

VULNERABLE C CODE BUFFER OVERFLOW EXPLOIT

**By Garrett Gmeiner, Maria
Linkins-Nielson, and Logan
Klaproth**

SOME BACKGROUND: CIA TRIAD

- **Confidentiality** - ensures sensitive information is only accessible to authorized individuals
- **Integrity** - guarantees that data is accurate and unaltered
- **Availability** - ensures system and data remain accessible and usable when needed

SOURCE CODE

```
#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();
}
```

COMPILATION & EXECUTION

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

EXECUTION

```
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 >>> AAAAAAAAAA
```

```
Password >>> BBBB BBBBCSE5272!
```

```
<<< Incorrect password: CS345678AAAAAAAAA
```

EXECUTION

```
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$ |
```

SOLUTION

```
ggmeiner22@LAPTOP-MR9L4PB5:~/cyberthreats/final_proj$ ./vuln
User >>> AAAAAAAAAA
Password >>> BBBB BBBBCSE5272!

You have won!
```

SECURING THE CODE

INPUT LENGTH VALIDATION

```
#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

CIA Principle:
Confidentiality &
Availability

```
ggmeiner22@LAPTOP-MR9L4PB5:~/cyberthreats/final_proj$ ./val
User >>> AAAAAAAAAA
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!
```

STACK CANARY

```
#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

CIA Principle:
Integrity

```
ggmeiner22@LAPTOP-MR9L4PB5:~/cyberthreats/final_proj$ ./canary
User >>> AAAAAAAAAA
Password >>> BBBB BBBBCSE5272!
Stack corruption detected!
ggmeiner22@LAPTOP-MR9L4PB5:~/cyberthreats/final_proj$ E5272!
E5272!: command not found
```

INPUT SIZE LIMITING AND NULL-TERMINATION

```
#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)
    {
        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

CIA Principle:
Confidentiality &
Integrity

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

FINAL REMARKS

- **Confidentiality:** Secure input handling avoids unintended memory reads.
- **Integrity:** Stack canaries detect tampering of sensitive variables.
- **Availability:** Safe input bounds and fail-safes keep the program stable.
- These changes not only eliminate the buffer overflow vulnerability but demonstrate how core cybersecurity principles can be enforced through secure coding practices.