Examining Smokeloader's Anti Hooking technique

malwareandstuff.com/examining-smokeloaders-anti-hooking-technique/

May 24, 2020

```
[+][+] Returned from strcmpiA
[+][+] Called strcmpiA
[+][+] Arg1Addr = 0x18eedc
[+][+] Buffer
          0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F  0123456789ABCDEF
00000000  77 69 6e 69 6e 69 74 2e 65 78 65 00 65 73 73 5d  wininit.exe.ess]
00000010  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................

[+][+] Arg2Addr = 0x360f978
[+][+] Buffer
          0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F  0123456789ABCDEF
00000000  61 76 67 63 73 72 76 78 2e 65 78 65 00 fa 60 03  avgcsrvx.exe..`.
00000010  34 ce 3a 36 46 96 00 0c 61 76 67 73 76 63 78 2e  4.:6F...avgsvcx.
00000020  65 78 65 00 92 20 53 f1 34 ce 3a 36 46 96 00 0b  exe.. S.4.:6F...

[+][+] Returned from strcmpiA
[+][+] Called strcmpiA
[+][+] Arg1Addr = 0x18eedc
[+][+] Buffer
          0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F  0123456789ABCDEF
00000000  77 69 6e 69 6e 69 74 2e 65 78 65 00 65 73 73 5d  wininit.exe.ess]
00000010  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................

[+][+] Arg2Addr = 0x360f990
[+][+] Buffer
          0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F  0123456789ABCDEF
00000000  61 76 67 73 76 63 78 2e 65 78 65 00 92 20 53 f1  avgsvcx.exe.. S.
00000010  34 ce 3a 36 46 96 00 0b 61 76 67 63 73 72 76 61  4.:6F...avgcsrva
00000020  2e 65 78 65 00 01 00 00 34 ce 3a 36 46 96 00 0c  .exe....4.:6F...
```

Published by **hackingump** on May 24, 2020

Hooking is a technique to intercept function calls/messages or events passed between software, or in this case malware. The technique can be used for malicious, as well as defensive cases.

Rootkits for example can hook API calls to make themselves invisible from analysis tools, while we as defenders can use hooking to gain more knowledge of malware or build detection mechanisms to protect customers.

Cybersecurity continues to be a game of cat and mouses, and while we try to build protections, blackhats will always try to bypass these protection mechanisms. Today I want to show you how SmokeLoader bypasses hooks on `ntdll.dll` and how Frida can be used to hook library functions.

The bypass was also already explained in a blog article from Checkpoint[1] written by Israel Gubi. It also covers a lot more than I do regarding Smokeloader, so it is definitely worth reading too.

Hooking with Frida

If you've read my previous blog articles about QBot, you are familiar with the process iteration and AV detection[3]. It iterates over processes and compares the process name with entries in a black list containing process names of common AV products. If one process name matches with an entry, QBot quits its execution.

Frida is a Dynamic Instrumentation Toolkit which can be used to write dynamic analysis scripts in high level languages, in this case JavaScript. If you want to know more about this technology, I advice you to read to visit this website[4] and read its documentation.

We can write a small Frida script to hook the `lstrcmpiA` function in order to investigate which process names are in the black list.

```
def main():
    """Main."""
    # argv[1] is our malware sample
    pid = frida.spawn(sys.argv[1])
    sess = frida.attach(pid)
    script = sess.create_script("""
        console.log("[+] Starting Frida script")
        var lstrcmpiA = ptr("0x76B43E8E")
        console.log("[+] Hooking lstrcmpiA at " + lstrcmpiA)
        Interceptor.attach(lstrcmpiA, {
            onEnter: function(args) {
                console.log("[+][+] Called strcmpiA");
                console.log("[+][+] Arg1Addr = " + args[0]);
                console.log("[+][+] Buffer");
                pretty_print(args[0], 0x30);
                console.log("[+][+] Arg2Addr = " + args[1]);
                console.log("[+][+] Buffer");
                pretty_print(args[1], 0x30);
            },
            onLeave: function(retval) {
                console.log("[+][+] Returned from strcmpiA")
            }
        });

        function pretty_print(addr, sz) {
            var bufptr = ptr(addr);
            var bytearr = Memory.readByteArray(bufptr, sz);
            console.log(bytearr);
        };

        """)
    script.load()
    frida.resume(pid)
    sys.stdin.read()
    sess.detach()
```

We attach to the malicious process and hook the `lstrcmpiA` function at static address. When analysing malware, we have (most of the time) the privilege to control and adjust our environment as much as we want. If you turn off `ASLR` and use snapshots, using Frida with static pointers is pretty convenient, because most functions will always have the same address. However, it's also possible to calculate the addresses dynamically. `lstrcmpiA` has 2 arguments, which are both pointers of type `LPSTR`. So we just resolve the pointers, fill `0x30` bytes starting at pointer address into a ByteArray and print it.

```
[+][+] Returned from strcmpiA
[+][+] Called strcmpiA
[+][+] Arg1Addr = 0x18eedc
[+][+] Buffer
           0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F  0123456789ABCDEF
00000000  77 69 6e 69 6e 69 74 2e 65 78 65 00 65 73 73 5d  wininit.exe.ess]
00000010  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................

[+][+] Arg2Addr = 0x360f978
[+][+] Buffer
           0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F  0123456789ABCDEF
00000000  61 76 67 63 73 72 76 78 2e 65 78 65 00 fa 60 03  avgcsrvx.exe..`.
00000010  34 ce 3a 36 46 96 00 0c 61 76 67 73 76 63 78 2e  4.:6F...avgsvcx.
00000020  65 78 65 00 92 20 53 f1 34 ce 3a 36 46 96 00 0b  exe.. S.4.:6F...

[+][+] Returned from strcmpiA
[+][+] Called strcmpiA
[+][+] Arg1Addr = 0x18eedc
[+][+] Buffer
           0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F  0123456789ABCDEF
00000000  77 69 6e 69 6e 69 74 2e 65 78 65 00 65 73 73 5d  wininit.exe.ess]
00000010  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................

[+][+] Arg2Addr = 0x360f990
[+][+] Buffer
           0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F  0123456789ABCDEF
00000000  61 76 67 73 76 63 78 2e 65 78 65 00 92 20 53 f1  avgsvcx.exe.. S.
00000010  34 ce 3a 36 46 96 00 0b 61 76 67 63 73 72 76 61  4.:6F...avgcsrva
00000020  2e 65 78 65 00 01 00 00 34 ce 3a 36 46 96 00 0c  .exe....4.:6F...
```

Result of Frida Script

Smokeloader's Anti Hooking technique

So how does Smokeloader bypass hooks? Well it can do it atleast for the `ntdll.dll` library. During execution Smokeloader retrieves the Temp folder path and generates a random name. If a file with the generated name already exists in the temp folder, it is deleted with `DeleteFileW`.

```
~~2204~~ ntdll.dll!RtlMultiByteToUnicodeN
    arg 0: 0x001897f4
    arg 1: 0x00000002
~~2204~~ ntdll.dll!memmove
    arg 0: 0x00189aba
    arg 1: 0x74f81df4
~~2204~~ KERNEL32.dll!DeleteFileW
    arg 0: C:\Users\BLACKB~1\AppData\Local\Temp\9A26.tmp (type=wchar_t*, size=0x0)
~~2204~~ KERNELBASE.dll!DeleteFileW
    arg 0: C:\Users\BLACKB~1\AppData\Local\Temp\9A26.tmp (type=wchar_t*, size=0x0)
~~2204~~ ntdll.dll!RtlDosPathNameToRelativeNtPathName_U_WithStatus
```

drltrace output DeleteFileW call, deleting 9A26.tmp in Temp Folder

Next the original `ntdll.dll` file is copied from `system32` to the temp folder with the exact name it just generated. This leads to a copy of this mentioned library being placed in the temp directory.

Property	Value
File Name	C:\Users\blackbeard\AppData\Local\Temp\9A26.tmp
File Type	Portable Executable 32
File Info	Microsoft Visual C++ vx.x DLL
File Size	1.23 MB (1292096 bytes)
PE Size	1.22 MB (1277440 bytes)
Created	Saturday 23 May 2020, 11.45.58
Modified	Sunday 21 November 2010, 05.24.01
Accessed	Saturday 23 May 2020, 11.45.58
MD5	D124F55B9393C976963407DFF51FFA79
SHA-1	2C7BBEDD79791BFB866898C85B504186DB610B5D

Meta data of disguised

Property	Value
CompanyName	Microsoft Corporation
FileDescription	NT Layer DLL
FileVersion	6.1.7601.17514 (win7sp1_rtm.101119-1850)
InternalName	ntdll.dll
LegalCopyright	© Microsoft Corporation. All rights reserved.
OriginalFilename	ntdll.dll
ProductName	Microsoft® Windows® Operating System

ntdll.dll

Ordinal	Function RVA	Name Ordinal	Name RVA	Name
N/A	00000634	N/A	N/A	N/A
(nFunctions)	Dword	Word	Dword	szAnsi
00000088	0004399A	0087	00015C70	LdrLoadAlternateResourceModuleEx
00000089	0003C43A	0088	00015C91	LdrLoadDll
0000008A	00036B95	0089	00015C9C	LdrLockLoaderLock
0000008B	00063588	008A	00015CAE	LdrOpenImageFileOptionsKey
0000008C	000AE9CF	008B	00015CC9	LdrProcessRelocationBlock
0000008D	0004C132	008C	00015CE3	LdrQueryImageFileExecutionOptions
0000008E	0004C159	008D	00015D05	LdrQueryImageFileExecutionOptionsEx
0000008F	00062FD2	008E	00015D29	LdrQueryImageFileKeyOption
00000090	000A04FE	008F	00015D44	LdrQueryModuleServiceTags
00000091	000A04D4	0090	00015D5E	LdrQueryProcessModuleInformation
00000092	0006C8A5	0091	00015D7F	LdrRegisterDllNotification
00000093	0005FAA2	0092	00015D9A	LdrRemoveLoadAsDataTable
00000094	0004E29C	0093	00015DB3	LdrResFindResource
00000095	0003DA15	0094	00015DC6	LdrResFindResourceDirectory
00000096	00047C5F	0095	00015DE2	LdrResGetRCConfig
00000097	000AEF42	0096	00015DF4	LdrResRelease
00000098	0003CD5C	0097	00015E02	LdrResSearchResource
00000099	000436DD	0098	00015E17	LdrRscIsTypeExist
0000009A	000A04F4	0099	00015E29	LdrSetAppCompatDllRedirectionCallback
0000009B	000515F6	009A	00015E4F	LdrSetDllManifestProber

Export functions of the disguised ntdll file

Instead of loading the real `ntdll.dll` file, the copy is loaded into memory by calling `LdrLoadDll`.

Name	Base address	Size	Description
axio.exe	**0x400000**	**0.98 MB**	**Arching Iil Pronunciation Pour Xp Yrks**
9A26.tmp	0x73ee0000	1.5 MB	NT Layer DLL
advapi32.dll	0x74d00000	640 kB	Advanced Windows 32 Base API
apisetschema.dll	0x40000	4 kB	ApiSet Schema DLL
avicap32.dll	0x6bb70000	76 kB	AVI Capture window class
avicap32.dll.mui	0x250000	8 kB	AVI Capture window class
clbcatq.dll	0x76c60000	524 kB	COM+ Configuration Catalog
comctl32.dll	0x6a890000	528 kB	User Experience Controls Library
comctl32.dll	0x6e2a0000	1.62 MB	User Experience Controls Library

9A26.tmp as ntdll.dll

Most AV vendors, as well as analysts probably implemented their hooks on `ntdll.dll`, so the references to the copied `ntdll.dll` file will be missed.

Smokeloader continues to call functions from this copied DLL, using for example function calls like `NtQueryInformationProcess` to detect wether a debugger is attached to it.

Final Words

While analysing SmokeLoader at work, I stumbled across this AntiHook mechanism, which I haven't seen before, so I wanted to share it here :-).

I've also only scratched on the surface of what Frida is capable of. I might work on something more complex next time.