# Why Don’t Developers Detect Improper Input Validation DROP TABLE Papers;

**PAPER PRESENTATION**

Presentation By Parth Shah, Sankaranarayanan Srinivasan, Taruni Movva

February 20, 2024

# Abstract


- Improper Input Validation (IIV) causes systems to unsafely handle input data, leading to software vulnerabilities.
- Despite being easy to detect and fix, IIV remains a common issue in software development.
- The study investigates developers' ability to detect IIV through an online experiment with 146 participants, 105 of whom have over three years of professional experience.
- A significant number of developers could identify IIV upon being alerted to its existence, improving detection rates. However, 60 participants failed to detect the IIV even after receiving a warning.
- The frequency of conducting code reviews among participants influences the ability to detect IIV, suggesting practice improves detection skills.
- Findings reveal that visible attack scenarios significantly aid in detecting IIV vulnerabilities.
- The study aims to better understand how to support developers in creating secure software systems by identifying key factors affecting IIV detection

# Problem

- The research paper addresses the problem of Improper Input Validation (IIV) vulnerabilities in software systems. Specifically, it investigates why developers frequently fail to detect IIV vulnerabilities.

- Additionally, the paper investigates the reasons developers report for either identifying or failing to identify these vulnerabilities during the code review process.

**Research Questions from paper**

1. Do developers detect Improper Input Validation (IIV) vulnerabilities during code review?

2. What is the effect of warning developers who missed the IIV about the existence of a vulnerability (i.e., prompting) on the detection of an IIV?

# Motivation

- Understanding Developer Efficacy in Detecting Improper Input Validation (IIV)
- Examining the Role of Attack Scenarios :

  SQLI (SQL Injection) and IVQI (Improper Validation of Specified Quantity Input)
- Evaluating Prompting as a Detection Catalyst
- Identifying Barriers to Effective Vulnerability Detection
- Guiding Improvement in Code Review Processes

# Proposed Solution

- Comprehensive approach to understanding developers' ability to detect Improper Input Validation (IIV) vulnerabilities through an **online experiment**. (CR-Experiment tool)

1. Solution involves a code review task and a survey to capture developers' self-reported knowledge and attitudes towards security.

2. Prompting participants about the presence of a vulnerability to assess if this increases detection rates.

3. Study focuses on two types of IIV vulnerabilities: SQL Injection (SQLI) and Improper Validation of Specified Quantity Input (IVQI).

4. Particular interest in understanding the effect of traditional attack scenario visibility and prompting on developers' detection capabilities.

# Design and flow of the online Experiment

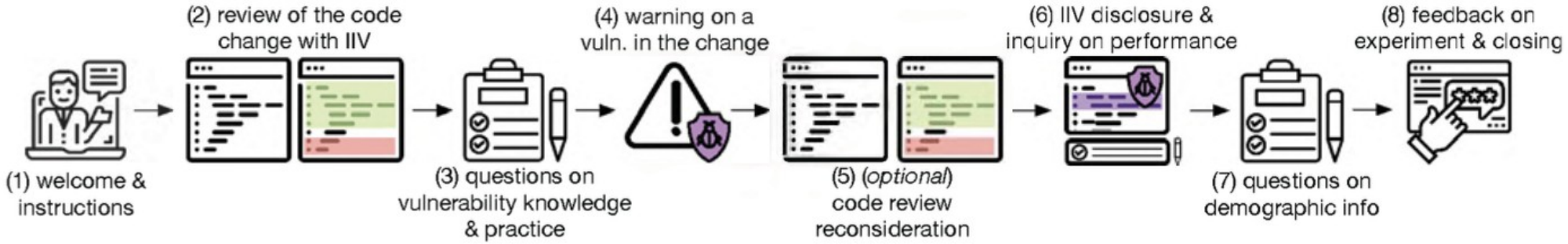


Figure 1. Design and flow of the online experiment.

# Evaluation for Research Question 1 - Detection of IIV Vulnerabilities

**1. Initial Detection of Vulnerabilities:**
Context: RQ1 explores the extent to which developers can detect IIV vulnerabilities during code reviews, focusing on SQLI and IVQI types.

**Findings:**
SQL Injection (SQLI) was detected by 65% of participants assigned to SQLI tasks.
Improper Validation of Specified Quantity Input (IVQI) had only an 18% detection rate.

**2. Logistic Regression Analysis:**
Context: A logistic regression model was used to understand the factors influencing the detection of vulnerabilities.

**Findings:**
The model identified **VulnType** as a significant predictor, showing SQLI is significantly more likely to be detected than IVQI.

# Evaluation for Research Question 1 - Detection of IIV Vulnerabilities

**Odds Ratio:**
Context: The odds ratio quantifies how much more likely one outcome is compared to another.

**Findings:**
An odds ratio of 6.90 indicates SQLI is substantially more likely to be detected compared to IVQI, emphasizing the impact of visible attack scenarios.

Table II
DETECTION OF THE VULNERABILITY IN THE FIRST REVIEW (STEP 2).

| IIV | SQLI | IVQI | Total |
|---|---|---|---|
| Found | 52 | 14 | 66 |
| Not Found | 28 | 52 | 80 |

Odds Ratio: 6.90 (3.27,14.57)
$p < 0.001$

# Evaluation for Research Question 2 - Effect of Prompting on Detection

**Odds Ratio for Review Reconsideration:**
Context: This measure assesses the change in detection likelihood after participants were prompted about a vulnerability.

**Findings:**
The odds ratio post-prompting was 5.57, indicating a significant improvement in detection rates for the reconsidered review, especially for SQLI.

**Logistic Regression Analysis for Reconsidered Review:**
Context: This analysis focuses on data from participants who initially missed the vulnerability but were given a second chance after being prompted.

**Findings:**
The regression confirms **VulnType** significantly affects detection upon reconsideration, with SQLI more likely to be identified than IVQI.

These structured evaluations provide a comprehensive view of the study's findings, elucidating the factors that influence the detection of IIV vulnerabilities during code reviews and the substantial impact of prompts in enhancing this detection.

# Evaluation for Research Question 2 - Effect of Prompting on Detection

**1. Effect of Prompting:**
Context: RQ2 investigates the impact of prompting developers about the existence of a vulnerability on its subsequent detection.

**Findings:**
After prompting, 20 additional participants identified the vulnerability they previously missed, demonstrating the effectiveness of prompts.

**2. McNemar's Test:**
Context: This statistical test was applied to evaluate the change in detection rates before and after prompting.

**Findings:**
The significant p-value (2.152e-05) from the McNemar's test confirms that prompting significantly improves vulnerability detection.

# Evaluation for Research Question 2 - Effect of Prompting on Detection

Table IV
ODDS RATIO FOR DETECTING THE VULNERABILITY IN THE REVIEW RECONSIDERATION (STEP 5).

| IIV | SQLI | IVQI | Total |
|---|---|---|---|
| Found | 13 | 7 | 20 |
| Not Found | 15 | 45 | 60 |

Odds Ratio: 5.57 (1.88,16.55)

$p < 0.001$

# Novelty of the Proposed Solution Considering State-of-Art

- This research uniquely combines the exploration of attack scenario visibility and prompting in the context of IIV detection, offering new insights into the cognitive and process-oriented aspects influencing developers' security practices.
- Utilizes a step-by-step online experiment to replicate the software development environment, specifically the code review process, to study vulnerability detection.
- Hands-on approach allows for the collection of both qualitative and quantitative data related to developers' actions and decisions during code review.
- Provides a nuanced understanding of the factors influencing vulnerability detection.

# Assumption made

- Developers lack basic software security knowledge
- Developers' attitudes towards security may not be adequately proactive.
- The security regulations and understanding of developers are not aligned
- Developers might rely too much on security experts rather than taking initiative themselves.
- The development lifecycle phase, such as the design phase, could influence secure development practices.
- External factors like deadlines, budgets, customer demands, and regulations can hinder secure development efforts.

# Limitation of the proposed solution

**Experiment Setup vs. Real-World Complexity:**

The controlled environment of an online experiment might not capture the full complexity and pressures of a real-world software development setting. This could limit the generalizability of the findings.

**Limited Vulnerability Types:**

The study focuses on specific types of IIV vulnerabilities (SQLI and IVQI). Other vulnerabilities may not be as easily detected with the methods proposed, limiting the scope of the solution.

**Simulated Code Review Process:**

The code review process in the experiment is a simulation and might not fully reflect the collaborative and iterative nature of code reviews in practice.

# Practical Singificance

- The solution increases developers' awareness of security vulnerabilities, particularly SQL Injection and Improper Validation, through hands-on code review tasks, promoting better security practices.
- It demonstrates that alerting developers to the presence of vulnerabilities can significantly boost their ability to detect them, suggesting an effective strategy to improve code security during the review process.
- Collecting data on developers' security knowledge and attitudes provides insights that can inform the creation of targeted educational programs and resources to bridge knowledge gaps.
- The adaptable and scalable design of the study makes it applicable in various development environments, offering a versatile approach for organizations to strengthen their security protocols.
- By revealing the effectiveness of current vulnerability detection strategies and suggesting improvements, the study encourages a culture of continuous learning and enhancement in software security education and practices.

# Discussion

**Attack Scenarios**
1. Applying the study's methodology to other vulnerabilities like XSS or CSRF could broaden understanding of developers' detection capabilities across various security risks.

**Tool Support for Vulnerability Detection**
2. Integrating SAST and DAST tools such as SonarQube and Checkmarx into code review processes can enhance detection of complex vulnerabilities and support developers' security education.

**Impact of Diverse Experience Levels:**
3. The findings are based on a demographic with 3+ years of experience, but the industry includes a wide range of experience levels. how developers with less than 3 years of experience might fare in detecting IIV vulnerabilities and how their fresh perspectives might influence detection rates.

# THANK YOU