CNS ASSIGNMENT 7
DIV 2
BATCH : B4

Group members:

112003132 Umesh Shirsat

142103013 Snehal Shinde

142103002 - Anuj Mohite

PREVENTION OF BUFFER OVERFLOW ATTACK

Buffer overflow is a common software vulnerability that occurs when data overflows from one buffer to adjacent memory locations, potentially leading to unexpected and harmful consequences. In this report, we examine a simple demonstration of a buffer overflow attack and the prevention mechanisms implemented in the provided C code.

Code Explanation:

1. **Buffer Declaration:**
   - The code defines a character array called `buffer` with a size of `BUFFER_SIZE` (set to 5). This buffer is used to store character data.
2. **Command-Line Arguments:**
   - The program accepts command-line arguments. If no arguments are provided (`argc < 2`), a usage message is displayed, and the program exits.
3. **Buffer Overflow Prevention:**
   - Before copying data to `buffer` using `strncpy`, the code checks if the length of the argument (`argv[1]`) is

greater than or equal to `BUFFER_SIZE`. If the input exceeds the buffer size, the program prints a message indicating that the input is too large and does not copy the data.
  - If the input size is within the buffer size limit, the code uses `strncpy` to copy the data to `buffer`. The size is limited to `BUFFER_SIZE - 1` to ensure that there is room for the null-terminating character. This helps prevent a buffer overflow.
  - The code then explicitly null-terminates `buffer` to ensure it is a valid null-terminated C string.
  - The content of `buffer` is printed along with a message indicating that `strncpy()` was executed.

Buffer Overflow Attack:

A buffer overflow attack typically occurs when an attacker provides input that exceeds the size of the buffer, causing data to overflow into adjacent memory. In some cases, an attacker can inject malicious code or exploit vulnerabilities to take control of the program or even the entire system.

Buffer Overflow Prevention:

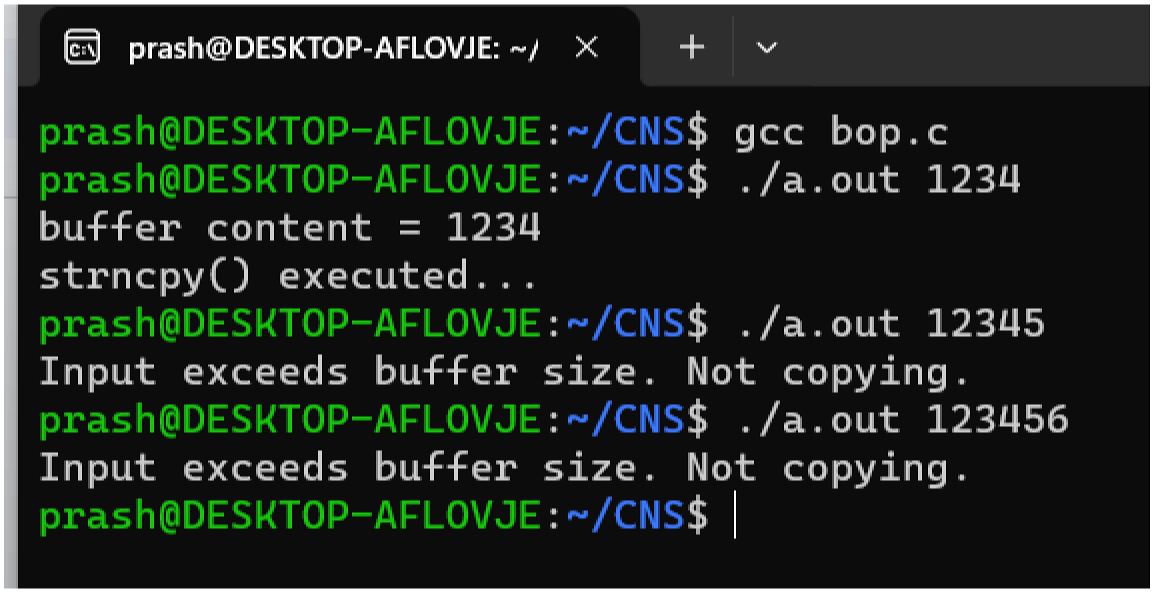
The code demonstrates a fundamental prevention technique for buffer overflow:

1. **Boundary Checking:**
   - Before copying data into the buffer, the code checks the length of the input against the buffer size. If the input exceeds the buffer size, the copying is prevented, and an error message is displayed.

2. **Null-Termination:**
   - The code ensures that the buffer is null-terminated by explicitly placing the null character (`'\0'`) at the end of the copied data. This is essential for safely working with C-style strings.

Code :

```
#include <stdio.h>

#include <string.h>

#include <stdlib.h>


#define BUFFER_SIZE 5 // Define the buffer size


int main(int argc, char *argv[]) {

    char buffer[BUFFER_SIZE];


    if (argc < 2) {

        printf("strcpy() NOT executed....\n");

        printf("Syntax: %s <characters>\n", argv[0]);

        exit(0);

    }


    if (strlen(argv[1]) >= BUFFER_SIZE) {
        printf("Input exceeds buffer size. Not copying.\n");
    } else {
        strncpy(buffer, argv[1], sizeof(buffer) - 1);
        buffer[sizeof(buffer) - 1] = '\0'; // Ensure null-terminated string

        printf("buffer content = %s\n", buffer);
        printf("strncpy() executed...\n");
    }

    return 0;
}
```



```
prash@DESKTOP-AFLOVJE:~/CNS$ gcc bop.c
prash@DESKTOP-AFLOVJE:~/CNS$ ./a.out 1234
buffer content = 1234
strncpy() executed...
prash@DESKTOP-AFLOVJE:~/CNS$ ./a.out 12345
Input exceeds buffer size. Not copying.
prash@DESKTOP-AFLOVJE:~/CNS$ ./a.out 123456
Input exceeds buffer size. Not copying.
prash@DESKTOP-AFLOVJE:~/CNS$
```

Conclusion:

The provided C code demonstrates a simple example of preventing buffer overflow by performing boundary checks and null-terminating the buffer. While this is a basic preventive measure, real-world buffer overflow vulnerabilities can be more complex and require advanced protection mechanisms. Developers should follow secure coding practices, use safer functions like `strncpy`, and implement additional security measures to safeguard against buffer overflow attacks.