# Technical Software Failures or Errors

## Buffer overrun (or buffer overflow)

**Definition:** An application error that occurs when more data is sent to a program buffer than it is designed to handle.

**Example:**

* Imagine you’re using an old version of Internet Explorer (let’s say version 4.0).
* You decide to type a special type of URL that starts with res://.
* But here’s the catch: this URL is longer than 256 characters.
* When you hit Enter, the browser crashes because it can’t handle such a lengthy URL.
* Now, the interesting part: anything after the 256th character in that URL can be executed as code on your computer!
* Hackers can take advantage of this by adding malicious code (like a virus or harmful program) to the end of the URL.
* If successful, they gain control over your system, which can lead to serious security issues.

## Catching Exceptions

**Definition:** exceptions are considered “expected but irregular situations at runtime”.

**Example:**

* Dividing by zero: If you try to divide a number by zero, an exception occurs.
* File not found: When a program expects a file but cannot find it, an exception is raised.
* Network connection issues: If a network operation fails, an exception can be caught.

## Command Injection

**Definition:** An application error that occurs when user input is passed directly to a compiler.

**Example:** The simplest example demonstrated using the Windows command shell:

@echo off

set /p myVar=”Enter the string>”

set someVar=%myVar%

echo

* The problem lies in the line: set /p myVar=”Enter the string>”:
* This line prompts the user to input a string and assigns it to the variable myVar.
* The problem arises when an attacker provides a malicious input.
* For example, if the user enters: Hello&del\*.\*
* The & character acts as a command chaining operator.
* It allows the attacker to append additional commands after the user’s input.
* In this case, the del\*.\* command is added to the end of the string.

## Cross-Site Scripting (XSS)

Definition: A Web application fault that occurs when an application running on a Web server inserts commands into a user’s browser session and causes information to be sent to a hostile server.

**Example:**

* Imagine a legitimate(safe) website that allows users to post comments.
* An attacker posts a seemingly harmless comment containing a malicious script (JavaScript).
* When other users view this comment, their browsers execute the injected script.
* The script can:
  + Steal cookies (including session tokens) from the victim’s browser.
  + Redirect the user to a phishing site.
  + Modify the page content to display fake messages.
  + Perform actions on behalf of the user (if they are logged in).
* The attacker encodes the malicious link to make it look less suspicious.
* After collecting data (e.g., session tokens), the hostile application appears to respond normally to the intended server.

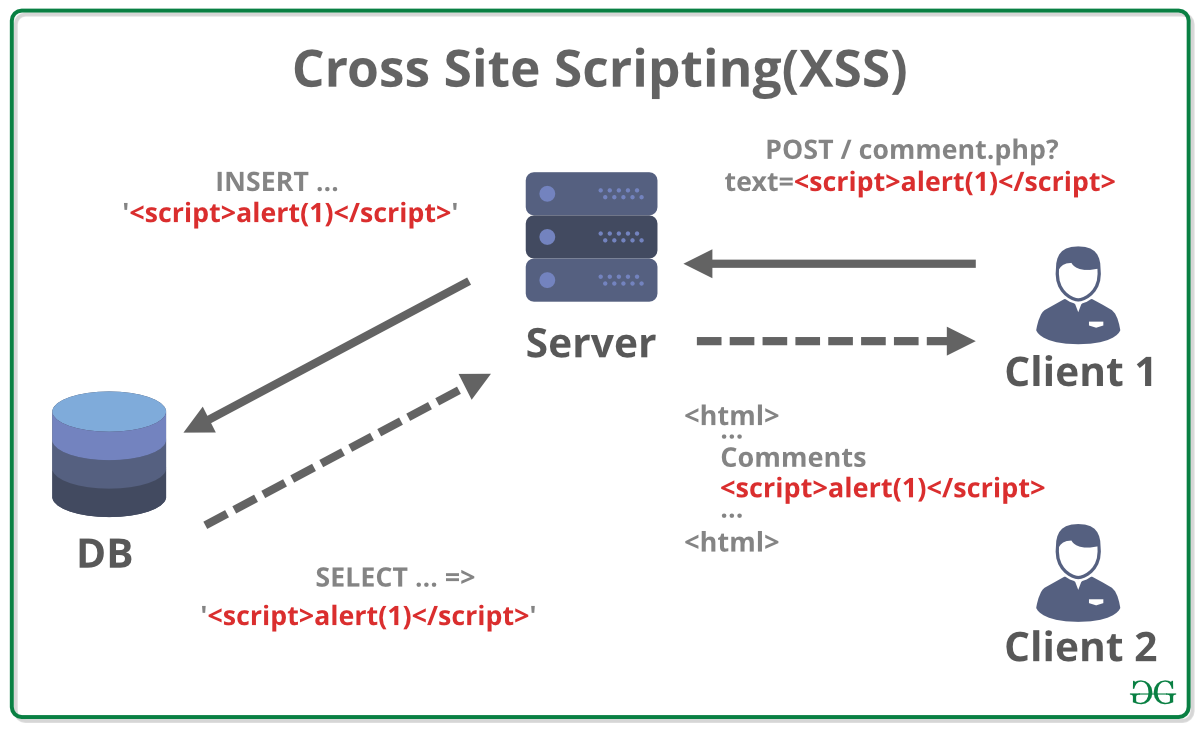


Figure 1: XSS

## Failure to Handle Errors

Failure to handle errors can cause a variety of unexpected system behaviors. Programmers are expected to anticipate(predict) problems and prepare their application code to handle them.

## Failure to Protect Network Traffic

### Wireless Network

Without encryption like WPA (Wi-Fi Protected Access), attackers can intercept and view the data being transmitted between a client (like your laptop or smartphone) and the wireless access point (the router providing Wi-Fi).

### Wired Network

In networks using hubs instead of switches, any user can install a packet sniffer (a tool for intercepting and logging traffic passing over a network) and collect communications to and from other users on that network.

Both wireless and wired networks are vulnerable to interception, but implementing encryption and taking security measures can help mitigate these risks.

## Failure to Store and Protect Data Securely

* Proper implementation of access controls is crucial for data security. Failure to implement strong access controls can leave the data vulnerable to unauthorized access or manipulation.
* Programmers sometimes integrate secret information directly into programs. This could include "hard coding" passwords, encryption keys, or other sensitive information. Storing such sensitive information within the code can pose a significant risk of disclosure if the code is accessed by unauthorized individuals.

## Failure to Use Cryptographically Strong Random Numbers

* Cryptosystems, which are used for secure communication and data protection, often rely on random number generators.
* Not all systems require the same level of randomness. For example, a decision support system using random or pseudorandom numbers for Monte Carlo method forecasting may not need as rigorous randomness as a system implementing cryptographic procedures.
* Pseudorandom number generators (PRNGs) are algorithms that use a seed value and other system components, such as the computer clock, to produce a sequence of numbers that appear random. However, these numbers are not truly random; they are deterministic and predictable given the initial seed value.
* Because PRNGs are deterministic, someone who understands the algorithm and has knowledge of the seed value and system components used can predict the sequence of "random" numbers generated by the PRNG.

In summary, while pseudorandom number generators are sufficient for certain applications, they are not suitable for cryptographic purposes where strong randomness is essential. Cryptographically strong random number generators are required to ensure the security and confidentiality of sensitive data in cryptographic systems.

## Format String Problems

Some programmers may use data from untrusted sources as a format string.

An attacker can take advantage of this by inserting special characters like %x, %d, %p, etc., into their input. If the program treats this input as a format string (like when using the printf function in C), the attacker might be able to see secret information or even change specific parts of the program's memory, like the stack.

Example:

char user\_input[100];

scanf("%s", user\_input);

printf(user\_input); // <-- Attacker provides "%x %x %x %x" as input

* If the attacker's input is %x %x %x %x, it might reveal memory addresses from the stack, potentially exposing sensitive information or causing a crash.

## Improper Use of SSL (Secure Sockets Layer)

### Purpose of SSL:

SSL is used to encrypt data transmitted between a client (such as a web browser) and a server, ensuring that sensitive information like credit card numbers and personal data remains secure during transmission.

### Assumption of Security:

Many programmers assume that using SSL guarantees the security of data transmission. However, this assumption can lead to mishandling of SSL technology.

### Importance of Certificate Validation:

SSL and its successor TLS (Transport Layer Security) require certificate validation to ensure true security.

Certificate validation involves verifying the authenticity of the server's certificate and ensuring that it is issued by a trusted Certificate Authority (CA). This process is crucial for establishing a secure connection.

### Risks of Improper Validation:

Failure to properly validate the server's certificate, such as not using HTTPS to validate the certificate authority and the certificate itself, can compromise the security of SSL traffic.

Additionally, failing to validate the information against a Certificate Revocation List (CRL), which contains revoked certificates, can also pose a security risk.

## Information Leakage

One of the most common methods of obtaining inside and classified information is directly or indirectly from one person, usually an employee.

A famous World War II military poster warned that “loose lips sink ships,” emphasizing the risk to naval deployments from enemy attack if sailors, marines, or their families disclosed the movements of U.S. vessels.

By warning employees against disclosing information, organizations can protect the secrecy of their operation.

## Integer Bugs (Overflows/Underflows)

Computers use fixed-length binary representations for numbers, which means they can only handle numbers within a certain range determined by the number of bits used to represent them.

Programmers need to anticipate the size of numbers they'll be working with to avoid integer bugs.

An overflow happens when a calculation produces a result larger than what can be represented with the available number of bits. For instance, adding 1 to the maximum value of a 16-bit signed integer (32,767) results in an overflow, giving an unexpected negative value (-32,768).

An underflow occurs when a calculation produces a result smaller than what can be represented. For example, subtracting 5 from the minimum value of a 16-bit signed integer (-32,767) results in an underflow, giving an unexpected positive value (32,764).

Integer bugs are categorized into four main classes: overflows, underflows, truncations, and signedness errors.

These bugs can be exploited by attackers to corrupt other areas of memory, potentially gaining control of the application.

In summary, integer bugs occur when programmers don't properly handle the limitations of computer arithmetic, leading to unexpected results in calculations. These bugs can have serious consequences and may be exploited by attackers to compromise the security or stability of the system.