PoC 2023: Fuzzing as Reinforcement Learning

nedwill@google.com

Agenda

- Introduction and Background
- Challenges in Vulnerability Research
- Traditional Approaches to Fuzzing
- Introduction to Machine Learning Concepts
- Transformers and Their Potential in Fuzzing
- Case Studies
- Future Directions and Challenges
- Summary and Final Thoughts

Whoami

- nedwill, Project Zero Researcher
- Focus on fuzzing and memory corruption vulnerabilities
- Today, we'll explore how machine learning, particularly transformers, can revolutionize the field of fuzzing and vulnerability research.

Key Questions

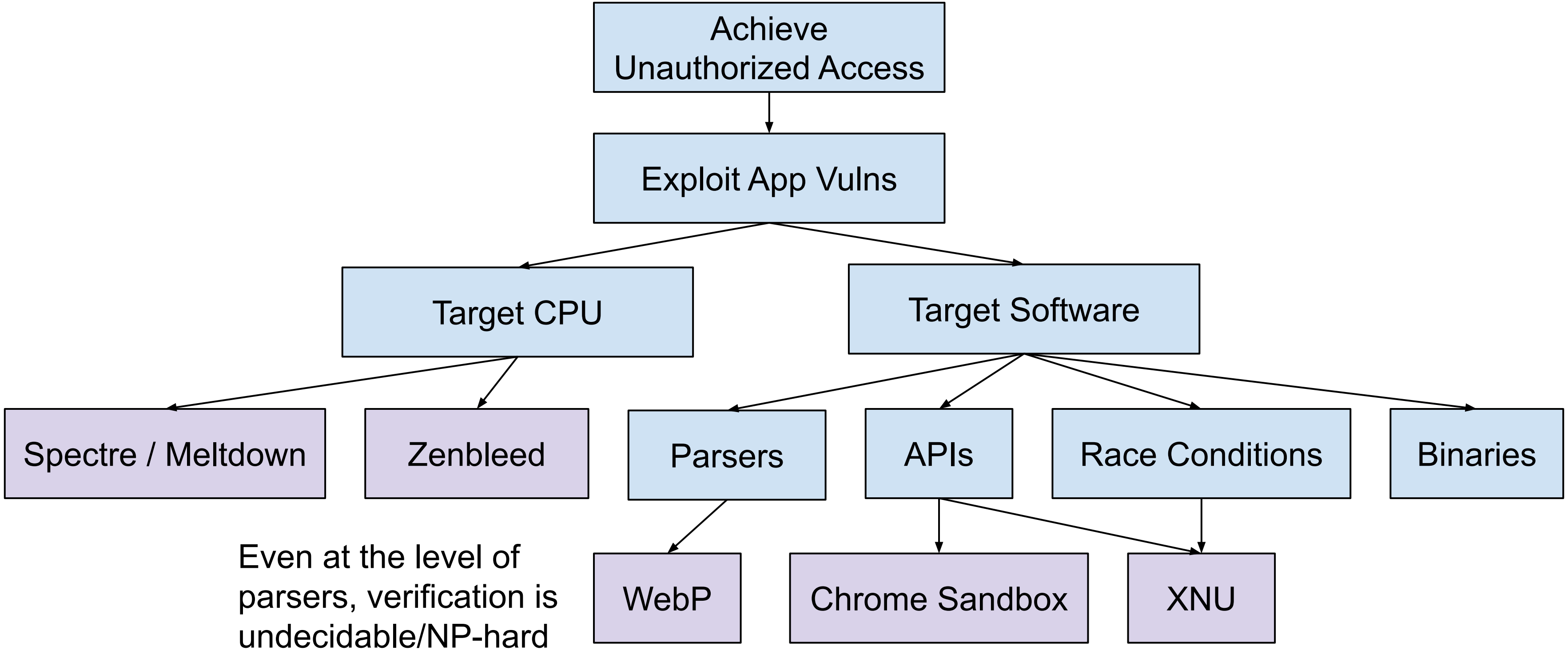
- How can we automate vulnerability research more effectively?
- What are the limitations of traditional fuzzing techniques?
- Can machine learning, specifically transformers, offer a new paradigm in fuzzing?

Background and Challenges in Vulnerability Research

The Multi-Layered Problem in Vulnerability Research

- Identifying a goal (e.g., remote code execution)
- Deciding on a method (e.g., finding a memory corruption vulnerability)
- Defining properties about security and safety
- Searching for violations by auditing and/or using computational methods

Vulnerability Research as Constrained Optimization



Even at the level of parsers, verification is undecidable/NP-hard

Complexity and Decidability Constraints

- Proving safety w.r.t. a security property in a Turing complete language is reducible to SAT for many properties, i.e. it is NP-hard
- These are fundamental barriers that this presentation won't overcome

Complexity and Decidability Constraints

- Vuln research aims to expose frequent security flaws in programs, which are often far from provably secure.
- The objective is to disprove security by finding a counterexample (bad input), making it a tractable search problem in many cases.
- Due to the lack of formally proven complex software, we rely on empirical evidence to assess safety, especially when language type systems don't guarantee it.

Approximation and Heuristics: The Power of Randomness

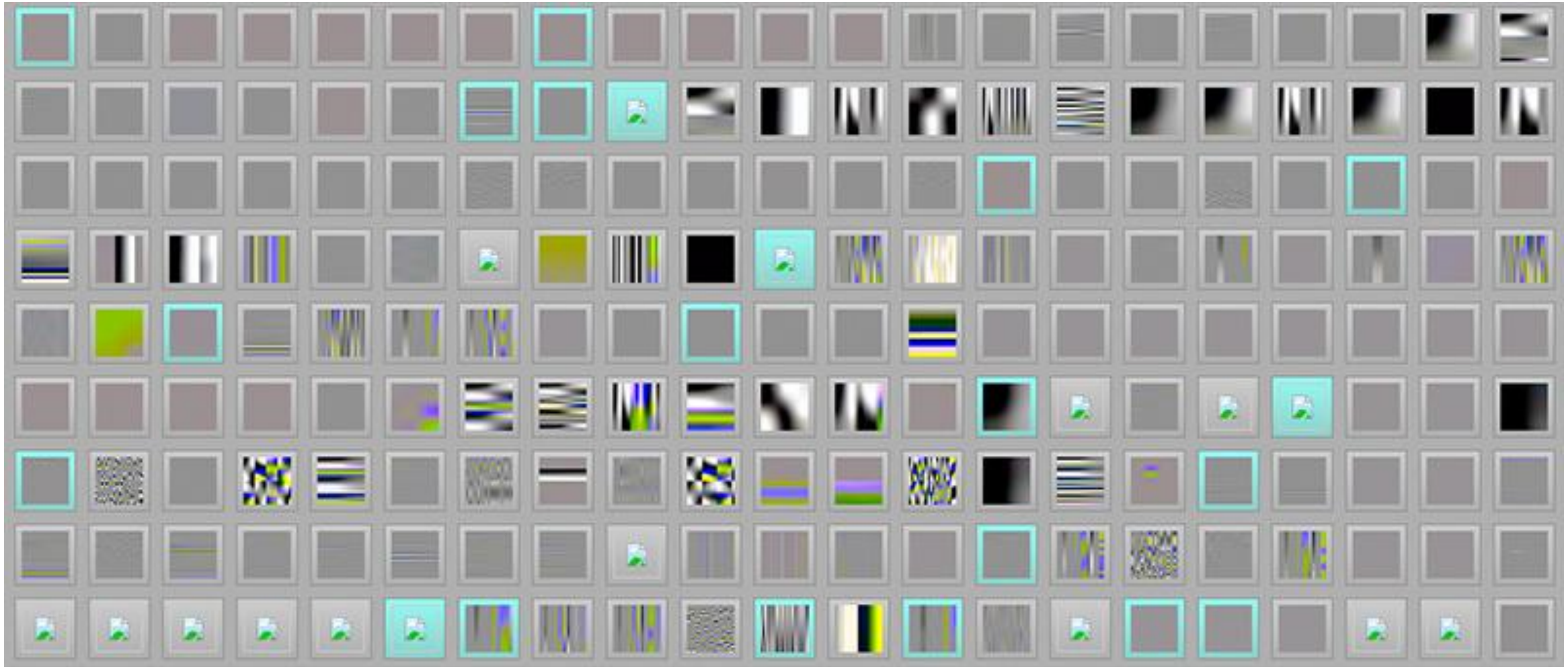
- Approximation algorithms for NP-hard problems provide value in practice
- Randomized algorithms can provide near-optimal solutions
- Similarly, many vulnerabilities are "shallow" and can be found with a randomized search (fuzzing!)

Traditional Approaches to Fuzzing

AFL: Pioneering Work in Smart Fuzzing

- Coverage-guided genetic algorithm
- We don’t just try random inputs: we learn from the behavior
- We provide a reward function (edge hitcounts, comparisons, etc.)
- Mutate and evolve a corpus that maximizes global coverage reward
- Explores and learns a suite of inputs

“Pulling JPEGs Out of Thin Air”



Zalewski, M. (2014, November 7). Pulling JPEGs out of thin air. lcamtuf's blog. https://lcamtuf.blogspot.com/2014/11/pulling-jpegs-out-of-thin-air.html

AFL’s Approach

- Heuristic-based policy drives input generation
- Reward
    - Coverage map
- State
    - Global coverage map, as well as a collection of saved inputs with known (data, covered edges)
- Action
    - Select a test case at random and apply a mutation, if we have a reward (more coverage than known state) we save the input into our state

Fuzzing Research since 2014

- Reward*
  - Come up with new features
  - e.g. Centipede supports control flow edges, data flow edges, bounded control flow paths, instrumented CMP instructions
- Action*
  - Improve the mutation quality to select from input space better according to researcher insight (protobuf-mutator)

AFL is playing a Markov Decision Process Game

- **Reward**
- **State**
- **Action**
- Partially Observable Markov Decision Process. A framework for decision-making with partial observability, relevant for fuzzing where we can't trivially predict the state of the program for a given input.
- The attempt to learn a good policy for exploring a POMDP is the goal of **model-free reinforcement learning**.
- AFL as-is is well represented by the multi-armed bandit sub-problem: no state space (only prior knowledge), single-stage decision making, simpler policy

AFL(LibFuzzer/Centipede)’s Policy

- Hard-coded, based on human intuition and experience (very successful!)
- Mostly based on mapping each (input, observed features) pair to a weight
- Weighted random selection is then employed

AFL’s Shortcomings

- Reward: We want bugs, not coverage.
- State: Do we need to memorize every input we see? Might there be insights in test cases we discard, wasting execution cycles that revealed sub-feature granularity observations that a human might find useful?
- Action: is mutating a known test case the correct way to generate an input?
- Policy: we are generally limited to features that scale linearly since we treat features opaquely

AFL's Shortcomings

- Overfitting: AFL tends to over-rely on its existing corpus of test cases, leading to a narrow exploration of the potential vulnerabilities in a target system.
- Underfitting: Despite its capabilities, AFL does not harness enough information to develop a sufficiently expressive generative model for test cases, limiting its coverage and bug discovery rate.
- Efficiency: AFL is designed to keep track of all past inputs, leading to resource inefficiencies and further exacerbating the issue of overfitting. AFL is more akin to gzip than GPT for compressing a model of the program's behavior: don't learn a generative distribution-matching representation, just explore and record a min set that covers the code while preserving perfect information about how to reach the paths in question.

Trying to Abstract the Problem

- Reward
  - Bug
- State
  - Previously found bugs, compressed representation of previous inputs, behavior, and static program information (source/binary)
- Action
  - Generate new input string
- Policy
  - Q-Learning, Decision Transformer, etc.

RL Formulation Issues

- State Space: The state space in fuzzing is expansive and can vary in size depending on the target application or even during individual runs if dynamic tracing is utilized.
- Action Space: The range of possible actions is broad and doesn't easily fit into a single 'token' as traditionally understood in RL. This is because the actions can involve creating any number of unique strings as inputs. While AFL narrows down this action space by selecting and modifying test cases, it does so at the expense of expressiveness and presupposes a collection of test cases.
- Sparse Rewards: The infrequency of discovering new vulnerabilities makes reward signals sparse, complicating the RL agent's learning process. This is a foundational problem in the reinforcement learning community.

Machine Learning Warmup

The Current Tech Zeitgeist: Generative AI

- Exciting, but induces skepticism in VR types
- ML historically hasn't made a big impact with bug discovery
- My thesis: Transformers may change this
- Effective sequence modelling has parallels to static vs. dynamic analysis
  - What makes static analysis hard to scale probably also made earlier ML attempts ineffective
  - What makes dynamic analysis easier to scale probably makes sequence models a better fit

Types of Machine Learning

- Supervised Learning: Learning from labeled data.
- Unsupervised Learning: Learning from unlabeled data.
- Reinforcement Learning: Learning by interacting with an environment to achieve a goal.
- Self-supervised Learning: Learning from unlabeled sequence data by predicting future tokens as the supervision.

Relationship with Vulnerability Research

- Traditional methods in vulnerability research often involve manual code review, static and dynamic analysis.
- Machine learning offers automated, data-driven approaches, but has historically had limited impact in vulnerability discovery.

Challenges in Applying ML to Vulnerability Research

- Sparse Rewards, High-Dimensional State Space, Computational Costs
- Combined world knowledge, reasoning, planning, and computation needs set a high bar for bug discovery.

Why Now?

- Advances in ML, particularly transformers, are accelerating and applied research in our domain hasn’t caught up to these changes.
- LLMs can teach me these topics! (transformers are good enough that they can convince me in natural language to use transformers)

Transformers: A Paradigm Shift in Computational Models

- Created in 2017, Decision Transformer (RL variant) in 2021
- Unexpectedly high performance and generality
- Simple interface: Predict an output vector from a sequence of input vectors
- Self-supervised learning is possible at very large scales
- Meta/in-context learning observable in large models

Andrej Karpathy
@karpathy

The Transformer is a magnificient neural network architecture because it is a general-purpose differentiable computer. It is simultaneously:
1) expressive (in the forward pass)
2) optimizable (via backpropagation+gradient descent)
3) efficient (high parallelism compute graph)

2:54 PM · Oct 19, 2022

Karpathy, A. (2022, October 19). The Transformer is a magnificient neural network architecture because it is a general-purpose differentiable computer. It is simultaneously: 1) expressive [Tweet]. Twitter. https://twitter.com/karpathy/status/1582807367988654081?lang=en

Andrej Karpathy on Transformers' Flexibility

- “Transformers have been applied to various fields... even in areas like computational chemistry, at the heart of algorithms like AlphaFold, you'll find a Transformer.”
- “This simple baseline of chopping up large images into small squares and feeding them into the Transformer actually works fairly well.”
- “You can chop up whatever additional information you have and feed it in… The self-attention mechanism figures out how everything should communicate.”

Karpathy, A. [Stanford]. (2023, January 10). CS25I Stanford Seminar - Transformers United 2023: Introduction to Transformers [Video]. YouTube. https://www.youtube.com/watch?v=XfpMkf4rD6E

Decision Transformer

- Paper from 2021
- Record (reward, state, action) tuples from games
- Embed these tuples into embedding space for transformer input
- Self-supervise over trajectory of (reward to go, state, action)
- Handles long term dependencies (credit assignment) using attention mechanism

Chen, L., Lu, K., Rajeswaran, A., Lee, K., Grover, A., Laskin, M., ... & Mordatch, I. (2021). Decision transformer: Reinforcement learning via sequence modeling. Advances in neural information processing systems, 34, 15084-15097.

AFL + Transformers

- Out-of-the-box idea
  - Run AFL, record (reward, state, action) tuples as is.
  - Train DT on this trajectory.
  - DT can learn a better policy than the one employed by AFL even though the data only reflects AFL's policy.
- Issues
  - Don't want transformer to just memorize crashing inputs
  - AFL log can be sparse with millions of inputs
  - Potentially too low-level

Many Potential Directions

- Record full execution trace including memory read/write, mix in source code/binary as the program executes
    - Learn a model of program execution that can be fine-tuned or few-shot prompted to generate inputs that reach specific lines of code
- Record human code review behavior: what source files w/ current corpus coverage are reviewed in what order to inform an accurate prediction about a fuzz target edit to make?
    - This requires LLM-level world model, and RL-driven LLM might be doable soon (RLHF?)
- Higher-level: mix source code review with generating and executing inputs
    - Consider an agent that can browse the code or execute inputs and observe them
    - Replicate the auditing process

Datasets

- Years of artifacts to leverage
- Fuzz targets, the targets themselves, unit tests, and git histories
- Fuzz corpora (OSS-Fuzz, Chromium, etc.)
- Crashing test cases submitted to bug trackers
- Tracker comments related to root causing bugs from high level descriptions
- Source code and binary static data, currently unused beyond opaque features
- Scaling laws
  - We need to choose an underlying function well.
  - The complexity of the function to learn dictates the size of the model needed, which in turn dictates the quantity of high quality data needed.
  - Our goal in the near term is to identify datasets that already exist or can be generated to enable us to learn functions that are useful and match the scale we can afford.

Academic research

Year	Name	Model/Architecture
2021	SyzVegas[1]	Multi-Armed Bandit (MAB), Exp3 algorithms
2021	Decision Transformer[2] (not fuzzing related)	Reinforcement Learning, Transformer architecture
9/2022	DL for Coverage-Guided Fuzzing[3]	FNN-3, FNN-5, RNN, Seq2Seq
12/2022	Hybrid Fuzzing with RL[4]	MDP with Policy Gradients
2023	Seed Scheduling with RL[5]	Temporal Difference (TD(0))
1/2023	Rainfuzz[6]	Reinforcement Learning, PPO, FFNN
7/2023	LLMs are Zero-Shot Fuzzers[7]	Codex (GPT-3 based), InCoder (bi-directional context)
10/2023	LLM-guided Protocol Fuzzing[8]	Large Language Model (LLM)

Case Studies: Zenbleed and WebP

Zenbleed: Going Beyond Coverage

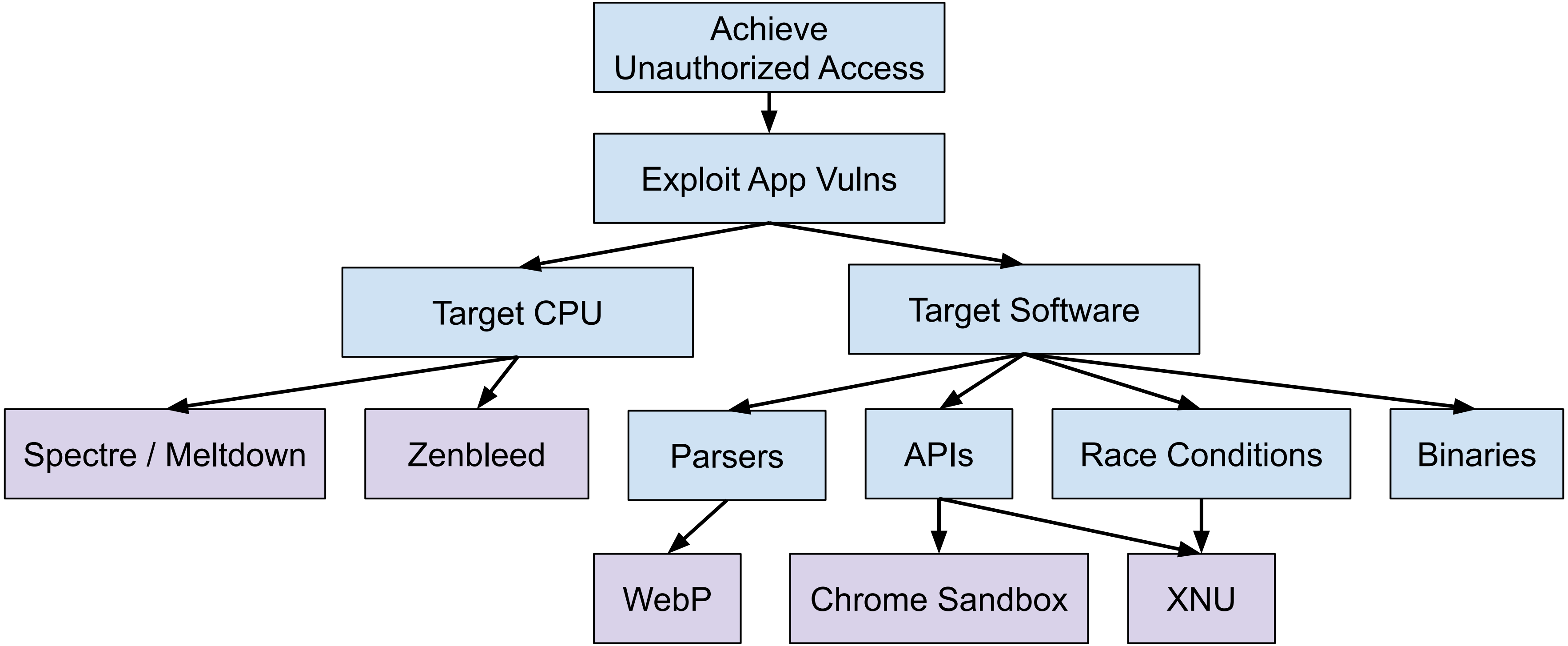
- CPU vulnerability discovered by Tavis Ormandy
- Used genetic-mutation fuzzing with performance counters as feedback
- Used Oracle Serialization as a validation mechanism
  - Create two versions of program where one uses serializing instructions (e.g. memory fences)
- Reveals two concerns
  - We want to expose as much information as possible about executed input and allow a learning algorithm to figure out what to attend to, rather than us trying to predict it a priori with prescriptive feature sets
  - We need to be able to recognize bugs based on a broader world model if we want to discover more bugs, not just ASan crashes

WebP Vulnerability: CVE-2023-4863

- Vulnerability in WebP's "lossless compression" support
- Issue with Huffman table overflow when decoding an untrusted image
- Complex state space involving valid and invalid Huffman trees
- This bug was not discovered even though a fuzz target existed
- Needed to understand how to properly structure inputs and how to create an unbalanced huffman tree
- Coverage was not enough fidelity to represent a hill-climbing path to this bug

Hawkes, B. (2023, September 21). The WebP 0day. Isosceles Blog.
https://blog.isosceles.com/the-webp-0day/

Meta-security: how do we traverse this automatically?



Compressing a World Model

- Likely needed to automatically identify new attack surfaces and vulnerabilities
- Could be partially captured by large language models
- Extremely broad search space requiring reasoning across domains
- No data to learn from besides known bugs that clearly leads to new directions
- We’ll probably get better at fuzzing before we get better at creativity
- But generative AI has shown creative capability given enough data

Issues with Turning Towards Transformers

- Poor context length scaling
  - May not apply well to vuln research where rewards are extremely sparse
- Slow learning
  - Lots of data needed to learn properly without starting from a pre-trained model
  - Learning schedules mean we may not be able to fine-tune an existing model trivially
- Slower inference than AFL
  - It may turn out that raw throughput outperforms smarter generation
- Hardware requirements and their associated costs
- Energy consumption as an ethical and practical concern
- Maintenance and updating of models is not free

Summary of Discussion

- Vulnerability research is a rich, multi-layered optimization problem that presents an excellent research area for AI applications
- Its importance for security and privacy and its geopolitical relevance lend itself to a grand challenge for AI systems
- Transformers have had a transformative and convergent impact on ML research; while they have limitations, defenders should prepare early

Full Self Researching Achieved Internally

- Autonomous driving had incremental progress over the last decade, with lots of hand-coded architectures and logic
- Recently we see that GPTs are surprisingly capable of scene understanding
- I foresee parallels in vulnerability research: lots of incremental and potentially embarrassing steps forward and backward before we can go end to end

Prompt:

Describe the image

GPT-4V:

The image is a screenshot from a dashcam, taken from the inside of a car, showing the view of the road ahead. The car is driving on a two-lane road, with a speed limit sign of 35 mph visible on the right side of the road. There are trees on both sides of the road, and the sky is filled with dark clouds, suggesting that it might rain soon. Orange cones are visible on the left side of the road, indicating a construction zone. In the distance, there is a bridge overpass.

Yang, Z., Li, L., Lin, K., Wang, J., Lin, C. C., Liu, Z., & Wang, L. (2023). The dawn of LMMs: Preliminary explorations with GPT-4V (ision). *arXiv preprint arXiv:2309.17421*.

Final Thoughts

- Vulnerability research and security are important parts of the safety and ethics discussion for generative AI and reinforcement learning systems
- Ethical considerations and real-world constraints must guide our advancements
- We should engage early with the machine learning community from an informed standpoint to best communicate how our needs and goals overlap the capability of SOTA and future architectures
- The future is promising but requires a concerted, responsible effort from all stakeholders: researchers, practitioners, and policymakers
- My plan: continue state of the art vulnerability discovery research while transitioning to increasingly diverse automated methods

References

[1] Wang, D., Zhang, Z., Zhang, H., Qian, Z., Krishnamurthy, S. V., & Abu-Ghazaleh, N. (2021). {SyzVegas}: Beating kernel fuzzing odds with reinforcement learning. In 30th USENIX Security Symposium (USENIX Security 21) (pp. 2741-2758).

[2] Chen, L., Lu, K., Rajeswaran, A., Lee, K., Grover, A., Laskin, M., ... & Mordatch, I. (2021). Decision transformer: Reinforcement learning via sequence modeling. Advances in neural information processing systems, 34, 15084-15097.

[3] Li, S., Xie, X., Lin, Y., Li, Y., Feng, R., Li, X., ... & Dong, J. S. (2022). Deep Learning for Coverage-Guided Fuzzing: How Far are We?. IEEE Transactions on Dependable and Secure Computing.

[4] Yao, D., Shen, K., & Wei, X. Towards Hybrid Fuzzing with Multi-level Coverage Tree and Reinforcement Learning in Greybox Fuzzing.

[5] Choi, G., Jeon, S., Cho, J., & Moon, J. (2023). A Seed Scheduling Method With a Reinforcement Learning for a Coverage Guided Fuzzing. IEEE Access, 11, 2048-2057.

[6] Rullo, L. (2021). Rainfuzz: reinforcement-learning driven heat-maps for boosting coverage-guided fuzzing.

[7] Meng, R., Mirchev, M., Böhme, M., & Roychoudhury, A. Large Language Model guided Protocol Fuzzing.

[8] Deng, Y., Xia, C. S., Peng, H., Yang, C., & Zhang, L. (2023, July). Large language models are zero-shot fuzzers: Fuzzing deep-learning libraries via large language models. In Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis (pp. 423-435).