{"payload":{"allShortcutsEnabled":false,"fileTree":{"Pikabot":{"items":[{"name":"Images","path":"Pikabot/Images","contentType":"directory"},
{"name":"Pikabot Loader.md","path":"Pikabot/Pikabot Loader.md","contentType":"file"}],"totalCount":2},"":{"items":
[{"name":".obsidian","path":".obsidian","contentType":"directory"},{"name":"Pikabot","path":"Pikabot","contentType":"directory"},
{"name":"WSHRAT","path":"WSHRAT","contentType":"directory"}],"totalCount":3}},"fileTreeProcessingTime":4.83934,"foldersToFetch":
[],"reducedMotionEnabled":null,"repo":
{"id":675143377,"defaultBranch":"main","name":"MalwareAnalysisReports","ownerLogin":"VenzoV","currentUserCanPush":false,"isFork":false,"isEr
08-05T23:42:05.000Z","ownerAvatar":"https://avatars.githubusercontent.com/u/107503502?
v=4","public":true,"private":false,"isOrgOwned":false},"symbolsExpanded":false,"treeExpanded":true,"refInfo":
{"name":"main","listCacheKey":"v0:1691279081.0","canEdit":false,"refType":"branch","currentOid":"9b9f614920c7801fe030af46fef8e311dfa8df43"},
Loader.md","currentUser":null,"blob":{"rawLines":null,"stylingDirectives":null,"csv":null,"csvError":null,"dependabotInfo":
{"showConfigurationBanner":false,"configFilePath":null,"networkDependabotPath":"/VenzoV/MalwareAnalysisReports/network/updates","dismissCo
notice/dependabot_configuration_notice","configurationNoticeDismissed":null,"repoAlertsPath":"/VenzoV/MalwareAnalysisReports/security/depend
Loader.md","displayUrl":"https://github.com/VenzoV/MalwareAnalysisReports/blob/main/Pikabot/Pikabot%20Loader.md?
raw=true","headerInfo":{"blobSize":"16.5 KB","deleteInfo":{"deleteTooltip":"You must be signed in to make or propose changes"},"editInfo":
{"editTooltip":"You must be signed in to make or propose
changes"},"ghDesktopPath":"https://desktop.github.com","gitLfsPath":null,"onBranch":true,"shortPath":"ae0c3cb","siteNavLoginPath":"/login?
return_to=https%3A%2F%2Fgithub.com%2FVenzoV%2FMalwareAnalysisReports%2Fblob%2Fmain%2FPikabot%2FPikabot%2520Loader.md","is
[{"level":2,"text":"Sample Information","anchor":"sample-information","htmlText":"Sample Information"},
{"level":2,"text":"Introduction","anchor":"introduction","htmlText":"Introduction"},{"level":2,"text":"High level behavior","anchor":"high-level-
behavior","htmlText":"High level behavior"},{"level":2,"text":"PEB access","anchor":"peb-access","htmlText":"PEB access"},{"level":2,"text":"RC4
Inline Decryption","anchor":"rc4-inline-decryption","htmlText":"RC4 Inline Decryption"},{"level":2,"text":"Dynamic API
resolving","anchor":"dynamic-api-resolving","htmlText":"Dynamic API resolving"},{"level":2,"text":"Decrypted Strings","anchor":"decrypted-
strings","htmlText":"Decrypted Strings"},{"level":2,"text":"Anti Analysis","anchor":"anti-analysis","htmlText":"Anti Analysis"},{"level":2,"text":"Core
Module Extraction","anchor":"core-module-extraction","htmlText":"Core Module Extraction"},{"level":2,"text":"Core Module
Decryption","anchor":"core-module-decryption","htmlText":"Core Module Decryption"},{"level":2,"text":"Indirect Sycalls","anchor":"indirect-
sycalls","htmlText":"Indirect Sycalls"},{"level":2,"text":"References","anchor":"references","htmlText":"References"}],"lineInfo":
{"truncatedLoc":"484","truncatedSloc":"400"},"mode":"file"},"image":false,"isCodeownersFile":null,"isPlain":false,"isValidLegacyIssueTemplate":false
issue-and-pull-request-
templates","issueTemplate":null,"discussionTemplate":null,"language":"Markdown","languageID":222,"large":false,"loggedIn":false,"newDiscussionP
{"repoIsFork":null,"repoOwnedByCurrentUser":null,"requestFullPath":"/VenzoV/MalwareAnalysisReports/blob/main/Pikabot/Pikabot%20Loader.md"
{"dismissActionNoticePath":"/settings/dismiss-notice/publish_action_from_dockerfile","dismissStackNoticePath":"/settings/dismiss-
notice/publish_stack_from_file","releasePath":"/VenzoV/MalwareAnalysisReports/releases/new?
marketplace=true","showPublishActionBanner":false,"showPublishStackBanner":false},"rawBlobUrl":"https://github.com/VenzoV/MalwareAnalysisR

Sample Information

\n
Packed

\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n

SHA25	SHA1	M
DBDD22025131EEBE52EFC5FBE70E2E87723FF1934C808901BBB176F6130F23F6	66CBE1E120A28E812B265880406305E578560FFF	C

\n
Unpacked

\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n

SHA25	SHA1
75CCCAE5F0B726F23DAA6BE69DD7C5E8FCD25A41C06191B84EB00EF945E5F7FA	F269DDFFA7A741C879D712D7009A112402AAA0B2

\n

Introduction

\n

Pikabot is a relatively new malware. It has been analyzed and reversed before (see references).\nThis is my take and analysis on the updated version of the loader.\nEarlier during the year the sample was smaller and also used different string encryption.\nStack strings are still used, but now RC4 is used to decrypt them.

\n
Pikabot is divided into two modules, the loader and the core.\nIn this part we will take a look at the loader, which essentially has the job to load the core module which will be responsible for C2 communication.

\n

High level behavior

\n
So before going into the details the sample will perform the following actions, and during the analysis below I will show case the assembly, decompiler and debugger evidence.

\n

\n
- The malware uses a lot of junk code to try to hinder analysis.
\n
- Accesses the PEB to get handle to kernel32.dll to fetch LoadlLibraryA & GetProcAddress this will be used to dynamically load API.
\n
- Strings, in particular the API names passed to the API resolving function, are encrypted using RC4.
\n
- The core module is decrypted from png files located in the resource section.
\n
- Legitimate windows binary process is started, and core module is decrypted and injected into the process
\n
- Malware uses indirect syscalls
\n

\n

PEB access

\n
The first function to analyze is the one responsible for fetching LoadlLibraryA & GetProcAddress. To do this, the malware goes through the PEB to reach to get the base address of the kernel32.dll.

\n
PEB structure is accessed, and the the code walks through InLoadOrderModuleList twice and finally reaches the third entry which is always kernel32.dll.\nI have added references below to read more on PEB structure and how it can be used.

\n
\"image\"\n\"image\"

\n
Once the module base for kernel32.dll is found, the two API can now be fetched. Two hashes are used and passed to a function which will resolve the API.

\n

\n
- 0xB89FB14B - GetProcAddress
\n
- 0x7FA21D8F - LoadLibraryA
\n

\n
\"Image\"

\n

RC4 Inline Decryption

\n

Checking the sample, it uses RC4 to decrypt the strings. The malware uses \"legit\" strings for the keystream. We can receognize RC4 by typical 0x100 loops followed by another loop with XOR operation.\nBelow is what the code looks like.\nKeep in mind that the malware uses a lot of junk code between the two loops and final decryption loop.\nAlso, the decryption happens in line and is not a function.\nBoth factors make static analysis bothersome, and emulation also bothersome.\nThe decrypted strings can be fetched all at once using the debugger and some conditional break points. I will add the full list below.

\n

```
 do\n  {\n    v333[v3 + 24] = v3;\n    ++v3;\n  }\n  while ( v3 < 0x100 );\n  v4 = 0;\n  v338 = 0xF;\n  do\n  {\n    v5 = v333[v4 + 24];\n    a1 = (a1 + *(dbg_key_rc4 + (v4 & 0xF)) + v5);\n    v333[v4++ + 24] = v333[a1 + 24];\n    v333[a1 + 24] = v5;\n  }\n  while ( v4 < 0x100 );\n  v6 = v352;\n  jj = 0;\n  LOBYTE(v7) = 0;\n  for ( i = 0; i < 12; ++i )\n  {\n    v345 = (v7 + 1);\n    v9 = v333[v345 + 24];\n    v352 = -339480793 * v6;\n    jj = (v9 + jj);\n    v333[v345 + 24] = v333[jj + 24];\n    v333[jj + 24] = v9;\n    v7 = (v7 + 1);\n    v6 = v352;\n    v312[i] = *(&encrypted_blob[2] + i) ^ v333[(v9 + v333[v7 + 24]) + 24];\n  }\n
```

\n

Dynamic API resolving

\n
The First analyzed function and the RC4 encryption method, both are the main core of the APi resolving function.\nThe function accepts two arguements:

\n

\n

- DLL flag -> this is just a numerical value that tells the function in which DLL the API is; 1: Kernel32.dll, 2: User32.dll, 3: ntdll.dll
  \n
- API name in cleartext
  \n

\n
Whichever dll is used, the end result is always a jump to LABEL 88 seen below which peforms LoadLibraryA and GetProcAddress to retrieve the address of the API.

\n
\"Image\"\n\"Image\"\n\"Image\"

\n

Decrypted Strings

\n
Setting two conditional break points on the API resolving function it is possible to have the debugger decrypt all the strings and log them.

\n

\n

- First breakpoint is at the start of the funciton when the decrypted string passed as argument is saved to a variable
  \n
- Second breakpoint is on the return, so we can read ESP to also get the return address and so we know on IDA where this value needs to be added as comment and rename functions.
  \n

\n
These are the parameters used for the conditional break point, the addresses refer to how may binary was rebased in IDA.\n\"##APICALL {utf8(edx)}\" -> 0x6AB277B3\n\"##APICALL Address 0x{[esp]} \"-> 0x6AB27F86\n\"Image\"\n\"Image\"\n\"Image\"

\n
Output:

\n

```
##APICALL HeapAlloc\n##APICALL Address 0x6AB1ECFA\n##APICALL LoadLibraryA\n##APICALL Address 0x6AB1908A\n##APICALL
FreeLibrary\n##APICALL Address 0x6AB190E3\n##APICALL LoadLibraryA\n##APICALL Address 0x6AB190FD\n##APICALL FreeLibrary\n##APICALL
Address 0x6AB19246\n##APICALL LoadLibraryA\n##APICALL Address 0x6AB19260\n##APICALL FreeLibrary\n##APICALL Address
0x6AB192FA\n##APICALL LoadLibraryA\n##APICALL Address 0x6AB19314\n##APICALL LoadLibraryA\n##APICALL Address 0x6AB1950B\n##APICALL
LoadLibraryA\n##APICALL Address 0x6AB1959A\n##APICALL GetCurrentProcess\n##APICALL Address 0x6AB19D5C\n##APICALL
GetTickCount\n##APICALL Address 0x6AB2C823\n##APICALL GetCurrentThread\n##APICALL Address 0x6AB1A42C\n##APICALL
GetThreadContext\n##APICALL Address 0x6AB1A44B\n##APICALL FindResourceA\n##APICALL Address 0x6AB30F35\n##APICALL
LoadResource\n##APICALL Address 0x6AB30F55\n##APICALL LockResource\n##APICALL Address 0x6AB30F7B\n##APICALL
SizeofResource\n##APICALL Address 0x6AB30F9F\n##APICALL FreeResource\n##APICALL Address 0x6AB31F8A\n##APICALL
FindResourceA\n##APICALL Address 0x6AB30F35\n##APICALL LoadResource\n##APICALL Address 0x6AB30F55\n##APICALL
LockResource\n##APICALL Address 0x6AB30F7B\n##APICALL SizeofResource\n##APICALL Address 0x6AB30F9F\n##APICALL
FreeResource\n##APICALL Address 0x6AB31F8A\n##APICALL FindResourceA\n##APICALL Address 0x6AB30F35\n##APICALL
LoadResource\n##APICALL Address 0x6AB30F55\n##APICALL LockResource\n##APICALL Address 0x6AB30F7B\n##APICALL
SizeofResource\n##APICALL Address 0x6AB30F9F\n##APICALL FreeResource\n##APICALL Address 0x6AB31F8A\n##APICALL
FindResourceA\n##APICALL Address 0x6AB30F35\n##APICALL LoadResource\n##APICALL Address 0x6AB30F55\n##APICALL
LockResource\n##APICALL Address 0x6AB30F7B\n##APICALL SizeofResource\n##APICALL Address 0x6AB30F9F\n##APICALL
FreeResource\n##APICALL Address 0x6AB31F8A\n##APICALL FindResourceA\n##APICALL Address 0x6AB30F35\n##APICALL
LoadResource\n##APICALL Address 0x6AB30F55\n##APICALL LockResource\n##APICALL Address 0x6AB30F7B\n##APICALL
SizeofResource\n##APICALL Address 0x6AB30F9F\n##APICALL FreeResource\n##APICALL Address 0x6AB31F8A\n##APICALL
FindResourceA\n##APICALL Address 0x6AB30F35\n##APICALL LoadResource\n##APICALL Address 0x6AB30F55\n##APICALL
LockResource\n##APICALL Address 0x6AB30F7B\n##APICALL SizeofResource\n##APICALL Address 0x6AB30F9F\n##APICALL
FreeResource\n##APICALL Address 0x6AB31F8A\n##APICALL FindResourceA\n##APICALL Address 0x6AB30F35\n##APICALL
LoadResource\n##APICALL Address 0x6AB30F55\n##APICALL LockResource\n##APICALL Address 0x6AB30F7B\n##APICALL
SizeofResource\n##APICALL Address 0x6AB30F9F\n##APICALL FreeResource\n##APICALL Address 0x6AB31F8A\n##APICALL
FindResourceA\n##APICALL Address 0x6AB30F35\n##APICALL LoadResource\n##APICALL Address 0x6AB30F55\n##APICALL
LockResource\n##APICALL Address 0x6AB30F7B\n##APICALL SizeofResource\n##APICALL Address 0x6AB30F9F\n##APICALL
FreeResource\n##APICALL Address 0x6AB31F8A\n##APICALL FindResourceA\n##APICALL Address 0x6AB30F35\n##APICALL
LoadResource\n##APICALL Address 0x6AB30F55\n##APICALL LockResource\n##APICALL Address 0x6AB30F7B\n##APICALL
SizeofResource\n##APICALL Address 0x6AB30F9F\n##APICALL FreeResource\n##APICALL Address 0x6AB31F8A\n##APICALL
FindResourceA\n##APICALL Address 0x6AB30F35\n##APICALL LoadResource\n##APICALL Address 0x6AB30F55\n##APICALL
LockResource\n##APICALL Address 0x6AB30F7B\n##APICALL SizeofResource\n##APICALL Address 0x6AB30F9F\n##APICALL
FreeResource\n##APICALL Address 0x6AB31F8A\n##APICALL IsBadReadPtr\n##APICALL Address 0x6AB165DE\n##APICALL HeapAlloc\n##APICALL
Address 0x6AB1ECFA\n##APICALL HeapFree\n##APICALL Address 0x6AB1EFB4\n##APICALL InitializeProcThreadAttributeList\n##APICALL
Address 0x6AB24435\n##APICALL HeapAlloc\n##APICALL Address 0x6AB1ECFA\n##APICALL InitializeProcThreadAttributeList\n##APICALL
Address 0x6AB24474\n##APICALL UpdateProcThreadAttribute\n##APICALL Address 0x6AB24541\n##APICALL CreateProcessW\n##APICALL Address
0x6AB245D5\n##APICALL DeleteProcThreadAttributeList\n##APICALL Address 0x6AB246D4\n##APICALL HeapFree\n##APICALL Address
0x6AB1EFB4\n##APICALL HeapAlloc\n##APICALL Address 0x6AB1ECFA\n##APICALL HeapFree\n##APICALL Address 0x6AB1EFB4\n##APICALL
HeapFree\n##APICALL Address 0x6AB1EFB4\n##APICALL Sleep\n##APICALL Address 0x6AB13B32\n
```

\n

Anti Analysis

\n
There are three anti analysis functions I have identified so far.\nThe first two are simple, and basically check for DLLs associated with known sandbox/Vms.\nAgain here there DLL names are RC4 encrypted.\nI have made use of conditional breakpoints from x32 debug to extract from logging all the decrypted strings.

\n

```
mw_anti_vm():\n##DLL String Decrypted cmdvrt32.dll\n##Address 0x6AB1C945\n##DLL String Decrypted cmdvrt64.dll\n##Address
0x6AB1C95D\n##DLL String Decrypted dbghelp.dll\n##Address 0x6AB1C972\n##DLL String Decrypted cuckoomon.dll\n##Address
0x6AB1C987\n##DLL String Decrypted pstorec.dll\n##Address 0x6AB1C99C\n##DLL String Decrypted avghookx.dll\n##Address
0x6AB1C9B1\n##DLL String Decrypted avghooka.dll\n##Address 0x6AB1C9C6\n##DLL String Decrypted snxhk.dll\n##Address
0x6AB1C9DB\n##DLL String Decrypted api_log.dll\n##Address 0x6AB1C9F0\n##DLL String Decrypted dir_watch.dll\n##Address
0x6AB1CA05\n##DLL String Decrypted wpespy.dll\n##Address 0x6AB1CA1A\n\nmw_anti_vm1():\n##DLL Load Kernel32.dll\n##DLL Load
Kernel32.DLL\n##DLL Load networkexplorer.DLL\n##DLL Load NlsData0000.DLL\n##DLL Load NetProjW.DLL\n##DLL Load Ghofr.DLL\n##DLL Load
fg122.DLL\n
```

\n
\"Image\"

\n
The third anti analysis check, seems to perform some indirect syscalls by finding the address to functions for the ntdll.dll directly. I have not been able to understand this 100% yet.\nAlso, in this function the only resolved API which is also called is GetTickCount.\nThis is used to check the time since start of process, also typically used to check if process is running through a debugger.

\n
\"Image\"

\n

Core Module Extraction

\n
After the anti-analysis checks, the malware will proceed to fetch the core module from PNG files located in the resource section.\nEach png file has a section of data which needs to be combined with the others. As a delimiter the sample uses a 4 byte string as start of section.\nEach section is written to an allocated heap,thus combining them.\nIn total the sample uses 12 PNG files to store the core module.\nThe function called has the following arguments:

\n
\n
1. pointer to process
\n
2. PNG file name
\n
3. \"png\" string extension
\n
4. 4 byte delimiter string
\n
5. Heap offset
\n

\n
\"Image\"

\n
\"Image\"

\n
The function called above performs the following:

\n
\n
Call the API related to resource fetch - FindResourcaA, LoadResourceA, LockResourceA and finally SizeOfResource
\n

\n
\"Image\"

\n
\n
- Once the resource is loaded, the malware parses the PNG chunks and compare the name to the one passed as argument which is a 4 byte string. More on chunks check references, it is how PNG files are structured.
\n
- When the correct chunk is found, all the data from that chuck is written to the allocated heap but it is first xored.
\n

\n
\"Image\"

\n
\"Image\"\n\"Image\"

\n
\"Image\"

\n
\"Image\"

\n

Core Module Decryption

\n
Once the module is loaded into the heap memory, some more modifications need to be made to change it to a PE file. At the start of the DLL the key is RC4 decrypted:

\n

\n

1EmXwEpOYt6Cf8GyJVGXYUaqPnUapVrk

\n

\n

The call to decrypted has the key argument the heap with encrypted payload and new heap which will store the decrypted payload.\nThe decryption seems to be AES 256, but need to check further.\n\"Image\"

\n

\"Image\"

\n

\"Image\"

\n

Finally, once the core module is extracted the final function analyzed calls the following API and injects the code into \"SearchProtocolHost.exe\", which is spawned in a suspended state.

\n

\n

- InitializeProcThreadAttributeList
  \n
- UpdateProcThreadAttribute
  \n
- CreateProcessW
  \n
- DeleteProcThreadAttribute
  \n

\n

\"Image\"

\n

\"Image\"

\n

\"Image\"

\n

Checking process hacker we can observe memory being allocated in the process and then the core payload is written here.

\n

\"Image\"

\n

\"Image\"

\n

Indirect Sycalls

\n

As mentioned above, the sample makes use of indirect syscalls. The calls made can be referenced on the eax register by their IDs. I expect NtAllocateVirtualMemory and NtWriteVirtualMemory to be called after process creation.\nWe can run the code until CreateProcessW is called and then set two breakpoints on the wrapper function for the indirect syscalls. Once inside the syscall id is processed and then called.\nWe observe 0x18 and 0x3a being loaded to eax which correspond to the funcitons we expect. Soon after these calls the memory is allocated and the corepayload is written to further evidence the usage of these indirect syscall.

\n

Special thanks to @xleandr0 for helping to understand this.

\n

Following the code seen in IDA:\n\"Image\"

\n

Following the debugger view of the syscall ID:

\n
[\"Image\"]

\n
[\"Image\"]

\n
Using conditional breakpoints as above, we can print out all the syscall IDs used by the malware, to see what API are used.\nI dumped it all out, but there are a lot of repetitions and can't paste them all here, but the following are the API called without counting duplicates:

\n

```
##SyscallID 19 -> NtQueryInformationProcess -> NtQueryInformationProcess\nINT3 breakpoint at pika.6AB139CF!\n##SyscallID 19 ->
NtQueryInformationProcess\n##SyscallID 3F -> NtReadVirtualMemory \n##SyscallID 2A -> NtUnmapViewOfSection\n##SyscallID 18 ->
NtAllocateVirtualMemory\n##SyscallID 3A -> NtWriteVirtualMemory\n##SyscallID 3F -> NtReadVirtualMemory\n##SyscallID F3 ->
NtGetCurrentProcessorNumber\n##SyscallID 52 -> NtResumeThread\n
```

\n
The breakpoint after NtQueryInformationProcess checks if eax value is 1 or 0. If 1 the process ends, so manually changing the value to 0 avoids the check and continues execution.\nMost of the calls that generate volume are:

\n
\n
- NtReadVirtualMemory
\n
- NtAllocateVirtualMemory
\n
- NtWriteVirtualMemory
\n

\n
We can see the final Native API is NtResumeThread which makes sense, since the execution will continue from the injected code.

\n

References

\n

\n
- https://d01a.github.io/pikabot/
\n
- https://research.openanalysis.net/pikabot/debugging/string%20decryption/emulation/memulator/2023/11/19/new-pikabot-strings.html
\n
- https://www.zscaler.com/blogs/security-research/technical-analysis-pikabot
\n
- https://research.openanalysis.net/pikabot/debugging/string%20decryption/2023/11/12/new-pikabot.html
\n
- https://www.ired.team/offensive-security/code-injection-process-injection/finding-kernel32-base-and-function-addresses-in-shellcode
\n
- http://undocumented.ntinternals.net/index.html?page=UserMode%2FUndocumented%20Functions%2FNT%20Objects%2FProcess%2FPEB.html
\n
- https://www.geoffchappell.com/studies/windows/km/ntoskrnl/inc/api/pebteb/peb/index.htm
\n
- https://www.w3.org/TR/PNG-Chunks.html
\n
- https://0xk4n3ki.github.io/posts/Heavens-Gate-Technique/
\n
- https://www.gosecure.net/blog/2021/12/03/trickbot-leverages-zoom-work-from-home-interview-malspam-heavens-gate-and-spamhaus/
\n
- https://j00ru.vexillium.org/syscalls/nt/64/
\n
- https://twitter.com/leandrofr0es
\n

\n","renderedFileInfo":null,"shortPath":null,"tabSize":8,"topBannersInfo":
{"overridingGlobalFundingFile":false,"globalPreferredFundingPath":null,"repoOwner":"VenzoV","repoName":"MalwareAnalysisReports","showInvali
cloning-and-archiving-repositories/creating-a-repository-on-github/about-citation-
files","showDependabotConfigurationBanner":false,"actionsOnboardingTip":null},"truncated":false,"viewable":true,"workflowRedirectUrl":null,"symb
{"timed_out":false,"not_analyzed":false,"symbols":[{"name":"Sample
Information","kind":"section_2","ident_start":3,"ident_end":21,"extent_start":0,"extent_end":430,"fully_qualified_name":"Sample
Information","ident_utf16":{"start":{"line_number":0,"utf16_col":3},"end":{"line_number":0,"utf16_col":21}},"extent_utf16":{"start":
{"line_number":0,"utf16_col":0},"end":{"line_number":14,"utf16_col":0}}},
{"name":"Introduction","kind":"section_2","ident_start":433,"ident_end":445,"extent_start":430,"extent_end":983,"fully_qualified_name":"Introduction
{"start":{"line_number":14,"utf16_col":3},"end":{"line_number":14,"utf16_col":15}},"extent_utf16":{"start":{"line_number":14,"utf16_col":0},"end":
{"line_number":25,"utf16_col":0}}},{"name":"High level
behavior","kind":"section_2","ident_start":986,"ident_end":1005,"extent_start":983,"extent_end":1704,"fully_qualified_name":"High level
behavior","ident_utf16":{"start":{"line_number":25,"utf16_col":3},"end":{"line_number":25,"utf16_col":22}},"extent_utf16":{"start":
{"line_number":25,"utf16_col":0},"end":{"line_number":37,"utf16_col":0}}},{"name":"PEB
access","kind":"section_2","ident_start":1707,"ident_end":1717,"extent_start":1704,"extent_end":2457,"fully_qualified_name":"PEB
access","ident_utf16":{"start":{"line_number":37,"utf16_col":3},"end":{"line_number":37,"utf16_col":13}},"extent_utf16":{"start":
{"line_number":37,"utf16_col":0},"end":{"line_number":54,"utf16_col":0}}},{"name":"RC4 Inline
Decryption","kind":"section_2","ident_start":2460,"ident_end":2481,"extent_start":2457,"extent_end":3713,"fully_qualified_name":"RC4 Inline
Decryption","ident_utf16":{"start":{"line_number":54,"utf16_col":3},"end":{"line_number":54,"utf16_col":24}},"extent_utf16":{"start":
{"line_number":54,"utf16_col":0},"end":{"line_number":97,"utf16_col":0}}},{"name":"Dynamic API
resolving","kind":"section_2","ident_start":3716,"ident_end":3737,"extent_start":3713,"extent_end":4302,"fully_qualified_name":"Dynamic API
resolving","ident_utf16":{"start":{"line_number":97,"utf16_col":3},"end":{"line_number":97,"utf16_col":24}},"extent_utf16":{"start":
{"line_number":97,"utf16_col":0},"end":{"line_number":110,"utf16_col":0}}},{"name":"Decrypted
Strings","kind":"section_2","ident_start":4305,"ident_end":4322,"extent_start":4302,"extent_end":9735,"fully_qualified_name":"Decrypted
Strings","ident_utf16":{"start":{"line_number":110,"utf16_col":3},"end":{"line_number":110,"utf16_col":20}},"extent_utf16":{"start":
{"line_number":110,"utf16_col":0},"end":{"line_number":304,"utf16_col":0}}},{"name":"Anti
Analysis","kind":"section_2","ident_start":9738,"ident_end":9751,"extent_start":9735,"extent_end":11356,"fully_qualified_name":"Anti
Analysis","ident_utf16":{"start":{"line_number":304,"utf16_col":3},"end":{"line_number":304,"utf16_col":16}},"extent_utf16":{"start":
{"line_number":304,"utf16_col":0},"end":{"line_number":355,"utf16_col":0}}},{"name":"Core Module
Extraction","kind":"section_2","ident_start":11359,"ident_end":11381,"extent_start":11356,"extent_end":12696,"fully_qualified_name":"Core
Module Extraction","ident_utf16":{"start":{"line_number":355,"utf16_col":3},"end":{"line_number":355,"utf16_col":25}},"extent_utf16":{"start":
{"line_number":355,"utf16_col":0},"end":{"line_number":392,"utf16_col":0}}},{"name":"Core Module
Decryption","kind":"section_2","ident_start":12699,"ident_end":12721,"extent_start":12696,"extent_end":13783,"fully_qualified_name":"Core
Module Decryption","ident_utf16":{"start":{"line_number":392,"utf16_col":3},"end":{"line_number":392,"utf16_col":25}},"extent_utf16":{"start":
{"line_number":392,"utf16_col":0},"end":{"line_number":426,"utf16_col":0}}},{"name":"Indirect
Sycalls","kind":"section_2","ident_start":13786,"ident_end":13802,"extent_start":13783,"extent_end":15856,"fully_qualified_name":"Indirect
Sycalls","ident_utf16":{"start":{"line_number":426,"utf16_col":3},"end":{"line_number":426,"utf16_col":19}},"extent_utf16":{"start":
{"line_number":426,"utf16_col":0},"end":{"line_number":471,"utf16_col":0}}},
{"name":"References","kind":"section_2","ident_start":15859,"ident_end":15869,"extent_start":15856,"extent_end":16868,"fully_qualified_name":"R
{"start":{"line_number":471,"utf16_col":3},"end":{"line_number":471,"utf16_col":13}},"extent_utf16":{"start":
{"line_number":471,"utf16_col":0},"end":{"line_number":484,"utf16_col":0}}}]}},"copilotInfo":null,"copilotAccessAllowed":false,"csrf_tokens":
{"/VenzoV/MalwareAnalysisReports/branches":
{"post":"U7qWJvySv7aUnGgew85pYvlZv0h48PoiT5pjgaBPVB09nGbRMfihSedxTnui1K_fizv1NIBp67IXEOHJITTXMQ"},"/repos/preferences":
{"post":"Cdo7QKhBCbsoQHcIg_lYfoCXpsEYQLO40OWHoLjYZuw8OsDm-2nuvtdDFo2B7-TEj3mSO-
yRorO3_EuEFwKaug"}}},"title":"MalwareAnalysisReports/Pikabot/Pikabot Loader.md at main · VenzoV/MalwareAnalysisReports"}