



Presentation

Only for course Teacher					
	Needs Improvement	Fair	Good	Excellent	Total Mark
Delivery					
Content/Organization					
Enthusiasm/Audience Awareness					
Comments					

Semester: Fall-2023

Student Name: Md Eyashin

Student ID: 211-35-712

Batch: 34th

Section: A

Course Code: SE332

Course Name: Information System Security

Course Teacher Name: Md. Maruf Hassan

Designation: Associate Professor, Department of Software Engineering

Submission Date: 01/12/2023



CODE VULNERABILITY & COMPROMISED SW QUALITY-A MAJOR THREAT

MD. EYASHIN

ID: 211-35-712

SECTION: 34-(A)

INFORMATION SYSTEM SECURITY
DEPARTMENT OF SOFTWARE ENGINEERING



[Click here](#)

Code Quality.

Quality Matters:

The code quality is important, as it impacts the overall software quality. And quality impacts how safe, secure, and reliable your codebase is. High quality is critical for many development teams today. And it's especially important for those developing safety-critical systems.

Vulnerable Code:

Vulnerable code is software programming that contains weaknesses. These weaknesses can lead to security breaches, unauthorized access, and other threats to the integrity and functionality of a system or application.

Code Quality Analysis

Good Code vs Bad Code:

Good code is high quality. And it's clean code. It stands the test of time. Bad code is low quality. It won't last long.

Good Code:

- Readability
- Maintainability
- Efficiency
- Comments and Documentation
- Scalability
- Security

Bad Code:

- Poor Readability
- Hard-Coding
- Duplication
- No Documentation
- Security Vulnerabilities
- Lack of Error Handling

Code Quality Analysis

Poor Quality Code is a Security Threat:

Poor quality code can pose significant security threats. Security vulnerabilities often arise from common coding mistakes and bad practices.

Mitigate Risk:

"Prevention through detection" succinctly captures the idea that identifying issues or potential threats early on is a key strategy for preventing more significant problems or damage later. This phrase emphasizes the proactive nature of addressing concerns before they escalate.

Code Quality Analysis

Improve Code Quality:

Using four ways to improve our code quality:-

- Use a coding standard
- Analyze code - before code reviews
- Follow code review best practices
- Refactor legacy code (when necessary)

Static Code analysis

Static code analysis is a technique used to evaluate source code without executing it. It involves analyzing the code for potential vulnerabilities, bugs, or other issues before the program is run. This analysis is typically performed by specialized tools that examine the code structure, syntax, and other properties to identify potential problems.

Advantage:

- Early Detection of Issues
- Consistent Code Standards
- Automated Code Reviews
- Time and Cost Efficiency
- Code Quality Improvement

Security Issues through Code Analysis

Top web application security flaws or security issues that can be found in a vulnerable code are –

- Unvalidated Input
- Broken Access Control
- Broken Authentication and Session Management
- Cross Site Scripting (XSS) Flaws
- Buffer Overflows
- Injection Flaws
- Improper Error Handling
- Insecure Data Handling
- Insecure Storage
- Denial of Service
- Insecure Configuration Management



Buffer Overflow

Buffer overflow is a critical security risk. It occurs when an operation that writes data to an allocated buffer exceeds the buffer boundaries. Thus, the operation accesses adjacent memory locations that it should not.

```
#define MAXSIZE 100
...
...
Char localBuf [MAXSIZE]
...
...
Gets (localBuf)
```



Severity & Impact:

A severe security breach could empower attackers to manipulate your code, potentially leading to the worst-case scenario where they gain full control over your system.

Post build analysis issues

```
43 OutputStream stream = null;
44 try {
45     for (String property : propertyList) {
46         stream = new FileOutputStream(property);
47     }
48 } catch (Exception e) {
49     //Catch the exception
50 } finally {
51     stream.close();
```

A "NullPointerException" could be thrown; "stream" is nullable here. Why is this an issue? 27 minutes ago L51

Bug Major Open Not assigned 10min effort Comment cert, cwe

The stream variable (line 51) could be nullable here. So, closing the variable without checking null can throw an exception.

The result will not be correct due to missing type casting. In this example, the expected value is 1.25 where it produces output as 1.0

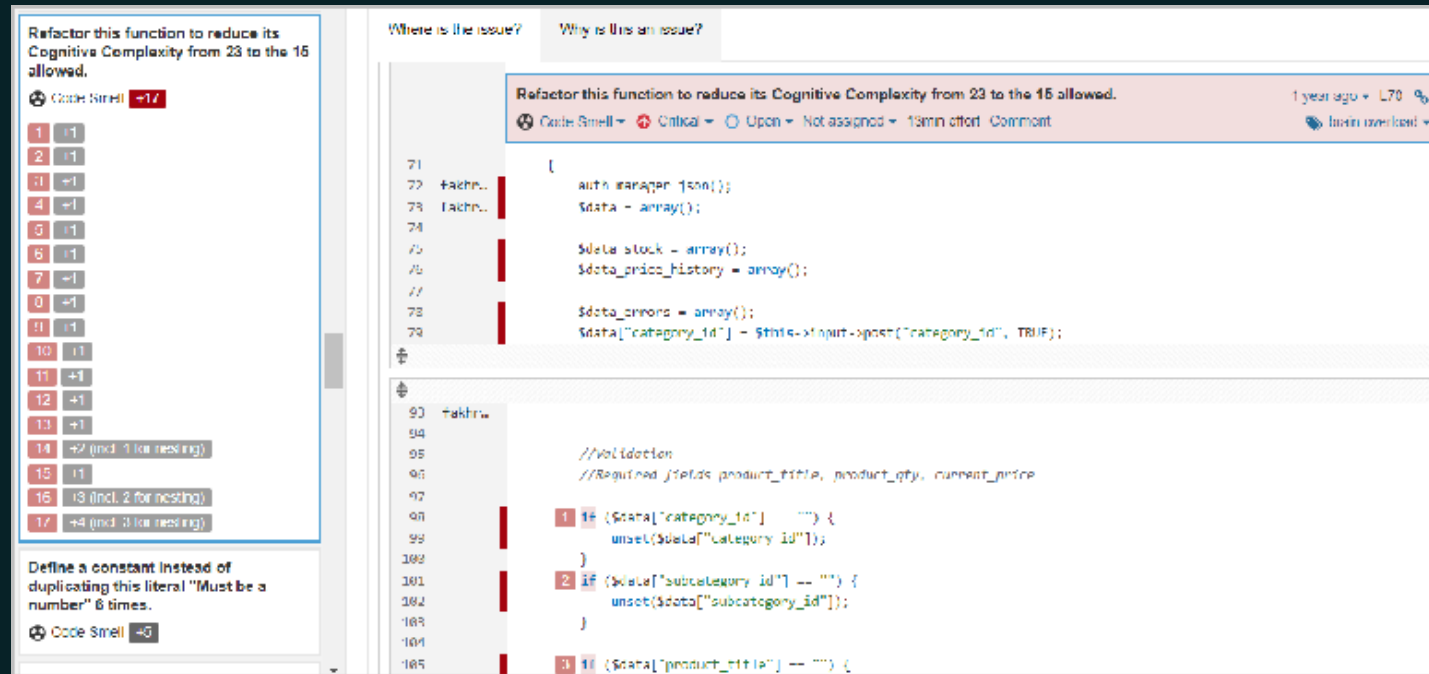
```
56 private double getBestNumber(){
57     String tvDisplayText = tvDisplay.getText().toString();
58     // double n = (double)10/(double)8;
59     double n = 10/8;

    Cast one of the operands of this division operation to a "double". Why is this an issue? 12 mi

    Bug Minor Open Not assigned 5min effort Comment cert, cwe, overlo

60     return n;
```

Pre build analysis issues







Here, a function is defined without adding any codes in function body (line 11 > 13) which is meaningless.

Too many nested condition and loops were used in function body which makes the function too complex to understand.

```
6      public function __construct()  
7      {  
8          parent::__construct();  
9      }  
10  
11      public function save_color(){  
12  
13      }
```

Add a nested comment explaining why this method is empty, throw an Exception or complete the implementation. Why is this an issue?

 Code Smell  Critical  Open  Not assigned 5min effort [Comment](#)

Limitations of SA

- When we can use SA tools, we can found Large amount of false alarms
- Manual triggering results in a time-consuming process
- Significant reliance on programming languages
- Executed according to predetermined rule sets, devoid of intelligent adaptation
- Challenges with Third-Party Libraries
- Lack of Understanding of Business Logic
- Analyzing highly complex code may be challenging for SA tools, leading to potential oversights
- SA lacks awareness of the runtime environment, making it challenging to accurately predict the dynamic behavior of the code

Future Scopes of SA

Difficulties or limitations often open the doors to opportunities, and the realm of static analysis is no exception. Here are some key areas for improvement and potential advancements:

- Focus on refining static analyzers to minimize false alarms and enhance accuracy through algorithm improvements and optimized rule sets
- Explore the use of AI-based techniques to augment static analyzers, improving their adaptability to complex code patterns and enhancing issue detection capabilities
- Envision a future where static analysis includes 'auto fix with high accuracy,' automating issue resolution and providing developers with actionable solutions



THANK YOU

