Volatility Contest Submission

by Dima Pshoul

Motivation for writing the plugins:

As a long time volatility user I wanted to contribute to the framework and also better it by writing unique and powerful plugins.

On top of that, I wanted to find a way to cover the shortcomings of malfind and be able to detect malware using other techniques. For a long time malfind has been my go to plugin for volatility and if it didn't output any obvious pages then I would not know what to do. As my skills and experience increased I became aware of malfinds weaknesses.

For instance, one of those weaknesses is the fact that it relies on the VAD to enumerate pages. This means that pages enumerated by malfind are accounted for their initial state, therefore if a malware writer would use VirtualAllocEx with the protection constant of PAGE_READWRITE and then use VirtualProtect to change the protection constant to PAGE_EXECUTE_READWRITE, malfind would not be able to find the injected page.

With that being said, I would like to introduce you to what I call *"The Advanced Malware Hunters Kit"*.

The plugins I chose to write and submit are the following:

Malfofind (short for "Malicious File Object Find").

Callstacks.

Malthfind (short for "Malicious Thread Find").

Malfofind

Description:

This plugins purpose is to find code injected using the technique often dubbed “Process Hollowing”[1], from my experience this technique is a malware favorite.

This plugin will scan currently loaded modules (using the VAD) for each process and will check if they are all accordingly mapped in the process’ PEB. So why should this scanning technique work? If we examine the implementations of this technique in malware [2]or on github[3] we will notice how all the implementations contain the step of NtUnmapViewOfSection(ProcessHandle, ProcessImageBase) and then using VirtualAllocEx to map the injected executable into the same address. This will create an inconsistency between the VAD mapped images and the PEB linked images.

Demonstration:

Taking the second sample from this blog post[4] and running malfofind on the dump we receive the following output:

```
Process: svchost.exe Pid: 2460 Ppid: 2220
Address: 0x1c0000 Protection: PAGE_EXECUTE_READWRITE
Initially mapped file object: C:\Windows\system32\svchost.exe
Currently mapped file object: None
0x001c0000  4d 5a 90 00 03 00 00 00 04 00 00 00 ff ff 00 00   MZ..............
0x001c0010  b8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00   ........@.......
0x001c0020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0x001c0030  00 00 00 00 00 00 00 00 00 00 00 00 d8 00 00 00   ................

0x001c0000 4d              DEC EBP
0x001c0001 5a              POP EDX
0x001c0002 90              NOP
0x001c0003 0003            ADD [EBX], AL
0x001c0005 0000            ADD [EAX], AL
0x001c0007 000400          ADD [EAX+EAX], AL
0x001c000a 0000            ADD [EAX], AL
0x001c000c ff              DB 0xff
0x001c000d ff00            INC DWORD [EAX]
0x001c000f 00b800000000    ADD [EAX+0x0], BH
0x001c0015 0000            ADD [EAX], AL
0x001c0017 004000          ADD [EAX+0x0], AL
0x001c001a 0000            ADD [EAX], AL
0x001c001c 0000            ADD [EAX], AL
0x001c001e 0000            ADD [EAX], AL
0x001c0020 0000            ADD [EAX], AL
0x001c0022 0000            ADD [EAX], AL
0x001c0024 0000            ADD [EAX], AL
0x001c0026 0000            ADD [EAX], AL
0x001c0028 0000            ADD [EAX], AL
0x001c002a 0000            ADD [EAX], AL
0x001c002c 0000            ADD [EAX], AL
0x001c002e 0000            ADD [EAX], AL
0x001c0030 0000            ADD [EAX], AL
0x001c0032 0000            ADD [EAX], AL
0x001c0034 0000            ADD [EAX], AL
0x001c0036 0000            ADD [EAX], AL
0x001c0038 0000            ADD [EAX], AL
0x001c003a 0000            ADD [EAX], AL
0x001c003c d800            FADD DWORD [EAX]
0x001c003e 0000            ADD [EAX], AL
```

[1] http://www.autosectools.com/process-hollowing.pdf
[2] https://countuponsecurity.com/2015/12/07/malware-analysis-dridex-process-hollowing/
[3] https://github.com/m0n0ph1/Process-Hollowing
[4] http://journeyintoir.blogspot.co.il/2015/02/process-hollowing-meets-cuckoo-sandbox.html

Used dump file and output given in the following link:
https://drive.google.com/open?id=0B4hBibjzdaPqN1FqdEQ0eElwQ3c

Callstacks

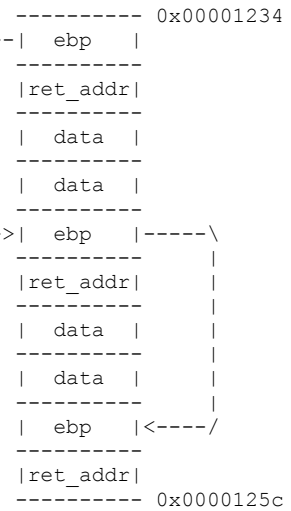
Description:

This plugins purpose is extracting the callstack for each running thread on the system. I chose to write this plugin separate of malthfind so that it could be used more easily as a class to inherit from and process it's output.

The plugin extracts the initial EBP of the thread using _KTRAP_FRAME and then walks the callstack and extracts return addresses using the following methods:

Next EBP = [EBP] (To walk the callstack)

ReturnAddress = [EBP+4] (To extract the return address of the frame)



It's important to note that this plugin and all inheriting plugins will not function properly on 64bit based windows operating systems since the fastcall[5] calling convention does not specify the creation of a stack frame as an obligatory action.

Demonstration:

Analyzing a dump with callstacks will yield the following output:

```
-snip-

ETHREAD: 0x85c902b0 Pid: 2204 Tid: 2484
Owning Process: dllhost.exe
Attached Process: dllhost.exe
Thread Flags: PS_CROSS_THREAD_FLAGS_DEADTHREAD

No.      Ebp        RetAddr    Function
[eip]    0x00000000 0x774b70b0 ntdll.dll!KiFastSystemCall+0x4
[0]      0x0157f818 0x7760bab0 kernel32.dll!WaitForSingleObjectEx+0x43
[1]      0x0157f82c 0x7760ba90 kernel32.dll!WaitForSingleObject+0x12
[2]      0x0157f840 0x6d3ef053 catsrv.dll!GetCatalogCRMClerk+0x1242
[3]      0x0157f860 0x6d3d037b catsrv.dll!DllUnregisterServer+0x7364
[4]      0x0157f898 0x771e123b msvcrt.dll!_itow_s+0x4c
[5]      0x0157f8a0 0x771e12bc msvcrt.dll!_endthreadex+0x6c
```

[5] https://msdn.microsoft.com/en-us/library/6xa169sk.aspx

```
[6]      0x0157f8ac 0x77613c33 kernel32.dll!BaseThreadInitThunk+0x12
[7]      0x0157f8ec 0x774d3706 ntdll.dll!RtlInitializeExceptionChain+0xef
[8]      0x0157f904 0x774d3706 ntdll.dll!RtlInitializeExceptionChain+0xc2


------

ETHREAD: 0x1f7f8030 Pid: 2100 Tid: 2192
Owning Process: dllhost.exe
Attached Process: dllhost.exe
Thread Flags: PS_CROSS_THREAD_FLAGS_DEADTHREAD

No.      Ebp        RetAddr    Function
[eip]    0x00000000 0x774b70b0 ntdll.dll!KiFastSystemCall+0x4
[0]      0x00cff8d0 0x6e142498 COMSVCS.DLL!CoLoadServices+0x59126
[1]      0x00cff8e0 0x6e142498 COMSVCS.DLL!CoLoadServices+0x593b2
[2]      0x00cff918 0x771e123b msvcrt.dll!_itow_s+0x4c
[3]      0x00cff920 0x771e12bc msvcrt.dll!_endthreadex+0x6c
[4]      0x00cff92c 0x77613c33 kernel32.dll!BaseThreadInitThunk+0x12
[5]      0x00cff96c 0x774d3706 ntdll.dll!RtlInitializeExceptionChain+0xef
[6]      0x00cff984 0x774d3706 ntdll.dll!RtlInitializeExceptionChain+0xc2

-snip-
```

Used dump and output can be found in the following link:

https://drive.google.com/open?id=0B4hBibjzdaPqbHZWaGwxaG9NMXc

Malthfind

Description:

This plugins purpose is to detect code injections of varied types.

The plugin uses the callstacks plugins output and analyzes it using given patterns. Currently two patterns are implemented.

The first "*MalthfindRuleDynamicallyAllocatedExecutionAddress"* will check for return addresses found in dynamically allocated code, this technique has the ability to find code injection techniques that rely on allocating executable space inside the injected process. This scan will also, if timed properly will find packed/crypted malware as they rely on allocating pages to unpack/decrypt their payload.

The second *"MalthfindRuleDllInjection"* will check if any thread has the function ntdll.dll!LdrLoadLibrary in its first eight return addresses (meaning its start address is one of the library loading family functions) and will find classic dll injection[6] techniques.

Demonstration:

Dynamic Execution (aka MalthfindRuleDynamicallyAllocatedExecutionAddress*)*:

```
-snip-


ETHREAD: 0x85979420 Pid: 1604 Tid: 3896
Owning Process: explorer.exe
Attached Process: explorer.exe
Thread Flags: PS_CROSS_THREAD_FLAGS_DEADTHREAD
Malicious patterns detected: Thread executing from dynamically allocated memory
Callstack:
        No.      Ebp        RetAddr    Function
        [eip]    0x00000000 0x774b70b0 ntdll.dll!KiFastSystemCall+0x4
        [0]      0x035df6a8 0x7760bab0 kernel32.dll!WaitForSingleObjectEx+0x43
        [1]      0x035df6bc 0x7760ba90 kernel32.dll!WaitForSingleObject+0x12
        [2]      0x035df754 0x0350cc6c Unknown!Unknown+0x0
        [3]      0x035df7e8 0x03519f96 Unknown!Unknown+0x0
        [4]      0x035df7f4 0x77613c33 kernel32.dll!BaseThreadInitThunk+0x12
        [5]      0x035df834 0x774d3706 ntdll.dll!RtlInitializeExceptionChain+0xef
        [6]      0x035df84c 0x774d3706 ntdll.dll!RtlInitializeExceptionChain+0xc2


------

ETHREAD: 0x840ced48 Pid: 1532 Tid: 3864
Owning Process: taskhost.exe
Attached Process: taskhost.exe
Thread Flags: PS_CROSS_THREAD_FLAGS_DEADTHREAD
Malicious patterns detected: Thread executing from dynamically allocated memory
Callstack:
        No.      Ebp        RetAddr    Function
        [eip]    0x00000000 0x774b70b0 ntdll.dll!KiFastSystemCall+0x4
        [0]      0x0248fa54 0x7760bab0 kernel32.dll!WaitForSingleObjectEx+0x43
        [1]      0x0248fa68 0x7760ba90 kernel32.dll!WaitForSingleObject+0x12
        [2]      0x0248faf8 0x008ea104 Unknown!Unknown+0x0
        [3]      0x0248fb04 0x77613c33 kernel32.dll!BaseThreadInitThunk+0x12
        [4]      0x0248fb44 0x774d3706 ntdll.dll!RtlInitializeExceptionChain+0xef
        [5]      0x0248fb5c 0x774d3706 ntdll.dll!RtlInitializeExceptionChain+0xc2

-snip-
```

[6] http://blog.opensecurityresearch.com/2013/01/windows-dll-injection-basics.html

We can cross reference our results to see how the same looks in malfind:

Pid 1604:

-snip-

```
Process: explorer.exe Pid: 1604 Address: 0x34f0000
Vad Tag: VadS Protection: PAGE_EXECUTE_READWRITE
Flags: CommitCharge: 66, MemCommit: 1, PrivateMemory: 1, Protection: 6


0x034f0000  4d 5a 00 00 00 00 00 00 00 00 00 00 00 00 00 00   MZ..............
0x034f0010  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0x034f0020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0x034f0030  00 00 00 00 00 00 00 00 00 00 00 00 28 01 00 00   ............(...

0x034f0000 4d              DEC EBP
0x034f0001 5a              POP EDX
0x034f0002 0000            ADD [EAX], AL
0x034f0004 0000            ADD [EAX], AL
0x034f0006 0000            ADD [EAX], AL
0x034f0008 0000            ADD [EAX], AL
0x034f000a 0000            ADD [EAX], AL
0x034f000c 0000            ADD [EAX], AL
0x034f000e 0000            ADD [EAX], AL
0x034f0010 0000            ADD [EAX], AL
0x034f0012 0000            ADD [EAX], AL
0x034f0014 0000            ADD [EAX], AL
0x034f0016 0000            ADD [EAX], AL
0x034f0018 0000            ADD [EAX], AL
0x034f001a 0000            ADD [EAX], AL
0x034f001c 0000            ADD [EAX], AL
0x034f001e 0000            ADD [EAX], AL
0x034f0020 0000            ADD [EAX], AL
0x034f0022 0000            ADD [EAX], AL
0x034f0024 0000            ADD [EAX], AL
0x034f0026 0000            ADD [EAX], AL
0x034f0028 0000            ADD [EAX], AL
0x034f002a 0000            ADD [EAX], AL
0x034f002c 0000            ADD [EAX], AL
0x034f002e 0000            ADD [EAX], AL
0x034f0030 0000            ADD [EAX], AL
0x034f0032 0000            ADD [EAX], AL
0x034f0034 0000            ADD [EAX], AL
0x034f0036 0000            ADD [EAX], AL
0x034f0038 0000            ADD [EAX], AL
0x034f003a 0000            ADD [EAX], AL
0x034f003c 2801            SUB [ECX], AL
0x034f003e 0000            ADD [EAX], AL
```

-snip-

We can see this section corresponds to the the following lines in malthfinds output according to the return addresses and the VAD start address:

```
       [2]      0x035df754 0x0350cc6c Unknown!Unknown+0x0
       [3]      0x035df7e8 0x03519f96 Unknown!Unknown+0x0
```

Pid 1532:

-snip-

```
Process: taskhost.exe Pid: 1532 Address: 0x8c0000
Vad Tag: VadS Protection: PAGE_EXECUTE_READWRITE
```

```
Flags: CommitCharge: 66, MemCommit: 1, PrivateMemory: 1, Protection: 6

0x008c0000  4d 5a 00 00 00 00 00 00 00 00 00 00 00 00 00 00   MZ..............
0x008c0010  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0x008c0020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0x008c0030  00 00 00 00 00 00 00 00 00 00 00 00 28 01 00 00   ............(...

0x008c0000 4d              DEC EBP
0x008c0001 5a              POP EDX
0x008c0002 0000            ADD [EAX], AL
0x008c0004 0000            ADD [EAX], AL
0x008c0006 0000            ADD [EAX], AL
0x008c0008 0000            ADD [EAX], AL
0x008c000a 0000            ADD [EAX], AL
0x008c000c 0000            ADD [EAX], AL
0x008c000e 0000            ADD [EAX], AL
0x008c0010 0000            ADD [EAX], AL
0x008c0012 0000            ADD [EAX], AL
0x008c0014 0000            ADD [EAX], AL
0x008c0016 0000            ADD [EAX], AL
0x008c0018 0000            ADD [EAX], AL
0x008c001a 0000            ADD [EAX], AL
0x008c001c 0000            ADD [EAX], AL
0x008c001e 0000            ADD [EAX], AL
0x008c0020 0000            ADD [EAX], AL
0x008c0022 0000            ADD [EAX], AL
0x008c0024 0000            ADD [EAX], AL
0x008c0026 0000            ADD [EAX], AL
0x008c0028 0000            ADD [EAX], AL
0x008c002a 0000            ADD [EAX], AL
0x008c002c 0000            ADD [EAX], AL
0x008c002e 0000            ADD [EAX], AL
0x008c0030 0000            ADD [EAX], AL
0x008c0032 0000            ADD [EAX], AL
0x008c0034 0000            ADD [EAX], AL
0x008c0036 0000            ADD [EAX], AL
0x008c0038 0000            ADD [EAX], AL
0x008c003a 0000            ADD [EAX], AL
0x008c003c 2801            SUB [ECX], AL
0x008c003e 0000            ADD [EAX], AL


-snip-
```

We can see this section corresponds to the following line in malthfinds output according to the return address and the VAD start address:

```
        [2]      0x0248faf8 0x008ea104 Unknown!Unknown+0x0
```

The memory sample and output can be found here:

https://drive.google.com/open?id=0B4hBibjzdaPqbHZWaGwxaG9NMXc

Dll Injection (aka MalthfindRuleDllInjection):

```
------

ETHREAD: 0x85c19580 Pid: 2620 Tid: 3672
Owning Process: cmd.exe
Attached Process: cmd.exe
Thread Flags: PS_CROSS_THREAD_FLAGS_DEADTHREAD
Malicious patterns detected: Dll injection pattern
Callstack:
        No.      Ebp        RetAddr    Function
        [eip]    0x00000000 0x77bf70b0 ntdll.dll!KiFastSystemCall+0x4
        [0]      0x0246f3e4 0x76dd3588 USER32.dll!DrawStateW+0x59f
```

```
[1]      0x0246f488 0x76dfda4b USER32.dll!SoftModalMessageBox+0x68a
[2]      0x0246f5d8 0x76dfda4b USER32.dll!SoftModalMessageBox+0xc0e
[3]      0x0246f640 0x76dfe70d USER32.dll!MessageBoxTimeoutW+0x7f
[4]      0x0246f674 0x76dfe795 USER32.dll!MessageBoxTimeoutA+0xa1
[5]      0x0246f694 0x76dfe9c9 USER32.dll!MessageBoxExA+0x1b
[6]      0x0246f6b0 0x76dfea11 USER32.dll!MessageBoxA+0x45
[7]      0x0246f6c8 0x7487101c injd_dll.dll!Unknown+0x0
[8]      0x0246f708 0x7487123b injd_dll.dll!Unknown+0x0
[9]      0x0246f71c 0x748711c2 injd_dll.dll!Unknown+0x0
[10]     0x0246f73c 0x77c0898c ntdll.dll!wcsncmp+0x4c
[11]     0x0246f830 0x77c15b0c ntdll.dll!EtwEventRegister+0x135
[12]     0x0246f99c 0x77c1011d ntdll.dll!LdrUnlockLoaderLock+0x411
[13]     0x0246f9d0 0x77c122b8 ntdll.dll!LdrLoadDll+0x74
[14]     0x0246fa08 0x75e3883a KERNELBASE.dll!FreeLibrary+0xb4
[15]     0x0246fa28 0x75e38d66 KERNELBASE.dll!LoadLibraryExA+0x26
[16]     0x0246fa48 0x76f5395c kernel32.dll!LoadLibraryA+0x32
[17]     0x0246fa54 0x76f53c33 kernel32.dll!BaseThreadInitThunk+0x12
[18]     0x0246fa94 0x77c13706 ntdll.dll!RtlInitializeExceptionChain+0xef
[19]     0x0246faac 0x77c13706 ntdll.dll!RtlInitializeExceptionChain+0xc2
```

The memory sample and output can be found here:

https://drive.google.com/open?id=0B4hBibjzdaPqQ0VFbDNKS09SaTQ

Using the plugins:

Place the three plugins in volatility-master\volatility\plugins\malware and run volatility, the plugins will be available now.

Plugins download:

https://drive.google.com/open?id=0B4hBibjzdaPqT3lYZE50QlBudWM

Why this submission should win:

My submission should win the contest for the following reasons:

1. The plugins are core plugins for malware detection, I usually place volatility plugins in two groups, one being the “initial detection”, containing malfind, apihooks and such. The other being “investigation” plugins, containing pslist, vadinfo and such. In my opinion the first group of plugins is more crucial than the latter as it gives you a place to start looking in, and I believe the plugins malfofind and malthfind both fall under the description of the first group.

2. The plugins are innovative. Before deciding upon writing the code powering the plugins I have checked to see that this logic has not been written or implemented in other tools.

3. Strengthen the frameworks weaknesses. The plugins are written with knowledge of the frameworks weaknesses and strengths, therefore they are written to compliment the framework at its weakest points.

4. The plugins are relevant to the current state of malware. They are written with knowledge of current trends in the malware writing world. The malfofind plugin detects the widely popular and used Process Hollowing.

5. Low false positive rates. The plugins were meticulously tested and verified to detect only malicious behavior.

6. The plugins are designed with modularity in mind. As an example I would like to propose the patterns introduced in the malthfind plugin. The plugins detection patterns are easily modifiable and easily added to or removed from.

Thanks:

Thanks for the time you took to read and evaluate my submission, I have enjoyed delving into the source code of volatility and writing a plugin to augment the framework.