdecl control_sock_open()
control_sock_open proc near
rc= dword ptr -4
; __unwind {
push rbp
mov rbp, rsp
sub rsp, 10h
mov [rbp+rc], 0FFFFFFFFh
call control_timeout_get
mov rdi, rax ; timeout
call rrootkit_net_sock_connection_timeout
call control_proto_get
cmp al, TCP
jnz short loc_403DCC
loc_403DCC:
call control_proto_get
cmp al, UDP
jnz short loc_403DF6
movzx eax, cs:g_server_port
movzx ecx, ax
mov eax, cs:g_server_addr
mov edx, offset g_net_notifier ; notifier callback
mov esi, ecx ; lport
mov edi, eax ; lipaddr
call rrootkit_net_tcp_listening_open
mov [rbp+rc], eax
jmp short loc_403DF6
movzx eax, cs:g_server_port
movzx ecx, ax
mov eax, cs:g_server_addr
mov edx, offset g_net_notifier ; notifier
mov esi, ecx ; lport
mov edi, eax ; lipaddr
call rrootkit_net_udp_listening_open
mov [rbp+rc], eax
loc_403DF6:
mov eax, [rbp+rc]
leave
retn
; } // starts at 403D84
control_sock_open endp

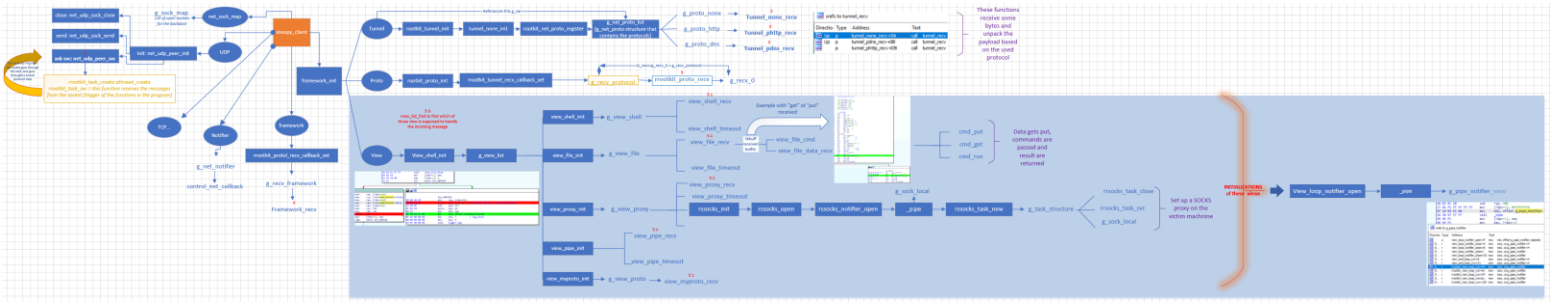

; Attributes: bp-based frame
; int __cdecl rrootkit_net_udp_listening_open(uint32_t lipaddr, uint16_t lport, net_notifier *notifier)
public rrootkit_net_udp_listening_open
rrootkit_net_udp_listening_open proc near
notifier= qword ptr -50h
lport= word ptr -48h
lipaddr= dword ptr -44h
value= dword ptr -38h
bufSize= dword ptr -34h
bind_addr= sockaddr_in ptr -30h
rc= dword ptr -18h
sk= dword ptr -14h
; __unwind {
55 push rbp
48 89 E5 mov rbp, rsp
53 push rbx
48 83 EC 48 sub rsp, 48h
89 7D BC mov [rbp+lipaddr], edi
89 F0 mov eax, esi
48 89 55 B0 mov [rbp+notifier], rdx
66 89 45 B8 mov [rbp+lport], ax
C7 45 EB FF FF FF FF mov [rbp+rc], 0FFFFFFFFh
C7 45 EC FF FF FF FF mov [rbp+sk], 0FFFFFFFFh
C7 45 CC 00 80 00 00 mov [rbp+bufSize], 8000h
C7 45 C8 01 00 00 00 mov [rbp+value], 1
BA 00 00 00 00 mov edx, 0 ; protocol
BE 02 00 00 00 mov esi, 2 ; type
BF 02 00 00 00 mov edi, 2 ; domain
E8 B6 F8 FF FF call _socket
89 45 EC mov [rbp+sk], eax
83 7D EC FF cmp [rbp+sk], 0FFFFFFFFh
75 2A jnz short loc_408D8F
loc_408D8F:
48 8D 45 D0 lea rax, [rbp+bind_addr]
BE 10 00 00 00 mov esi, 10h ; n
48 89 C7 mov rdi, rax ; s
E8 08 67 FF FF call _bzero
66 C7 45 D0 02 00 mov [rbp+bind_addr.sin_family], 2
8B 45 BC mov eax, [rbp+lipaddr]
89 45 D4 mov [rbp+bind_addr.sin_addr.s_addr], eax
0F B7 45 B8 movzx eax, [rbp+lport]
66 89 45 D2 mov [rbp+bind_addr.sin_port], ax
48 8D 4D D0 lea rcx, [rbp+bind_addr]
8B 45 EC mov eax, [rbp+sk]
BA 10 00 00 00 mov edx, 10h ; len
48 89 CE mov rsi, rcx ; addr
89 C7 mov edi, eax ; fd
E8 E8 8A FF FF call _bind
89 45 E8 mov [rbp+rc], eax
83 7D E8 00 cmp [rbp+rc], 0
74 2A jz short loc_408DFD
loc_408DFD:
48 8D 55 C8 lea rdx, [rbp+value]
8B 45 EC mov eax, [rbp+sk]
41 B8 04 00 00 00 mov r8d, 4 ; optlen
48 89 D1 mov rcx, rdx ; optval
BA 02 00 00 00 mov edx, SO_REUSEADDR ; optname
BE 01 00 00 00 mov esi, SO_DEBUG ; level
89 C7 mov edi, eax ; fd
E8 0A 66 FF FF call _setsockopt ; sock arg are provided
48 8D 55 CC lea rdx, [rbp+bufSize]
8B 45 EC mov eax, [rbp+sk]
41 B8 04 00 00 00 mov r8d, 4 ; optlen
48 89 D1 mov rcx, rdx ; optval
BA 08 00 00 00 mov edx, SO_RCVBUF ; optname
BE 01 00 00 00 mov esi, SO_DEBUG ; level
89 C7 mov edi, eax ; fd
E8 E9 65 FF FF call _setsockopt
48 8D 55 CC lea rdx, [rbp+bufSize]
8B 45 EC mov eax, [rbp+sk]
41 B8 04 00 00 00 mov r8d, 4 ; optlen
48 89 D1 mov rcx, rdx ; optval
BA 07 00 00 00 mov edx, SO_SNDBUF ; optname
BE 01 00 00 00 mov esi, SO_DEBUG ; level
89 C7 mov edi, eax ; fd
E8 C8 65 FF FF call _setsockopt
0F B7 45 B8 movzx eax, [rbp+lport]
89 C7 mov edi, eax ; netshort
E8 2D 67 FF FF call _ntohs
0F B7 D8 movzx ebx, ax
8B 45 BC mov eax, [rbp+lipaddr]
89 C7 mov edi, eax ; netlong
E8 40 67 FF FF call _ntohl
89 C2 mov edx, eax
8B 45 EC mov eax, [rbp+sk]



Creation of the socket->bind call->the socket is passed to the connect API function

Behavior

The program is difficult to follow because of its architecture (event-based logic), but the scheme is attached to this report



To summarize, **the initial setup prepares all the components** of the program **and** sets up **the callbacks**, then, when the program is running, **the program waits for new connections ands and go to svc functions** [UDP or TCP]. Then, **the functions get passed to a protocol** and **the client is unpacked** by the protocol function. And when the data is obtained, **it goes into rrootkit_proto_recv**. Then frameword_recv and data ends up in the view function

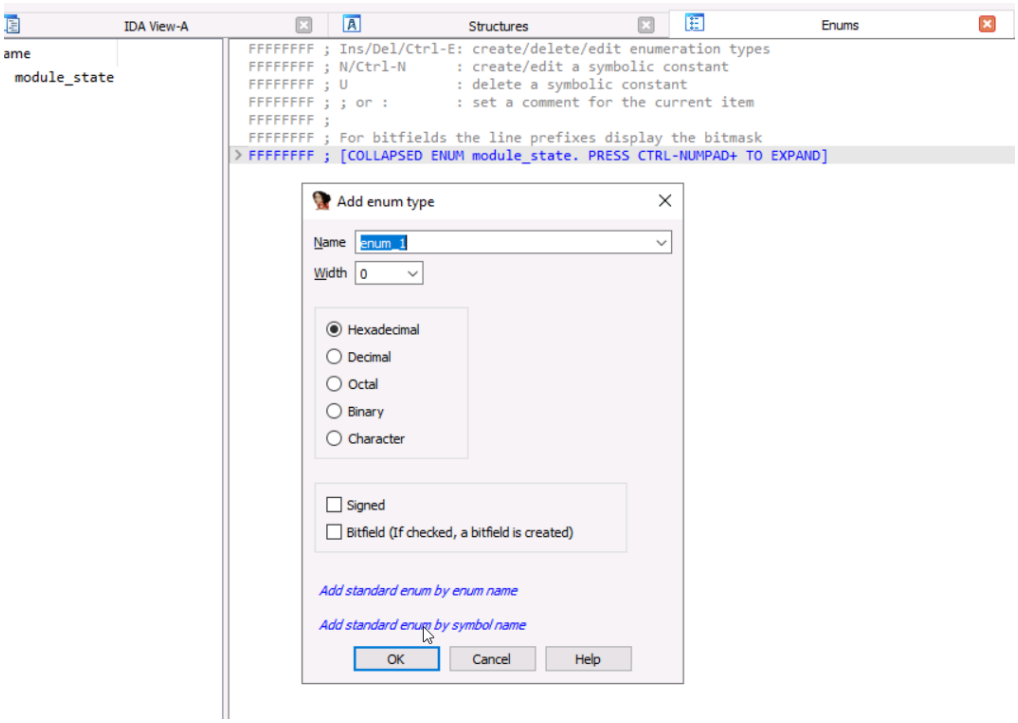
Reverse-engineer the rootkit

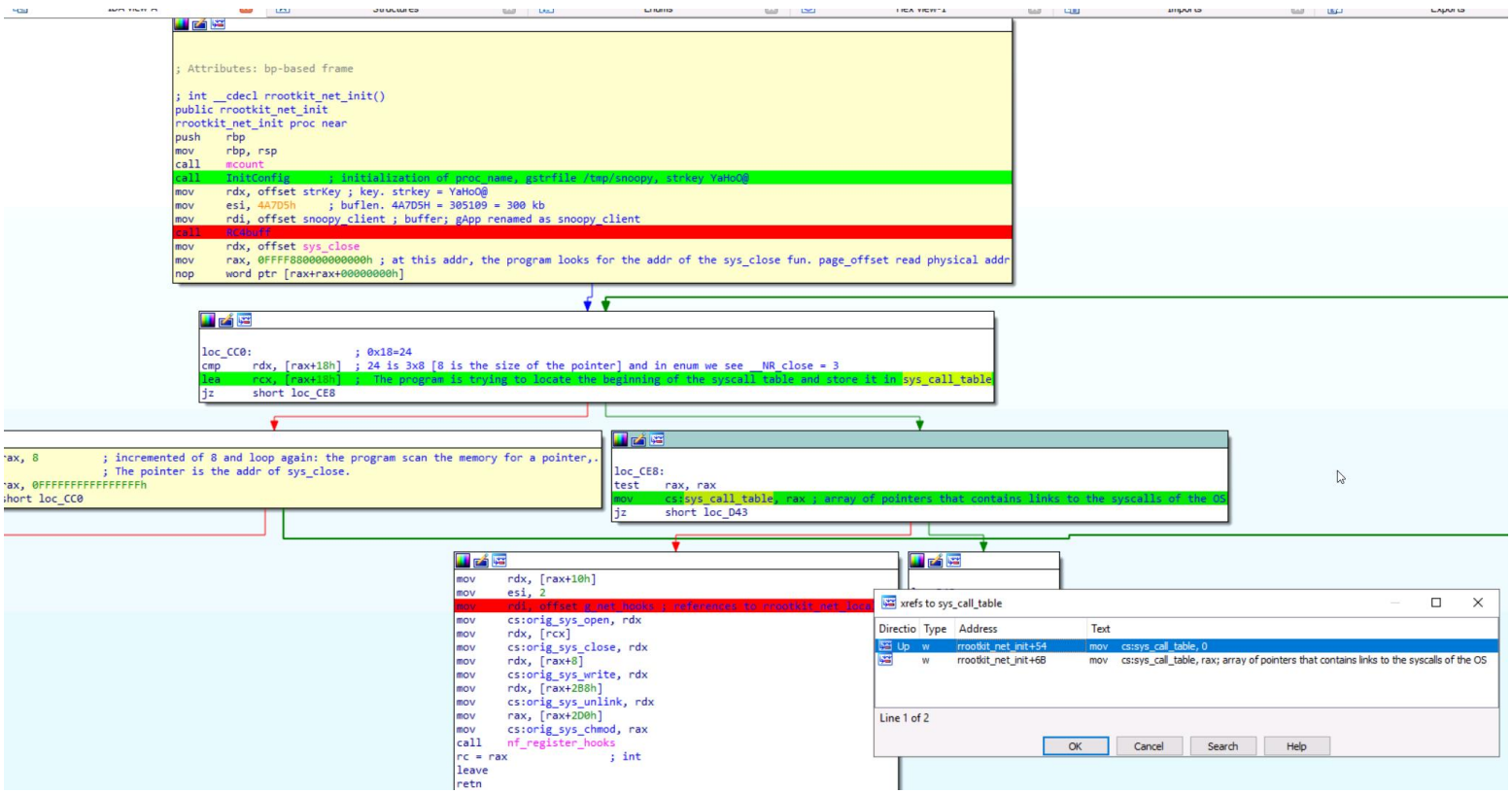
Reversing a rootkit is quite challenging because documentation for kernel functions and structures is hard to find, but we could look at the kernel's source code.

Minimodule -> rrootkit_net_init -> g_net_hook -> rrootkit_net_local_in

rrootkit_net_init

Which standard structures are expected by which function?

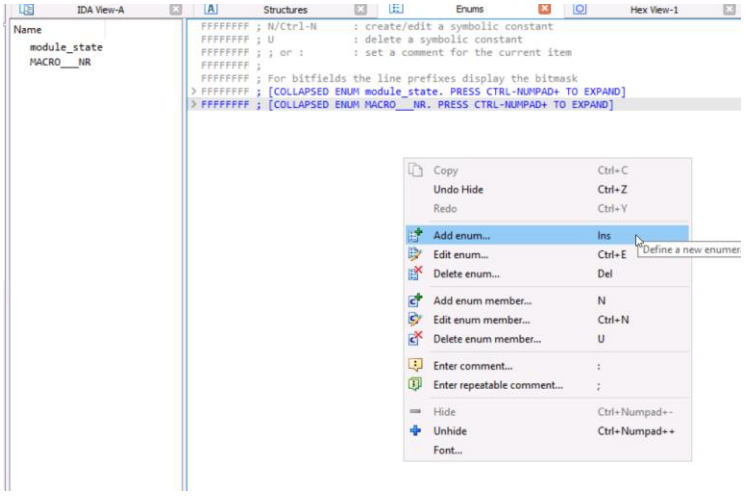


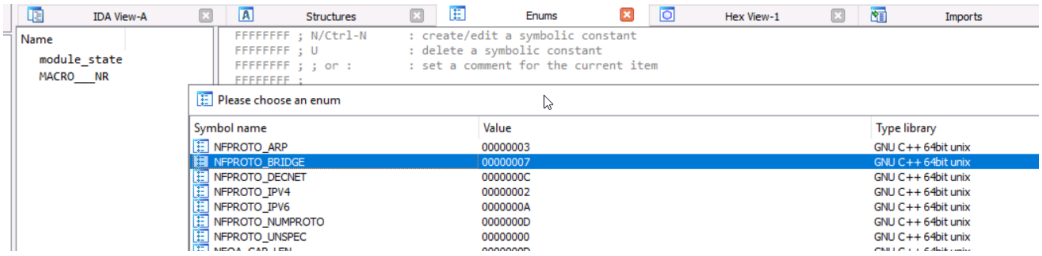


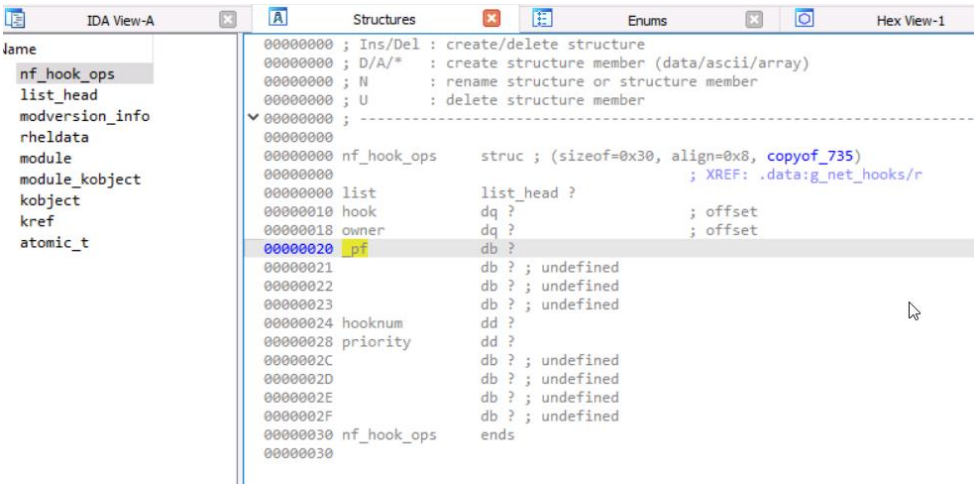
Not used but could be groundwork for future variants

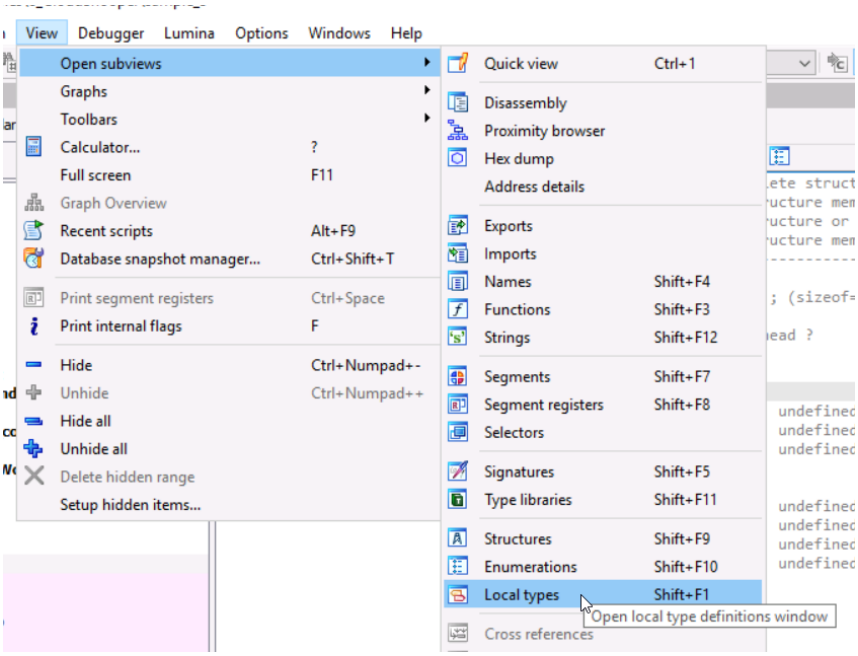
Continue the investigation of this function by checking the kernel code, and updating structures and enums:

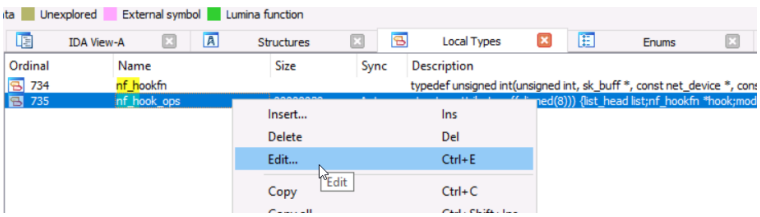
```
EXPORT_SYMBOL_GPL(br_netfilter_enable);

/* For br_nf_post_routing, we need (prio = NF_BR_PRI_LAST), because
 * br_dev_queue_push_xmit is called afterwards */
static struct nf_hook_ops br_nf_ops[] __read_mostly = {
        {
                .hook = br_nf_pre_routing,
                .pf = NFPROTO_BRIDGE,
                .hooknum = NF_BR_PRE_ROUTING,
                .priority = NF_BR_PRI_BRNF,
        },
        {
                .hook = br_nf_local_in,
                .pf = NFPROTO_BRIDGE,
                .hooknum = NF_BR_LOCAL_IN,
```

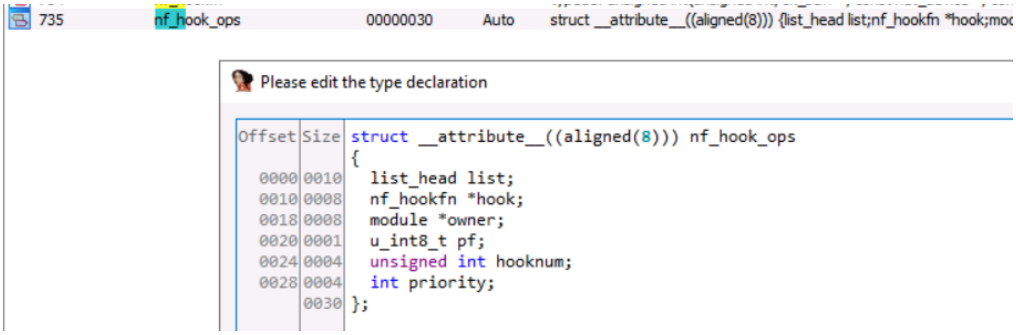


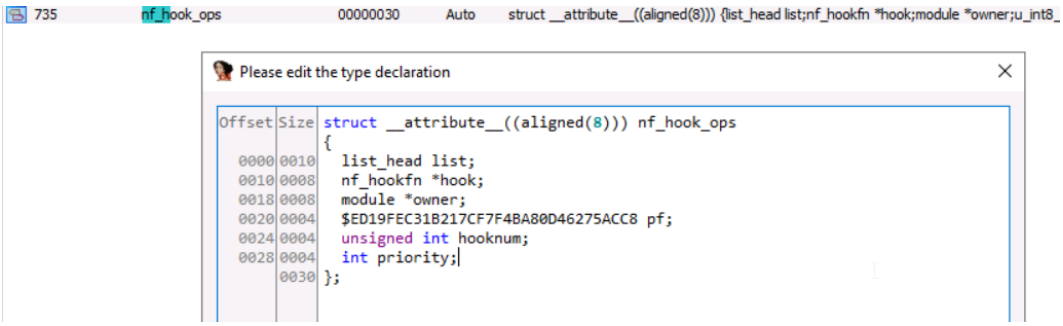

IDA View-A
Structures
Enums
Hex View-1
Imports
Name
module_state
MACRO___NR
FFFFFFFF ; N/Ctrl-N : create/edit a symbolic constant
FFFFFFFF ; U : delete a symbolic constant
FFFFFFFF ; ; or : : set a comment for the current item
Please choose an enum
Symbol name
Value
Type library
NFPROTO_ARP 00000003 GNU C++ 64bit unix
NFPROTO_BRIDGE 00000007 GNU C++ 64bit unix
NFPROTO_DECNET 0000000C GNU C++ 64bit unix
NFPROTO_IPV4 00000002 GNU C++ 64bit unix
NFPROTO_IPV6 0000000A GNU C++ 64bit unix
NFPROTO_NUMPROTO 0000000D GNU C++ 64bit unix
NFPROTO_UNSPEC 00000000 GNU C++ 64bit unix


IDA View-A
Structures
Enums
Hex View-1
nf_hook_ops
list_head
modversion_info
rheldata
module
module_kobject
kobject
kref
atomic_t
00000000 ; Ins/Del : create/delete structure
00000000 ; D/A/* : create structure member (data/ascii/array)
00000000 ; N : rename structure or structure member
00000000 ; U : delete structure member
00000000 nf_hook_ops struc ; (sizeof=0x30, align=0x8, copyof_735)
00000000 ; XREF: .data:g_net_hooks/r
00000000 list list_head ?
00000010 hook dq ? ; offset
00000018 owner dq ? ; offset
00000020 pf db ?
00000021 db ? ; undefined
00000022 db ? ; undefined
00000023 db ? ; undefined
00000024 hooknum dd ?
00000028 priority dd ?
0000002C db ? ; undefined
0000002D db ? ; undefined
0000002E db ? ; undefined
0000002F db ? ; undefined
00000030 nf_hook_ops ends
00000030


View
Debugger
Lumina
Options
Windows
Help
Open subviews
Graphs
Toolbars
Calculator... ?
Full screen F11
Graph Overview
Recent scripts Alt+F9
Database snapshot manager... Ctrl+Shift+T
Print segment registers Ctrl+Space
Print internal flags F
Hide Ctrl+Numpad+-
Unhide Ctrl+Numpad++
Hide all
Unhide all
Delete hidden range
Setup hidden items...
Quick view Ctrl+1
Disassembly
Proximity browser
Hex dump
Address details
Exports
Imports
Names Shift+F4
Functions Shift+F3
Strings Shift+F12
Segments Shift+F7
Segment registers Shift+F8
Selectors
Signatures Shift+F5
Type libraries Shift+F11
Structures Shift+F9
Enumerations Shift+F10
Local types Shift+F1
Cross references
Open local type definitions window





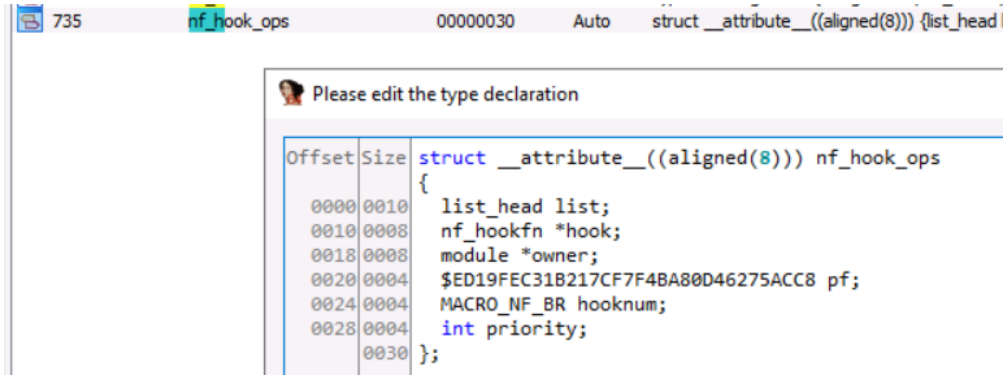


Before:

```
; nf_hook_ops g_net_hooks[2]
g_net_hooks     dq ?                    ; list.next
                                        ; DATA XREF: rrootkit_net_exit+E1o
                                        ; rrootkit_net_init+7D1o
                dq ?                    ; list.prev
                dq offset rrootkit_net_local_in; hook
                dq offset __this_module ; owner
                db 2                    ; _pf
                db 3 dup(0)
                dd 1                    ; hooknum
                dd 80000000h            ; priority
                db 4 dup(0)
                dq ?                    ; list.next
                dq ?                    ; list.prev
                dq offset rrootkit_net_local_out; hook
                dq offset __this_module ; owner
                db 2                    ; _pf
                db 3 dup(0)
                dd 3                    ; hooknum
                dd 80000000h            ; priority
                db 4 dup(0)
_data           ends
```

After:

```
; nf_hook_ops g_net_hooks[2]
g_net_hooks     nf_hook_ops <<0>, offset rrootkit_net_local_in, offset __this_module, \
                                        ; DATA XREF: rrootkit_net_exit+E1o
                                        ; rrootkit_net_init+7D1o
                                NFPROTO_IPV4, 1, 80000000h>
                nf_hook_ops <<0>, offset rrootkit_net_local_out, offset __this_module,\
                                NFPROTO_IPV4, 3, 80000000h>
_data           ends
```

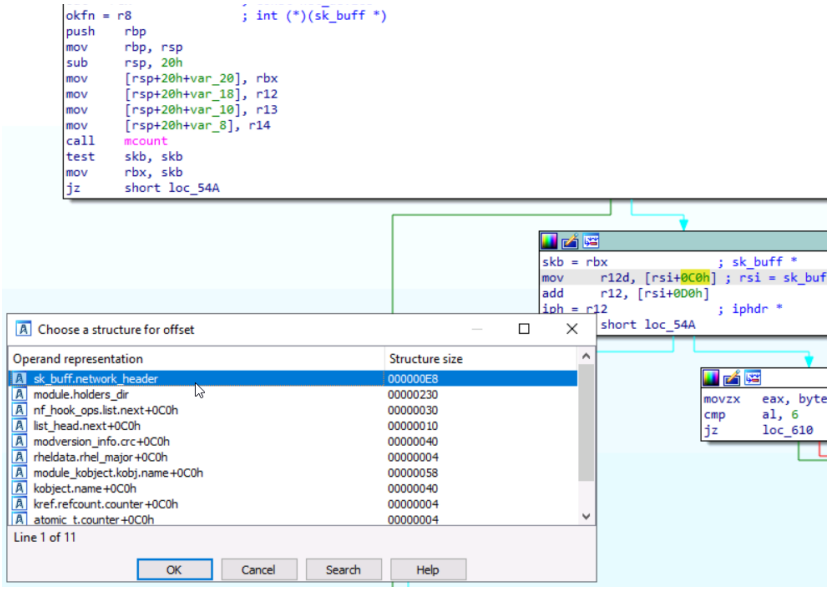


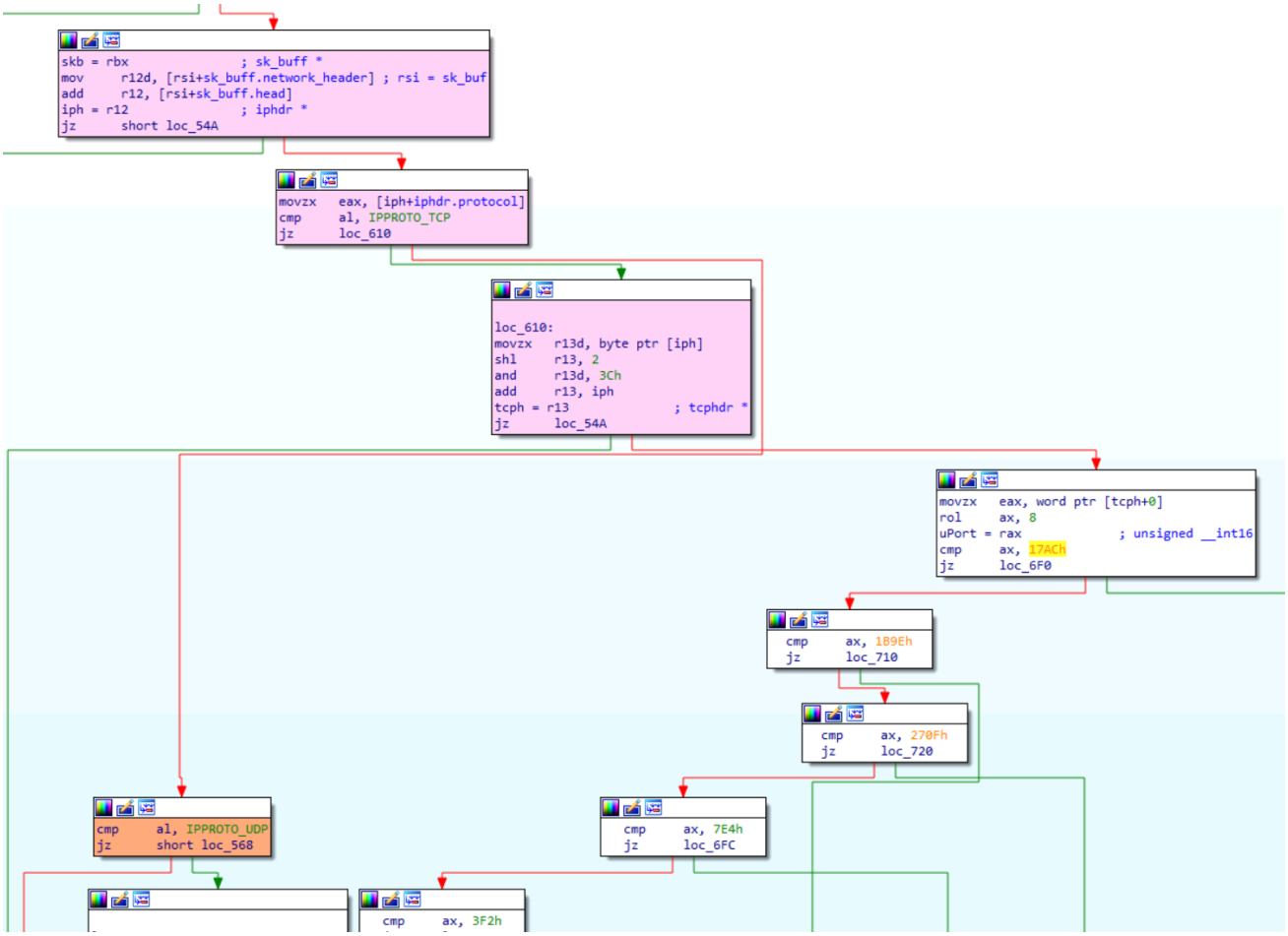

```
                align 20h
; nf_hook_ops g_net_hooks[2]
g_net_hooks     nf_hook_ops <<0>, offset rrootkit_net_local_in, offset __this_module, \
                                ; DATA XREF: rrootkit_net_exit+E1o
                                ; rrootkit_net_init+7D1o
                          NFPROTO_IPV4, NF_BR_LOCAL_IN, 80000000h>
                nf_hook_ops <<0>, offset rrootkit_net_local_out, offset __this_module,\
                          NFPROTO_IPV4, NF_BR_LOCAL_OUT, 80000000h>
_data           ends
```

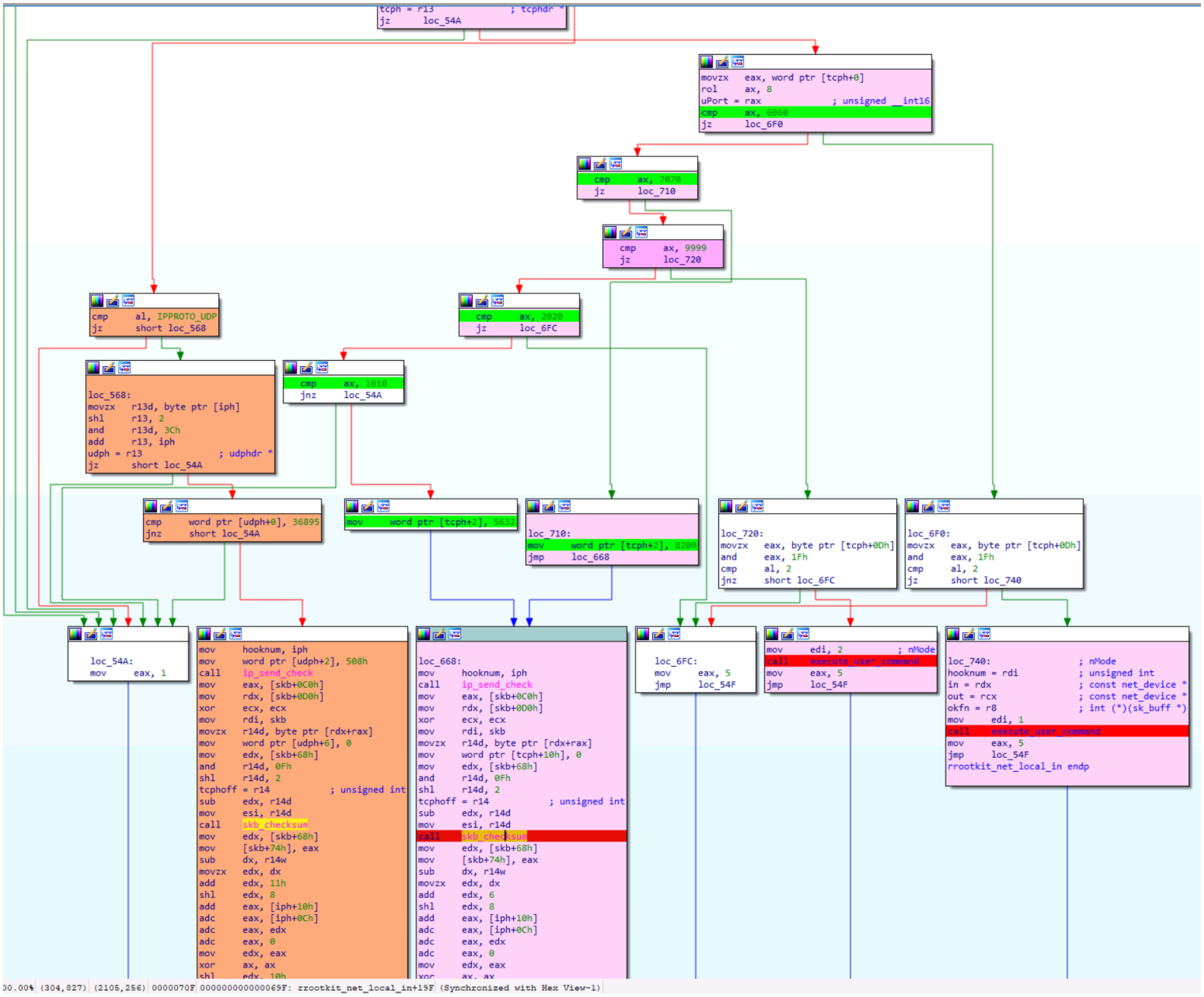
```
/* Bridge Hooks */
/* After promisc drops, checksum checks. */
#define NF_BR_PRE_ROUTING       0
/* If the packet is destined for this box. */
#define NF_BR_LOCAL_IN          1
/* If the packet is destined for another interface. */
#define NF_BR_FORWARD           2
/* Packets coming from a local process. */
#define NF_BR_LOCAL_OUT         3
/* Packets about to hit the wire. */
#define NF_BR_POST_ROUTING      4
/* Not really a hook, but used for the ebtables broute table */
#define NF_BR_BROUTING          5
#define NF_BR_NUMHOOKS          6

enum nf_br_hook_priorities {
        NF_BR_PRI_FIRST = INT_MIN,
        NF_BR_PRI_NAT_DST_BRIDGED = -300,
        NF_BR_PRI_FILTER_BRIDGED = -200,
        NF_BR_PRI_BRNF = 0,
        NF_BR_PRI_NAT_DST_OTHER = 100,
        NF_BR_PRI_FILTER_OTHER = 200,
        NF_BR_PRI_NAT_SRC = 300,
        NF_BR_PRI_LAST = INT_MAX,
};
```

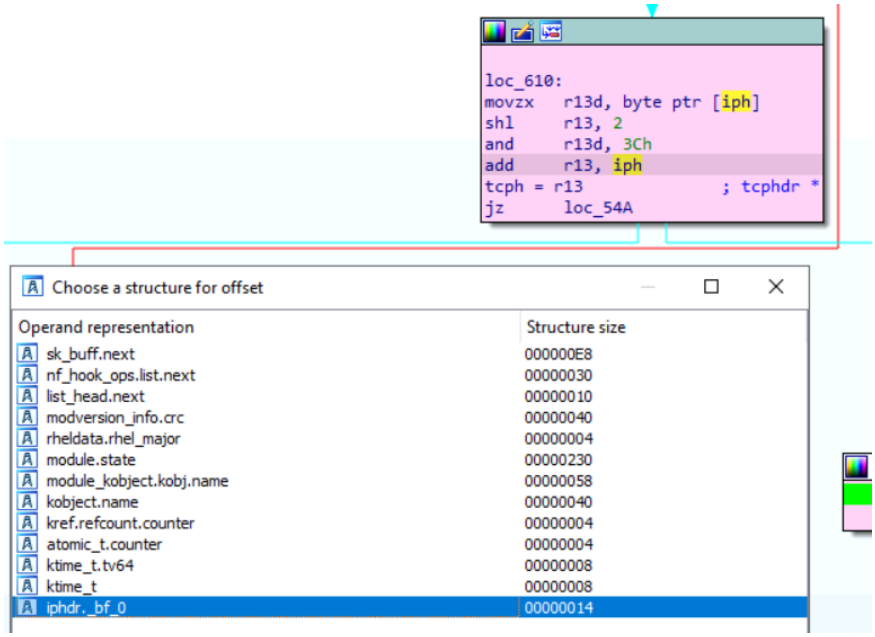
These updates confirm the references to rrootkit_net_local_in, and also rrootkit_net_local_out

rrootkit_net_local_in






tcph = r13 ; tcphdr *
jz loc_54A
movzx eax, word ptr [tcph+0]
rol ax, 8
uPort = rax ; unsigned __int16
cmp ax, 6868
jz loc_6F0
cmp ax, 7070
jz loc_710
cmp ax, 9999
jz loc_720
cmp al, IPPROTO_UDP
jz short loc_568
cmp ax, 2020
jz loc_6FC
loc_568:
movzx r13d, byte ptr [iph]
shl r13, 2
and r13d, 3Ch
add r13, iph
udph = r13 ; udphdr *
jz short loc_54A
cmp ax, 1010
jnz loc_54A
cmp word ptr [udph+0], 36895
jnz short loc_54A
mov word ptr [tcph+2], 5632
loc_710:
mov word ptr [tcph+2], 8200
jmp loc_668
loc_720:
movzx eax, byte ptr [tcph+0Dh]
and eax, 1Fh
cmp al, 2
jnz short loc_6FC
loc_6F0:
movzx eax, byte ptr [tcph+0Dh]
and eax, 1Fh
cmp al, 2
jz short loc_740
loc_54A:
mov eax, 1
mov hooknum, iph
mov word ptr [udph+2], 508h
call ip_send_check
mov eax, [skb+0C0h]
mov rdx, [skb+0D0h]
xor ecx, ecx
mov rdi, skb
movzx r14d, byte ptr [rdx+rax]
mov word ptr [udph+6], 0
mov edx, [skb+68h]
and r14d, 0Fh
shl r14d, 2
tcphoff = r14 ; unsigned int
sub edx, r14d
mov esi, r14d
call skb_checksum
mov edx, [skb+68h]
mov [skb+74h], eax
sub dx, r14w
movzx edx, dx
add edx, 11h
shl edx, 8
add eax, [iph+10h]
adc eax, [iph+0Ch]
adc eax, edx
adc eax, 0
mov edx, eax
xor ax, ax
loc_668:
mov hooknum, iph
call ip_send_check
mov eax, [skb+0C0h]
mov rdx, [skb+0D0h]
xor ecx, ecx
mov rdi, skb
movzx r14d, byte ptr [rdx+rax]
mov word ptr [tcph+10h], 0
mov edx, [skb+68h]
and r14d, 0Fh
shl r14d, 2
tcphoff = r14 ; unsigned int
sub edx, r14d
mov esi, r14d
call skb_checksum
mov edx, [skb+68h]
mov [skb+74h], eax
sub dx, r14w
movzx edx, dx
add edx, 6
shl edx, 8
add eax, [iph+10h]
adc eax, [iph+0Ch]
adc eax, edx
adc eax, 0
mov edx, eax
mov eax, 5
jmp loc_54F
mov edi, 2 ; nMode
call execute_user_command
mov eax, 5
jmp loc_54F
loc_740: ; nMode
hooknum = rdi ; unsigned int
in = rdx ; const net_device *
out = rcx ; const net_device *
okfn = r8 ; int (*)(sk_buff *)
mov edi, 1
call execute_user_command
mov eax, 5
jmp loc_54F
rrootkit_net_local_in endp



IPv4 header format

Offsets	Octet	0								1								2								3							
Octet	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	Version				IHL				DSCP						ECN		Total Length															
4	32	Identification																Flags			Fragment Offset											
8	64	Time To Live								Protocol								Header Checksum															
12	96	Source IP Address																															
16	128	Destination IP Address																															
20	160	Options (if IHL > 5)																															
⋮	⋮																																
56	448																																

"The fields in the header are packed with the most significant byte first (big endian)"

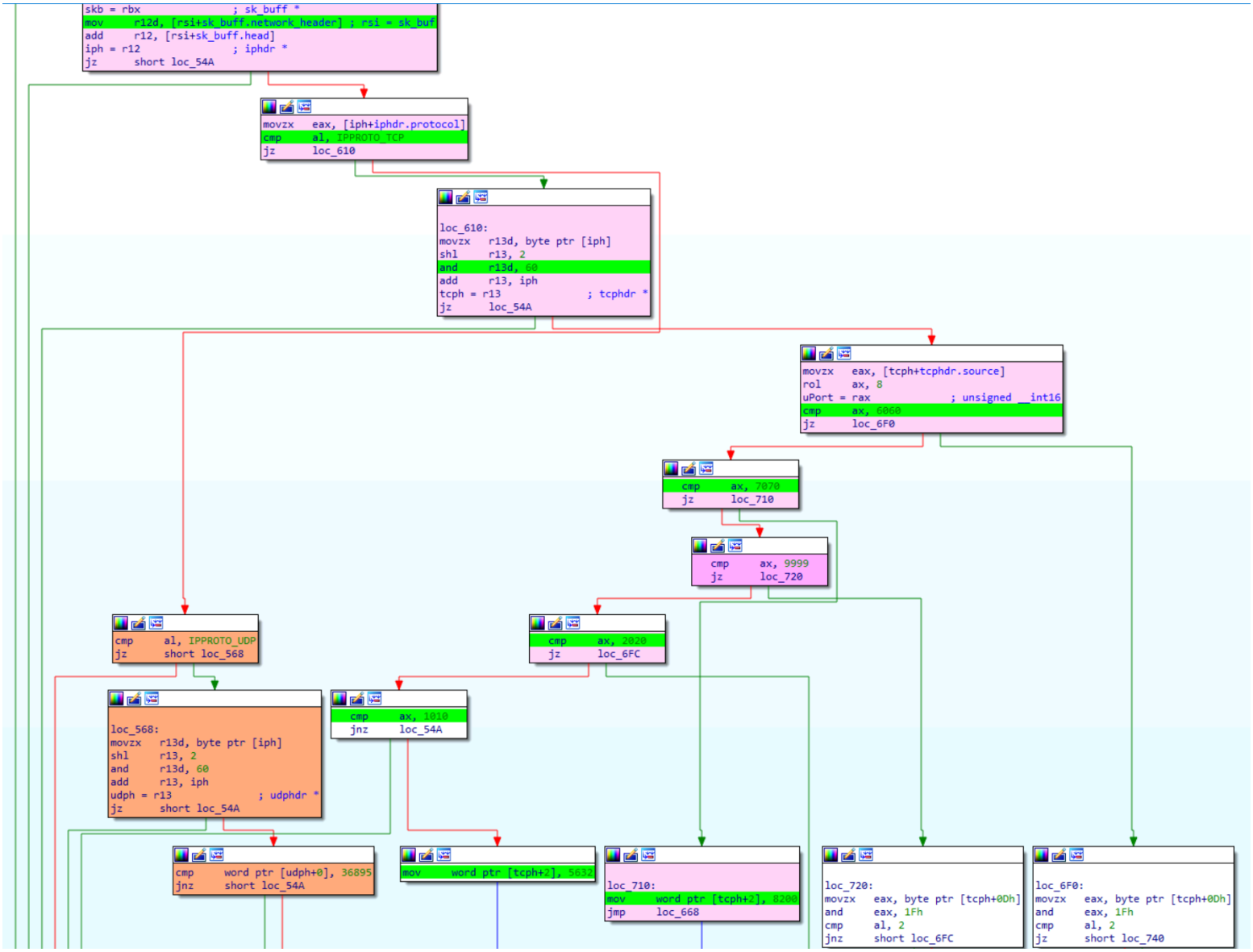
ip and iphdr struct:

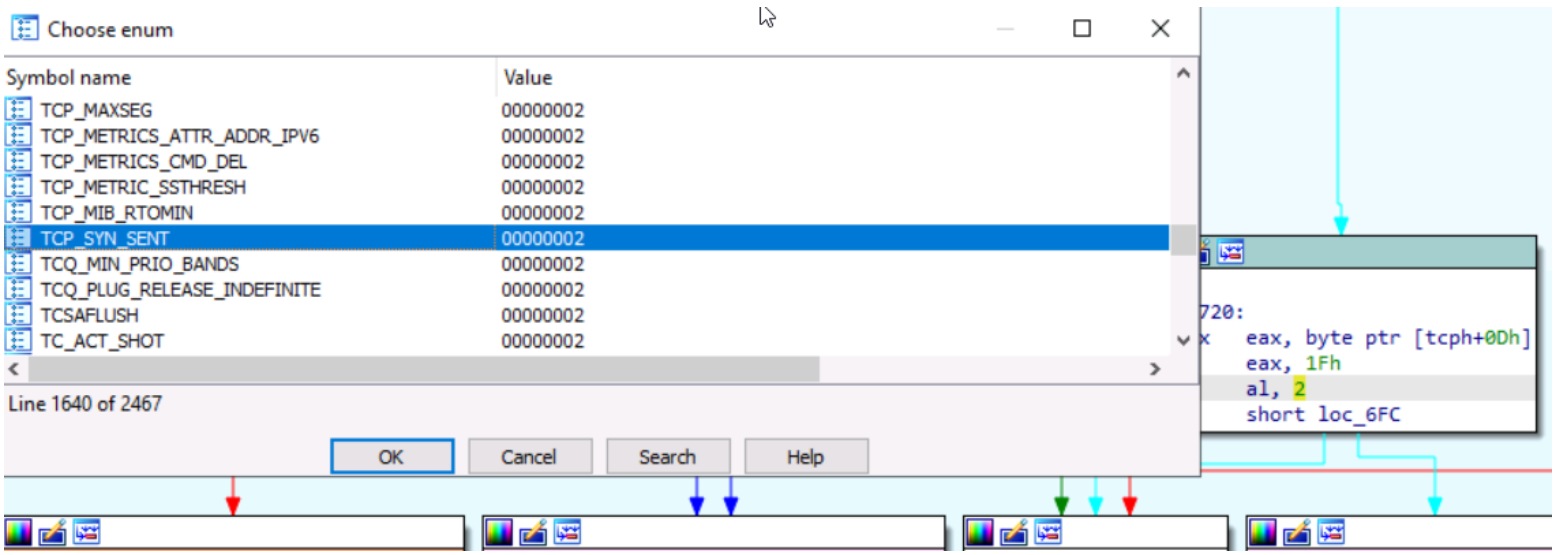
```
struct iphdr {
    #if defined(__LITTLE_ENDIAN_BITFIELD)
        __u8    ihl:4,
                version:4;
    #elif defined (__BIG_ENDIAN_BITFIELD)
        __u8    version:4,
                ihl:4;
    #else
        #error  "Please fix <asm/byteorder.h>"
    #endif
         __u8   tos;
         __u16  tot_len;
         __u16  id;
         __u16  frag_off;
         __u8   ttl;
         __u8   protocol;
         __u16  check;
         __u32  saddr;
         __u32  daddr;
         /*The options start here. */
};
```

→ 4 first bits = IHL [Internet Header Lengh]=number of fields in the IP header

Internet Header Length (IHL)

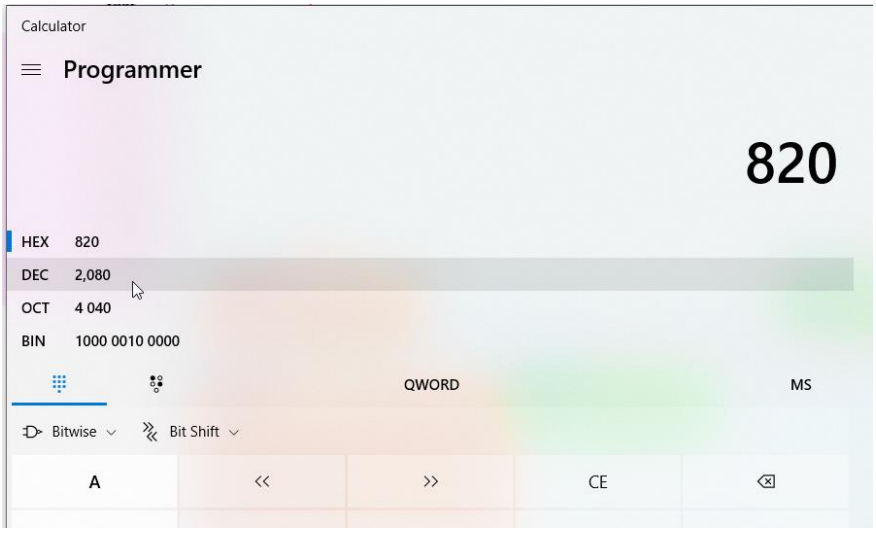
The IPv4 header is variable in size due to the optional 14th field (options). The IHL field contains the size of the IPv4 header; it has 4 bits that specify the number of 32-bit words in the header. The minimum value for this field is 5,[37] which indicates a length of 5 × 32 bits = 160 bits = 20 bytes. As a 4-bit field, the maximum value is 15; this means that the maximum size of the IPv4 header is 15 × 32 bits = 480 bits = 60 bytes.





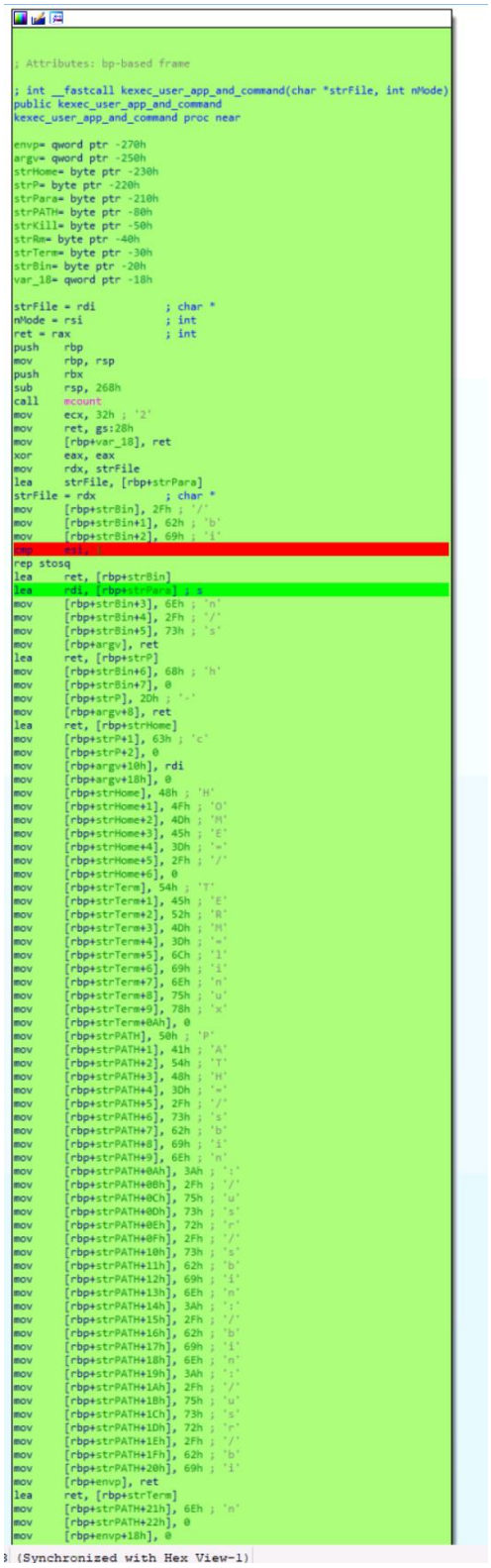
If tcp packet with 6060 source port -> execute_user_command

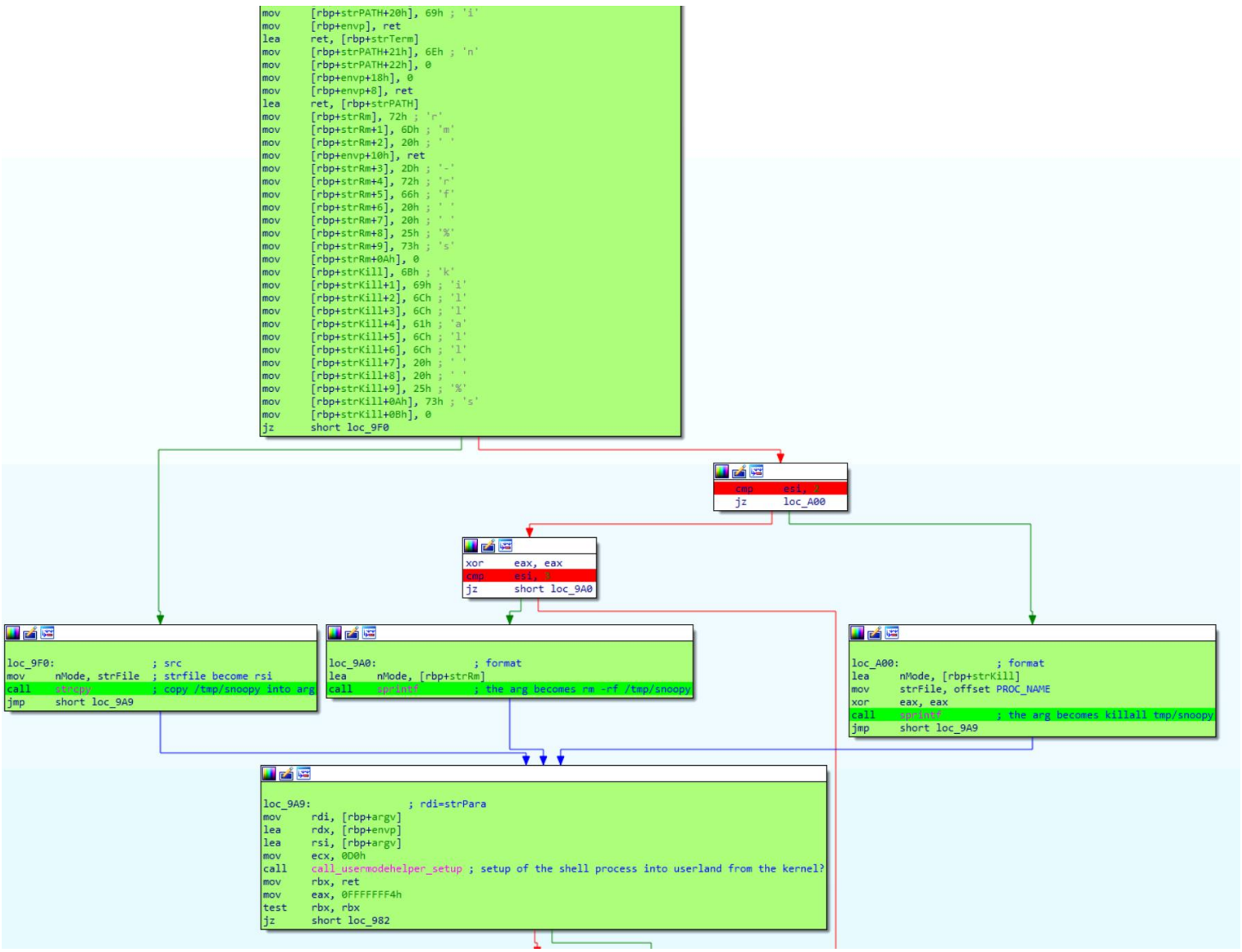
If tcp packet with 7070 source port → update the dest tcp port as 2080

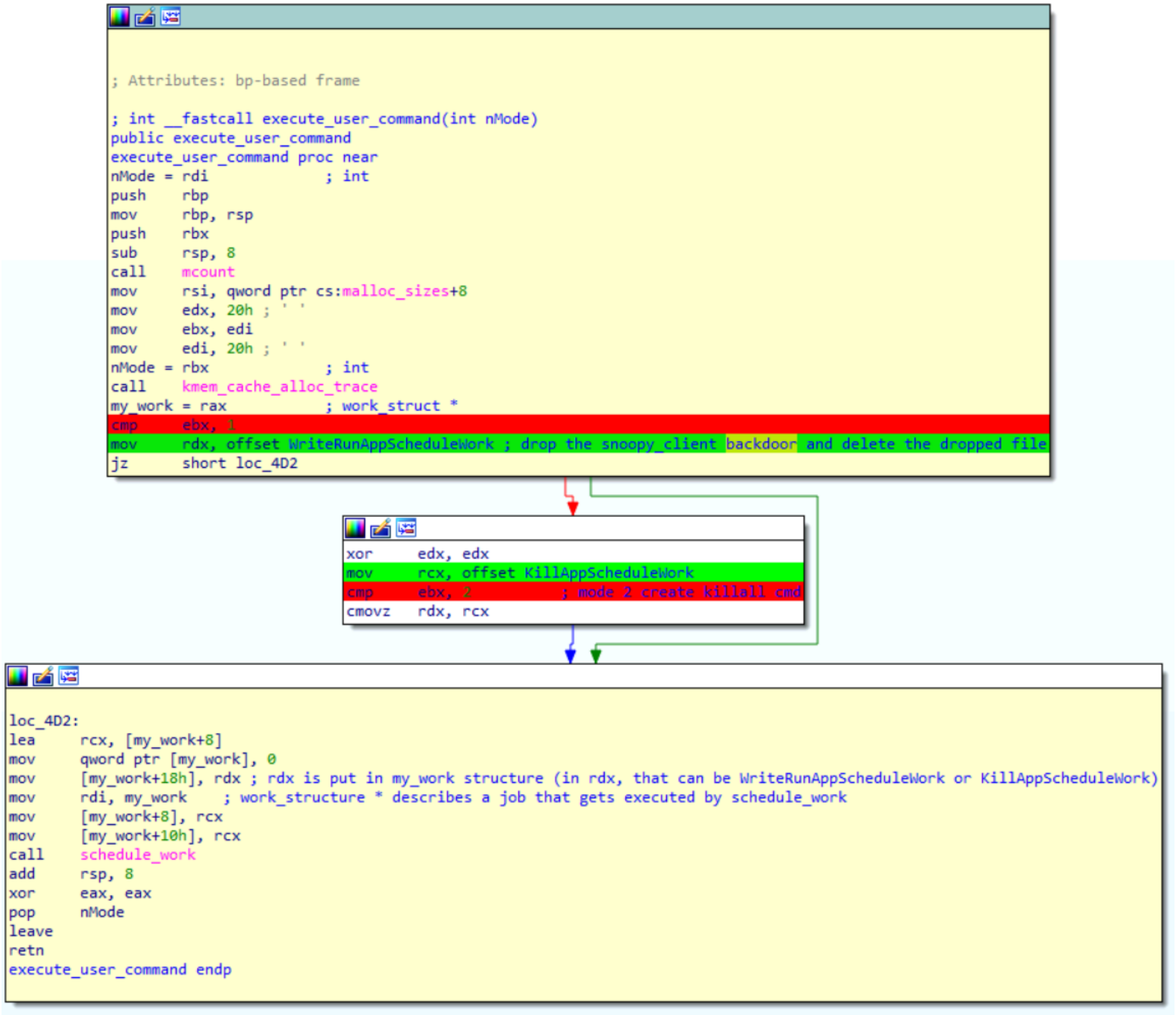


If tcp packet with 9999 source port → -> execute_user_command

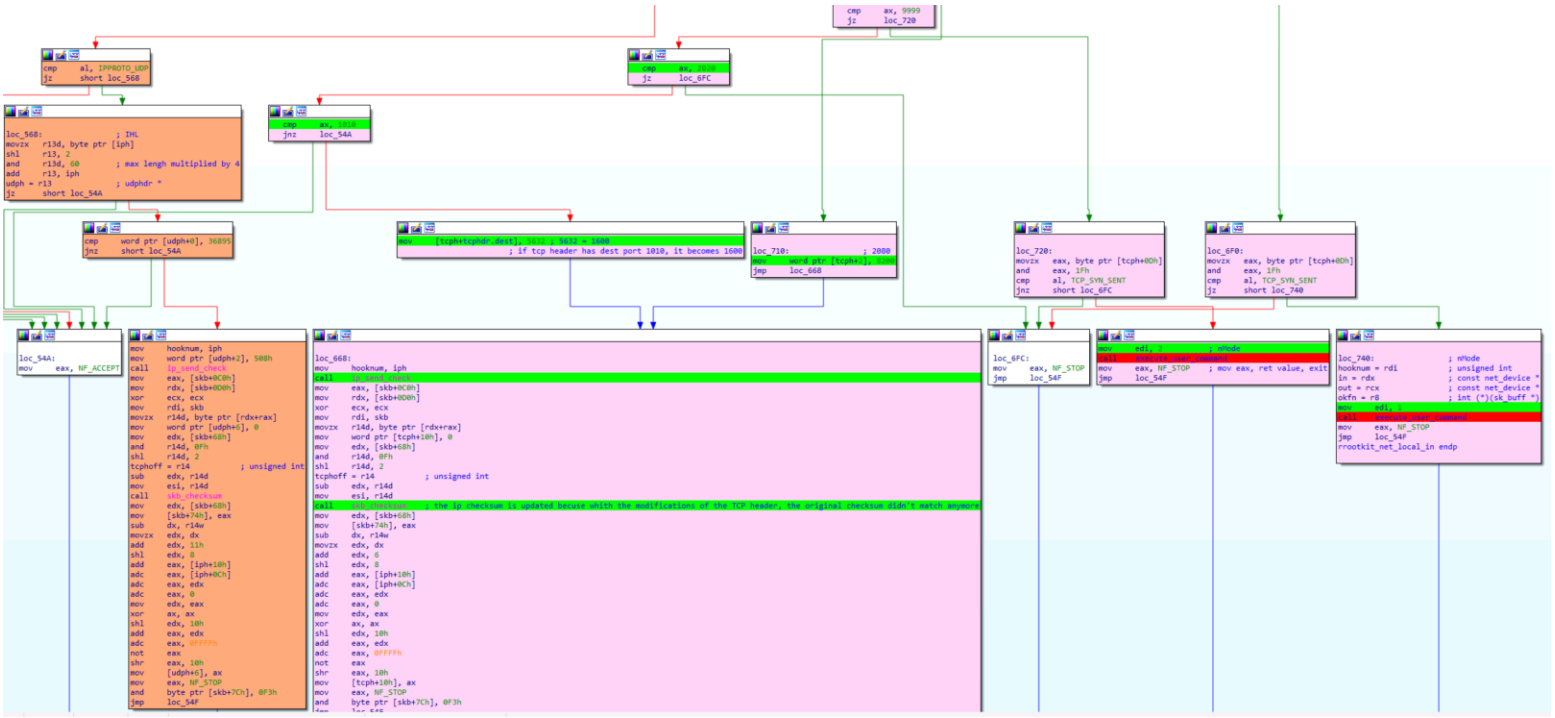
Investigating the Kexec_user_app_command

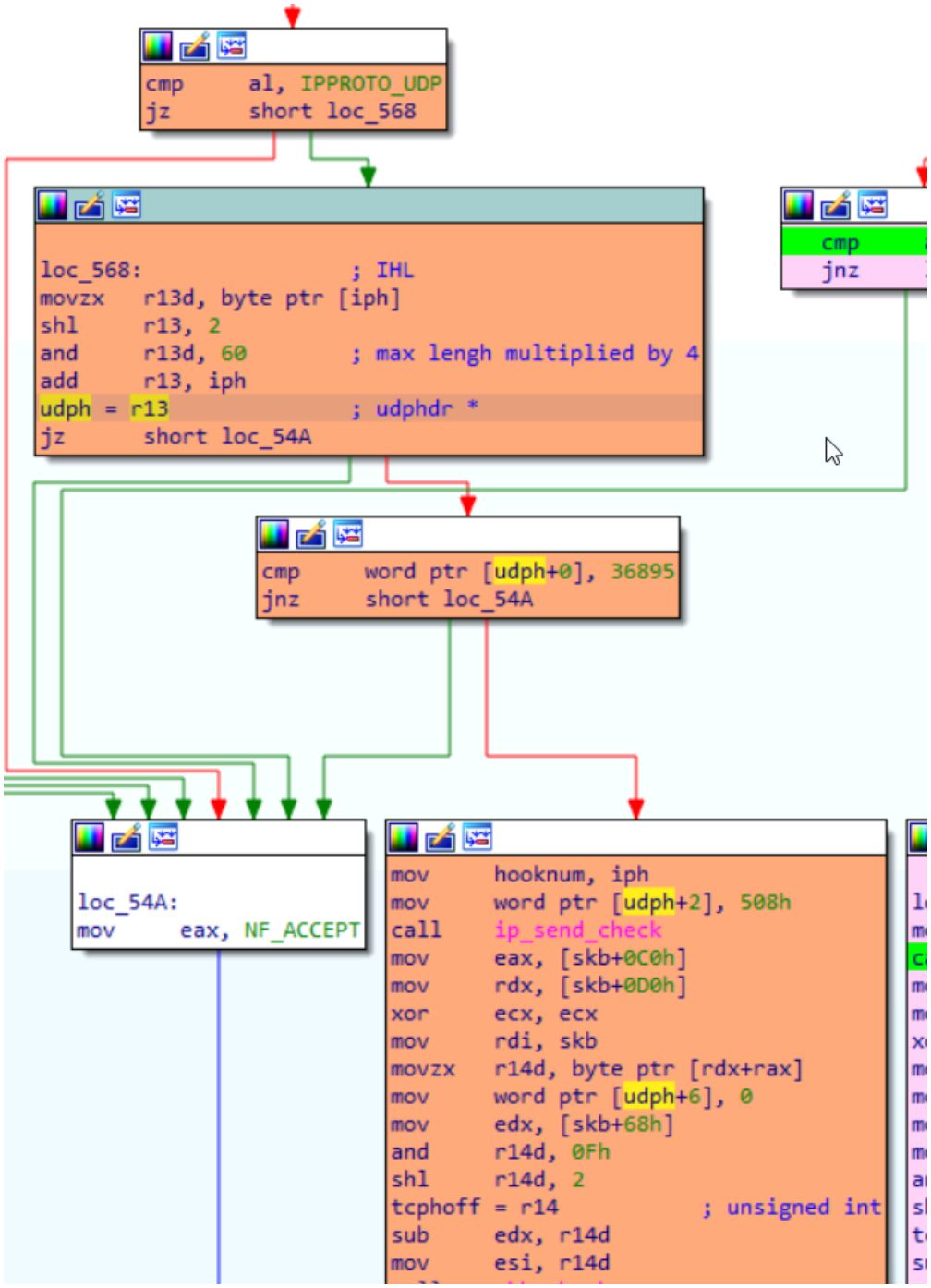



mov [rbp+strPATH+20h], 69h ; 'i'
mov [rbp+envp], ret
lea ret, [rbp+strTerm]
mov [rbp+strPATH+21h], 6Eh ; 'n'
mov [rbp+strPATH+22h], 0
mov [rbp+envp+18h], 0
mov [rbp+envp+8], ret
lea ret, [rbp+strPATH]
mov [rbp+strRm], 72h ; 'r'
mov [rbp+strRm+1], 6Dh ; 'm'
mov [rbp+strRm+2], 20h ; ' '
mov [rbp+envp+10h], ret
mov [rbp+strRm+3], 2Dh ; '-'
mov [rbp+strRm+4], 72h ; 'r'
mov [rbp+strRm+5], 66h ; 'f'
mov [rbp+strRm+6], 20h ; ' '
mov [rbp+strRm+7], 20h ; ' '
mov [rbp+strRm+8], 25h ; '%'
mov [rbp+strRm+9], 73h ; 's'
mov [rbp+strRm+0Ah], 0
mov [rbp+strKill], 6Bh ; 'k'
mov [rbp+strKill+1], 69h ; 'i'
mov [rbp+strKill+2], 6Ch ; 'l'
mov [rbp+strKill+3], 6Ch ; 'l'
mov [rbp+strKill+4], 61h ; 'a'
mov [rbp+strKill+5], 6Ch ; 'l'
mov [rbp+strKill+6], 6Ch ; 'l'
mov [rbp+strKill+7], 20h ; ' '
mov [rbp+strKill+8], 20h ; ' '
mov [rbp+strKill+9], 25h ; '%'
mov [rbp+strKill+0Ah], 73h ; 's'
mov [rbp+strKill+0Bh], 0
jz short loc_9F0
cmp esi, 2
jz loc_A00
xor eax, eax
cmp esi, 3
jz short loc_9A0
loc_9F0: ; src
mov nMode, strFile ; strfile become rsi
call strcpy ; copy /tmp/snoopy into arg
jmp short loc_9A9
loc_9A0: ; format
lea nMode, [rbp+strRm]
call sprintf ; the arg becomes rm -rf /tmp/snoopy
loc_A00: ; format
lea nMode, [rbp+strKill]
mov strFile, offset PROC_NAME
xor eax, eax
call sprintf ; the arg becomes killall tmp/snoopy
jmp short loc_9A9
loc_9A9: ; rdi=strPara
mov rdi, [rbp+argv]
lea rdx, [rbp+envp]
lea rsi, [rbp+argv]
mov ecx, 0D0h
call call_usermodehelper_setup ; setup of the shell process into userland from the kernel?
mov rbx, ret
mov eax, 0FFFFFFF4h
test rbx, rbx
jz short loc_982


; Attributes: bp-based frame
; int __fastcall execute_user_command(int nMode)
public execute_user_command
execute_user_command proc near
nMode = rdi ; int
push rbp
mov rbp, rsp
push rbx
sub rsp, 8
call mcount
mov rsi, qword ptr cs:malloc_sizes+8
mov edx, 20h ; ' '
mov ebx, edi
mov edi, 20h ; ' '
nMode = rbx ; int
call kmem_cache_alloc_trace
my_work = rax ; work_struct *
cmp ebx, 1
mov rdx, offset WriteRunAppScheduleWork ; drop the snoopy_client backdoor and delete the dropped file
jz short loc_4D2
xor edx, edx
mov rcx, offset KillAppScheduleWork
cmp ebx, 2 ; mode 2 create killall cmd
cmovz rdx, rcx
loc_4D2:
lea rcx, [my_work+8]
mov qword ptr [my_work], 0
mov [my_work+18h], rdx ; rdx is put in my_work structure (in rdx, that can be WriteRunAppScheduleWork or KillAppScheduleWork)
mov rdi, my_work ; work_structure * describes a job that gets executed by schedule_work
mov [my_work+8], rcx
mov [my_work+10h], rcx
call schedule_work
add rsp, 8
xor eax, eax
pop nMode
leave
retn
execute_user_command endp

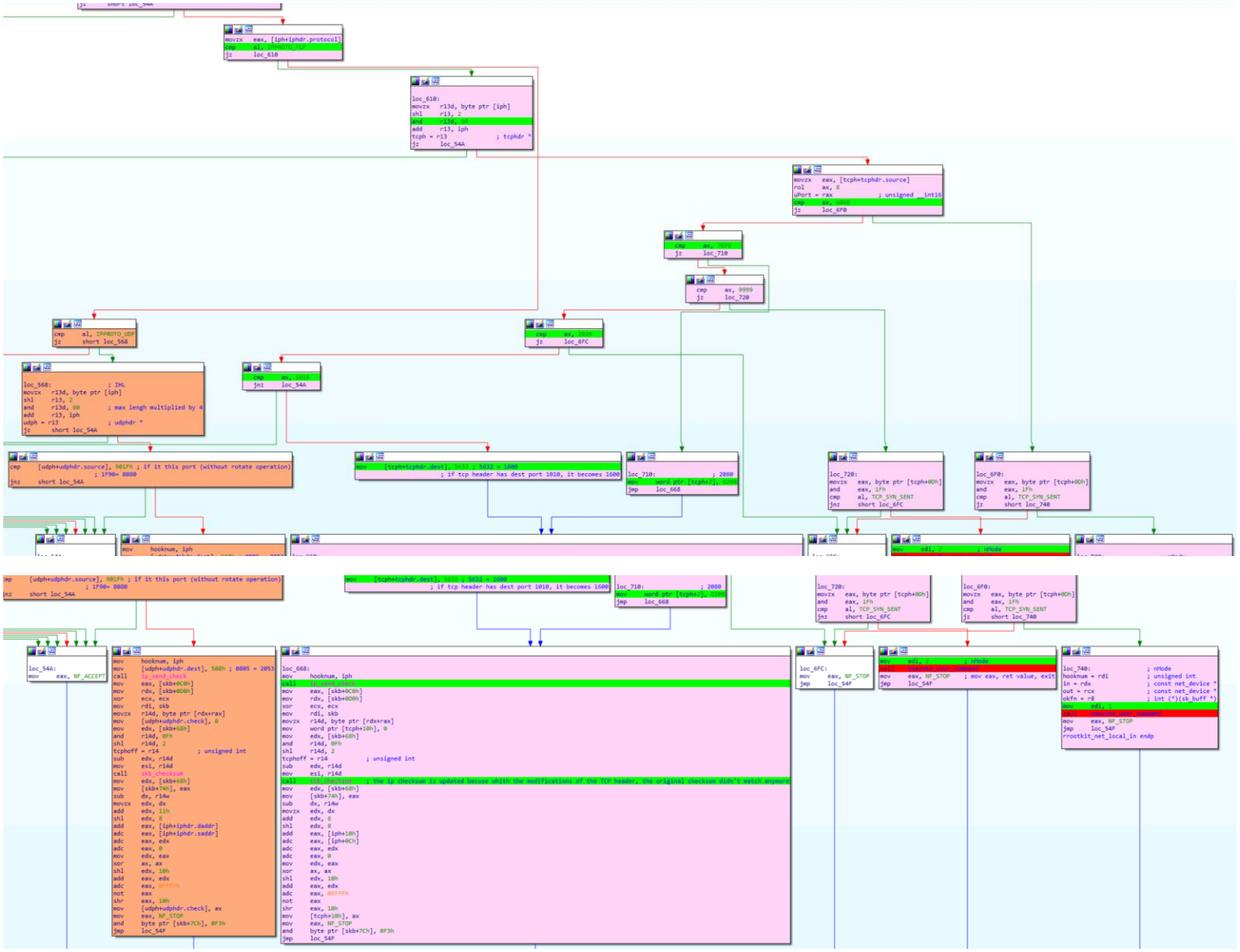
After investigating TCP branch (pink), we investigate UDP branch (orange):





Import udph structure and update the code.

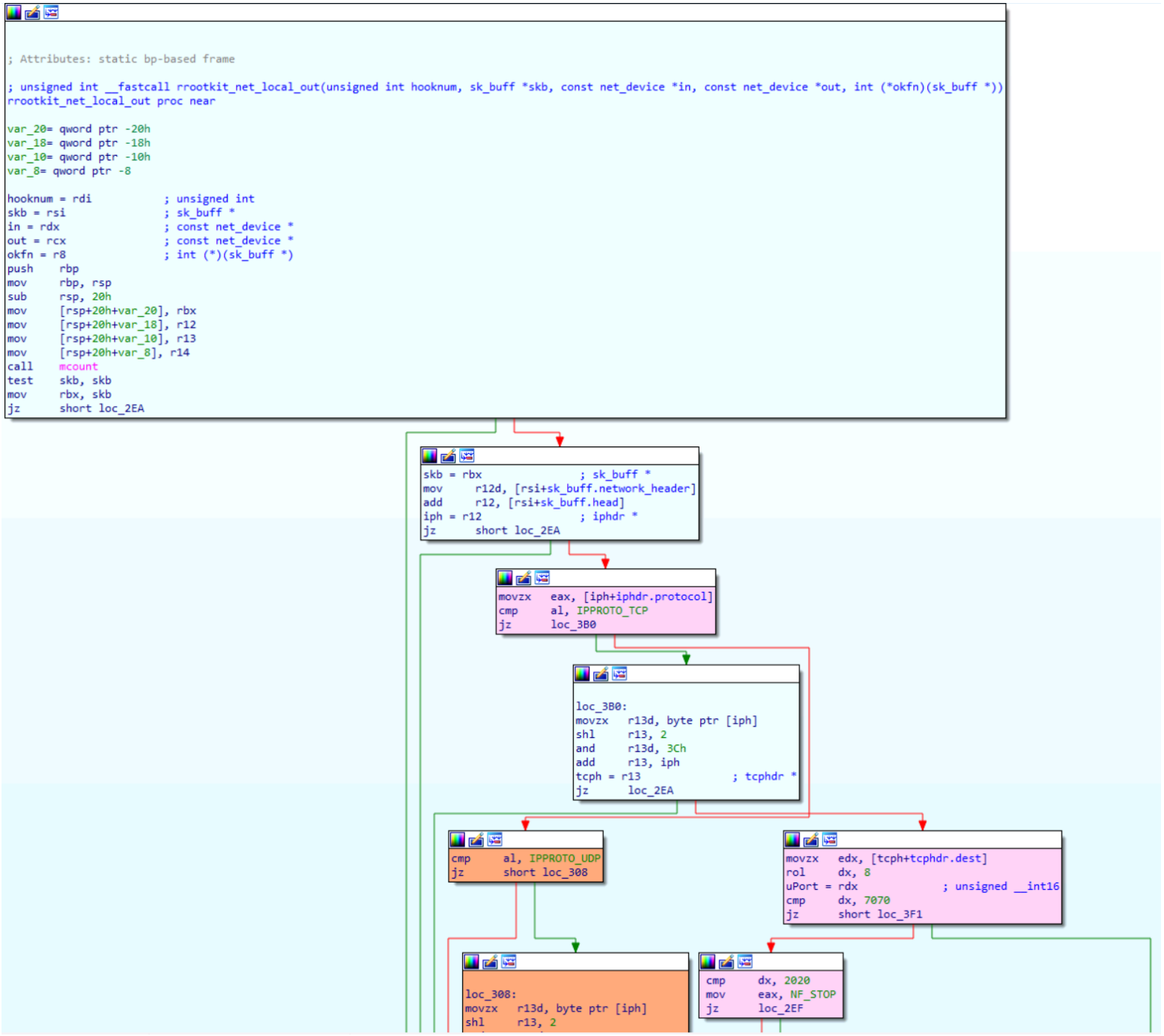
How the attacker is able to establish a connection with the snoopy client? Focus on the firewall evasion technique

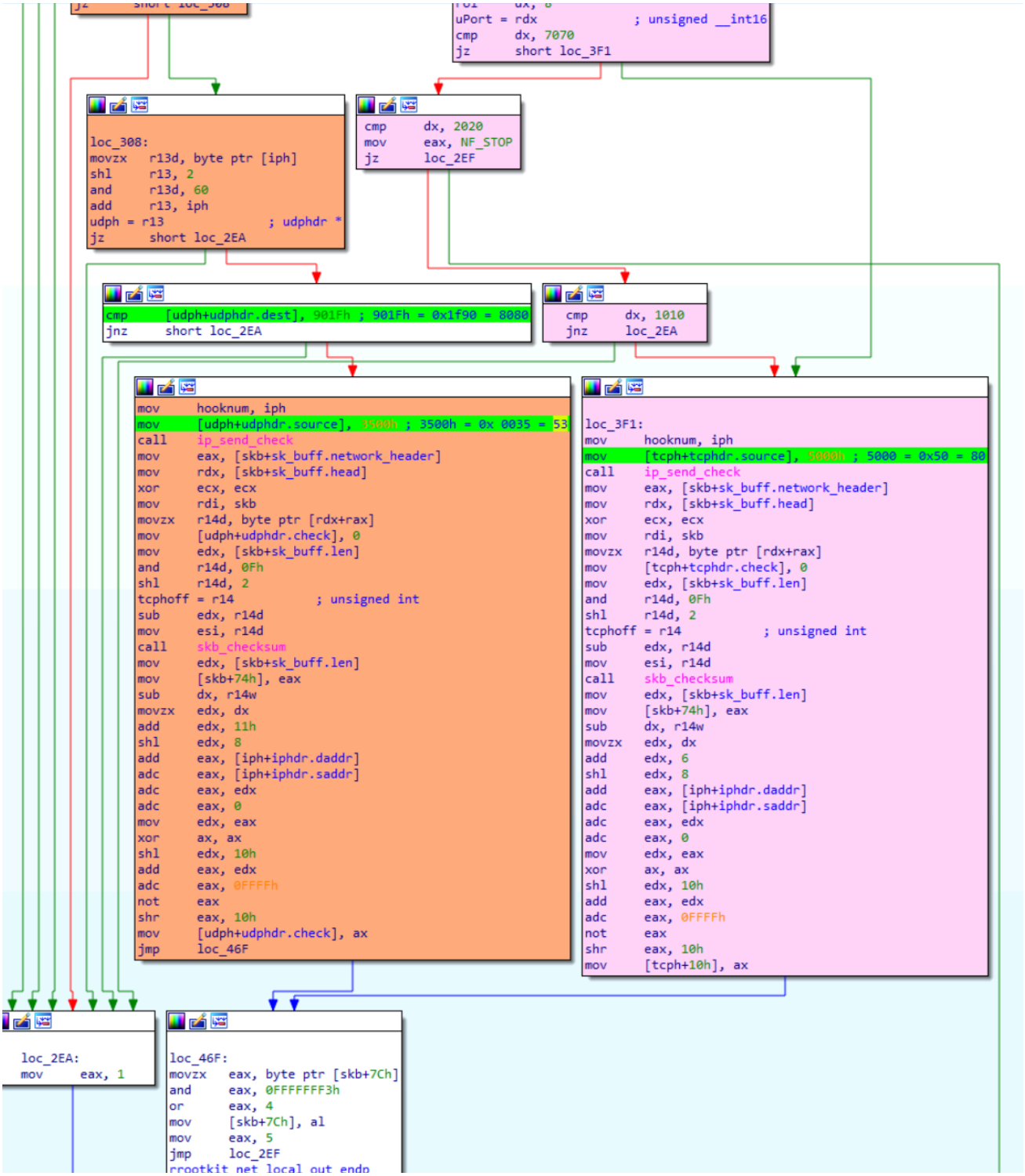


Depending on the protocol is UDP or TCP, if the source port is provided by the user, the rootkit will update the dest port of the packet and fix it = if the firewall is up on the machine, it doesn't matter what the firewall rules are because the packet can come on the port 60 as long as the source port is correct, then the rootkit will update the dest port after the firewall has inspected the packet. This is a firewall evasion technique.

rrootkit_net_local_out

repeating the same steps for this function.



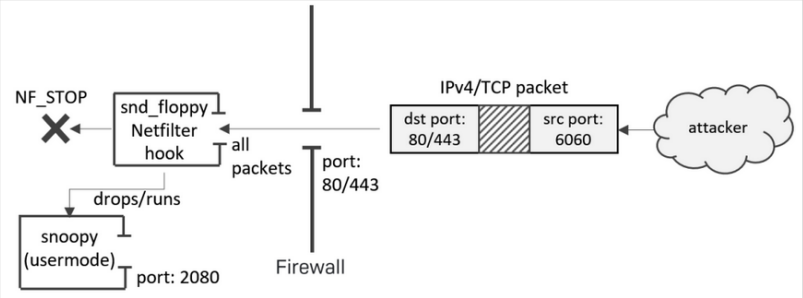


This function intercept outbound traffic, and depending on which protocol is detected, UDP, or TCP, it changes the source port as 80 or 53, as if it is webserver request, or dns request, in order to bypass the firewall.

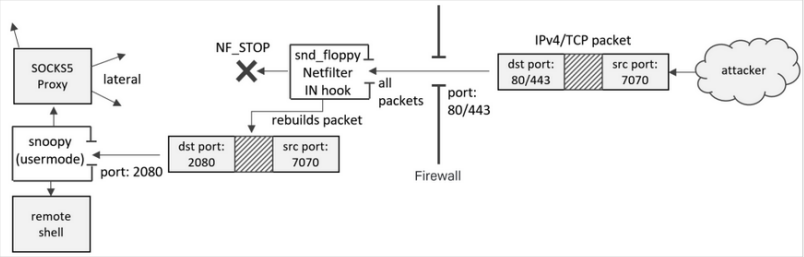
Explanation from SophosLab:

Explanation

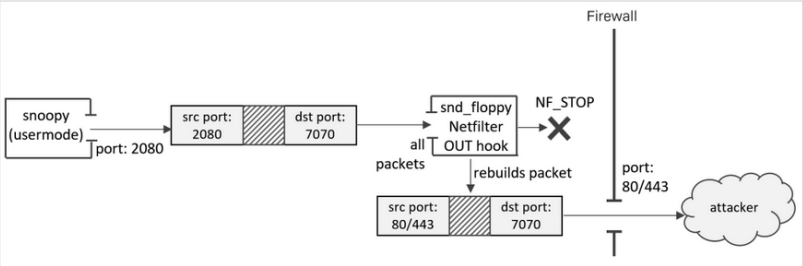
To trigger the payload (`snoopy`) activation, an attacker would send the following packet:



Next, the `snoopy` module would be accessed by the C2, using source port 7070 for TCP-based or 8080 for UDP-based control:



On the way back, the `NF_INET_LOCAL_OUT` hook handler rebuilds the packet again, to make sure its source port is restored back to the original port where the incoming packet was destined for. This way, the C2 traffic transparently flows through the port(s) allowed by AWS SGs:



No other Netfilter hooks within the chain, such as iptables INPUT/OUTPUT rules, will process the packet if the hook returns `NF_STOP` . This appears to be the purpose of the TCP command 2020: to bypass other Netfilter hooks.

https://news.sophos.com/en-us/2020/02/25/cloud-snooper-attack-bypasses-firewall-security-measures/