**Component 4**

**Buffer overflow exploitation**

What is a buffer?

A buffer is a small memory allocation used when a program or a process is executed.

Buffer overflow: it is a type of software vulnerability that occurs when a program writes more data in an array or buffer then it is allocated for. This can lead to data being overwritten, unpredictable behaviour, crashes, and potentially, security vulnerabilities that attackers could exploit. If attackers overwrite important pieces of the code such as return addresses or function pointers, they can gain control over the program.

A black line with numbers and a mathematical equation

Description automatically generated with medium confidence

This row represents the memory, and you want to store a sequence of numbers in it. The containers are labelled from 1 to 5.

Here's what happens:

1. You successfully store the numbers 1, 2, 3, 4, and 5 in the first five containers.
2. However, when you try to add a sixth number (let's say 5), there's no space left. It's like trying to put a sixth item into a row of only five containers.

This attempt to add more data than the containers can hold is akin to a buffer overflow. The "overflowing" number (in this case, 5) doesn't fit in the allocated space, potentially causing issues or unintended behaviour, much like how a buffer overflow in computer programs can lead to unpredictable consequences.

A white board with black numbers and a rectangular box

Description automatically generated

No matter how much numbers are added they are all going to be overflowing.

a buffer overflow becomes more dangerous when the attacker has knowledge of the code. When an attacker understands the inner workings of the target software, they can craft more sophisticated and targeted attacks, increasing the likelihood of successful exploitation.

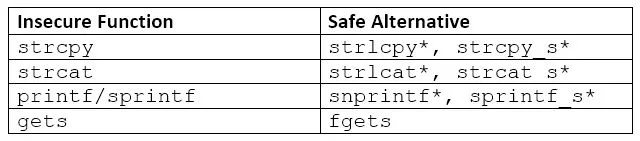
**How to prevent buffer overflows**

Developers can protect their software against buffer overflow vulnerabilities through a combination of secure coding practices and by using programming languages that provide built-in protection mechanisms. C for example is a bad choice as it allows their vulnerabilities through direct access to memory. Back in the day buffer overflow was much easier to implement. However, modern operating systems (OS) and API designs incorporate several features and security measures to make buffer overflows and other common security vulnerabilities more difficult to exploit.

Three common countermeasures are:

1. Address space layout randomization (ASLR): By randomly moving around the addresses of various memory sections, ASLR introduces an additional layer of defence against buffer overflow attacks. In a buffer overflow scenario, an attacker usually relies on knowing the precise location of executable code to inject and execute malicious instructions.
2. Data execution prevention: flags certain areas of memory as non-executable or executable, which stops an attack from running code in a non-executable region. DEP effectively creates a boundary between data and code, disrupting the common strategy employed by buffer overflow attacks.
3. Replace standard unsafe string functions (e.g., strcpy, strcat) with safer alternatives such as strncpy, strncat, or safer string manipulation functions provided by libraries like the C Standard Library. These functions allow you to specify the maximum number of characters to copy or concatenate.

C functions and their alternatives:



\*Asterisks denote functions that are not part of C Standard Libraries.

**My cp command exploitation**

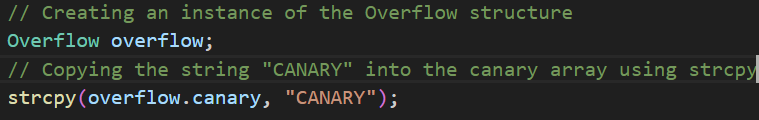
I implemented my own cp command using C language and here is how I managed to exploit it.

**A screen shot of a computer code

Description automatically generated**

I used the typedef keyword to create a structure called Overflow.

‘source’ is an array of 5 character to store the source file name, ‘canary’ is an array of 8 characters used as a security measure to detect if the ‘source’ array is being overflowed.



As written in the comments, I created an instance of the Overflow structure and copied the “CANARY” string into the canary array.

A screen shot of a computer code

Description automatically generated

This line uses the strcpy function to copy the second command line argument into the source array of the overflow structure. If the length of the argument is greater than 5 characters, it will result in a buffer overflow.

The following if statement checks if the value of the canary array is different from the string “CANARY”. If the comparison is not equal (indicating a modification to the canary array), it prints a message indicating an overflow and then prints the current content of the canary buffer. So if for example the source file name entered into the argv[1] had more characters than 5 (the storage size is 5) the extra characters will be overflowing into the canary array and will be printed.

A screenshot of a computer

Description automatically generated

As shown in the image above I ran my program on linux and it worked, no buffer overflow detected as the source file name is less than 5 characters.

A screenshot of a computer

Description automatically generated

I ran it again and as shown in the image above bufferoverflow was detected. The source file name had 7 characters while the array size is 5, the extra characters were t and s.

A screenshot of a computer

Description automatically generated

The exploit still works with the use of flags as show in the image above.