Empowering Large Language Models with Static Analysis for Effective Vulnerability Detection

Rafael Ramires*, Ana Respício*, Mike Papadakis†, and Ibéria Medeiros*
*LASIGE, DI, Faculdade de Ciências, Universidade de Lisboa, Portugal
†SnT, University of Luxembourg, Luxembourg
{rframires, alrespicio, ivmedeiros}@fc.ul.pt, michail.papadakis@uni.lu

***Abstract*—Web applications are pervasive and security-critical, making accurate vulnerability detection essential for protecting sensitive data and service integrity. Although static analysis tools (SATs) are widely used for vulnerability detection, they often suffer from high false positive (FP) rates and limited adaptability to evolving code patterns and vulnerability variants. Recently, Large Language Models (LLMs) have emerged as a promising alternative; however, when used in isolation, they often overestimate vulnerabilities, resulting in higher FP rates. In this paper, we propose a *Hybrid Framework* that leverages SAT-extracted data flows and vulnerability metadata to decompose vulnerability detection into smaller, well-scoped reasoning steps for LLMs, thereby improving detection and reducing FPs. Rather than using LLMs to guide or replace static analysis, the proposed framework relies on verified static analysis outputs to bound and refine LLM-based vulnerability reasoning. We evaluate the framework's effectiveness against standalone KAVE SAT and LLMs with different prompt strategies for SQL injection and XSS detection, using both a controlled benchmark and 15 real-world web applications.The results show that the best hybrid solution is composed of *GPT-4o LLM tuned with the role-based prompt strategy empowered by* KAVE *output on code properties and data flow*. The solution discovered 16 zero-day vulnerabilities, and enhances detection, where it reduces the FP rate from 65% to under 16% on average over standalone LLM detection. In real-world applications, it improves the F1-score by 6–12% over the standalone LLM and improves recall by 31% for XSS, while offering a better precision–recall trade-off than the standalone SAT. These results illustrate the framework's practical applicability and indicate that it constitutes a promising direction for advancing vulnerability detection.**

***Index Terms*—Web Application Vulnerabilities, Static Analysis Tools, Large Language Models, Software Security**

I. Introduction

The rapid evolution of web technologies has created unprecedented opportunities for innovation while simultaneously exposing applications to a broad range of security vulnerabilities [1], [2], as confirmed by their growing number each passing year [3], [4]. In particular, vulnerabilities have been more frequent in PHP web applications, the most used language for building the server-side of such applications [5], due to their widespread use, accessibility, and reliance on dynamic user inputs. Injection vulnerabilities have been ranked among the top risks in the OWASP Top 10 for several years and have been detected in up to 94% of analysed applications [6], [7]. Among this category, SQL injection (SQLi) and Cross-site scripting (XSS) vulnerabilities have been revealed as the most prevalent and critical in PHP web applications, and the easiest to reproduce and exploit [6]. Together, these two classes of vulnerabilities have accounted for at least 33% of the vulnerabilities discovered each year, with a growing trend over time, and with XSS being the most frequently reported vulnerability of all [3]. This undesired trend makes robust detection frameworks an urgent priority, with effective vulnerability detection and mitigation being critical to protect sensitive user data and preserve the integrity of applications.

Despite their widespread adoption for software security testing, static analysis tools (SATs) [8], [9], [10] face significant challenges in real-world applications, namely the difficulty of adapting to emerging code patterns, novel vulnerabilities without frequent updates or even the manner in which they analyse the application code to discover vulnerabilities. These limitations result in high false positive (FP) rates, which often overwhelm developers with unnecessary alerts, leading to inefficiencies in vulnerability remediation workflows. This scenario is particularly pronounced in PHP applications, where dynamic typing and flexible coding practices introduce complexities that SATs often fail to address comprehensively, resulting in over-approximations and consequently high FP rates. Because of this, we focus our evaluation on PHP not only because of its dominance in the server-side market [5], but also as a rigorous benchmark for hybrid analysis effectiveness. In contrast, large language models (LLMs) [11], [12], [13], [14] exhibit exceptional natural language processing capabilities and context-sensitive reasoning and have been applied to vulnerability detection. However, LLMs still do not improve upon SATs, often overestimating risks, providing limited precision, or even producing FPs, despite their ability to process software and identify specific code properties. Recent experiments have demonstrated that they can better solve complex logical problems when these are decomposed into multiple simpler ones (or asked in multiple steps) [15].

Considering the capabilities of LLMs and the information that SATs produce, we hypothesise that *(i) code properties and data flows provided by SATs can decompose the vulnerability detection problem into smaller subproblems; (ii) the LLM's advanced contextual reasoning to find complex vulnerabilities can improve detection, thus reducing FNs; and (iii) the SAT's verified boundaries prevent the LLM from hallucinating, i.e., generating FPs.*

In this paper, we present the methodology employed to pursue our hypothesis and the results we obtained. We explore

hybrid approaches that integrate LLMs with traditional SAT tools by decomposing the vulnerability detection problem into subproblems that LLMs can reason about effectively. We thus merge detailed outputs from static analysis, such as taint data flow information and identification of critical code elements (e.g., entry points, sensitive sinks), with the contextual reasoning capabilities of LLMs to investigate whether we can improve detection precision and ultimately propose a scalable, effective, and robust solution for web application vulnerability detection. Unlike LLM-assisted static analysis approaches [16], [11], the methodology we propose does not use LLMs to guide or replace static analysis; instead, it investigates how static analysis results can be leveraged to constrain and improve LLM-based vulnerability reasoning.

In the methodology, and to achieve such an integrated and hybrid solution, the effectiveness of standalone LLMs and SAT tools is examined and compared with that of our hybrid solution. To do so, two evaluation stages are performed.

In a first stage, after buildind a dataset for validation purposes, we assessed and validated the methodology, including the combination of four LLM models (GPT-4o-mini [17], GPT-4o [18], Gemini 2.0 Flash [19], and DeepSeek-V3 [20]) with six prompts (baseline, contextual, few-shot, role-based, chain-of-thought, and a combination of the previous), and measured these combinations with a set of metrics (e.g., recall, precision, F1-score) on a well-established dataset of PHP code snippets with SQLi and XSS vulnerabilities. Comparative experiments across multiple combinations of LLM prompts revealed that more advanced LLMs offer higher recall rates and naturally evolve their detection capabilities with improvements in the underlying architectures; with the *GPT-4o tuned with role-based prompting* ($\texttt{GPT-4o}_{\texttt{rbp}}$ for short) emerging as the most effective combination. Simultaneously, we used the same dataset to assess the effectiveness of KAVE, a modern SAT tool for detecting PHP vulnerabilities, whose performance has outperformed other SATs [21], [22]. Clearly, KAVE revealed a lower FP rate than $\texttt{GPT-4o}_{\texttt{rbp}}$, denoting its capacity for data flow analysis considering the propagation of taintdness. In contrast, $\texttt{GPT-4o}_{\texttt{rbp}}$ presented much fewer FNs compared to KAVE, justified by its ability to find code properties and role-based reasoning.

In a subsequent phase, information from KAVE was incorporated into LLMs tuned to the role-based prompt, enabling notable enhancements and a substantial reduction in the FP rate. The best hybrid approach was $\texttt{GPT-4o}_{\texttt{rbp}}$ with KAVE ($\texttt{GPT-4o}_{\texttt{rbp}}$+KAVE), achieving an F1-score of 83% and a FP rate of 15%. $\texttt{GPT-4o}_{\texttt{rbp}}$+KAVE compared to the $\texttt{GPT-4o}_{\texttt{rbp}}$ baseline significantly improved it ($\sim$70% of F1-score and $\sim$64% of FP rate). Furthermore, $\texttt{GPT-4o}_{\texttt{rbp}}$+KAVE compared to KAVE standalone slightly increased the FP rate while drastically decreasing the FN rate. Such comparative results lead us to propose a final *Hybrid Framework* composed of $\texttt{GPT-4o}_{\texttt{rbp}}$+KAVE to improve precision and recall, supported by the results of the static analysis tool KAVE.

Lastly, we tested this final hybrid framework with a controlled benchmark of code snippets and 15 real-world web applications of varying size and complexity. The framework discovered 16 Zero-days, i.e., previously unregistered vulnerabilities. In addition, the results confirmed previously observed trends: standalone LLMs achieved high recall but at the cost of many FP, while KAVE alone reduced FP but missed a significant number of vulnerabilities, resulting in numerous FN. In contrast, the hybrid $\texttt{GPT-4o}_{\texttt{rbp}}$+KAVE framework achieved a stronger precision–recall trade-off than the standalone LLM, lowering FP in secure projects and recovering missed vulnerable cases. Overall, the framework improved over the standalone LLM in both precision and recall, although the standalone SAT still attained a higher F1-score on these real-world applications.

Both findings demonstrate that the framework can generalise beyond controlled datasets to realistic codebases, though its effectiveness remains bounded by the context limitations of LLMs when applied to large applications. Also, they indicate that the proposed hybrid framework provides a practical and effective approach for improving vulnerability detection precision in web applications. Furthermore, this work contributes to broader efforts in secure software development practices and opens the way for extending this hybrid approach to other languages, vulnerabilities, and contexts.

The main contributions of this work are threefold: (1) A static-analysis–grounded framework for LLM-based vulnerability detection in PHP applications, in which verified dataflow information extracted by the KAVE static analysis tool is integrated into LLM reasoning to reduce FPs while improving recall and precision. (2) A controlled empirical study comparing standalone SAT tools, standalone LLM-based detection, and their hybrid integration, using multiple security metrics to quantify FPs, recall, and overall detection effectiveness. (3) An evaluation on a balanced dataset of PHP code snippets covering SQLi and XSS vulnerabilities, complemented by experiments on real-world PHP web applications, demonstrating the applicability of the approach beyond synthetic benchmarks.

II. Background & Related Work

The detection of vulnerabilities in web applications has been a longstanding challenge in software security. This section reviews the most relevant research on static analysis and LLMs for vulnerability detection.

A. Web Application Vulnerabilities

Web applications remain prime targets for cyberattacks due to their exposure to untrusted inputs, with SQLi and XSS vulnerabilities being ranked as critical due to their prevalence and severity of exploitation [6], [23].

An SQLi [24], [25] occurs when an application incorporates unsanitised or unvalidated external data, such as user inputs, into queries and executes them in databases. For instance, the PHP code in Listing 1 receives the username from the `$_POST` entry point, inserts it into an SQL query without validating or sanitising it, and then sends the query to be executed into the database through the sensitive sink `mysqli_query`. An attacker can exploit this by submitting specially crafted input

as `' OR '1'='1`, which transforms the query into `SELECT * FROM users WHERE user = '' OR '1'='1'` and thus returns all user records when executed in the database.

```
1 <?php
2 $u = $_POST['username'];
3 $q = "SELECT * FROM users WHERE user = '.$u."'";
4 $r = mysqli_query($con, $query);
5 ?>
```

Listing 1. Example of a PHP script vulnerable to SQLi.

An XSS vulnerability [26], [27] arises when an application embeds user-controlled scripts into server-side output functions without proper encoding or sanitisation. For instance, the instruction `echo $_POST['content']` contains an XSS, in which an attacker can supply a JavaScript, such as `<script>stealCookies();</ script>`, that is executed in the victim's browser.

B. Static Code Analysis

Static Analysis Tools (SATs) inspect the code without requiring its execution by examining its syntax, control flow, and data flow to surface potential vulnerabilities early in the software development life cycle [28], [29]. Traditional SAT tools are based on code property graphs (CPGs) [30], [31], [32] that represent the code of applications, such as the abstract syntax tree (AST) [8], [33], [34], control flow graph (CFG) and program dependency graph (PDG) [35], [36], [37], [30]. These tools typically employ taint analysis [38], [33], [39], [40], [41] over these structures, traversing them and propagating the taintedness of input data to search for vulnerabilities.

Despite their utility, traditional SAT tools often suffer from high FP rates [42], [43], [44] because their reliance on rigid static rule sets and the way they are programmed to traverse such CPGs forces them to over-approximate data flows to avoid missing vulnerabilities. Moreover, their reliance on static rule sets limits them in adaptability to new vulnerability patterns [45]. These limitations have motivated the exploration of alternative and complementary approaches. KAVE [21] is one of these, which integrates the above code graphs into a multi-layer knowledge graph and extends it with semantic and security property layers that track taint flows and known vulnerability patterns using a multi-agent system. Other tools combine context-aware analysis and adaptive heuristics [16], [46], static analysis with machine-learning [47], [8], and others employ pruning of such code graphs to discard irrelevant code paths prior to in-depth analysis, reducing FPs [21], [48], [49].

C. Large Language Models (LLMs) in Security

The advent of Large Language Models (LLMs) has introduced new possibilities for automated vulnerability detection by combining natural language understanding with code analysis. Modern LLMs like GPT-4 and Gemini leverage transformer-based architectures with self-attention mechanisms, enabling holistic analysis of code semantics and data flows. Unlike SATs that rely on predefined rules, LLMs learn vulnerability patterns from vast corpora of code and security advisories. Transformer architectures process code as sequences of tokens, capturing long-range dependencies through multi-head attention layers. This allows models to contextualise vulnerabilities across function boundaries and nested control structures [11], [12], [50]. The pretraining phase typically involves masked language modelling objectives in code repositories, enabling LLMs to infer implicit program logic. Subsequent fine-tuning on vulnerability datasets (e.g., CVE descriptions, exploit proofs-of-concept) adapts these models for security-specific reasoning tasks [11], [51].

Recent research has applied LLMs as static code analysers, using them as classifiers, and employing prompt engineering to identify insecure code patterns. This enhances LLMs' contextual understanding, allowing them to detect vulnerabilities missed by traditional SATs [16], [11]. Regardless of their suboptimal efficacy in processing complex applications, LLMs offer three main advantages over traditional SAT tools: *(i) Contextual adaptation:* LLMs excel at recognising novel vulnerability patterns through in-context learning. For example, they can identify insecure deserialisation vulnerabilities in PHP code by correlating object injection patterns with known exploit techniques, even under conditions where sanitisation functions are obfuscated [12], [51], which contrasts with rule-based tools that require explicit signature updates for new vulnerabilities; *(ii) Cross-language reasoning:* Unified tokenisation strategies enable LLMs to detect vulnerabilities in polyglot codebases: a model trained on SQLi can recognise injection risks in both PHP's `mysqli_query` and Python's SQLAlchemy ORM patterns, despite syntactic differences [11], [52]; and *(iii) Enhanced exploit chain analysis:* LLMs can mimic attacker reasoning by tracing multi-step vulnerability exploitation paths. For instance, when analysing XSS vulnerabilities, LLMs can infer whether unsanitised `$_GET` parameters propagate through template engines or third-party libraries before reaching sensitive sinks like `echo` [53], [12].

LLMs have also been adapted for advanced security tasks beyond basic vulnerability detection. Models like Codex-Sec generate context-aware SQLi fixes, suggesting parameterised queries or ORM migrations based on database schemas [11], [52]. LLMs link CVE descriptions with codebases to prioritise vulnerabilities. For example, GPT-4 was able to map a high percentage of Log4j-related CVEs to vulnerable Java dependencies in a financial services codebase [53]. By analysing API documentation and authentication flows, LLMs predict potential privilege escalation paths in microservices, reducing overlooked endpoints in AWS Lambda deployments [52], [51].

Although LLMs can augment SATs, they cannot completely replace them. Instead, integrating LLM with learning-based techniques [54], [55], adversarial testing [11], [12], or using LLM-generated outputs to enhance SAT results [16] offers a more effective approach.

Recent work such as IRIS [16] proposes LLM-assisted static analysis for Java applications, where the LLM acts as the primary analysis engine to hypothesise potential data flows. Unlike our approach, IRIS does not integrate an existing SAT to verify these flows, focusing instead on an empirical assessment of LLM-guided coverage expansion (Recall). This

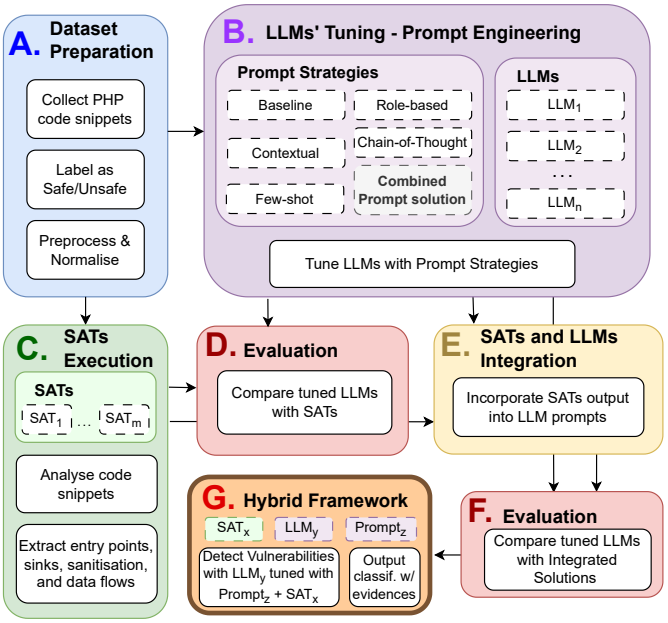

Fig. 1. Methodology overview for a Hybrid Framework

represents a fundamental inversion of our methodology. While IRIS utilises the LLM to *expand* the search space at the cost of potential hallucinations, our framework uses a dedicated SAT as the *ground truth* to *constrain* LLM reasoning, explicitly optimising for **Precision** (FP reduction). Consequently, these approaches operate on orthogonal optimisation objectives and distinct technology stacks (Java vs. PHP), rendering a direct performance comparison methodologically unsound.

III. Methodology Overview

Our primary objective is to develop a *Hybrid Framework* for detecting web vulnerabilities by enhancing LLMs' capabilities with SATs' results. In this framework, SATs augment the reasoning capabilities of LLMs by providing static analysis insights into vulnerabilities and associated data flows. To achieve this goal, we established a methodology to systematically examine the respective strengths and limitations of LLMs and SATs in vulnerability detection. To ensure generalizability, we define an abstract interface for the Static Analysis component, in which the SAT is required to output structured Taint Flows in the tuple format $T =< Entry, Sink, Sanitization_Status, Location >$. While commercial tools (e.g., CodeQL, Checkmarx) possess this data internally, they often require complex custom queries to export it in a structured format usable by LLMs. For this study, we instantiate this component using KAVE [22] because it natively serialises these verified data flow paths, allowing us to focus on the integration methodology rather than on building export adapters for proprietary tools. Then, the methodology explores how these strengths can be synergistically leveraged to mitigate their limitations, thereby building a hybrid framework to support effective vulnerability detection systems.

Fig. 1 illustrates the methodology overview. It begins with the composition and preparation of the data set (**A.**), which will be used to assess standalone LLMs and SATs, and the framework obtained. Using the dataset, **B.** and **C.** are performed to evaluate the current capabilities of SATs and LLMs exercised with different prompt strategies. Both SATs and LLMs are executed independently, followed by a comparative evaluation of their effectiveness (**D.**). This comparative analysis aims to uncover the strengths and limitations of LLMs, particularly in scenarios where SATs have traditionally excelled but have also faced challenges such as high false positive rates (FPR) or limited adaptability to new patterns. Afterwards, it explores an innovative hybrid approach that takes advantage of the complementary strengths of LLMs and SATs (**E.**). Specifically, it investigates how insights from static analysis, such as detailed taint flow information, identification of sensitive sinks, and entry points, can be used to empower and guide LLMs in their reasoning processes. Finally, it evaluates and compares the detection effectiveness of the hybrid solution against standalone LLMs. (**F.**), thus confirming that the hybrid prevails and obtaining the hybrid framework comprising the best combination (**G.**).

Throughout the proposed methodology for evaluating the effectiveness of LLMs, SAT, and the framework across diverse tests (**D.** and **F.**), we use standard metrics derived from the confusion matrix, including false positives (FP), true positives (TP), false negatives (FN), and true negatives (TN). These metrics, presented in Table I, provide a comprehensive understanding of the ability of the models and framework to detect vulnerabilities accurately. Through the analysis of FP and FN, we seek to elicit the trade-offs between precision and recall associated with each approach.

IV. Methodology Instantiation for a Hybrid Framework

This section details the methodology instantiation, including the choices we made regarding the dataset, SATs, LLMs and prompts, in order to construct an effective hybrid framework for vulnerability detection of PHP web applications. The dataset and processing framework scripts are available at [56].

A. Dataset Characterisation and Preparation

To pursue our methodology and experimental evaluation in detecting vulnerabilities by the models and combinations on different scenarios, including the effectiveness of our approach, we built a dataset comprising 2,000 PHP code snippets sourced from the SARD [57] public repository, and others handcrafted designed to reflect real-world scenarios. The dataset was evenly divided into SQLi (CWE-79 [58]) and XSS (CWE-89 [59]) vulnerability classes. For each class, 1,000 cases were collected, with half labelled as *Unsafe* and the other half as *Safe*, i.e., vulnerable and not vulnerable. This balanced distribution ensures the dataset is suitable for evaluating binary classification tasks and vulnerability detection models. Unsafe cases include code patterns with exploitable vulnerabilities, such as unsanitised user inputs being passed

TABLE I
METRICS USED TO EVALUATE THE SOLUTIONS

Metric	Formula
Accuracy (Acc): Overall correctness of the model's predictions	Acc = (TP+TN) / (TP+TN+FP+FN)
Precision (Prc): Proportion of correctly identified vulnerabilities among all positive predictions.	Prc = TP / (TP+FP)
Recall (Rcl): Proportion of actual vulnerabilities that were correctly identified.	Rcl = TP / (TP+FN)
Specificity (Spc): Proportion of safe code snippets correctly identified as safe.	Spc = TN / (TN+FP)
False Positive Rate (FPR): Proportion of safe code snippets incorrectly classified as vulnerable.	FPR = 1 - Specificity
F1-Score (F1): Harmonic mean of precision and recall, providing a balanced effectiveness measure.	F1 = 2 * (Precision * Recall) / (Precision + Recall)

to sensitive sinks (e.g., `echo` for XSS or `mysqli_query` for SQLi). In contrast, safe examples contained proper input validation or sanitisation mechanisms (e.g., `htmlentities` for XSS or prepared statements for SQLi).

Before feeding the dataset into the models and SATs, we applied a series of preprocessing steps to clean and normalise the code. This includes removing extraneous comments (with a special focus on unsafe or safe labelling and explanations), while preserving the logic related to vulnerabilities and normalisation by standardising variable names and formatting to reduce noise that could affect model effectiveness. These preprocessing steps remove noise while preserving important structural and semantic information essential for vulnerability detection. In addition, normalising the dataset helps models to focus on identifying vulnerability patterns without being influenced by coding style or naming variations. In addition, this reduces data leakage risk [60], where test set data unintentionally impacts training, causing overly optimistic effectiveness estimates.

B. Large Language Models & Prompt Strategies

1) Large Language Models:

With the goal of understanding which LLM performs better with prompt strategies, we opted for using four LLM models: GPT-4o-mini [17], GPT-4o [18], Gemini 2.0 Flash [19], and DeepSeek-V3 [20].

2) LLMs' Prompt Strategies:

We defined six prompt strategies with the goal of evaluating how different prompt structures and levels of guidance influence the LLM model's ability to identify vulnerabilities accurately and provide actionable insights. Subsequently, we present the six prompt strategies, including an XSS example for those deemed most pertinent.

Baseline. Designed to provide minimal contextual information, allowing for an assessment of the LLM's general understanding of vulnerabilities without additional guidance. Their purpose is to evaluate whether the model can identify vulnerabilities based solely on its pre-trained knowledge. So, they serve as a reference point for assessing the model's accuracy and completeness in detecting vulnerabilities without explicit prompting.

```
Prompt:
Respond with only 'Safe' or 'Unsafe' based on
whether this PHP code has XSS vulnerabilities:
{code_snippet}

Code Snippet example:
<?php echo $_GET['name']; ?>
```

Contextual. These prompts include additional information about vulnerability types and ask for specific details, such as identifying vulnerable lines or considering fixes. Their purpose is to test whether providing additional context improves detection accuracy.

Few-Shot. Few-shot prompts include examples of both vulnerable and secure code snippets to guide the model's reasoning process. This approach aims to test whether providing examples improves the model's ability to generalise and reduces the occurrence of false positives and false negatives.

```
Prompt:
Here are examples of vulnerable and secure PHP code
snippets for XSS.

Example 1 (Vulnerable):
<?php echo $_GET['name']; ?>

{other_examples}

Example 2 (Secure):
<?php echo htmlspecialchars($_GET['name'],
    ENT_QUOTES, 'UTF-8'); ?>

{other_examples}

Analyze the following code and determine if it is
vulnerable or not.
Code to analyze:
{code_snippet}
```

Role-based. The LLM is prompted to adopt specific professional roles, such as a security auditor or a penetration tester, to assess whether role-playing improves its ability to meet task-specific requirements.

```
Prompt:
You are a Penetration Tester specialising in
identifying Cross-Site Scripting (XSS)
vulnerabilities. Your task is to analyse PHP
code for XSS vulnerabilities.
- Check if user input is sanitised or escaped before
  being outputted to HTML.
- Respond ONLY with 'Safe' if there are no
  vulnerabilities or 'Unsafe' if there are any
  vulnerabilities.

PHP Code:
{code_snippet}
```

Chain-of-Thought. These prompts explicitly guide the LLM through step-by-step reasoning processes. This approach aims to improve detection accuracy by encouraging systematic reasoning and reducing false positives or negatives.

```
Prompt:
```

```
Analyse this PHP code for Cross-Site Scripting (XSS)
vulnerabilities step by step:
1. Identify where user input is accepted.
2. Determine if user input is sanitised or escaped
   before being outputted.
3. Check how user input is used in HTML output.
4. Based on your analysis, respond with ONLY 'Safe'
   if there are no vulnerabilities or 'Unsafe' if
   there are any vulnerabilities.

PHP Code:
{code_snippet}
```

Combined Solution. In an attempt to achieve an accurate solution for vulnerability detection in PHP code, we formulated a combined prompt approach that integrates the most effective strategies identified in earlier prompt-engineering iterations. This solution integrates contextualised role-based prompting, few-shot learning examples, and chain-of-thought reasoning while prioritising concise and interpretable output.

C. Static Analysis Tools

Among different SATs for vulnerability detection in PHP code [8], [61], [62], [39], [34], [22], we chose the KAVE tool [22], [21], [63] since it has demonstrated better results in real-world web application security analysis and outperformed existing SAT tools [21].

Unlike conventional SAT tools, KAVE constructs a multi-layer knowledge graph (MLKG) that integrates syntactic, semantic and security features of the code and employs a knowledge-based multi-agent system to traverse MLKG for discovering web vulnerabilities. This novel and advanced approach enables KAVE to have a better understanding of the code properties and their dependencies, thereby achieving high precision in vulnerability detection.

KAVE outputs structured representations in the form of code property graphs and detailed vulnerability metadata, including the identification and location in the code of entry points, sensitive sinks, and sanitisation functions. Listing 2 illustrates a brief output of the tool considering the PHP code of Listing 1 (the code property graph is not presented for reasons of space). However, whenever KAVE reports a specific `<entry,sink>` pair (in the listing, the pair `<line 2, line 4>`), it has already verified that a valid data and control flow path connects them. Thus, these tagged and verified pairs serve as concise summaries of taint data flow paths.

```
Entry point: $_POST['username'] (line 2)
Sensitive sink: mysqli_query (line 4)
Sanitization function: None detected
Data flow pairs: <line 2, line 4>
Vulnerabilities detected: 1
```

Listing 2. KAVE output when inspects the code of Listing 1.

D. Hybrid Framework

We propose a hybrid framework that combines the strengths of SAT tools and LLMs, aiming to reduce the FPR and thereby improve the vulnerability detection effectiveness. Our intuition is that the KAVE's outputs are particularly valuable for integration with LLMs, as they provide rich, context-aware information that can be directly incorporated into LLM prompts or used as auxiliary input. Using KAVE's code security properties (e.g., entry points and sinks) and their location in the code, LLMs gain access to a comprehensive view of the application's data and control flows, enabling more informed and precise reasoning about potential security issues. Crucially, KAVE's identification of specific `<entry,sink>` pairs allows our framework to focus on the relevant code slices (e.g., the function containing the sink and its immediate scope) for the LLM. This bypasses the context window limitations typically faced when feeding entire files to LLMs, making the solution scalable to large applications. This synergy enhances the LLM's ability to detect and explain vulnerabilities and also supports advanced use cases such as automated remediation and explainable diagnostics. Therefore, the framework is composed of KAVE SAT and the best LLM-prompt pair, which will work in the following manner.

Static Analysis with KAVE: The PHP code is inspected by KAVE to extract detailed information about potential vulnerabilities, like the one depicted in Listing 2. The `<entry,sink>` pairs represent concise taint data flow paths. This information provides a structured overview of how data is connected in the code and highlights areas requiring further inspection. Therefore, it will guide LLMs to focus their attention on paths where untrusted input reaches a sink.

Guiding the LLM: The outputs from KAVE are incorporated into the LLM prompts to guide its analysis. Taking the role-based prompt as example, KAVE output can be integrated as exemplified in Listing 3:

```
You are a Security Auditor specializing in web
application vulnerabilities. Your task is to analyze
PHP code for SQL Injection vulnerabilities.
- Check if user input is sanitized or validated
  before being used in database queries.
- Use the following information from an external
  tool to guide your analysis:
[--- SAT output ---]
  { SAT output from Listing 2 }
[------------------]

- Respond ONLY with 'Safe' if there are no
  vulnerabilities or 'Unsafe' if there are any
  vulnerabilities.

PHP Code:
{ Code from Listing 1 }
```

Listing 3. Example of the integration of KAVE output in a role-based prompt.

Output Generation: The final output combined insights from both tools. The LLM provides binary classifications (*Safe* or *Unsafe*), while KAVE supports classification evidence.

V. EVALUATION

The main question we want to answer is: *Can a hybrid framework, integrating* KAVE*'s vulnerability information and data analysis on LLMs' prompt and reasoning processing, be more effective than these standalone techniques?*

TABLE II
CONFUSION MATRIX AND METRICS OF PROMPT ENGINEERING STRATEGIES EVALUATED WITH LLM'S MODEL `GPT-4O-MINI`

Prompt Strategy	SQL Injection (SQLi)										Cross-Site Scripting (XSS)									
	Conf. Matrix				Metrics (%)						Conf. Matrix				Metrics (%)					
	FP	TP	FN	TN	Acc	Prc	Rcl	Spc	FPR	F1	FP	TP	FN	TN	Acc	Prc	Rcl	Spc	FPR	F1
Baseline	481	454	46	19	47	49	91	4	96	63	454	452	48	46	50	50	90	9	91	64
Contextual	486	500	0	14	51	51	100	3	97	67	430	458	42	70	53	52	92	13	86	66
Few-Shot	493	500	0	7	51	50	100	1	99	67	429	457	43	71	53	52	91	13	86	66
Role-Based	465	500	0	35	54	52	100	7	**93**	**68**	428	458	42	72	53	52	92	14	**86**	**66**
Chain-of-Thought	498	500	0	2	50	50	100	0.4	100	67	476	468	32	24	49	50	94	5	95	65
Combined Solution	496	500	0	4	50	50	100	0.8	99	67	460	463	37	40	50	50	93	8	92	65

To answer this question, we performed the experimental procedure explained next, for the detection of SQLi and XSS vulnerabilities in PHP code using the methodology and instantiation presented in Sections III and IV over the dataset built in Section IV and a set of real web applications, and assessed by the metrics of Section III.

A. Experimental Procedure

The experimental procedure involves the following steps:

1) Determine the prompt strategy that provides the best results (Section V-B).
2) Determine the most accurate LLM by employing the prompt strategy of (1) (Section V-C).
3) Identify strengths and weaknesses of KAVE tool and the best LLM-prompt of (2) (Section V-D).
4) Determine which hybrid solution detects vulnerabilities more effectively, considering LLMs tuned with the prompt strategy from (1) and KAVE (Section V-E), which will consequently constitute the hybrid framework.
5) Evaluate how the hybrid framework from (4) performs with real-world applications containing previously discovered vulnerabilities (Section V-E).
6) Assess the framework's behaviour in realistic deployment scenarios through real-world applications and an additional zero-day discovery study (Section V-F).

We explicitly exclude LLM-assisted exploration tools (e.g., IRIS [16]) from the quantitative baseline. As discussed in Section II-C, such tools are designed to generate hypothetical data flows to boost recall in incomplete code, whereas our objective is to filter false positives using verified data flows. Comparing a hallucination-prone exploration tool against a precision-focused filtering framework would yield mismatched metrics that do not accurately reflect the distinct utility of either approach.

B. Prompt Engineering Comparison

The first stage of the evaluation focused on comparing different prompt engineering strategies applied to `GPT-4o-mini`, to identify the most effective strategy. We opted to use only that LLM, as it is the least evolved in terms of its ability to process and gain a comprehensive understanding of the code. Table II summarises this evaluation. It is noticeable that `GPT-4o-mini` has a strong tendency to label code snippets as *Unsafe*, resulting in a high number of false positives (FP) and making it unreliable.

The role-based prompting strategy consistently outperformed other prompts, achieving the highest F1-scores for both SQLi (68%) and XSS (66%), and particularly with a precision better than the others. This improvement can be attributed to its ability to explicitly frame tasks by assigning roles such as Security Auditor or Penetration Tester, which likely helped to guide the LLM's reasoning process more effectively. Curiously, the Combined Solution strategy did not yield significant improvements over role-based prompting alone and, in some cases, performed slightly worse, possibly due to the added complexity or redundancy in guidance. We attribute this to a form of prompt overload: combining role-play, few-shot examples, and chain-of-thought reasoning into a single dense prompt dilutes the model's attention and can introduce conflicting instructions, leading to less consistent verdicts than the focused role-based prompt.

The relatively modest effectiveness observed across all prompting strategies further reinforces the integrity of our evaluation, with false positive rates (FPR) remaining above 86%, the models clearly did not capitalise on memorised examples or spurious correlations within the dataset. This poor result gives us confidence in stating that if any inadvertent data leakage or deliberate poisoning had occurred, we would expect anomalously high accuracy or unusually low error rates. Instead, these restrained results underscore that the LLMs' behaviour was driven by genuine reasoning over the inputs rather than by prior exposure to test-set artifacts.

Chosen prompt strategy: *Role-based prompt* (`rbp`) is the chosen for the following tests to compare different LLMs' effectiveness. We note that prompt effectiveness can be model-dependent; the role-based prompt was selected on `GPT-4o-mini` as a cost-effective proxy, and we acknowledge that a different prompt could be preferable for other models.

C. LLMs using Role-Based Prompts

We tested the four standalone LLM models using the role-based prompting strategy, aiming to understand which performs better in the vulnerability detection task.

Table III, rows 4 to 7, summarises the metrics for each model. The results indicate that the more advanced models generally performed better, with `GPT-4o` achieving the highest overall scores across SQLi and XSS detection tasks – a F1-score between 71% and 73% in both vulnerability classes. The higher tendency to label the code as *Unsafe* is still noticeable (i.e., FPR rates) but less significant in the most advanced models (`GPT-4o` and `Gemini-2.0-Flash`). Yet, `GPT-4o` had lower FPR rates than `Gemini-2.0-Flash`, reaching 72.6% and 57% for SQLi and XSS. This suggests that enhancements in the `GPT-4o` model architecture and the breadth of training data directly contribute to improved vulnerability detection effectiveness.

Final LLM model: *GPT-4o tuned with role-based prompt* ($\texttt{GPT-4o}_{\texttt{rbp}}$) was the best LLM-prompt combination.

D. Comparison of KAVE with LLM $\textit{GPT-4o}_{rbp}$

In this third experiment, we evaluated the effectiveness of KAVE and compared its results with the best LLM model

TABLE III
CONFUSION MATRIX AND METRICS OF STANDALONE LLMS USING ROLE-BASED PROMPT, KAVE, AND HYBRID FRAMEWORK EMPOWERING LLMS WITH KAVE AND TUNED WITH ROLE-BASED PROMPT.

	Model / SAT	SQL Injection (SQLi)										Cross-Site Scripting (XSS)									
		Confusion Matrix				Metrics (%)						Confusion Matrix				Metrics (%)					
		FP	TP	FN	TN	Acc	Prc	Rcl	Spc	FPR	F1	FP	TP	FN	TN	Acc	Prc	Rcl	Spc	FPR	F1
LLMs	GPT-4o-mini	465	500	0	35	53.5	51.8	100.0	7.0	93.0	68.3	428	458	42	72	53.0	51.7	91.6	14.4	85.6	66.1
	GPT-4o	363	500	0	137	63.7	57.9	**100.0**	27.4	**72.6**	**73.4**	285	438	62	215	65.0	60.6	**87.6**	43.0	**57.0**	**71.6**
	Gemini-2.0-Flash	371	500	0	129	62.9	57.4	100.0	25.8	74.2	72.9	370	463	37	130	59.0	55.6	92.6	26.0	74.0	69.5
	DeepSeek V3	496	500	0	4	50.4	50.2	100.0	0.8	99.2	66.8	392	464	36	108	57.0	54.2	92.8	21.6	78.4	68.4
SAT	**KAVe**	67	103	397	433	53.6	60.6	**20.6**	86.6	**13.4**	**30.7**	50	78	422	450	53.0	60.9	**15.6**	90.0	**10.0**	**24.8**
LLMs w/ KAVE	GPT-4o-mini + KAVE	352	485	15	148	63.3	57.9	97.0	29.6	70.4	72.6	244	436	64	256	69.0	64.1	87.2	51.2	48.8	73.9
	GPT-4o + KAVE	83	436	64	417	85.3	84.0	**82.7**	83.4	**16.6**	**85.6**	73	400	100	427	83.0	84.6	**80.0**	85.4	**14.6**	**82.2**
	Gemini 2.0 Flash + KAVE	146	460	40	354	81.4	75.9	92.0	70.8	29.2	83.2	138	414	86	362	78.0	75.0	82.8	72.4	27.6	78.7
	DeepSeek V3 + KAVE	206	499	1	294	79.3	70.8	99.8	58.8	41.2	82.8	167	435	65	333	77.0	72.3	87.0	66.6	33.4	78.9
	GPT-4o + KAVE (unlabelled)	178	500	0	322	82.2	73.7	**100.0**	64.4	**35.6**	**84.9**	201	467	33	299	76.6	69.9	**93.4**	59.8	**40.2**	**80.0**

using role-based prompts – $\texttt{GPT-4o}_{\texttt{rbp}}$ – in order to identify their strengths and weaknesses, and thereby to understand how LLMs alone perform when compared to SATs.

Table III, rows 5 and 8, presents the metrics for $\texttt{GPT-4o}_{\texttt{rbp}}$ and KAVE, respectively. While both approaches demonstrated similar accuracy and precision (although not so high), significant differences were observed in recall and FPR. KAVE achieved a lower FPR than $\texttt{GPT-4o}_{\texttt{rbp}}$ (on average 12% against 65%), while its recall was much lower than the LLM recall (on average 18% against 94%). Although KAVE is specifically designed to reduce false positives through precise taint analysis and pruning [22], it is limited to the knowledge it has about vulnerabilities. With a deep analysis of the code snippets, we noticed that they contain sensitive sinks that do not belong to KAVE's vulnerability database, and thereby explain its low recall. This bottleneck is intrinsic to the static sink database: sinks absent from it cannot be flagged, capping KAVE's recall regardless of the hybrid integration. Dynamically extending the SAT sink database is therefore a key direction to fully unlock the hybrid framework's potential. On the other hand, $\texttt{GPT-4o}$ achieved a very high recall, due to its capacity to discover code properties associated with vulnerabilities. However, due to this capacity, it also tends to overestimate vulnerabilities by flagging too many cases as *Unsafe*, thus reaching very high FPRs. Despite this trade-off, $\texttt{GPT-4o}_{\texttt{rbp}}$ offers significant advantages in adaptability and achieved a much higher F1-score overall (on average 72% against 28% from KAVE). Although the findings seem contradictory, they also highlight complementary strengths between SATs and LLMs. This suggests that combining these approaches could lead to a more robust vulnerability detection framework capable of addressing their respective limitations.

Findings: KAVE *excels in precision and FPR, while* $\texttt{GPT-4o}_{\texttt{rbp}}$ *demonstrate superior recall and adaptability.*

E. Hybrid Framework Evaluation

In this testing stage, we performed three types of tests: i) determine which is the best integration of KAVE within LLMs tuned with role-based prompts in order to constitute the final hybrid framework; ii) the hybrid framework composed in i) when processing the dataset without their instances labelled as *Unsafe* and *Safe*; and iii) the hybrid framework when processing a set of real-world web applications containing vulnerabilities discovered in the past. The next sections present the results.

1) Integrating KAVE *into LLMs:* We evaluated the integration of KAVE's structured outputs into LLMs' contextual understanding using the role-based prompts, aiming to understand whether improvements in detection effectiveness can be achieved while reducing FP, and thereby discover which is the best combination to constitute the final hybrid framework. Table III, rows 9 to 12, shows the results. When comparing the effectiveness metrics of hybrid solutions and standalone LLM using role-based prompts (rows 4 to 7 of the table), the results clearly reveal that hybrid solutions significantly improve effectiveness in most metrics, with $\texttt{GPT-4o}_{\texttt{rbp}}$+KAVE being the best, and thus supporting our intuition that SAT's outputs can empower LLMs in vulnerability detection tasks. Next, we discuss the results, highlighting what was improved in terms of already existing strengths and weaknesses.

Improved Accuracy and Precision: The hybrid framework led to substantial improvements in accuracy and precision. For example, for SQLi, $\texttt{GPT-4o}_{\texttt{rbp}}$ alone achieved 63.7% and 57.9% on these metrics, while the $\texttt{GPT-4o}_{\texttt{rbp}}$+KAVE hybrid framework improved them by 22% and 26%. This improvement can be attributed to KAVE providing structured information that helps the LLM to identify safe code better and assess vulnerabilities more accurately.

Reduction in False Positives: A significant reduction in FP was observed when integrating KAVE with LLMs. For SQLi, $\texttt{GPT-4o}_{\texttt{rbp}}$ alone produced 363 FP, whereas the framework reduced this number to 83. This suggests that KAVE's output effectively guided the LLM in reducing unnecessary alerts by more accurately distinguishing safe code, as it identifies sanitisation functions in the code. This shows that while LLMs alone increase FPs due to overestimation, anchoring their reasoning in SAT-verified paths successfully mitigates this weakness.

Balanced Detection: While standalone LLMs excelled at recall (e.g., $\texttt{GPT-4o}_{\texttt{rbp}}$ achieved 100% recall for SQLi), they often overestimated vulnerabilities, leading to high FPR. The hybrid framework maintained high recall, while significantly improving FPR by 49% in $\texttt{GPT-4o}_{\texttt{rbp}}$+KAVE, on average. This balance makes the hybrid approach more practical for real-world applications where both effective vulnerability detection and a reduced number of false positives are critical.

Complementary Strengths: The results highlight comple-

mentary strengths between static analysis tools like KAVE and LLMs. While KAVE excels at identifying specific patterns and minimising false positives, LLMs bring flexibility and adaptability to detect a broader range of vulnerabilities.

Final Hybrid Framework: *GPT-4o LLM tuned with role-based prompts empowered by* KAVE *results* (GPT-4o$_{rbp}$+KAVE) was the best combination in the detection tasks.

2) Comparison with unlabelled dataset: To further investigate the impact of KAVE's output on LLM effectiveness, we tested GPT-4o$_{rbp}$ receiving only KAVE's intermediate security metadata (entry points, sensitive sinks, sanitisation functions) without explicit vulnerability labels. This configuration achieved lower effectiveness compared to providing full KAVE conclusions as shown in the last row of Table III. However, even in this reduced-context setting, GPT-4o$_{rbp}$+KAVE (unlabelled) still outperformed the standalone GPT-4o$_{rbp}$ (row 5 of the table), demonstrating that access to static analysis metadata alone can boost vulnerability detection.

3) Evaluation with vulnerable real web applications: Lastly, we evaluate our final hybrid framework with a set of 10 real web applications to verify how it behaves in a real-world context and when processing large projects. The selected projects vary in size and complexity, ranging from 270 to 6K lines of code (LoC). The dataset includes intentionally vulnerable applications (e.g., Butterfly insecure, OWASP Vicnum, and Peruggia), as well as security-focused variants (e.g., Butterfly secure and PHP X Template). This diversity ensures coverage of different vulnerability densities, coding styles, and levels of security hardening.

The results in Tables IV and V show that the hybrid framework outperformed the standalone GPT-4o$_{rbp}$ detection, improving the F1-score by 6–12% and the recall by 31% (from 58% to 89%) in XSS detection. In these real-world applications, the standalone KAVE still attained the highest F1-score (77% for SQLi and 92% for XSS), as it is tuned to minimise FPR; the hybrid framework, however, improves KAVE's SQLi recall (from 65% to 74%) while recovering from the high FPR of the standalone LLM. In more detail, the integration of KAVE into GPT-4o$_{rbp}$ reduced FN in some applications (e.g., Butterfly insecure), while also preventing the large number of FP observed in LLM-only detection for larger projects (e.g., OWASP Vicnum).

When comparing with the SARD-based experiments, several insights emerge. First, the effectiveness trends are consistent: standalone LLMs achieve high recall but suffer from excessive false positives, while KAVE achieves low FPR but misses vulnerabilities, especially when obfuscation or non-trivial flows are present. The hybrid framework again reconciles these trade-offs, achieving a higher F1-score than standalone LLMs. However, the real-world applications' results also reveal additional challenges: projects with mixed or legacy coding styles (e.g., OWASP Vicnum) introduce noise that increases false positives even in the hybrid setting, and highly secure variants (e.g., Butterfly secure) test the ability of the framework to avoid over-classification, which it handled more effectively than the LLM alone.

F. Zero-Day Vulnerability Discovery in Real Web Applications

In addition to the controlled benchmark and real-world application (where vulnerabilities were already disclosed) evaluation presented above, we conducted an additional validation study to assess the practical effectiveness and generalisation capability of the proposed hybrid framework. This study focuses on the attempt to identify previously undocumented vulnerabilities (zero-day vulnerabilities) in real-world open-source PHP web applications.

1) Study Setup: We analysed 5 other open-source PHP web applications (i.e., distinct from the 10 applications evaluated in Section V-E) selected from GitHub, accounting for 881 *.php* files and over 170K lines of code (LoC), each with moderate community adoption, ranging from approximately 150 to 850 stars. At the time of analysis, none of these applications were associated with existing CVE records, i.e., vulnerabilities already disclosed. The analysis was performed exclusively using the best-performing configuration identified in our evaluation, namely the hybrid framework GPT-4o$_{rbp}$+KAVE.

2) Discovered Vulnerabilities: Using the proposed hybrid framework, we discovered 16 zero-days vulnerabilities across the analysed applications. Specifically, we found 5 XSS vulnerabilities and 11 vulnerabilities belonging to other classes, including header injection, path traversal, command injection, and log injection. Unlike the strict SQLi/XSS-specific prompts of Section IV-B2, this study guided the LLM with a generalised Security Auditor role-based prompt. Because KAVE's multi-layer knowledge graph extracts entry–sink pairs regardless of vulnerability class, feeding these verified flows into the generalised prompt allows the LLM to reason about and discover a wider variety of vulnerability types. All identified undocumented vulnerabilities were manually validated through careful code inspection to confirm their root cause and exploitability. No exploit payloads or sensitive implementation

TABLE IV
WEB APPLICATION RESULTS OF KAVE, LLM GPT-4O$_{RBP}$, AND HYBRID FRAMEWORK GPT-4O$_{RBP}$+KAVE.

Web Application	KAVE						GPT-4o$_{rbp}$						GPT-4o$_{rbp}$+KAVE					
	SQLi	FP	FN	XSS	FP	FN	SQLi	FP	FN	XSS	FP	FN	SQLi	FP	FN	XSS	FP	FN
Butterfly insecure	0	0	9	8	0	0	9	13	0	8	17	0	9	7	0	8	16	0
Butterfly secure	0	0	10	5	0	0	9	13	1	4	21	1	10	8	0	4	18	1
currentcost	3	0	2	6	0	0	5	0	0	6	3	0	5	0	0	6	5	0
Ghost	1	0	3	17	0	0	4	6	0	13	5	4	4	7	0	14	5	3
gilbitron-PIP	0	0	0	0	0	1	0	2	0	1	0	0	0	2	0	1	0	0
Mfm 0.13	0	0	0	6	0	0	0	5	0	4	3	2	0	2	0	4	3	2
OWASP Vicnum	19	3	0	23	1	0	7	12	12	20	26	3	6	8	13	20	22	3
Peruggia	18	0	0	8	0	3	12	3	6	11	4	0	13	3	5	11	3	0
PHP X Template 0.4	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	2	0
ZiPEC 0.32	4	0	0	0	0	7	4	6	0	4	20	3	4	7	0	7	18	0
Total	45	3	24	73	1	11	50	60	19	71	101	13	51	44	18	75	92	9

TABLE V
PERFORMANCE METRICS OF KAVE, LLM GPT-4O$_{RBP}$, AND HYBRID FRAMEWORK GPT-4O$_{RBP}$+KAVE WHEN PROCESSING REAL WEB APPLICATIONS.

	SQL Injection (SQLi)			Cross-Site Scripting (XSS)		
Metric (%)	KAVE	GPT-4o$_{rbp}$	GPT-4o$_{rbp}$+KAVe	KAVE	GPT-4o$_{rbp}$	GPT-4o$_{rbp}$+KAVe
Prc	94	45	54	99	41	45
Rcl	65	72	74	87	58	89
F1	77	56	62	92	48	60

details are disclosed in this paper because of ethical considerations (see next section). While the benchmark evaluation in this study focuses on SQLi and XSS, the discovery of additional vulnerability classes in real-world applications highlights the potential benefits of integrating statically extracted program properties with LLM-based reasoning. In particular, the LLM component can leverage broader execution context and data-flow information provided by static analysis to reason about security-relevant behaviours beyond predefined vulnerability patterns. This additional validation study is not intended as a comprehensive vulnerability discovery assessment. Instead, it provides complementary evidence that the proposed hybrid approach can generalise beyond benchmark scenarios and assist in identifying zero-day vulnerabilities in practical settings.

3) Ethical Considerations: The zero-day vulnerabilities identified in this study were discovered during an additional validation phase on open-source PHP applications. During this test, project maintainers of the affected applications, i.e., those for which we found zero-days, were contacted to disclose the identified issues before publication. Because of this, the affected applications are anonymised in this paper, and no exploit payloads, sensitive configuration details, or vulnerability-specific identifiers are revealed. This study was conducted solely for research and validation purposes, with the objective of improving vulnerability detection techniques, and no actions were taken that could disrupt deployed systems or negatively impact users or organisations. If the reported vulnerabilities are fixed by the maintainers in time, the affected projects and vulnerability details (including CVEs) may be disclosed in the camera-ready version of this paper.

Overall, the evaluation confirms that the hybrid framework offers advantages in real applications similar to those observed in the snippet-based experiments, reducing false positives while maintaining high recall. However, its effectiveness in full-scale web applications remains more limited. A key constraint lies in the context window of LLMs, which restricts their ability to reason over large and complex projects without code segmentation or summarisation strategies. While the framework adapts well to small and medium-sized projects, scaling to larger applications will require future work on hierarchical analysis, code partitioning, or multimodal representations to overcome these inherent limitations.

VI. Threats to Validity

a) Internal Validity: To reduce confounding effects, we evaluate the proposed hybrid framework against clearly defined baselines, including standalone SAT tools and standalone LLM-based detection, using identical datasets and metrics. Vulnerabilities identified in real-world applications, including zero-day, were manually validated through code inspection.

b) Construct Validity: We measure vulnerability detection effectiveness using commonly adopted security metrics, including false positives, false negatives, precision, recall, and F1-score. While these metrics may not fully capture the practical severity of vulnerabilities in deployment environments, they provide a consistent basis for comparative evaluation.

c) External Validity: Our evaluation focuses on PHP applications and on SQLi and XSS vulnerabilities, which are among the most prevalent web vulnerability classes. While this limits the direct empirical evidence to a single language, PHP serves as a representative proxy for dynamic, loosely-typed web languages (e.g., JavaScript/Node.js, Python) which suffer from similar static analysis challenges. Furthermore, since PHP's dynamic features historically generate higher SAT false positive rates than strongly-typed languages (like Java), our results likely represent a lower bound on the framework's effectiveness; we hypothesise that performance would be equivalent or better in languages with more rigid type systems. Finally, scalability to very large applications may also be affected by current LLM context window limitations, necessitating future work on hierarchical context management.

d) Tool Selection Bias: Our evaluation relies on the KAVE static analysis tool. We acknowledge that different SATs have varying false negative rates and output formats, which could influence the hybrid framework's effectiveness. However, the core contribution of this work, leveraging *verified data flows* to ground LLM reasoning, remains tool-agnostic. We mitigate this threat by using KAVE merely as a provider of 'Entry-Sink' pairs, a fundamental unit of analysis common to almost all modern taint analysis engines (e.g., CodeQL's data-flow queries).

e) Conclusion Validity: The non-deterministic nature of LLMs and sensitivity to prompt design may introduce variability in the results. To mitigate this, we report results across multiple datasets and applications and avoid drawing conclusions from isolated cases.

VII. Framework Extensibility

A key strength of the proposed hybrid framework lies in its modular architecture, which makes it adaptable to other vulnerability classes, diverse static analysis tools, and different programming languages. Because the framework consumes SAT outputs rather than replacing static analysis, it can naturally integrate alternative analysers or be combined with LLM-assisted analysis techniques. In terms of vulnerabilities, the framework can naturally evolve beyond SQLi and XSS by reusing its decomposition strategy based on *entry–sink* flows and sanitisation checks. By adapting the underlying SAT rules to capture additional classes, such as command injection, Cross-site Request Forgery (CSRF), or insecure deserialisation, and by adjusting the LLM prompts to reflect the semantics of these threats, the framework can address a broader range of Common Weakness Enumeration (CWE) [23] categories without requiring a major redesign.

The design of the framework also supports the integration of different SATs. This is possible through the abstraction of SAT outputs into a unified intermediate representation; e.g., code property graphs or normalised vulnerability metadata. Such an approach would allow cross-validation between analysers and even ensemble-based detection strategies, ultimately increasing robustness and reducing FPs.

Although our study focused on PHP, the methodology generalises to other languages. Many SATs already provide multi-language support, and LLMs can reason about vulnerabilities in a syntax-agnostic manner. Extending the framework to Java, Python, or JavaScript needs only minimal adaptations, such as replacing language-specific sinks and sanitisation functions, rather than retraining or redesigning the pipeline.

VIII. Conclusions

This paper presented a *Hybrid Framework* for detecting web application vulnerabilities that addresses the high FP rates of both SAT tools and LLM-based detection. By integrating the results of SAT tools, such as KAVE, into LLMs, it is possible to create a more balanced solution that reduces FPs without significantly compromising the ability to detect vulnerabilities.

The hybrid approach addresses two of the main limitations of SAT tools – the inability to adapt to new patterns or scenarios without subsequent updates, and the generation of FPs. Concurrently, it mitigates the tendency of LLMs to excessively classify vulnerabilities, thereby also resulting in a high number of FPs. The findings for SQLi and XSS detection indicated that this framework improves software security, achieving a maximum F1 score and a minimum false positive rate of 86% and 15%, respectively, with the SARD dataset, and over a set of real-world applications, a higher F1-score and recall than the standalone LLM, particularly for XSS. In addition, the framework was capable of discovering 16 zero-day vulnerabilities in 5 real-world web applications. These results demonstrate a clear improvement over the standalone LLM and a stronger precision–recall trade-off than both the standalone SAT KAVE and the LLM, even though the standalone SAT retains a higher F1-score on the real-world applications. While the study focuses on SQL injection and XSS vulnerabilities, two of the most prevalent and security-critical vulnerability classes in PHP applications, the underlying methodology is applicable to other vulnerability classes and languages, provided suitable static analysis support is available.

Acknowledgments

This work was partially supported by P2030 through project I2DT, ref. COMPETE2030-FEDER-00389100, an ITEA4 European project (ref. 22025), and by FCT through the LASIGE Research Unit, ref. UID/408/2025.

References

[1] Veracode, "State of Software Security 2023. Annual Report on the State of Application Security," https://info.veracode.com/rs/790-ZKW-291/images/Veracode_State_of_Software_Security_2023.pdf, 2023.

[2] J. Daley, "Insecure software is eating the world: Promoting cybersecurity in an age of ubiquitous software embedded systems." *Stanford Technology Law Review*, vol. 19, no. 3, 2017.

[3] CVE, "CVE Details," https://www.cvedetails.com/browse-by-date.php, 2026.

[4] NIST NVD, "Vulnerability metrics," https://nvd.nist.gov/vuln-metrics/cvss, 2026.

[5] W3Techs - Web Technology Surveys, "Usage statistics of server-side programming languages for websites," https://w3techs.com/technologies/overview/programming_language, 2026.

[6] OWASP Top 10 Team, "OWASP Top Ten 2025," https://owasp.org/Top10/2025/0x00_2025-Introduction/, 2025.

[7] Veracode, "State of Software Security 2023," https://www.veracode.com/state-of-software-security, 2023, accessed: 2025-03-08.

[8] I. Medeiros, N. F. Neves, and M. Correia, "Automatic detection and correction of web application vulnerabilities using data mining to predict false positives," in *Proceedings of the 23rd International Conference on World Wide Web*, April 2014, p. 63–74.

[9] M. Hills, P. Klint, and J. Vinju, "Enabling php software engineering research in rascal," *Science of Computer Programming*, vol. 134, pp. 37–46, 2017.

[10] J. Dahse and T. Holz, "Experience report: an empirical study of php security mechanism usage," *Proceedings of the 2015 International Symposium on Software Testing and Analysis*, 2015.

[11] Z. Sheng, Z. Chen, S. Gu, H. Huang, G. Gu, and J. Huang, "Llms in software security: A survey of vulnerability detection techniques and insights," *arXiv preprint arXiv:2502.07049*, Feb 2025. [Online]. Available: https://arxiv.org/abs/2502.07049

[12] X. Zhou, S. Cao, X. Sun, and D. Lo, "Large language model for vulnerability detection and repair: literature review and the road ahead," *ACM Transactions on Software Engineering and Methodology*, 2024.

[13] S. M. Taghavi and F. Feyzi, "Using large language models to better detect and handle software vulnerabilities and cyber security threats," 2024.

[14] B. Mata and V. Madisetti, "Large language models (llms) for software security analysis," *Journal of Information Security*, vol. 16, no. 2, pp. 341–357, 2025.

[15] G. Ryan, S. Jain, M. Shang, S. Wang, X. Ma, M. K. Ramanathan, and B. Ray, "Code-aware prompting: A study of coverage-guided test generation in regression setting using LLM," *Proc. ACM Softw. Eng.*, vol. 1, no. FSE, pp. 951–971, 2024. [Online]. Available: https://doi.org/10.1145/3643769

[16] Z. Li, S. Dutta, and M. Naik, "Iris: Llm-assisted static analysis for detecting security vulnerabilities," 2025. [Online]. Available: https://arxiv.org/abs/2405.17238

[17] OpenAI, "Gpt-4o mini," 2024, large language model. [Online]. Available: https://openai.com/product/gpt-4o-mini

[18] ——, "Gpt-4o," 2024, large language model. [Online]. Available: https://openai.com/product/gpt-4o

[19] Google, "Gemini 2.0 flash," 2025, large language model. [Online]. Available: https://gemini.google.com/

[20] DeepSeek-AI, "Deepseek-v3," 2024, large language model. [Online]. Available: https://arxiv.org/abs/2412.19437

[21] R. Ramires, A. Respicio, and I. Medeiros, "Kave: A knowledge-based multi-agent system for web vulnerability detection," in *2024 IEEE International Conference on Web Services (ICWS)*, 2024, pp. 489–500.

[22] Rafael Ramires, "KAVe: Knowledge-Based Multi-Agent System Vulnerability Detector," https://github.com/rframires/KAVe.git, 2024.

[23] T. M. Corporation, "Common weakness enumeration (CWE)," https://cwe.mitre.org/, 2026.

[24] Kingthorin, "Sql injection," https://owasp.org/www-community/attacks/SQL_Injection, 2026.

[25] S. Kost, "An introduction to sql injection attacks for oracle developers," 2007.

[26] A. Klein, "Dom based cross site scripting or xss of the third kind," http://www.webappsec.org/projects/articles/071105.shtml, 2005.

[27] D. Wichers, A. Dabirsiaghi, S. D. Paolo, M. Heiderich, E. A. V. Nava, and J. Williams, "Types of xss," https://owasp.org/www-community/Types_of_Cross-Site_Scripting, 2026.

[28] T. D. Oyetoyan, B. Milosheska, M. Grini, and D. S. Cruzes, "Myths and facts about static application security testing tools: An action research at telenor digital," in *Agile Processes in Software Engineering and Extreme Programming - 19th International Conference, XP 2018, Porto, Portugal, May, Proceedings*, ser. Lecture Notes in Business Information Processing, J. Garbajosa, X. Wang, and A. Aguiar, Eds., vol. 314. Springer, 2018, pp. 86–103. [Online]. Available: https://doi.org/10.1007/978-3-319-91602-6_6

[29] C. Vassallo, S. Panichella, F. Palomba, S. Proksch, H. Gall, and A. Zaidman, "How developers engage with static analysis tools in different contexts," *Empirical Software Engineering*, vol. 25, pp. 1419–1457, 2019.

[30] M. Backes, K. Rieck, M. Skoruppa, B. Stock, and F. Yamaguchi, "Efficient and flexible discovery of PHP application vulnerabilities," in

Proceedings of the 2017 IEEE European Symposium on Security and Privacy (EuroS&P), Apr. 2017, pp. 334–349.

[31] R. Johnson, D. Pearson, and K. Pingali, "The program structure tree: Computing control regions in linear time," ser. PLDI '94. New York, NY, USA: Association for Computing Machinery, 1994, p. 171–185. [Online]. Available: https://doi.org/10.1145/178243.178258

[32] F. E. Allen, "Control flow analysis," *SIGPLAN Not.*, vol. 5, no. 7, p. 1–19, jul 1970. [Online]. Available: https://doi.org/10.1145/390013.808479

[33] I. Medeiros, N. Neves, and M. Correia, "Detecting and removing web application vulnerabilities with static analysis and data mining," *IEEE Transactions on Reliability*, vol. 65, no. 1, pp. 54–69, 2016.

[34] J. Dahse and T. Holz, "Simulation of built-in PHP features for precise static code analysis," in *Proceedings of the Annual Network and Distributed System Security Symposium, NDSS 2014, San Diego, California, USA, February*. The Internet Society, 2014. [Online]. Available: https://www.ndss-symposium.org/ndss2014/simulation-built-php-features-precise-static-code-analysis

[35] M. Wijngaard, "Dependence analysis in php," Amsterdam, The Netherlands, Aug. 2016. [Online]. Available: http://www.scriptiesonline.uba.uva.nl/618176

[36] J. Ferrante, K. J. Ottenstein, and J. D. Warren, "The program dependence graph and its use in optimization," *ACM Trans. Program. Lang. Syst.*, vol. 9, no. 3, p. 319–349, jul 1987. [Online]. Available: https://doi.org/10.1145/24039.24041

[37] A. Johnson, L. Waye, S. Moore, and S. Chong, "Exploring and enforcing security guarantees via program dependence graphs," *SIGPLAN Not.*, vol. 50, no. 6, p. 291–302, jun 2015. [Online]. Available: https://doi.org/10.1145/2813885.2737957

[38] O. Tripp, M. Pistoia, S. J. Fink, M. Sridharan, and O. Weisman, "Taj: effective taint analysis of web applications," *ACM Sigplan Notices*, vol. 44, no. 6, pp. 87–97, 2009.

[39] A. Figueiredo, T. Lide, D. Matos, and M. Correia, "Merlin: Multi-language web vulnerability detection," in *Proceedings of the IEEE 19th International Symposium on Network Computing and Applications (NCA)*, 2020, pp. 1–9.

[40] S. Wi, S. Woo, J. J. Whang, and S. Son, "Hiddencpg: Large-scale vulnerable clone detection using subgraph isomorphism of code property graphs," in *Proceedings of the ACM Web Conference 2022*, ser. WWW '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 755–766. [Online]. Available: https://doi.org/10.1145/3485447.3512235

[41] Y. Shi, Y. Zhang, T. Bai, L. Zhang, X. Tan, and M. Yang, "Recurscan: Detecting recurring vulnerabilities in php web applications," in *Proceedings of the ACM Web Conference 2024*, ser. WWW '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 1746–1755. [Online]. Available: https://doi.org/10.1145/3589334.3645530

[42] J. Bessey, T. Nguyen, S. Rajan, K. Sen, Z. Spanner, D. Xu, and D. Engler, "A few billion lines of code later: Using static analysis to find bugs in the real world," *Communications of the ACM*, vol. 53, no. 2, pp. 66–75, Feb. 2010.

[43] M. Christakis and C. Bird, "What developers want and need from program analysis: An empirical study," in *Proceedings of the 38th International Conference on Software Engineering (ICSE '16)*. ACM, 2016, pp. 332–343.

[44] B. Chess and G. McGraw, "Static analysis for security," *IEEE Security & Privacy*, vol. 2, no. 6, pp. 76–79, Dec. 2004.

[45] S. Lipp, S. Banescu, and A. Pretschner, "An empirical study on the effectiveness of static c code analyzers for vulnerability detection," in *Proceedings of the 31st ACM SIGSOFT international symposium on software testing and analysis*, 2022, pp. 544–555.

[46] M. Keltek, R. Hu, M. F. Sani, and Z. Li, "Lsast: Enhancing cybersecurity through llm-supported static application security testing," in *IFIP International Conference on ICT Systems Security and Privacy Protection*. Springer, 2025, pp. 166–179.

[47] W. Melicher, C. Fung, L. Bauer, and L. Jia, "Towards a lightweight, hybrid approach for detecting dom xss vulnerabilities with machine learning," in *Proceedings of the Web Conference 2021*, ser. WWW '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 2684–2695. [Online]. Available: https://doi.org/10.1145/3442381.3450062

[48] H. Park and K. Yoon, "Exploiting multi-layer graph factorization for multi-attributed graph matching," *Pattern Recognition Letters*, vol. 127, pp. 85–93, 2019.

[49] Z. Zeng, Q. Cheng, and Y. Si, "Logical rule-based knowledge graph reasoning: a comprehensive survey," *Mathematics*, vol. 11, p. 4486, 2023.

[50] A. Hilbig, D. Lehmann, and M. Pradel, "An empirical study of real-world webassembly binaries: Security, languages, use cases," in *Proceedings of the Web Conference 2021*, ser. WWW '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 2696–2708. [Online]. Available: https://doi.org/10.1145/3442381.3450138

[51] Y. Guo, C. Patsakis, Q. Hu, Q. Tang, and F. Casino, "Outside the comfort zone: Analysing llm capabilities in software vulnerability detection," in *European symposium on research in computer security*. Springer, 2024, pp. 271–289.

[52] M. A. Ferrag, F. Alwahedi, A. Battah, B. Cherif, A. Mechri, N. Tihanyi, T. Bisztray, and M. Debbah, "Generative ai in cybersecurity: A comprehensive review of llm applications and vulnerabilities," *Internet of Things and Cyber-Physical Systems*, 2025.

[53] Y. Yao, J. Duan, K. Xu, Y. Cai, Z. Sun, and Y. Zhang, "A survey on large language model (llm) security and privacy: The good, the bad, and the ugly," *High-Confidence Computing*, p. 100211, 2024.

[54] Y. Sun, H. Chen, Z. Fu, W. Lv, Z. Liu, and H. Liu, "Wasmguard: Enhancing web security through robust raw-binary detection of webassembly malware," in *Proceedings of the ACM on Web Conference 2025*, ser. WWW '25. New York, NY, USA: Association for Computing Machinery, 2025, p. 1942–1950. [Online]. Available: https://doi.org/10.1145/3696410.3714696

[55] C. Chen, J. Su, J. Chen, Y. Wang, T. Bi, J. Yu, Y. Wang, X. Lin, T. Chen, and Z. Zheng, "When chatgpt meets smart contract vulnerability detection: How far are we?" *ACM Trans. Softw. Eng. Methodol.*, vol. 34, no. 4, Apr. 2025. [Online]. Available: https://doi.org/10.1145/3702973

[56] A. Author(s), "Replication package for hybrid_kave," https://anonymous.4open.science/r/Hybrid_KAVe-6D61/README.md, 2026.

[57] National Institute of Standards and Technology (NIST), "NIST Software Assurance Reference Dataset (SARD)," 2026. [Online]. Available: https://samate.nist.gov/SARD

[58] MITRE Corporation, "CWE-79: Improper Neutralization of Input During Web Page Generation (Cross-site Scripting)," https://cwe.mitre.org/data/definitions/79.html, February 2026.

[59] ——, "CWE-89: Improper Neutralization of Special Elements used in an SQL Command (SQL Injection)," https://cwe.mitre.org/data/definitions/89.html, February 2026.

[60] S. Kaufman, S. Rosset, and C. Perlich, "Leakage in data mining: Formulation, detection, and avoidance," vol. 6, 01 2011, pp. 556–563.

[61] N. Jovanovic, C. Kruegel, and E. Kirda, "Pixy: a static analysis tool for detecting web application vulnerabilities," in *2006 IEEE Symposium on Security and Privacy (SP'06)*, 2006, pp. 6 pp.–263.

[62] R. Morgado, I. Medeiros, and N. Neves, "Towards web application security by automated code correction," in *Proceedings of the 15th International Conference on Evaluation of Novel Approaches to Software Engineering*, Apr. 2020, pp. 86–96.

[63] R. Ramires, A. Respício, I. Medeiros, and M. Papadakis, "Kave: A tool to detect xss and sqli vulnerabilities using a multi-agent system over a multi-layer knowledge graph," in *33rd ACM International Conference on the Foundations of Software Engineering (FSE)*, 07 2025, pp. 1158–1162.