COMMON VULNERABILITIES AND SCALABILITY

Sessions 1A and 1B
Common Vulnerabilities

- **Buffer overflow**

```c
int main(){
  if( login() ){
    start_session();
  }
  return 0;
}

int login(){
  char passwd[4];
  gets( passwd );
  return (strcmp(passwd, "mpw")==0);
}

int start_session() { ... }
```
Common Vulnerabilities

- Buffer overflow

```c
int main()
{
    if( login() ){
        start_session();
    }
    return 0;
}

int login()
{
    char passwd[4];
    gets( passwd );
    return (strcmp(passwd, "mpw")==0);
}

int start_session() { ... }
```
Common Vulnerabilities

- **Buffer overflow (Solution)**

  - Activate a “Stack Guard” to insert Canary.
  - Give minimum privileges to a program.
  - **Never trust user input**: don’t use unsafe string functions in C (like `strcpy`, `gets`, `strcat`, `sprintf`, etc.).
Common Vulnerabilities

- **Client-State Manipulation**

  ```html
  <form>
    <input type="hidden" name="price" value="50" />
  </form>
  
  Value = $50
  
  Value = $1
  
  - Don't store **sensitive** information in the client (only a Session ID).
  - **Encrypt a checksum** (using a **signature** prior storing in the client).
  - Attach either an **expiration date** or the Session ID to client’s state.
Common Vulnerabilities

- SQL Injection

```
“SELECT * FROM product
WHERE id = ‘ + user_input + ‘;’
123; DROP TABLE product;
```

- Create users with minimum privileges.
- Use prepared statements.
- Encrypt sensitive data in DBMS.
- Don’t trust user input!
Common Vulnerabilities

- Command Injection

```java
system("cp f1.dat $user_input");

f2.dat; rm /etc/passwd
```

- Don’t use the `System` command. Use `Runtime.exec()` instead.
- “Taint” variables.
- Give minimum privileges to your application.
Common Vulnerabilities

- **Cross-Site Scripting (XSS)**

  `<body>`
  
  Welcome to $user_input$ website
  
  `</body>`

  You’re doomed `<script>hacked();</script>`

  - Why is it called **Cross-Site**?
  - It’s very difficult to protect against it — we want to allow users insert HTML.
  - 2 options: **white listing**, **black listing**. *Which one is better?*

  `<div fu="alert('XSS!');" STYLE="background-image: url(javascript:eval(this.fu))">`
Cross-Site Request Forgery (XSRF)

- Always logout!
- On server side, use an action-token.

www.evil.com submits a form
Browser sends valid cookie along the request
Scalability

How to estimate our server capabilities:

- **HDD**
  - 10ms/access

- **RAM**
  - 100MB/s
  - Sequential access: 10,000 pages/s
  - Random access: 100 pages/s

- **Network**
  - 1Gb/s
  - Up to 10,000 pages/s

When a static page is 10KB
Scalability

How to estimate our server capabilities:

- **HDD**: 10ms/access
- **RAM**: 100MB/s
- **Network**: 1Gb/s

When a **dynamic** page is generated, people tolerate up to **100ms** for a page to be displayed.

- **10 requests/s/CPU**
Scaling Web Applications

- Two types of scaling:
  - Scale **up** ↑
  - Scale **out** ⇆
Scaling Web Applications

- The Load Balancer
  - TCP-NAT Request Distributor
    - DNS Round Robin, or software

- In our projects Apache Tomcat is both in the application and HTTP layers.
Scaling the Storage Layer

- **Scenario 1: Read Only**
  - Information doesn’t change. Clients only read data.
  - Use replication.

![Diagram of master and slaves for synchronization](image-url)
Scaling the Storage Layer

- **Scenario 2: Local Read/Write**
  - Reads and writes are scoped to individual users.
  - Use **shard** or **partitioning**.

![Diagram showing partitions]
Scaling the Storage Layer

- **Scenario 3: Global Read/Write**
  - Reads and writes are global, and all users can see everyone's updates.
  - Use **partitioning** and **replication**.

![Diagram of partitioning and replication in two partitions.](image)