Secure Software Design and Engineering
(CY-321)

A Day at the Races

Dr. Zubair Ahmad

Race Condition

A type of programming error that can occur when multiple processes or threads can access shared data or resources concurrently

The outcome depends on the timing or order of execution

Change state property

Concurrency property

When Race Condition happens?

Shared object property

What happens to software if race conditions happen?

The software can produce incorrect or inconsistent outputs, such as wrong calculations, corrupted data, or missing information.

The software can behave unpredictably or unreliably

The software can become **vulnerable or insecure**, such as leaking sensitive information, allowing unauthorized access, or executing malicious code

Why do race conditions happen?

Processes or threads are running on different machines or cores with different clocks or speeds.

Communicate through different channels or protocols and have different latencies or reliabilities

Have different logic or goals because they use different algorithms or strategies

Race Condition - Example

```
unsigned char* read_file(const char* filename) {
if (access(filename, R_OK) == 0) {
        int fd = open(filename, O_RDONLY);
        if (fd < 0) {
            return NULL;
        }

        unsigned char* buf = malloc(1024);
        if (buf != NULL) {
            (void) read(fd, buf, 1024);
        }

        (void) close(fd);
        return buf;
    } else {
        return NULL;
    }
}
```

Avoiding Race Condition

Use open() Directly Without access() **(Race Window)**

Use O_NOFOLLOW to Prevent Symlink Attacks

```
int fd = open(filename, O_RDONLY | O_NOFOLLOW);
```

Use flock() to Lock the File

Use Temporary Directories

Avoiding Race Condition

Using Mutex Operations (Mutual Exclusion Lock)

```
void withdraw(int amount) {
    if (balance >= amount) {  // 🚨
Another thread might change
"balance" here
        balance -= amount;
    }
}
```

```
pthread_mutex_t lock =
PTHREAD_MUTEX_INITIALIZER;

void withdraw(int amount) {
    pthread_mutex_lock(&lock);  //  Lock before
modifying balance
    if (balance >= amount) {
        balance -= amount;
    }
    pthread_mutex_unlock(&lock);  //  Unlock
after modification
}
```

File Attacks

Attacks against software are also prevalent when data is exchanged in files

Any feature in software that use external object references (such as URLs and file system references) and which allow the upload of images (.gif,.jpg,.png, etc.), documents (.docx,.xlsx,.pdf, etc.) and other files, are potential sources of attack vectors

File Attacks

Malicious File Execution

When an application allows users to upload or include executable files, leading to remote code execution (RCE

File Attacks

Malicious File Execution

A **hacker** uploads `shell.php`

```
const express = require('express');
const fileUpload = require('express-fileupload');
const app = express();

app.use(fileUpload());

app.post('/upload', (req, res) => {
    let file = req.files.file;
    file.mv('./uploads/' + file.name, (err) => {
        if (err) return res.status(500).send(err);
        res.send('File uploaded successfully!');
    });
});
```

Accessing `https://example.com/uploads/shell.php?cmd=whoami` executes the command on the server.

File Attacks

Avoiding Malicious File Execution

Only allow image or document file types.

Ensure that the uploaded file is of the correct format.

Prevent direct execution of uploaded files.

File Attacks

Path Traversal Attack

Exploits improper handling of file paths, allowing attackers to access restricted files

File Attacks

Path Traversal Attack

Attacker requests:

```
https://example.com/readfile?file=../../etc/passwd
```

This reads the system's password file.

```
const express = require('express');
const app = express();
const fs = require('fs');

app.get('/readfile', (req, res) => {
    let filename = req.query.file;
    let filePath = './uploads/' + filename;

    fs.readFile(filePath, 'utf8', (err, data) => {
        if (err) return res.status(500).send("Error reading file");
        res.send(data);
    });
});
```

File Attacks

Path Traversal Attack

Use path.join() to normalize paths.

Restrict access to files within a specific directory.

Validate user input to prevent ../ sequences

File Attacks

Download of Code Without Integrity Check

when an application downloads and executes external code without verifying its integrity.

Attackers can inject malicious scripts if integrity checks are missing.

File Attacks

Download of Code Without Integrity Check

```
fetch("https://cdn.example.com/script.js")
    .then(response => response.text())
    .then(code => eval(code));  // Executing
unverified external code
```

If an attacker compromises https://cdn.example.com/script.js, they can inject malicious JavaScript, leading to **Cross-Site Scripting (XSS)**

File Attacks

Avoiding Download of Code Without Integrity Check

Use Subresource Integrity (SRI) when loading external scripts.

Validate SSL/TLS Certificates when fetching remote content.

Avoid eval() and execute external code cautiously.

Phishing

Vishing

Social Engineering Techniques

Pharming

SMSishing

Semester Project

Semester Project will be shared today
or tomorrow on github and Teams

Questions??

zubair.ahmad@giki.edu.pk

Office: G14 FCSE lobby