Assignment 5

Buffer Overflow Attack

For this assignment, you are given a compiled executable and its source code. The program reads from the input and then prints it in the output.

You have to exploit the buffer overflow vulnerability present in the application and execute custom shellcode with the ultimate goal of achieving code execution by spawning a shell.

Tasks

1. Run the program and understand the workflow
2. Examine the source code and correlate it with the execution (Use the built-in comments to direct your attention)
3. Find the vulnerability and understand how it could be leveraged to achieve the goal
4. Place a custom shellcode on a memory segment that you control
5. Find the amount of data until you get control of the **RIP** register (why is that important?)
6. Use the vulnerability and redirect code execution to said memory segment so that the program executes your custom shellcode!

Write a python script that creates the desired payload (and write it into a file). Then feed your payload into the input of the program as so: **cat ./payload.bin | ./bof** (You will need some additional steps, found in **Resources**)

Tips

1. Learn to use **GDB** and also a helper (GEF or pnwdbg)
2. **DEBUG**, if it doesn't work find where it fails and why
3. Understand how the **stack** works, and how functions are **called** and how they **return**
4. Search the web for shellcodes (or read **Resources**)
5. Do one thing at a time and understand how and why it's happening (or not):
    a. Does your payload end up in the right place?
    b. Does your payload end up intact?
    c. Are you returning to the correct address?
    d. Is the **syscall** working? Why not?
6. If you get stuck, ask for help!

Notes

1. The program is compiled in such a way that:
    a. allows memory segments to be written and then executed
    b. Program symbols (global variables, functions, etc) will ALWAYS remain the SAME
2. Use tools such as **objdump** or **readelf** to find those addresses (aka **symbols**).

3. This program was compiled for 64-bit machines and was compiled on Ubuntu 22.04 (although you should be able to run it anywhere)
4. You don't need to compile the source code. But, If you do, the addresses mentioned above will change (but will remain the same during subsequent executions).
5. Do not bother trying to understand everything about the program. Use the comments to guide you
6. Google (or any other search engine) is your best friend!
7. This type of pwning is called a **ret2shellcode** technique, because you return to the shellcode

Resources

- https://www.youtube.com/playlist?list=PLhixgUqwRTjxglIswKp9mpkfPNfHkzyeN (watch videos: 0x0A - 0x0E, notice that this is for 32-bit machines)
- https://shell-storm.org/shellcode/index.html - Shellcodes - Use Intel x86-64
- http://www.phrack.com/issues/49/14.html#article original Phrack Article (pdf here)
- Guidance here

Deliverables

- The python script that creates a working payload, thus exploiting the program
- A Readme.txt file that:
    - explains the process you followed
    - what does the shellcode do
    - proof that your exploit works
    - and anything else you deem important to share