CS 473: Network Security - Task 2 Guide

Tips :

- Go through the comments in the defenses.py file, you will find most of the hints there.
- Try printing out program memory, stack pointer, etc at different points of the execution to get a better idea of the working of the program.

Running defenses.py

Task 2 provides you with a python file defenses.py which can be run using the following command:

```
$ python3 defenses.py
```

Try giving this input: `pwnd000011112222333362`

Program Simulator?

You might have played old nintendo 3ds or gameboy games on your mobile or PC using a simulator, these simulators are programs that trick the game codes into behaving like they are running on a nintendo 3ds or gameboy. While, in reality, they are just Windows, Linux programs that understand the instructions for these Nintendo 3ds, or gameboy Similarly, we are simulating a program here.

Code Structure Explanation

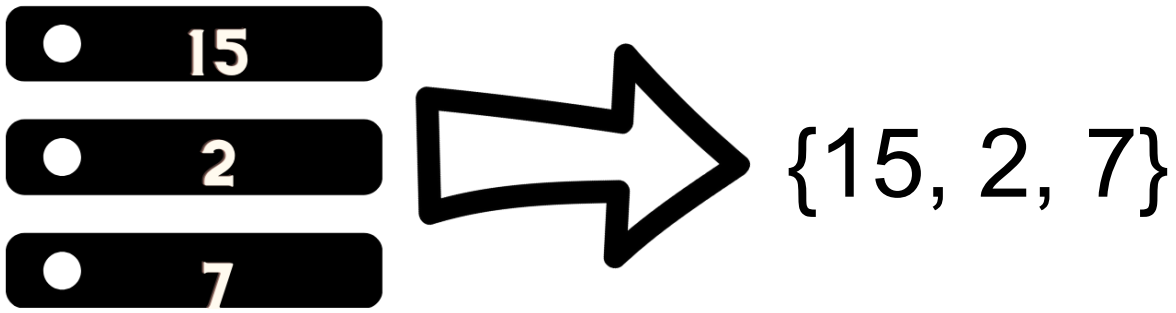
In defenses.py we are creating a simple simulator.

```
class RunProgramSimulator:
    start_address = None
    mem_size = None
    execstack = None
    stack_start = None
    memory = None
    f_addr = {}
```

It simulates the stack of a real program. It has its own memory, jump table (more details on what this is are in the table below) and registers.

```
code = [
    'func main',
    'call vuln',
    'puts good',
    'exit',
    'func vuln',
    'aloc 5',
    'gets',
    'movb rax *esp',
    'dloc 5',
    'retb'
]
jumpTable = {}
registers = {
    'esp': 0,
    'rip': 0,
    'rax': 0
}
```

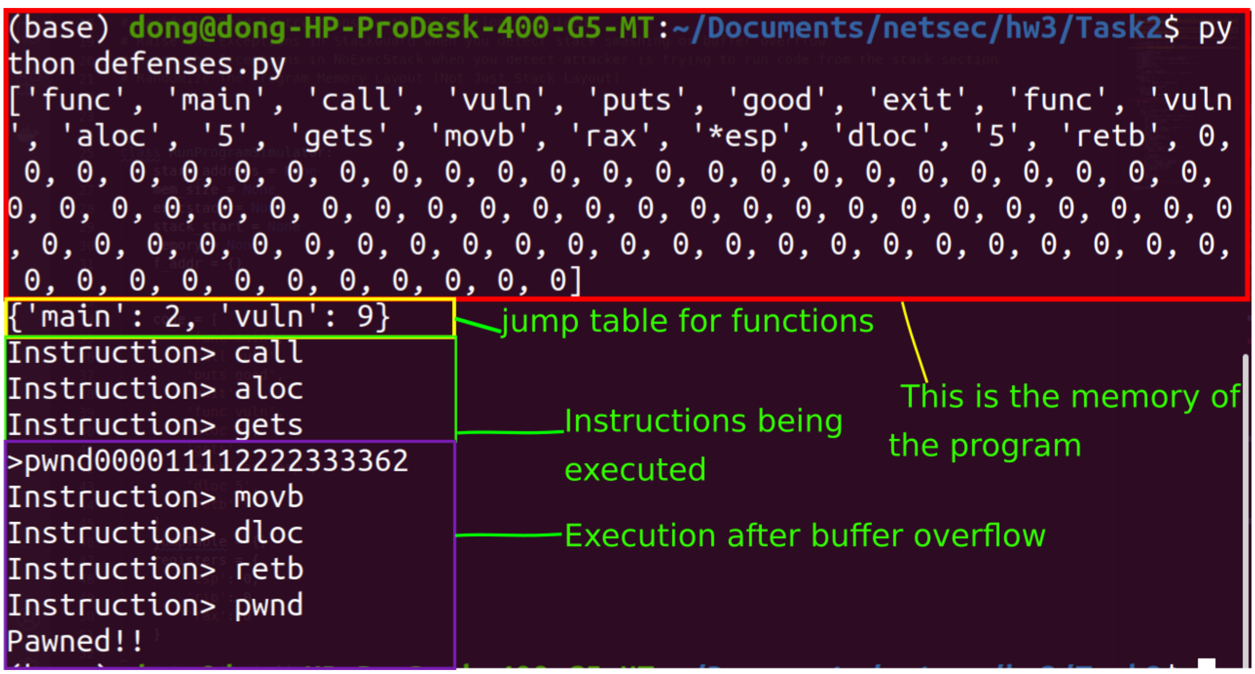
Variable Name	Purpose
memory	This list represents the memory of the program that is being run. It is filled with zeros.
Stack	Stack in this program is represented by an array and each array element in this magical simulation world is considered worth one byte. Stack starts from the *stack_start* index in the memory, and grows by decrementing. Here is an example of how some values from the stack would look in the array in python array.



code	The code for the program must not be changed, and is stored in the class variable code.
jumpTable	This variable is a dictionary that stores the addresses of the function in the program, keys are function names and the values are the function indices(addresses) in the self.memory.
registers	This is another dictionary in the class RunProgramSimulator, it stores the values of the registers used by the simulators, these values are looked up using register name as keys.

Goal of the Task

However, there is a problem, this simulation is vulnerable to buffer overflow attacks. Here is a sample run of the program.



You are supposed to code the following defenses for this simulation:

- Raise the Exceptions in StackGuard when you detect stack smashing or buffer overflow
- Raise the Exceptions in NoExecStack when you detect attacker is trying to run code from the stack section
- Randomize The Program Memory Layout (Not Just Stack Layout)