Roll No: 2103163
Batch: C32
Name: Om Shete

Experiment No 10

Aim: Simulate Buffer overflow attack using Splint

Description:

Buffer overflow is a mistake that exist in some C implementations. These classes of bugs are dangerous as they write past the end of a buffer or array and hence corrupt the process stack They often change the return address of a process after a function call to a secret memory location where a malicious code is planted.

There are main two types

- Stack based attacks
- Heap based attacks

Heap-based attacks flood the memory space reserved for a program, but the difficulty involved with performing such an attack makes them rare. Stack-based buffer overflows are by far the most common.

Splint is a tool for statically checking C programs for security vulnerabilities and programming mistakes. Splint does many of the traditional lint checks including unused declarations, type inconsistencies, use before definition, unreachable code, ignored return values, execution paths with no return, likely infinite loops, and fall through cases. More powerful checks are made possible by additional information given in source code annotations. Annotations are stylized comments that document assumptions about functions, variables, parameters and types. In addition to the checks specifically enabled by annotations, many of the traditional lint checks are improved by exploiting this additional information. Splint is designed to be flexible and allow programmers to select appropriate points on the effort benefit curve for particular projects. As different checks are turned on and more information is given in code annotations the number of bugs that can be detected increases dramatically.

Problems detected by Splint include:

- Dereferencing a possibly null pointer
- Using possibly undefined storage or returning storage that is not properly defined
- Type mismatches, with greater precision and flexibility than provided by C compilers
- Violations of information hiding
- Memory management errors including uses of dangling references and memory leaks

Roll No: 2103163
Batch: C32
Name: Om Shete

• Dangerous aliasing

• Modifications and global variable uses that are inconsistent with specified interfaces

• Problematic control flow such as likely infinite loops, fall through cases or incomplete switches and suspicious statements

• Buffer overflow vulnerabilities

• Dangerous macro implementations or invocations

• Violations of customized naming conventions

Steps :

1. Installation

$ sudo apt-get install splint

2. Checking Vulnerability

$ splint program1.c

Implementation:

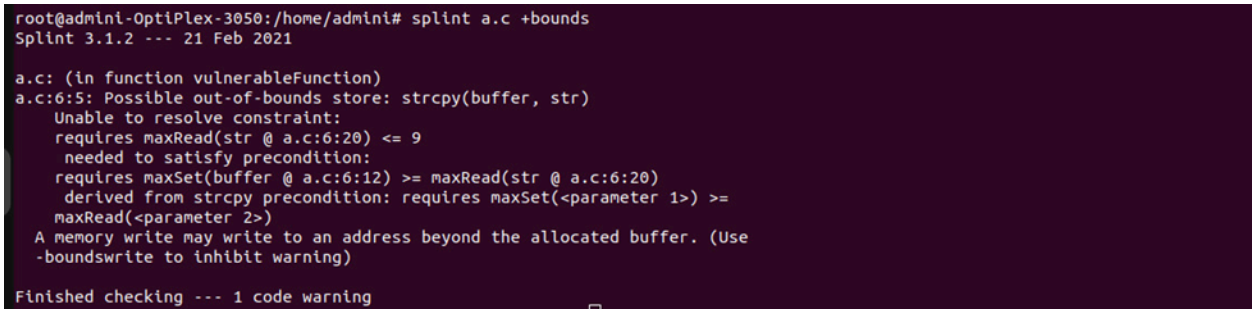
Program 1:

```
#include <stdio.h>

#include <string.h>


void vulnerableFunction(char *str) {

        char buffer[10];
```

Roll No: 2103163
Batch: C32
Name: Om Shete

```
strcpy(buffer, str); // Copies the input str to buffer without checking its length

}


int main() {


    char largeString[] = "This string is way too long for the buffer";

    vulnerableFunction(largeString);

    printf("Returned normally from vulnerableFunction\n");

    return 0;

}
```

Output:



Program 2:

```
#include <stdio.h>

#include <string.h>

char password[] = "password";

int get_password()
```

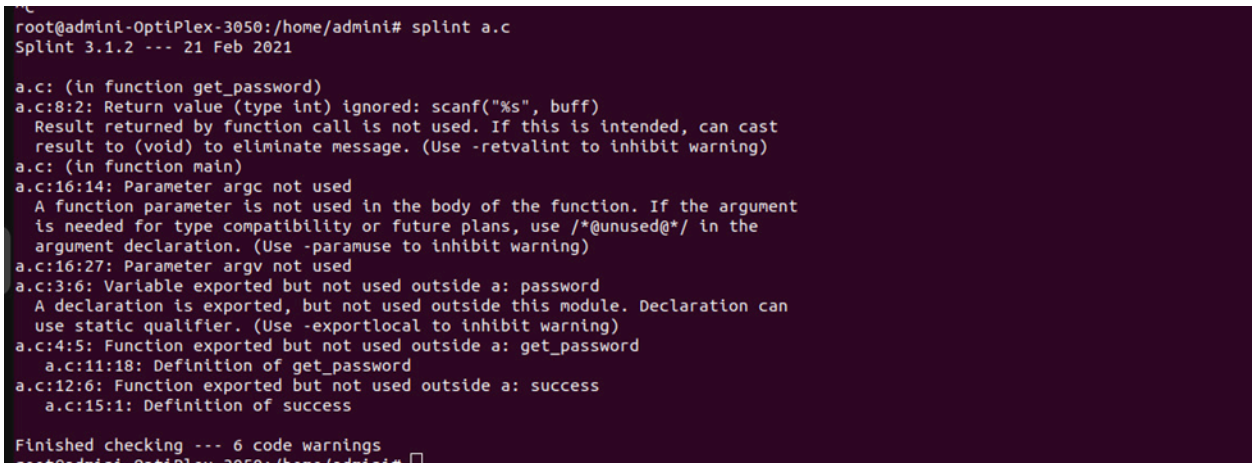
Roll No: 2103163
Batch: C32
Name: Om Shete

```
{
        int auth_ok = 0;
        char buff[16];
        printf("Enter password: ");
        scanf("%s", buff);

        if (strncmp(buff, password, sizeof(password)) == 0)
        auth_ok = 1;
        return auth_ok;
}
void success()
{
        printf("Success!\n");
}
int main(int argc, char **argv)
{
        int res = get_password();
        if (res == 0)
        {
        printf("Failure \n");
        return 0;
        }
```

Roll No: 2103163
Batch: C32
Name: Om Shete

```
        success();

        return 0;

}
```

Output:



Program 3:

```
#include<stdio.h>

main()

{

char buff[5];

printf("My stack looks

like:\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n");

buff[5]='abcdefghijklmnophsgkgfks';

printf("%c\n",buff[5]);

printf("My new stack looks
```

Roll No: 2103163
Batch: C32
Name: Om Shete

like:\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n");

}

Output: