CALIFORNIA STATE UNIVERSITY

FULLERTON

Project 4: *Depends22_x64*

Malware Analysis CPSC 458-03, Fall 2024

Group 12

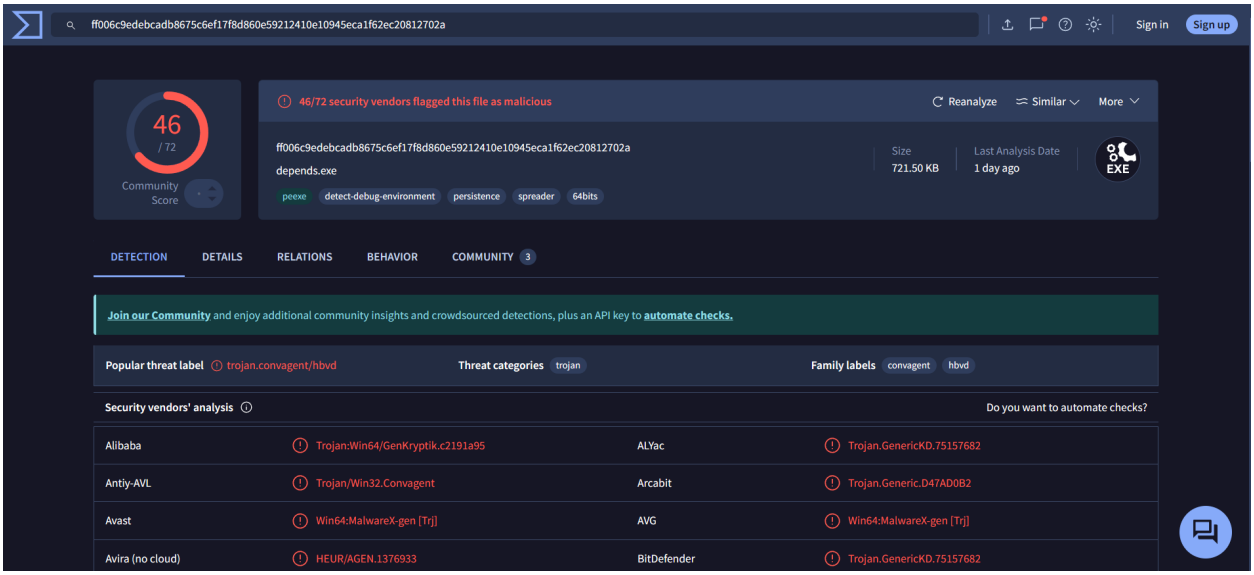
Allen Dai, Paul Le, Terry Ma, Zohair Mamdani, Wayne Muse, Zakariye Samatar

Basic Static Analysis

Virustotal

We got the hash of depends.exe and submitted it to virus total for inspection. Multiple security vendors flag Depends22_x64 as trojan malware. We also see that the malware imports KERENAL32.dll and msvcrt.dll

```
PS C:\Users\IEUser\Documents\project4\depends22_x64> Get-FileHash -Path C:\Users\IEUser\Documents\project4\depends22_x64\depends.exe

Algorithm       Hash                                                                   Path
---------       ----                                                                   ----
SHA256          FF006C9EDEBCADB8675C6EF17F8D860E59212410E10945ECA1F62EC20812702A       C:\Users\IEUser\Documents\pro...

PS C:\Users\IEUser\Documents\project4\depends22_x64> Get-FileHash -Path C:\Users\IEUser\Documents\project4\depends22_x64\depends.dll

Algorithm       Hash                                                                   Path
---------       ----                                                                   ----
SHA256          9A72A304EC596F1A62C996B3DA54B31505B150E274002D8621A9CEF93020382A       C:\Users\IEUser\Documents\project4\depends22_x64\dep...

PS C:\Users\IEUser\Documents\project4\depends22_x64> Get-FileHash -Path C:\Users\IEUser\Documents\project4\depends22_x64\depends.chm

Algorithm       Hash                                                                   Path
---------       ----                                                                   ----
SHA256          E5A4E001FBFE731B5D8B9D2046C57FA1786599364366704A800D59239D0C064D       C:\Users\IEUser\Documents\project4\depends22_x64\dep...

PS C:\Users\IEUser\Documents\project4\depends22_x64>
```



Basic properties

MD5	c638a62bd2fe7b52788183edbc85d335
SHA-1	a67299ab73bc6d952a52481bf640167f2f3f5750
SHA-256	ff006c9edebcadb8675c6ef17f8d860e59212410e10945eca1f62ec20812702a
Vhash	0750c76d156d05551c0d1az383a=z
Authentihash	e3bff51a41ac7a5aafe87361e88472e7cb8aaea40778baad1eca8924559109fd
Imphash	1e0eea6d40f280d0c7563b489ebe8da2
SSDEEP	12288:CUSwoH3X8DKGLf5WewEouWDl44dvBZKWdGrH12:/SDH3X6KGtwEot85Tf
TLSH	T195F48C26BB78407BD0A7D1B5C5928BA5F6717C410B3052CB276643293F2B6F05E7836E
File type	Win32 EXE executable windows win32 pe peexe
Magic	PE32+ executable (GUI) x86-64 (stripped to external PDB), for MS Windows
TrID	Windows Control Panel Item (generic) (53.4%) | Microsoft Visual C++ compiled executable (generic) (15.3%) | Win64 Executable (generic) (9.7%) | DOS Borland compile…
DetectItEasy	PE64 | Compiler: MinGW (GCC: (MinGW-W64 x86_64-posix-seh, built by Brecht Sanders) 11.1.0) | Compiler: Nim | Linker: GNU linker ld (GNU Binutils) (2.36) [GUI64]
Magika	PEBIN
File size	721.50 KB (738816 bytes)

History

Creation Time	2024-12-10 11:38:55 UTC
First Submission	2024-12-15 00:04:18 UTC
Last Submission	2024-12-21 02:10:58 UTC
Last Analysis	2024-12-19 22:01:33 UTC

Names

depends.exe

depends.exe.malz

Portable Executable Info

Header

Target Machine	x64
Compilation Timestamp	2024-12-10 11:38:55 UTC
Entry Point	4389
Contained Sections	12

Sections

Name	Virtual Address	Virtual Size	Raw Size	Entropy	MD5	Chi2
.text	4096	128664	129024	6.08	a60a614d25be4a23fa4a5185527ef43a	1102994.62
.data	135168	1680	2048	1.57	04e8b4655c53e32cfe84588f73de66a7	346935
.rdata	139264	581984	582144	6.73	693cda8391596ea6b485febb3c6062a1	6018519.5
.eh_fram	724992	4	512	0	bf619eac0cdf3f68d496ea9344137e8b	130560
.pdata	729088	8412	8704	5.29	6411c74add09b74ad878d56a77abd067	323015.53

Imports

+ KERNEL32.dll
+ msvcrt.dll

Contained Resources By Type

RT_MANIFEST	1

Contained Resources By Language

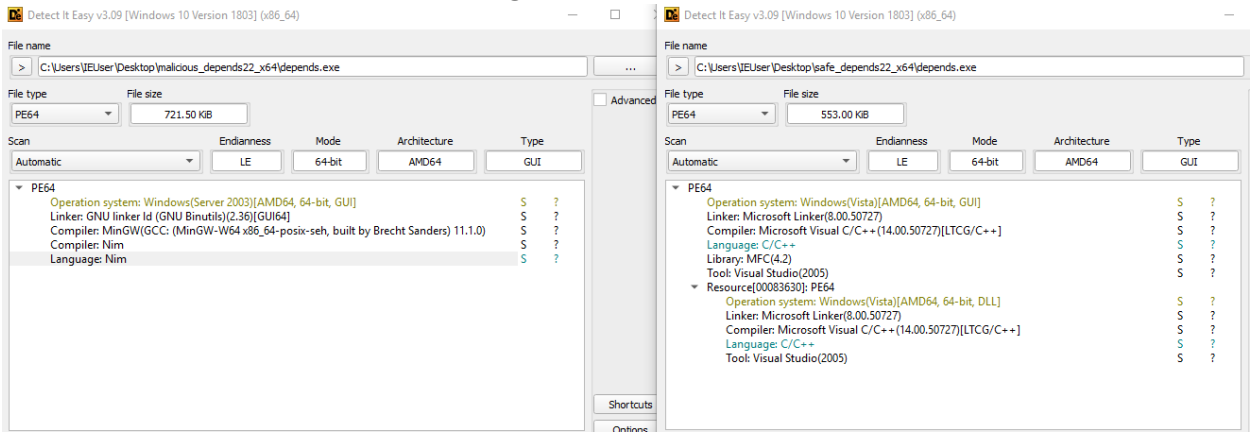
ENGLISH US	1

Detect-it-Easy

Detect-It-Easy (DIE) is used to analyze the suspicious depends.exe (in the malicious folder) as well as a depends.exe (in the safe folder) from the dependencywalker.com website. From the start, we can see a clear difference between the two: the suspicious executable is written and compiled in Nim language using MinGW compiler while the real one is written in C/C++ and compiled using Microsoft Visual. Of course in the context of this being an alternative, it could be that the program was wholly rewritten from the ground up, but it is more likely to be posing as depends22 to trick users. Additionally, the file size of the malicious file is around 30% more than the safe file from the dependencywalker.com website, which suggests that the malicious depends.exe is doing more than just its intended functions.

- According to a Wikipedia article (https://en.wikipedia.org/wiki/Nim_(programming_language)), "Nim is designed to be 'efficient, expressive, and elegant … by providing several features such as compile time code generation, … , a foreign function interface (FFI) with C, C++, Objective-C, and JavaScript, and supporting compiling to those same languages as intermediate representations". Basically, Nim can compile to C/C++ and can alter or implement code while compiling.
- According to this article (https://www.proofpoint.com/uk/blog/threat-insight/nimzaloader-ta800s-new-initial-access-malware), "Malware developers may choose to use a rare programming language to avoid detection, as reverse engineers may not be familiar with Nim's implementation, or focused on developing detection for it, and therefore tools and sandboxes may struggle to analyze samples of it". It seems that Nim is a rarely used, though fairly aged language (released in 2006, currently maintained). This means it is very likely that the malicious depends.exe is using Nim to obfuscate its code to probably avoid disassembled analysis as well as other anti-analysis such as what we can see with the Compiler and Language being analyzed as Nim.

File Info: Malicious (left) and Safe (right) depends.exe



Next we look at the imports and find only 2 dlls: KERNEL32.dll and msvcrt.dll. This is suspicious because this is relatively few dlls compared to the many other executables we've analyzed in the past (both malicious and safe).

Malicious depends.exe

#	OriginalFirstThunk	TimeDateStamp	ForwarderChain	Name	FirstThunk	Hash	Name
0	000b803c	00000000	00000000	000b8f6c	000b83dc	11c62e69	KERNEL32.dll
1	000b8204	00000000	00000000	000b9064	000b85a4	af446bfa	msvcrt.dll

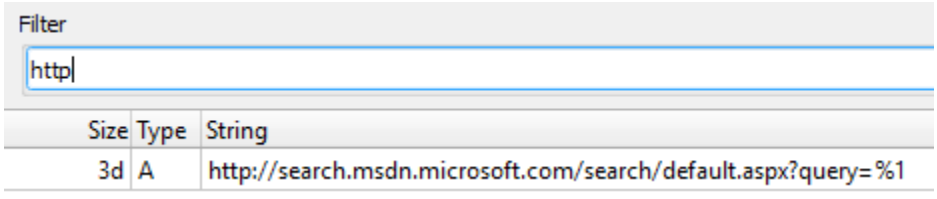
So the safe depends.exe is analyzed and we find 8 dlls. This means that the malware author likely used Nim language to hide, likely during compile time, the dlls used, meaning we won't be able to easily analyze the executable with only static analysis.

Safe depends.exe

#	OriginalFirstThunk	TimeDateStamp	ForwarderChain	Name	FirstThunk	Hash	Name
0	00048c50	00000000	00000000	0004a228	00001000	dd724fa1	ADVAPI32.dll
1	00048d58	00000000	00000000	0004a8d0	00001108	a0a9d50a	KERNEL32.dll
2	00048cd8	00000000	00000000	0004a9d4	00001088	6d3ae172	GDI32.dll
3	00049e00	00000000	00000000	0004ace8	000021b0	0a38c621	USER32.dll
4	00049088	00000000	00000000	0004acf4	00001438	6e6b2a6f	MFC42.dll
5	00049f90	00000000	00000000	0004aef4	00002340	25b5bbbe	msvcrt.dll
6	00048cb8	00000000	00000000	0004b00a	00001068	07a0eab1	COMDLG32.dll
7	00048ca8	00000000	00000000	0004b02a	00001058	a2bcbb94	COMCTL32.dll
8	00049dc8	00000000	00000000	0004b0b4	00002178	0e801c98	SHELL32.dll

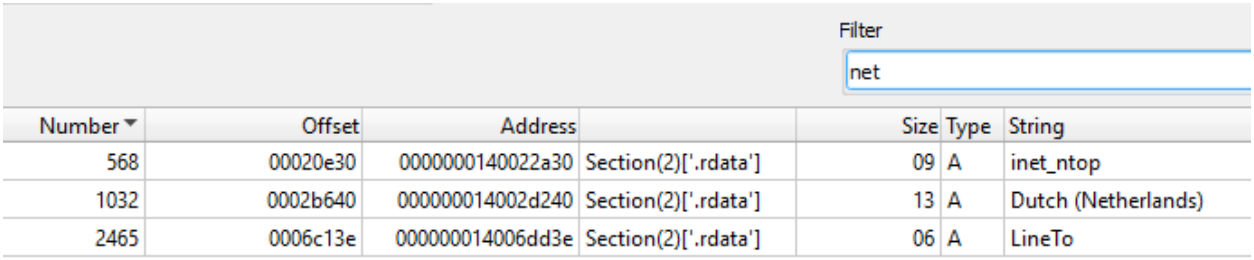
For strings, it seems that the malicious depends.exe is not packed and obfuscated. To start off, some common indicators are searched for.

First is "http" which returns one string with a microsoft.com link. The safe depends.exe also has this link, so it is not an indicator and likely is not related to the malware functions.



Next is “net”. We get 3 strings returned from this search. However, the safe depends.exe does not have any strings with “inet_ntop”. According to this article (https://learn.microsoft.com/en-us/windows/win32/api/ws2tcpip/nf-ws2tcpip-inet_ntop), inet_ntop is used to “[convert] an IPv4 or IPv6 Internet network address into a string in Internet standard format. The ANSI version of this function is inet_ntop”. This is a clear indication that the malicious depends.exe is doing something related to an IP address, likely the user’s.

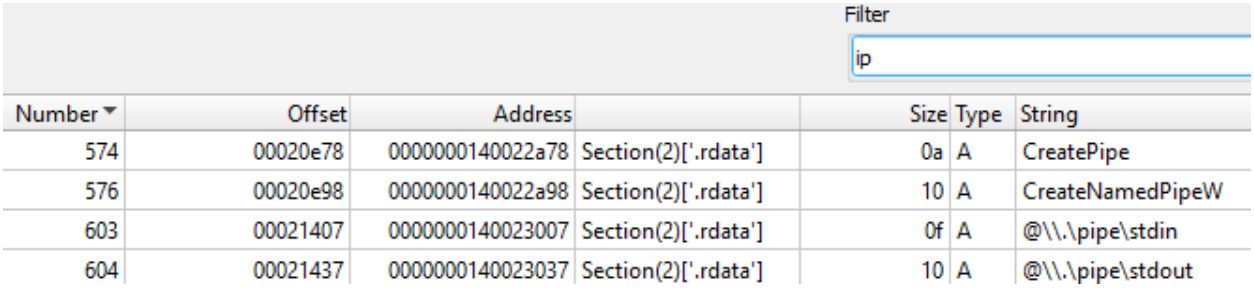
- inet_ntop function is from ws2tcpip.h from WS2.dll. This means that the malicious depends.exe is obfuscating its libraries through using Nim language and compiler and suggests that there are more network-related libraries hidden.
- It is most likely that the malicious depends.exe is performing network-related actions.


Filter: net

Number	Offset	Address		Size	Type	String
568	00020e30	0000000140022a30	Section(2)['.rdata']	09	A	inet_ntop
1032	0002b640	000000014002d240	Section(2)['.rdata']	13	A	Dutch (Netherlands)
2465	0006c13e	000000014006dd3e	Section(2)['.rdata']	06	A	LineTo

“ip” is then filtered, displaying similar strings to the safe depends.exe, except for five of the strings. According to this article (https://learn.microsoft.com/en-us/windows/win32/api/namedpipeapi/nf-namedpipeapi-createpipe), CreatePipe function “[c]reates an anonymous pipe, and returns handles to the read and write ends of the pipe”. According to this article (https://learn.microsoft.com/en-us/windows/win32/api/namedpipeapi/nf-namedpipeapi-createnamedpipew), CreateNamedPipeW function “[c]reates an instance of a named pipe and returns a handle for subsequent pipe operations”.

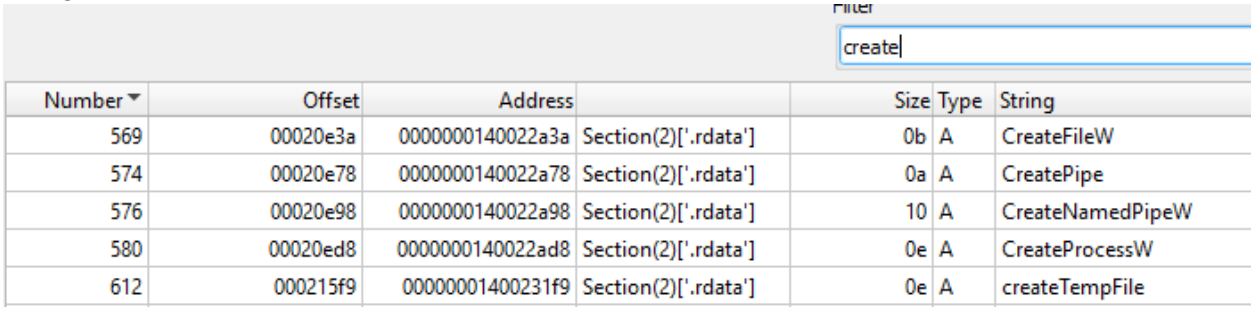
- The CreatePipe and CreateNamedPipeW functions are both from namedpipeapi.h from WIN32.dll. This means the malicious depends.exe is performing ReadFile, WriteFile, and potentially CreateFile and other functions.


Filter: ip

Number	Offset	Address		Size	Type	String
574	00020e78	0000000140022a78	Section(2)['.rdata']	0a	A	CreatePipe
576	00020e98	0000000140022a98	Section(2)['.rdata']	10	A	CreateNamedPipeW
603	00021407	0000000140023007	Section(2)['.rdata']	0f	A	@\\.\pipe\stdin
604	00021437	0000000140023037	Section(2)['.rdata']	10	A	@\\.\pipe\stdout

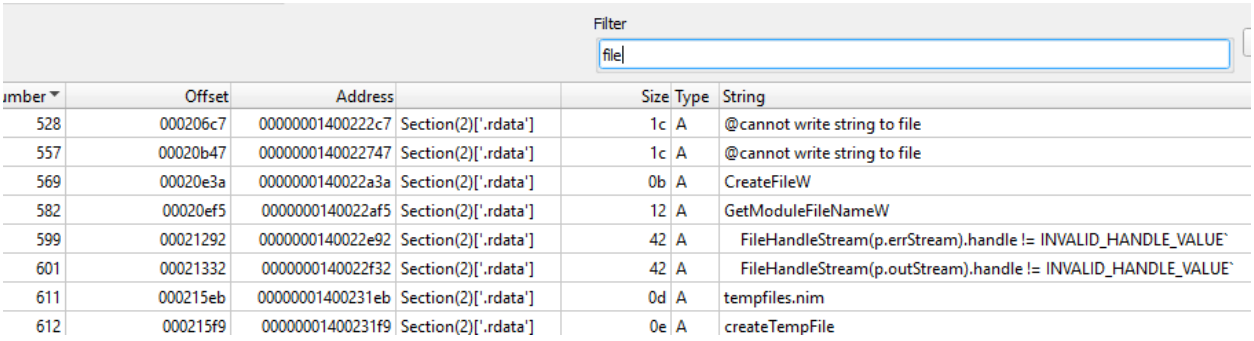
“create” is then filtered and we see CreateFileW, CreateProcessW, and createTempFile are strings not seen in the safe depends.exe. The focus will be on CreateProcessW, a function from processthreadsapi.h from WIN32.dll. According to this article (https://learn.microsoft.com/en-

us/windows/win32/api/processthreadsapi/nf-processthreadsapi-createprocessw), the function "[c]reates a new process and its primary thread. The new process runs in the security context of the calling process".
When searching for createTempFile, it is found to be a function from Java and not C/C++, though C/C++ will use CreateFileW.


Filter: create

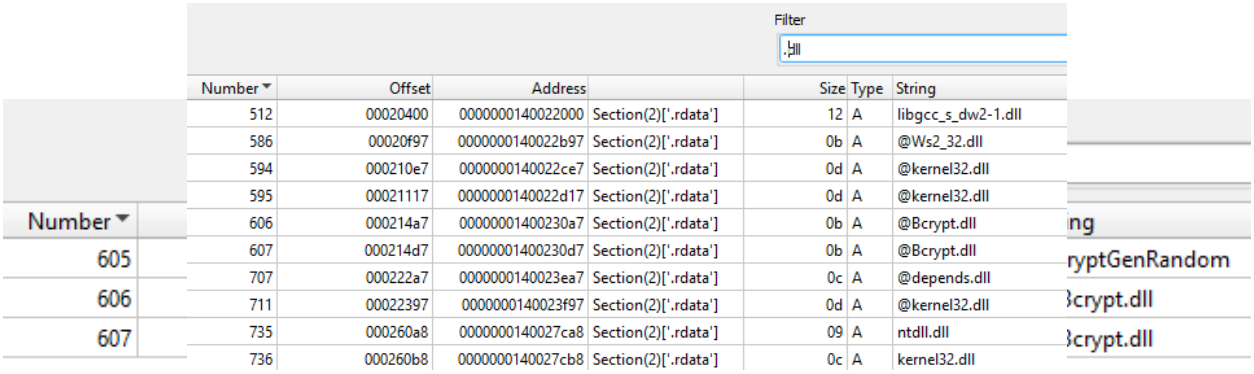
Number	Offset	Address		Size	Type	String
569	00020e3a	0000000140022a3a	Section(2)['.rdata']	0b	A	CreateFileW
574	00020e78	0000000140022a78	Section(2)['.rdata']	0a	A	CreatePipe
576	00020e98	0000000140022a98	Section(2)['.rdata']	10	A	CreateNamedPipeW
580	00020ed8	0000000140022ad8	Section(2)['.rdata']	0e	A	CreateProcessW
612	000215f9	00000001400231f9	Section(2)['.rdata']	0e	A	createTempFile

"file" is filtered. We can see some strings not found in the safe depends.exe. It looks like the malicious depends.exe is creating a temporary file somewhere as expressed by "tempfiles.nim".


Filter: file

ımber	Offset	Address		Size	Type	String
528	000206c7	00000001400222c7	Section(2)['.rdata']	1c	A	@cannot write string to file
557	00020b47	0000000140022747	Section(2)['.rdata']	1c	A	@cannot write string to file
569	00020e3a	0000000140022a3a	Section(2)['.rdata']	0b	A	CreateFileW
582	00020ef5	0000000140022af5	Section(2)['.rdata']	12	A	GetModuleFileNameW
599	00021292	0000000140022e92	Section(2)['.rdata']	42	A	FileHandleStream(p.errStream).handle != INVALID_HANDLE_VALUE`
601	00021332	0000000140022f32	Section(2)['.rdata']	42	A	FileHandleStream(p.outStream).handle != INVALID_HANDLE_VALUE`
611	000215eb	00000001400231eb	Section(2)['.rdata']	0d	A	tempfiles.nim
612	000215f9	00000001400231f9	Section(2)['.rdata']	0e	A	createTempFile

Considering that the executable is unpacked, ".dll" is filtered and we find some dlls not found in the safe depends.exe: libgcc_s_dw2-1.dll and @Bcrypt.dll. Using the Wikipedia article (https://en.wikipedia.org/wiki/Bcrypt) and this security policy PDF (https://csrc.nist.gov/csrc/media/projects/cryptographic-module-validation-program/documents/security-policies/140sp892.pdf) as references, we find that the Bcrypt.dll is related to cryptography and password hashing. When we filter for "bcrypt", we also get the string "BCryptGenRandom" which "fills a buffer with random bytes".


Filter: .dll

Number	Offset	Address		Size	Type	String
512	00020400	0000000140022000	Section(2)['.rdata']	12	A	libgcc_s_dw2-1.dll
586	00020f97	0000000140022b97	Section(2)['.rdata']	0b	A	@Ws2_32.dll
594	000210e7	0000000140022ce7	Section(2)['.rdata']	0d	A	@kernel32.dll
595	00021117	0000000140022d17	Section(2)['.rdata']	0d	A	@kernel32.dll
606	000214a7	00000001400230a7	Section(2)['.rdata']	0b	A	@Bcrypt.dll
607	000214d7	00000001400230d7	Section(2)['.rdata']	0b	A	@Bcrypt.dll
707	000222a7	0000000140023ea7	Section(2)['.rdata']	0c	A	@depends.dll
711	00022397	0000000140023f97	Section(2)['.rdata']	0d	A	@kernel32.dll
735	000260a8	0000000140027ca8	Section(2)['.rdata']	09	A	ntdll.dll
736	000260b8	0000000140027cb8	Section(2)['.rdata']	0c	A	kernel32.dll

Number	ng
605	ryptGenRandom
606	3crypt.dll
607	3crypt.dll

"Get" is filtered. The notable strings not found in the safe depends.exe are "GetConsoleOutputCP", "GetConsoleCP", "GetStdHandle", "GetTempPathW",

“DispGetIDsOfNames”, and “@Incompatible architecture (between injector and target process).”.

- GetConsoleOutputCP
  - https://learn.microsoft.com/en-us/windows/console/getconsoleoutputcp
  - “Retrieves the output code page used by the console associated with the calling process. A console uses its output code page to translate the character values written by the various output functions into the images displayed in the console window.”
  - Related: https://learn.microsoft.com/en-us/windows/win32/intl/code-page-identifiers
- GetConsoleCP
  - https://learn.microsoft.com/en-us/windows/console/getconsolecp
  - “Retrieves the input code page used by the console associated with the calling process. A console uses its input code page to translate keyboard input into the corresponding character value.”
- GetStdHandle
  - https://learn.microsoft.com/en-us/windows/console/getstdhandle
  - “Retrieves a handle to the specified standard device (standard input, standard output, or standard error).”
  - Malicious depends.exe could potentially be reading input/output buffers of the console and storing it in a temporary file.
- GetTempPathW
  - https://learn.microsoft.com/en-us/windows/win32/api/fileapi/nf-fileapi-gettemppathw
  - “Retrieves the path of the directory designated for temporary files.”
  - Related to tempfile of malicious depends.exe.
- DispGetIDsOfNames
  - https://learn.microsoft.com/en-us/windows/win32/api/oleauto/nf-oleauto-dispgetidsofnames
  - “Low-level helper for Invoke that provides machine independence for customized Invoke.”
- @Incompatible architecture (between injector and target process).
  - Simple internet search suggests this is an error string related to the process of DLL-injection.

Filter: Get

Number	Offset	Address		Size	Type	String
524	00020660	0000000140022260	Section(2)['.rdata']	12	A	GetConsoleOutputCP
525	00020673	0000000140022273	Section(2)['.rdata']	0c	A	GetConsoleCP
570	00020e46	0000000140022a46	Section(2)['.rdata']	0c	A	GetLastError
577	00020ea9	0000000140022aa9	Section(2)['.rdata']	11	A	GetCurrentProcess
579	00020ecb	0000000140022acb	Section(2)['.rdata']	0c	A	GetStdHandle
582	00020ef5	0000000140022af5	Section(2)['.rdata']	12	A	GetModuleFileNameW
592	000210a0	0000000140022ca0	Section(2)['.rdata']	0c	A	GetTempPathW
625	0002181c	000000014002341c	Section(2)['.rdata']	10	A	GetModuleHandleA
626	0002182d	000000014002342d	Section(2)['.rdata']	0e	A	GetProcAddress
630	0002186a	000000014002346a	Section(2)['.rdata']	0c	A	GetLastError
632	00021886	0000000140023486	Section(2)['.rdata']	11	A	GetCurrentProcess
638	00021930	0000000140023530	Section(2)['.rdata']	11	A	DispGetIDsOfNames
708	000222c7	0000000140023ec7	Section(2)['.rdata']	41	A	@Incompatible architecture (between injector and target process).
734	00026098	0000000140027c98	Section(2)['.rdata']	0e	A	GetProcAddress
764	000268f8	00000001400284f8	Section(2)['.rdata']	14	A	GetModuleFileNameExA

PEStudio

PEStudio reveals several suspicious imports from KERNEL32.dll and msvcrt.dll in the executable. These imports indicate the malware is designed for process manipulation and memory injection, enabling it to execute malicious actions stealthily.

- Process Injection: Functions like OpenProcess, VirtualAlloc, VirtualProtect, and VirtualQuery are used for injecting code into processes by manipulating memory.
- Process Discovery: Functions such as GetCurrentProcessId, GetCurrentProcess, and GetCurrentThreadId help the malware identify and interact with the current process and thread.

indicator (21)	detail	level
imports > flag	AddVectoredExceptionHandler | GetCurrentProcess | GetCurrentProcessI...	+++++
file > signature | tooling	Nim Compiler	++
strings > flag	count: 80	++
section > file	signature: chtml, offset: 0x0007D780, size: 200320 bytes	++
sections > virtualized	name: .bss	++
string > url-pattern	http://search.msdn.microsoft.com/search/default.aspx?query=%1	++
file > entropy	6.668	+
file > sha256	FF006C9EDEBCADB8675C6EF17F8D860E59212410E10945ECA1F62EC2081...	+
file > size	738816 bytes	+
file > type	executable, 64-bit, GUI	+
virustotal > status	The server name or address could not be resolved	+
compiler > stamp	Tue Dec 10 11:38:55 2024	+
resource > items	count: 1, size: 458 bytes, file-ratio: 0.06%	+
manifest > general	name: winim, description: n/a, level: unknown	+
debug	n/a	+
thread-local-storage > callbacks	3	+
entry-point > address	0x00001125	+
certificate	n/a	+
imphash > md5	1E0EEA6D40F280D0C7563B489EBE8DA2	+
exports	n/a	+
overlay	n/a	+

library (2)	duplicate (0)	flag (0)	first-thunk-original (INT)	first-thunk (IAT)	type (1)	imports (114)
KERNEL32.dll	-	-	0x000B803C	0x000B83DC	implicit	56
msvcrt.dll	-	-	0x000B8204	0x000B85A4	implicit	58

pestudio 9.59 - Malware Initial Assessment - www.winitor.com (read-only)

file settings about

imports (114)	flag (18)	first-thunk-original (INT)	first-thunk (IAT)	hint	group (0)	technique (6)
GetCurrentProcessId	x	0x00000000000B8830	0x00000000000B8830	553 (0x0229)	reconnaissance	T1057 | Process Discovery
GetThreadContext	x	0x00000000000B8904	0x00000000000B8904	783 (0x030F)	reconnaissance	T1055 | Process Injection
GetThreadPriority	x	0x00000000000B8918	0x00000000000B8918	793 (0x0319)	reconnaissance	-
QueryPerformanceFrequency	x	0x00000000000B89FC	0x00000000000B89FC	1132 (0x046C)	reconnaissance	-
VirtualAlloc	x	0x00000000000B8B54	0x00000000000B8B54	1486 (0x05CE)	memory	T1055 | Process Injection
VirtualProtect	x	0x00000000000B8B72	0x00000000000B8B72	1492 (0x05D4)	memory	T1055 | Process Injection
VirtualQuery	x	0x00000000000B8B84	0x00000000000B8B84	1494 (0x05D6)	memory	T1055 | Process Injection
GetCurrentProcess	x	0x00000000000B881C	0x00000000000B881C	552 (0x0228)	execution	T1057 | Process Discovery
GetCurrentThread	x	0x00000000000B8846	0x00000000000B8846	556 (0x022C)	execution	-
GetCurrentThreadId	x	0x00000000000B885A	0x00000000000B885A	557 (0x022D)	execution	T1057 | Process Discovery
OpenProcess	x	0x00000000000B89BE	0x00000000000B89BE	1069 (0x042D)	execution	T1055 | Process Injection
SetProcessAffinityMask	x	0x00000000000B8A9A	0x00000000000B8A9A	1345 (0x0541)	execution	-
SetThreadContext	x	0x00000000000B8AB4	0x00000000000B8AB4	1368 (0x0558)	execution	T1055 | Process Injection
SuspendThread	x	0x00000000000B8B02	0x00000000000B8B02	1418 (0x058A)	execution	T1055 | Process Injection
AddVectoredExceptionHandler	x	0x00000000000B877C	0x00000000000B877C	20 (0x0014)	exception	-
RaiseException	x	0x00000000000B8A18	0x00000000000B8A18	1153 (0x0481)	exception	-
RemoveVectoredExceptionH...	x	0x00000000000B8A3E	0x00000000000B8A3E	1207 (0x04B7)	exception	-
OutputDebugStringA	x	0x00000000000B89CC	0x00000000000B89CC	1078 (0x0436)	diagnostic	-
CreateEventA	-	0x00000000000B87A8	0x00000000000B87A8	197 (0x00C5)	synchro	-
CreateSemaphoreA	-	0x00000000000B87B8	0x00000000000B87B8	243 (0x00F3)	synchro	-
DeleteCriticalSection	-	0x00000000000B87CC	0x00000000000B87CC	283 (0x011B)	synchro	-
EnterCriticalSection	-	0x00000000000B87F6	0x00000000000B87F6	319 (0x013F)	synchro	-
InitializeCriticalSection	-	0x00000000000B893C	0x00000000000B893C	892 (0x037C)	synchro	-
LeaveCriticalSection	-	0x00000000000B8980	0x00000000000B8980	984 (0x03D8)	synchro	-
ReleaseSemaphore	-	0x00000000000B8A2A	0x00000000000B8A2A	1196 (0x04AC)	synchro	-
ResetEvent	-	0x00000000000B8A60	0x00000000000B8A60	1214 (0x04BE)	synchro	-
SetEvent	-	0x00000000000B8A7E	0x00000000000B8A7E	1306 (0x051A)	synchro	-
TryEnterCriticalSection	-	0x00000000000B8B3A	0x00000000000B8B3A	1452 (0x05AC)	synchro	-
WaitForMultipleObjects	-	0x00000000000B8B94	0x00000000000B8B94	1501 (0x05DD)	synchro	-
WaitForSingleObject	-	0x00000000000B8BAE	0x00000000000B8BAE	1503 (0x05DF)	synchro	-
GetStartupInfoA	-	0x00000000000B88D8	0x00000000000B88D8	743 (0x02E7)	reconnaissance	-

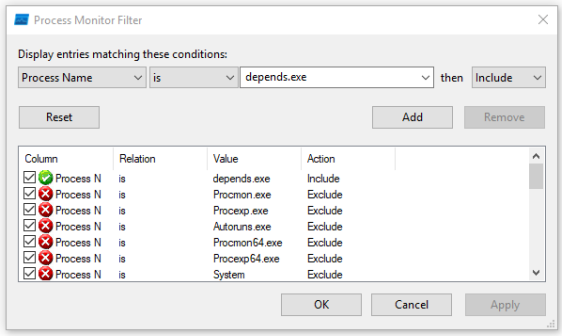
sha256: FF006C9EDEBCADB8675C6EF17F8D860E59212410E10945ECA1F62EC20812702A | cpu: 64-bit | file > type: executable | subsystem: GUI | entr

Basic Dynamic Analysis

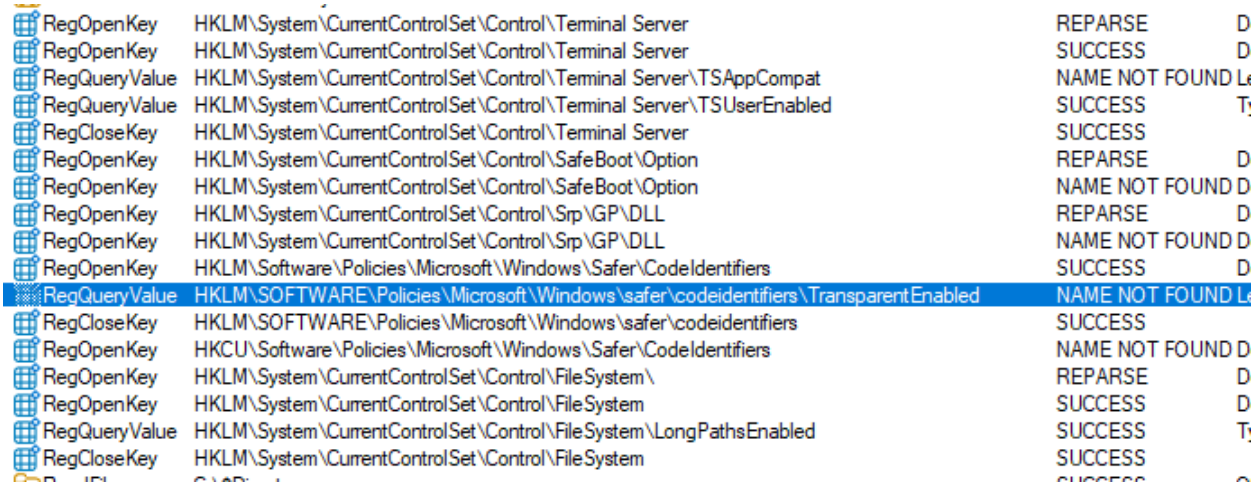
Process Monitor

Process Monitor is set to filter process names that are depends.exe. The malicious depends.exe is executed and we can see many ReadFiles, RegOpenKeys, and RegQueryValues. Creating DEPENDS.EXE seems to be an intended function.

Time ...	Process Name	PID	Operation	Path	Result	Detail
4:06:1...	depends.exe	4604	Process Start		SUCCESS	Parent PID: 3012, ...
4:06:1...	depends.exe	4604	Thread Create		SUCCESS	Thread ID: 1928
4:06:1...	depends.exe	4604	Load Image	C:\Users\IEUser\Desktop\depends22_...	SUCCESS	Image Base: 0x7ff7...
4:06:1...	depends.exe	4604	Load Image	C:\Windows\System32\ntdll.dll	SUCCESS	Image Base: 0x7ffc...
4:06:1...	depends.exe	4604	ReadFile	C:\$Directory	SUCCESS	Offset: 20,480, Len...
4:06:1...	depends.exe	4604	ReadFile	C:\$Directory	SUCCESS	Offset: 32,768, Len...
4:06:1...	depends.exe	4604	CreateFile	C:\Windows\Prefetch\DEPENDS.EXE-...	NAME NOT FOUND	Desired Access: G...
4:06:1...	depends.exe	4604	ReadFile	C:\Windows\System32\ntdll.dll	SUCCESS	Offset: 1,392,128, ...
4:06:1...	depends.exe	4604	RegOpenKey	HKLM\SYSTEM\CurrentControlSet\Con...	REPARSE	Desired Access: Q...
4:06:1...	depends.exe	4604	RegOpenKey	HKLM\System\CurrentControlSet\Contr...	NAME NOT FOUND	Desired Access: Q...
4:06:1...	depends.exe	4604	RegOpenKey	HKLM\SYSTEM\CurrentControlSet\Con...	REPARSE	Desired Access: Q...
4:06:1...	depends.exe	4604	RegOpenKey	HKLM\System\CurrentControlSet\Contr...	SUCCESS	Desired Access: Q...
4:06:1...	depends.exe	4604	RegQueryValue	HKLM\System\CurrentControlSet\Contr...	NAME NOT FOUND	Length: 24
4:06:1...	depends.exe	4604	RegCloseKey	HKLM\System\CurrentControlSet\Contr...	SUCCESS	
4:06:1...	depends.exe	4604	CreateFile	C:\Users\IEUser\Desktop\depends22_...	SUCCESS	Desired Access: E...
4:06:1...	depends.exe	4604	Load Image	C:\Windows\System32\kernel32.dll	SUCCESS	Image Base: 0x7ffc...
4:06:1...	depends.exe	4604	ReadFile	C:\Windows\System32\kernel32.dll	SUCCESS	Offset: 679,424, Le...
4:06:1...	depends.exe	4604	ReadFile	C:\Windows\System32\kernel32.dll	SUCCESS	Offset: 644,608, Le...
4:06:1...	depends.exe	4604	Load Image	C:\Windows\System32\KernelBase.dll	SUCCESS	Image Base: 0x7ffc...
4:06:1...	depends.exe	4604	ReadFile	C:\Windows\System32\KernelBase.dll	SUCCESS	Offset: 2,335,232, ...
4:06:1...	depends.exe	4604	ReadFile	C:\Windows\System32\KernelBase.dll	SUCCESS	Offset: 2,235,392, ...
4:06:1...	depends.exe	4604	ReadFile	C:\Windows\System32\KernelBase.dll	SUCCESS	Offset: 2,399,232, ...
4:06:1...	depends.exe	4604	ReadFile	C:\Windows\System32\KernelBase.dll	SUCCESS	Offset: 1,575,936, ...
4:06:1...	depends.exe	4604	ReadFile	C:\Windows\System32\KernelBase.dll	SUCCESS	Offset: 1,559,552, ...
4:06:1...	depends.exe	4604	ReadFile	C:\Windows\System32\KernelBase.dll	SUCCESS	Offset: 2,325,504, ...
4:06:1...	depends.exe	4604	ReadFile	C:\Windows\System32\kernel32.dll	SUCCESS	Offset: 660,992, Le...
4:06:1...	depends.exe	4604	ReadFile	C:\Windows\System32\kernel32.dll	SUCCESS	Offset: 677,376, Le...
4:06:1...	depends.exe	4604	ReadFile	C:\Windows\System32\locale.nls	SUCCESS	Offset: 688,128, Le...
4:06:1...	depends.exe	4604	ReadFile	C:\Windows\System32\locale.nls	SUCCESS	Offset: 753,664, Le...
4:06:1...	depends.exe	4604	ReadFile	C:\Windows\System32\locale.nls	SUCCESS	Offset: 786,432, Le...
4:06:1...	depends.exe	4604	RegOpenKey	HKLM\System\CurrentControlSet\Contr...	REPARSE	Desired Access: R
4:06:1...	depends.exe	4604	RegOpenKey	HKLM\System\CurrentControlSet\Contr...	SUCCESS	Desired Access: R
4:06:1...	depends.exe	4604	RegQueryValue	HKLM\System\CurrentControlSet\Contr...	NAME NOT FOUND	Length: 548
4:06:1...	depends.exe	4604	RegQueryValue	HKLM\System\CurrentControlSet\Contr...	SUCCESS	Type: REG_DWO...
4:06:1...	depends.exe	4604	RegCloseKey	HKLM\System\CurrentControlSet\Contr...	SUCCESS	
4:06:1...	depends.exe	4604	RegOpenKey	HKLM\System\CurrentControlSet\Contr...	REPARSE	Desired Access: Q...
4:06:1...	depends.exe	4604	RegOpenKey	HKLM\System\CurrentControlSet\Contr...	NAME NOT FOUND	Desired Access: Q...

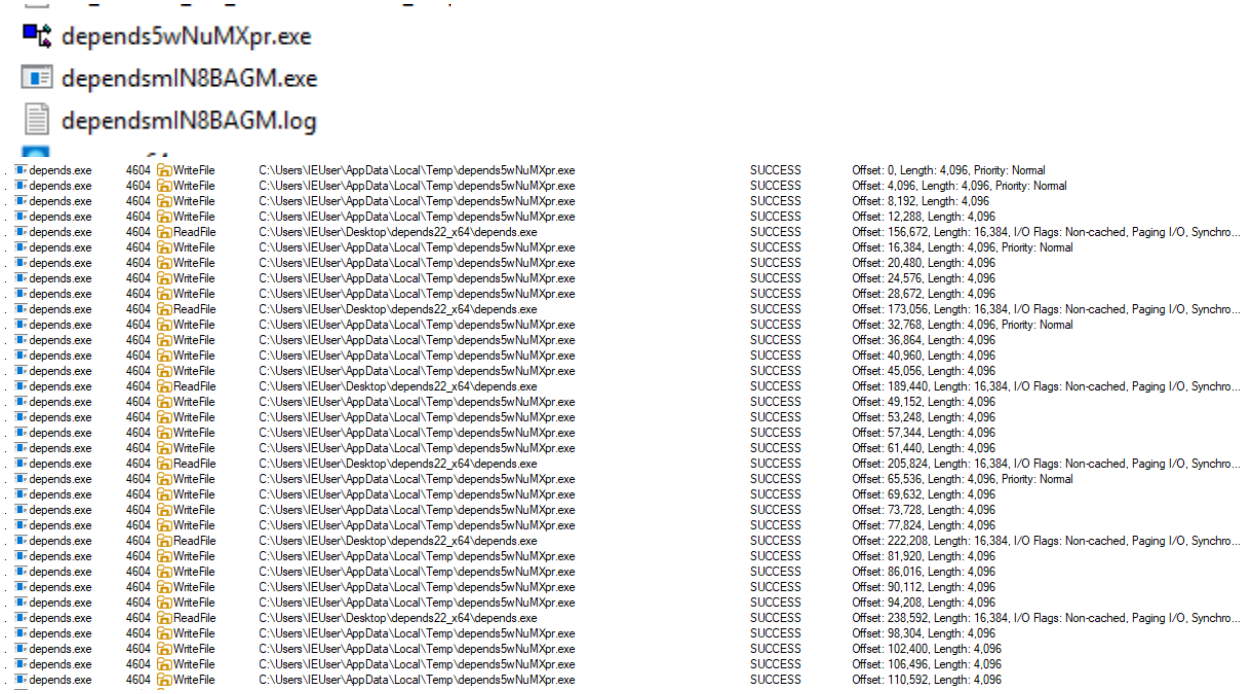


Here are some likely malicious actions performed by the process; we know previously that the malicious file is doing something related to code identifiers in the console. Here we can see the process performing RegQueryValue to modify registry keys related to Terminal Server, Code Identifiers, and FileSystem. It also reads the registry keys related to SafeBoot.



RegOpenKey	HKLM\System\CurrentControlSet\Control\Terminal Server	REPARSE	D
RegOpenKey	HKLM\System\CurrentControlSet\Control\Terminal Server	SUCCESS	D
RegQueryValue	HKLM\System\CurrentControlSet\Control\Terminal Server\TSAppCompat	NAME NOT FOUND	La
RegQueryValue	HKLM\System\CurrentControlSet\Control\Terminal Server\TSUserEnabled	SUCCESS	Ty
RegCloseKey	HKLM\System\CurrentControlSet\Control\Terminal Server	SUCCESS	
RegOpenKey	HKLM\System\CurrentControlSet\Control\SafeBoot\Option	REPARSE	D
RegOpenKey	HKLM\System\CurrentControlSet\Control\SafeBoot\Option	NAME NOT FOUND	D
RegOpenKey	HKLM\System\CurrentControlSet\Control\Srp\GP\DLL	REPARSE	D
RegOpenKey	HKLM\System\CurrentControlSet\Control\Srp\GP\DLL	NAME NOT FOUND	D
RegOpenKey	HKLM\Software\Policies\Microsoft\Windows\Safer\CodeIdentifiers	SUCCESS	D
RegQueryValue	HKLM\SOFTWARE\Policies\Microsoft\Windows\safer\codeidentifiers\TransparentEnabled	NAME NOT FOUND	La
RegCloseKey	HKLM\SOFTWARE\Policies\Microsoft\Windows\safer\codeidentifiers	SUCCESS	
RegOpenKey	HKCU\Software\Policies\Microsoft\Windows\Safer\CodeIdentifiers	NAME NOT FOUND	D
RegOpenKey	HKLM\System\CurrentControlSet\Control\FileSystem\	REPARSE	D
RegOpenKey	HKLM\System\CurrentControlSet\Control\FileSystem	SUCCESS	D
RegQueryValue	HKLM\System\CurrentControlSet\Control\FileSystem\LongPathsEnabled	SUCCESS	Ty
RegCloseKey	HKLM\System\CurrentControlSet\Control\FileSystem	SUCCESS	

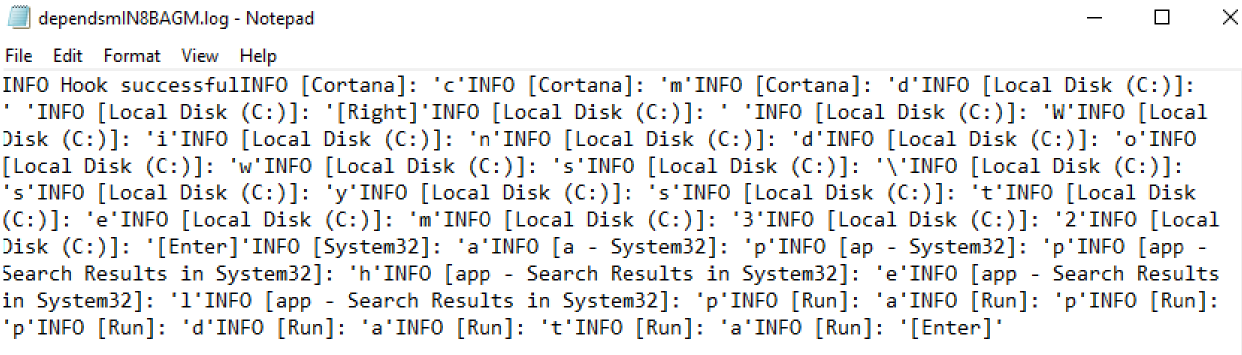
The process also creates an exe file in a temporary directory. The values after ‘depends’ is most likely generated using the bcrypt library.



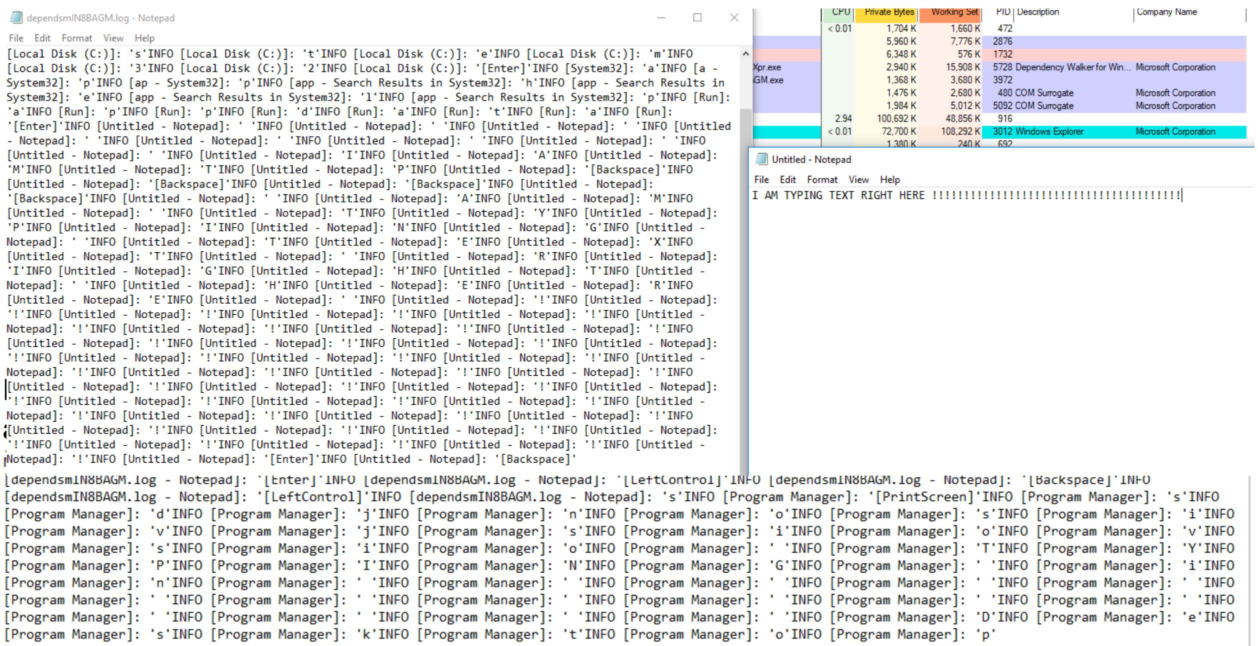
depends.exe	4604	WriteFile	C:\Users\IEUser\AppData\Local\Temp\depends5wNuMXpr.exe	SUCCESS	Offset: 0, Length: 4,096, Priority: Normal
depends.exe	4604	WriteFile	C:\Users\IEUser\AppData\Local\Temp\depends5wNuMXpr.exe	SUCCESS	Offset: 4,096, Length: 4,096, Priority: Normal
depends.exe	4604	WriteFile	C:\Users\IEUser\AppData\Local\Temp\depends5wNuMXpr.exe	SUCCESS	Offset: 8,192, Length: 4,096
depends.exe	4604	WriteFile	C:\Users\IEUser\AppData\Local\Temp\depends5wNuMXpr.exe	SUCCESS	Offset: 12,288, Length: 4,096
depends.exe	4604	ReadFile	C:\Users\IEUser\Desktop\depends22_x64\depends.exe	SUCCESS	Offset: 156,672, Length: 16,384, I/O Flags: Non-cached, Paging I/O, Synchro...
depends.exe	4604	WriteFile	C:\Users\IEUser\AppData\Local\Temp\depends5wNuMXpr.exe	SUCCESS	Offset: 16,384, Length: 4,096, Priority: Normal
depends.exe	4604	WriteFile	C:\Users\IEUser\AppData\Local\Temp\depends5wNuMXpr.exe	SUCCESS	Offset: 20,480, Length: 4,096
depends.exe	4604	WriteFile	C:\Users\IEUser\AppData\Local\Temp\depends5wNuMXpr.exe	SUCCESS	Offset: 24,576, Length: 4,096
depends.exe	4604	WriteFile	C:\Users\IEUser\AppData\Local\Temp\depends5wNuMXpr.exe	SUCCESS	Offset: 28,672, Length: 4,096
depends.exe	4604	ReadFile	C:\Users\IEUser\Desktop\depends22_x64\depends.exe	SUCCESS	Offset: 173,056, Length: 16,384, I/O Flags: Non-cached, Paging I/O, Synchro...
depends.exe	4604	WriteFile	C:\Users\IEUser\AppData\Local\Temp\depends5wNuMXpr.exe	SUCCESS	Offset: 32,768, Length: 4,096, Priority: Normal
depends.exe	4604	WriteFile	C:\Users\IEUser\AppData\Local\Temp\depends5wNuMXpr.exe	SUCCESS	Offset: 36,864, Length: 4,096
depends.exe	4604	WriteFile	C:\Users\IEUser\AppData\Local\Temp\depends5wNuMXpr.exe	SUCCESS	Offset: 40,960, Length: 4,096
depends.exe	4604	WriteFile	C:\Users\IEUser\AppData\Local\Temp\depends5wNuMXpr.exe	SUCCESS	Offset: 45,056, Length: 4,096
depends.exe	4604	ReadFile	C:\Users\IEUser\Desktop\depends22_x64\depends.exe	SUCCESS	Offset: 189,440, Length: 16,384, I/O Flags: Non-cached, Paging I/O, Synchro...
depends.exe	4604	WriteFile	C:\Users\IEUser\AppData\Local\Temp\depends5wNuMXpr.exe	SUCCESS	Offset: 49,152, Length: 4,096
depends.exe	4604	WriteFile	C:\Users\IEUser\AppData\Local\Temp\depends5wNuMXpr.exe	SUCCESS	Offset: 53,248, Length: 4,096
depends.exe	4604	WriteFile	C:\Users\IEUser\AppData\Local\Temp\depends5wNuMXpr.exe	SUCCESS	Offset: 57,344, Length: 4,096
depends.exe	4604	WriteFile	C:\Users\IEUser\AppData\Local\Temp\depends5wNuMXpr.exe	SUCCESS	Offset: 61,440, Length: 4,096
depends.exe	4604	ReadFile	C:\Users\IEUser\Desktop\depends22_x64\depends.exe	SUCCESS	Offset: 205,824, Length: 16,384, I/O Flags: Non-cached, Paging I/O, Synchro...
depends.exe	4604	WriteFile	C:\Users\IEUser\AppData\Local\Temp\depends5wNuMXpr.exe	SUCCESS	Offset: 65,536, Length: 4,096, Priority: Normal
depends.exe	4604	WriteFile	C:\Users\IEUser\AppData\Local\Temp\depends5wNuMXpr.exe	SUCCESS	Offset: 69,632, Length: 4,096
depends.exe	4604	WriteFile	C:\Users\IEUser\AppData\Local\Temp\depends5wNuMXpr.exe	SUCCESS	Offset: 73,728, Length: 4,096
depends.exe	4604	WriteFile	C:\Users\IEUser\AppData\Local\Temp\depends5wNuMXpr.exe	SUCCESS	Offset: 77,824, Length: 4,096
depends.exe	4604	ReadFile	C:\Users\IEUser\Desktop\depends22_x64\depends.exe	SUCCESS	Offset: 222,208, Length: 16,384, I/O Flags: Non-cached, Paging I/O, Synchro...
depends.exe	4604	WriteFile	C:\Users\IEUser\AppData\Local\Temp\depends5wNuMXpr.exe	SUCCESS	Offset: 81,920, Length: 4,096
depends.exe	4604	WriteFile	C:\Users\IEUser\AppData\Local\Temp\depends5wNuMXpr.exe	SUCCESS	Offset: 86,016, Length: 4,096
depends.exe	4604	WriteFile	C:\Users\IEUser\AppData\Local\Temp\depends5wNuMXpr.exe	SUCCESS	Offset: 90,112, Length: 4,096
depends.exe	4604	WriteFile	C:\Users\IEUser\AppData\Local\Temp\depends5wNuMXpr.exe	SUCCESS	Offset: 94,208, Length: 4,096
depends.exe	4604	ReadFile	C:\Users\IEUser\Desktop\depends22_x64\depends.exe	SUCCESS	Offset: 238,592, Length: 16,384, I/O Flags: Non-cached, Paging I/O, Synchro...
depends.exe	4604	WriteFile	C:\Users\IEUser\AppData\Local\Temp\depends5wNuMXpr.exe	SUCCESS	Offset: 98,304, Length: 4,096
depends.exe	4604	WriteFile	C:\Users\IEUser\AppData\Local\Temp\depends5wNuMXpr.exe	SUCCESS	Offset: 102,400, Length: 4,096
depends.exe	4604	WriteFile	C:\Users\IEUser\AppData\Local\Temp\depends5wNuMXpr.exe	SUCCESS	Offset: 106,496, Length: 4,096
depends.exe	4604	WriteFile	C:\Users\IEUser\AppData\Local\Temp\depends5wNuMXpr.exe	SUCCESS	Offset: 110,592, Length: 4,096

The .log file shows keys that were typed (input) and their location (output). This must be where the malicious executable is utilizing console-related functions to receive bytes within console

buffers and iostreams.

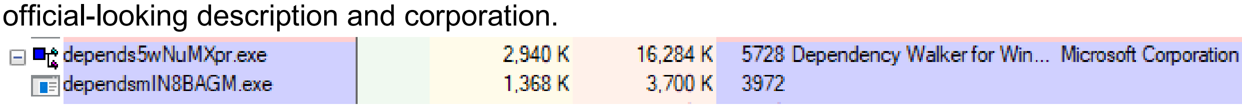


Further testing and demonstration of keylogging by the process.

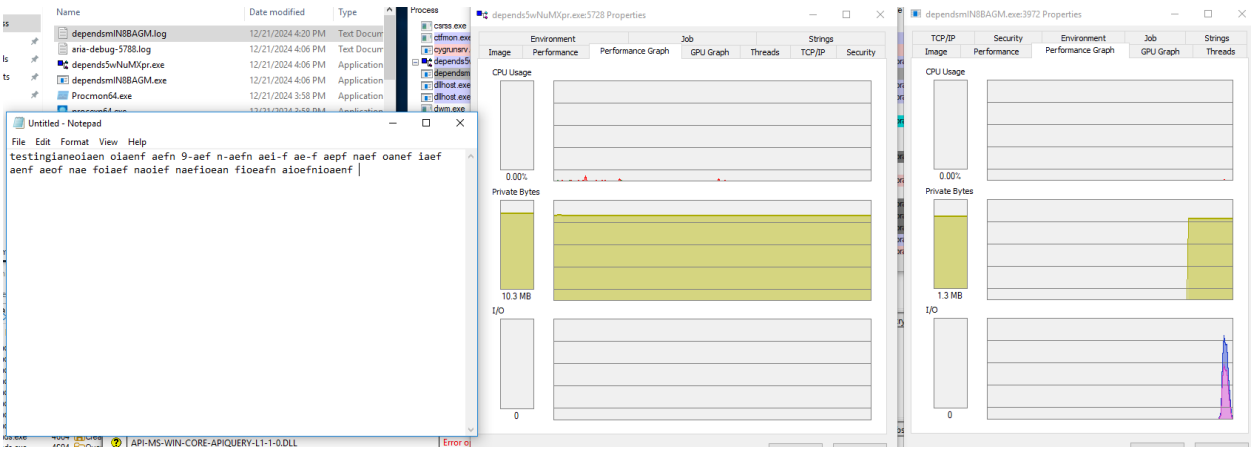


Process Explorer

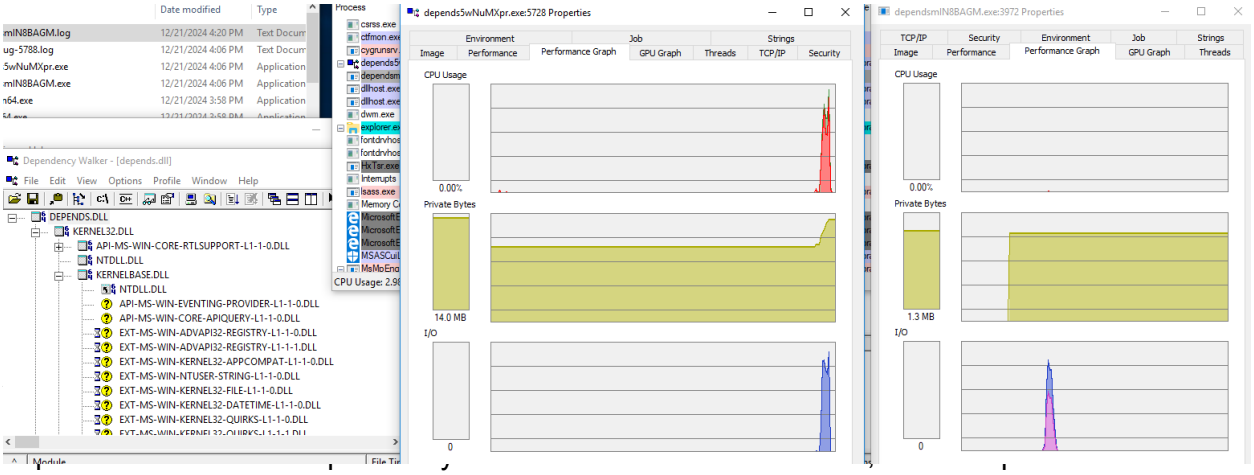
We find that the keylogger process is running under a functional depends.exe process with official-looking description and corporation.



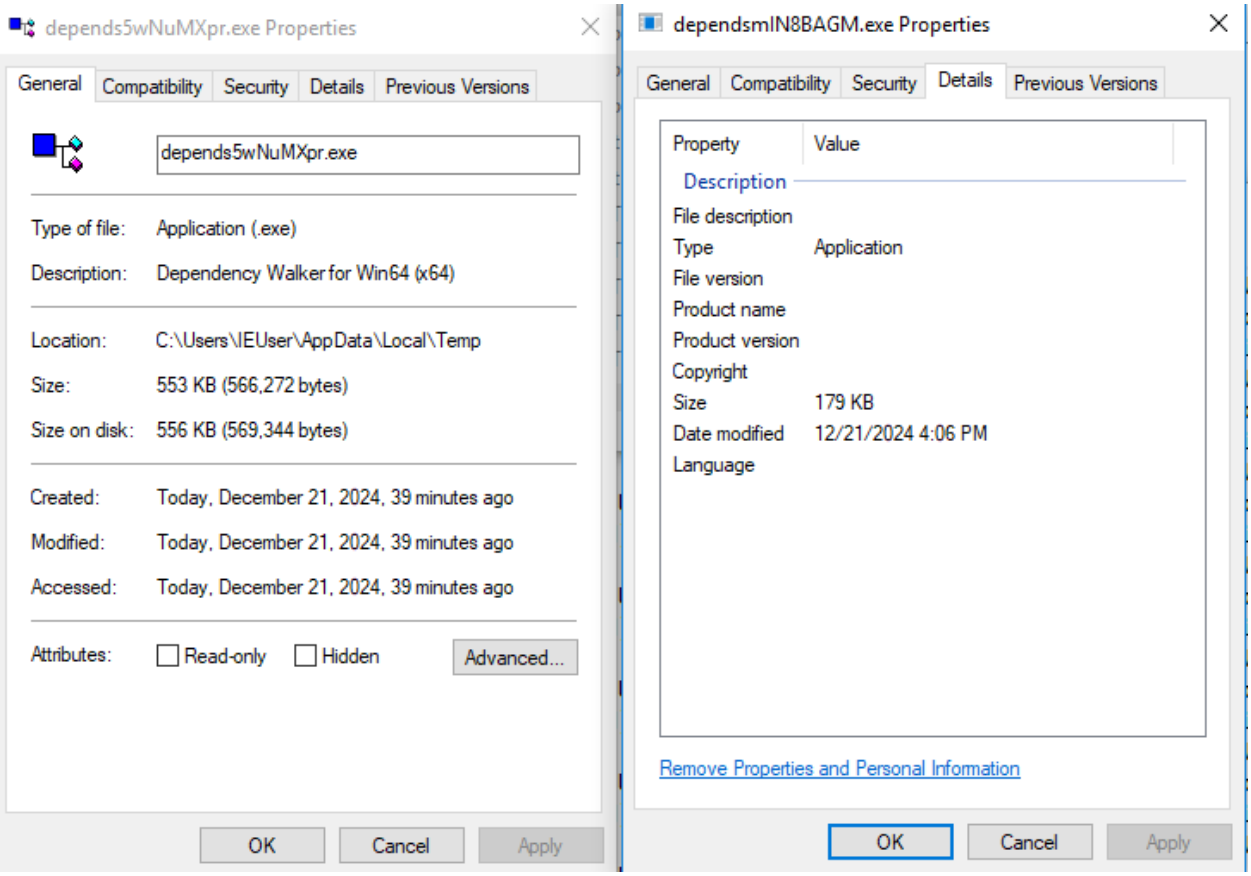
Here is another demonstration of the keylogger affecting performance; when inputs/outputs are performed, such as typing, the child process shows signs of I/O activity. Additionally, there are no seeable changes in private bytes, meaning the 1.3mb must either be reserved to handle writing to log or to obscure changes in activity.



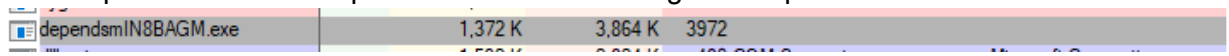
Here is a demonstration of the usage of the Dependency Walker application that the keylogger is hidden under; There are noticeable increases in CPU usage, Private Bytes, and I/O activity, but no changes are seen in the child process. It is possible that the parent process is only for concealment and has no actual malicious functions.



executable generated by the malicious depends.exe is 179 KB, roughly the difference between the malicious depends.exe and the safe depends.exe. This confirms that the parent process is only a disguise with no malicious behavior while the child process with keylogging functions performs most if not all of the malicious actions.

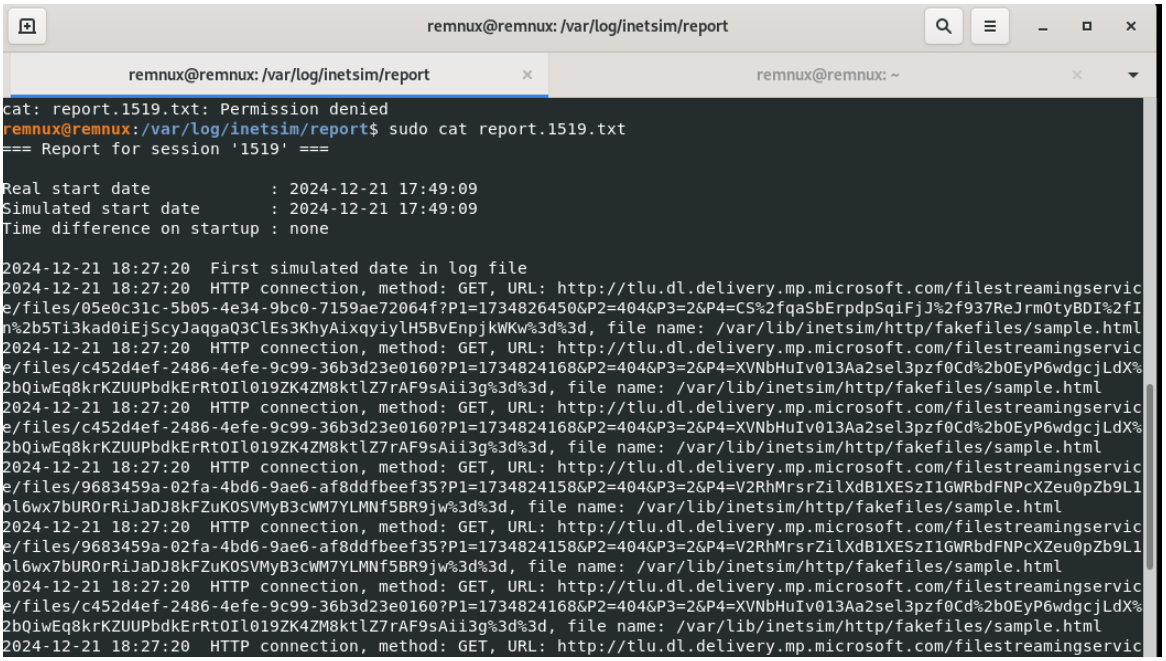


When the user exits the program (such as by using the top-right X button), the malicious process persists as its own process rather than being a child process.



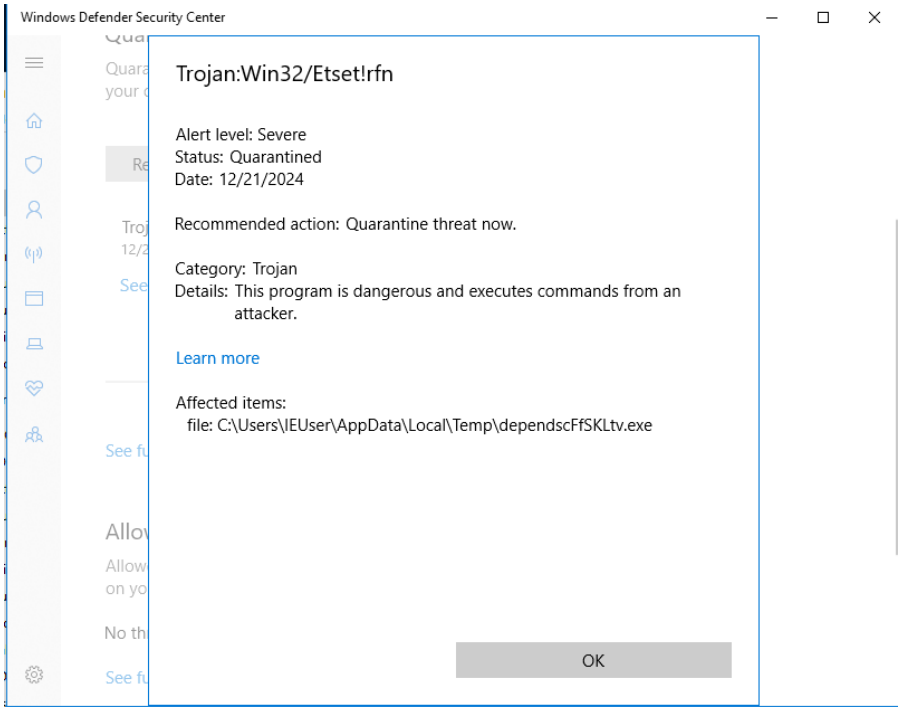
On restarting the virtual machine, the program does not appear to automatically execute.

INetSim



Inetsim logs don't show any abnormalities when the malware sample is run. It just has basic Microsoft requests for updates.

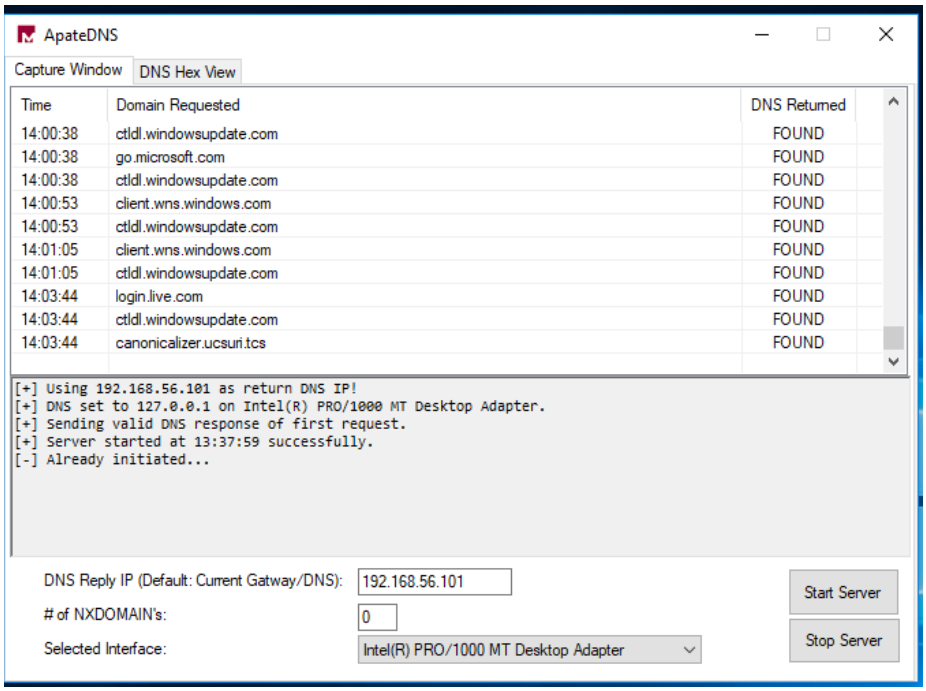
Windows Defender



Windows Defender catches the malware sample when it is run and marks it as a trojan virus. This means that the virus pretends to be helpful in some way to enter a system. Windows

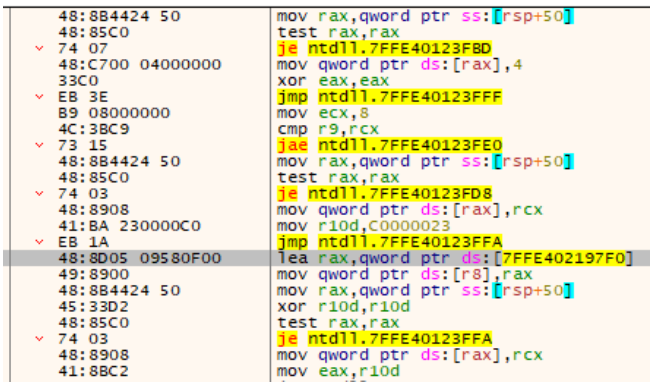
Defender states that the malware affects the file at C:\Users\IEUser\AppData\Local\dependscFfSKLtv.exe.

apateDNS



Running the malware with apateDNS, there wasn't anything worth mentioning that was tracked.

Advanced AnalysisX64dbg



The code involves conditional logic, memory manipulations, and control flow obfuscation, which are typical of techniques used to evade detection and analysis. The jumps, **XOR** operations, and memory writes suggest that the program may be altering critical data structures, potentially to control the flow of execution or to hide malicious intent. The manipulation of low-level system structures (**ldrinit.c**) further hints at potential malicious behavior related to process or module manipulation.

```
add byte ptr ds:[rax],al
add byte ptr ds:[rax],al
add byte ptr ds:[rax],al
add byte ptr ds:[rax],al
add byte ptr ds:[rax],al
add byte ptr ds:[rax],al
add byte ptr ds:[rax],al
add byte ptr ds:[rax],al
add byte ptr ds:[rax],al
add byte ptr ds:[rax],al
add byte ptr ds:[rax],al
add byte ptr ds:[rax],al
add byte ptr ds:[rax],al
add byte ptr ds:[rax],al
add byte ptr ds:[rax],al
add byte ptr ds:[rax],al
add byte ptr ds:[rax],al
add byte ptr ds:[rax],al
```

The repeated execution of the instruction **add byte ptr ds:[rax], al** suggests potential malicious activity, as it continuously modifies memory at locations pointed to by the **RAX** register. This could be an indication of a buffer overflow, data corruption, or memory manipulation. The repetitive nature of the operation could also suggest attempts to manipulate or inject harmful data into key areas of memory, potentially leading to unauthorized actions, data tampering, or system instability.

Ghidra

Decompile: FUN_14000189b - (depends.exe)

```
1
2  char FUN_14000189b(longlong *param_1)
3
4  {
5    ulonglong uVar1;
6    char cVar2;
7
8    cVar2 = '\0';
9    if (param_1 != (longlong *)0x0) {
10     uVar1 = param_1[-2];
11     if ((uVar1 & 0xfffffffffffffff0) == 0) {
12       cVar2 = '\x01';
13     }
14     else {
15       if (SBORROW8(uVar1,0x10)) {
16         FUN_140003fea();
17         return '\0';
18       }
19       param_1[-2] = uVar1 - 0x10;
20     }
21     FUN_140004e1d(cVar2,(ulonglong *)(param_1 + -2),*param_1);
22   }
23   return cVar2;
```

The function **FUN_14000189b** performs low-level memory manipulation, including checking and adjusting a memory value at **param_1[-2]**. It subtracts 0x10 from this value and checks for an underflow. If an underflow is detected, it calls another function (**FUN_140003fea()**) to handle the situation. This type of memory handling is often associated with exploits or obfuscation techniques, suggesting the function might be used for malicious purposes.

```
Decompile: FUN_140003fea -  (depends.exe)

{
  undefined8 *puVar1;

  puVar1 = (undefined8 *)FUN_140004894(0x40,8);
  puVar1[2] = "OverflowDefect";
  *puVar1 = &PTR_LAB_140021160;
  puVar1[3] = 0x12;
  puVar1[4] = &DAT_1400227a0;
  FUN_14000452e((longlong)puVar1,"OverflowDefect",0x1400223f8,0x1400223ee,0x35);
  return;
}
```

The function **FUN_140003fea** seems to be related to error handling for a memory underflow condition. It allocates memory, initializes some values, and triggers a function (**FUN_14000452e**) that may be logging or handling the underflow event. The use of the string "**OverflowDefect**" suggests that this could be part of a vulnerability exploitation process, where memory defects like overflows or underflows are being monitored or triggered, possibly to control program flow or for malicious purposes.

```
  local_18 = 0;
  while( true ) {
    local_40 = &DAT_1400b7fe8;
    local_48 = local_28;
    local_50 = 0;
    LOCK();
    local_20 = DAT_1400b7fe8;
    if (DAT_1400b7fe8 == 0) {
      DAT_1400b7fe8 = local_28;
      local_20 = 0;
    }
    UNLOCK();
    if (local_20 == 0) goto LAB_14000122f;
    if (local_20 == local_28) break;
    Sleep(1000);
  }
  local_18 = 1;
LAB_14000122f:
  if (DAT_1400b7fe0 == 1) {
    _amsg_exit(0x1f);
  }
  else if (DAT_1400b7fe0 == 0) {
    DAT_1400b7fe0 = 1;
    _initterm(&DAT_1400ba018,&DAT_1400ba030);
  }
```

The function **FUN_140001154** shows some potentially suspicious activity, such as using **LOCK** and **UNLOCK** for synchronization and employing **Sleep(1000)** loops, which could be attempts to evade debugging or slow analysis. It also modifies global variables and sets an exception handler with SetUnhandledExceptionFilter, which could indicate preparation for handling unexpected conditions or hiding behavior. These actions warrant further investigation to determine their intent.

```
void FUN_140001591(int param_1,longlong *param_2)

{
  longlong lVar1;
  void *pvVar2;
  size_t sVar3;
  void *pvVar4;
  undefined4 local_1c;

  pvVar2 = malloc((longlong)(param_1 + 1) << 3);
  lVar1 = *param_2;
  for (local_1c = 0; local_1c < param_1; local_1c = local_1c + 1) {
    sVar3 = strlen(*(char **)(lVar1 + (longlong)local_1c * 8));
    pvVar4 = malloc(sVar3 + 1);
    *(void **)((longlong)local_1c * 8 + (longlong)pvVar2) = pvVar4;
    memcpy(*(void **)((longlong)pvVar2 + (longlong)local_1c * 8),
           *(void **)(lVar1 + (longlong)local_1c * 8),sVar3 + 1);
  }
  *(undefined8 *)((longlong)pvVar2 + (longlong)local_1c * 8) = 0;
  *param_2 = (longlong)pvVar2;
  return;
}
```

The function **FUN_140001591** dynamically allocates memory and copies strings using **malloc**, **strlen**, and **memcpy,** which could potentially lead to buffer overflows or memory misuse if input data is not properly validated.

```
void UndefinedFunction_1400016e3(void)

{
  HMODULE hModule;
  code *pcVar1;

  hModule = (HMODULE)(*DAT_1400b844c)("libgcc_s_dw2-1.dll");
  if (hModule == (HMODULE)0x0) {
    pcVar1 = FUN_1400016d0;
    DAT_140021010 = (FARPROC)&LAB_1400016e0;
  }
  else {
    DAT_1400b7040 = LoadLibraryA("libgcc_s_dw2-1.dll");
    pcVar1 = GetProcAddress(hModule,"__register_frame_info");
    DAT_140021010 = GetProcAddress(hModule,"__deregister_frame_info");
    if (pcVar1 == (FARPROC)0x0) goto LAB_140001757;
  }
  (*pcVar1)(&DAT_1400b1000,&DAT_1400b7060);
LAB_140001757:
  FUN_140001698((_onexit_t)&LAB_1400017a0);
  return;
}
```

The function **UndefinedFunction_1400016e3** dynamically loads a library (**libgcc_s_dw2-1.dll**) and retrieves the addresses of specific functions (**__register_frame_info** and **__deregister_frame_info**). While this behavior is not inherently malicious, it can be suspicious, especially if the library is not a standard or expected dependency, or if it is downloaded or provided dynamically. The function's reliance on dynamic loading and function resolution could potentially be used to obfuscate malicious intent or evade detection mechanisms.

How To Undo Damage if Compromised

Sure Fixes: guaranteed to remove malware but with potentially major losses to the system.

- Recommended: Revert to a previous backup before malware execution.

- Unrecommended: Perform a Factory Reset or install a new OS.
    - Likely overkill in context of this malware's actions.

General Fix: most likely to remove malware with minimal system losses.

1) Kill/Terminate the malicious processes.
2) Remove depends[rand].exe executables and related logs from AppData/local/Temp.
3) Delete/Uninstall related files and executables that were downloaded.
4) Restart your computer.

Malware Behavior Catalog

ID	**—**
Type	**Trojan, Keylogger**
Aliases	**depends.exe, depends[rand].exe**
Platforms	**Windows**
Year	**2024**
Associated ATT&CK Software	**None**

The Malware logs user key inputs and the input location.

ATT&CK Techniques

Defense Evasion::Process Injection (T1055) Defense Evasion::Process Injection::Portable Executable Injection (T1055.002)	The malware creates two new executables, one legitimate and another malicious, in appdata/local/temp and runs the malicious executable under the context/process of the legitimate executable.
Collection::Input Capture (T1056) Collection::Input Capture::Keylogging (T1056.001)	The malware captures key inputs and the location/process of the key input and stores it in a .log file within appdata/local/temp.

MBC Behaviors

Executable Code Obfuscation (B0032)	Conceal the true purpose of code to hide malware

Indicators of Compromise

MD5 Hashes	c638a62bd2fe7b52788183edbc85d335

SHA256 Hashes	ff006c9edebcadb8675c6ef17f8d860e59212410e10945eca1f62ec20812702a

Yara

Suspicious Network Activity Rule:

```
File  Edit  Format  View  Help
rule Malware_NetworkActivity
{
    meta:
        description = "Detects malware communicating with remote servers"
        author = "Group 12"
        date = "2024-12-21"
        version = "1.1"

    strings:
        $url_http = /https?:\/\/[a-zA-Z0-9\.\-_\/]+/ nocase
        $ip_hardcoded = /(\d{1,3}\.){3}\d{1,3}/
        $dns_resolve = "GetAddrInfoW"
        $api_socket = "WSAStartup"

    condition:
        uint16(0) == 0x5A4D and
        (any of ($url_http, $ip_hardcoded) or $dns_resolve or $api_socket)
}
```

This rule flags malware that communicates with remote servers. It identifies hardcoded URLs or IP addresses, which are often used to send stolen data or receive commands. Additionally, it looks for network-related APIs like GetAddrInfoW and WSAStartup, which are commonly used to establish connections or resolve domain names in malicious programs.

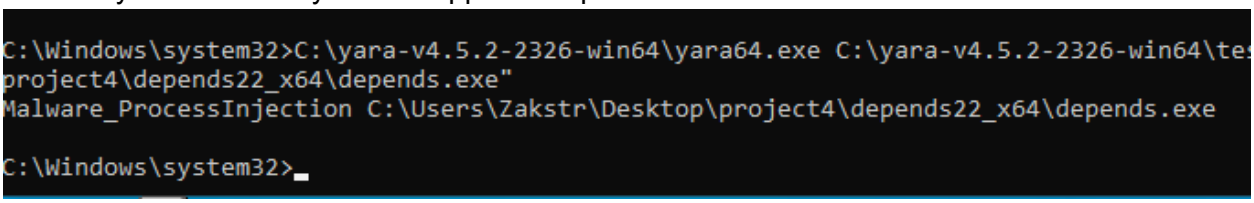
```
C:\Windows\system32>C:\yara-v4.5.2-2326-win64\yara64.exe C:\yara-v4.5.2-2326-win64\test_rule
project4\depends22_x64\depends.exe"
warning: rule "Malware_NetworkActivity" in C:\yara-v4.5.2-2326-win64\test_rule.yara(11): str
wn scanning
Malware_NetworkActivity C:\Users\Zakstr\Desktop\project4\depends22_x64\depends.exe

C:\Windows\system32>_
```

The depends.exe file ended up being flagged positive after running it with this rule, which shows that there is likely malicious activity regarding communications. Although there is a risk that this was a false positive.

Process Injection Rule:

```
rule Malware_ProcessInjection
{
    meta:
        description = "Detects process injection activities such as memory manipulation or thread hijacking"
        author = "Group 12"
        date = "2024-12-21"

    strings:
        $api1 = "VirtualAllocEx"
        $api2 = "WriteProcessMemory"
        $api3 = "CreateRemoteThread"
        $api4 = "NtQueueApcThread"
        $dll_inject = "LoadLibrary"

    condition:
        uint16(0) == 0x5A4D and // PE file check
        (2 of ($api*) or $dll_inject)
}
```

This rule detects malware techniques that inject malicious code into other running processes. By using APIs like VirtualAllocEx, WriteProcessMemory, and CreateRemoteThread, malware can execute its code within legitimate processes. This helps it evade security tools by blending its activity with trusted system or application processes.

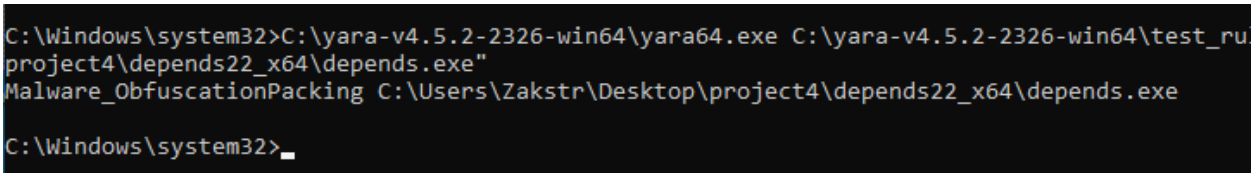


The executable was flagged for using persistence mechanisms, likely modifying registry keys or placing files in startup locations to ensure it runs automatically after a reboot. This suggests the malware is attempting to maintain a foothold on the system

Malware Obfuscation Packing Rule:

```
rule Malware_ObfuscationPacking
{
    meta:
        description = "Detects packed or obfuscated files with specific characteristics"
        author = "Group 12"
        date = "2024-12-21"
        version = "1.2"

    strings:
        // Signatures for common packers
        $packer_upx = "UPX" nocase
        $packer_aspack = "ASPack" nocase
        $packer_pecompact = "PECompact" nocase
        $packer_fsg = "FSG" nocase
        $packer_mpress = "MPRESS" nocase
        $packer_nsis = "NSIS" nocase

        // Specific obfuscation-related API calls
        $obfuscation_api1 = "VirtualProtect" nocase
        $obfuscation_api2 = "VirtualAlloc" nocase
        $obfuscation_api3 = "RtlCreateUserThread" nocase
        $obfuscation_api4 = "NtAllocateVirtualMemory" nocase

        // Strings that appear in unpacked regions or compressed sections of packed files
        $unpacked_data = "This program cannot be run in DOS mode" nocase
        $compressed_data = "This is a compressed file" nocase

    condition:
        uint16(0) == 0x5A4D and // PE file header (MZ)
        (
            any of ($packer_upx, $packer_aspack, $packer_pecompact, $packer_fsg, $packer_mpress, $packer_nsis)
            any of ($obfuscation_api1, $obfuscation_api2, $obfuscation_api3, $obfuscation_api4) or
            any of ($unpacked_data, $compressed_data)
        )
}
```

The Obfuscation or Packing rule is designed to detect files that are packed or obfuscated to evade detection. It identifies common packing tools like UPX, ASPack, and PECompact, which are used to compress or encrypt executable files. The rule also looks for specific memory manipulation APIs such as VirtualProtect, VirtualAlloc, and RtlCreateUserThread, which are frequently used in the unpacking or execution of obfuscated code. Additionally, it checks for strings commonly found in unpacked sections or compressed files.



The executable was flagged by the Obfuscation or Packing rule, indicating it uses packing or obfuscation techniques to hide its malicious behavior. This typically involves compressing or encrypting the file to evade detection.

Conclusion

The analysis of the **Depends22_x64.exe** malware sample demonstrated its capabilities as a sophisticated threat leveraging multiple evasion and attack techniques. Through basic and advanced static and dynamic analysis, we uncovered its key behaviors, such as keylogging, process injection, obfuscation using the Nim programming language, and cryptographic functionalities for potential data exfiltration or securing malicious communications. The malware's use of temporary files, suspicious DLLs, and hidden network activity further highlights its capacity to operate stealthily within a compromised system.

Tools like VirusTotal, Detect-It-Easy, PEStudio, Process Monitor, and Ghidra played an instrumental role in uncovering the true intent and functions of the malware. From identifying obfuscated libraries to logging registry and console input-output activity, we systematically analyzed the malicious executable. The findings suggest that the malware is tailored for reconnaissance, data collection, and persistence, using keylogging and network operations to achieve its objectives.

In conclusion, this project illustrates the importance of employing both static and dynamic analysis techniques for malware investigation. The insights gained emphasize the evolving nature of malware development, particularly the use of less common programming languages and advanced evasion tactics. These findings not only deepen our understanding of malware behavior but also highlight the critical need for continuous advancements in cybersecurity measures to counteract such sophisticated threats.

Appendix (Team Contribution)

Allen Dai: Inetseim, Remnux, Windows Defender

Paul Le: Detect-It-Easy, Process Monitor, Process Explorer, How to Undo Damage if

Compromised, Malware Behavior Catalog

Terry Ma:

Zohair Mamdani: MBC Behaviors, Conclusion

Wayne Muse: Discord Setup, documentation setup, basic static analysis

Zakariye Samatar: Yara analysis and rule creation; basic ghidra and x64dbg static analysis