# EXPERIMENT NO 09

CLASS: TE CMPN A PID: 182027

NAME: REBECCA DIAS ROLL NO. : 19

**Aim:** Simulate buffer overflow attack using Ollydbg, Splint, Cppcheck etc

**Theory:**

**a) What is Buffer Overflow Attack**

Buffer overflows are a common vulnerability on all platforms, but are by far the most commonly exploited bug on the Linux/Unix Operating systems. Buffer over flow occurs when you try to insert data into consecutive memory addresses more than its capacity of storage. Commonly buffer overflows are exploited to change the flow in a programs execution, so that it points to a different memory address or overwrites crucial memory segments. memory is organized in 4byte (32 bit) segments, consisting of a hex memory address, and will need to be converted to little endian byte memory addresses which follows last in first out terminology ( LIFO) data comes first in stack will go out from stack in last. The stack and EIP is the most important part of the buffer over flow vulnerabilities.

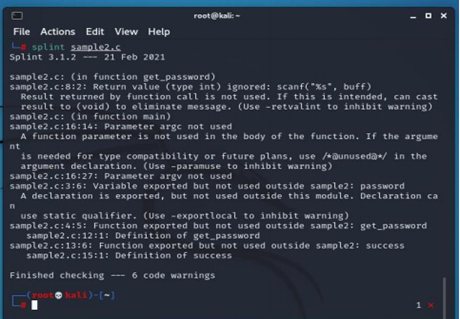
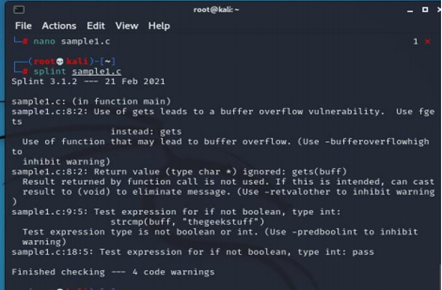
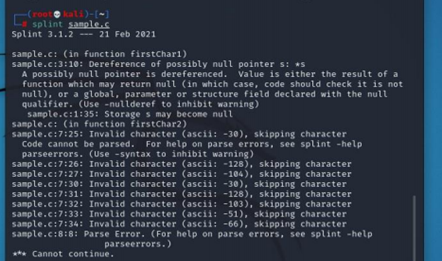
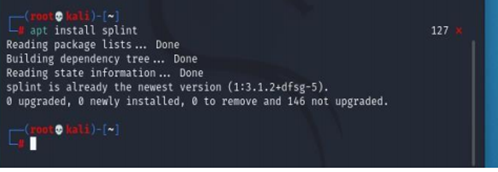
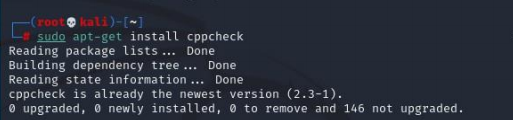
**b) Write in short about Cppcheck**

* Cppcheck is a [static code analysis](https://en.wikipedia.org/wiki/Static_code_analysis) tool for the [C](https://en.wikipedia.org/wiki/C_(programming_language)) and [C++](https://en.wikipedia.org/wiki/C%2B%2B) [programming languages](https://en.wikipedia.org/wiki/Programming_language). It is a versatile tool that can check non-standard code.The creator and lead developer is Daniel Marjamäki.
* Cppcheck is [free software](https://en.wikipedia.org/wiki/Free_software) under the [GNU General Public License](https://en.wikipedia.org/wiki/GNU_General_Public_License).
* Cppcheck supports a wide variety of static checks that may not be covered by the compiler itself. These checks are static analysis checks that can be performed at a source code level. The program is directed towards static analysis checks that are rigorous, rather than [heuristic](https://en.wikipedia.org/wiki/Heuristic) in nature.

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;

**c) Install and Cppcheck and take screen shot of installation**

****

**d) Perform Buffer Overflow Attack and Take screen shot**

**Vulnerable C Code**

int Function(char \*str){

char buffer[10];

strcpy(buffer,str);

return 0;

}

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

{

char code[]="AAAA";

printf("You are in main fucntion now\n");

Function(code);

checking();

printf("Quitting vuln.exe\n");

getch();

return 0;

}

int checking(){

printf("\*\*\*\*\*\*\* You have done it! \*\*\*\*\*\*\*\n");

printf("\*\*\*\*\*\*\* This is checking() executing \*\*\*\*\*\*\*\n");

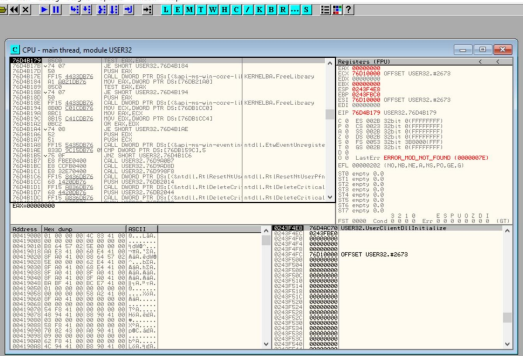
}

This is a simple C program which has one array that contains some string and later in the program this string is pass to vulnerable function which is Function(char \*str). The Function has a buffer of 10 bytes.1 char is equal to1 byte and in program we have char buffer[10] ,so size is equal to 10 bytes. Moreover, the Function copies the content of variable code into buffer without checking the size of variable code, and this is the

vulnerability.

strcpy(buffer,str); //the vulnerable command

We will keep changing the value of array code ( cha code[]=”AAAa”; ) and testing the program. So first let us check the program by simply taking the value of array code to 3 A’s char code[]=”AAA”; Change the value of array code to 3 A’s and Compile the program.



Now we will confirm the vulnerability of this program by changing the value of char

code to about 30 A’s.

char code[]=”AAAAAAAAAAAAAAAAAAAAAAAAAAAAA”;

As you know our buffer is capable of to fill maximum of 10 characters and if we fill it

with 30 A’sthen it will cause our program to crash as when 30 A’s will pass to Function

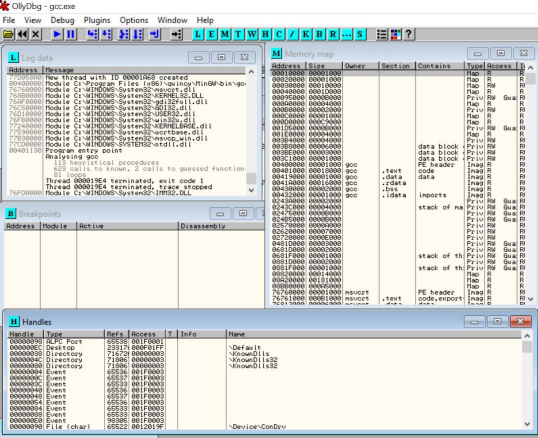
,this function will try to fill the buffer with it and our program will crash because our

buffer has the capacity to fill with 10 characters and due to this value of EIP will be

change. Let us check now. Change the value of char code to 30 A’s in the original

program.

char code[]=”AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA”;



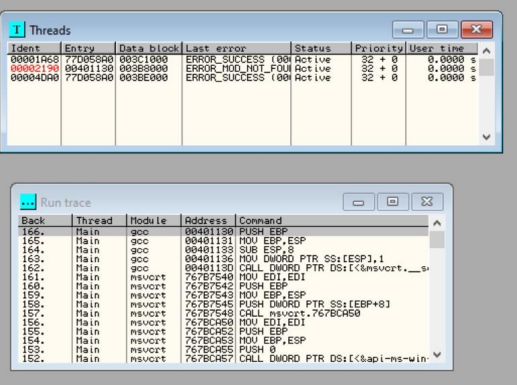
This happens because when the buffer is filling with more size than it is capable of, now remember that in stack after it has frame pointer and after that it has EIP pointer. So, 30 A’s first fill the stack then they fill the frame pointer and then finally the EIP value, which has the return address so when return address is overwrite by us the program crash.

**CHECK WHAT HAPPENS ON OLLYDBG**

THE EIP has value 41414141 which is ASCII value of A and not any physical address. What is happening due to over flow of buffer our EIP is overwritten by what we provide via char code. The value of EIP is overwritten by ASCII value of A which is 42, and this can be seen in OllyDbg showing EIP content to 41414141 (ASCII A =41). Now when we use 30 A’s the program will never execute after Function is call

and check function will never be executed as program crash in Function. As we successfully change the content of EIP with ASCII value of A

This is the basic idea of Buffer over flow, now days people make exploits on buffer over flow and run their own shell codes via buffer over flow. This can simply be done by storing your shell code somewhere middle in buffer and then as you know EBP pointer has the memory address of the top of the stack so using it you can get the stack top position address and your shell code is already store in buffer so you can change the value of EIP to the value of EBP.



**Conclusion:**

In this experiment we learnt to perform a buffer overflow attack and identify it using the Ollydbg software tool. We also learnt to analyze C language programs using Cppcheck and identify the errors and warnings that make the code inefficient or unable to compile.