Using AI to Find Security Defects in Code / Build More Secure Software

Defense Science & Technology Agency (DSTA) Workshop
Sept. 16, 2020

Dr. Lori Flynn on behalf of the Office of the Undersecretary of Defense for Research and Engineering

Senior Software Security Researcher
Using AI to Find Security Defects in Code / Build More Secure Software

DM20-0739
Software Cost and Vulnerability Threat to Missions

Finding and fixing software flaws late in the acquisition lifecycle drives up cost and delays delivery.

Latent software defects put missions at risk. Sometimes those defects are exposed during operations.

AI to automate and improve what humans do, to develop and analyze code for security, and to secure AI software itself.
Fixing Problems Late Drives Costs, Delays Deployment

AI to automate and improve what humans do, to develop and analyze code for security, and to secure AI software itself.
“The IBM Mathematical Formula Translating System or briefly, FORTRAN, will comprise a large set of programs to enable the IBM 704 to accept a concise formulation of a problem in terms of a mathematical notation and to produce automatically a high speed 704 program for the solution of the problem.”

AI in Automatic Programming: Generating Code thru Search – High Assurance SPIRAL

"High Assurance SPIRAL aims to solve the last mile problem for the synthesis of high assurance implementations of controllers for vehicular systems that are executed in todays and future embedded and high performance embedded system processors."

Using AI For Autocompletion

Safe, correct code could be written incrementally

- Using n-grams
- Using deep learning (Generative Pretrained Transformer 2)

Sources:


© 2019 TabNine, See https://tabnine.com/eula
Finding Code Defects Using AI: Classifiers & Active Learning

Using classifiers for static analysis alerts

Active learning updates predictions as new data is received.

- **[Ruthruff]:** 85% accurate false positive prediction for FindBugs, Logistic Regression, adaptive using code-fix decisions
- **[Heckman] ARM:**
  - 81% true positive alerts after investigating only 20% of alerts (vs. avg. of 50 random orderings found 22% after investigating 20%)
  - Code locality and alert type accuracy (new adjudication feedback)
  - Code fixer feedback to system
- **[Kremenec] Feedback-Rank**
  - 2-8x improvement of performance ratio over random
  - Performance ratio: ratio between random and shift per bug from optimal
  - Code locality and alert type accuracy (new determinations feedback)

Finding Code Defects Using AI: Data Quality

Static Analysis Tool(s) → Alerts → Potential Rule Violations → Determinations

Alert Consolidation (SCALe) → Auditing

Select candidate code bases for evaluation

Training Data → ML Classifier Development

Codebase 1 → Codebase 2 → Codebase 3
Finding Code Defects Using AI: Data Quality

**Static Analysis Tool(s)**

**Alerts**

**Alert Consolidation (SCALE)**

**Potential Rule Violations**

**Auditing**

**Determinations**

**Run SA Tool(s) collecting code alerts and metrics (e.g. complexity)**

**Training Data**

**ML Classifier Development**

1. Codebase 1
2. Codebase 2
3. Codebase 3
Finding Code Defects Using AI: Data Quality

- Codebase 1
- Codebase 2
- Codebase 3

Static Analysis Tool(s) ➔ Alert Consolidation (SCALE) ➔ Auditing

Alerts ➔ Potential Rule Violations

Convert alerts to common format and map to CERT Secure Coding Rules/CWEs

ML Classifier Development
Finding Code Defects Using AI: Data Quality

Humans evaluate the violations, e.g. marking them as TRUE or FALSE.
Finding Code Defects Using AI: Data Quality

Use the training data to build machine learning classifiers that predict TRUE and FALSE determinations for new alerts.
Finding Code Defects Using AI: Data Quality

Static Analysis Tool(s) → Alerts
Alert Consolidation (SCALe) → Potential Rule Violations

Codebase 1 → Codebase 2 → Codebase 3

Auditing → Determinations
Training Data → ML Classifier Development

What do TRUE/FALSE mean? Are there other determinations I can use?
One collaborator reported using the determination `True` to indicate that the issue reported by the alert was a real problem in the code.

Another collaborator used `True` to indicate that `something` was wrong with the diagnosed code, even if the specific issue reported by the alert was a `false positive`!
Finding Code Defects Using AI: Data Quality

Inconsistent assignment of audit determinations may have a negative impact on classifier development!
Data Quality: Lexicon And Rules

- We developed a **lexicon** and auditing **rule set** for our collaborators
- Includes a standard set of well-defined **determinations** for static analysis alerts
- Includes a set of **auditing rules** to help auditors make consistent decisions in commonly-encountered situations

<table>
<thead>
<tr>
<th>Different auditors</th>
<th>should make the <strong>same determination</strong> for a given alert</th>
</tr>
</thead>
<tbody>
<tr>
<td>Improve the <strong>quality and consistency</strong> of audit data for the purpose of building <strong>machine learning classifiers</strong></td>
<td></td>
</tr>
<tr>
<td>Help organizations make <strong>better-informed</strong> decisions about <strong>bug-fixes, development, and future audits</strong></td>
<td></td>
</tr>
</tbody>
</table>

Lexicon: Audit Determinations

Basic Determinations:
- True
- False
- Complex
- Dependent
- Unknown (default)

Choose ONE per alert!

Supplemental Determinations:
- Inapplicable environment
- Dangerous construct
- Dead
- Ignore

Choose ANY NUMBER per alert!

Data Quality: Audit Rules

Goals

• Clarify ambiguous or complex auditing scenarios
• Establish assumptions auditors can make
• Overall: help make audit determinations more consistent

We developed 12 rules

• Drew on our own experiences auditing code bases at CERT
• Trained 3 groups of engineers on the rules, and incorporated their feedback

FY16-19 My SEI Static Analysis Alert Classification Research

Goal: Enable practical automated classification, so all meta-alerts can be addressed.

FY16
- Issue addressed: classifier accuracy
- Novel approach: multiple static analysis tools as features
- Result: increased accuracy

FY17
- Issues addressed: data quality, too little labeled data for accurate classifiers for some conditions (e.g., CWEs, coding rules)
- Novel approach: audit rules+lexicon, use test suites to automate the production of labeled (True/False) meta-alert data* for many conditions
- Result: high precision for more conditions

FY18-19
- Issue addressed: little use of automated alert classifier technology (requires $$, data, experts)
- Novel approach: develop extensible architecture with novel test-suite data method
- Result: enabled wider use of classifiers (less $$, data, experts) with extensible architecture, API, software to instantiate architecture, and adaptive heuristic research

* By the end of FY18, ~38K new labeled (T/F) alerts from eight SA tools on the Juliet test suite (vs. ~7K from CERT audit archives over 10 years)
- L. Flynn publications at SEI Digital Library: https://resources.sei.cmu.edu/library/author.cfm?authorid=31216
Finding Code Defects Using AI: Data Quantity & Quality

CERT-Audited Archives Characterization
- 58 CERT coding rules with 20 or more audited (labeled) alerts
- 25 rules all (or nearly all) determined one way (True or False)
- Other 324 CERT rules have little or no labeled data
- Labeled data for 158 of 382 CERT rules
- 2,487 True and 4,980 False

Test Suites Used for AI Data Generation: Juliet Initial Data

<table>
<thead>
<tr>
<th>Alert Type</th>
<th>Labeled Meta-alert (counts a fused alert once)</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>13,330</td>
</tr>
<tr>
<td>FALSE</td>
<td>24,523</td>
</tr>
</tbody>
</table>

Big savings: manual audit of 37,853 alerts from non-test-suite programs would take an unrealistic minimum of 1,230 hours (117 seconds per alert*).

- First 37,853 alert audits wouldn’t cover many conditions (and sub-conditions) covered by the Juliet test suite!
- Need true and false labels for classifiers.

**Realistically:** enormous amount of manual auditing time to develop that much data.

These are initial metrics (more data to follow as we use more tools and test suites).

FY20 My SEI Static Analysis Alert Classification Research

Goal: Enable **practical** automated classification, so all meta-alerts can be addressed.

• Issue addressed: It takes too much time to adjudicate static analysis alerts/meta-alerts during continuous integration (CI).

• Novel approach: During CI builds, use **classifiers** with **precise cascading** and **CI/CD features**

• Results:
  - Design for CI-SCAIFE system integration
  - Cascading API defined, for true/false adjudications ‘cascade’ to subsequent versions of code
  - Less-precise cascading implemented, test results
  - Significant progress on CI-SCAIFE system integration development
  - Deployment and testing by DoD collaborators (multiple rounds), & public API + code subset publications

• Also: RC_Data open dataset for improved classifier research. Published our own data to begin, plan to grow, with our data and data from others. University of Virginia plans to add data.


Finding Code Defects Using AI: Cross-Project Prediction

Cross-project defect prediction:
- Compares 2 types of unsupervised classifiers compared to manual efforts to make homogenous datasets. Connectivity-based classification using spectral clustering worked well (but supervised better). [Baishakhi]
- 9% improvement in cross-project defect prediction, using semantic features [Wang]
- At SEI, ongoing work on cross-project prediction and active learning with mix of test suite and natural program data [Flynn]

Finding Code Defects – AI that Considers Source Code as Natural Language

Analyze Source Code for Insecure Coding

- Supplements Compiler-style Checking
- Treats Programs Like Natural Language


Using AI to Drive Test Inputs – Fuzzing

“Fuzzing:” Generating and Testing Random Inputs
Original: Random or Deterministic
Now: Use AI to Guide Generation of Sample Inputs

Sources:


Using AI to Improve Penetration Testing

Variety and combination of manual techniques can be executed by an AI system

- AI planning using an attack graph against attack surfaces
- Markov Decision Process (or Partially Observable Markov Decision Process) over application state
- Reinforcement learning

Sources:


Automated Program Repair – DARPA Cyber Grand Challenge

“Mayhem” demonstrated automated cyber defense
• Detect attack on program
• Analyze changes to program
• Deploy updated software

AI Supporting Judgement – IBM Watson to Improve Assurance

• Acquisition programs generate voluminous documentation
• Assurance is based on assembling and reviewing relevant evidence from documents
• Finding appropriate evidence or explanations can be challenging
• SEI proof of concept

AI Attacks Are Different

Pixel Manipulation

Feature Differentiation

“Milla Jovovich”

$0.22 to print

“Milla Jovovich”


Some Technical Approaches for Defending AI Systems

**Training Defenses**
Wong & Kolter (2017)
output bound

**Causal Defenses**
Tsipras et al. (2018)
adversarial data augmentation

**Engineering Defenses**
Su et al. (2018) empirically demonstrates robustness/accuracy trade off in ImageNet models

---

Sources:
AI is Playing an Increasing Role in Cybersecurity

Classifying Malware
Spotting Deep Fakes
Detecting Campaigns

- Detecting misinformation
- Spotting command and control paths
- Cyber training

- Technical debt detection
- Satellite image recognition
- Insider threat detection
Summary: Using AI to Build More Secure Software

Problem: The Need to Build Secure Software
Threat Analysis: What To Protect Against
Code Development: Assisting Programmers to Build More Secure Software
Building AI Systems Securely: Next Generation of Software Face New Attacks