Analysis of Oreans Themida (x86), Version 2.3.5.10, Anti-Debugger Detections

17 min read · Oct 8, 2023

Reversing Follow

Written 27/AUG/2020

Anti-AI Notice

> *I do not support or condone the use of artificial intelligence (AI) tools in the creation or distribution of any content I produce. All work published by me is made by a human, and I will not grant permission for my content, artwork, writing, or any related materials to be used in training or generating AI models.*
>
> *I request that no one uploads, scrapes, or utilizes my work in any AI datasets or projects. Any violation of this request goes against my explicit wishes.*
>
> *Thank you for respecting human creativity.*

Disclaimer

This write-up is focused on the analysis of an executable protected with Oreans Themida 2.3.5.10 (x86). The documentation provided may be changed at any given time.

Assumptions

Going into the analysis of Themida's anti-debugger detections we must first assume the following:

- We were able to reach the REAL OEP.
- We have not yet “flagged” the target process.
- We have access to the Oreans sections `osvpmwup` and `mohgxlku`.
- We have the ability to access the process threads.
- We have the ability to break and trace the process.

The following address table will also be referenced in this write-up.

Address	Size	File	PE Section	Type	Protection	State
00400000	00001000	antidebug.exe		IMAGE	READONLY	COMMIT
00401000	00002000	antidebug.exe		IMAGE	EXECUTE_READWRITE	COMMIT
00403000	00001000	antidebug.exe	|.rsrc	IMAGE	WRITECOPY	COMMIT
00404000	00001000	antidebug.exe	.rsrc |.idata	IMAGE	READWRITE	COMMIT
00405000	0000B000	antidebug.exe	.idata |osvpmwup	IMAGE	EXECUTE_READWRITE	COMMIT
00410000	00028000	antidebug.exe	osvpmwup	IMAGE	EXECUTE_WRITECOPY	COMMIT
00438000	00002000	antidebug.exe	osvpmwup	IMAGE	EXECUTE_READWRITE	COMMIT
0043A000	00002000	antidebug.exe	osvpmwup	IMAGE	EXECUTE_WRITECOPY	COMMIT
0043C000	0000B000	antidebug.exe	osvpmwup	IMAGE	EXECUTE_READWRITE	COMMIT
00447000	00003000	antidebug.exe	osvpmwup	IMAGE	EXECUTE_WRITECOPY	COMMIT
0044A000	00011000	antidebug.exe	osvpmwup	IMAGE	EXECUTE_READWRITE	COMMIT
0045B000	00001000	antidebug.exe	osvpmwup	IMAGE	EXECUTE_WRITECOPY	COMMIT
0045C000	0000D000	antidebug.exe	osvpmwup	IMAGE	EXECUTE_READWRITE	COMMIT
00469000	00013000	antidebug.exe	osvpmwup	IMAGE	EXECUTE_WRITECOPY	COMMIT
0047C000	0006A000	antidebug.exe	osvpmwup	IMAGE	EXECUTE_READWRITE	COMMIT
004E6000	00004000	antidebug.exe	osvpmwup	IMAGE	EXECUTE_WRITECOPY	COMMIT
004EA000	00001000	antidebug.exe	osvpmwup|mohgxlku	IMAGE	EXECUTE_READWRITE	COMMIT
004EB000	00005000		mohgxlku	UNDEF	UNDEF	FREE

Now, we must also consider the methods that could be used to execute anti-debugging. We must assume:

- **PREFIX** — Walking the unpacking process uses checks like PEB and STI to check for signs of debugging. Before reaching the REAL OEP new “anti-debugger” threads are created upon entry. Threads then use obfuscated recursive sleep looping to check for windows, classes, drivers, and flags.
- **POSTFIX** — Upon reaching the real OEP, existing “anti-debugger” threads have been executed and are running co-currently with the process.

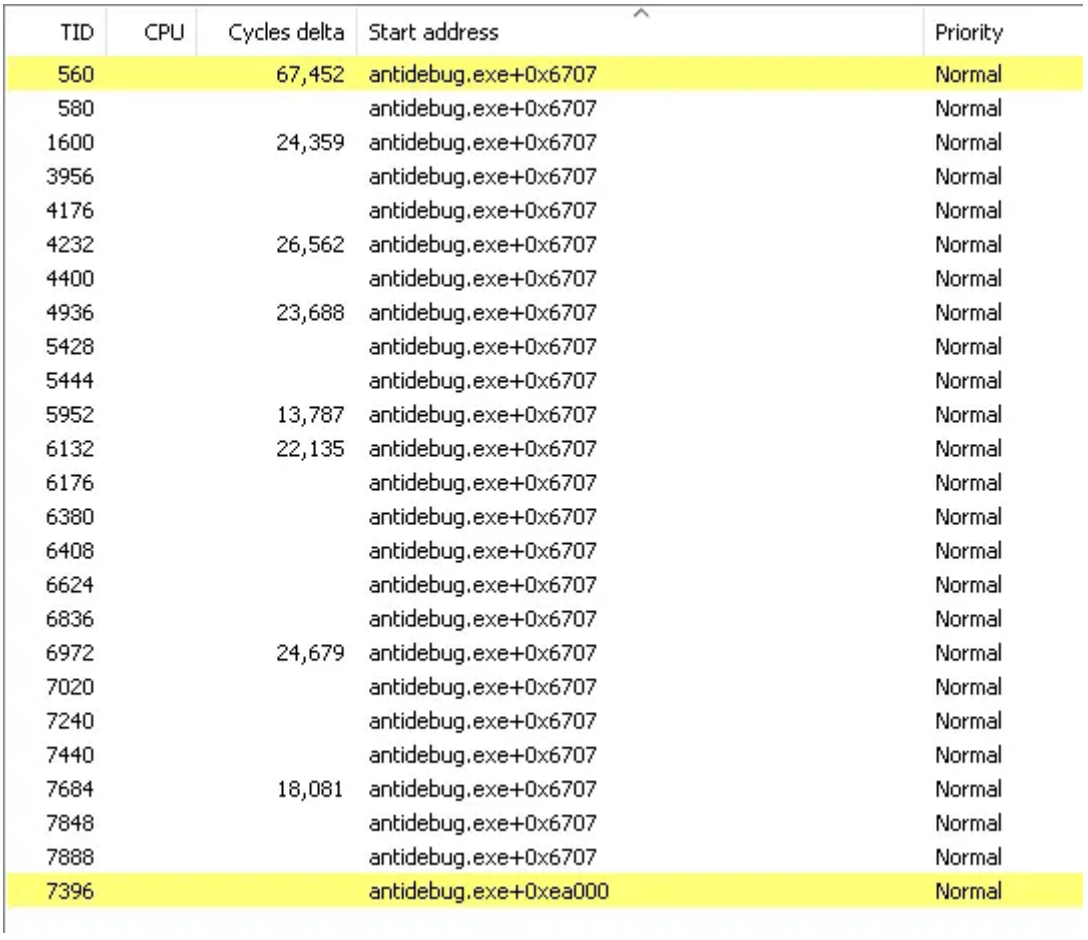
Both methods will end up accessing the same code regions used by the threads. The choice of applying either method is up to the reader depending on their own

approaches.

For this analysis, I will be using the **POSTFIX** approach as I find it would be better for the reader and also easier to digest at this time. The **PREFIX** approach will also be covered once further write-ups are reviewed and ready to be released as some of those topics are very dependent on each other.

Threading

Upon starting the process, several threads can be seen already running. The number of threads will vary for each process. Process Hacker can be used to help identify threads.



TID	CPU	Cycles delta	Start address	Priority
560		67,452	antidebug.exe+0x6707	Normal
580			antidebug.exe+0x6707	Normal
1600		24,359	antidebug.exe+0x6707	Normal
3956			antidebug.exe+0x6707	Normal
4176			antidebug.exe+0x6707	Normal
4232		26,562	antidebug.exe+0x6707	Normal
4400			antidebug.exe+0x6707	Normal
4936		23,688	antidebug.exe+0x6707	Normal
5428			antidebug.exe+0x6707	Normal
5444			antidebug.exe+0x6707	Normal
5952		13,787	antidebug.exe+0x6707	Normal
6132		22,135	antidebug.exe+0x6707	Normal
6176			antidebug.exe+0x6707	Normal
6380			antidebug.exe+0x6707	Normal
6408			antidebug.exe+0x6707	Normal
6624			antidebug.exe+0x6707	Normal
6836			antidebug.exe+0x6707	Normal
6972		24,679	antidebug.exe+0x6707	Normal
7020			antidebug.exe+0x6707	Normal
7240			antidebug.exe+0x6707	Normal
7440			antidebug.exe+0x6707	Normal
7684		18,081	antidebug.exe+0x6707	Normal
7848			antidebug.exe+0x6707	Normal
7888			antidebug.exe+0x6707	Normal
7396			antidebug.exe+0xea000	Normal

Searching directly for the thread start addresses we are greeted with the following.

```
antidebug.exe+0x6707:

004066DD - B8 00000000           - mov eax,00000000
004066E2 - 85 C0                 - test eax,eax
004066E4 - 74 03                 - je 004066E9
```

```
004066E6 - C2 1800              - ret 0018

004066E9 - 8B 44 24 0C          - mov eax,[esp+0C]
004066ED - 53                   - push ebx
004066EE - E8 00000000          - call 004066F3
004066F3 - 5B                   - pop ebx
004066F4 - 83 C3 14             - add ebx,14
004066F7 - 89 5C 24 10          - mov [esp+10],ebx
004066FB - 89 44 24 14          - mov [esp+14],eax
004066FF - 5B                   - pop ebx
00406700 - B8 801A1777          - mov eax, KERNEL32.CreateThread
00406705 - FF E0                - jmp eax

00406707 - 8B 44 24 04          - mov eax,[esp+04]     ; Start Address
0040670B - FF E0                - jmp eax
```

```
antidebug.exe+0xea000:

004EA000 - 56                   - push esi             ; Start Address
004EA001 - 50                   - push eax
004EA002 - 53                   - push ebx
004EA003 - E8 01000000          - call 004EA009
004EA008 - 00
004EA009 - 58                   - pop eax
004EA00A - 89 C3                - mov ebx,eax
004EA00C - 40                   - inc eax
004EA00D - 2D 00500E00          - sub eax,000E5000
004EA012 - 2D 8CAC0B10          - sub eax,100BAC8C
004EA017 - 05 83AC0B10          - add eax,100BAC83
004EA01C - 80 3B CC             - cmp byte ptr [ebx],-34
004EA01F - 75 19                - jne 004EA03A
004EA021 - C6 03 00             - mov byte ptr [ebx],00
004EA024 - BB 00100000          - mov ebx,00001000
004EA029 - 68 923A7E1F          - push 1F7E3A92
004EA02E - 68 00E4C15A          - push 5AC1E400
004EA033 - 53                   - push ebx
004EA034 - 50                   - push eax
004EA035 - E8 0A000000          - call 004EA044

004EA03A - 83 C0 14             - add eax,14
004EA03D - 89 44 24 08          - mov [esp+08],eax
004EA041 - 5B                   - pop ebx
004EA042 - 58                   - pop eax
004EA043 - C3                   - ret

004EA044 - 55                   - push ebp
004EA045 - 89 E5                - mov ebp,esp
004EA047 - 50                   - push eax
004EA048 - 53                   - push ebx
```

```
004EA049 - 51                  - push ecx
004EA04A - 56                  - push esi
004EA04B - 8B 75 08            - mov esi,[ebp+08]
004EA04E - 8B 4D 0C            - mov ecx,[ebp+0C]
004EA051 - C1 E9 02            - shr ecx,02
004EA054 - 8B 45 10            - mov eax,[ebp+10]
004EA057 - 8B 5D 14            - mov ebx,[ebp+14]

004EA05A - 85 C9               - test ecx,ecx
004EA05C - 74 0A               - je 004EA068
004EA05E - 31 06               - xor [esi],eax
004EA060 - 01 1E               - add [esi],ebx
004EA062 - 83 C6 04            - add esi,04
004EA065 - 49                  - dec ecx
004EA066 - EB F2               - jmp 004EA05A

004EA068 - 5E                  - pop esi
004EA069 - 59                  - pop ecx
004EA06A - 5B                  - pop ebx
004EA06B - 58                  - pop eax
004EA06C - C9                  - leave
004EA06D - C2 1000             - ret 0010
```

Referencing from the address table provide we are able to locate the segments for these two starting addresses at `00405000` located within the extracted segment of Oreans and `004EA000` located at the starting stub segment. The main process thread for this sample is `antidebug.exe+0xea000` as the sample itself is not multi-threaded. Even though the starting address points to the Themida OEP, it itself is no longer associated with the “unpacking” stage at this given point in time. However, the start address threads `antidebug.exe+0x6707`, associated within the blocks `004066DD` to `0040670B` are children from the “unpacking” stage produced from the main process thread `antidebug.exe+0xea000`. Break pointing within the blocks `004066DD` to `0040670B` at this stage is useless as they already are running elsewhere.

Now the question arises: Where to go from here?

Dynamic Thread Analysis

Sorting the threads by cycles we notice there are currently eight “plausible” anti-debugging threads running in this sample.

TID	CPU	Cycles delta	Start address	Priority
8084			antidebug.exe+0x6707	Normal
7440			antidebug.exe+0x6707	Normal
7380			antidebug.exe+0x6707	Normal
7368			antidebug.exe+0x6707	Normal
7192			antidebug.exe+0x6707	Normal
6956			antidebug.exe+0x6707	Normal
6464			antidebug.exe+0x6707	Normal
6300			antidebug.exe+0x6707	Normal

Using API Monitor we are able to "visually map" out the "plausible" threads.

Thread #1

Module	API
KERNELBASE.dll	NtDelayExecution (FALSE, 0x028bff34)
KERNELBASE.dll	NtDelayExecution (FALSE, 0x028bff34)
KERNELBASE.dll	NtDelayExecution (FALSE, 0x028bff34)
KERNELBASE.dll	NtDelayExecution (FALSE, 0x028bff34)
KERNELBASE.dll	NtDelayExecution (FALSE, 0x028bff34)
KERNELBASE.dll	NtDelayExecution (FALSE, 0x028bff34)
KERNELBASE.dll	NtDelayExecution (FALSE, 0x028bff34)
KERNELBASE.dll	NtDelayExecution (FALSE, 0x028bff34)
KERNELBASE.dll	NtDelayExecution (FALSE, 0x028bff34)

Thread #2

Module	API
KERNELBASE.dll	NtDelayExecution (FALSE, 0x02b3ff34)
KERNELBASE.dll	NtDelayExecution (FALSE, 0x02b3ff34)
KERNELBASE.dll	NtDelayExecution (FALSE, 0x02b3ff34)
KERNELBASE.dll	NtDelayExecution (FALSE, 0x02b3ff34)
KERNELBASE.dll	NtDelayExecution (FALSE, 0x02b3ff34)
KERNELBASE.dll	NtDelayExecution (FALSE, 0x02b3ff34)
KERNELBASE.dll	NtDelayExecution (FALSE, 0x02b3ff34)
KERNELBASE.dll	NtDelayExecution (FALSE, 0x02b3ff34)
KERNELBASE.dll	NtDelayExecution (FALSE, 0x02b3ff34)

Thread #3

Module	API
KERNELBASE.dll	NtDelayExecution (FALSE, 0x0277ff34)
KERNELBASE.dll	NtDelayExecution (FALSE, 0x0277ff34)
KERNELBASE.dll	NtDelayExecution (FALSE, 0x0277ff34)
KERNELBASE.dll	NtDelayExecution (FALSE, 0x0277ff34)
KERNELBASE.dll	NtDelayExecution (FALSE, 0x0277ff34)
KERNELBASE.dll	NtDelayExecution (FALSE, 0x0277ff34)
KERNELBASE.dll	NtDelayExecution (FALSE, 0x0277ff34)
KERNELBASE.dll	NtDelayExecution (FALSE, 0x0277ff34)
KERNELBASE.dll	NtDelayExecution (FALSE, 0x0277ff34)

Thread #4

Module	API
KERNELBASE.dll	NtDelayExecution (FALSE, 0x02c7ff34)
KERNELBASE.dll	NtDelayExecution (FALSE, 0x02c7ff34)
KERNELBASE.dll	NtDelayExecution (FALSE, 0x02c7ff34)
KERNELBASE.dll	NtDelayExecution (FALSE, 0x02c7ff34)
KERNELBASE.dll	NtDelayExecution (FALSE, 0x02c7ff34)
KERNELBASE.dll	NtDelayExecution (FALSE, 0x02c7ff34)
KERNELBASE.dll	NtDelayExecution (FALSE, 0x02c7ff34)
KERNELBASE.dll	NtDelayExecution (FALSE, 0x02c7ff34)
KERNELBASE.dll	NtDelayExecution (FALSE, 0x02c7ff34)

Thread #5

Module	API
USER32.dll	RtlAllocateHeap (0x00620000, 0, 16)
USER32.dll	RtlMultiByteToUnicodeN (0x00648b08, 16, 0x024fff14, "OLLYDBG", 8)
USER32.dll	RtlFreeHeap (0x00620000, 0, 0x00648b08)
USER32.dll	RtlAllocateHeap (0x00620000, 0, 16)
USER32.dll	RtlMultiByteToUnicodeN (0x00648bc8, 16, 0x024fff14, "GBDYLLO", 8)
USER32.dll	RtlFreeHeap (0x00620000, 0, 0x00648bc8)
USER32.dll	RtlAllocateHeap (0x00620000, 0, 16)
USER32.dll	RtlMultiByteToUnicodeN (0x00648b38, 16, 0x024fff14, "pediy06", 8)
USER32.dll	RtlFreeHeap (0x00620000, 0, 0x00648b38)
KERNELBASE.dll	NtDelayExecution (FALSE, 0x024fff34)

Thread #6

Module	API
KERNELBASE.dll	NtDelayExecution (FALSE, 0x0263ff34)
KERNELBASE.dll	NtDelayExecution (FALSE, 0x0263ff34)
KERNELBASE.dll	NtDelayExecution (FALSE, 0x0263ff34)
KERNELBASE.dll	NtDelayExecution (FALSE, 0x0263ff34)
KERNELBASE.dll	NtDelayExecution (FALSE, 0x0263ff34)
KERNELBASE.dll	NtDelayExecution (FALSE, 0x0263ff34)
KERNELBASE.dll	NtDelayExecution (FALSE, 0x0263ff34)
KERNELBASE.dll	NtDelayExecution (FALSE, 0x0263ff34)
KERNELBASE.dll	NtDelayExecution (FALSE, 0x0263ff34)

Thread #7

Module	API
KERNELBASE.dll	NtDelayExecution (FALSE, 0x029fff34)
KERNELBASE.dll	NtDelayExecution (FALSE, 0x029fff34)
KERNELBASE.dll	NtDelayExecution (FALSE, 0x029fff34)
KERNELBASE.dll	NtDelayExecution (FALSE, 0x029fff34)
KERNELBASE.dll	NtDelayExecution (FALSE, 0x029fff34)
KERNELBASE.dll	NtDelayExecution (FALSE, 0x029fff34)
KERNELBASE.dll	NtDelayExecution (FALSE, 0x029fff34)
KERNELBASE.dll	NtDelayExecution (FALSE, 0x029fff34)
KERNELBASE.dll	NtDelayExecution (FALSE, 0x029fff34)

Thread #8

Module	API
KERNELBASE.dll	NtDelayExecution (FALSE, 0x023bff34)
KERNELBASE.dll	NtDelayExecution (FALSE, 0x023bff34)
KERNELBASE.dll	NtDelayExecution (FALSE, 0x023bff34)
KERNELBASE.dll	NtDelayExecution (FALSE, 0x023bff34)
KERNELBASE.dll	NtDelayExecution (FALSE, 0x023bff34)
KERNELBASE.dll	NtDelayExecution (FALSE, 0x023bff34)
KERNELBASE.dll	NtDelayExecution (FALSE, 0x023bff34)
KERNELBASE.dll	NtDelayExecution (FALSE, 0x023bff34)
KERNELBASE.dll	NtDelayExecution (FALSE, 0x023bff34)

Thread #5, is the most suspicious and should be analyzed first. Following the API call stack we can see USER32.FindWindowA is being called to check against a block list.

#	Module	Address	Offset	Location
1	USER32.dll	0x753b55a0	0x255a0	GetClassInfoExA + 0x230
2	USER32.dll	0x753b52fe	0x252fe	GetClassInfoA + 0x7e
3	USER32.dll	0x753b4f96	0x24f96	FindWindowA + 0x16
4	antidebug.exe	0x00463be9	0x63be9	

There are a couple of ways to get around this situation.

One way would be to simply ret USER32.FindWindowA and let the KERNEL32.Sleep continues in the background. However, it is an invalid solution as it can create issues for the application itself.

```
USER32.FindWindowA:
xor eax, eax
ret
```

The approach that will be documented in this write-up will be using an alternative method that avoids creating additional complications.

Looking at 0x00463BE9 and 0x0046395a from the call stacks we can make the safe

assumption that any obscure ```call dword ptr [ebp+11FE27A2]``` is a call to USER32.FindWindowA and ```call dword ptr [ebp+11FE20CB]``` is a call to KERNEL32.Sleep. Further information regarding how Oreans creates, uses, and stores information with ```dword ptr [ebp+X]``` for “plain-sight” API and “variable” obfuscation will be covered in future documentation.

```
00463BD9 - E9 05000000           - jmp 00463BE3
...
00463BE3 - FF 95 A227FE11        - call dword ptr [ebp+11FE27A2] ; USER32.Fi
00463BE9 - 81 C6 3CBA5F02        - add esi,025FBA3C
00463BEF - 8D 8D E908FE11        - lea ecx,[ebp+11FE08E9]
00463BF5 - 23 B5 FD29FE11        - and esi,[ebp+11FE29FD]
00463BFB - C7 01 00000000        - mov [ecx],00000000
00463C01 - 6A 00                 - push 00
00463C03 - 55                    - push ebp
00463C04 - E8 03000000           - call 00463C0C
```

```
00463952 - 8B F7                 - mov esi,edi
00463954 - FF 95 CB20FE11        - call dword ptr [ebp+11FE20CB] ; KERNEL32.
0046395A - 90                    - nop
0046395B - 40                    - inc eax
0046395C - 6A 00                 - push 00
0046395E - 52                    - push edx
0046395F - E8 03000000           - call 00463967
```

Analysis by breaking and tracing from 00463BE3 onwards leads to a mess nested of obscurity.

```
                        00463BE3 - call dword ptr [ebp+11FE27A2] ; USER32.Fi
                        00463BE9 - add esi,025FBA3C
                        00463BEF - lea ecx,[ebp+11FE08E9]
                        00463BF5 - and esi,[ebp+11FE29FD]
                        00463BFB - mov [ecx],00000000
                        00463C01 - push 00
                        00463C03 - push ebp
                        00463C04 - call 00463C0C
                          00463C0C - pop ebp
                          00463C0D - mov [esp+04],ebp
                          00463C11 - add [esp+04],00000017
                          00463C19 - inc ebp
```

```
      00463C1A - push ebp
      00463C1B - ret
    00463C0A - pop ebp
    00463C0B - ret
  00463C20 - pushad
  00463C21 - xor eax,5DFCBF81
  00463C27 - mov bh,3B
  00463C29 - popad
  00463C2A - sub esi,[ebp+11FE2F40]
  00463C30 - mov [ecx+04],00000000
  00463C37 - push 00
  00463C39 - push edx
  00463C3A - call 00463C42
    00463C42 - pop edx
    00463C43 - mov [esp+04],edx
    00463C47 - add [esp+04],0000001A
    00463C4F - inc edx
    00463C50 - push edx
    00463C51 - ret
  00463C40 - pop edx
  00463C41 - ret
00463C59 - call 00463C71
  00463C71 - mov edi,3668C040
  00463C76 - pop edi
  00463C77 - or eax,eax
  00463C79 - je 00463CDD
  00463CDD - lea ecx,[ebp+11FE08E9]
  00463CE3 - jg 00463CE9
  00463CE9 - mov [ecx],44AA727C
  00463CEF - push 00
  00463CF1 - push ebp
  00463CF2 - call 00463CFA
    00463CFA - pop ebp
    00463CFB - mov [esp+04],ebp
    00463CFF - add [esp+04],00000014
    00463D07 - inc ebp
    00463D08 - push ebp
    00463D09 - ret
  00463CF8 - pop ebp
  00463CF9 - ret
00463D0B - mov esi,edi
00463D0D - pushad
00463D0E - call 00463D1B
  00463D1B - call 00463D2D
    00463D2D - pop eax
    00463D2E - pop edx
    00463D2F - movzx edx,ax
    00463D32 - popad
    00463D33 - add [ecx],1499CFCB
    00463D39 - push 00
    00463D3B - push edi
    00463D3C - call 00463D44
```

```
        00463D44 - pop edi
        00463D45 - mov [esp+04],edi
        00463D49 - add [esp+04],00000018
        00463D51 - inc edi
        00463D52 - push edi
        00463D53 - ret
      00463D42 - pop edi
      00463D43 - ret
    00463D59 - xor edi,[ebp+11FE1CC8]
    00463D5F - mov [ecx+04],CE78753C
    00463D66 - push 00
    00463D68 - push edi
    00463D69 - call 00463D71
      00463D71 - pop edi
      00463D72 - mov [esp+04],edi
      00463D76 - add [esp+04],00000014
      00463D7E - inc edi
      00463D7F - push edi
      00463D80 - ret
    00463D6F - pop edi
    00463D70 - ret
00463D82 - push edx
00463D83 - mov [ebp+11FE2833],edi
00463D89 - pop edi
00463D8A - add [ecx+04],31D6D710
00463D91 - ja 00463DB1
00463D97 - call 00463DAA
  00463DAA - sub edi,4954DDD2
  00463DB0 - pop edi
  00463DB1 - push 00
  00463DB3 - push 00
  00463DB5 - push edi
  00463DB6 - call 00463DBE
    00463DBE - pop edi
    00463DBF - mov [esp+04],edi
    00463DC3 - add [esp+04],00000016
    00463DCB - inc edi
    00463DCC - push edi
    00463DCD - ret
  00463DBC - pop edi
  00463DBD - ret
00463DD1 - mov edi,[ebp+11FE1F52]
00463DD7 - mov si,B9AE
00463DDB - lea eax,[ebp+11FE08E9]
00463DE1 - mov si,dx
00463DE4 - push eax
00463DE5 - jno 00463DF9
00463DF9 - mov [esp],eax
00463DFC - jno 00463E08
00463E08 - call dword ptr [ebp+11FE27A2] ; USER32.FindWi
00463E0E - mov [ebp+11FE22F7],esi
00463E14 - mov [ebp+12026754],ebx
```

```
      00463E1A - lea ecx,[ebp+11FE08E9]
      00463E20 - push 00
      00463E22 - push ecx
      00463E23 - call 00463E2B
        00463E2B - pop ecx
        00463E2C - mov [esp+04],ecx
        00463E30 - add [esp+04],00000017
        00463E38 - inc ecx
        00463E39 - push ecx
        00463E3A - ret
      00463E29 - pop ecx
      00463E2A - ret
    00463E3F - adc si,45B2
    00463E44 - mov [ecx],00000000
    00463E4A - xor di,E35F
    00463E4F - mov [ecx+04],00000000
    00463E56 - mov esi,[ebp+11FE29C1]
    00463E5C - or eax,eax
    00463E5E - je 00463E94
    00463E94 - lea ecx,[ebp+11FE08E9]
    00463E9A - jp 00463EA6
    00463EA6 - mov [ecx],54CA95A5
    00463EAC - push 00
    00463EAE - push edx
    00463EAF - call 00463EB7
      00463EB7 - pop edx
      00463EB8 - mov [esp+04],edx
      00463EBC - add [esp+04],0000001A
      00463EC4 - inc edx
      00463EC5 - push edx
      00463EC6 - ret
    00463EB5 - pop edx
    00463EB6 - ret
  00463ECE - jmp 00463EDE
  00463EDE - add [ecx],1499CFCB
  00463EE4 - push 00
  00463EE6 - push edx
  00463EE7 - call 00463EEF
    00463EEF - pop edx
    00463EF0 - mov [esp+04],edx
    00463EF4 - add [esp+04],00000014
    00463EFC - inc edx
    00463EFD - push edx
    00463EFE - ret
  00463EED - pop edx
  00463EEE - ret
00463F00 - mov [ecx+04],CE5F5969
00463F07 - or di,00A2
00463F0C - add [ecx+04],31D6D710
00463F13 - or [ebp+11FE245E],eax
00463F19 - push 00
00463F1B - mov si,ax
```

```
    00463F1E - lea eax,[ebp+11FE08E9]
    00463F24 - or [ebp+11FE26A9],edx
    00463F2A - nop
    00463F2B - pushad
    00463F2C - push eax
    00463F2D - push edx
    00463F2E - ja 00463F34
    00463F34 - rdtsc
    00463F36 - push eax
    00463F37 - pop eax
    00463F38 - pop edx
    00463F39 - pop eax
    00463F3A - jmp 00463F51
    00463F51 - popad
    00463F52 - push eax
    00463F53 - jmp 00463F61
    00463F61 - nop
    00463F62 - push 00
    00463F64 - push ebx
    00463F65 - call 00463F6D
      00463F6D - pop ebx
      00463F6E - mov [esp+04],ebx
      00463F72 - add [esp+04],00000019
      00463F7A - inc ebx
      00463F7B - push ebx
      00463F7C - ret
    00463F6B - pop ebx
    00463F6C - ret
  00463F83 - call dword ptr [ebp+11FE27A2] ; USER32.FindWindowA
  00463F89 - push 00
  00463F8B - push ecx
  00463F8C - call 00463F94
    00463F94 - pop ecx
    00463F95 - mov [esp+04],ecx
    00463F99 - add [esp+04],00000017
    00463FA1 - inc ecx
    00463FA2 - push ecx
    00463FA3 - ret
  00463F92 - pop ecx
  00463F93 - ret
00463FA8 - push ebx
00463FA9 - mov [ebp+11FE1EBA],edx
00463FAF - pop esi
00463FB0 - mov si,bx
00463FB3 - lea ecx,[ebp+11FE08E9]
00463FB9 - push 00
00463FBB - push ebx
00463FBC - call 00463FC4
  00463FC4 - pop ebx
  00463FC5 - mov [esp+04],ebx
  00463FC9 - add [esp+04],0000001A
  00463FD1 - inc ebx
```

```
      00463FD2 - push ebx
      00463FD3 - ret
    00463FC2 - pop ebx
    00463FC3 - ret
  00463FDB - or di,5920
  00463FE0 - mov [ebp+11FE1CE3],ebx
  00463FE6 - mov [ecx],00000000
  00463FEC - mov [ebp+12019E53],edx
  00463FF2 - mov [ecx+04],00000000
  00463FF9 - push 00
  00463FFB - push ecx
  00463FFC - call 00464004
    00464004 - pop ecx
    00464005 - mov [esp+04],ecx
    00464009 - add [esp+04],00000016
    00464011 - inc ecx
    00464012 - push ecx
    00464013 - ret
  00464002 - pop ecx
  00464003 - ret
00464017 - or eax,eax
00464019 - je 0046407A
0046407A - add ecx,edx
0046407C - jmp 0046327F
0046327F - call 00463284
  00463284 - pop ebp
  00463285 - sub ebp,1203E270
  0046328B - mov esi,[ebp+11FE27EA] ; [ebp+11FE27EA] = [004077FE] = 004
  00463291 - cmp [ebp+11FE2CDF],000007D0 ; Delay Time
  0046329B - jbe 004633F9
  004632A1 - push 00
  004632A3 - push edi
  004632A4 - call 004632AC
    004632AC - pop edi
    004632AD - mov [esp+04],edi
    004632B1 - add [esp+04],00000018
    004632B9 - inc edi
    004632BA - push edi
    004632BB - ret
  004632AA - pop edi
  004632AB - ret
004632C1 - jmp 004632CF
004632CF - mov ebx,[ebp+11FE0921]
004632D5 - push esi
004632D6 - jp 004632DE
004632DE - pop edi
004632DF - and ebx,00000100
004632E5 - push 00
004632E7 - push ebp
004632E8 - call 004632F0
  004632F0 - pop ebp
  004632F1 - mov [esp+04],ebp
```

```
          004632F5 - add [esp+04],00000016
          004632FD - inc ebp
          004632FE - push ebp
          004632FF - ret
        004632EE - pop ebp
        004632EF - ret
      00463303 - mov [ebp+11FE2EAE],ebx
      00463309 - test ebx,ebx
      0046330B - jne 0046336A
      0046336A - mov [ebp+1203E0A9],00000000
      00463374 - push 00
      00463376 - push ebp
      00463377 - call 0046337F
        0046337F - pop ebp
        00463380 - mov [esp+04],ebp
        00463384 - add [esp+04],00000018
        0046338C - inc ebp
        0046338D - push ebp
        0046338E - ret
      0046337D - pop ebp
      0046337E - ret
    00463394 - mov si,8A21
    00463398 - lea ebx,[ebp+11FE0921]
    0046339E - push 00
    004633A0 - push esi
    004633A1 - call 004633A9
      004633A9 - pop esi
      004633AA - mov [esp+04],esi
      004633AE - add [esp+04],00000014
      004633B6 - inc esi
      004633B7 - push esi
      004633B8 - ret
    004633A7 - pop esi
    004633A8 - ret
  004633BA - or esi,16DC4DEE
  004633C0 - jmp 004633D0
  004633D0 - and [ebx],FFFFFEFF
  004633D6 - mov [ebp+11FE1CF8],esi
  004633DC - call 004633F2
    004633F2 - mov [ebp+11FE228B],esi
    004633F8 - pop esi
    004633F9 - mov bx,ds
    004633FC - push esi
    004633FD - call 00463416
      00463416 - mov [ebp+11FE1F09],ecx
      0046341C - pop edi
      0046341D - pop esi
      0046341E - test bl,04
      00463421 - je 00463474
      00463474 - jmp 00463865
      00463865 - mov eax,ebx
      00463867 - mov ebx,eax
```

```
      00463869 - push 00
      0046386B - push ecx
      0046386C - call 00463874
        00463874 - pop ecx
        00463875 - mov [esp+04],ecx
        00463879 - add [esp+04],0000001C
        00463881 - inc ecx
        00463882 - push ecx
        00463883 - ret
      00463872 - pop ecx
      00463873 - ret
    0046388D - jno 00463898
    00463898 - mov [ebp+11FE0245],55B8ED13
    004638A2 - add edi,[ebp+11FE2D37]
    004638A8 - mov eax,eax
    004638AA - sub di,671C
    004638AF - cmp [ebp+11FE2CDF],000007D0 ; CONSTANT Delay Time! SAME VALUE F
    004638B9 - jne 004638FB
    004638FB - mov eax,[ebp+11FE2CDF] ; Store Delay Time into Register
    00463901 - jmp 0046391A
    0046391A - push eax
    0046391B - jmp 00463933
    00463933 - mov [esp],eax ; Load Register into Parameter
    00463936 - push 00
    00463938 - push ecx
    00463939 - call 00463941
      00463941 - pop ecx
      00463942 - mov [esp+04],ecx
      00463946 - add [esp+04],00000014
      0046394E - inc ecx
      0046394F - push ecx
      00463950 - ret
    0046393F - pop ecx
    00463940 - ret
  00463952 - mov esi,edi
  00463954 - call dword ptr [ebp+11FE20CB] ; KERNEL32.Sleep
```

From the API Monitor logs, we see the following strings being “passed” into USER32.FindWindowA but we are unable to find the strings when searching in the process’s memory range.

```
"OLLYDBG"
"pediy06"
"GBDYLLO"
```

```
push lpWindowName
push lpClassName
call dword ptr [ebp+11FE27A2] ; USER32.FindWindowA
; returning hWnd -> eax
```

Why?

Blacklists Explained

Knowing that the values must somehow be “added” and “removed” for the USER32.FindWindowA parameters we look into the transfer and algebraic mnemonics being used after the USER32.FindWindowA call. One instruction after 00463BE3 sticks out like a sore thumb.

```
00463BFB - mov [ecx],00000000
```

Following ```[ecx]``` leads to some interesting operations regarding ```[ecx]``` and ```[ecx+04]```.

A spread of movement (transfer) and add (algebraic) can be seen within.

```
00463EA6 - mov [ecx],54CA95A5
00463EAC - push 00
00463EAE - push edx
00463EAF - call 00463EB7
00463ECE - jmp 00463EDE
00463EDE - add [ecx],1499CFCB
...
00463F00 - mov [ecx+04],CE5F5969
00463F07 - or di,00A2
00463F0C - add [ecx+04],31D6D710
```

A close coupling of movements (transfers) zeroing can also be seen.

```
00463FE0 - mov [ebp+11FE1CE3],ebx
00463FE6 - mov [ecx],00000000
00463FEC - mov [ebp+12019E53],edx
00463FF2 - mov [ecx+04],00000000
00463FF9 - push 00
00463FFB - push ecx
```

The fact that ```[ecx]``` and ```[ecx+04]``` are always written to provides more speculation over the usages. When computing ```[ecx]``` and ```[ecx+04]``` the truth of the operations starts to be revealed.

```
[ecx] = 54CA95A5 + 1499CFCB = 69646570 = "idep"
[ecx+04] = CE5F5969 + 31D6D710 = 363079 = "60y"
...
call USER32.FindWindowA
...
[ecx] = 00000000
[ecx+04] = 00000000
```

The result is dereferencing the contents of ecx “60yidep” which is “pediy06” backwards!

In short, blacklisted strings are stored in split parts as immutables due to size constraints and are then transferred into the pointers of registers! As we look into other features we will start noticing a similar pattern start to shape with Oreans.

From the trace above we are then able to obtain the following:

```
00463EA6 - mov [ecx],54CA95A5
00463D33 - add [ecx],1499CFCB
00463EDE - add [ecx],1499CFCB
00463F00 - mov [ecx+04],CE5F5969
00463F0C - add [ecx+04],31D6D710
00463D5F - mov [ecx+04],CE78753C
00463D8A - add [ecx+04],31D6D710
```

To make matters simple I've provided signatures that were tested from 2.3.0.0 to 2.4.6.0 that can identify anti-debugger blacklists.

```
{"anti_debug_a":"C7 01 A5 95 CA 54",
"anti_debug_b":"81 01 CB CF 99 14",
"anti_debug_c":"C7 41 04 3C 75 78 CE",
"anti_debug_d":"C7 41 04 69 59 5F CE",
"anti_debug_e":"81 41 04 10 D7 D6 31",
"anti_debug_f":"C7 41 04 34 6B 70 CE",
"anti_debug_g":"81 02 51 FD 67 0C"}
```

One could now just simply search the signatures and zero out the immutables thus allowing a blacklisted application like OllyDBG to be able to run along with the Themida'd application.

```
81 01 CB CF 99 14 -> 81 01 00 00 00 00
C7 41 04 69 59 5F CE -> C7 41 04 00 00 00 00
```

But there is an easier solution than this as well as more to cover!

Debugger Detections

Now for following from the trace onwards starting at 00463954. We will encounter the following instruction with ```dword ptr [ebp+11FE255B]```.

```
cmp dword ptr [ebp+11FE255B],00
```

If we were to use static analysis we would not easily be able to determine what ```[ebp+11FE255B]``` points to without understanding how Themida creates itself and the magic that is stored within ```[ebp]```. Luckily stepping into the trace we can safely assume that ```[ebp+11FE255B]``` is KERNEL32.IsDebuggerPresent.

```
 0046395A - nop
 0046395B - inc eax
 0046395C - push 00
 0046395E - push edx
 0046395F - call 00463967
  00463967 - pop edx
  00463968 - mov [esp+04],edx
  0046396C - add [esp+04],00000014
  00463974 - inc edx
  00463975 - push edx
  00463976 - ret
 00463965 - pop edx
 00463966 - ret
00463978 - adc si,4B4E
0046397D - cmp dword ptr [ebp+11FE255B],00 ; Validity Check
00463984 - je 00463A47
0046398A - add si,DE92
0046398F - cmp dword ptr [ebp+11FE2B47],00 ; Is Difference Check
00463996 - je 00463A47
0046399C - push 00
0046399E - push edx
0046399F - call 004639A7
 004639A7 - pop edx
 004639A8 - mov [esp+04],edx
 004639AC - add [esp+04],00000014
 004639B4 - inc edx
 004639B5 - push edx
 004639B6 - ret
004639A5 - pop edx
004639A6 - ret
004639B8 - jmp 004639C7
004639C7 - call dword ptr [ebp+11FE255B] ; KERNEL32.IsDebuggerPresen
004639CD - add [ebp+11FE2F7D],edx
004639D3 - or eax,eax
004639D5 - je 00463A47
00463A47 - cmp dword ptr [ebp+11FE27A2],00 ; Validity Check
00463A4E - je 0046407A
00463A54 - push 00
00463A56 - push ebx
00463A57 - call 00463A5F
 00463A5F - pop ebx
 00463A60 - mov [esp+04],ebx
 00463A64 - add [esp+04],0000001C
 00463A6C - inc ebx
 00463A6D - push ebx
 00463A6E - ret
00463A5D - pop ebx
00463A5E - ret
```

```
  00463A78 - sub [ebp+11FE2347],edx
  00463A7E - add edi,57633FD2
  00463A84 - cmp dword ptr [ebp+11FE2B47],00 ; Is Difference Check
  00463A8B - je 0046407A
  00463A91 - push 00
  00463A93 - push ebp
  00463A94 - call 00463A9C
   00463A9C - pop ebp
   00463A9D - mov [esp+04],ebp
   00463AA1 - add [esp+04],0000001A
   00463AA9 - inc ebp
   00463AAA - push ebp
   00463AAB - ret
  00463A9A - pop ebp
  00463A9B - ret
 00463AB3 - lea ecx,[ebp+11FE08E9]
 00463AB9 - push 00
 00463ABB - push edi
 00463ABC - call 00463AC4
  00463AC4 - pop edi
  00463AC5 - mov [esp+04],edi
  00463AC9 - add [esp+04],00000014
  00463AD1 - inc edi
  00463AD2 - push edi
  00463AD3 - ret
 00463AC2 - pop edi
 00463AC3 - ret
00463AD5 - mov di,dx
00463AD8 - mov [ecx],44B27C84
00463ADE - mov si,cx
00463AE1 - add [ecx],1499CFCB
00463AE7 - mov edi,44AA2780
00463AEC - mov esi,[ebp+11FE29A9]
00463AF2 - mov [ecx+04],CE706B34
00463AF9 - push 00
00463AFB - push ecx
00463AFC - call 00463B04
 00463B04 - pop ecx
 00463B05 - mov [esp+04],ecx
 00463B09 - add [esp+04],00000016
 00463B11 - inc ecx
 00463B12 - push ecx
 00463B13 - ret
00463B02 - pop ecx
00463B03 - ret
00463B17 - mov esi,edi
00463B19 - and si,A640
00463B1E - add [ecx+04],31D6D710
00463B25 - push 00
00463B27 - push ecx
00463B28 - call 00463B30
 00463B30 - pop ecx
```

```
    00463B31 - mov [esp+04],ecx
    00463B35 - add [esp+04],0000001D
    00463B3D - inc ecx
    00463B3E - push ecx
    00463B3F - ret
   00463B2E - pop ecx
   00463B2F - ret
  00463B4A - mov edi,[ebp+11FE2833]
  00463B50 - push 00
  00463B52 - push 00
  00463B54 - push esi
  00463B55 - call 00463B5D
   00463B5D - pop esi
   00463B5E - mov [esp+04],esi
   00463B62 - add [esp+04],00000015
   00463B6A - inc esi
   00463B6B - push esi
   00463B6C - ret
  00463B5B - pop esi
  00463B5C - ret
 00463B6F - lea eax,[ebp+11FE08E9]
 00463B75 - push 00
 00463B77 - push edi
 00463B78 - call 00463B80
  00463B80 - pop edi
  00463B81 - mov [esp+04],edi
  00463B85 - add [esp+04],00000018
  00463B8D - inc edi
  00463B8E - push edi
  00463B8F - ret
 00463B7E - pop edi
 00463B7F - ret
00463B95 - jp 00463B9B
00463B9B - movzx edi,bx
00463B9E - push eax
00463B9F - push eax
00463BA0 - push edx
00463BA1 - pushad
00463BA2 - push eax
00463BA3 - pop eax
00463BA4 - ja 00463BAA
00463BAA - popad
00463BAB - rdtsc
00463BAD - pushad
00463BAE - push ebx
00463BAF - pop ebx
00463BB0 - js 00463BB6
00463BB6 - popad
00463BB7 - pop edx
00463BB8 - pop eax
00463BB9 - mov [esp],eax
00463BBC - push 00
```

```
 00463BBE - push ebp
 00463BBF - call 00463BC7
  00463BC7 - pop ebp
  00463BC8 - mov [esp+04],ebp
  00463BCC - add [esp+04],00000015
  00463BD4 - inc ebp
  00463BD5 - push ebp
  00463BD6 - ret
 00463BC5 - pop ebp
 00463BC6 - ret
00463BD9 - jmp 00463BE3
00463BE3 - call dword ptr [ebp+11FE27A2] ; USER32.FindWindowA
```

To bypass this check there are a couple ways one could approach it. KERNEL32.IsDebuggerPresent returns a boolean to `eax`. Force writing the `eax` to `0` would in return always return false. This can be done by hooking `KERNEL32.IsDebuggerPresent` and setting the `eax` to `0`.

There are four compares being used throughout this region split nicely into two groups.

```
0046397D - cmp dword ptr [ebp+11FE255B], 00 ; Validity Check
; [ebp+11FE255B] = [EE425014+11FE255B] = [0040756F] == 75F72B80 == KERNEL32.
if KERNEL32.IsDebuggerPresent > 00000000:
    ZF = 0, CF = 0
else if KERNEL32.IsDebuggerPresent == 00000000:
    ZF = 1, CF = 0
else:
    ZF = 0, CF = 1

0046398F - cmp dword ptr [ebp+11FE2B47],00 ; Is Difference Check
; [ebp+11FE2B47] = [EE425014+11FE2B47] = [00407B5B] == 00000001
if [00407B5B] > 00000000:
    ZF = 0, CF = 0
else if [00407B5B] == 00000000:
    ZF = 1, CF = 0
else:
    ZF = 0, CF = 1
```

```
00463A47 - cmp dword ptr [ebp+11FE27A2],00 ; Validity Check
; [ebp+11FE27A2] = [EE425014+11FE27A2] = [004077B6] == 75624F80 == USER32.Fi
if USER32.FindWindowA > 00000000:
```

```
    ZF = 0, CF = 0
else if USER32.FindWindowA == 00000000:
    ZF = 1, CF = 0
else:
    ZF = 0, CF = 1

00463A84 - cmp dword ptr [ebp+11FE2B47],00 ; Is Difference Check
; [ebp+11FE2B47] = [EE425014+11FE2B47] = [00407B5B] == 00000001
if [00407B5B] > 00000000:
    ZF = 0, CF = 0
else if [00407B5B] == 00000000:
    ZF = 1, CF = 0
else:
    ZF = 0, CF = 1
```

Normally, once a difference is detected Themida will attempt to terminate the process. Bypassing this area, for now, will just require force checking and setting ZF and CF to 0 as we are handling KERNEL32.IsDebuggerPresent within the API and USER32.FindWindowA with a blacklist parameter smearing approach.

Restoring API

Upon execution, Themida tends to patch the following addresses which can create issues for debuggers. More details about how these patches are created will be given in the PREFIX approach write-up. You can find these patches by using any hook detection tool. I still use HookShark in most situations.

```
NTDLL.DbgBreakPoint:
ret
```

```
NTDLL.DbgUiRemoteBreakin:
jmp ntdll.LdrShutdownProcess
```

The API can simply be restored by re-patching the original instructions without any issues.

```
NTDLL.DbgBreakPoint:
int3
```

```
NTDLL.DbgUiRemoteBreakin:
push 08h
```

Conclusion

The best way to handle Themida's anti-debugging is to:

- Restore `NTDLL.DbgBreakPoint` and `NTDLL.DbgUiRemoteBreakin`.
- Always falsify `KERNEL32.IsDebuggerPresent` returns.
- Hide PEB.BeingDebugged by hijacking an existing thread and reiterating PEB.BeingDebugged as zero.

No Themida thread should ever be considered your friend unless you are hijacking it for your own advantage.

Killing off the blacklists anti-debug thread should be a must and can be done by simply searching from the signatures provided, hooking, terminating the current thread, and restoring the original hooked memory to avoid any future possible integrity check issues.

The other non-identified Themida threads that have the start address of `antidebug.exe+0x6707` in this sample should ultimately also be killed off. We will get into ultimately why in another write-up. Keep watch!

The best way to kill off Themida threads is by setting a breakpoint on `KERNEL32.Sleep` then determining where the call is coming from reading the topmost ESP and basing the location on its segment range. After determining if the caller is within the Themida segment find the current thread's start address. If the start address's format or "signature" closely resembles that of the sample at `antidebug.exe+0x6707` and is located within the Themida segment then terminate the current thread and add to your incrementing. Rinse and repeat

until your increment reaches eight or until 10–20 seconds have been reached.

Alternatively, hooking NtQueryInformationThread and NtSetInformationThread can produce the same results as well as check for the ThreadHideFromDebugger within the thread context. It is ultimately up to the reader in regard to their own style, technique, and preferences.

By killing off Themida generated threads one can greatly improve a process's speed and remove annoyances when attempting to debug. It essentially chops the heads off of the many-headed beast in one go without much backlash as it is a universal approach and works against various anti-protection features developed by Oreans.

Note: More information will be provided for the PREFIX approach. Keep watch!

written by dovezp@github

Software Security | Code Debugging | Reverse Engineering | Code Obfuscation

Follow

Written by Reversing

5 followers · 0 following

https://github.com/dovezp

No responses yet

N Niguelas

What are your thoughts?

More from Reversing

Reversing

Analysis of Oreans Themida (x86), Version 3.0.8.0, Advanced API-Wrapping

Written 23/MAY/2020

Oct 8, 2023 1

Reversing

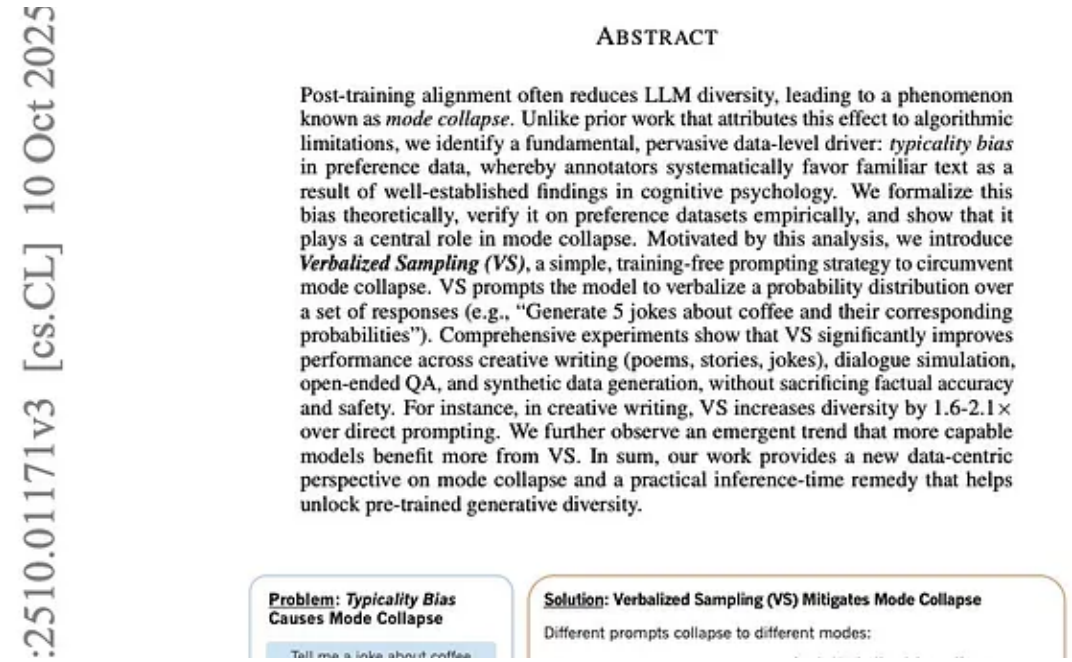
Obfuscation - A Simple Introduction to EIP

Written 05/NOV/2017

Oct 8, 2023

See all from Reversing

Recommended from Medium



ABSTRACT

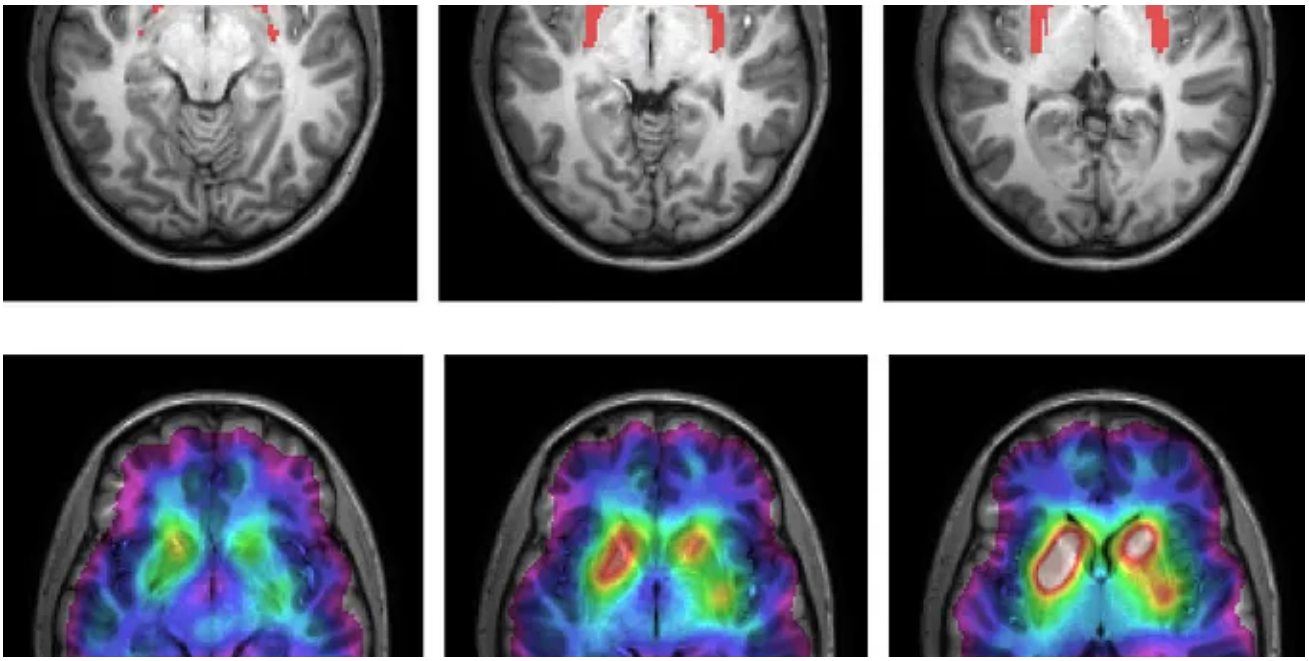
Post-training alignment often reduces LLM diversity, leading to a phenomenon known as *mode collapse*. Unlike prior work that attributes this effect to algorithmic limitations, we identify a fundamental, pervasive data-level driver: *typicality bias* in preference data, whereby annotators systematically favor familiar text as a result of well-established findings in cognitive psychology. We formalize this bias theoretically, verify it on preference datasets empirically, and show that it plays a central role in mode collapse. Motivated by this analysis, we introduce ***Verbalized Sampling (VS)***, a simple, training-free prompting strategy to circumvent mode collapse. VS prompts the model to verbalize a probability distribution over a set of responses (e.g., "Generate 5 jokes about coffee and their corresponding probabilities"). Comprehensive experiments show that VS significantly improves performance across creative writing (poems, stories, jokes), dialogue simulation, open-ended QA, and synthetic data generation, without sacrificing factual accuracy and safety. For instance, in creative writing, VS increases diversity by 1.6-2.1× over direct prompting. We further observe an emergent trend that more capable models benefit more from VS. In sum, our work provides a new data-centric perspective on mode collapse and a practical inference-time remedy that helps unlock pre-trained generative diversity.

In Generative AI by Adham Khaled

Stanford Just Killed Prompt Engineering With 8 Words (And I Can't Believe It Worked)

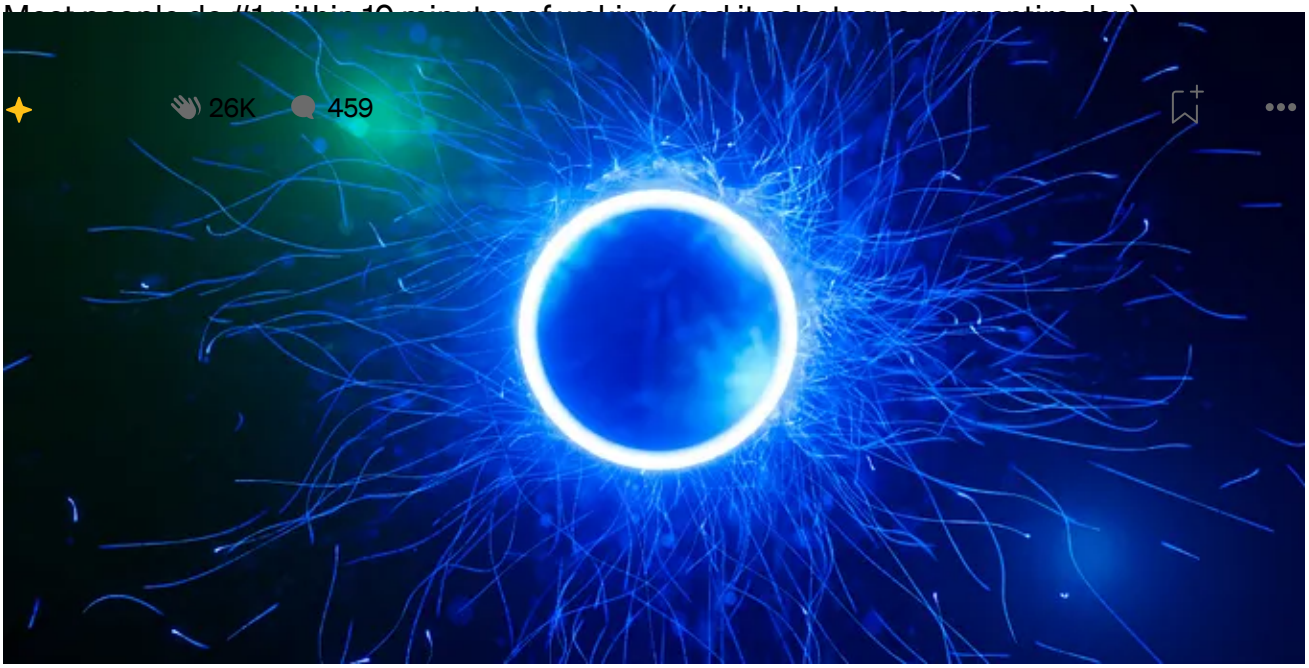
ChatGPT keeps giving you the same boring response? This new technique unlocks 2× more creativity from ANY AI model — no training required...

Oct 20, 2025 23K 606



In Write A Catalyst by Dr. Patricia Schmidt

As a Neuroscientist, I Quit These 5 Morning Habits That Destroy Your Brain

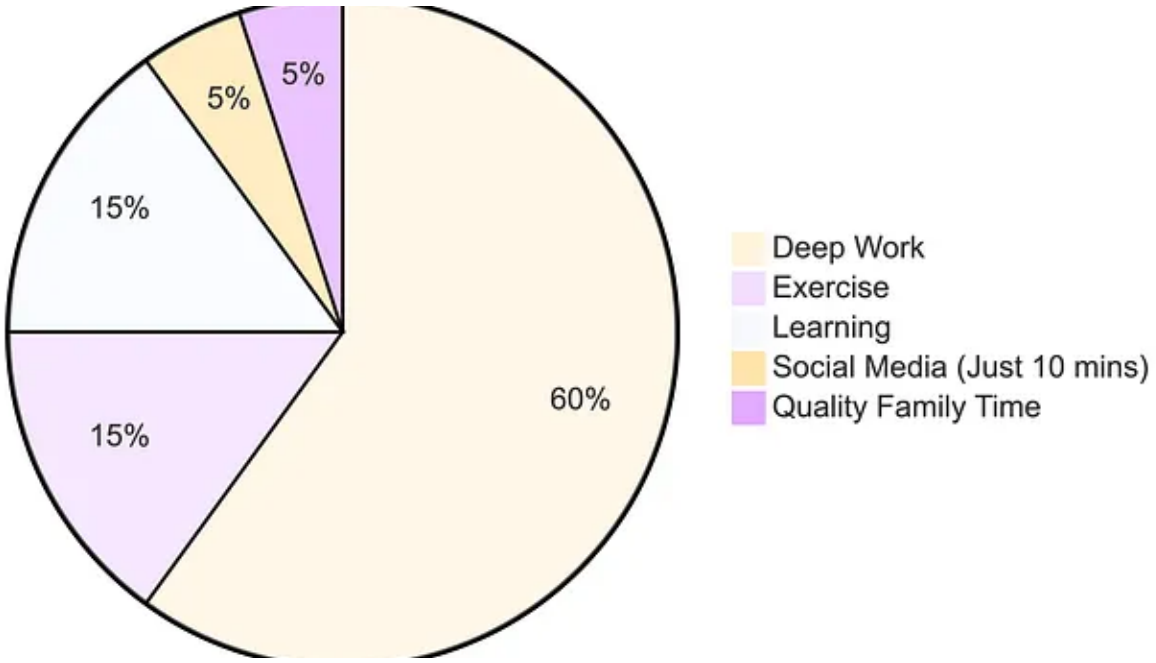


Will Lockett

The AI Bubble Is About To Burst, But The Next Bubble Is Already Growing

Techbros are preparing their latest bandwagon.

Sep 14, 2025 22K 937



In Level Up Coding by Teja Kusireddy

I Stopped Using ChatGPT for 30 Days. What Happened to My Brain Was Terrifying.

91% of you will abandon 2026 resolutions by January 10th. Here's how to be in the 9% who actually win.

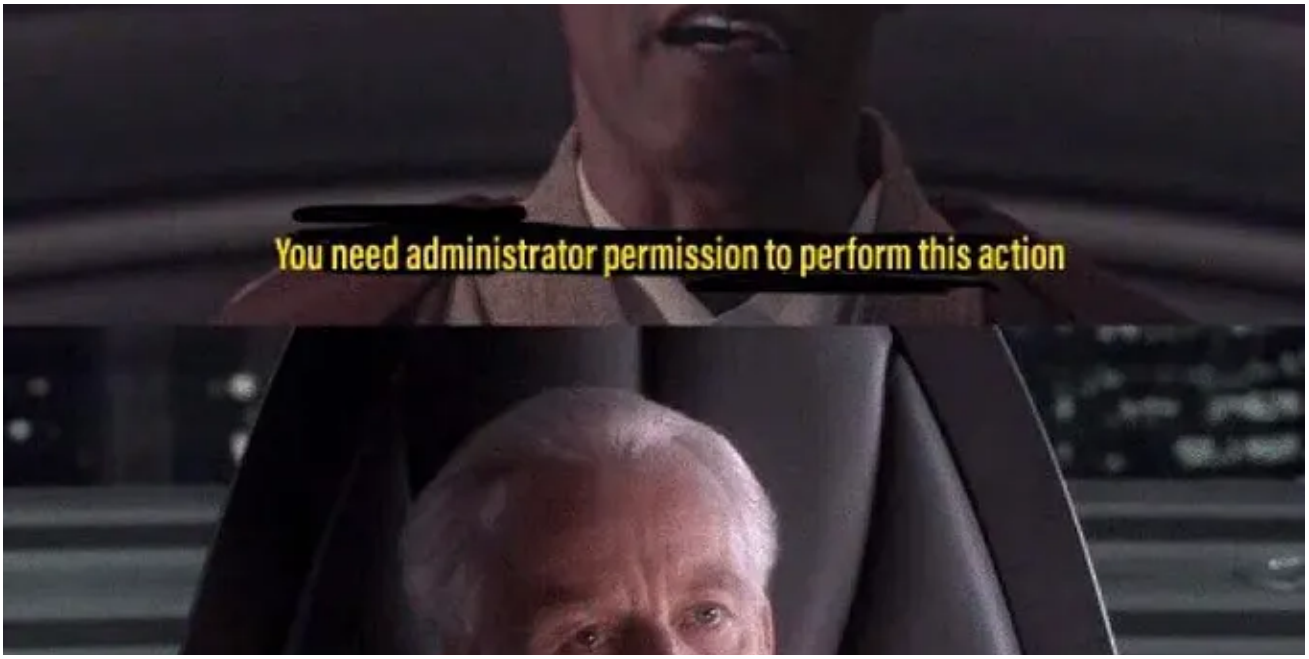
Dec 28, 2025 3.8K 163

Hartarto

15 Free OSINT Tools That Reveal Everything Online (2026 Guide)

Everything about you online leaves a trail. Emails, websites, servers, and devices continuously expose information — not because you were...

Jan 7 800 9



RootRouteway

Breaking Boundaries: Mastering Windows Privilege Escalation with Boxes

In today's security landscape, gaining and maintaining system access is only part of the story — understanding how privileges are...

Nov 9, 2025 9 2