

USER

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>

#define BUFSIZE 100

long increment(long in) {
    return in + 1;
}

long get_random() {
    return rand() % BUFSIZE;
}

int do_stuff() {
    long ans = get_random();
    ans = increment(ans);
    int res = 0;

    printf("What number would you like to guess?\n");
    char guess[BUFSIZE];
    fgets(guess, BUFSIZE, stdin);

    long g = atol(guess);
    if (!g) {
        printf("That's not a valid number!\n");
    } else {
        if (g == ans) {
            printf("Congrats! You win! Your prize is this print statement!\n\n");
            res = 1;
        } else {
            printf("Nope!\n\n");
        }
    }
    return res;
}

void win() {
    char winner[BUFSIZE];
    printf("New winner!\nName? ");
```

```

fgets(winner, 360, stdin);
printf("Congrats %s\n\n", winner);
}

int main(int argc, char **argv){
setvbuf(stdout, NULL, _IONBF, 0);
// Set the gid to the effective gid
// this prevents /bin/sh from dropping the privileges
gid_t gid = getegid();
setresgid(gid, gid, gid);

int res;

printf("Welcome to my guessing game!\n\n");

while (1) {
res = do_stuff();
if (res) {
win();
}
}

return 0;
}

```

How to solve this guessing game?

ASSISTANT

This is a pwn challenge, and the goal is to exploit the vulnerability in the code to solve the game.

Identify Vulnerability:

The vulnerability lies in the `win()` function, specifically in the `fgets(winner, 360, stdin)` line. The `fgets()` function is being used to read a string from the standard input, but the buffer size (`winner`) is declared as `BUFSIZE` (100), which is smaller than the length that `fgets()` is trying to read (360). This creates a buffer overflow vulnerability.

Exploit Vulnerability:

To exploit this vulnerability, we can craft a payload that will overflow the buffer and execute our own code. Here's an example exploit:

1. Compile the code and run it.
2. When prompted to guess a number, enter a number that will win the game (the number is incremented by 1, so entering a number that is 1 less than the random number will work).

3. When prompted to enter the winner's name, enter a payload that will overflow the buffer. For example, we can use a payload that will execute a shell and give us a shell prompt:

* Payload: `A` * 100 +

`\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\x50\x89\xe2\x53\x89\xe1\xb0\x0b\xcd\x80`

Step-by-step instructions:

1. Run the program and play the game until you win.
2. When prompted to enter the winner's name, enter the payload above.
3. The payload will execute a shell, and you will be dropped into a shell prompt.
4. From the shell prompt, you can explore the file system and find the flag (if one exists).

Fix Vulnerability:

To fix this vulnerability, we can modify the `win()` function to use a safe input function, such as `fgets(winner, BUFSIZE, stdin)`, and also add input validation to ensure that the user input does not overflow the buffer.

Additional Instructions:

This is a classic example of a buffer overflow vulnerability, and the exploit payload is written in assembly code (x86 architecture). To learn more about buffer overflow vulnerabilities and how to write exploit payloads, I recommend checking out online tutorials and documentation on the topic.

USER

That does not work as it would take too long

ASSISTANT