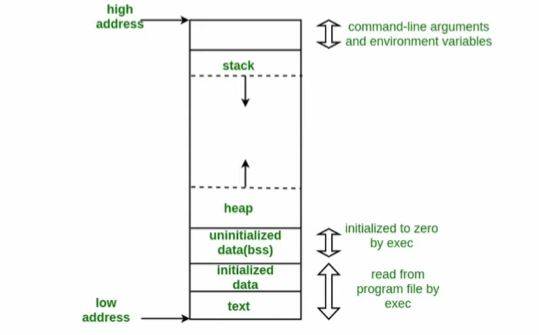
Buffer overflow

24 May 2021

22:46

**What is buffer**

* + A buffer is a sequential section of memory allocated to contain anything from a character string to an array of integers.
  + These buffer typically live in RAM. Computer frequently use buffer to help improve perfomaance; most modern hard drives take advantage of buffering to effficiently access data, and many online services also use buffers.
  + Buffer are designed to contain specific amount of data. Unless the program utilizing the buffer has build-in-instruction to discard data when too mach is sent to the buffer, the program will overwrite data in memory adjacent to the buffer.



**What is Buffer Overflow?**

* + A buffer overflow, or buffer overrun, occurs when more data is put into a fixed-length buffer than the buffer can handle.
  + The extra information, which has to go somewhere, can overflow into adjacent memory space, corrupting or overwriting the data held in that space.
  + This overflow usually results in a system crash, but it also create the opportunity for an attacker to run arbitrary code or manipulate the coding errors to prompt malicious actions.
  + In 2014 a threat known as 'heardbleed' exposed hundres of millions of users to attack because of a hundred of a buffer overflow vulnerability in SSL software.

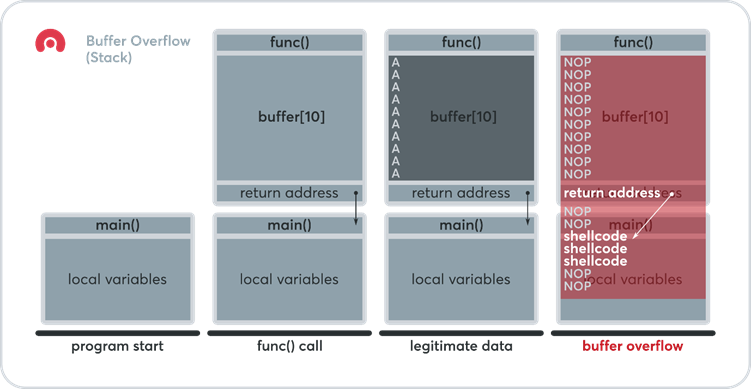
**(example)**

* + In a simple program, you may want the user to enter an email address. Therefore, you create a string variable.
  + You allocate 64 bytes to the variable because you do not expect an email sting to be longer than 64 characters. You do not check if the length of the entered string exceeds the size of the buffer.
  + As a results, the user enter 100 characters and the remaining 36 characters stored in memory allocated to another variables.
  + In most cases, this leads to a simple memory segmentation fault but it may have more serious consequences, program behavior of the program may change as well.

**(if attacker know there is a buffer overflow vulnerability so what he will do and how exploit buffer)**

* + The malicious content that the attacker sends to a faulty program is usually composed of three parts:
    - A chain of bytes that represent the **NOP** instruction.
    - A new return address that point to the NOP Bytes.
    - Arbitrary code (usually a **shellcode**) locate somewhere in the middle of the chain of NOP byte.

( NOP= no operation instruction)



**Types of buffer overflow**

* + **Stack overflow attack:** this is the most common type of buffer over flow attack and involves overflowing a buffer on the call stack.

* + **Heap Overflow attack:** this type of attack targets data in the open memory pool known as the heap.

* + **Integer overflow attack:** In a integer overflow, an arithmetic operation results in a integer (whole number) that is too large for the integer types meant to store it; this can result is a buffer overflow.

* + **Unicode overflow:** A unicode overflow creates a buffer overflow by inserting unicode characters into an input that expect ASCII characters**.**

--->**Perl, javascript** is generally not susceptible to buffer overflow.

--->Buffer overflow is a program written in **C, C++, Fortran or assembly** could allow the attacker to fully compromise the targeted system.

---> the safest basic method in C is to avoid the following seven unsafe function that can lead to a buffer overflow vulnerability: **scanf, gets, printf, sprintf, strcat, strcpy, and gets**. Use **fget, sprintf\_s, strcpy\_s and strcat\_s,** which perform bounds checking.

**Key Concepts**

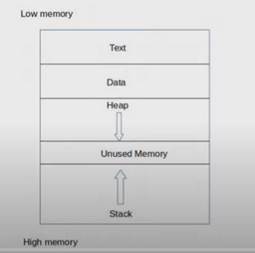
* + This error occurs when there is more data is a buffer then it can handle, causing data to overflow into adjacent storage.

* + This vulnerability can cause a system crash, code execution, or worse, create an entry point for a cyberattack.

* + C and C++ are more susceptible to buffer overflow.

**Understanding the call stack**

* + A **call stack** is a stack data structure that stores information about the active subroutines(functions) of a computer system.
  + The main reason for having one is to keep track of the point to which each active subroutine should return control when it finishes executing.
  + Each active subroutine reserve a part of the call stack called **stack frame**, which represent a single procedure call.
    - **PUSH**: ADDs as item to the top pf the stack
    - **POP**: Removes an item from the top of the stack.

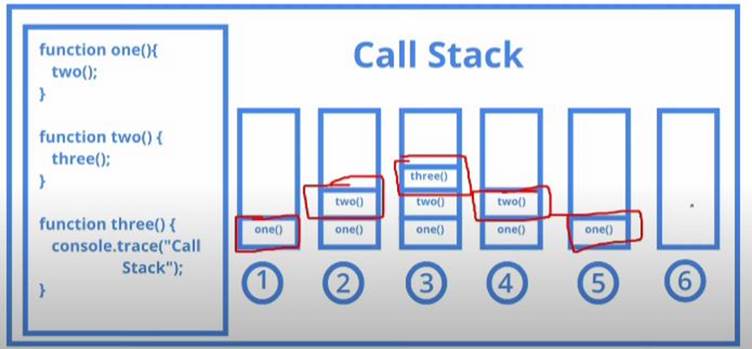


Stack grow high to low

**Example**



Example 2





Cpu resisters used to store temporary data.

