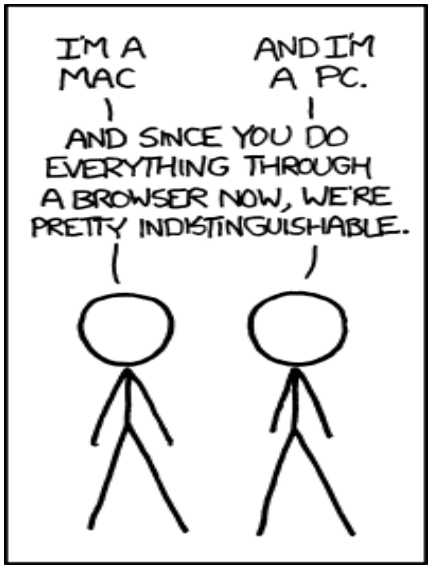
MIS6070

Web Based Information Systems



Lesson 8

Security Characteristics for Computer Systems

- Security for the computer systems is usually defined as a combination of required characteristics.
  - This also varies depending on the application, primary requirements include
- **Confidentiality:** refers to the sensitivity of data.
  - Confidential data needs to be protected from unauthorized access, use, or disclosure.
- **Integrity:** refers to the reliability of data. To have integrity, data needs to be protected from unauthorized modification.
- **Availability:** refers to the accessibility of data.
  - To be available, data needs to be protected from disruption of service.
- **Authentication:** End parties to a transaction can be reliably identified
- **Authorization:** Each user is given only appropriate privileges
- **Accountability:** Each user is bound to his/her actions (cannot repudiate/deny actions

Web Application Security Fundamentals

- When you hear talk about Web application security, there is a tendency to immediately think about attackers **defacing Web sites**, **stealing credit card numbers**, and bombarding Web sites with **denial of service attacks**.
- You might also think about **viruses**, **Trojan horses**, and **worms**.
- These are the types of problems that **receive the most press** because they represent some of **the most significant threats faced by today's Web applications**.
- **These are only some of the problems.**

Web Application Security

- Other significant problems are frequently overlooked.
- **Internal threats** posed by rogue administrators, disgruntled employees, and the casual user who mistakenly stumbles across sensitive data pose significant risk.
- The biggest problem of all may be **ignorance**.
- The solution to Web application **security is more than technology**.
  - It is an ongoing process involving people and practices.

Web Application Security

- **What Do We Mean By Security?**
- Security is fundamentally about protecting assets.
  - Assets may be tangible items, such as a Web page or your customer database or they may be less tangible, such as your company's reputation.
- Security is a path, not a destination.
  - As you analyze your infrastructure and applications, you identify potential threats and understand that each threat presents a degree of risk.
- Security is about risk management and implementing effective countermeasures.

Web Application Security

- **Threats, Vulnerabilities, and Attacks Defined**
- A threat is any potential occurrence, malicious or otherwise, that could harm an asset.
  - In other words, a threat is any bad thing that can happen to your assets.
- A vulnerability is a weakness that makes a threat possible.
  - This may be because of poor design, configuration mistakes, or inappropriate and insecure coding techniques.
  - Weak input validation is an example of an application layer vulnerability, which can result in input attacks.

Web Application Security

- **Threats, Vulnerabilities, and Attacks Defined**
- An attack is an action that exploits a vulnerability or enacts a threat.
  - Examples of attacks include sending malicious input to an application or flooding a network in an attempt to deny service.
- **To summarize**,
  - a threat is a potential event that can adversely affect an asset, whereas a successful attack exploits vulnerabilities in your system.

Web Application Security

- **How Do You Build a Secure Web Application?**
  - It is not possible to design and build a secure Web application until you know your threats.
  - An increasingly important discipline and one that is recommended to form part of your application's design phase is threat modeling.
  - The purpose of threat modeling is to analyze your application's architecture and design and identify potentially vulnerable areas that may allow a user, perhaps mistakenly, or an attacker with malicious intent, to compromise your system's security.

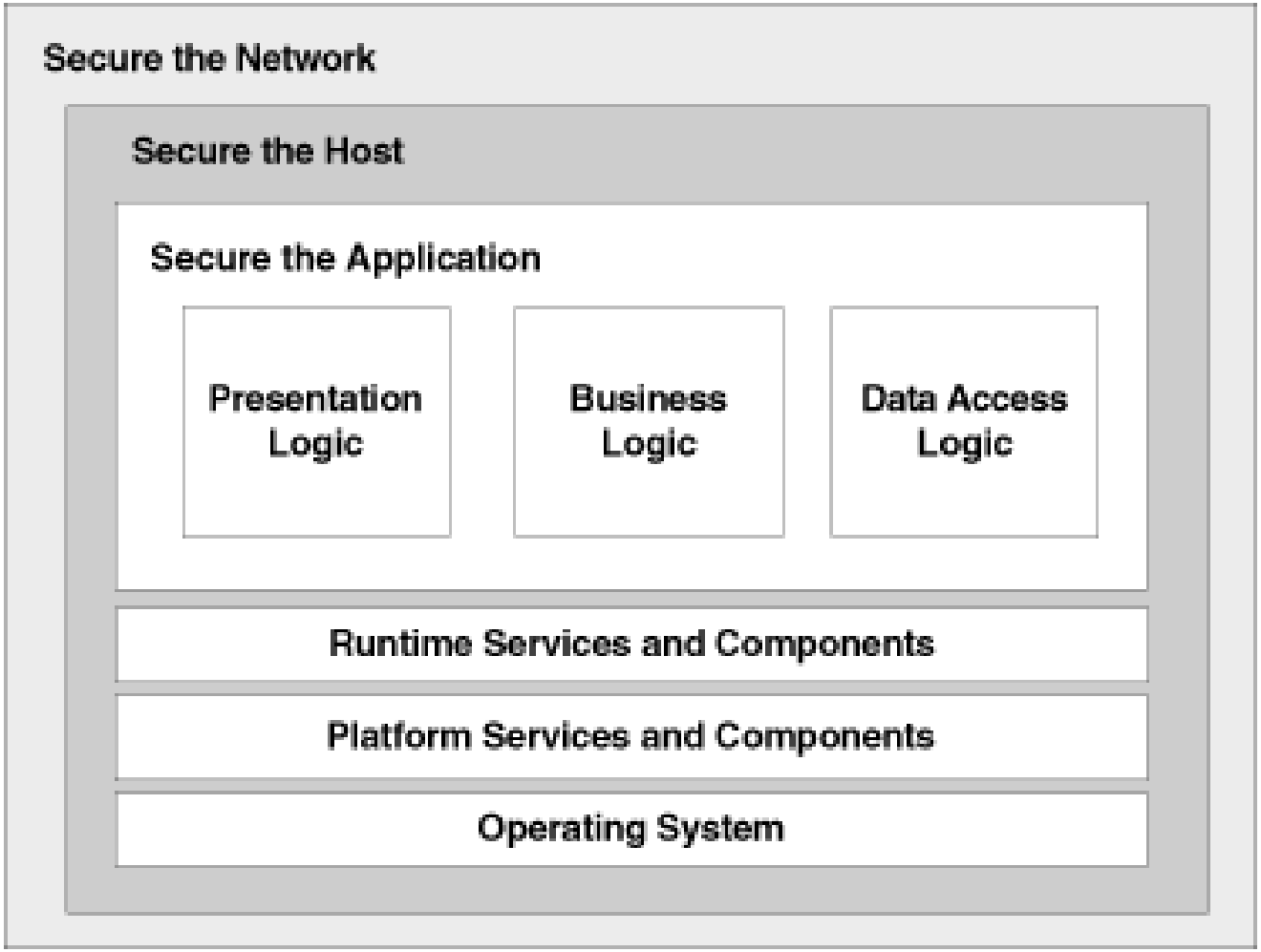
Web Application Security

- **How Do You Build a Secure Web Application?**
  - After you know your threats, design with security in mind by applying timeworn and proven security principles.
  - As developers, you must follow secure coding techniques to develop secure, robust, and hack-resilient solutions.
  - The design and development of application layer software must be supported by a secure network, host, and application configuration on the servers where the application software is to be deployed.

Web Application Security

- **Secure Your Network, Host, and Application**
  - *"A vulnerability in a network will allow a malicious user to exploit a host or an application.*
    A vulnerability in a host will allow a malicious user to exploit a network or an application. A vulnerability in an application will allow a malicious user to exploit a network or a host."
- To build secure Web applications, a holistic approach to application security is required and security must be applied at all three layers.

Web Application Security



A holistic approach to security

Web Application Security

Anatomy of an Attack

- The basic approach used by attackers to target your Web application
  - **Survey and assess**
  - **Exploit and penetrate**
  - **Escalate privileges**
  - **Maintain access**

Security and Encryption

- Developing a secure web application requires a combination of **strategies that are applied throughout the life cycle of the application**.
- **During design,** developers should assess potential threats and built in appropriate counter measures.
- Comprehensive testing should be employed to ensure that these threats have indeed been mitigated.
- After the application is deployed, constant monitoring and system upgrades are necessary to counter new threats and maintain a high level security.

Web Application Security

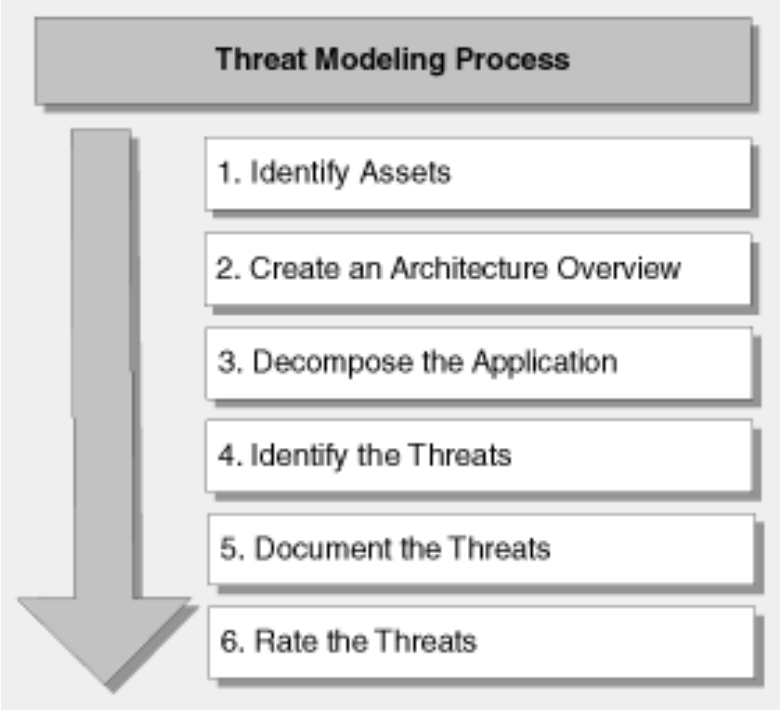
- The threat modeling process can be performed using a six-stage process
  - **Identify assets**. Identify the valuable assets that your systems must protect.
  - **Create an architecture overview**. Use simple diagrams and tables to document the architecture of your application, including subsystems, trust boundaries, and data flow.
  - **Decompose the application**. Decompose the architecture of your application, including the underlying network and host infrastructure design, to create a security profile for the application. The aim of the security profile is to uncover vulnerabilities in the design, implementation, or deployment configuration of your application.

Web Application Security

- The threat modeling process can be performed using a six-stage process
  - **Identify the threats**. Keeping the goals of an attacker in mind, and with knowledge of the architecture and potential vulnerabilities of your application, identify the threats that could affect the application.
  - **Document the threats**. Document each threat using a common threat template that defines a core set of attributes to capture for each threat.
  - **Rate the threats**. Rate the threats to prioritize and address the most significant threats first.
    - These threats present the biggest risk.
    - The rating process weighs the probability of the threat against damage that could result should an attack occur.
    - It might turn out that certain threats do not warrant any action when you compare the risk posed by the threat with the resulting mitigation costs.

Web Application Security

- The threat modeling process can be performed using a six-stage process



Threat Modeling – Definitions

- A **threat** is a hypothetical interaction in which an application is misused in a way that causes harm.
  - It is a potential misuse of an application (event) that will cause harm if it does occur
- An **attack** is the manifestation of a threat, an actual act of misusing or attempting to misuse a system.
  - An actual attempt to misuse an application
- A **vulnerability** is a **flaw** within the design, coding, or operation of a system that enables an attack.
  - A flaw within an application that enables an attack to succeed

Building a Threat Model

- A **threat model** is a comprehensive list of threats that can be reasonably assumed to apply to the application in question.
- A comprehensive threat model also includes the **profile and objective of the attacker**, **the means of attack**, and **the harm that would likely result from the attack**.
  - The scope of the **threat model depends** upon the **value of the assets maintained within an application.**

Building a Threat Model

- The scope of a threat model depends upon the value of the assets maintained within the application.
- A complete threat model includes a list of threats, including for each:
  - Threat description
  - Attacker profile (skill, resources, motivation)
  - Means of attack
  - Likely damage if the attack succeeds

A threat model for web apps

- Threat models are useful in several ways during web Application development
  - Used to guide software design.
    - Knowing what threats exists enables a designer to build appropriate counter measures into the design
  - Can be one focus of a comprehensive testing plan during software development and subsequent revisions
  - Can also guide system administration.
    - Knowing what threats exist help the administrators to effectively monitor system usage and to spot attacks

Example: A Threat Model Component for an Online Ticketing System

- **Threat:** Attacker gains root control of the system
- **Possible methods:** Buffer overflow attack
  - Any of the **EIGHT** key attacks methods used in Internet Applications
- **Objectives:** Gain root authority, enabling attacker to run arbitrary commands
- **Attacker (goal, experience, resources):** Attacker is a professional criminal, seeking to gain financially from theft of tickets or to cause financial harm to company.
  - Attacker may have access to a botnet, and information about recently discovered vulnerabilities.
- **Likely harm:** Attacker steals tickets, causing financial loss to the system, or attacker compromises the system, causing down-time and ultimate loss of business.

Common Attacks on Web Applications

- Buffer Overflow
- Cross-Site Scripting
- Denial of Service
- Insider Misuse
- Password Guessing
- Sniffing
- Spoofing
- SQL Injection

Buffer Overflow

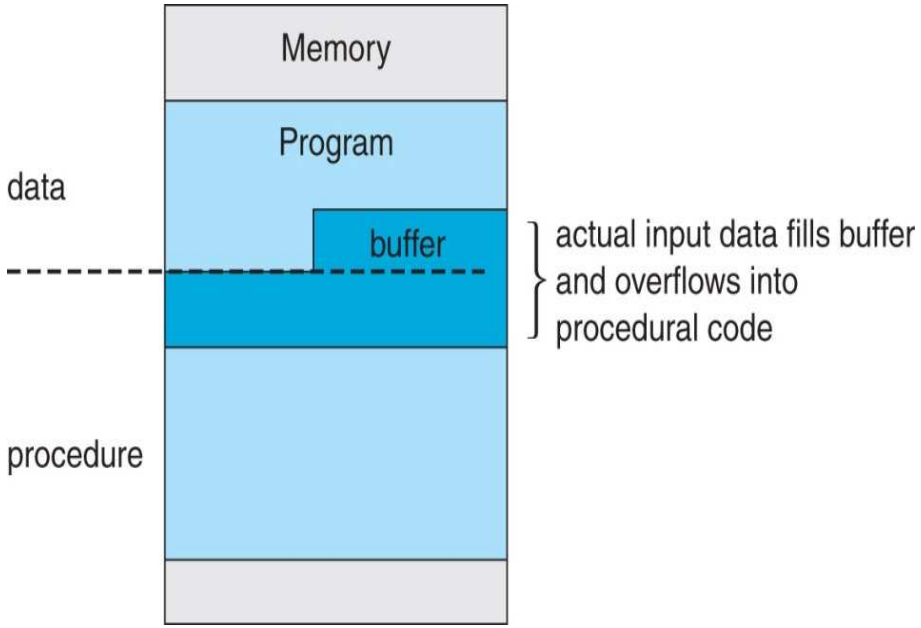
- A buffer is a place in memory that holds data temporarily as it is being input or output.
- An input buffer is a storage location where input data is stored prior to being processed.
- When the data placed into a buffer exceeds its capacity
  - The buffer automatically expends
  - The incoming information is truncated to fit
  - The buffer will be over filled, with the excess data overwriting adjacent areas of the memory.
    - This is known as a **buffer overflow**

Buffer Overflow

- Since the overflow is written into an area of memory that was likely intended for some other purpose, a buffer overflow causes damage by improperly changing memory content.
- **If that memory content happens to be an executable code, the very behaviour of the program is changed**
- A skilled attacker can **exploit** a buffer overflow vulnerability by **entering input that is deliberately larger than the buffer size** and **that contains harmful executable code on its overflow portion**.

Buffer Overflow

- **Buffer:** a memory allocation intended to hold input or output
- **Buffer overflow:** input is too large; excess input overflows into subsequent memory
- If the overflow area contains instructions, the attacker can effectively take control of the application by rewriting critical parts



Detection and defense

- **Detection**
  - Discovered by examining source or executable code, or by trial and error.
- **Defence**
  - Write applications in safe languages that **do not permit buffer overflows** such as Java as opposed to C and C++.
    - C++ is the **WORST** choice
    - Java creates adequate buffers for new strings as needed and checks all subscripts to ensure that they are within legal range.
  - The programmer should also ensure that **every operation on strings and arrays is checked for size limitations**.
    - If the language allows overflows, check each buffer operation against size limits.

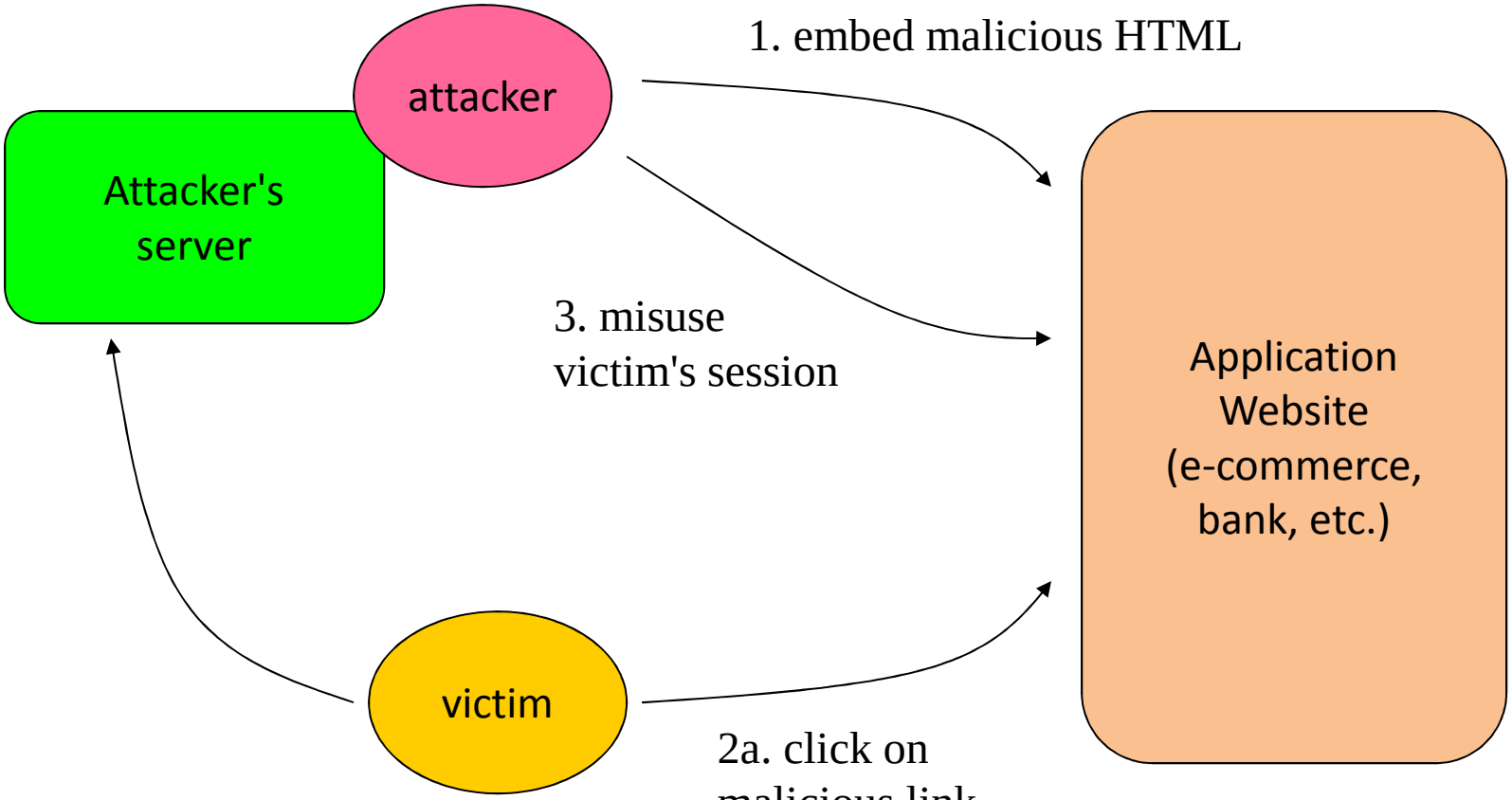
Cross-Site Scripting

- Involves theft of client-side information belonging to one user (victim) by another user (attacker)
- **It requires that the attacker has the authority to post HTML code that the victim can download.**
- Example:
  - An attacker might use a common feature in an e-commerce site to **post a product review containing malicious link**.
  - The link is mislabelled to encourage the victim to click.
  - The link would contain HTML code and Java Script directing the victim's browser to send cookie contents to the attacker's website.
  - The attacker uses victim's session cookie contents to take over victim's session, changes victim's password and then makes fraudulent purchase.

Cross-Site Scripting(XSS) - Example

- The **attacker embeds malicious HTML code** in a public website
- The victim (another user) **clicks on a link in that code**, causing **the victim's cookie to be sent to the attacker's website**
- The attacker then uses the victims cookie to misuse the victim's open session
  - Improper transaction, theft of information, etc.

Cross-Site Scripting



Detection and defense

- **Detection:**
  - To obscure the intent of XSS code make detection more difficult by converting the individual characters in the HTML code to **escape sequences** of the form &#NNN;, where NNN is the numeric code for a specific character
- **Defence:**
  - HTML tag delimiters (< and >) contained in user input intended for public access should be converted to HTML entities (< and >)
    - HTML entities are displayed as characters by browsers rather than being interpreted as HTML code elements and acted upon
  - Establish a list of legal characters and reject any postings that contain any other characters.

Protection Against XSS

- Any text posted by users (comments, reviews, etc.), must be filtered
- Only legal characters should be allowed; delimiters in particular should not be allowed
- All illegal characters should be deleted or replaced by escape characters
  - for example, replace < and > by < and >

Cross-Site Scripting(XSS)

- XSS attacks can generally be categorized as:
  - **Stored,**
  - **Reflected**
  - **DOM Based**
- **Stored XSS Attacks**
  - Stored attacks are those where the injected code is **permanently stored on the target servers**, such as in a database, in a message forum, visitor log, comment field, etc.
  - The victim then retrieves the malicious script from the server when it requests the stored information.
    - So Stored XSS attacks typically consists of two things to do.
    - Initially the attackers visits the application and enters the malicious script into the application.
    - Secondly the user enters into the application and visits the vulnerable page. And the script is executed in the back-end without the knowledge of the user.

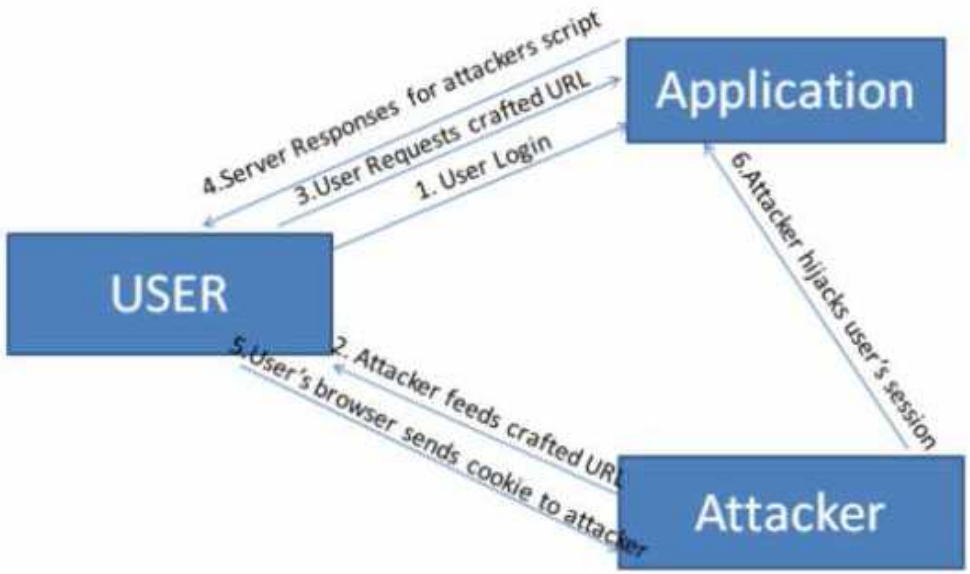
Cross-Site Scripting(XSS)

- Normally these kind of vulnerability exist when an application takes an input from the end-user and stores it in the application, then displays to the other users.
  - For example:
    - Consider Facebook application which allows to comment on pictures or status updates and then displays to all other users on their wall.
    - If the application doesn't sanitize the input properly then an attacker can write a script in the comment area, so that the users who visits or views particular page or post will be effected.
    - These kind of vulnerability normally exists in forums, bidding applications

Cross-Site Scripting(XSS)

- **Reflected XSS Attacks**
  - Vulnerability normally exists in the application that uses dynamic pages to display content to the user (for example: error messages).
  - Reflected attacks are those where the **injected code is reflected off the web server**, such as in an error message, search result, or any other response that includes some or all of the input sent to the server as part of the request.
  - Reflected attacks are **delivered to victims via** another route, such as in **an e-mail message**, or on some other web server.
  - When a user is tricked into clicking on a malicious link or submitting a specially crafted form, the **injected code travels to the vulnerable web server**, **which reflects the attack back to the user's browser.**
  - **The browser then executes the code because it came from a "trusted" server.**

Cross-Site Scripting(XSS)



- 1. User login's to the web application and application issues an unique id 1. (session id) to the URL
- 2. Attacker sends crafted URL to the user through some means of communication
- 3. User requests the crafted URL fed by the attacker
- Server processes the url and responds to the user's request. As the URL is framed with the malicious script, script is executed in the user's browser in the same way it does on other code.
- The code causes the browser to send a request to attackers site. And the request contains current session id
- 6. Attacker uses the Session Id or Cookie Id and hijacks the users session.
- Now the attacker can perform all the actions in the same way how the user does.

Cross-Site Scripting(XSS)

- **DOM Based XSS** (or as it is called in some texts, "type-0 XSS") is an XSS attack wherein the **attack payload is executed as a result of modifying the DOM "environment" in the victim's browser** used by the original client side script, so that the client side code runs in an "unexpected" manner.
- That is, the page itself (the HTTP response that is) does not change, but **the client side code contained in the page executes differently due to the malicious modifications that have occurred in the DOM environment.**

Cross-Site Scripting(XSS)

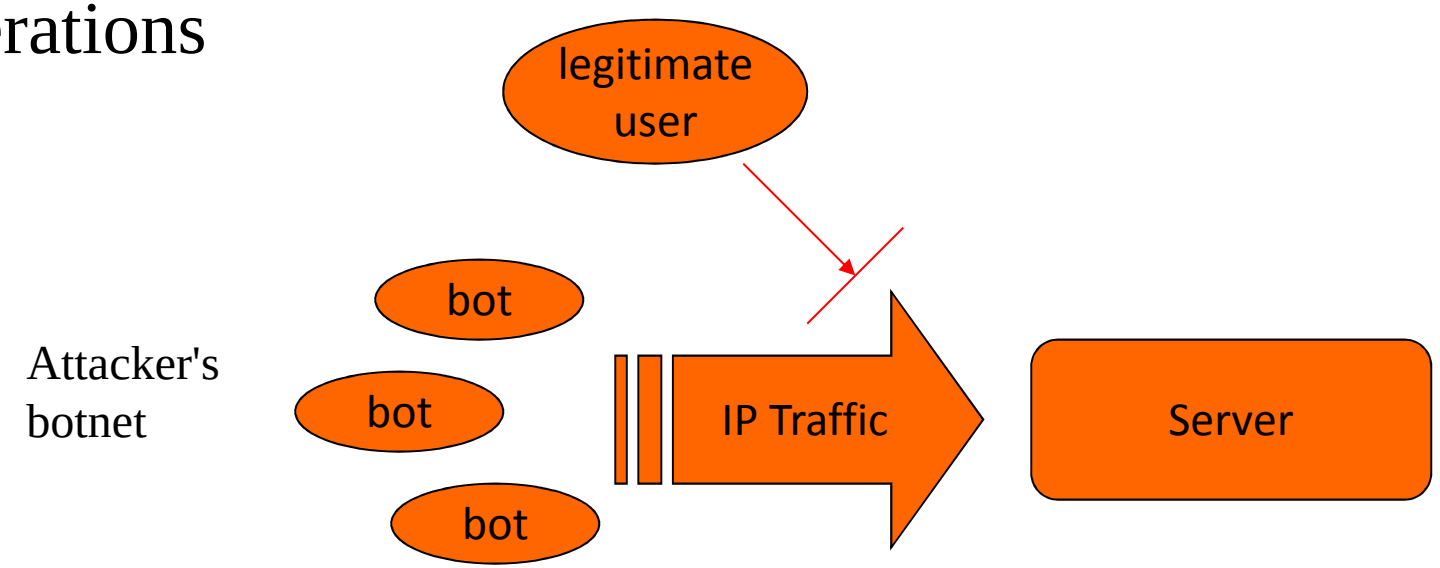
- In DOM based when the user clicks on the crafted URL, server response doesn't consist of any attackers script.
- Instead the browser executes the attacker's script while processing the response.
- This is because the Document Object Model of the browser has a capability to determine the URL used to load the current page.
- Script issued by the application may extract the data from the URL and process it.
- Then it uploads the content of the page dynamically depending upon the script executed through the URL.

Denial of Service

- A denial of service (DOS) is an attempt to prevent legitimate users of a web application from accessing and using it.
  - **Overwhelm the network connection used by the web application** with messages that keep network components so busy that they are **unable to handle legitimate traffic**.
  - **The web application is not directly affected but it becomes inaccessible.**
  - To send HTTP requests to the web application itself such as login attempts thus keeping the application so busy that it cannot respond to legitimate requests.
  - If multiple attack platforms are used, it is **Distributed DOS** (DDOS)

Denial of Service

- DOS attacks do not compromise the integrity or confidentiality of a web application, but it is intended to harm the business/organization
- The attacker bombards the victim's server with spurious internet traffic, so that it is unable to carry out normal operations



Detection and defense

- DOS and DDOS are mitigated at the architectural level with design that anticipates and protects against them.
- **Defence**
  - Use of a **powerful firewall** can detect and delete some or most of the false network traffic accompanying a DOS attack.
    - Use a firewall to filter suspicious IP traffic
  - **Multiple Internet connections** with different IP addresses can be used so that if one connection is blocked, the others remain available.
    - Use multiple internet portals, from different vendors

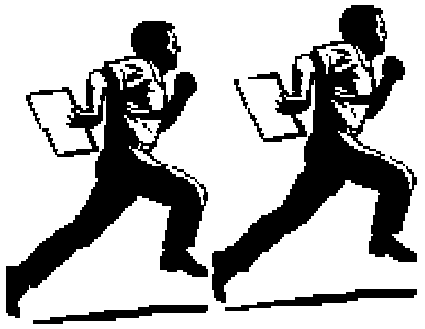
Insider Misuse

- Corporate insiders are the source of many attacks, including
  - Theft
  - Espionage
  - Sabotage (revenge)

- Security mechanisms are often directed at external attackers only, leaving the door open to inside attacks

Insider Misuse

- This is when the ***Expected User behaviour*** is usually different from ***Actual User behaviour***
- This is in the form of:
  - A Disgruntled Employee
  - A Vengeful Employee
  - A Curious Employee
  - A Neglectful or an Irresponsible Employee
  - A Greedy Employee



Insider Misuse

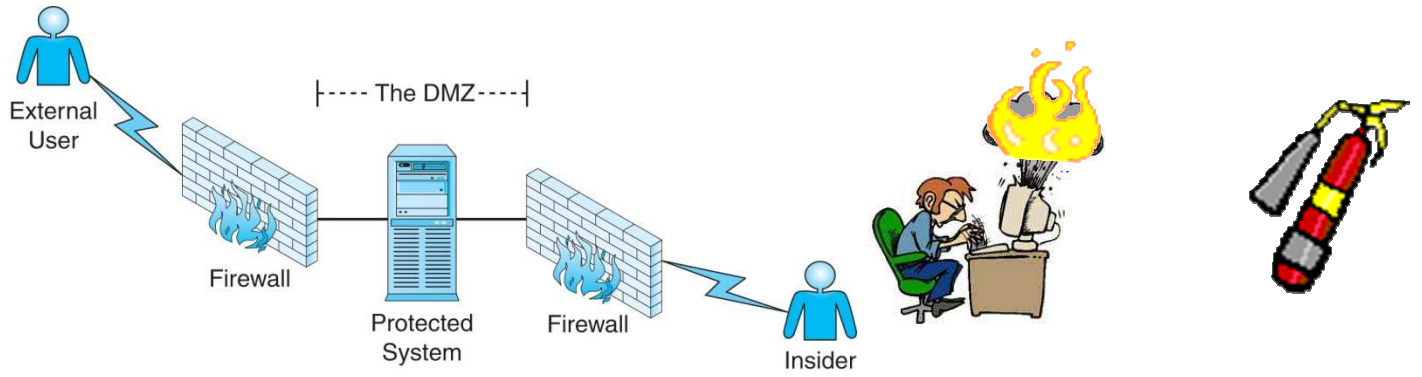
- Insiders include employees, ex-employees, contractors, and other associates who have access to proprietary computer systems
- Typical attacks include fraud, theft of data, and physical property and malicious damage
- Guarding:
  - Proper security policies, procedures, education, and oversight. E.g. requiring strict confidentiality, limiting authorized access
  - Security training and Security audits should be conducted regularly in order to ensure compliance with the policies and procedures
  - Good architectural designs such as internal and external firewalls, creating a demilitarized zone (DMZ)

Insider Misuse

- The main objective is to prevent
  - Internal users from making conscious or subconscious mistakes and other general forms of misuse and abuse of Information Systems
- Develop effective users through
  - Proper training in the use Information Systems
  - Sensitization as to the need to adhere to agreed upon Information Systems security measures
  - Enhance professional maturity
  - Data use is restricted to authorized and monitored access only

Protection Against Insider Misuse

- A so-called "demilitarized zone" protects the application with firewalls from both outside and inside attack



- Insiders are restricted to authorized and monitored access only

Is DMZ a Solution?

- Successive use of IS **MUST** have the user support
  - The protection of the Great Wall of China was only as secure as its gatekeepers. *The invaders simply bribed the gatekeepers who opened the gates, in a 100 years 13 times*
  - KRA employees tampering with the Simba System as reported in the Daily Nation, 6th June 2011, *KRA staff 'tamper with system'*
  - Safaricom employee arrested over allegations of fraudulent loss of funds through M-PESA transactions, as reported in Capital FM online news, October 9, 2009, 'Safaricom staff in MPESA fraud probe'
- **Even the best of security technologies lose their effectiveness if not properly used by the end-users.**

Is DMZ a Solution?

- Employees tend to disregard security policy (Gill, 2006).
- **Therefore**
  - There is a need for a paradigm shift , to a human centric focus: From 'the user is my enemy' to 'the user is my security asset' (Ghonaimy, El Hadidi and Aslan, 2002).
  - Have a list of employee related threats, including for each threat
    - Description, skills required by the attacker, resources available to the attacker, motivation and means of attack, likely damage if the attack succeeds and possible Solution

Is DMZ a Solution?

- There are no 100% full proof solutions as the heart of man is desperately wicked
- Have essential information security policies and controls
  - Have a comprehensive ICT policy
    - Disseminate this to users
  - Have awareness programs and rewards
    - Conduct awareness and training campaigns
  - Considering users as customers, whose needs are to be satisfied.
  - Staff training on security: Provide training and online forums for the staff to be updated on security challenges.

Password Guessing

- People often create passwords from **familiar words, dates, names,** etc.
- Password guessing uses combinations of these in repeated attempts to login
- Account-ids may be guessed also, or may be derived from scanning other sources (e.g., email addresses)

Password Guessing

- It is easy to guess user-ids based on names and e-mail address.
- With a valid user-id in hand, guessing a corresponding password is possible with several automated guessing techniques.
  - **A dictionary based attack:** repeated login are made using common dictionary word or individual names as passwords.
    - This succeeds because users like using passwords that they can remember.
  - **Brute force-attack** that systematically tries all possible password strings and trying random password strings.

Protection Against Password Guessing

- Insist on strong passwords
  - include digits, special symbols
- Delay response for a few seconds after a failed login, to slow down guessing
- Lock the user's account for a short period after repeated failed attempts
  - e.g., 1-hour lock after five consecutive failed attempts

Detection and defence

Defence

- Delay responding to repeated failed login attempts such as adding 5 second delay to login processing
- Lock the account after a number of failed login attempts and require either a personal interaction with the support team or a waiting period before further login attempts
  - These two measures have the drawback of inconveniencing legitimate users who have problems remembering their password.

Detection and defence

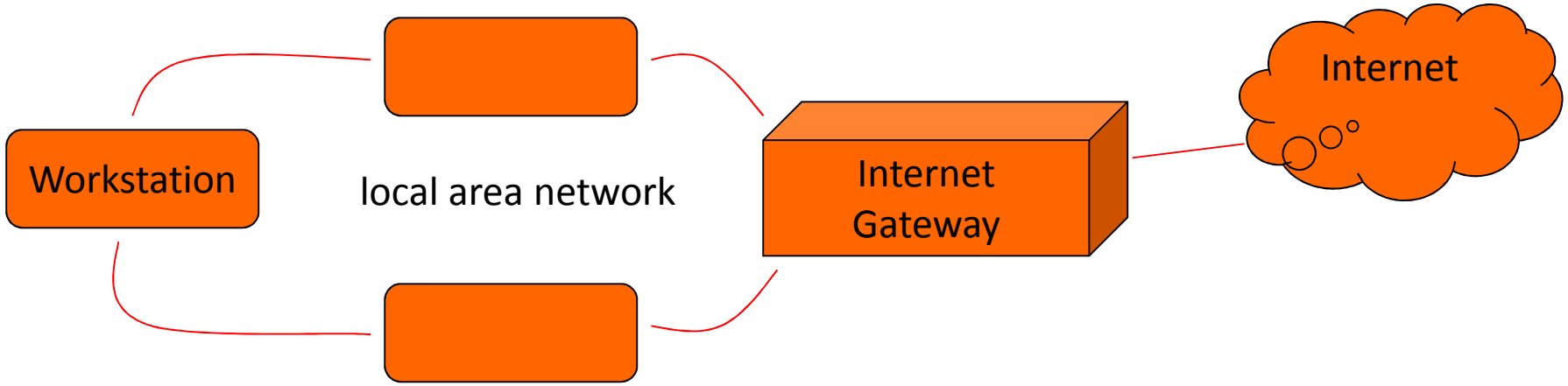
- Impose a minimum constraint on password strength, in its dissimilarity to known words and names, such as requiring each password to be at least eight characters long, must contain at least one digit and one special character and must not contain unbroken dictionary word.
- This increases the cost of dictionary attacks since many more possible combinations must be attempted.
  - The drawback of the measure is that it makes passwords more difficulty to remember and increases the chance that passwords might be written down, which makes them vulnerable to theft

Detection and defence

- Another method to deter automated password guessing is to **add a challenge-response test**.
  - The user attempting to access a protected resource might be presented with a personal question, a simple word problem or a fuzzy image that must be interpreted.
  - These tests are easy for an authorized human user to pass but difficult for a computer as it involve personal memory, language processing, image processing or other tasks that are computationally difficult and expensive.
- Others risks to Internet Applications include **phishing**, which can only be deterred through constant education and monitoring of suspicious activity

Sniffing

- Internet traffic also passes through local networks, where it can be monitored
- Sensitive information that is transmitted in plain text (unencrypted) can be read by other parties on a local network or at an intermediate internet hub



Sniffing

- Web application attacks can also target the link between the server and the client.
- **A packet sniffer** is a program that analyses internet traffic (made up of data packets) and searches for certain patterns.
- The **purpose** of the a sniffer can be benign (intrusion detection) or malicious (theft of confidential information).
- Passwords, personal information, credit card numbers are subject to theft in this manner if not adequately protected

Sniffing

- Breaches of confidentiality can be prevented by **encrypting all transactions** that include sensitive information by using Secure Hypertext Transfer Protocol (HTTPS).
- **Using HTTPS** is more expensive than using HTTP.
  - The practical solution is to use HTTPS only when transactions require confidentiality.
- All hyperlinks, HTTP form action attributes, and AJAX operations should be carefully considered.

Protection against Sniffing

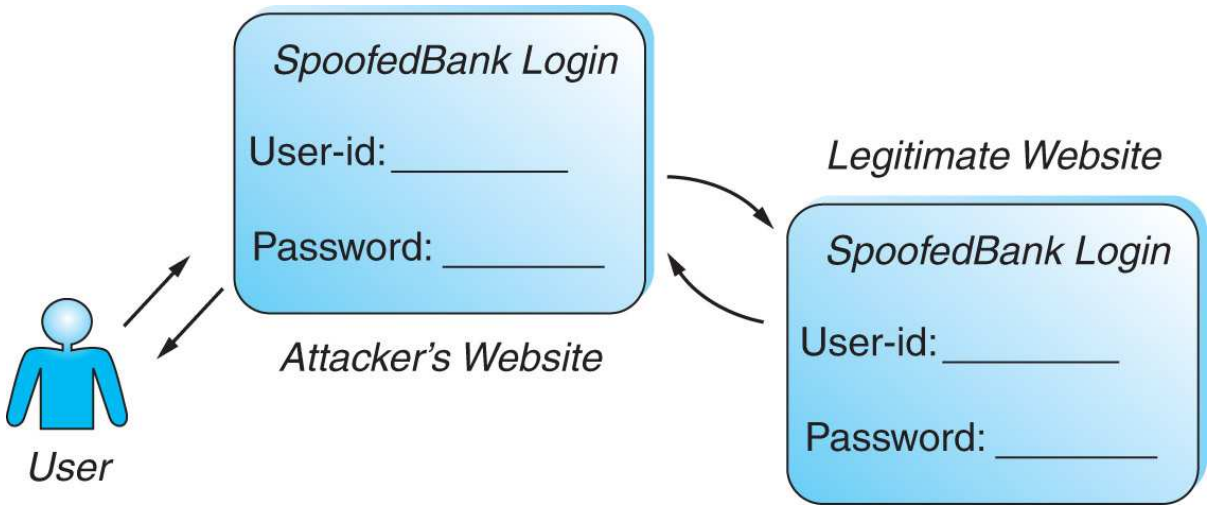
- Use HTTPS for any transactions containing sensitive information
  - Login credentials
  - Financial information
  - Personal information, etc.

Spoofing

- This is not a direct attack on the application but an attempt to steal confidential information directly from application users.
- A spoofing attack entails engaging a user in a transaction in which the user believes he or she is interacting with a legitimate web application when in fact the interaction is with the attacker's web application instead.
- **A simple spoofing is to create a replica of a legitimate web application on attacker's server and then trick users into logging into the attacker's version of the application enabling theft of password.**
- **A sophisticated spoofing will pass actions through the legitimate application that is spoofing and then return the legitimate responses to the user while recording confidential data**

Spoofing

- The attacker creates a phony website that spoofs a real website (bank, etc.)
- Victims are tricked into logging in to the phony site
- The attacker uses stolen credentials to access victims' accounts
- With real-time spoofing, the attacker's website communicates with the spoofed website



Spoofing

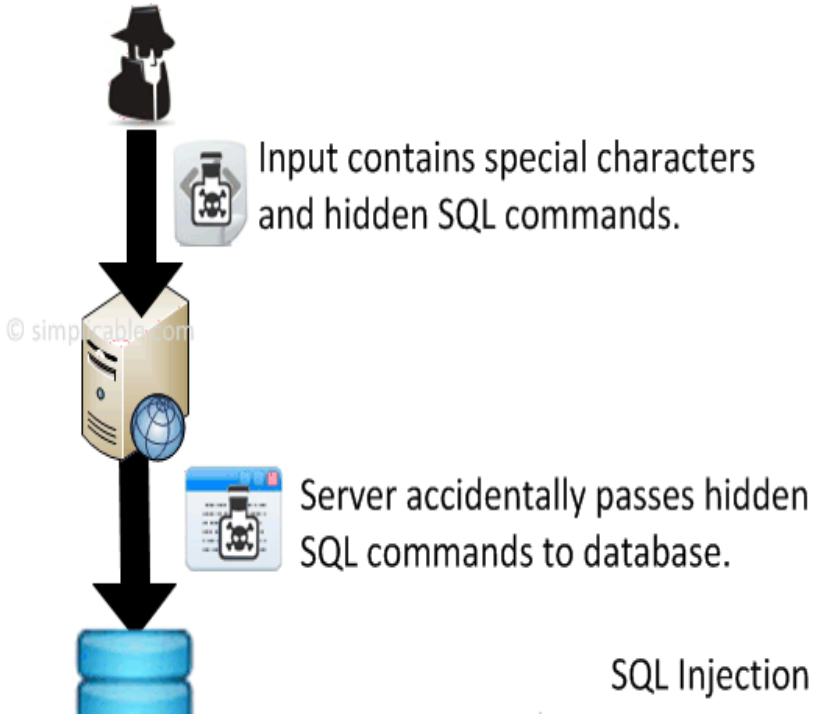
- The most effective countermeasure for spoofing is user education.
- **Users should learn not to click on hyperlinks found on e-mail messages or any untrusted websites.**
- A URL of a trusted and secure web application should be typed directly into the browser URL field, or selected from an existing bookmark list.
- Another countermeasure is to present customized picture on the home page of an application, so that each user sees something unique.

Protection Against Spoofing

- Educate users never to click on emailed hyperlinks, and to use their own bookmarks or a typed URL instead
- Use a challenge-response test to verify user identify
  - (doesn't help with real-time spoofing)

SQL Injection

- SQL injection attack consists of injection of malicious SQL commands via input data from the client to the application that are later passed to an instance of a database for execution and aim to affect the execution of predefined SQL commands.



SQL Injection

- This involves entering SQL code into a text field that will become part of an SQL expression.
  - SQL injection consists of direct insertion of code into user-input variables which are concatenated with SQL commands and executed.
  - A less direct attack injects malicious code into strings that are destined for storage in a table or as metadata.
  - When the stored strings are subsequently concatenated into a dynamic SQL commands, the malicious code is then executed.
- If it succeeds, the attacker is able to bypass security controls and write arbitrary SQL expressions and thereby change any information in an application database.
  - The defence against SQL injection is to ensure each **data field must be filtered by comparing its character against a set of legal characters.**

SQL Injection

- A successful SQL injection exploit can
  - Access sensitive data in the database,
  - Modify database data,
  - Execute administrative operations within the database (e.g. shutdown the DBMS),
  - Recover the content of a given file present on the DBMS file system
  - And in some cases issue commands to the operating system.

SQL Injection

- Attacker inserts SQL into an input field
- The application embeds the inserted syntax into its own SQL commands
- The attackers SQL is executed

Userid: xyz

Password: 1' OR 'x'='x

```
String sqlCommand =
  "SELECT user FROM users" +
  "WHERE userid = ' "  + userid_in  +
  "  AND password = ' " + password_in;
```

Resulting SQL Command:

```
SELECT user FROM users
  WHERE userid = 'xyz' AND password = '1' OR 'x' = 'x'
```

Always TRUE!

Secure Design

- Security cannot be added after development is complete, but rather it must be built in during every step of application development.
- Several activities or principles must be followed during specification design, coding, and testing of a web application that will help to ensure security characteristics of the final products.

Secure Design

- Develop a realistic threat model
  - It is essential to understand the viable threats in order to counter them effectively This includes:
    - The attacker's characteristics (motivation, skill, experience)
    - The resources available to the attacker (time, computing power, assistance)
    - The objective of the attacker (data manipulation, data capture)
    - Possible means of attack (buffer overflow, password guessing)

Secure Design

- During the design this information can help ensure that necessary security features are included and that vulnerabilities are excluded from the application.
- During testing, the threat model provide a test strategy
- It can also ensure that the operational procedures for a deployed application are sound and secure.
- There are several to document a threat model such as abuse cases and security oriented risk analysis, a list of common vulnerabilities, and case studies

Secure Design

- Follow a reliable architectural pattern, such as MVC
  - MVC support efficiency and security through separation of concerns
  - MVC separates design problems and operation of a web application and it helps to compartmentalize application resources in order to protect them.
  - The view component in particular provides public access to a web application and therefore must be given particular attention with respect to security, in order transactions are properly secured and there is no possibilities of misuse.

Secure Design

- Limit Capabilities to what is actually necessary
  - It is relatively easy to prove that the required capabilities are present but very difficulty to prove that only the required capabilities are present.
  - Vulnerabilities create unrequired/unwanted capabilities
  - During design and coding, only add the required functionalities
  - Necessary rules should be enforced on the server side
    - Why? Client validation can be circumvented easily.
    - You **should always validate sensitive data on server**, regardless of client validation.
    - Validating them on client too is just a matter of improved user experience, but the client side validation saves the user the trouble of submitting data when you already know it will get rejected.

Secure Design

- Require and enforce Authorization
  - Many web applications require authorization or login in order to access certain services and must always be used universally to control access
  - It is essential to guard resources that should be accessed only by logged-in users
- Use HTTPS and ensure secure access
  - It is necessary to use HTTPS universally, HTTP can be disabled at the server level, leaving HTTPS as the only option.

Principles for Secure Design

1. Develop a realistic threat model – know the potential attackers
2. Follow a reliable design pattern, such as MVC
3. Limit user capabilities to only what is needed (least privilege)
4. Require and enforce authorization (don't allow users to sneak past login pages!)
5. Require and enforce HTTPS access for secure resources

New HTML Security Management

- "Content Security Policy"
  - Simple policy format, that tells the browser which JavaScripts are legitimate
    - Baseline rules
    - No inline scripts
    - No string-to-code conversation
- XSS Filters
  - Compare input parameters with JavaScript content of the HTTP response
  - If a match can be spotted, disarm the script
    - (In theory) capable of stopping reflected XSS
- Sandboxed Iframe
  - In sandboxed Iframe, JS execution is prevented
    - Render untrusted data in sandboxed Iframes to stop XSS-based JS
    - Problem:
      - Layout loses rendering flexibility

Web Application Security