Shyren More – c22 – 1902104

Experiment 10

Aim: Simulate buffer overflow attack using Splint, Cppcheck etc

Theory:
1. Cppcheck : Cppcheck is a tool for static C/C++ code analysis (CLI). Cppcheck is a command-
line tool that tries to detect bugs that your C/C++ compiler doesn't see. It is versatile, and can
check non-standard code including various compiler extensions, inline assembly code, etc. Its
internal preprocessor can handle includes, macros, and several pre-processor commands. While
Cppcheck is highly configurable, you can start using it just by giving it a path to the source code.
It includes checks for:
* pointers to out-of-scope auto variables;
* assignment of auto variables to an effective parameter of a function;
* out-of-bounds errors in arrays and STL;
* missing class constructors;
* variables not initialized by a constructor;
* use of memset, memcpy, etcetera on a class;
* non-virtual destructors for base classes;
* operator= not returning a constant reference to itself;
* use of deprecated functions (mktemp, gets, scanf);
* exceptions thrown in destructors;
memory leaks in class or function variables;
* C-style pointer cast in C++ code;
* redundant if;
* misuse of the strtol or sprintf functions;
* unsigned division or division by zero;
* unused functions and struct members;
* passing parameters by value;
* misuse of signed char variables;
* unusual pointer arithmetic (such as "abc" + 'd');
* dereferenced null pointers;
* incomplete statements;
* misuse of iterators when iterating through a container;
* dereferencing of erased iterators;
* use of invalidated vector iterators/pointers;

Step 1: Installation of cppcheck
$sudo apt-get install cppcheck

Step 2: Checking Vulnerability
$ cppcheck program.c

2. Splint : 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
• 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

Step : Checking Vulnerability
$ splint program.c

Code:

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

int main(void)
{
    char buff[15];
    int pass = 0;

    printf("\n Enter the password : \n");
    gets(buff);

    if (strcmp(buff, "thegeekstuff"))
    {
        printf("\n Wrong Password \n");
    }
```

```
    else
    {
        printf("\n Correct Password \n");
        pass = 1;
    }

    if (pass)
    {
        /* Now Give root or admin rights to user*/
        printf("\n Root privileges given to the user \n");
    }

    return 0;
}
```

OP:

splint buffer.c
Splint 3.1.2 --- 03 May 2009

buffer.c: (in function main)
buffer.c:10:5: Use of gets leads to a buffer overflow vulnerability. Use fgets
instead: gets
Use of function that may lead to buffer overflow. (Use -bufferoverflowhigh to
inhibit warning)
buffer.c:10:5: Return value (type char *) ignored: gets(buff)
Result returned by function call is not used. If this is intended, can cast
result to (void) to eliminate message. (Use -retvalother to inhibit warning)
buffer.c:12:8: Test expression for if not boolean, type int:
strcmp(buff, "thegeekstuff")
Test expression type is not boolean or int. (Use -predboolint to inhibit
warning)
buffer.c:22:8: Test expression for if not boolean, type int: pass

Finished checking --- 4 code warnings