AMRITA VISHWA VIDYAPEETHAM
DEEMED TO BE UNIVERSITY UNDER SECTION 3 OF UGC ACT, 1956

AI-Augmented Efficient, Priority-Based Hybrid Fuzzer for Vulnerability Discovery

STUDENT: THARUNADITYA ANUGANTI

MENTOR: DR. SRIRAM SANKARAN

M.TECH CYBERSECURITY SYSTEMS AND NETWORKS

Overview

- This project builds a system to find security bugs in software more efficiently by combining several methods of AI
- It uses a coverage-guided fuzzer that creates test cases to explore the program, and a symbolic execution engine that helps solve complex program checks that are hard to reach by simple fuzzing.
- An AI agent watches how well these methods are working and decides:
  - When to fuzz,
  - When to use symbolic execution,
  - How to focus on the most likely bug-prone parts of the program.
- The system learns to prioritize efforts on code regions that are more likely to contain problems, saving time and computing power.
- It also uses AI to generate smart test inputs for complex input formats.
- By combining these parts, the system finds bugs faster with less wasted effort compared to traditional fuzzing tools.

Introduction

- Software is becoming more complex, and security bugs can cause serious problems.
- Traditional fuzzing tools often spend a lot of time testing parts of the program that are not likely to have bugs.
- Some bugs are hidden behind complex checks and are hard to find with basic fuzzing methods.
- There is a need for a system that can focus testing on the parts of the program most likely to have bugs, making the process faster and more effective.
- This project aims to build such a system by combining fuzzing, symbolic execution, and artificial intelligence to guide testing in a smart way.

Problem Statement

- Current hybrid fuzzers use fixed rules to switch between fuzzing and symbolic execution, which is not efficient.
- There is no adaptive system that prioritizes testing based on how likely different parts of the program are to have vulnerabilities.
- This project addresses these issues by building an AI-driven hybrid fuzzer that focuses resources on the most promising code regions to find bugs more effectively and efficiently.

Literature Review Introduction

- Fuzzing is a widely used technique to find software vulnerabilities by providing unexpected or malformed inputs to a program.
- Over time, fuzzing methods have evolved from simple random input generation to more advanced techniques involving coverage guidance, symbolic execution, and machine learning.
- Recent research combines hybrid fuzzing (fuzzing + symbolic execution) with AI methods like reinforcement learning to improve efficiency and bug discovery rates.
- Despite many advances, current systems often lack adaptive resource management and semantic vulnerability prioritization.
- This review summarizes key recent works that relate to our project, highlighting their strengths and the gaps our system aims to fill.

Title: BertRLFuzzer: A BERT and Reinforcement Learning Based Fuzzer

Journal/Conference: AAAI Conference on Artificial Intelligence

Authors: P. Jha, J. Scott, J. S. Ganeshna, M. Singh, V. Ganesh

Year: 2024

Citations: Low/Recent

Google h5-index: 220 (AAAI)

Advantages: Combines BERT transformer model with RL for grammar-aware mutation and attack vector generation.

Limitations: Focuses primarily on web application fuzzing; no symbolic execution integration.

Gap: Not a true hybrid fuzzer; lacks semantic vulnerability prioritization for general software.

Title: GDFuzz: An efficient directed fuzzing method based on XAI

Authors: K. Zhang, S. Liu, W. Li, X. Xiao

Venue: Journal of Systems and Software (JSS)

Year: 2025

Impact Factor: 3.8 (2024)

Advantages:

Integrates explainable AI (XAI) to direct fuzzing at likely-vulnerable code portions, boosting bug-finding effectiveness.

Limitations:

Not a true hybrid (lacks symbolic execution), and does not factor in overall resource constraints or RL-based adaptive scheduling.

Gap:

Does not offer comprehensive adaptive, multi-component orchestration for resource efficiency.

Title: LLAMA: Multi-Feedback Smart Contract Fuzzing Framework with LLM-Guided Seed Generation

Journal/Conference: arXiv preprint

Authors: K. Gai, H. Liang, J. Yu, L. Zhu, D. Niyato

Year: 2024

Citations: N/A (too recent)

Google h5-index: N/A (preprint)

Advantages: Uses LLMs for semantic seed generation; incorporates multi-feedback optimization and hybrid fuzzing.

Limitations: Domain-specific to smart contracts; high computational overhead from LLM usage.

Gap: Not applicable to general software; lacks resource-efficient prioritization for diverse targets.

Title: SyML: Guiding Symbolic Execution Toward Vulnerable States Through Pattern Learning

Journal/Conference: International Symposium on Research in Attacks, Intrusions and Defenses (RAID)

Authors: N. Ruaro, L. Dresel, K. Zeng, T. Bao, M. Polino, A. Continella, S. Zanero, C. Kruegel, G. Vigna

Year: 2021

Citations: 15

Google h5-index: 34 (RAID)

Advantages: First to use supervised ML to guide symbolic execution toward vulnerable program states using pattern learning.

Limitations: Focuses only on symbolic execution side; no comprehensive hybrid fuzzing orchestration.

Gap: No RL-based adaptive control or multi-modal feedback integration.

Title: Efficient Hybrid Fuzzing Through Conservative Constraint Debloating

Authors: X. Mi, Z. Li, X. Sun, X. Wang, T. Wei

Venue: International Symposium on Research in Attacks, Intrusions and Defenses (RAID)

Year: 2021

Citations: 15

Google h5-index: RAID (34)

Advantages:

Improved efficiency in hybrid fuzzing by minimizing symbolic execution efforts via constraint reduction, enabling greater speed and scalability.

Limitations:

Still coverage-centric; does not use semantic vulnerability ranking or adaptive RL for deciding when/where to focus resources.

Gap: No resource-aware or vulnerability-prioritized control; lacks ML-guided adaptive decision-making.

Title: Driller: Augmenting Fuzzing Through Selective Symbolic Execution

Authors: N. Stephens, J. Grosen, C. Salls, A. Dutcher, R. Wang, J. Corbetta, Y. Shoshitaishvili, C. Kruegel, G. Vigna, R. Sekar

Venue: Network and Distributed System Security Symposium (NDSS)

Year: 2016

Citations: 900

Google h5-index: 91

Advantages: Pioneered hybrid fuzzing, effectively combining coverage-guided fuzzing with selective symbolic execution to bypass input-dependent checks and reach more program states.

Limitations: Resource allocation and decision-making are based on static, pre-defined heuristics; no dynamic prioritization or ML integration.

Gap: No semantic feedback or AI-guided resource prioritization; inefficient on large-scale, resource-constrained tasks.

Title: Reinforcement Learning-Based Fuzzing Technology

Journal/Conference: Various (International Conference on Innovative Mobile and Internet Services, ScienceDirect)

Authors: Z. Zhang, B. Cui, C. Chen

Year: 2020-2025

Citations: 5

Google h5-index: Variable by venue

Advantages: Applies RL (DDPG/DQN algorithms) to guide fuzzing decisions and mutation strategies for better edge coverage.

Limitations: Focuses on single-component RL fuzzing; no symbolic execution or semantic vulnerability integration.

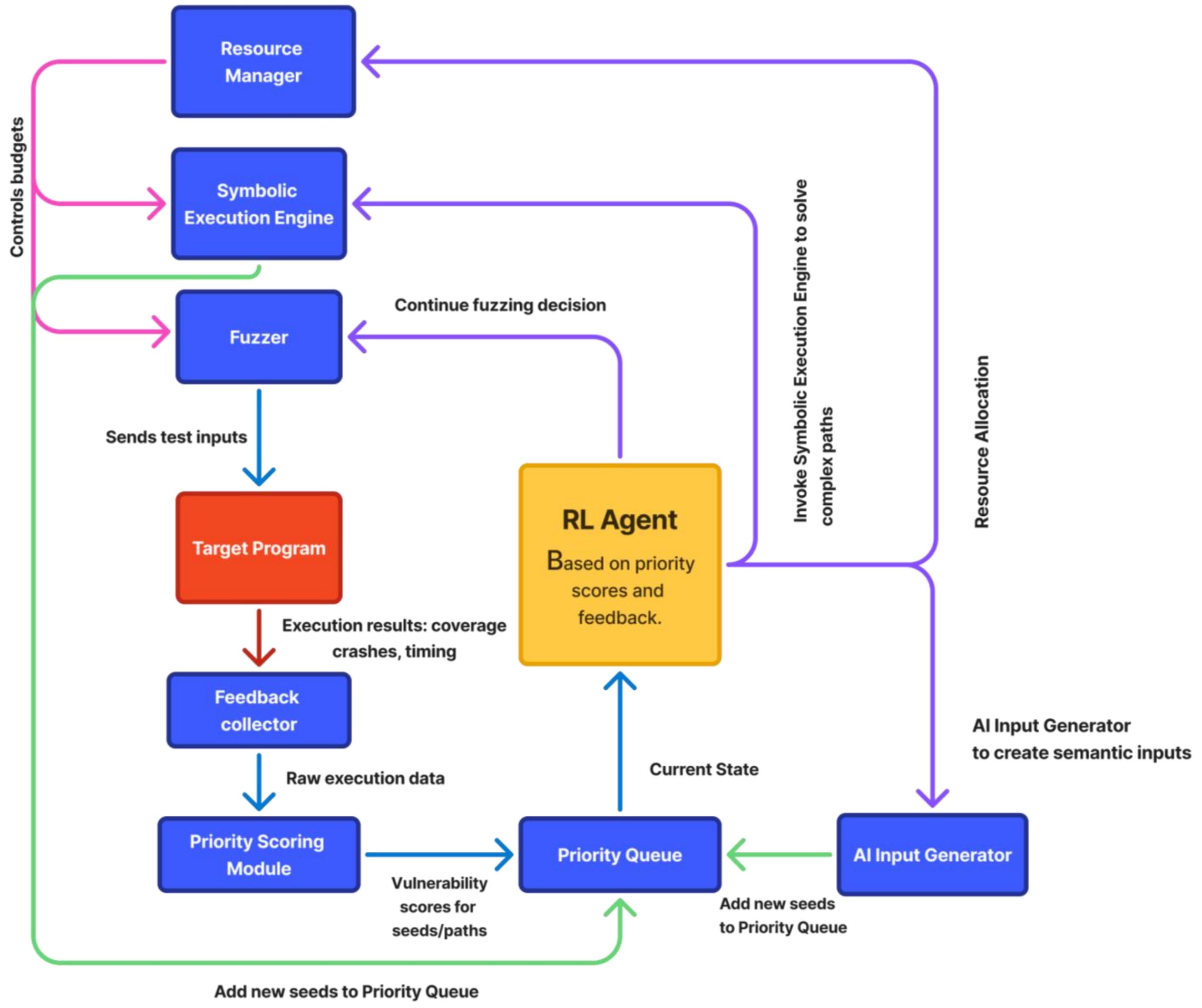
Gap: Lacks hybrid fuzzing capabilities and comprehensive resource management.

The Plan:

What is your plan?

- Build a hybrid fuzzing system combining:
    1. A coverage-guided fuzzer that generates and mutates inputs.
    2. A symbolic execution engine to solve complex path constraints.
    3. A reinforcement learning (RL) agent to dynamically control and coordinate fuzzing and symbolic execution.
- Develop a semantic vulnerability scoring module that analyzes program behavior and assigns priority scores to code regions and seeds.
- Use the RL agent to:
    1. Prioritize testing efforts based on vulnerability likelihood.
    2. Adapt resource allocation dynamically to focus on high-priority areas.
    3. Choose between fuzzing, symbolic execution, and input mutation strategies.
- Integrate AI-assisted input generation for protocols or formats with complex structures.
- Evaluate performance on real-world benchmarks, comparing with baseline fuzzers and conducting ablation studies.

Architecture Diagram


Resource Manager
Controls budgets
Symbolic Execution Engine
Fuzzer
Continue fuzzing decision
Sends test inputs
Target Program
Execution results: coverage crashes, timing
Feedback collector
Raw execution data
Priority Scoring Module
Vulnerability scores for seeds/paths
Priority Queue
Current State
RL Agent
Based on priority scores and feedback.
Invoke Symbolic Execution Engine to solve complex paths
Resource Allocation
AI Input Generator to create semantic inputs
AI Input Generator
Add new seeds to Priority Queue
Add new seeds to Priority Queue

What is your novelty? How is it novel?

- Unlike prior works, the system combines RL-based adaptive orchestration with semantic vulnerability prioritization for efficient, targeted fuzzing.
- Introduces a resource-aware, priority-driven feedback loop ensuring computational effort targets the most promising parts of the program.
- Integrates multi-modal feedback including coverage, crash info, taint analysis, and semantic scores a more comprehensive state input for RL decision making.
- Employs AI-assisted input generation in conjunction with hybrid fuzzing, increasing the ability to reach deep program states.
- The system is modular and extensible, enabling flexible adaptation to various software types and resource constraints.

Road-map / Timeline

Period	Tasks & Milestones
September (1- 30)	1. Stabilize baseline integrations (AFL, Angr/Triton, RL scaffold) 2. Implement multi-modal feedback collector and logging 3. Develop and integrate semantic vulnerability scoring
October (1-15)	1. Build and test priority queue manager with scoring feedback 2. Integrate RL agent with priority queue and resource manager 3. Implement symbolic execution triggering based on RL decisions
October (16 - 31)	1. Add AI-assisted input generator for complex formats 2. Conduct initial small-scale experiments and tune RL policies 3. Begin ablation studies disabling one component at a time
November (1 - 20)	1. Scale experiments across full benchmark suite and collect final metrics (coverage, bugs/CPU-hour) 2. Compare results against baselines (AFL, Driller, BertRLFuzzer) 3. Finalize charts, tables, and write analysis
November (21 - 30)	1. Draft final report and paper draft 2. Prepare and rehearse presentation slides 3. Incorporate mentor feedback and polish deliverables 4. Submit all materials and deliver the final review by end of November

Conclusion:

- The project aims to build an AI-augmented hybrid fuzzing system that efficiently finds software vulnerabilities by combining fuzzing, symbolic execution, and reinforcement learning.
- It prioritizes testing efforts on parts of the software likely to contain bugs, leading to better use of computational resources.
- The system adapts dynamically to program behavior and feedback, improving bug discovery rates especially for complex and deep bugs.
- By integrating AI-based input generation and multi-modal feedback, this approach increases coverage and efficiency compared to traditional fuzzers.
- This work will help improve software security testing by focusing on finding more impactful vulnerabilities faster and using fewer resources.

References:

1. N. Stephens, J. Grosen, C. Salls, A. Dutcher, R. Wang, J. Corbetta, Y. Shoshitaishvili, C. Kruegel, G. Vigna, and R. Sekar, “Driller: Augmenting Fuzzing Through Selective Symbolic Execution,” in Proc. Network and Distributed System Security Symposium (NDSS), 2016.
2. X. Mi, Z. Li, X. Sun, X. Wang, and T. Wei, “Efficient Hybrid Fuzzing Through Conservative Constraint Debloating,” in Proc. Int. Symp. Research in Attacks, Intrusions and Defenses (RAID), 2021.
3. N. Ruaro, L. Dresel, K. Zeng, T. Bao, M. Polino, A. Continella, S. Zanero, C. Kruegel, and G. Vigna, “SyML: Guiding Symbolic Execution Toward Vulnerable States Through Pattern Learning,” in Proc. RAID, 2021.
4. K. Zhang, S. Liu, W. Li, and X. Xiao, “GDFuzz: An efficient directed fuzzing method based on XAI,” J. Syst. Softw., vol. 202, p. 111798, 2025.
5. P. Jha, J. Scott, J. S. Ganeshna, M. Singh, and V. Ganesh, “BertRLFuzzer: A BERT and Reinforcement Learning Based Fuzzer,” in Proc. AAAI Conf. Artificial Intelligence, 2024.
6. K. Gai, H. Liang, J. Yu, L. Zhu, and D. Niyato, “LLAMA: Multi-Feedback Smart Contract Fuzzing Framework with LLM-Guided Seed Generation,” arXiv:2507.12084, 2024.
7. Z. Zhang, B. Cui, and C. Chen, “Reinforcement Learning-Based Fuzzing Technology,” in Proc. Int. Conf. Innovative Mobile and Internet Services, 2025.

Thank you!