Semantic Debugging

Martin Eberlein[1] Marius Smytzek[2] Dominic Steinhöfel[2] Lars Grunske[1] Andreas Zeller[2]

Abstract: Why does my program fail? We present an automated technique for determining failure causes and conditions using logical properties over input elements: "The program fails if and only if int(⟨*length*⟩) > len(⟨*payload*⟩) holds, i.e., the given ⟨*length*⟩ is larger than the ⟨*payload*⟩ length." Our AVICENNA prototype uses modern techniques for inferring properties of passing and failing inputs and validating and refining hypotheses by having a constraint solver generate supporting test cases. AVICENNA produces crisp and expressive diagnoses close to those of human experts.

Keywords: Automated Debugging; Testing; Behavior Explanation

1 Introduction

When software fails, it is necessary to *debug* it—find the error in the code that causes the failure and fix it. Before delving into the code, one must identify the *circumstances* under which the failure occurs. Such circumstances provide essential hints on how and where to fix the bug, offer insights into the problem's severity, and aid in producing *exact fixes*.

Let us illustrate the role of failure circumstances by referring to the well-known *Heartbleed* problem. Between 2012 and 2014, TLS servers using OpenSSL were vulnerable to the Heartbleed attack, where an attacker could extract internal server memory contents. The vulnerability was in the *Heartbeat* TLS extension, where a client checks if a server is alive. Fig. 1 shows the elements, i.e., the format, of a Heartbeat client request and server response. The client sends a `0x1` byte, a specified length (*payload length*), a given payload of that length, and random padding; the server responds with a `0x2` byte and—normally—the same payload. An attacker would send a payload length of, say, 4,000, and a shorter payload like `"Hello"`. The server would then return `"Hello"` followed by 3,995 bytes from its memory.

Fixing the problem requires understanding the *circumstances* under which it occurs: "The given payload length differs from the payload's actual length." We present AVICENNA [Eb23], a *precise, general, and extensible* approach to automatically determine such failure circumstances. Using AVICENNA, one can quickly derive the correct failure constraint for the Heartbleed bug: $\text{int}(\langle \textit{length} \rangle) > \text{len}(\langle \textit{payload} \rangle)$.

[1] Humboldt-Universität zu Berlin, Berlin {martin.eberlein, grunske}@informatik.hu-berlin.de
[2] CISPA Helmholtz Center for Information Security, Saarbrücken
{marius.smytzek, dominic.steinhoefel, zeller}@cispa.de

doi:10.18420/sw2024_44

⟨*heartbeat-request*⟩ ::= '`0x1`' ⟨*length*⟩ ⟨*payload*⟩ ⟨*padding*⟩
⟨*heartbeat-response*⟩ ::= '`0x2`' ⟨*length*⟩ ⟨*payload*⟩ ⟨*padding*⟩
⟨*length*⟩ ::= ⟨*int*⟩ ⟨*payload*⟩ ::= ϵ | ⟨*byte*⟩ ⟨*payload*⟩ ⟨*padding*⟩ ::= ϵ | ⟨*byte*⟩ ⟨*padding*⟩

Fig. 1: Syntax of TLS Heartbeat exchanges

2 Mining Failure Constraints

AVICENNA takes a program, an input grammar, and *failing* inputs. It decomposes each input into syntactical constituents based on the grammar. The resulting feature vectors train a machine-learning model associating the failure with grammar *nonterminal symbols.* To determine the most relevant nonterminals, AVICENNA employs SHAP, a method in explainable AI that quantifies each feature's contribution to a prediction. In the case of Heartbleed, it pinpoints ⟨*length*⟩ and ⟨*payload*⟩ as key contributors.

AVICENNA learns failures circumstances expressed in the ISLa [SZ22] language via *pattern matching* using *ISLearn* [SZ22]. The pattern matching process is streamlined by focusing only on previously determined input elements (such as ⟨*length*⟩ and ⟨*payload*⟩). An initial hypothesis might be $\text{len}(\langle payload \rangle) > 6$. To refine these hypotheses, AVICENNA generates new inputs based on the most discriminating constraints between failing and passing inputs. AVICENNA generates inputs from *original and negated constraints* using the ISLa fuzzer, labeling them *failing*/*passing* based on program feedback. This iterative process, a *feedback loop*, refines the constraints further. As a result, within just three iterations, AVICENNA is able to accurately produce the constraint: $\text{int}(\langle length \rangle) > \text{len}(\langle payload \rangle)$.

3 Evaluation

AVICENNA was evaluated against 24 bugs, using expert diagnoses from the DBGBench study. The key findings are: (1) AVICENNA's failure circumstances closely match those determined by human experts, often sharing identical semantics. (2) Diagnoses from AVICENNA are only one-eighth the length of those from ALHAZEN, with comparable precision and higher recall. (3) The feedback loop in AVICENNA is essential for precise failure circumstances, outperforming ISLearn in speed and minimizing invariant candidates, yielding optimal solutions.

Bibliography

[Eb23] Eberlein, Martin; Smytzek, Marius; Steinhöfel, Dominic; Grunske, Lars; Zeller, Andreas: Semantic Debugging. In: ESEC/FSE'23. 2023.

[SZ22] Steinhöfel, Dominic; Zeller, Andreas: Input Invariants. In: ESEC/FSE'22. 2022.