Final Project CTF Challenge

CSE 5272 Cyber Threats
Group Members: Garrett Gmeiner, Maria Linkins-Nielson, Logan Klaproth

Source Code (vuln.c)

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

void vuln() {
    int password_size = 0xa;
    char buf1[8];
    char secret[8]="12345678";
    char buf2[8];
    printf("User >>> ");
    fflush(stdout);
    read(0, buf1, password_size);
    printf("Password >>> ");
    fflush(stdout);
    read(0, buf2, password_size);
    if (strncmp(secret, "CSE5272!",8) == 0) {
        printf("\nYou have won!\n");
    } else {
        printf("\n<<< Incorrect password: %s\n",&secret);
    }
}

int main (int argc, char *argv[]) {
    vuln();
}
```

How to compile

```
$ gcc vuln.c -o vuln
```

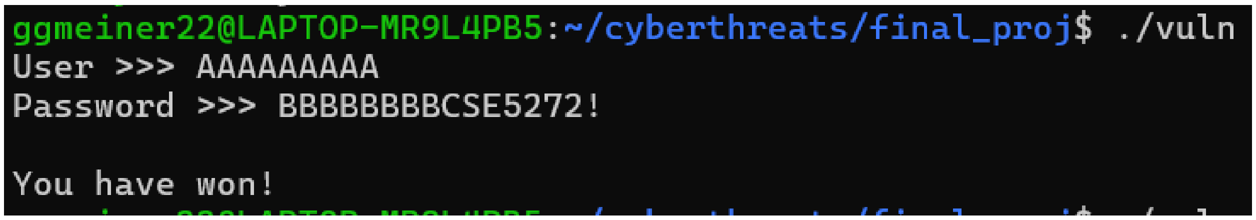
or if the OS doesn't allow:

```
$ gcc -std=gnu89 -fno-stack-protector -z execstack vuln.c -o vuln
```

Run the object file

```
./vuln
```

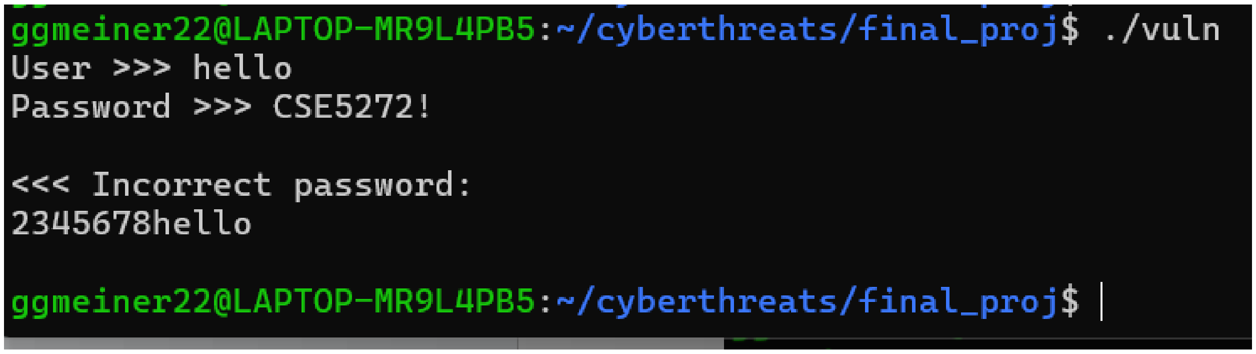
Solution



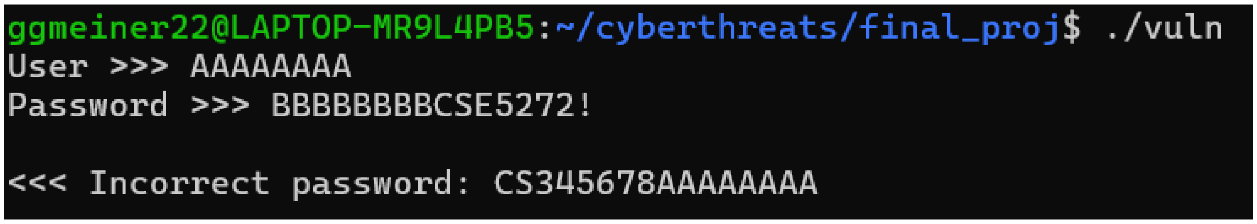
```
ggmeiner22@LAPTOP-MR9L4PB5:~/cyberthreats/final_proj$ ./vuln
User >>> AAAAAAAAA
Password >>> BBBBBBBBCSE5272!

You have won!
```

Must have a username of 9 bytes
Password must use 8 bytes to fill buf2 and the remaining 8 slip into the variable secret

How it fails

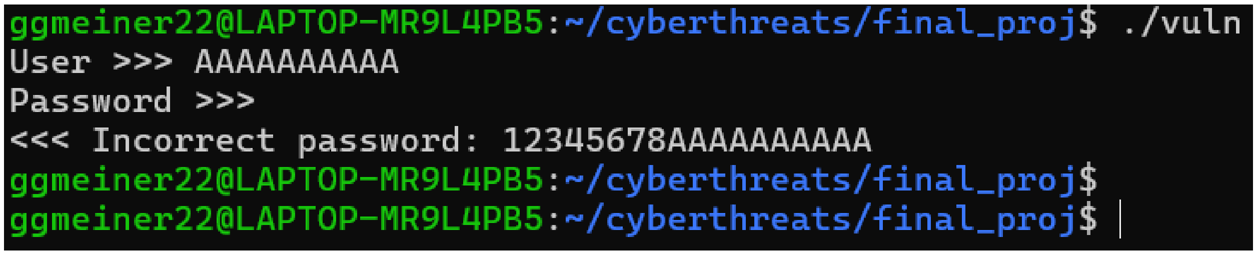


```
ggmeiner22@LAPTOP-MR9L4PB5:~/cyberthreats/final_proj$ ./vuln
User >>> hello
Password >>> CSE5272!

<<< Incorrect password:
2345678hello

ggmeiner22@LAPTOP-MR9L4PB5:~/cyberthreats/final_proj$
```



```
ggmeiner22@LAPTOP-MR9L4PB5:~/cyberthreats/final_proj$ ./vuln
User >>> AAAAAAAA
Password >>> BBBBBBBBCSE5272!

<<< Incorrect password: CS345678AAAAAAAA
```

Username less than or equal to 8 bytes


```
ggmeiner22@LAPTOP-MR9L4PB5:~/cyberthreats/final_proj$ ./vuln
User >>> AAAAAAAAAA
Password >>>
<<< Incorrect password: 12345678AAAAAAAAAA
ggmeiner22@LAPTOP-MR9L4PB5:~/cyberthreats/final_proj$
ggmeiner22@LAPTOP-MR9L4PB5:~/cyberthreats/final_proj$
```

Username of more than 10 bytes

Securing the Source Code

In this project, we will discuss three (3) ways to secure the vulnerability in vuln.c. Below are three methods to secure the code against buffer overflow attacks. Each method involves a different approach to ensuring that no more data is written than the buffers can hold.

1. Input Length Validation

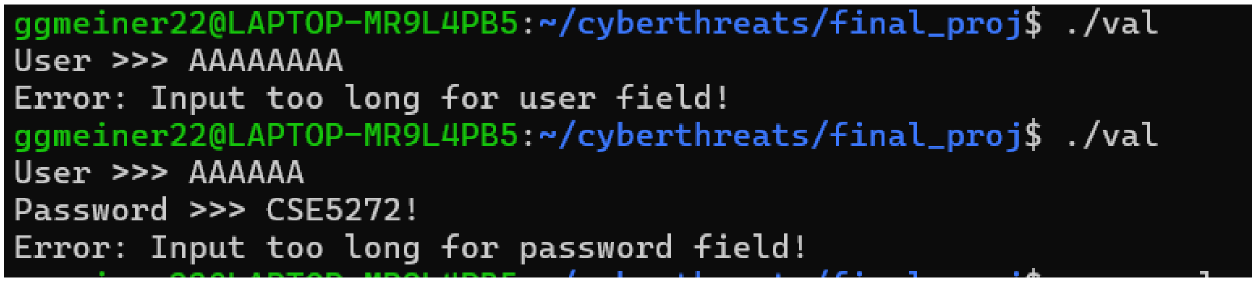
Before processing the input, it should check that the number of bytes read does not exceed the buffer size. If it does, it should reject the input or handle it appropriately.

Source Code

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

void vuln() {
    char tmp[100];   // Temporary buffer to hold input
    char buf1[8];
    char secret[8] = "12345678";
    char buf2[8];
    ssize_t bytes_read;

    printf("User >>> ");
    fflush(stdout);
    bytes_read = read(0, tmp, sizeof(tmp));
    if (bytes_read < 0) {
        perror("read error");
        exit(EXIT_FAILURE);
    }
    tmp[bytes_read] = '\0';
    if (bytes_read >= sizeof(buf1)) {
        fprintf(stderr, "Error: Input too long for user field!\n");
        exit(EXIT_FAILURE);
    }
    strcpy(buf1, tmp);

    printf("Password >>> ");
    fflush(stdout);
    bytes_read = read(0, tmp, sizeof(tmp));
    if (bytes_read < 0) {
        perror("read error");
        exit(EXIT_FAILURE);
    }
    tmp[bytes_read] = '\0';
    if (bytes_read >= sizeof(buf2)) {
        fprintf(stderr, "Error: Input too long for password field!\n");
        exit(EXIT_FAILURE);
    }
    strcpy(buf2, tmp);

    if (strncmp(secret, "CSE5272!", 8) == 0) {
        printf("\nYou have won!\n");
    } else {
        printf("\n<<< Incorrect password: %s\n", secret);
    }
}

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

Solution

Due to the input length validation, buffer overflows no longer pose a threat. This is because the bytes read is checked after each input stream.



```
ggmeiner22@LAPTOP-MR9L4PB5:~/cyberthreats/final_proj$ ./val
User >>> AAAAAAAA
Error: Input too long for user field!
ggmeiner22@LAPTOP-MR9L4PB5:~/cyberthreats/final_proj$ ./val
User >>> AAAAAA
Password >>> CSE5272!
Error: Input too long for password field!
```

2. Stack Canary

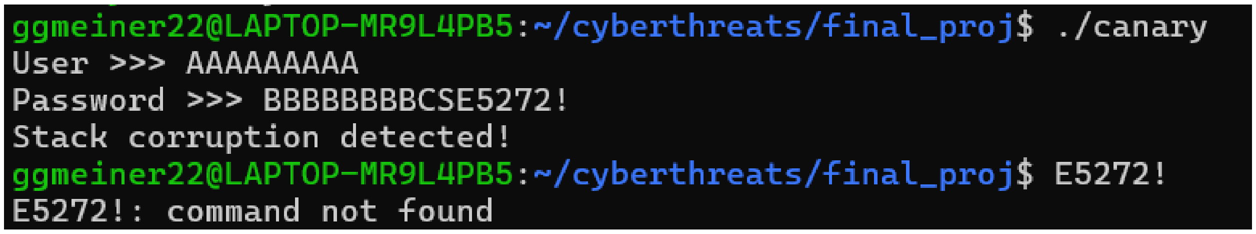
Stack canaries are predetermined values placed between local variables and the return address on the stack. If a buffer overflow changes the canary, the program detects the change and aborts execution to prevent exploitation. Modern compilers can automatically insert these protections using flags like -fstack-protector.

Source Code

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

#define CANARY 0xDEADBEEF

void vuln() {
    int password_size = 0xa;
    char buf1[8];
    char secret[8] = "12345678";
    char buf2[8];
    unsigned int canary = CANARY;

    printf("User >>> ");
    fflush(stdout);
    read(0, buf1, password_size);
    printf("Password >>> ");
    fflush(stdout);
    read(0, buf2, password_size);

    if (canary != CANARY) {
        fprintf(stderr, "Stack corruption detected!\n");
        exit(EXIT_FAILURE);
    }

    if (strncmp(secret, "CSE5272!", 8) == 0) {
        printf("\nYou have won!\n");
    } else {
        printf("\n<<< Incorrect password: %s\n", secret);
    }
}

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

Solution

Due to the stack canary, the program is no longer vulnerable to buffer overflow attacks.



```
ggmeiner22@LAPTOP-MR9L4PB5:~/cyberthreats/final_proj$ ./canary
User >>> AAAAAAAAA
Password >>> BBBBBBBBCSE5272!
Stack corruption detected!
ggmeiner22@LAPTOP-MR9L4PB5:~/cyberthreats/final_proj$ E5272!
E5272!: command not found
```

3. Input Size Limiting and Null-Termination

fgets() reads a specified number of characters from the input stream and automatically appends a null terminator. This prevents overflow by ensuring that no more data than the buffer can hold is read.

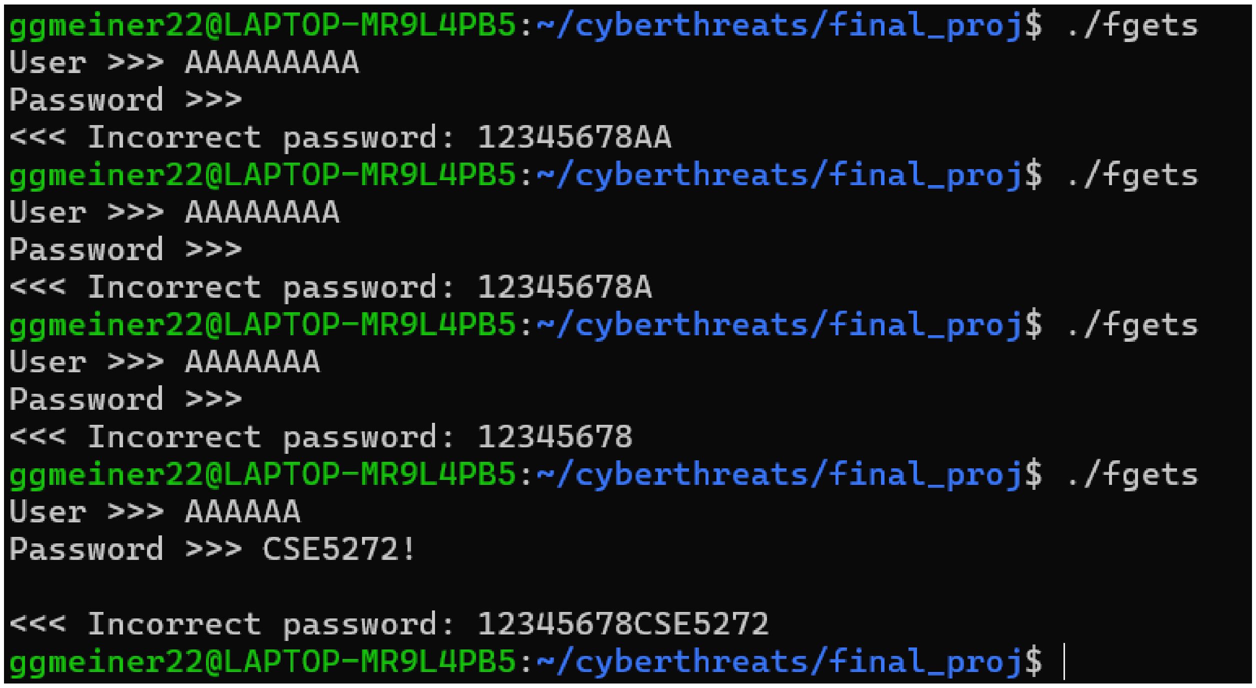
Source Code

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

void vuln() {
    char buf1[8];
    char secret[8] = "12345678";
    char buf2[8];

    printf("User >>> ");
    fflush(stdout);
    if (fgets(buf1, sizeof(buf1), stdin) == NULL) {
        perror("fgets error");
        exit(EXIT_FAILURE);
    }
    buf1[strcspn(buf1, "\n")] = '\0';

    printf("Password >>> ");
    fflush(stdout);
    if (fgets(buf2, sizeof(buf2), stdin) == NULL) {
```

```c
        perror("fgets error");
        exit(EXIT_FAILURE);
    }
    buf2[strcspn(buf2, "\n")] = '\0';

    if (strncmp(secret, "CSE5272!", 8) == 0) {
        printf("\nYou have won!\n");
    } else {
        printf("\n<<< Incorrect password: %s\n", secret);
    }
}

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

Solution

Due to fgets(), the input length is bounded. This ensures that buffer overflow attacks will not get through.



```
ggmeiner22@LAPTOP-MR9L4PB5:~/cyberthreats/final_proj$ ./fgets
User >>> AAAAAAAAA
Password >>>
<<< Incorrect password: 12345678AA
ggmeiner22@LAPTOP-MR9L4PB5:~/cyberthreats/final_proj$ ./fgets
User >>> AAAAAAAA
Password >>>
<<< Incorrect password: 12345678A
ggmeiner22@LAPTOP-MR9L4PB5:~/cyberthreats/final_proj$ ./fgets
User >>> AAAAAAA
Password >>>
<<< Incorrect password: 12345678
ggmeiner22@LAPTOP-MR9L4PB5:~/cyberthreats/final_proj$ ./fgets
User >>> AAAAAA
Password >>> CSE5272!

<<< Incorrect password: 12345678CSE5272
ggmeiner22@LAPTOP-MR9L4PB5:~/cyberthreats/final_proj$
```