3. 32 bit - Return to Libc

Exercise 3: Mitigation

Exercise 3: Mitigation

In this exercise, you will explore different mitigations that can be employed to protect against return-to-libc attacks in a 32-bit binary. These mitigations typically include:

1. **Stack Canaries**: Special values placed on the stack that help detect stack overflows.
2. **Non-executable Stack (NX)**: Ensures that the stack cannot be executed.
3. **Address Space Layout Randomization (ASLR)**: Randomizes the memory addresses used by system and application processes, making it difficult to predict the location of return-to-libc targets.
4. **Position Independent Executables (PIE)**: Ensures that the entire binary is randomized in memory.

Steps:

1. **Enable Stack Canaries:**

   - Use the `fstack-protector` or `fstack-protector-all` option during compilation to enable stack canaries.

   ```
   gcc -fstack-protector -o vulnerable vulnerable.c
   ```

2. **Enable Non-executable Stack:**

   - By default, modern systems have this enabled. You can ensure this by compiling with the `z noexecstack` option.

   ```
   gcc -z noexecstack -o vulnerable vulnerable.c
   ```

3. **Enable ASLR:**

- ASLR can be enabled in the Linux kernel by setting the `/proc/sys/kernel/randomize_va_space` to 2.

```
echo 2 | sudo tee /proc/sys/kernel/randomize_va_space
```

4. **Enable PIE:**

- Use the `fPIE` and `pie` options during compilation.

```
gcc -fPIE -pie -o vulnerable vulnerable.c
```

5. **Test Your Mitigations:**

- After applying the above mitigations, test your binary to see how the mitigations affect the exploitation process. Try to perform a return-to-libc attack and observe how each mitigation provides a layer of protection.

Example:

Let's create a vulnerable C program and apply these mitigations.

Vulnerable Program (vulnerable.c):

```
#include <stdio.h>
#include <string.h>
#include <unistd.h>

void vuln() {
    char buffer[64];
    read(STDIN_FILENO, buffer, 128);
}

int main() {
    vuln();
    return 0;
```

Compile without Mitigations:

```
gcc -o vulnerable vulnerable.c
```

Compile with Mitigations:

```
gcc -fstack-protector -z noexecstack -fPIE -pie -o vulnerable
_secure vulnerable.c
```

Testing:

1. Run both `vulnerable` and `vulnerable_secure` binaries.
2. Attempt to exploit both binaries using a return-to-libc attack, remember from Exercise 2.

By comparing the results, you will observe how each mitigation helps in preventing the attack.

Conclusion:

These mitigations significantly enhance the security of binaries and make exploitation mch more challenging. Understanding and implementing these defenses is crucial for developing secure software.