Holistic Concolic Execution for Dynamic Web Applications via Symbolic Interpreter Analysis

Penghui Li[1], Wei Meng[2], Mingxue Zhang[3], Chenlin Wang[2], Changhua Luo[2]

[1]Zhongguancun Laboratory [2]The Chinese University of Hong Kong [3]Zhejiang University

Symbolic Execution for Web Applications

Symbolic execution enables many security tasks for web security such as vulnerablity detection and exploit generation.

Key Challenge—Multilingual Nature

Web applications comprise multiple components implemented in different programming languages. PHP-based web applications include application logic implemented in PHP and basic functionality implemented in C.

Prior Solutions—Modeling

Convert both components into a common representation, e.g., SMT formulas.

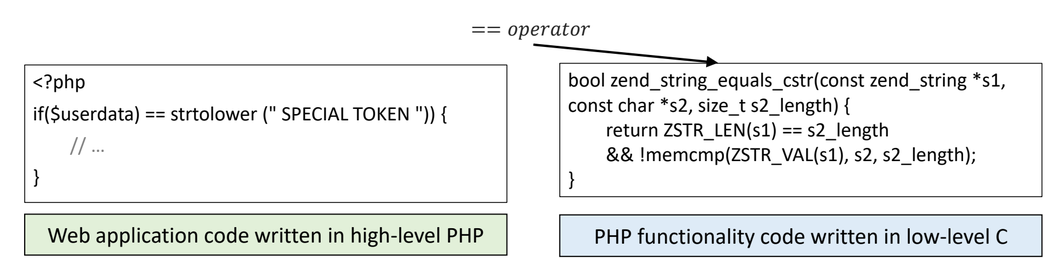
Limitations

- (1) Incomplete language syntax support
- (2) Excessive engineering efforts

SIA: Symbolic Interpreter Analysis

Symbolically analyze the interpreter code to indirectly analyze the web application symbolically.

- (1) Language interpreter embeds the complete language functionality and syntax.
- (2) Leverage existing mature symbolic execution engines to analyze the interpreter, implemented in static, compiled languages like C/C++



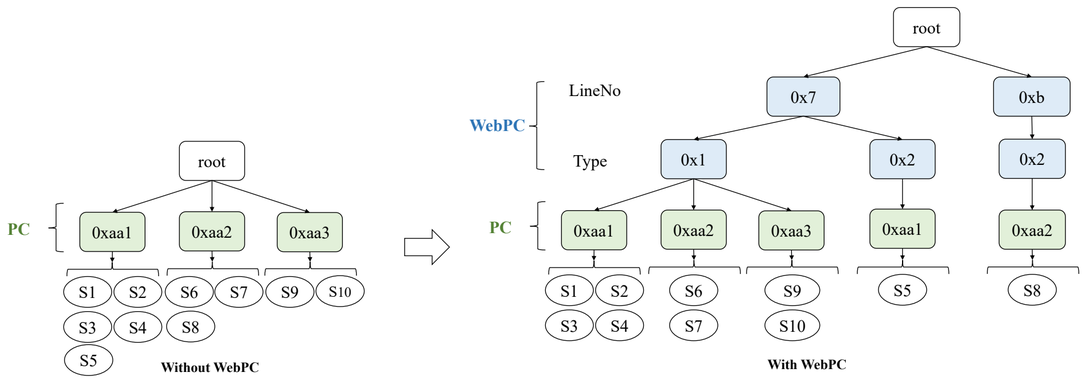
Technical Design

▶ The underlying engine is intrinsically designed to analyze interpreter code and expand interpreter code coverage.

Beyond interpreter state (PC), define web application state *WebPC*:

$$WebPC(ins) = LineNo_{web}(ins) << 8 \mid Type(ins)$$

Expose WebPC to the underlying engine and guide exploration towards diverse WebPC states.



▶ Web applications interact with external environments, including HTTP servers and database systems.

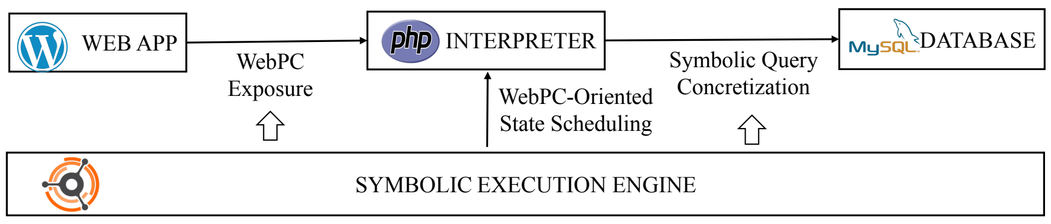
Leverage common gateway interface to avoid HTTP servers; concretize symbolic data during database interactions.

Implementation

We realized the core part of SIA atop S2E using 3K LoC, which is applicable to all interpreted languages.

SymPHP: Prototype for PHP-Based Web Applications

- Took **500 LoC** and **two person weeks** to support PHP 7 and PHP 8.
- In comparison, a recent work used 20K LoC and over 13 person-months to support only PHP 7.
- Seamlessly supported **all PHP syntaxes**.



Evaluation

Based on the initial inputs generated by the SOTA fuzzer Witcher [1], we apply SymPHP to 17 web applications.

App	% Coverage	# Known Vul.	# Detected
WordPress	13.37	384,394	α
MediaWiki	2.17	1,392	β
Drupal	3.14	83,742	δ
OpenEMR	7.59	974	γ
WebChess			
phpMyAdmin			

Code Coverage

- Achieved an average code coverage of 51.57%.
- Further advanced Witcher by 46.46%.

Vulnerability Detection

- Successfully detected 77.23% of ground-truth vulnerabilities.
- Outperformed prior solutions with significantly lower FNR.

Application: Hybrid Fuzzing

We integrate SymPHP with the SOTA fuzzer Witcher [1].

- Improved the code coverage by up to 85.71%, compared with the standalone fuzzer.
- Detected **10 previously unknown vulnerabilities**.

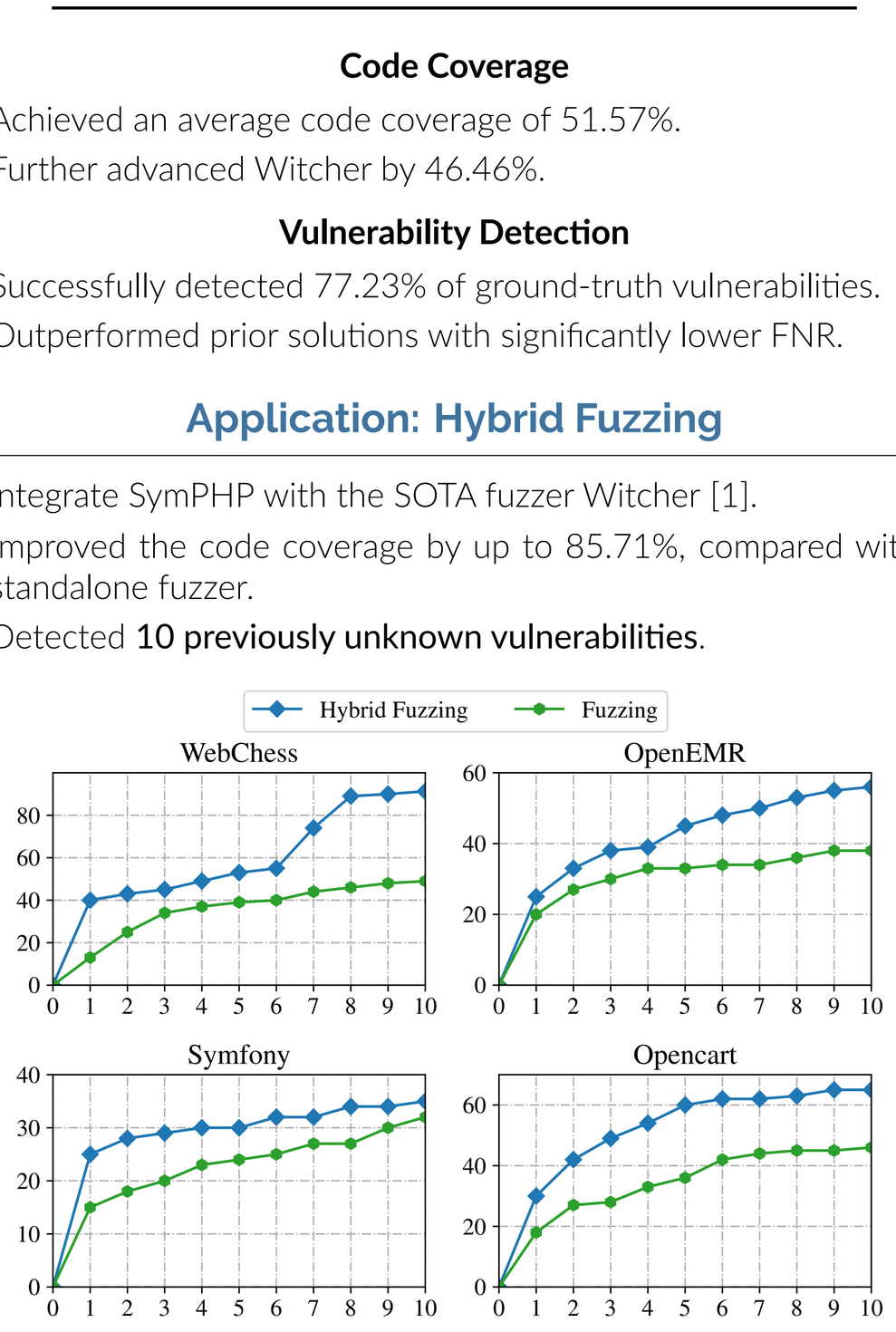


Figure 1. Code coverage (%) over time (hour).

References

[1] Erik Trickel, Fabio Pagani, Chang Zhu, Lukas Dresel, Giovanni Vigna, Christopher Kruegel, Ruoyu Wang, Tiffany Bao, Yan Shoshitaishvili, and Adam Doupé. Toss a fault to your witcher: Applying grey-box coverage-guided mutational fuzzing to detect sql and command injection vulnerabilities. In *IEEE S&P 2023*.