Introduction to Robust and Secure Programming

- Robust and secure programming is the foundation of reliable software systems.

Introduction to Robust and Secure Programming (Continued)

- Cybercrime damages are expected to reach $10.5 trillion annually by 2025.

Software Development Best Practices

- Adopting best practices ensures maintainability and security in software.

Software Development Best Practices (Continued)

- 'Code is like humor. When you have to explain it, it’s bad.' – Cory House

Input Validation and Output Encoding

- Sanitizing inputs prevents a majority of injection attacks.

Input Validation and Output Encoding

```
if (empty($_POST["name"])) {
        $errors['name'] = "Name is required.";
    } else {
        $name = filter_var(trim($_POST["name"]), FILTER_SANITIZE_STRING);
        if (!preg_match("/^[a-zA-Z ]*$/", $name)) {
            $errors['name'] = "Only letters and white spaces are allowed in the name.";
        }
    }
```

Input Validation and Output Encoding

```php
<?php
trim(string $string, string $characters = " \n\r\t\v\0"): string
?>
```

```php
<?php
filter_var(mixed $value, int $filter = FILTER_VALIDATE_EMAIL, array|int $options = 0): mixed
?>
```

PHP has a bad reputation, but is actually very good at solving this type of problem, using native tools.

A mix of built in methods (filter validate email according to RFC 5321 and RFC 5322) that basically acts like regex + some removal of whitespace (new strings, tabbing etc etc.)

Input Validation and Output Encoding

```php
<?php
declare(strict_types=1);

function processInput(string $name, string $email, ?int $age = null): void {
    if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
        throw new InvalidArgumentException("Invalid email format.");
    }

    if ($age !== null && ($age < 0 || $age > 120)) {
        throw new InvalidArgumentException("Age must be between 0 and 120.");
    }
?>
```

Input Validation and Output Encoding (Continued)

- OWASP ranks input validation issues among the top security risks.

Input Validation and Output Encoding (Continued)

```php
<?php
htmlspecialchars(string $string, int $flags = ENT_QUOTES | ENT_SUBSTITUTE, string $encoding = 'UTF-8', bool $double_encode = true): string
?>
```

Character	Conversion
&	&amp;
<	&lt;
>	&gt;
"	&quot;
'	' (if ENT_QUOTES is set)

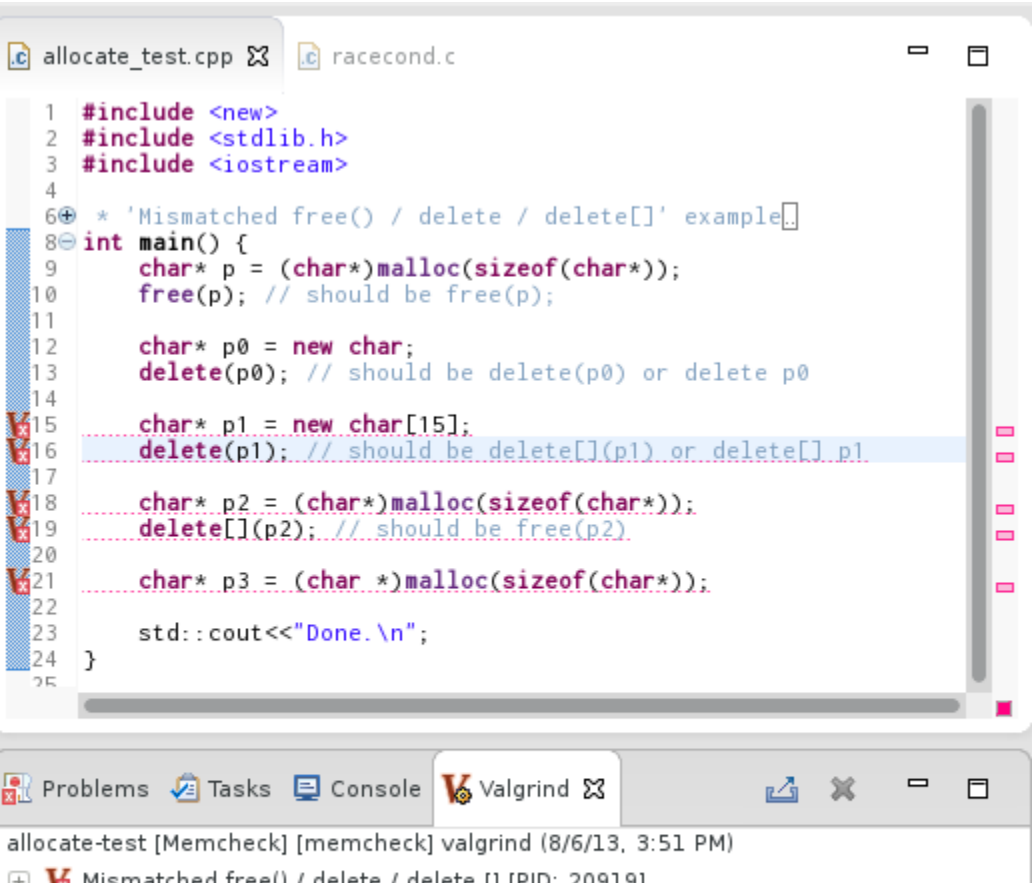
```php
<?php
$input = '<script>alert("XSS Attack!")</script>';
$safe = htmlspecialchars($input, ENT_QUOTES, 'UTF-8');

echo $safe;
Output: &lt;script&gt;alert(&quot;XSS Attack!&quot;)&lt;/script&gt;
?>
```

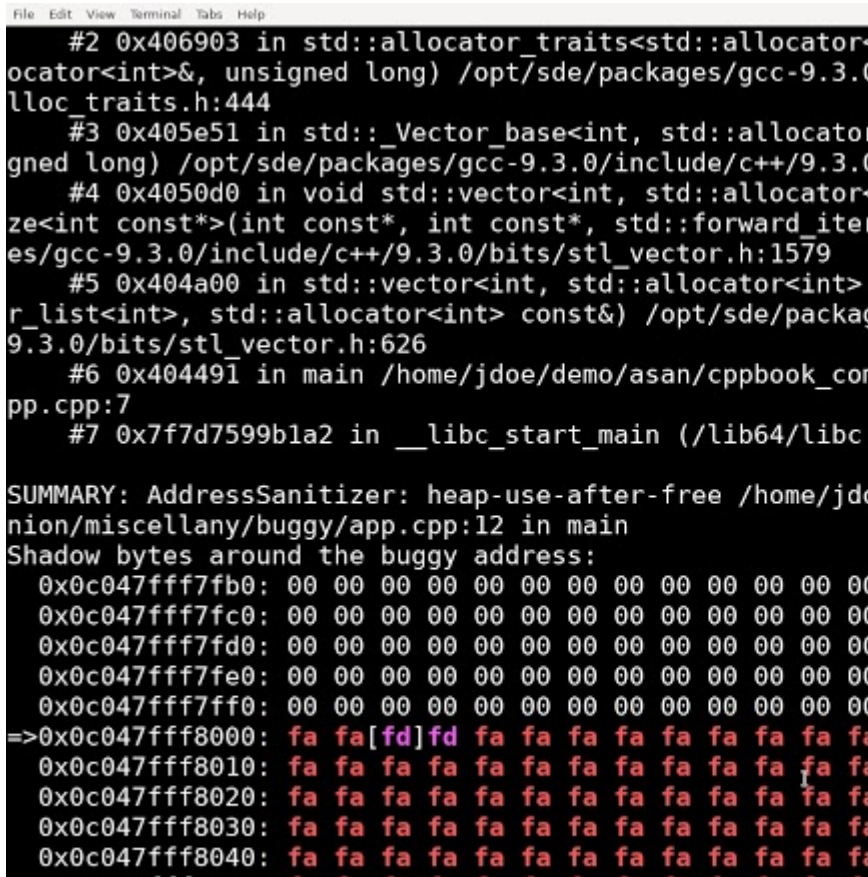
Secure Memory Management

- Memory safety errors account for 70% of all vulnerabilities in critical systems.

Secure Memory Management (Continued)

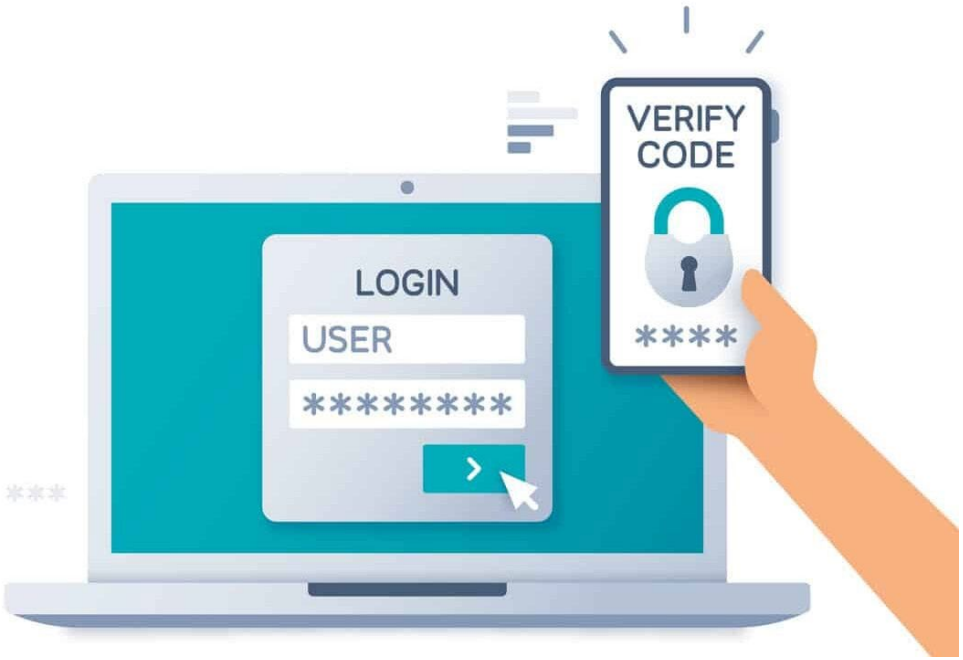
- Tools like Valgrind and AddressSanitizer can identify memory-related issues.





Authentication and Authorization

- Authentication ensures only legitimate users can access the system.



Authentication and Authorization (Continued)

- Multi-factor authentication blocks 99.9% of account compromise attacks.

```
const speakeasy = require('speakeasy');
const qrcode = require('qrcode');

// 1. Generate a secret and display the QR Code
const secret = speakeasy.generateSecret({ name: 'YourAppName' });
qrcode.toString(secret.otpauth_url, { type: 'terminal' }, (err, qr) => {
  console.log(qr); // Show QR code in the terminal for scanning
  console.log('Secret key:', secret.base32); // Backup key for the user
});

// 2. Verify user-provided token
const userToken = '123456'; // Replace with input from the user
const isValid = speakeasy.totp.verify({
  secret: secret.base32,
  encoding: 'base32',
  token: userToken,
});
```

Data Protection

- Encryption and hashing protect sensitive data from unauthorized access.

Data Protection (Continued)

- AES encryption is used by over 70% of secure applications worldwide.

AES

Imagine a super-secret blender!

- The Key: This is like your secret password, only way more complicated. It's a long string of numbers that only you and your friend know.
- Locking the Data: You put your milkshake recipe into the blender. AES chops it up then uses your secret password to whiz it all around, mixing and scrambling it until it looks like crumbs.
- Unlocking the Data: Your friend has the same blender and the same secret password. They put the crumbs in, type in the password, and the blender magically turns the milkshake back to fruit!

AES

- Why is it so secure?
- The configuration makes it an infeasible scenario to bruteforce. You would have to try all manner of combinations of berries, milk etc to crack

  the recipe!

Concurrency and Multithreading

- Securely managing threads avoids race conditions and data corruption.

Concurrency and Multithreading (Continued)

- Deadlocks and race conditions cost organisations millions in downtime.

Concurrency and Multithreading (Continued)

Deadlock

- Imagine two friends, each holding a different toy the other wants. They both refuse to give up their toy until they get the other's, resulting in a standstill where neither can play. They're stuck!

Concurrency and Multithreading (Continued)

Race Condition

- Two siblings race to grab the last cookie from the jar. Whoever reaches it first gets to eat it, but it's unpredictable who will win. The outcome of the "race" determines who gets the cookie.

Deadlock

```python
from threading import Lock, Thread

lock1 = Lock()
lock2 = Lock()

def task1():
    lock1.acquire()
    lock2.acquire()  # This will block if task2 has lock2
    # ... do something ...
    lock2.release()
    lock1.release()

def task2():
    lock2.acquire()
    lock1.acquire()  # This will block if task1 has lock1
    # ... do something ...
    lock1.release()
    lock2.release()

# Start both tasks concurrently
Thread(target=task1).start()
Thread(target=task2).start()
```

Race conditions

```python
from threading import Thread

counter = 0

def increment():
    global counter
    for _ in range(1000000):
        counter += 1

# Start multiple threads incrementing the counter
Thread(target=increment).start()
Thread(target=increment).start()

# The final value of counter will likely be less than 2000000
print(counter)
```

Secure Software Design Patterns

- Secure design principles proactively mitigate vulnerabilities.

Secure Software Design Patterns (Continued)

- 'Design is not just what it looks like and feels like. Design is how it works.' – Steve Jobs

Secure Deployment and Configuration

- A secure deployment ensures no sensitive data leaks into production.
- The reading task for this week covers some best practice to avoid cybersecurity issues.

Secure Deployment and Configuration (Continued)

- Misconfigurations are a leading cause of cloud breaches.
- Leaking secure keys, obscurities in the config etc can all cause issues.
- This is why tools like Docker are becoming more popular. Container-based systems take a ‘snapshot’ of a system and deploy it as is.

Vulnerability Assessment and Mitigation

- Vulnerability assessments help identify weak points in software.

Weapon Slot	Generates ammo for	Ammo generated by
1	Mostly 2, sometimes 3	Mostly 2, sometimes 3
2	Mostly 1, sometimes 3	Mostly 1, sometimes 3
3	Mostly 1, sometimes 2	Mostly 1, sometimes 2

Weapon Slot	**Generates ammo for**	**Ammo generated b**
1	Always 3	Only 3
2 [disabled]		
3	Always 1	Only 1

Vulnerability Assessment and Mitigation

- Vulnerability assessments help identify weak points in software.

Glitch in action

Vulnerability Assessment and Mitigation (Continued)

- 'Fixing a vulnerability in development costs 6 times less than in production.'

Compliance and Regulatory Considerations

- Security standards ensure compliance and trust in sensitive domains.

Compliance and Regulatory Considerations (Continued)

- GDPR fines totaled $1.2 billion in 2022, highlighting the cost of non-compliance.

Emerging Trends and Advanced Topics

- Quantum computing challenges traditional encryption methods.

  https://www.youtube.com/watch?v=WPZ2LHcN80I&ab_channel=Goldsmiths%2CUniversityofLondon

Emerging Trends and Advanced Topics (Continued)

- AI-powered threat detection systems reduce time-to-detection by 50%.

SolarWinds Hack

- The SolarWinds hack exposed vulnerabilities in the supply chain, impacting over 18,000 organizations.

- Lessons Learned: Importance of securing supply chains and monitoring third-party software.

  https://www.techtarget.com/whatis/feature/SolarWinds-hack-explained-Everything-you-need-to-know

Equifax Data Breach

- The Equifax breach exposed sensitive information of 147 million people.
- Lessons Learned: The importance of patch management and encrypting sensitive data.

  https://archive.epic.org/privacy/data-breach/equifax/

Log4j Vulnerability

- The Log4Shell vulnerability affected millions of Java-based applications worldwide.
- Lessons Learned: Importance of monitoring open-source dependencies and quick patching.

  https://www.ncsc.gov.uk/information/log4j-vulnerability-what-everyone-needs-to-know

Capital One Cloud Misconfiguration

- A misconfigured firewall exposed sensitive data of 100 million customers.
- Lessons Learned: Securing cloud configurations and implementing least privilege access.

  https://cert.europa.eu/publications/threat-intelligence/threat-memo-190802-1/pdf

Uber Data Breach

- A data breach compromised the information of 57 million riders and drivers.
- Lessons Learned: The importance of strong authentication and transparency in incident response.

  https://www.upguard.com/blog/what-caused-the-uber-data-breach