

From MS08-067 to EternalBlue

BsidesMCR, 2017

#whoami

- Denis Isakov a.k.a vinegrep
- Exploit development and RE enthusiast
- Doing pentest as a daily job
- OSCP/OSWP/OSCE/GXPN/GREM/BBQ/WTF
and other acronyms

What this talk is about?

- Follow growing exploit development difficulty based on 3 “zero-to-hero” exploits.
- Analyze how these exploits influenced exploit mitigation techniques.
- Should we expect something similar in recent time?

What this talk is not about?

- Not going to blame Microsoft.
- Not going to discuss conspiracy theories why Microsoft patch MS17-010 so quickly.
- Step-by-step exploit development.
- SMB protocol internals

MS08-067



imgflip.com

MS08-067 general info

- “Wormable” vulnerability in SMB that was triggered by malformed RPC request.
- Buffer overflow was in the way how Server service processes the PathName argument to the NetprPathCanonicalize function.
- Problem was in directory traversal sequences result in traversing past the root path.
- Kudos:

<https://blogs.technet.microsoft.com/johnla/2015/09/26/the-inside-story-behind-ms08-067/>

<https://labs.mwrinfosecurity.com/assets/BlogFiles/hello-ms08-067-my-old-friend.pdf>

<https://www.mysonicwall.com/sonicalert/searchresults.aspx?ev=article&id=74>

<https://dontstuffbeansupyournose.com/2008/10/23/looking-at-ms08-067/>

MS08-067 exploitation

```
Registers (FPU)
EAX 61736964
ECX 0108F4B2
EDX 0108F508 ASCII "GGGGHHHH"
EBX 0108005C
ESP 0108F47C ASCII "disadisadisadis"
EBP 44444444
ESI 0108F4B6
EDI 0108F464
EIP 41414141

C 0 ES 0023 32bit 0(FFFFFFFF)
P 1 CS 001B 32bit 0(FFFFFFFF)
A 0 SS 0023 32bit 0(FFFFFFFF)
Z 1 DS 0023 32bit 0(FFFFFFFF)
S 0 FS 003B 32bit 7FFAB000(FFF)
T 0 GS 0000 NULL
D 0
0 0 LastErr ERROR_SUCCESS (00000000)

0108F45C 00000000 ....
0108F460 0108F4B4 'ô■■
0108F464 4444005C \.DD
0108F468 44444444 DDDD
0108F46C 44444444 DDDD
0108F470 44444444 DDDD
0108F474 44444444 DDDD
0108F478 41414141 AAAA
0108F47C 61736964 disa
0108F480 61736964 disa
0108F484 61736964 disa
0108F488 61736964 disa
0108F48C 61736964 disa
0108F490 61736964 disa
0108F494 61736964 disa
0108F498 61736964 disa
0108F49C 61736964 disa
0108F4A0 61736964 disa
0108F4A4 61736964 disa
0108F4A8 47474747 GGGG
0108F4AC 48484848 HHHH
0108F4B0 000C0000 ....
0108F4B4 002E005C \...
0108F4B8 005C002E ..\.
```

MS08-067 exploitation (2)

- Crash service by overwriting EIP at 18 byte offset.
- ESP and EDX points to our buffer, EAX and EBP was overwritten with our buffer content.
- Overwrite EIP with address of instruction “JMP EDX”, as there is enough space between EIP and address where EDX points.
- Store NOP sled + egghunter after our return address.
- Jump to the end of our controller buffer.
- There is a short jump back instruction that will pass execution to egghunter.
- Egghunter will find final shellcode and PROFIT.

MS08-067 NX bypass

- From XP SP2 and 2003 SP1 we have NX bit enabled. NX bit separates between code and data.
- Excellent paper from skape and Skywing how to bypass it - <http://uninformed.org/?v=2&a=4&t=txt>
- Idea: disable DEP for process during execution.
- Search memory for opcodes and check whether they are in executable area.
- LdprCheckNXCompatibility in ntdll.dll is used to check whether NX should be enabled and then based on this check call to ZwSetInformationProcess is used to enable/disable DEP.
- Return to controlled buffer, jump back, egg hunt and final shellcode.

MS08-067 NX bypass before call

Immunity Debugger - svchost.exe - [CPU - thread 0000041C, module ntdll]

File View Debug Plugins ImmLib Options Window Help Jobs

Immunity Consulting Services Manager

l e m t w h c P k b z r ... s ?

Registers (FPU)

EAX 0108F460
ECX 0108F4B2
EDX 0108F508 ASCII "AAAAAAA"
EBX 0108005C
ESP 0108F474
EBP 0108F464
ESI 0108F4B6
EDI 0108F464
EIP 7C83E424 ntdll.7C83E424
C 0 ES 0023 32bit 0(FFFFFFFF)
P 1 CS 001B 32bit 0(FFFFFFFF)
A 0 SS 0023 32bit 0(FFFFFFFF)
Z 1 DS 0023 32bit 0(FFFFFFFF)
S 0 FS 003B 32bit 7FFAB000(FFF)
T 0 GS 0000 NULL
D 0
0 0 LastErr ERROR_SUCCESS (00000000)
EFL 00000246 (NO,NB,E,BE,NS,PE,GE,LE)
ST0 empty
ST1 empty
ST2 empty
ST3 empty
ST4 empty
ST5 empty
ST6 empty
ST7 empty
3 2 1 0 ESPUOZDI
FST 0000 Cond 0 0 0 0 Err 0 0 0 0 0 0 0 0 (GT)
FCW 027F Prec NEAR,53 Mask 1 1 1 1 1 1

7C83E413 C745 FC 02000001 MOV DWORD PTR SS:[EBP-4],2
7C83E41A 6A 04 PUSH 4
7C83E41C 8D45 FC LEA EAX,DWORD PTR SS:[EBP-4]
7C83E41F 50 PUSH EAX
7C83E420 6A 22 PUSH 22
7C83E422 6A FF PUSH -1
7C83E424 E8 2F3AFFFF CALL ntdll.ZwSetInformationProcess
7C83E429 ^E9 4775FFFF JMP ntdll.7C835975
7C83E42E 66:3935 CE4887 CMP WORD PTR DS:[7C88A4CE],SI
7C83E435 0F84 539A0100 JE ntdll.7C857E8E
7C83E43B 66:A1 CE4887C MOV AX,WORD PTR DS:[7C88A4CE]
7C83E441 ^E9 4983FFFF JMP ntdll.7C83678F
7C83E446 8BC7 MOV EAX,EDI
7C83E448 66:25 00FC AND AX,0FC00
7C83E44C 66:3D 0008 CMP AX,800
7C83E450 0F84 529A0100 JE ntdll.7C857E88
7C83E456 64:3935 18000001 CMP DWORD PTR FS:[18],ESI
7C83E45D ^0F85 1082FFFF JNZ ntdll.7C836673
7C83E463 ^E9 D65FFFFF JMP ntdll.7C83443E
7C83E468 8D45 CC LEA EAX,DWORD PTR SS:[EBP-34]
7C83E46B 50 PUSH EAX
7C83E46C 6A 01 PUSH 1
7C83E46E E8 6534FEFF CALL ntdll.ZwQueryDefaultLocale
7C83E473 8945 DC MOV DWORD PTR SS:[EBP-24],EAX
7C83E476 3BC6 CMP EAX,ESI
7C83E478 0F8C 369A0100 JL ntdll.7C857EB4
7C83E47E 8B45 CC MOV EAX,DWORD PTR SS:[EBP-34]
7C83E481 ^E9 A881FFFF JMP ntdll.7C83662E
7C83E486 81E1 FF03FFFF AND ECX,FFFF03FF
7C83E48C ^E9 BA81FFFF JMP ntdll.7C83664B
7C83E491 8D45 9C LEA EAX,DWORD PTR SS:[EBP-64]
7C83E494 50 PUSH EAX
7C83E495 56 PUSH ESI
7C83E496 E8 3D34FEFF CALL ntdll.ZwQueryDefaultLocale
7C83E49B 8945 DC MOV DWORD PTR SS:[EBP-24],EAX
7C83E49E 3BC6 CMP EAX,ESI
7C83E4A0 0F8C 0E9A0100 JL ntdll.7C857EB4

Address Hex dump ASCII
01004000 34 31 00 01 00 00 00 00 41.1....
01004008 BD DF 68 77 BD D5 6A 77 %bhwz0jw
01004010 FA 14 68 77 00 00 00 00 00kw....
01004018 00 00 00 00 00 00 00 00
01004020 00 00 00 00 00 00 00 00
01004028 00 00 00 00 00 00 00 00
01004030 00 00 00 00 00 00 00 00
01004038 00 00 00 00 00 00 00 00
01004040 E0 94 88 7C FF FF FF FF 00000000
01004048 00 00 00 00 00 00 00 00
01004050 94 02 00 00 00 00 00 00
01004058 29 FE 00 00 A0 3F 08 00 }p.. ?
01004060 28 00 00 00 00 00 00 00 (....
01004068 50 B8 08 00 E8 A3 0E 00 P...èè
01004070 F8 40 09 00 D6 01 FF FF 00000000
01004078 F8 39 08 00 00 00 67 77 09000000
01004080 00 00 00 00 38 49 09 008I..
01004088 FF FF FF FF 00 00 00 00 00000000
01004090 00 00 00 00 00 00 00 00
01004098 00 00 00 00 01 00 00 00
010040A0 00 00 00 00 00 00 00 00
0108F468 44444444 DDDD
0108F46C 44444444 DDDD
0108F470 44444444 DDDD
0108F474 FFFFFFFF 00000000 Arg1 = FFFFFFFF
0108F478 00000022 "..." Arg2 = 00000022
0108F47C 0108F460 "0" Arg3 = 0108F460
0108F480 00000004 "..." Arg4 = 00000004
0108F484 41414141 AAAA
0108F488 41414141 AAAA
0108F48C 41414141 AAAA
0108F490 41414141 AAAA
0108F494 41414141 AAAA
0108F498 41414141 AAAA
0108F49C 41414141 AAAA
0108F4A0 41414141 AAAA
0108F4A4 41414141 AAAA
0108F4A8 41414141 AAAA
0108F4AC 41414141 AAAA
0108F4B0 000C0000
0108F4B4 002E005C \...
0108F4B8 005C002E ..\
0108F4BC 44444444 DDDD

[17:26:14] Breakpoint at ntdll.7C83E424

Start Immunity Debugger ... 5:26 PM

MS08-067 NX bypass after call

7C83E413	C745 FC 02000000	MOV DWORD PTR SS:[EBP-4],2
7C83E41A	6A 04	PUSH 4
7C83E41C	8D45 FC	LEA EAX,DWORD PTR SS:[EBP-4]
7C83E41F	50	PUSH EAX
7C83E420	6A 22	PUSH 22
7C83E422	6A FF	PUSH -1
7C83E424	E8 2F3AFEFF	CALL ntdll.ZwSetInformationProcess
7C83E42E	^E9 4775FFFF	JMP ntdll.7C835975
7C83E435	66:3935 CEA4887C	CMP WORD PTR DS:[7C88A4CE],SI
7C83E43B	0F84 539A0100	JE ntdll.7C857E8E
7C83E441	66:A1 CEA4887C	MOV AX,WORD PTR DS:[7C88A4CE]
7C83E446	^E9 4983FFFF	JMP ntdll.7C83678F
7C83E448	8BC7	MOV EAX,EDI
7C83E44C	66:25 00FC	AND AX,0FC00
7C83E450	0F84 529A0100	JE ntdll.7C857EA8
7C83E456	64:3935 18000000	CMP DWORD PTR FS:[18],ESI
7C83E45D	^0F85 1082FFFF	JNZ ntdll.7C836673
7C83E463	^E9 D65FFFFF	JMP ntdll.7C83443E

7C835975=ntdll.7C835975

Registers (FPU)
EAX 00000000
ECX 0108F46C
EDX 7C82ED54 ntdll.KiFastSystemCallRet
EBX 0108005C
ESP 0108F484 ASCII "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"
EBP 0108F464
ESI 0108F4B6
EDI 0108F464
EIP 7C83E429 ntdll.7C83E429
C 0 ES 0023 32bit 0(FFFFFFFF)
P 1 CS 001B 32bit 0(FFFFFFFF)
A 0 SS 0023 32bit 0(FFFFFFFF)
Z 1 DS 0023 32bit 0(FFFFFFFF)
S 0 FS 003B 32bit 7FFAB000(FFF)
T 0 GS 0000 NULL
D 0
O 0
0 0 LastErr ERROR_SUCCESS (00000000)
EFL 00000246 (NO,NB,E,BE,NS,PE,GE,LE)

MS08-067 conclusions

- Wrong path validation leads to overflow.
- Easy to exploit vulnerability without any protections.
- Even on NX-enabled versions also exploitable.

MS09-050



imgflip.com

MS09-050 general info

- Found by Laurent Gaffié in 3 seconds of fuzzing SMBv2. Reliable exploit by Piotr Bania, first MSF version by Stephen Fewer.
- EAX register is initialized with a word from the SMB2 packet. In the next instruction our controlled value is used as an array index with one safety check whether the value is NULL. Accidentally there is no check if array index exceeds the number of elements in the array. Later, the location pointed by our controlled value is executed using the call instruction.

- Kudos:

<http://g-laurent.blogspot.be/2009/09/windows-vista7-smb20-negotiate-protocol.html>

<http://g-laurent.blogspot.be/2009/10/more-explication-on-cve-2009-3103.html>

http://piotrbania.com/all/articles/smb2_351_packets.pdf

MS09-050 exploitation

```
kd> g
Breakpoint 3 hit
srv2!SrvSnapshotScavengerTimer:
8e32eea0 6a01          push     1
kd> g
Breakpoint 2 hit
srv2!SrvProcCompleteRequest+0xd2:
8e33bb91 ffd0          call     eax
kd> r eax
eax=ffd00d09
kd> g
Breakpoint 0 hit
ffd00d09 5e          pop     esi
kd> u
ffd00d09 5e          pop     esi
ffd00d0a c3          ret
ffd00d0b 7c58        jl       ffd00d65
ffd00d0d 0fb75102    movzx   edx,word ptr [ecx+2]
ffd00d11 0fb77002    movzx   esi,word ptr [eax+2]
ffd00d15 663bd6      cmp     dx,si
ffd00d18 7fed        jg       ffd00d07
ffd00d1a 7c49        jl       ffd00d65
kd> p
ffd00d0a c3          ret
kd> p
84910bf8 00544d42    add     byte ptr [ebp+ecx*2+42h],dl
kd> g
```

MS09-050 conclusions

- Quick to find, fuzzer code from Laurent is straight-forward.
- Requires efforts to make exploitation reliable: Piotr's trampoline vs hard-coded addresses.
- ASLR were bypassed as address from kernel HAL memory region was used.
- NX HAL heap and NonPagedPoolNx from Windows 8



MS17-010

imgflip.com

MS17-010 general info

- Released by Shadow Brokers, developed by Equation Group.
- Complicated exploit requires several steps to trigger vulnerability.

Kudos:

http://risksense.com/_api/filesystem/466/EternalBlue_RiskSense-Exploit-Analysis-and-Port-to-Microsoft-Windows-10_v1_2.pdf

<http://blog.trendmicro.com/trendlabs-security-intelligence/ms17-010-eternalblue/>

<http://blogs.360.cn/360safe/2017/04/17/nsa-eternalblue-smb/>

<https://github.com/worawit/MS17-010>

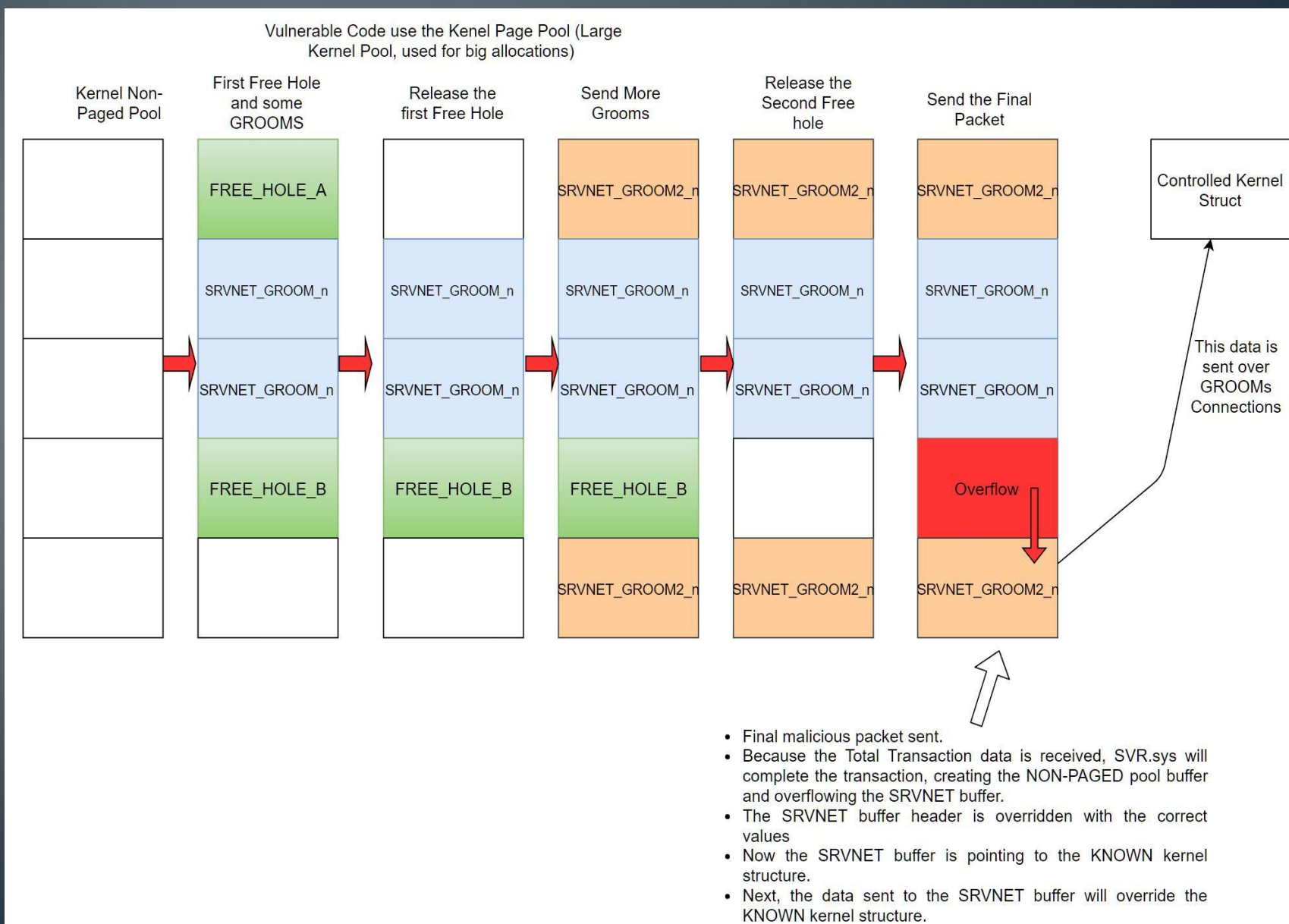
<https://packetstormsecurity.com/files/142548/MS17-010-EternalBlue-SMB-Remote-Windows-Kernel-Pool-Corruption.html>

MS17-010 general concept

- There is a buffer overflow before second memmove/memcpy operation in Srv!SrvOs2FeaToNt. The size is calculated in Srv!SrvOs2FeaListSizeToNt, with mathematical error where a DWORD is subtracted into a WORD.
- Size of NTFEA list requires Large Kernel Pool Allocation in SRV.sys
- Last iteration overwrite next memory area (SRVNET.sys). To achieve it both buffers should be aligned.
- Kernel Pool should be sprayed.
- Server returns 0xC000000D (status_invalid_parameter).
- Exploit may require several tries.

MS17-010 exploitation

From TrendMicro:



MS17-010 exploitation (2)

Command - Kernel 'com:pipe, resets=0, reconnect, port=\\.\pipe\kd_Windows_Server_2008_R2_x64' - WinDbg:6.12.0002.633 AMD64

```
1: kd> bp srv!SrvOs2FeaListToNt+0x108
1: kd> g
Breakpoint 4 hit
srv!SrvOs2FeaListToNt+0x108:
fffff880`03ec2a68 bb0d0000c0      mov     ebx, 0C000000Dh
1: kd> p
srv!SrvOs2FeaListToNt+0x10d:
fffff880`03ec2a6d ebe5      jmp     srv!SrvOs2FeaListToNt+0xf4 (fffff880`03ec2a54)
1: kd> g
Break instruction exception - code 80000003 (first chance)
ffffffff`ffd00201 cc      int     3
0: kd> dd
ffffffff`ffd00201 41c931cc b9c301e2 c0000082 bb48320f
ffffffff`ffd00211 ffd00ff8 ffffffff 89045389 058d4803
ffffffff`ffd00221 0000000a 48c28948 0f20eac1 010fc330
ffffffff`ffd00231 894865f8 00102524 48650000 a825248b
ffffffff`ffd00241 50000001 56525153 50415557 52415141
ffffffff`ffd00251 54415341 56415541 2b6a5741 2534ff65
ffffffff`ffd00261 00000010 336a5341 d1894c51 08ec8348
ffffffff`ffd00271 ec814855 00000158 24ac8d48 00000080
```

MS17-010 conclusions

- Sophisticated exploit which requires several steps to achieve reliable execution.
- Most of kernel structures used by exploit are undocumented.
- In-depth SMB protocol knowledge required.
- Redstone 1 update randomization for page table entries prevents the DEP bypass.
- Redstone 2 update (after the MS17-010 patch), the HAL heap is randomized, defeating the ASLR bypass.

Final thoughts

- Comparing these exploits we can see how Microsoft improved OS security in past few years.
- New exploit mitigation techniques make reliable exploitation much harder to achieve.
- On recently updated Windows 10 I don't expect that something similar will arise in future until new class of vulnerabilities would be introduced.
- Great work by Microsoft to introduce new mitigations based on vulnerability classes from previous OS releases.

Any questions?

Thank you

- Email: vinegrep@protonmail.com
- Twitter: [@vinegrep](https://twitter.com/vinegrep)