# The State of the Art in Web Application Security Testing: A Systematization of Knowledge from 2024-2025 Research

## Executive Summary

The landscape of web application security testing is undergoing a significant transformation, driven by the escalating complexity of modern web architectures and the advent of powerful new analytical paradigms. Research published in premier cybersecurity conferences throughout 2024 and early 2025 reveals a distinct departure from the traditional, pattern-matching vulnerability scanners that have long defined the field. The state of the art is rapidly moving towards more intelligent, context-aware, and deeply integrated analysis frameworks. This evolution is a direct response to the challenges posed by highly stateful applications, asynchronous client-server communication, and intricate server-side business logic, which have rendered conventional black-box testing methods increasingly ineffective.

This report synthesizes the most impactful advancements, identifying three predominant trends that characterize the current research frontier. First, the integration of Artificial Intelligence, particularly Large Language Models (LLMs), has matured from a tool for simple data generation to a sophisticated engine for semantic interpretation and guided exploration. State-of-the-art frameworks now leverage LLMs to automatically interpret natural language protocol specifications, synthesize complex fuzzing drivers, and navigate intricate application state machines, fundamentally reducing the manual effort required for deep security analysis.[1]

Second, there is a clear and accelerating trend towards hybrid and grey-box analysis methodologies that intentionally break the black-box abstraction. The most innovative tools are no longer content to operate from an external vantage point. Instead, they achieve unprecedented precision and coverage by gaining privileged access to internal application components. This includes directly instrumenting the language interpreter to understand input structures [2], manipulating the application's database to bypass input validation logic [3], and analyzing the interpreter's binary to build a

ground-truth model of its sensitive functions.[4] This shift suggests that the most significant gains in vulnerability discovery are now being realized by finding clever ways to acquire and leverage internal context.

Third, the focus of advanced security testing has decisively shifted towards detecting deep, stateful, and server-side vulnerabilities. Researchers are moving beyond the detection of simple, reflected flaws like basic Cross-Site Scripting (XSS) and are instead developing specialized tools to tackle more elusive and often more critical bug classes. These include complex logic vulnerabilities, Server-Side Request Forgery (SSRF), insecure deserialization, and subtle state-management flaws that are deeply embedded within the application's server-side architecture.[2] This specialization underscores a growing recognition that a one-size-fits-all approach is no longer adequate for securing the modern web.

Collectively, these trends paint a picture of a field in dynamic flux, where the lines between systems security and web security are blurring. The most effective new frameworks are those that combine the speed and scalability of dynamic fuzzing with the deep contextual understanding previously associated with static analysis, augmented by the semantic reasoning capabilities of AI. This report provides an in-depth analysis of the key frameworks driving this change, a comparative look at their methodologies, and a discussion of the future directions this research trajectory suggests.

## In-depth Analysis of State-of-the-Art Frameworks

The research presented in 2024 and 2025 showcases a new generation of security testing frameworks that address long-standing challenges in web application analysis. By examining the most innovative and representative publications from top-tier conferences, we can dissect the core methodologies and design principles that define the current state of the art. The following sections provide a detailed analysis of four such pioneering frameworks, each representing a distinct and impactful approach to modern vulnerability discovery.

### Atropos: Snapshot-Based Fuzzing for Server-Side Vulnerabilities

The Atropos framework represents a significant step in adapting the highly successful principles of systems-level fuzzing to the unique and challenging environment of server-side web applications.[2] Traditional dynamic application security testing (DAST) tools often struggle with the stateful nature of web applications and the highly structured, developer-defined inputs they require. Atropos addresses these issues head-on by introducing a snapshot-based, feedback-driven fuzzing methodology specifically tailored for PHP, one of the most widely deployed languages for web development.[2]

**Core Idea**

The primary innovation of Atropos is the successful transplantation of mature techniques from systems fuzzing—such as snapshotting and coverage feedback—into the web security domain. Web applications present challenges that are foreign to typical fuzzing targets like file parsers or command-line utilities. They maintain complex session states, and their inputs are not simple byte streams but structured key-value pairs (e.g., from HTTP POST requests) where the keys themselves carry semantic meaning.[2] A fuzzer that merely mutates input values without understanding the required keys (e.g.,

username, password) will fail to penetrate beyond the initial input parsing logic. Atropos is designed to overcome these specific roadblocks by using a novel feedback mechanism to learn the required input structure and a snapshotting engine to manage application state, enabling deep and effective testing of server-side code.[2]

**Methodology**

Atropos's methodology is a carefully constructed hybrid of dynamic analysis techniques.

- **Snapshot-Based Execution:** To manage the statefulness of web applications, Atropos employs a snapshotting engine. Before each test case is executed, the fuzzer creates a snapshot of the web application's state. After the execution, the

state is reverted. This approach provides two critical benefits: it ensures that test cases are isolated and do not have unintended side effects on one another, and it dramatically reduces the performance overhead associated with re-initializing the entire web application stack (e.g., web server, database connections) for every single input.[2] This technique is analogous to those used in modern hypervisor and firmware fuzzing.[7]

- **Interpreter-Level Feedback:** The most novel component of Atropos is its feedback mechanism. Instead of relying solely on code coverage, which provides limited insight into why an input failed, Atropos instruments the PHP interpreter itself. It monitors accesses to superglobal arrays like $_GET, $_POST, and $_COOKIE. When the application code accesses a key in one of these arrays (e.g., $_POST['password']), Atropos captures this key. This information is then fed back to the mutation engine, which learns the valid key-value structure expected by the application. This allows the fuzzer to generate semantically valid inputs (e.g., password=some_mutated_value) that can pass the initial parsing stages and explore deeper program logic.[2]
- **Custom Bug Oracles:** Recognizing that many web vulnerabilities do not result in a simple process crash, Atropos implements eight new bug oracles. These oracles are designed to detect specific classes of common server-side vulnerabilities, including SQL Injection (SQLi), Remote Code Execution (RCE), PHP Object Injection, and Server-Side Request Forgery (SSRF).[2] This goes beyond the simple crash-based oracles used in many traditional fuzzers.

**Strengths and Key Features**

The design of Atropos yields several significant advantages over existing web testing tools. Its primary strength is its **effective state handling** via snapshotting, which solves a problem that has plagued web scanners for years. The **high-quality input generation**, driven by the novel interpreter-level feedback, allows it to achieve much deeper code coverage than competing tools; its evaluation showed a 46% increase in coverage over the next-best tool.[6] Furthermore, as a dynamic analysis tool that confirms vulnerabilities with concrete, reproducible inputs, it boasts

**zero false positives**, a critical advantage over static analysis tools that often overwhelm developers with unexploitable findings.[2]

## Limitations

The power of Atropos also defines its limitations. The framework is highly **language-specific**. Its feedback mechanism is deeply integrated with the internals of the PHP interpreter, and adapting it to other web development environments like Node.js, Python/Django, or Java would require a similar, non-trivial engineering effort to instrument each respective language runtime.[2] Additionally, its performance and applicability are fundamentally

**dependent on the ability to efficiently snapshot** the target application environment, which may not be feasible or performant in all deployment scenarios.

## Evaluation and Impact

Atropos was evaluated against a suite of state-of-the-art web application testing tools and demonstrated superior performance. On average, it found 32% more bugs than the best-performing static analysis tool in its test set.[2] Its practical impact was validated by the discovery of seven previously unknown, real-world vulnerabilities in popular, widely deployed web applications, including NextCloud, AltoCMS, MaxSite, and phpwcms. These vulnerabilities spanned critical classes such as PHP Object Injection, SSRF, and RCE, with one vulnerability in NextCloud being assigned CVE-2022-31132.[6] The research was presented at the 33rd USENIX Security Symposium in 2024, marking a notable contribution to the field.[8]

The development of Atropos illustrates a broader trend: the convergence of techniques from systems security and web security. By successfully applying principles like snapshot-based execution and coverage-guided mutation—long staples of binary and kernel fuzzing—to the web domain, Atropos serves as a bridge between two historically separate fields. It demonstrates that the core concepts of feedback-driven fuzzing are potent tools for web security, provided that the right abstractions, like its key-value feedback mechanism, are created to accommodate the unique characteristics of web applications. This suggests a future where the distinction between a "web scanner" and a "systems fuzzer" becomes increasingly

irrelevant, replaced by more holistic and adaptable testing frameworks.

### Spider-Scents: Grey-Box, Database-Aware Scanning for Stored XSS

The Spider-Scents framework, presented at USENIX Security 2024, introduces a paradigm-shifting approach to detecting stored Cross-Site Scripting (XSS) vulnerabilities.[3] Stored XSS, where a malicious payload is saved in a persistent data store (typically a database) and later rendered to a victim, is notoriously difficult for automated scanners to detect. This difficulty arises from the need to identify a complete vulnerability chain: a vulnerable input point (source), the storage mechanism, and an unsafe output point (sink). Spider-Scents radically simplifies this problem by changing the fundamental assumptions of the testing process.[3]

#### Core Idea

The core insight of Spider-Scents is to bypass the most challenging part of stored XSS detection—finding a vulnerable input source. Traditional black-box scanners must craft HTTP requests that can navigate complex input validation, multi-step forms, and client-side logic to successfully inject a payload into the database.[3] Spider-Scents circumvents this entirely. It operates from a grey-box perspective, assuming the tester has direct access to the application's database. It injects XSS payloads

*directly into the database* and then uses a black-box crawler to observe where these payloads are rendered and executed in the web application's front-end. This reframes the problem from finding a complete source-to-sink path to the much simpler task of identifying unsafe sinks for data originating from the database.[3]

#### Methodology

Spider-Scents is implemented as a semi-automated, grey-box scanner that combines direct database manipulation with dynamic front-end crawling.

- **Grey-Box Database Access:** The methodology is predicated on having privileged access to the application's backend database, for which it uses a standard MySQL connection.[10]
- **Direct Payload Injection:** The tool systematically iterates through all string-type columns in the database. For each suitable cell, it inserts a unique, identifiable XSS payload. This injection happens at the database level, thereby bypassing 100% of the web application's input validation, sanitization logic, and business rules.[3]
- **Reflection Scanning:** Once a payload is injected, a standard black-box web crawler (based on the Black Widow scanner) is deployed to navigate the application's pages. The crawler's goal is to find where the unique payload ID from the database is reflected in the HTML output and, more importantly, where the <script> tag is executed.[3]
- **Code Smell Reporting:** A key conceptual contribution is that Spider-Scents does not report confirmed vulnerabilities. Instead, it reports "unprotected outputs," which the authors define as a "code smell".[3] An unprotected output signifies a location where data from the database is rendered without proper output escaping. This represents a fragile design that is dangerous even if not immediately exploitable, as future changes to the application (e.g., adding a new input path) could activate the vulnerability.
- **Manual Verification:** The final step requires a human analyst to investigate the reported unprotected outputs. The analyst must determine if a corresponding input vector exists in the application's UI that would allow an attacker to place a payload in that specific database location, thereby confirming a true, exploitable stored XSS vulnerability.[10]

**Strengths and Key Features**

This novel methodology provides several powerful advantages. The most significant is its **exceptional coverage**. By directly targeting the database, it can test data storage locations that are unreachable for black-box scanners, achieving 79-100% coverage of database fields in its evaluation, compared to a mere 2-60% for state-of-the-art scanners.[3] This allows it to

**bypass complex input logic**, making it immune to the multi-step forms and sophisticated client-side validation that often defeat conventional tools. Consequently, it has a **high vulnerability yield**, discovering 85 stored XSS

vulnerabilities in its test set, far surpassing the 32 found by the union of three other leading scanners.[3] Finally, the concept of reporting "code smells" allows it to

**identify fragile design patterns** and "dormant XSS," providing proactive security guidance to developers even for non-exploitable issues.[3]

### Limitations

The primary limitation of Spider-Scents is inherent in its design: it **requires privileged database access**. This makes it unsuitable for purely black-box penetration tests where the tester has no prior knowledge or access. It is a tool designed for developers, internal auditors, or grey-box/white-box security assessments.[3] The second major limitation is the necessity of

**manual verification**. The tool's output is a list of potential issues, and an analyst must invest time (estimated at 15 minutes per report) to confirm exploitability.[3] Its scope is also

**limited to stored XSS originating from a database**, and it does not address other vulnerability classes or other forms of XSS (e.g., reflected, DOM-based).[3]

### Evaluation and Impact

Spider-Scents was rigorously evaluated on 12 web applications, including both older reference applications and modern, complex content management systems like Joomla and WordPress. It was compared against three well-regarded black-box scanners: Arachni, Black Widow, and OWASP ZAP.[3] The results were definitive, showing that Spider-Scents vastly outperformed the others in both database coverage and the number of stored XSS vulnerabilities detected.[3] The research was presented at USENIX Security 2024, and the prototype tool,

dbfuzz, has been made publicly available on GitHub, with its artifact passing the "Available" and "Functional" evaluations.[11]

The development of Spider-Scents highlights a crucial evolution in the philosophy of

security auditing. The traditional goal of a web scanner is to perfectly mimic an external, unprivileged attacker. Spider-Scents challenges this dogma by arguing that for the intended user—a developer or internal auditor—this assumption is unnecessarily restrictive and counterproductive. A developer *does* have database access, and by leveraging this fact, the problem of finding stored XSS can be dramatically simplified. This demonstrates that one of the most effective ways to advance vulnerability discovery is to pragmatically reframe the problem and the threat model to suit the context of the audit. This suggests a future where security tools become more versatile, perhaps offering different operational modes—such as a "developer mode" with privileged access versus a "pentester mode" that remains strictly black-box—to maximize effectiveness for different use cases.

### ChatAFL: Large Language Model Guided Protocol Fuzzing

The ChatAFL framework, presented at NDSS 2024, stands at the forefront of a major trend in security testing: the integration of Large Language Models (LLMs) to infuse fuzzers with semantic understanding.[1] Fuzzing network protocols is challenging because protocols are stateful and require messages with specific structures and in a specific order. Traditional mutation-based fuzzers are often "blind" to this structure and state, leading to inefficient testing where the vast majority of generated inputs are immediately rejected by the server.[1] ChatAFL addresses this by using an LLM as an on-demand expert to guide a state-of-the-art fuzzer.[15]

#### Core Idea

The central idea of ChatAFL is to overcome the primary limitations of protocol fuzzing—dependence on good seeds, ignorance of message structure, and blindness to the state machine—by systematically querying an LLM like ChatGPT.[1] The authors posit that since modern LLMs have been trained on vast amounts of public data, including technical specifications like Internet Engineering Task Force (IETF) Requests for Comments (RFCs), they possess latent knowledge about how protocols work. ChatAFL's innovation is to create a system that can extract this latent knowledge and translate it into actionable guidance for a fuzzer, effectively creating a hybrid system where the fuzzer handles the high-speed mutation and the LLM provides high-level,

semantic direction.[1]

### Methodology

ChatAFL is architected as a hybrid fuzzer, built upon the foundation of AFLNet, a well-regarded stateful network fuzzer. It integrates an LLM as a "co-pilot" through three distinct mechanisms.[15]

- **LLM-driven Grammar Extraction:** Manually creating a grammar for a complex protocol is a time-consuming and error-prone task. ChatAFL automates this by prompting the LLM to generate a machine-readable grammar for a given protocol message type. The fuzzer can then use this grammar for structure-aware mutation, ensuring that generated packets are syntactically valid and more likely to be processed by the target.[1]
- **LLM-driven Seed Enrichment:** Protocol fuzzing is often hampered by a lack of diverse initial seed inputs. ChatAFL uses the LLM to enrich the seed corpus by prompting it to generate new, valid message sequences based on the protocol's logic, thereby improving the starting point for the fuzzer.[1]
- **LLM-driven State Exploration:** This is the most dynamic part of the methodology. When the fuzzer detects that it has reached a coverage plateau (i.e., it is no longer discovering new states or code paths), it queries the LLM for help. It provides the LLM with the context of the current protocol state and asks it to predict what type of message is needed to transition to a new, unexplored state. For example, in the Real-Time Streaming Protocol (RTSP), if the fuzzer is stuck in the READY state, the LLM might correctly predict that a PLAY or RECORD message is required to advance to the PLAYING state.[1] This allows the fuzzer to intelligently escape local coverage maxima.

### Strengths and Key Features

ChatAFL's LLM-guided approach offers several compelling strengths. It **automates the understanding of protocol specifications**, a task that traditionally requires significant human expertise and effort.[1] This automation leads to

**superior state and code coverage**. In its evaluation, ChatAFL covered 29.6% more

states and 5.8% more code than AFLNet, and 25.8% more states and 6.7% more code than NSFuzz.[15] This improved coverage translates directly into more

**effective bug finding**. The framework discovered nine new and distinct vulnerabilities in well-tested, widely-used protocol implementations, including critical bugs with CVSS scores as high as 9.8. A majority of these bugs were missed by the other state-of-the-art fuzzers in the comparison.[1]

### Limitations

The framework's strengths are intrinsically linked to its limitations. Its effectiveness is fundamentally **dependent on the quality and knowledge of the underlying LLM**. If the LLM has not been trained on a specific protocol's documentation or if it "hallucinates" an incorrect response, it could misguide the fuzzer, wasting time and resources. There are also practical concerns regarding **cost and latency**. Fuzzing is a process that involves millions or billions of executions, and making frequent API calls to a commercial LLM can introduce significant delays and financial costs, which runs counter to the "high-speed" ethos of fuzzing. Finally, the approach is most naturally suited for **well-documented, text-based protocols** (like FTP, SMTP, RTSP) whose RFCs are likely to be part of the LLM's training corpus. Its effectiveness on obscure, proprietary, or complex binary protocols is less certain.

### Evaluation and Impact

ChatAFL was evaluated on a benchmark of real-world protocol implementations from ProFuzzBench, including pure-ftpd, exim, and live555.[1] It was compared against two leading stateful protocol fuzzers, AFLNet and NSFuzz, and demonstrated superior performance in both state/code coverage and vulnerability discovery.[1] The work was presented at the NDSS Symposium 2024, and the tool and its artifact have been made publicly available on GitHub, encouraging further research and adoption.[15] The paper is a key example of a broader movement to use LLMs in fuzzing, with related works like

mGPTFuzz for Matter devices and Fuzz4All for compilers appearing in the same timeframe.[18]

The ChatAFL paper provides a compelling vision for the future of automated security testing. A core challenge in the field has always been bridging the gap between human-readable specifications (like API documentation or a protocol RFC) and the machine-executable logic required by a testing tool. Historically, this bridge was built by a human expert who would manually write a test harness or a grammar. ChatAFL demonstrates that an LLM can automate this translation process. In this role, the LLM acts as a "semantic decompiler," translating the unstructured knowledge contained in natural language documents into a structured, machine-usable format like a grammar or a state transition hint. This reframes the role of LLMs in security from being simple "smart mutators" to being powerful "automated specification interpreters." This has profound implications not just for network protocols but for any domain with complex specifications, such as web APIs defined by OpenAPI or Swagger documents.

**Argus: Automated Sink Identification via Interpreter Analysis**

The Argus framework, presented at USENIX Security 2024, addresses a foundational weakness in a wide range of web application security tools: the reliance on incomplete, manually curated lists of sensitive functions, or "sinks".[4] In static and taint analysis, a vulnerability is often detected when tainted user input is tracked to a known sink (e.g., a function that executes a command or writes to an HTML page). Argus posits that because these sink lists are manually maintained, they are inevitably incomplete, causing scanners to miss entire classes of vulnerabilities (i.e., suffer from false negatives).[4]

**Core Idea**

The core contribution of Argus is a novel, generic, and automated method to comprehensively identify injection sinks not by looking at application code, but by performing a deep analysis of the underlying programming language interpreter itself.[4] The key insight is that for a given vulnerability class, such as insecure deserialization or command injection, all exploitable high-level API functions must eventually converge and call one of a very small number of core, low-level functions within the interpreter that actually perform the sensitive operation. Argus calls these deep internal functions "Vulnerability Indicator Functions" (VIFs). By identifying these

VIFs and then performing a reachability analysis, Argus can discover every single API function that can trigger them, thereby creating a near-complete, ground-truth list of sinks.[4]

**Methodology**

Argus implements its approach for the PHP interpreter using a sophisticated hybrid static-dynamic program analysis.

- **Hybrid Call-Graph Construction:** The first step is to build a highly accurate and comprehensive call-graph of the entire PHP interpreter, including its numerous extensions. Argus begins by performing a static analysis of the interpreter's compiled binary files to create an initial call-graph. However, static analysis struggles with indirect calls made via function pointers, which are used extensively in PHP's internal architecture (e.g., for stream wrappers). To solve this, Argus refines the graph using dynamic analysis. It runs the official PHP test suite with tracing enabled (uftrace) to observe which indirect calls are made in practice and adds these missing edges to the graph, resulting in a much more complete model.[4]
- **Vulnerability Indicator Function (VIF) Identification:** Argus identifies the VIFs for the targeted vulnerability classes. For insecure deserialization in PHP, this is the single function php_var_unserialize. For Cross-Site Scripting (XSS), it is php_output_write, which handles writing to the output buffer. For command injection, it is the set of functions that directly or indirectly make an execve system call.[4]
- **Reachability Analysis:** With the complete call-graph and the VIFs identified, Argus performs a graph traversal. It labels all exported PHP API functions as potential sources and then determines which of these sources have a valid execution path to one of the VIFs. Any API with such a path is identified as a potential sink.[4]
- **Automated Validation:** The reachability analysis can produce false positives if an API sanitizes its arguments before passing them to the VIF. To address this, Argus includes an automated validation step. For each potential sink it identifies, it generates a small PHP test snippet that calls the API with a malicious payload. It then executes this snippet and observes the outcome. For example, to validate a deserialization sink, it passes a serialized object with a __wakeup magic method; if the method executes, the sink is confirmed as valid.[19]

### Strengths and Key Features

Argus is not merely an incremental improvement; it is a **foundational tool** that provides a more accurate ground truth for an entire class of security analysis systems. Its primary strength is its **comprehensive discovery** of sinks. For PHP, it found an order of magnitude more deserialization sinks (284 vs. 26 in the best manual list) and doubled the number of known XSS sinks (22 vs. 12).[4] This has a direct and

**proven real-world impact**. When Argus's sink list was integrated into the popular static analysis tool Psalm, it enabled the discovery of 13 previously unknown, high-severity vulnerabilities in WordPress and its plugins, 11 of which were assigned CVEs.[4] While implemented for PHP, its VIF-based methodology is a

**generic principle** that could be applied to other interpreted languages, making it a broadly significant contribution.

### Limitations

The power of Argus comes at the cost of **analysis complexity**. Performing a hybrid static-dynamic analysis of a massive, complex C codebase like the PHP interpreter is a significant engineering feat. The validation step, while useful, is **not a full data-flow analysis**; it primarily checks for direct, unmodified argument passing and could miss more subtle sanitization routines, potentially leaving some false positives in its final list.[19] Finally, the analysis is

**interpreter-version-specific**. As the PHP interpreter evolves, the set of sinks can change—as demonstrated by the dramatic reduction in deserialization sinks from 284 in PHP 7.2 to just 13 in PHP 8.0 due to a security hardening change. This means the analysis must be re-run for different target versions.[4]

### Evaluation and Impact

Argus was evaluated on three major versions of the PHP interpreter (5.6, 7.2, and 8.0) to demonstrate its analysis capabilities and to track the evolution of sinks over time.[4] Its practical impact was compellingly demonstrated by integrating its results with three different types of existing tools—the static analyzers Psalm and RIPS, and the exploit generator FUGIO—and showing that this integration led directly to the discovery of new, real-world vulnerabilities.[4] The work was presented at USENIX Security 2024, and its artifact was made available to the research community.[12]

The development of Argus signals a noteworthy intellectual shift in security research—a move from treating symptoms to addressing the root cause. Many vulnerability scanners focus on improving their detection algorithms (the symptom). Argus, in contrast, asks a more fundamental question: "Why are our tools missing bugs?" It identifies a root cause: the underlying data model (the sink list) is flawed and incomplete. By focusing on creating a tool to generate a provably more complete data model, Argus provides a foundational improvement that elevates the entire ecosystem of tools that depend on it. This suggests that the next major leap in static analysis accuracy may not come from a new taint-tracking algorithm, but from more complete and automatically generated models of the platforms being analyzed.

## Special Report: Fudan University's Contributions to Security Analysis

A targeted investigation into the research output of Fudan University reveals a highly active and impactful program in systems security, application security, and program analysis, directly aligning with the core topics of this report. While a specific paper from 2024 matching the exact keywords "Fudan University," "taint analysis," and "web security" was not identified in the proceedings of the top four conferences, a broader analysis of the university's recent and forthcoming publications demonstrates a clear and sustained focus on these areas, led by prominent research labs such as the Fudan System Software & Security Lab and the Fudan CSR Lab.[20]

The university's research trajectory indicates a strong competency in applying sophisticated program analysis techniques to find real-world vulnerabilities at scale. This is best exemplified by a paper accepted to the NDSS Symposium 2025 and a

remarkable slate of papers accepted to the USENIX Security Symposium 2025.

### NDSS 2025: Empirical Analysis of Android API Misuse

A key publication from Fudan University researchers, including Xin Zhang and Professor Min Yang, is titled "An Empirical Study on Fingerprint API Misuse with Lifecycle Analysis in Real-world Android Apps," accepted to NDSS 2025.[23] This work, while focused on mobile security, is highly relevant as it showcases the university's methodological strengths in automated vulnerability discovery.

The core of this research is a large-scale empirical study that uses static analysis to detect misuses of Android's complex fingerprint authentication APIs (FpAPIs). The researchers developed a static analysis tool built on the Soot and FlowDroid frameworks to analyze app bytecode. The tool identifies four prevalent types of misuse across the entire authentication lifecycle, such as using obsolete APIs and performing inadequate cryptographic validation.[24] The impact of this work is substantial: the analysis of 1,333 real-world Android apps revealed that an alarming 97.15% contained at least one type of misuse. This research led to the responsible disclosure of numerous vulnerabilities, resulting in the assignment of 184 CVEs and 19 CNVD IDs, underscoring the real-world significance of their findings.[24] The authors are clearly affiliated with Fudan University, confirming its active role in top-tier security research.[25]

### USENIX Security 2025: A Strong Focus on Web Security

Looking ahead to 2025, the personal homepage of Professor Yuan Zhang of Fudan University lists an impressive number of papers accepted to the prestigious USENIX Security Symposium, several of which are directly focused on web application security testing.[26] This body of work signals a major and current investment in the field. Key titles include:

- **BACScan:** A tool for the "Automatic Black-Box Detection of Broken-Access-Control Vulnerabilities in Web Applications."
- **XSSky:** A framework for "Detecting XSS Vulnerabilities through Local Path-Persistent Fuzzing."

- A paper on "Effective Directed Fuzzing with Hierarchical Scheduling for Web Vulnerability Detection."
- A paper detailing the "Automatic Detection and Exploitation of Java Web Application Vulnerabilities via Concolic Execution guided by Cross-thread Object Manipulation."

These forthcoming publications clearly demonstrate a deep and active research agenda at Fudan University focused on applying advanced testing methodologies—including black-box analysis, fuzzing, and concolic execution—to discover critical vulnerabilities in web applications.

**Conclusion of Special Report**

While no single paper from 2024 precisely matches the initial query, the evidence strongly indicates that Fudan University is a leading institution in the broader field of automated security analysis. The NDSS 2025 paper on Android security demonstrates a core competency in using program analysis to conduct large-scale empirical studies and find high-impact, real-world vulnerabilities. The upcoming suite of USENIX Security 2025 papers shows this same core competency now being applied directly and prolifically to the domain of web security.

This pattern suggests a research pivot based on established expertise. The labs at Fudan have mastered a certain style of rigorous, automated security analysis and are now directing that capability towards the highly relevant and challenging area of web applications. It is highly probable that the user's inquiry was referencing this well-established reputation and this specific, cutting-edge research direction being pursued by groups like that of Professor Yuan Zhang. The university's active research labs, numerous publications in top venues, and specific focus on web security, fuzzing, and program analysis confirm its status as a key contributor to the field.[20]

## Comparative Analysis and Emerging Trends

The detailed examination of individual frameworks reveals not only their specific contributions but also broader patterns of evolution in the field. By comparing these

state-of-the-art approaches and synthesizing their commonalities and differences, we can identify the overarching trends that are shaping the future of web application security testing.

### Comparative Framework Analysis

To crystallize the distinct characteristics and trade-offs of the analyzed frameworks, the following table provides a side-by-side comparison. This synthesis is a crucial tool for researchers and practitioners, as it clarifies the current landscape and helps in selecting the appropriate conceptual approach for a given security testing challenge. The comparison highlights how progress is not monolithic but rather a multi-dimensional process involving different design choices, each with its own set of strengths and limitations.

| Framework | Methodology | Target Language / Environment | Target Vulnerabilities | Key Strengths | Key Limitations | Conference |
|---|---|---|---|---|---|---|
| **Atropos** | Snapshot-based, feedback-driven grey-box fuzzing [2] | PHP Web Applications | Server-side logic flaws (SQLi, RCE, Object Injection, SSRF) [2] | Handles state effectively; deep coverage via interpreter feedback; zero false positives [2] | PHP-specific; performance depends on snapshotting efficiency [2] | USENIX Sec '24 |
| **Spider-Scents** | Grey-box, database-aware dynamic scanning [3] | Language/ Framework-agnostic (requires database backend) | Stored XSS | Bypasses input validation; extremely high database coverage (79-100%); finds dormant | Requires direct DB access; output requires manual verification to confirm exploitabili | USENIX Sec '24 |

| | | | | | | |
|---|---|---|---|---|---|---|
| | | | | XSS [3] | ty [3] | |
| **ChatAFL** | LLM-guided stateful protocol fuzzing [1] | Network Protocol Implementations (e.g., C/C++) | Protocol implementation flaws, state machine bugs, memory corruption [1] | Automates understanding of specs (RFCs); superior state exploration; requires no manual grammar [1] | Dependent on LLM quality/cost/latency; best for documented, text-based protocols [1] | NDSS '24 |
| **Argus** | Hybrid (static + dynamic) binary analysis of the interpreter [4] | PHP Interpreter | Injection vulnerability sinks (XSS, Deserialization, Command Injection) [4] | Foundational improvement for all taint analysis tools; discovers far more sinks than manual lists; proven real-world impact [4] | Complex analysis; validation is not full data-flow; results are interpreter-version-specific [4] | USENIX Sec '24 |

### Summary of Emerging Trends

The analysis of these frameworks and the broader literature from 2024-2025 points to three dominant and interconnected trends that are defining the next generation of web application security testing.

### Trend 1: The Rise of LLM-Guided Security Testing

The integration of Large Language Models into security testing has evolved from a novelty into a significant research direction. This trend, clearly exemplified by ChatAFL but also visible in a wave of recent papers like mGPTFuzz, KernelGPT, and Fuzz4All [1], represents a qualitative shift in how automation is approached.

The progression is from syntax to semantics. Early AI-in-fuzzing work often focused on learning syntactic structures from sample inputs. In contrast, modern LLM-based approaches are used to infer semantic meaning. They can interpret natural language documentation like RFCs or API guides to generate syntactically *and* semantically valid inputs and test drivers.[1] This allows the fuzzer to engage with the target application in a much more intelligent and productive way.

The most effective models are hybrid. Purely LLM-driven generation can suffer from low validity rates and "hallucinations".[27] The success of

ChatAFL lies in its hybrid architecture, which combines the high-level semantic guidance of an LLM with the battle-tested, low-level mutation and coverage-tracking engine of AFLNet.[16] This suggests the most promising future is not one where AI replaces traditional fuzzers, but one where AI augments them, serving as a powerful "oracle" or "co-pilot" within the fuzzing loop. This principle is being applied across the entire testing lifecycle, from generating fuzz drivers and enriching seed corpuses to guiding state exploration and even automating high-level penetration testing strategies.[1]

**Trend 2: The Dominance of Privileged Hybrid Analysis**

The most impactful recent advances in vulnerability discovery are not coming from purely black-box approaches. Instead, they are the result of frameworks that intelligently fuse dynamic testing with privileged, white-box, or grey-box information. This trend signifies a pragmatic recognition that strict adherence to the black-box model can be a significant impediment to finding deep bugs.

This fusion occurs in multiple ways. We see the combination of static and dynamic analysis, as in Argus, which uses both to construct a complete call-graph of the PHP interpreter, overcoming the individual weaknesses of each method.[4] We also see the elevation of application state to a first-class analysis target.

Spider-Scents makes the database the primary object of analysis [3], while

Atropos uses VM-like snapshots to precisely manage and revert application state.[2] This demonstrates a clear understanding that for modern, complex web applications, analyzing the application's internal

*state* is just as critical as analyzing its code. This "privileged" access allows tools to bypass superficial defenses and gain a much deeper and more accurate view of the application's true security posture.

**Trend 3: A Sharpened Focus on Deep, Server-Side Vulnerabilities**

As the field matures, the research focus is shifting away from easily detectable, low-hanging fruit towards more complex and often more severe vulnerability classes. The state-of-the-art tools presented in 2024 are not designed to find simple reflected XSS; they are specialized instruments for hunting deep, server-side bugs.

The targets of these advanced frameworks include Stored XSS (Spider-Scents), Server-Side Request Forgery (Atropos, Artemis), PHP Object Injection (Atropos), and insecure deserialization (Argus).[2] These vulnerabilities are fundamentally harder to find because their exploitation often depends on complex data flows, multi-step interactions, state transitions, and intricate server-side logic.

The rise of specialized tooling like Atropos (for PHP logic) and Argus (for PHP sinks) indicates that the era of the generic, one-size-fits-all DAST scanner may be waning. Achieving high precision and recall for these complex bug classes requires deep, domain-specific knowledge of the target platform. The next generation of security tools is encoding this deep knowledge directly into their analysis engines, enabling them to find vulnerabilities that generic scanners would invariably miss.

## Conclusion and Future Directions

The research landscape of 2024 and 2025 marks a clear inflection point for web application security testing. The field is undergoing a period of rapid and

transformative innovation, characterized by the assimilation of powerful techniques from adjacent domains, most notably systems security and artificial intelligence. The most impactful advancements are no longer focused on building incrementally better black-box scanners. Instead, they are driven by a fundamental rethinking of the testing process itself, leveraging privileged access and deep semantic understanding to unearth complex, high-impact vulnerabilities that were previously beyond the reach of automated tools. The state of the art is defined by hybrid frameworks that fuse the speed of dynamic analysis with the contextual depth of static analysis, augmented by the reasoning capabilities of Large Language Models.

The limitations of these pioneering frameworks, however, illuminate a clear path for future research. Several key challenges remain open, and addressing them will be critical for the continued advancement of web application security.

- **Generalizing Privileged Analysis:** Many of the most powerful new techniques, such as the interpreter-level feedback in Atropos and the VIF-based sink discovery in Argus, are currently language-specific (primarily for PHP).[2] A major avenue for future work is the development of generalized frameworks for privileged, in-process dynamic analysis. This could involve creating standardized instrumentation hooks or APIs for common language runtimes (e.g., the Java Virtual Machine, Node.js's V8 engine, Python's CPython interpreter) that would allow security tools to query internal application state in a language-agnostic manner.
- **Tackling Asynchronous and Client-Side Complexity:** The majority of current state-of-the-art tools still perform best on traditional, synchronous, server-rendered web applications. A significant and largely unsolved challenge is the development of effective state-management and analysis techniques for highly asynchronous, client-side-heavy Single Page Applications (SPAs) built with frameworks like React, Angular, and Vue.js. Fuzzing these applications requires not only managing server-side state but also the complex, event-driven state of the client-side UI, which remains a formidable open problem.
- **Scalable and Trustworthy LLM-based Testing:** The use of LLMs in security testing is a powerful trend, but it comes with significant challenges related to cost, latency, and reliability.[1] Future research must focus on making these approaches more practical and trustworthy. This could involve exploring the use of smaller, domain-specific models that are fine-tuned for security tasks, developing more robust prompting strategies to minimize "hallucinations" and incorrect guidance, and potentially creating formal methods to verify or constrain the output of an LLM before it is used to direct a fuzzer.
- **Automating the Full Vulnerability Lifecycle:** While detection tools are

becoming increasingly automated and powerful, significant manual effort is still required for vulnerability verification (as in Spider-Scents) and for root cause analysis and remediation.[10] A promising direction for future research is to close this loop by integrating automated detection frameworks with emerging technologies for automated program repair and detailed exploit generation. A system that can not only find a bug but also pinpoint its root cause in the source code and suggest or even generate a patch would represent a true end-to-end solution, dramatically reducing the time from discovery to remediation.

## Bibliography


[12] USENIX Security '24 Technical Sessions. (2024).

*List of accepted papers on web security, fuzzing, static analysis, dynamic analysis, taint analysis, symbolic execution*. Retrieved from https://www.usenix.org/conference/usenixsecurity24/technical-sessions

[14] NDSS Symposium 2024. (2024).

*List of accepted papers on web security, fuzzing, static analysis, dynamic analysis, taint analysis, symbolic execution*. Retrieved from https://www.ndss-symposium.org/ndss2024/accepted-papers/

[8] Güler, E., Schumilo, S., Schloegel, M., Bars, N., Görz, P., Xu, X., Kaygusuz, C., & Holz, T. (2024).

*Atropos: Effective Fuzzing of Web Applications for {Server-Side} Vulnerabilities*. 33rd USENIX Security Symposium (USENIX Security 24).

[3] Olsson, E., Eriksson, B., Doupé, A., & Sabelfeld, A. (2024).

*Spider-Scents: Grey-box Database-aware Web Scanning for Stored XSS*. USENIX Security '24. Retrieved from https://www.usenix.org/system/files/sec24summer-prepub-286-olsson.pdf

[24] Zhang, X., Zhang, X., Liu, Z., Zhao, B., Yang, Z., & Yang, M. (2025).

*An Empirical Study on Fingerprint API Misuse with Lifecycle Analysis in Real-world*

*Android Apps*. Network and Distributed System Security (NDSS) Symposium 2025. Retrieved from https://www.ndss-symposium.org/wp-content/uploads/2025-699-paper.pdf

[23] NDSS Symposium 2025. (2025).

*Accepted Papers*. Retrieved from https://www.ndss-symposium.org/ndss2025/

[27] Jiang, Y., et al. (2024).

*When Fuzzing Meets LLMs: Challenges and Opportunities*. arXiv:2503.00795v1. Retrieved from https://arxiv.org/html/2503.00795v1

[8] Schloegel, M., et al. (2024).

*Google Scholar Citations for M. Schloegel*. Retrieved from Google Scholar.

[2] Güler, E., et al. (2024).

*Atropos: Effective Fuzzing of Web Applications for Server-Side Vulnerabilities*. USENIX Security '24. Retrieved from https://www.usenix.org/system/files/sec23winter-prepub-167-guler.pdf

[6] Güler, E., et al. (2024).

*Atropos: Effective Fuzzing of Web Applications for Server-Side Vulnerabilities (Slides)*. USENIX Security '24. Retrieved from https://www.usenix.org/system/files/usenixsecurity24_slides-guler.pdf

[30] Wessels, M., et al. (2025).

*Artemis: Toward Accurate Detection of Server-Side Request Forgeries*. arXiv:2502.21026. Retrieved from https://arxiv.org/pdf/2502.21026

[29] Liu, Z., et al. (2024).

*AutoPT: An LLM-driven Agent for Automated Penetration Testing*. arXiv:2411.01236. Retrieved from https://arxiv.org/pdf/2411.01236

[3] Olsson, E., Eriksson, B., Doupé, A., & Sabelfeld, A. (2024).

*Spider-Scents: Grey-box Database-aware Web Scanning for Stored XSS*. USENIX Security '24. Retrieved from

https://www.usenix.org/system/files/usenixsecurity24-olsson.pdf

[10] Olsson, E., Eriksson, B., Doupé, A., & Sabelfeld, A. (2024).

*Spider-Scents: Grey-box Database-aware Web Scanning for Stored XSS (Artifact Appendix)*. USENIX Security '24. Retrieved from https://www.usenix.org/system/files/usenixsecurity24-appendix-olsson.pdf

[9] Olsson, E. (2024).

*Scanning for Vulnerabilities in Modern Web Applications and Browser Extensions*. Chalmers University of Technology. Retrieved from https://research.chalmers.se/publication/546193/file/546193_Fulltext.pdf

[4] Jahanshahi, R., & Egele, M. (2024).

*Argus: All your (PHP) Injection-sinks are belong to us*. USENIX Security '24. Retrieved from https://www.usenix.org/system/files/usenixsecurity24-jahanshahi.pdf

[19] Jahanshahi, R., & Egele, M. (2024).

*Argus: All your (PHP) Injection-sinks are belong to us (Artifact Appendix)*. USENIX Security '24. Retrieved from https://www.usenix.org/system/files/usenixsecurity24-appendix-jahanshahi.pdf

[1] Meng, R., Mirchev, M., Böhme, M., & Roychoudhury, A. (2024).

*Large Language Model guided Protocol Fuzzing*. NDSS '24. Retrieved from https://abhikrc.com/pdf/NDSS24.pdf

[17] Meng, R., et al. (2024).

*Large Language Model guided Protocol Fuzzing (Slides)*. NDSS '24. Retrieved from https://www.ndss-symposium.org/wp-content/uploads/2024-556-slides.pdf

[28] Zhang, Y., et al. (2025).

*KernelGPT: Enhanced Kernel Fuzzing via Large Language Models*. ASPLOS '25. Retrieved from https://yangchenyuan.github.io/files/ASPLOS25-KernelGPT.pdf

[11] Spider-Scents Team. (2024).

*dbfuzz GitHub Repository*. Retrieved from https://github.com/Spider-Scents/dbfuzz

[13] USENIX Security '24 Artifact Evaluation Committee. (2024).

*Results*. Retrieved from https://secartifacts.github.io/usenixsec2024/results

[24] Zhang, X., et al. (2025).

*An Empirical Study on Fingerprint API Misuse with Lifecycle Analysis in Real-world Android Apps*. NDSS '25. Retrieved from https://www.ndss-symposium.org/wp-content/uploads/2025-699-paper.pdf

[25] Zhang, X. (2025).

*An Empirical Study on Fingerprint API Misuse with Lifecycle Analysis in Real-world Android Apps (Slides)*. NDSS '25. Retrieved from https://www.ndss-symposium.org/wp-content/uploads/9A-s0699-zhang.pdf

[5] IEEE S&P 2024. (2024).

*Accepted Papers - First Cycle*. Retrieved from https://sp2024.ieee-security.org/program-papers.html

[1] Meng, R., Mirchev, M., Böhme, M., & Roychoudhury, A. (2024).

*Large Language Model guided Protocol Fuzzing*. NDSS '24. Retrieved from https://abhikrc.com/pdf/NDSS24.pdf

[17] Meng, R., et al. (2024).

*Large Language Model guided Protocol Fuzzing (Slides)*. NDSS '24. Retrieved from https://www.ndss-symposium.org/wp-content/uploads/2024-556-slides.pdf

[28] Zhang, Y., et al. (2025).

*KernelGPT: Enhanced Kernel Fuzzing via Large Language Models*. ASPLOS '25. Retrieved from https://yangchenyuan.github.io/files/ASPLOS25-KernelGPT.pdf

[16] ChatAFLndss. (2024).

*ChatAFL GitHub Repository*. Retrieved from https://github.com/ChatAFLndss/ChatAFL

[15] Böhme, M. (2024).

*Large Language Model guided Protocol Fuzzing*. NDSS '24. Retrieved from https://mboehme.github.io/paper/NDSS24.pdf

[18] Ma, X., et al. (2024).

*mGPTFuzz: Fuzzing Matter Devices with Large Language Models*. USENIX Security '24. Retrieved from https://www.usenix.org/system/files/usenixsecurity24-ma-xiaoyue.pdf

[20] Fudan CS Labs. (2024).

*List of Fudan University Computer Science Labs*. Retrieved from https://fudan-cs-labs.vercel.app/

[26] Zhang, Y. (2024).

*Yuan Zhang's Publications*. Retrieved from https://yuanxzhang.github.io/

[21] Fudan System Software & Security Lab. (2024).

*Lab Homepage*. Retrieved from https://secsys.fudan.edu.cn/

[22] School of Computer Science, Fudan University. (2024).

*Faculty Profiles*. Retrieved from https://cs.fudan.edu.cn/7b/e4/c24781a490468/page.htm

[7] Bulekov, A., Liu, Q., Egele, M., & Payer, M. (2024).

*HYPERPILL: Fuzzing for Hypervisor-bugs by Leveraging the Hardware Virtualization Interface*. USENIX Security '24.

**Works cited**


1. Large Language Model guided Protocol Fuzzing - Abhik Roychoudhury, accessed June 13, 2025, https://abhikrc.com/pdf/NDSS24.pdf
2. Atropos: Effective Fuzzing of Web Applications for Server ... - USENIX, accessed June 13, 2025, https://www.usenix.org/system/files/sec23winter-prepub-167-guler.pdf
3. Spider-Scents: Grey-box Database-aware Web Scanning ... - USENIX, accessed June 13, 2025, https://www.usenix.org/system/files/sec24summer-prepub-286-olsson.pdf
4. Argus: All your (PHP) Injection-sinks are belong to us. | USENIX, accessed June 13, 2025, https://www.usenix.org/system/files/usenixsecurity24-jahanshahi.pdf
5. Accepted Papers - IEEE Symposium on Security and Privacy 2024, accessed June

13, 2025, https://sp2024.ieee-security.org/program-papers.html
6. Atropos: Effective Fuzzing of Web Applications for Server-Side Vulnerabilities - USENIX, accessed June 13, 2025, https://www.usenix.org/system/files/usenixsecurity24_slides-guler.pdf
7. 复旦大学白泽智能团队, accessed June 13, 2025, https://whitzard-ai.github.io/
8. Nils Bars - Google Scholar, accessed June 13, 2025, https://scholar.google.de/citations?user=lyz2qNcAAAAJ&hl=de
9. Spidering the Modern Web: Securing the Next Generation of Web Sites and Browser Extensions - Chalmers Research, accessed June 13, 2025, https://research.chalmers.se/publication/546193/file/546193_Fulltext.pdf
10. Spider-Scents: Grey-box Database-aware Web Scanning for Stored XSS - USENIX, accessed June 13, 2025, https://www.usenix.org/system/files/usenixsecurity24-appendix-olsson.pdf
11. Spider-Scents/dbfuzz - GitHub, accessed June 13, 2025, https://github.com/Spider-Scents/dbfuzz
12. USENIX Security '24 | USENIX, accessed June 13, 2025, https://www.usenix.org/conference/usenixsecurity24
13. Results - Security Research Artifacts, accessed June 13, 2025, https://secartifacts.github.io/usenixsec2024/results
14. NDSS Symposium 2024 Accepted Papers - NDSS Symposium, accessed June 13, 2025, https://www.ndss-symposium.org/ndss2024/accepted-papers/
15. Large Language Model guided Protocol Fuzzing - Marcel Boehme, accessed June 13, 2025, https://mboehme.github.io/paper/NDSS24.pdf
16. ChatAFLndss/ChatAFL: Large Language Model guided Protocol Fuzzing (NDSS'24) - GitHub, accessed June 13, 2025, https://github.com/ChatAFLndss/ChatAFL
17. Large Language Model guided Protocol Fuzzing, accessed June 13, 2025, https://www.ndss-symposium.org/wp-content/uploads/2024-556-slides.pdf
18. From One Thousand Pages of Specification to Unveiling Hidden Bugs: Large Language Model Assisted Fuzzing of Matter IoT Devices - USENIX, accessed June 13, 2025, https://www.usenix.org/system/files/usenixsecurity24-ma-xiaoyue.pdf
19. Argus: All your (PHP) Injection-sinks are belong to us. | USENIX, accessed June 13, 2025, https://www.usenix.org/system/files/usenixsecurity24-appendix-jahanshahi.pdf
20. Fudan CS Labs, accessed June 13, 2025, https://fudan-cs-labs.vercel.app/
21. 系统软件与安全实验室 - 复旦大学, accessed June 13, 2025, https://secsys.fudan.edu.cn/
22. 网络空间安全一级学科3：智能系统安全 - 复旦大学计算机学院, accessed June 13, 2025, https://cs.fudan.edu.cn/7b/e4/c24781a490468/page.htm
23. Network and Distributed System Security (NDSS) Symposium 2025, accessed June 13, 2025, https://www.ndss-symposium.org/ndss2025/
24. An Empirical Study on Fingerprint API Misuse with Lifecycle Analysis ..., accessed June 13, 2025, https://www.ndss-symposium.org/wp-content/uploads/2025-699-paper.pdf
25. An Empirical Study on Fingerprint API Misuse with Lifecycle Analysis, accessed

June 13, 2025, https://www.ndss-symposium.org/wp-content/uploads/9A-s0699-zhang.pdf
26. Yuan Zhang, accessed June 13, 2025, https://yuanxzhang.github.io/
27. Towards Reliable LLM-Driven Fuzz Testing: Vision and Road Ahead - arXiv, accessed June 13, 2025, https://arxiv.org/html/2503.00795v1
28. KernelGPT: Enhanced Kernel Fuzzing via Large Language Models - Chenyuan Yang, accessed June 13, 2025, https://yangchenyuan.github.io/files/ASPLOS25-KernelGPT.pdf
29. AutoPT: How Far Are We from the End2End Automated Web Penetration Testing? - arXiv, accessed June 13, 2025, https://arxiv.org/pdf/2411.01236
30. Artemis: Toward Accurate Detection of Server-Side Request Forgeries through LLM-Assisted Inter-procedural Path-Sensitive Taint A - arXiv, accessed June 13, 2025, https://arxiv.org/pdf/2502.21026