Analyzing and Securing Software via Robust and Generalizable Learning

Kexin Pei
Department of Computer Science

Columbia University in the City of New York

The University of Chicago

“Software is Eating the World”

- Marc Andreessen

Software is Plagued with Errors

"Bad software cost US businesses **$2.41 trillion** in 2022" **- SC Media**

"**280 days** average time companies need to detect and respond to cyber attacks…" **- Skybox**

"Cybercrime is predicted to cost the world **$7 trillion** in 2022" **- CISQ Report**

Hackers breach energy orgs via bugs in discontinued web server

By Sergiu Gatlan   November 22, 2022   02:55 PM   0

Florida Hack Exposes Danger to Water Systems

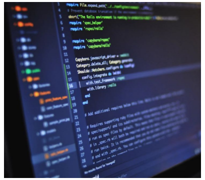
STATELINE ARTICLE | March 10, 2021 | By: Jenni Bergal | Read time: 7 min

CYBERSECURITY

Cyberattack on food supply followed years of warnings

Virtually no mandatory cybersecurity rules govern the millions of food and agriculture businesses that account for about a fifth of the U.S. economy. And now, the risk has become real.

Program Analysis is Crucial for Building Trustworthy Software



Software Programs

Questions

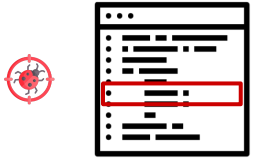
Can pointer p be NULL at line L?

Program Analysis

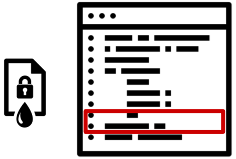
Answers

Help to Build

Vulnerability



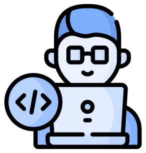
Privacy Leak



Trustworthy Software

- **Security**
- **Reliability**
- **Safety**
- **Privacy**
- **Performance**

Challenges of Traditional Program Analysis

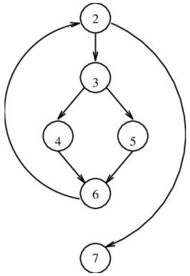


Human Expert

Hand-Curate

Rules and Heuristics

How to



Represent a program?

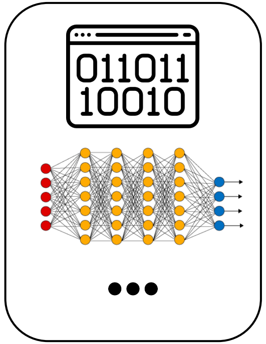
Input: $kill(B)$ and $gen(B)$ for every basic bloc B.

Output: $in(B)$ and $out(B)$ for every basic bloc B.

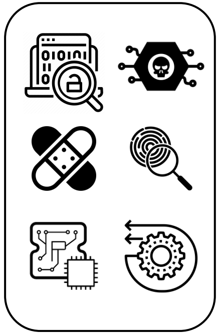
for each B **repeat**
  $out(B) := gen(B)$
while changes to any $out(B)$ occur **repeat**
  $in(B) := \bigcup_{B' \in \text{pred}(B)} out(B')$
  $out(B) := gen(B) \bigcup (in(B) \setminus kill(B))$

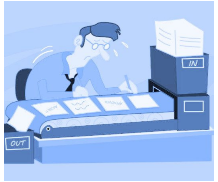
Design analysis rules?

Heterogeneous Software



Various Security Applications





Significant Manual Effort

Tune

Machine Learning Shows Promise for Analyzing Programs

Human Expert

Hand-Curate →

Rules and Heuristics

How to →

Represent a program?

Input: $kill(B)$ and $gen(B)$ for every basic bloc B.

Output: $in(B)$ and $out(B)$ for every basic bloc B.

for each B **repeat**
$out(B) := gen(B)$
while changes to any $out(B)$ occur **repeat**
$in(B) := \bigcup_{B' \in \text{pred}(B)} out(B')$
$out(B) := gen(B) \bigcup (in(B) \setminus kill(B))$

Design analysis rules?

Abundant Public Code

+

Abundant Compute

Learn →

Program Representations

Analysis Rules

Machine Learning Shows Promise for Analyzing Programs

CHATGPT Detecting and Exploiting Vulnerabilities

devtooligan
@devtooligan

omg. seriously mind-blown 🤯

Brandon Rossi
@0xConda

I told ChatGPT to pentest a scope of IPs and it found me $69,420 worth of bugs.

Are there any vulnerabilities in this Solidity contract? I want to make sure no one can steal other peoples' deposits. If there are, how do I fix it?

```
contract Vault {
  mapping(address => uint256) public balances;

  function deposit() public payable {
    balances[msg.sender] += msg.value;
  }

  function withdraw(uint256 amount, address from) public {
    balances[from] -= amount;
    payable(from).transfer(amount);
  }
}
```

This contract appears to have a potential vulnerability in the `withdraw` function. The `withdraw` function allows anyone to specify the `from` address when calling the function, which means that a malicious user could potentially withdraw funds from an arbitrary address by calling `withdraw` and specifying a different `from` address.

To fix this issue, you could modify the `withdraw` function to only allow the caller to withdraw funds from their own account. You could do this by replacing the `from` parameter with `msg.sender`, which will always represent the caller's address. Here is an example of how you could do this:

Limitations: Lack Understanding of Program Semantics

A code summarization example (Alon et al., 2019, Yefet et al., 2020, Henkel et al. 2022)
[code2vec.org / code2seq.org]

```
void f1(int[] array){
  boolean swapped = true;
  for (int i = 0;
    i < array.length && swapped; i++){
    swapped = false;
    for (int j = 0;
    j < array.length-1-i; j++) {
      if (array[j] > array[j+1]) {
        int temp = array[j];
        array[j] = array[j+1];
        array[j+1]= temp;
        swapped = true;
      }
    }
  }
}
```

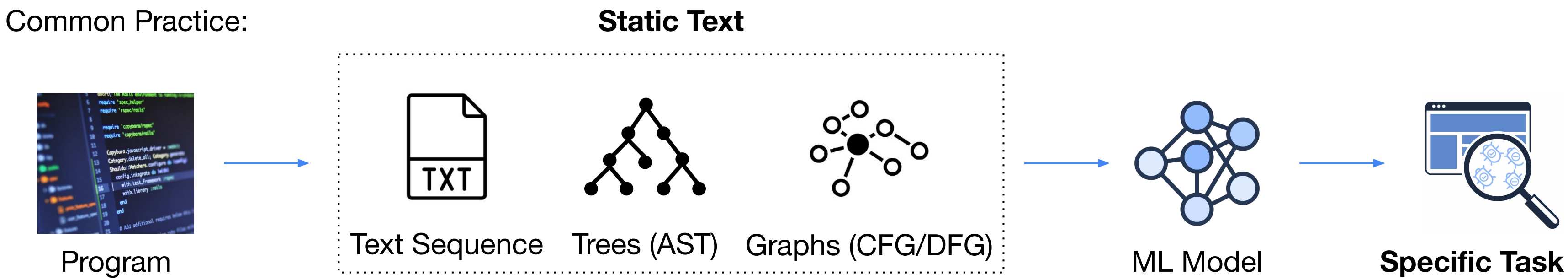
Prediction: **sort** (98.54%)

```
void f2(int[] ttypes){
  boolean swapped = true;
  for (int i = 0;
    i < ttypes.length && swapped; i++){
    swapped = false;
    for (int j = 0;
    j < ttypes.length-1-i; j++) {
      if (ttypes[j] > ttypes[j+1]) {
        int temp = ttypes[j];
        ttypes[j] = ttypes[j+1];
        ttypes[j+1]= temp;
        swapped = true;
      }
    }
  }
}
```

Prediction: **contains** (99.97%)

Yefet, Noam, Uri Alon, and Eran Yahav. "Adversarial examples for models of code." OOPSLA 2020.

Common Practice of ML on Code



Program semantics does not just manifest in **static text**

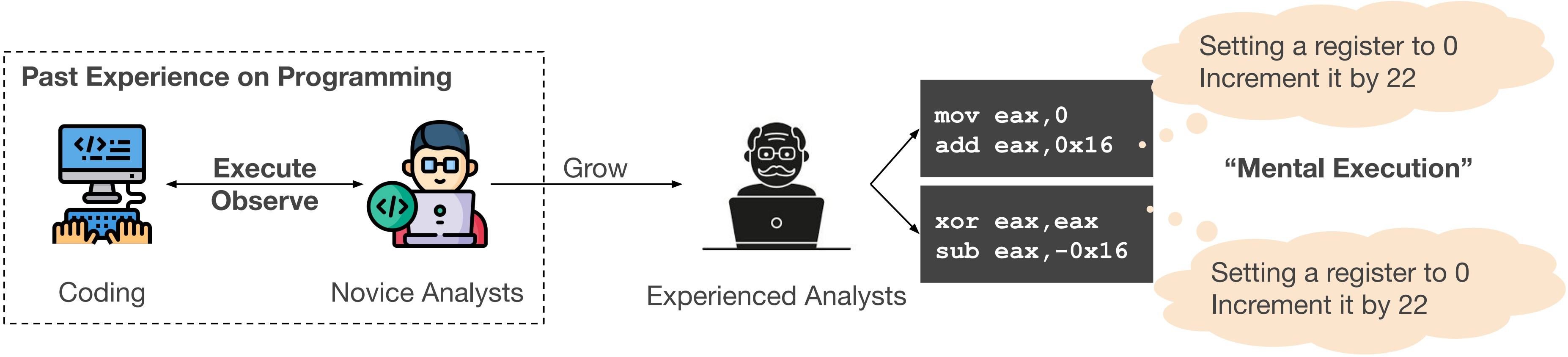
Consequences: Lacking Robustness and Generalization

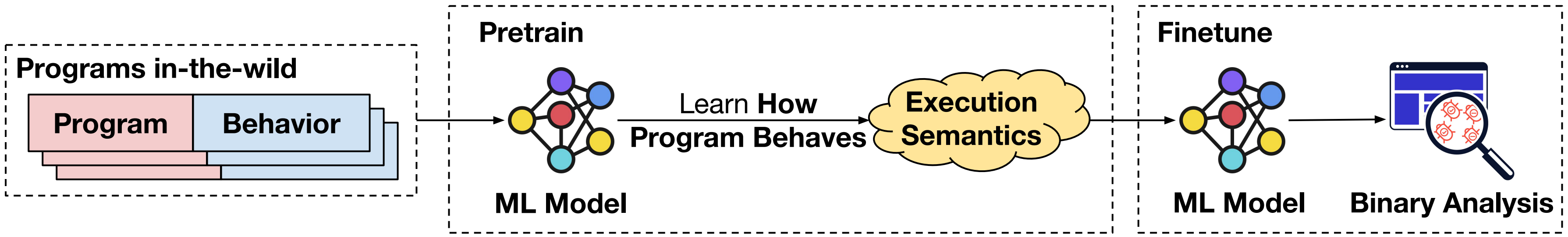
I. **Overfit** to **spurious textual** and **task-specific** patterns

II. **Distribution shift**: **program syntax** and **task requirement** changes

Security Applications Require More Rigorous Understanding of Program Semantics

Learning Execution Semantics and Transferring it without Dynamic Analysis





Case Studies: Vulnerability Search in Firmware

CVE	Library	Description
CVE-2019-1563	OpenSSL	Decrypt encrypted message
CVE-2017-16544	BusyBox	Allow executing arbitrary code
CVE-2016-6303	OpenSSL	Integer overflow
CVE-2016-6302	OpenSSL	Allows denial-of-service
CVE-2016-2842	OpenSSL	Allows denial-of-service
CVE-2016-2182	OpenSSL	Allows denial-of-service
CVE-2016-2180	OpenSSL	Out-of-bounds read
CVE-2016-2178	OpenSSL	Leak DSA private key
CVE-2016-2176	OpenSSL	Buffer over-read
CVE-2016-2109	OpenSSL	Allows denial-of-service
CVE-2016-2106	OpenSSL	Integer overflow
CVE-2016-2105	OpenSSL	Integer overflow
CVE-2016-0799	OpenSSL	Out-of-bounds read
CVE-2016-0798	OpenSSL	Allows denial-of-service
CVE-2016-0797	OpenSSL	NULL pointer dereference
CVE-2016-0705	OpenSSL	Memory corruption

16 Vulnerabilities (Compiled in x86)

Search →

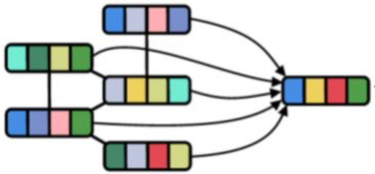
Firmware Images: `Arm`, `MIPS` with unknown compiler flags

Ubiquiti sunMax → 15 CVEs

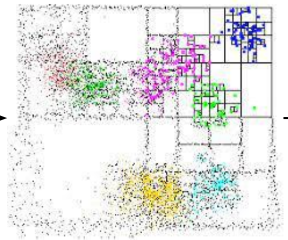
TP-Link Deco-M4 → 16 CVEs

NETGEAR R7000 → 7 CVEs

Linksys RE7000 → 14 CVEs



Learned Function Embeddings

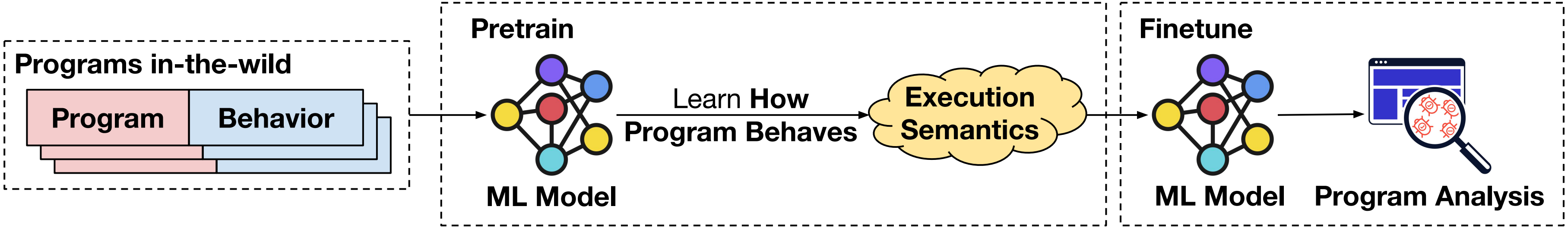


Approximate Nearest Neighbors

Search over **1.4M** functions within **6.3 seconds**

Pei, Xuan, Yang, Jana, Ray. Trex: Learning Execution Semantics from Micro-traces for Binary Similarity. TSE'22

Summary: Learning **Program Semantics** via **Execution-Aware Pre-training** Improves Program Analysis



Precise: Outperforms the state-of-the-art by up to **118%**

Efficient: Speedup over the off-the-shelf tool by up to **98.1x**

Generalizable and Robust: Remains accurate across

GCC Visual C++ — Compilers

x64 x86 — Architectures

-O0 -O1 -O2 -O3 -Od Ox — Optimizations

Obfuscations

Broad Application

- Detecting Semantically Similar Binary Code [1]
- Type Inference and Data Structure Recovery [2]
- Binary Memory Dependence Analysis [3]
- Inferring Program Invariance for Source Code [4]
- Source Code Vulnerability Detection [5]

[1] Pei et al. Trex: Learning Execution Semantics from Micro-traces for Binary Similarity. TSE'22
[2] Pei et al. StateFormer: Fine-grained type recovery from binaries using generative state modeling. ESEC/FSE'21
[3] Pei et al. NeuDep: neural binary memory dependence analysis. ESEC/FSE'22
[4] Pei et al. Can Large Language Models Reason about Program Invariants. ICML'23
[5] Ding et al. TRACED: Execution-aware Pre-training for Source Code. ICSE'24.