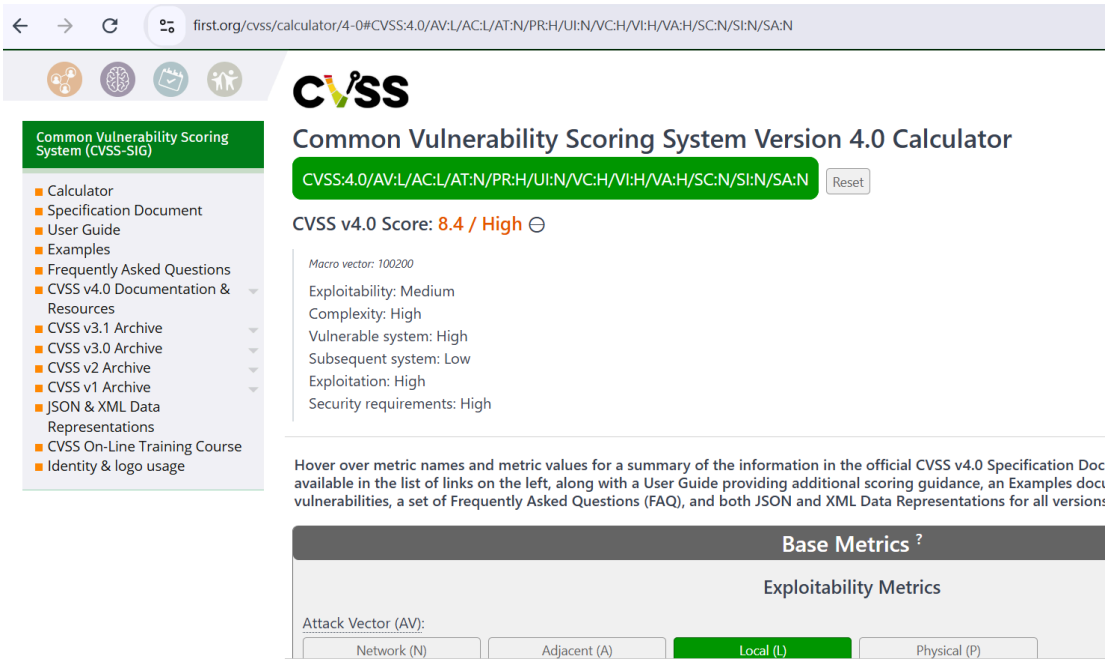
Lab 1: Threat Modelling, Kali Linux and Wireshark

1. Threat Modeling

a.



CVSS:4.0/AV:L/AC:L/AT:N/PR:H/UI:N/VC:H/VI:H/VA:H/SC:N/SI:N/SA:N

- **Attack Vector (AV: Local):** The exploit requires the attacker to already have local access to the machine to issue System Management Interrupt (SMI) calls. It cannot be launched remotely over a network.

- **Attack Complexity (AC: Low):** Once local privileges are obtained, no special conditions or uncommon configurations are required to trigger the vulnerability.

- **Attack Requirements (AT: None):** The exploit does not depend on external factors like specific hardware states or timing; it can be reliably executed once local access is present.

- **Privileges Required (PR: High):** An attacker must already have administrator or kernel-level privileges in order to access SMM and trigger the exploit.

- **User Interaction (UI: None):** The exploit runs entirely under the attacker's control and does not require any action from the victim.

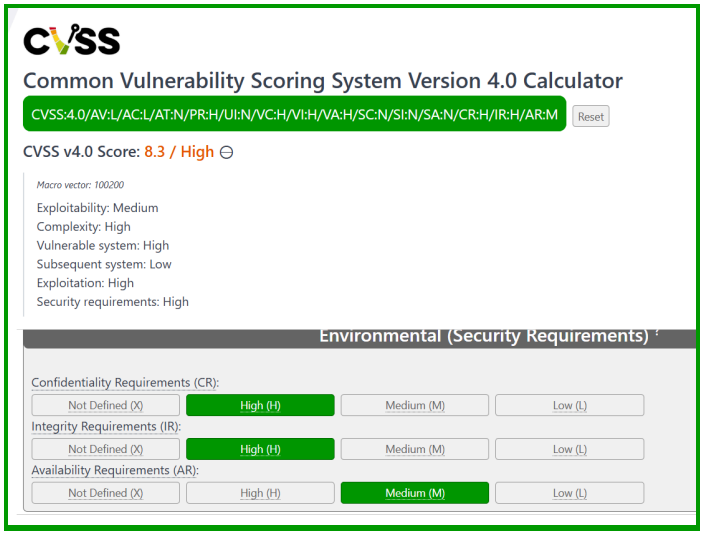
- **Confidentiality (VC: High):** The exploit allows full disclosure of the protected SMRAM contents, exposing sensitive data.

- **Integrity (VI: High):** The attacker can execute arbitrary code in SMM, enabling modification of firmware and system integrity.

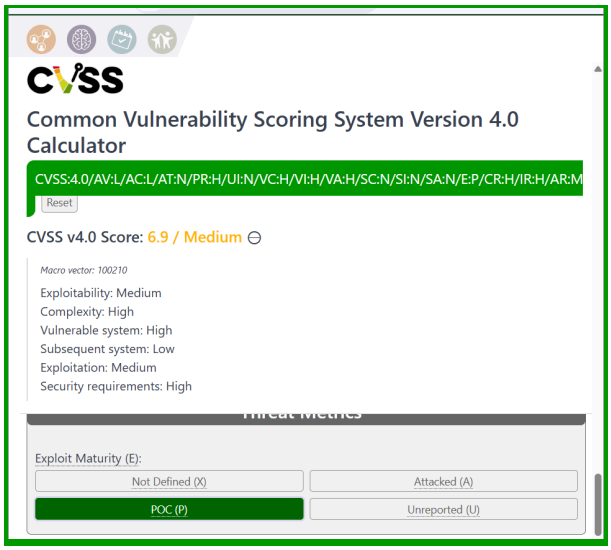
- **Availability (VA: High):** With control over SMM, the attacker could disable or crash the system entirely, impacting availability.

b.

i. Environmental Metrics (Security Requirements):
CVSS:4.0/AV:L/AC:L/AT:N/PR:H/UI:N/VC:H/VI:H/VA:H/SC:N/SI:N/SA:N/CR:H/IR:H/AR:M



ii. Threat Metric also change:
CVSS:4.0/AV:L/AC:L/AT:N/PR:H/UI:N/VC:H/VI:H/VA:H/SC:N/SI:N/SA:N/E:P/CR:H/IR:H/AR:M



- I choose proof-of-concept because there is no mention of active widespread attacks, but the Lenovo SMM exploit is well-documented and demonstrable, but not necessarily being mass exploited like a wormable bug.

2. **Kali Linux**
    a. Yes.
    b.

```
┌──(zach㉿kali)-[~]
└─$ cal
   September 2025
Su Mo Tu We Th Fr Sa
    1  2  3  4  5  6
 7  8  9 10 11 12 13
14 15 16 17 18 19 20
21 22 23 24 25 26 27
28 29 30


┌──(zach㉿kali)-[~]
└─$ ncal
    September 2025
Su     7 14 21 28
Mo  1  8 15 22 29
Tu  2  9 16 23 30
We  3 10 17 24
Th  4 11 18 25
Fr  5 12 19 26
Sa  6 13 20 27

┌──(zach㉿kali)-[~]
└─$ time

real    4210.80s
user    0.23s
sys     0.28s
cpu     0%

real    4210.80s
user    0.19s
sys     0.08s
cpu     0%
```

real - the total wall-clock time the command took (like timing with a stopwatch).

user - the amount of CPU time spent running your program's instructions in user mode.
sys - the amount of CPU time the kernel spent doing system calls on behalf of your program (like accessing the filesystem).

c. Whoami = zach ; pwd = /home/zach
d. Use ls -a to show hidden files. In Linux hidden files begin with . (dot). The ls -la command lists all files, including hidden ones (-a), in a long format (-l).
e. mkdir temp
f. cd temp. Nothing, the file is currently empty.
g. cat > new1.txt - then I typed 'First new file created'
h. cat new1.txt - First new file created
i. echo "First new file created." > new2.txt
j. echo "Append line via echo." >> new1.txt
k. cat new1.txt >> new2.txt ; cat new2.txt
l. sudo updatedb ; locate new1.txt - /home/zach/temp/new1.txt
m. cp new1.txt ~/ ; ls ~
n. Mv new2.txt ~/ ; ls ~
o. grep = Global Regular Expression Print. It searches for patterns in text files and prints matching lines. ; grep " " ~/new2.txt ; grep -n " " ~/new2.txt ; grep -c " " ~/new2.txt
p. ls -l – total 4 -rw-rw-r-- 1 zach zach 44 Sep 10 13:19 new1.txt
q. rm new1.txt
   ls -l
r. cd ~ …
   rm -r temp …
   ls -l
s. Linux ; Linux kali 6.12.25-amd64 #1 SMP PREEMPT_DYNAMIC Kali 6.12.25-1kali1 (2025-04-30) x86_64 GNU/Linux
t. users - zach
u. sudo adduser Bob
v. student (sorry I didn't ready v before already typing student as the password)
w. less ~/new1.txt
x. history
   1 ls
   2 ls -a
   3 mkdire temp
   4 mkdir temp

```
    5  cd temp
    6  ls
    7  cat new2.txt
    8  sudo updaatedb
    9  locate new1.txt
   10  sudo apt update
   11  sudo apt install -y plocate
   12  sudo updatedb
   13  locate new1.txt
   14  cp new1.txt ~/
   15  ls ~
   16  mv new2.txt ~/
   17  ls ~
   18  grep "space" ~/new2.txt
   19  grep " " ~/new2.txt
   20  grep -n " " ~/new2.txt
   21  grep -c " " ~/new2.txt
   22  cd ~/temp
   23  ls -l
   24  rm new1.txt
   25  ls -l
   26  cd ~
   27  rm -r temp
   28  ls -l
   29  uname
   30  uname -a
   31  users
   32  sudo adduser Bob
   33  less ~/new1.txt
```

y. .

```
┌──(zach㉿kali)-[~]
└─$ nano ~/new2.txt

┌──(zach㉿kali)-[~]
└─$ cat new2.txt
First new file created.
First new file createdAppend line via echo.
This line was added in nano.

┌──(zach㉿kali)-[~]
```

└─$

3. **Recover/Break Passwords**

    a. Comparing the same hash type (MD5) for the two passwords.

    pass@word (MD5) → 2.68 s
    Pass12word (MD5) → 5.77 s

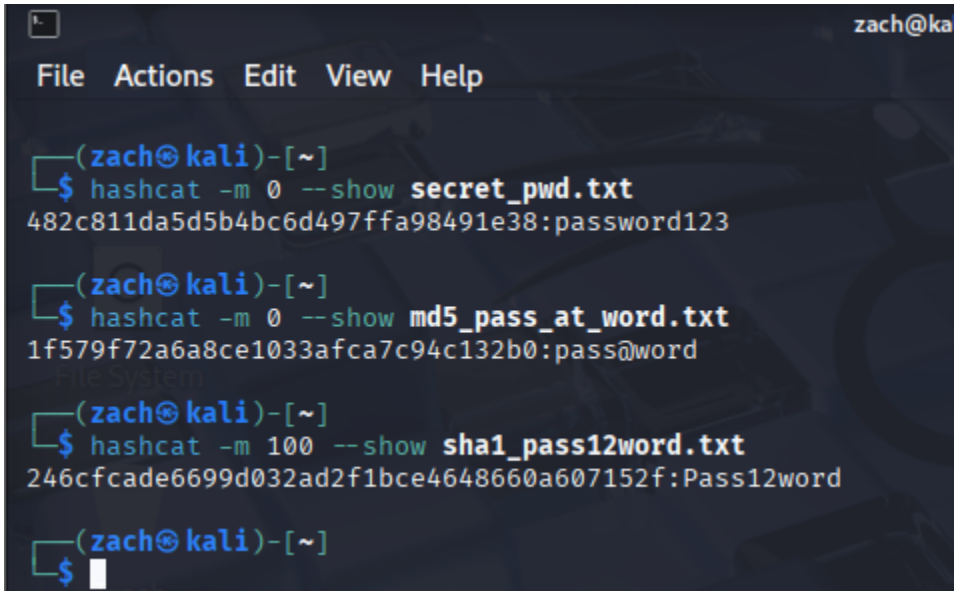
Pass12word was more difficult/took longer because the mix of uppercase + digits increases search space vs pass@word (lowercase + one symbol), so it appears later in the wordlist/mutations.

    b. Hash Pass12word with SHA-1 compared to MD5

    Pass12word (SHA-1) → 6.65 s
    Pass12word (MD5) → 5.77 s

SHA-1 took a bit longer than MD5 on this setup. I disabled the potfile with --potfile-disable when timing, so hashcat actually performed the attack each time rather than instantly returning from cache.

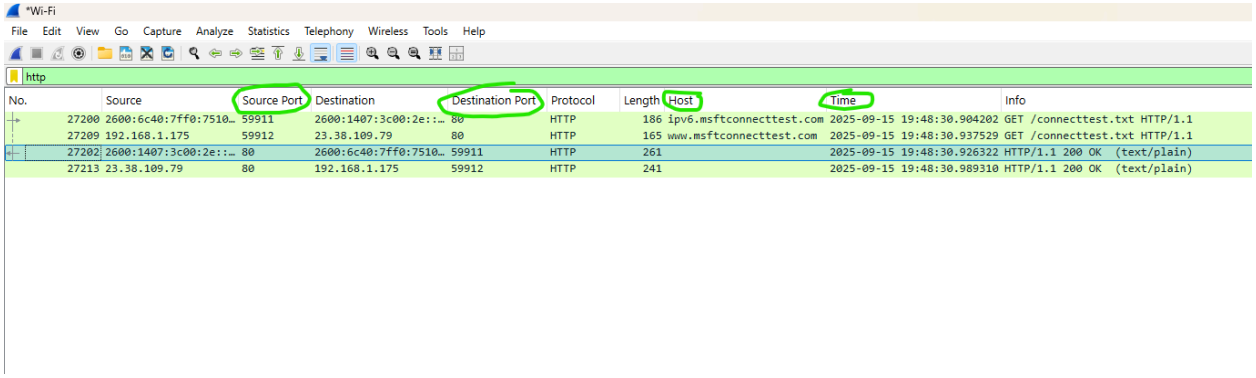
    c.



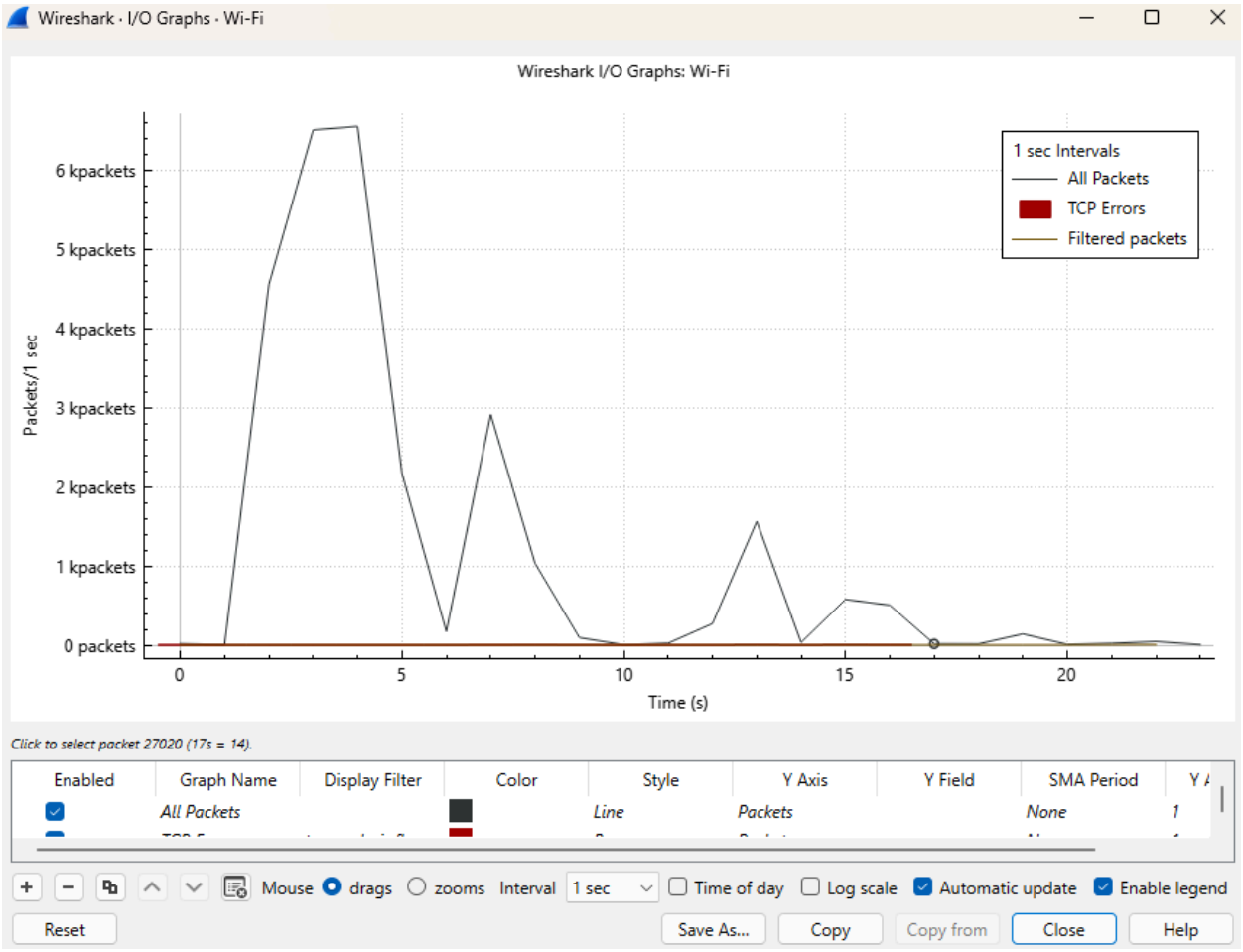
4. Wireshark

A. Host Machine

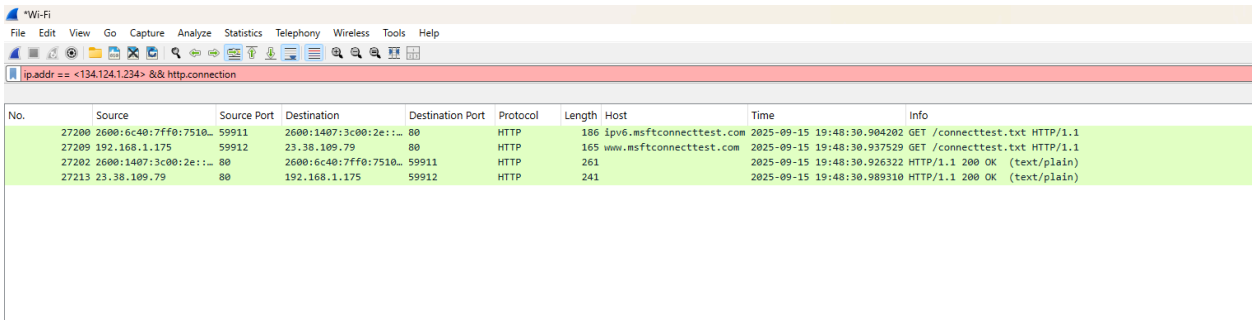
1. Done
2. I know I started with HTTP because you can see a GET / ... over port 80. I know it switched to HTTPS because after that, all traffic shows up as TLS over port 443 (encrypted).
3. First http packet with GET:
   i. (Frame 3902 → Ethernet II → IP Version 4 → TCP Src Port: 56133, Dst Port: 80 → HTTP)
   ii. IP
   iii. Host: www.msftconnecttest.com\r\n
   iv. In the TCP layer → Destination port = 80
   v. Content-Type: text/plain
   Last-Modified (or equivalent): Thu, 11 Sep 2025 18:05:36 GMT
   vi. Source Port 56133
   Destination Port: 80 (HTTP)
   vii. TCP Destination Port: 443
   Because HTTPS encrypts traffic using TLS/SSL, and by standard convention, it runs on port 443, not port 80 (which is reserved for unencrypted HTTP).
4. Change the display
   i. Microsoft's Server msftconnecttest.com was deliberately omitted for security reasons.



5. 4



6.



7. SYN → client to server (flags: S)
   SYN, ACK → server back to client (flags: S, A)
   ACK → client to server (flags: A)

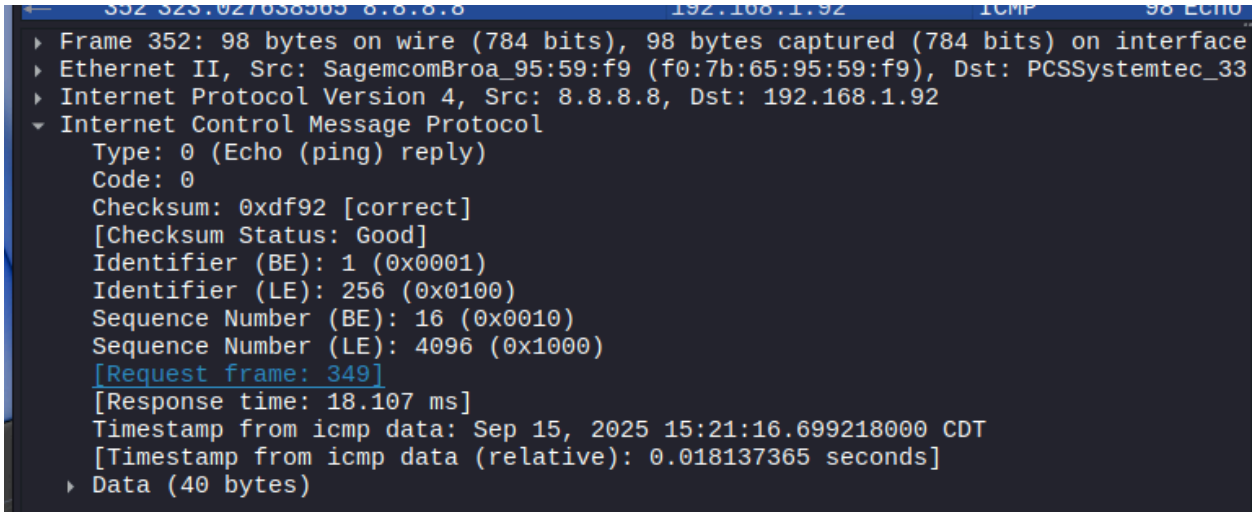
B. VM to ridge adapter

1. .
    i. Frame 311
    Ethernet II
    Internet Protocol Version 4
    Internet Control Message Protocol

    ii. Type 8, Code 0 = Echo Request
    Type 0, Code 0 = Echo Reply

    iii. This means Google's DNS server (8.8.8.8) is replying to my ping request. It received my Echo Request and is sending back an Echo Reply.

    iv. The sequence number in the Echo Request and Echo Reply match. The time since request field shows the response time (in milliseconds).



    v. I don't see any TCP details here because the traffic I captured is ICMP, not TCP. ICMP is used for diagnostics like ping, and it doesn't have ports or the TCP header fields. That's why no TCP information shows up in these packets.

    vi. I see that my Echo Request packets have a TTL of 64, while the Echo Replies from 8.8.8.8 have a different TTL. They are not the same because different systems use different starting TTL values, and routers along the path decrement TTL as packets move across the internet.

    vii. I don't see any source or destination port numbers because ICMP doesn't use ports. Ports are part of TCP and UDP at the transport

layer, but ICMP works at the network layer and only uses type and code fields instead of ports.

viii. I can see that the Echo Request has an identifier and sequence number, and the Echo Reply contains the same values. This matching confirms the reply is for my specific request.

ix. Its using Spectrum (Charter). It sends UDP probes with increasing TTL values, and routers reply with ICMP Time Exceeded messages.