Buffer Overflow vulnerability and defenses

TCS 591 : Unit 2

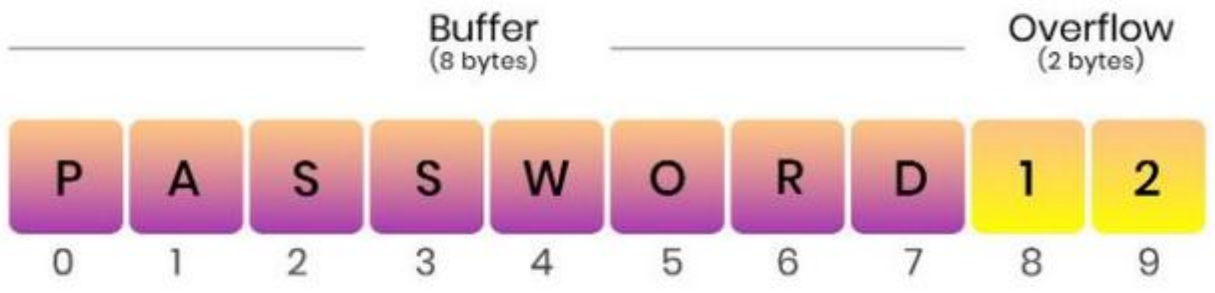
What is Buffer Overflow ?

- Buffer overflow attacks generally occur when you try to write to a memory location you do not own.
- The main reason behind them is poorly implemented bound checking on user input.
- Due to this, user-supplied input is written into the wrong memory space.

Real Life Situation

- Consider a bucket with a capacity of 1 liter. You want to fill that bucket with water and keep the floor dry.
- You start filling the bucket with water, and after a while, that bucket will be full.
- If you do not stop, water will spill on the floor, a place where you did not intend to fill water.

Buffer Overflow Attack



- The overwritten space does not belong to the user, resulting in a segmentation fault, and the program will crash.
- When this happens to production-level software, which is supposed to run seamlessly without any maintenance for an extended period, such as a **web server**, it will immediately crash.
- This crash would cause a **DOS, i.e., Denial of Service** attack, resulting in the server becoming unavailable to everyone.

Prevention Methods

1. The first step to protect your software against buffer overflow attacks is to write stable and robust code that is secure.
2. The next step should be to validate all incoming inputs from the user. Various checks, such as **bound checks**, must be done before any other input processing.
3. A developer should secure the software under the assumption that all user input is malicious.
4. If the user input is larger than the stack size and the developer has not implemented any bound checks, it can result in a stack overflow.

Prevention Methods

5. The first key to implementing proper input validation is to find which type of input can cause trouble.
6. Developers, as well as attackers, use guided automatic fuzzers to automate various inputs into the application to test which class of input can cause the application to **behave abnormally.**
7. A **Fuzzer** generally tries a combination of various special characters, alphabets, numerals, pure binary, metadata to check if an application is vulnerable.

Prevention Methods

8. **Handling Strings Safely** : Vulnerable implementations of strings account for the most considerable portion of buffer overflow attacks. Various unsafe functions do not use bound checking while working with strings. This results in the exploitation of the buffer overflow.
9. **Printf, sprintf, strcat, strcpy, and gets** are examples of functions responsible for these attacks. To altogether avoid overrun attacks on their production application, developers should avoid using these unsafe functions and use their safer versions, i.e., fgets, sprintf_s, strcpy_s, and strcat_s.
10. Developers should ensure that the destination buffer's size is determined to ensure that **function stops writing after reaching its end**.

Prevention Methods

11.Protections Provided by Operating Systems:

ASLR – Address space layout randomization

DEP – Data Execution Prevention

These are the functionalities operating systems provide to protect applications against memory corruption exploits such as Buffer Overflows.

Prevention Methods

12. Source Code Audit : A source code audit is a go-to technique to ultimately ensure that there are no unsafe functions or vulnerabilities present that can lead to a buffer overflow attack.

In addition to the audit of their code, developers should also audit libraries and frameworks used by them.

There are various tools which can help an audit, which are:

- Application Defense snapshot
- Flaw finder
- Rough Auditing tool for security
- Fortify Secure code Analysis

Prevention Methods

13. Using /GS in Visual C++: In Visual C++, you can use /GS as a compile-time flag. This flag enables the compiler to add stack canaries between various variables initialized on the stack. This ensures that even if your program is vulnerable to buffer overrun attacks, it won't be exploitable.
14. GNU C compiler provides similar stack canary functionality under the name of stack guard, which just like /GS in visual C++ can be bypassed.