Assignment 6-3 [OPTIONAL]: Navigate a CPU to be Hands on with a Buffer Overflow

Start Assignment

Due Jun 13 by 11:59pm **Points** 0 **Submitting** a text entry box
Available May 10 at 12am - Jun 13 at 11:59pm about 1 month

Ethical Hacking 200

Assignment 6-3: Navigate a CPU to be Hands on with a Buffer Overflow

Download the Assignment Files:

- CPU and BufferOverflow emulator.zip
- Hacking 200 - Assignment 6-3.pdf

This assignment is extra credit! It is worth 10 points. This implies you could get a extra 10 points or choose to replace another assignment (point wise) with this one.

Recap

In class we covered how a basic CPU functions and core concepts within it as it processes data. Depending on the specific CPU and architecture details will vary, such as the exact “OpCodes” available, structure, and even logic. However the core concepts such as reading OpCodes, instructions, registers, accumulators, counters, etc are key. If your path takes you into binary analysis, native code, forensics, CPUs, etc, this will be foundational.

Your Assignment:

- **Learn to Edit and Run Python Scripts –** if you have not already. The scripts were written in python version 3.7.9, as such run it using at least python3. https://www.python.org/downloads/
- **Buffer Overflows –** you will combine your knowledge you have accumulated till now to do some real world steps in working with a buffer overflow (of course simplified). The scenario is if you had found a buffer overflow, as a simple example say typing in too long a username in an input field, and a

program crashed with a buffer overflow.
What you will be doing is interacting with the CPU emulator in “CPU and BufferOverflow emulator.zip” through “run.py”.

- **Make it into an attack –** Turn the overflow “vulnerability” into an actual attack, so far all you did was break stuff. You need to figure out the “payload”. As an attacker, simply having the application crash is not useful to you, you want it to execute code or perform an action that benefits you.
  In “run.py” you will follow the tutorial if needed. At a minimum, you will then consider modifying the following values, save the file, rerun it, figure out what effect your changes are having on the CPU, what does what, repeat, experiment:
  - introduce_bufferoverflow_memory =
  - introduce_bufferoverflow_ProgramCounter =
  - bufferoverflow_value =
  - bufferoverflow_at =
- The original program behavior prints 1 to 10. **Your goal is to have it print “700” instead of the number “1” in the count down**. It does not matter if it continues to print the correct values or even stops after replacing the number ‘1’.
- You need to figure out what *combination* of the different values will work. Keep in mind “bufferoverflow_value” is the data that is overflowing. In the theoretical scenario this is data from the too long username that left the memory space it was meant to be in and is now overflowing into memory it should not be in, as it processes through the CPU. You are at the step in the theoretical scenario where you are trying to debug this overflow, and need to figure out what data you want it to overflow and where, such as to overwrite the counter or the current OpCode.

What to turn in:

- Two answers
  - The combination of “values” that worked for you, e.g:
    - introduce_bufferoverflow_memory = True
    - bufferoverflow_value = 999999
    - bufferoverflow_at = 101
  - A short description of what that combination of values is doing to cause the value 700 to print out. E.g:
    It overwrites value abc, which then produces instruction 777. 777 starts with 7, which has the affect of abc and is memory location 77. This then cause the effect of getting abc, doing xyz, and 700 is printed. (but obviously a more sensical answer)

- Any reasonable text format is acceptable

Im having a really hard time!

As always, the best thing you can do is reach out as early as possible to the instructor.

Feel free to discuss in group chat, but please do not give out examples or answers. An appropriate level of information in group chat would be sharing information sources discovered (e.g: hey I found this nice explanation on what “bits” vs “bytes” are). However, you should NOT directly address answer in group chat (e.g: hey did you know “x0b” on line 37 means…). It should be up to each student to read, absorb information, and build up and then apply their own practical skills to be able to do this on their own.

Hint: read all the comments in the code, they are mean to serve as a tutorial to help you understand what is going on.

Hint: Follow the instruction in the code such as “any comments which look like this ‘# >>>>> do something <<<<<<’ shows where the code file is meant for you to modify,”.

Hint: Pay attention to the explanation of the instructions, opcodes, and locations of things. This is a very friendly and helpful tool at your disposal.

Hint: If possibly do NOT “word wrap” the text being printed by the python scripts, it will make the helpful explanations less helpful. It also produces a “log.txt” which mirrors what is being printed to console. You can use the log file to help with reading the output if its easier.