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

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

DM20-0739

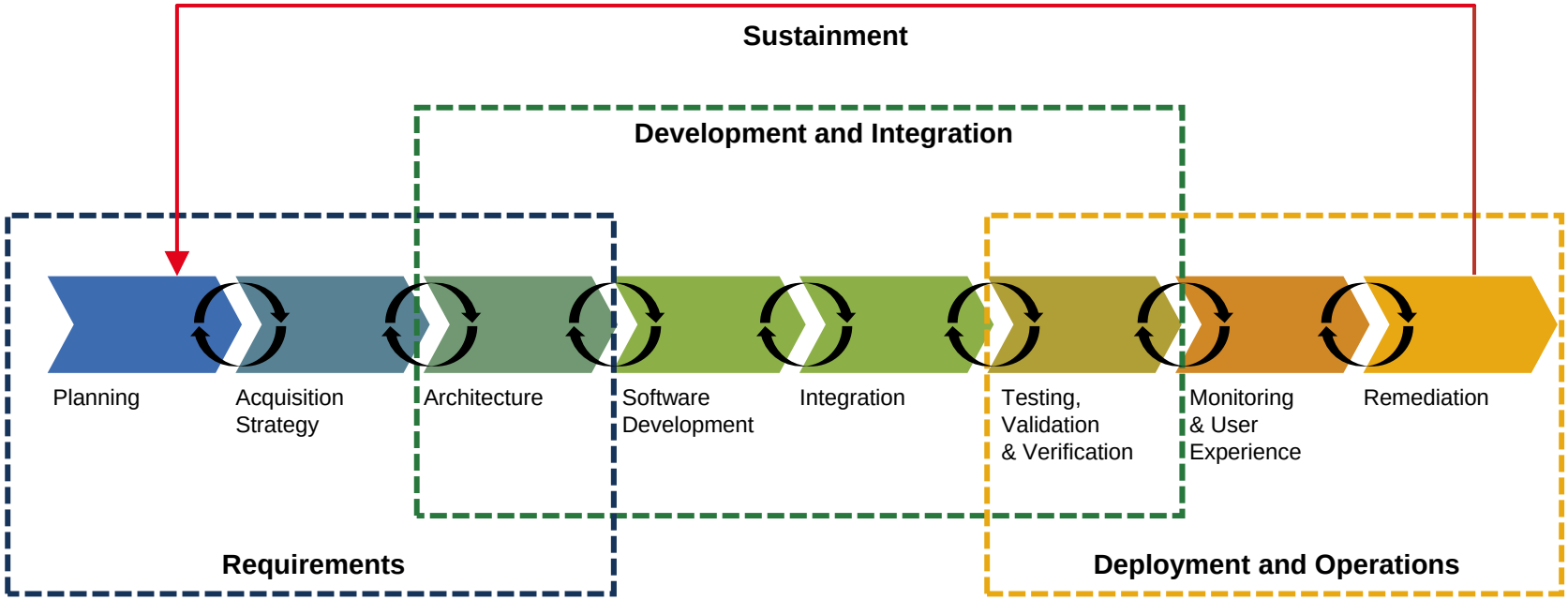
Software Cost and Vulnerability Threat to Missions

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.

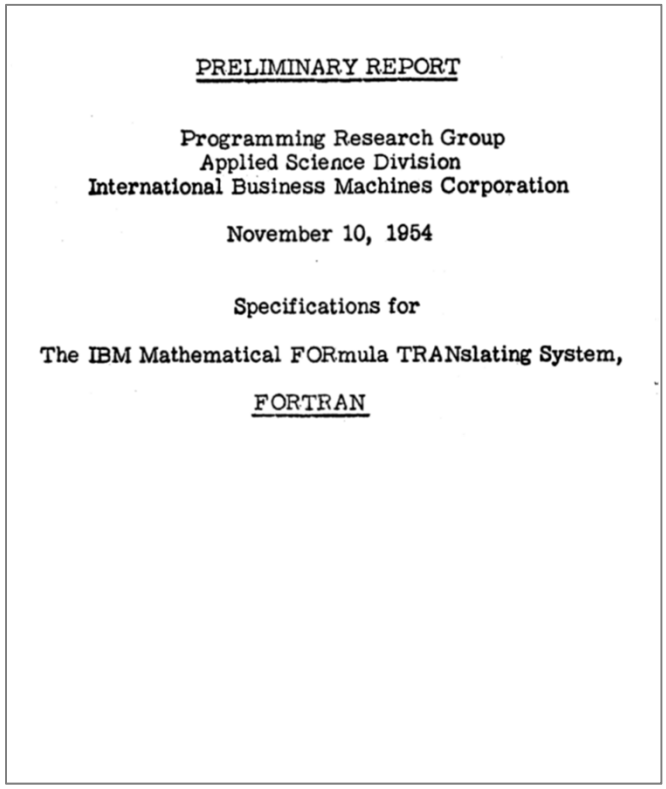
AI to automate and improve what humans do, to develop and analyze code for security, and to secure AI software itself.

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.

AI in Automatic Programming – The Beginning



PRELIMINARY REPORT

Programming Research Group
Applied Science Division
International Business Machines Corporation

November 10, 1954

Specifications for

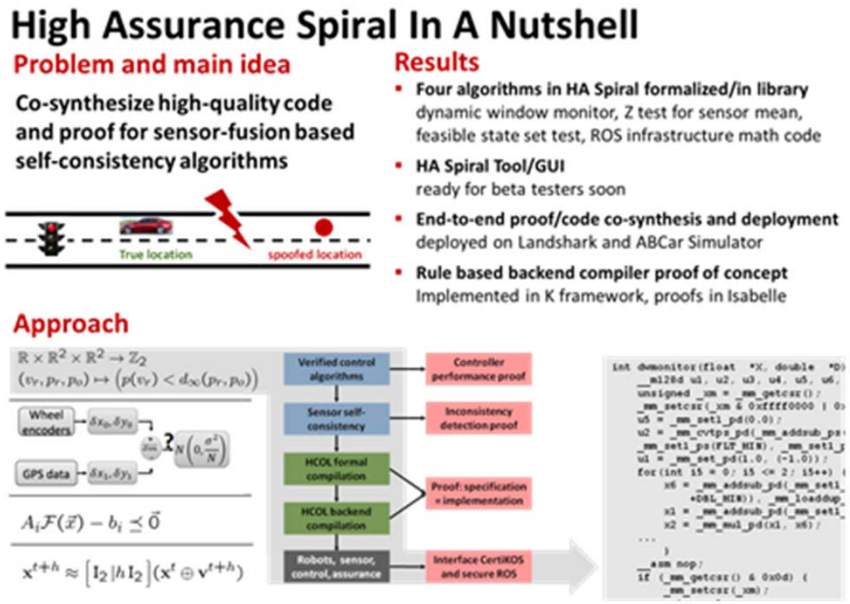
The IBM Mathematical FORmula TRANslating System,

FORTRAN

"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."

Source: J.W. Backus, H. Herrick and I. Ziller,
https://archive.computerhistory.org/resources/text/Fortran/102679231.05.01.acc.pdf

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."

Sources: Franz Franchetti, José M. F. Moura, Manuela Veloso, Andre Platzer, Soummya Kar, David Padua, Jeremy Johnson, Mike Franusich, High Assurance Spiral: Scalable and Performance Portable Domain-Specific Control System Synthesis, https://users.ece.cmu.edu/~franzf/hacms.htm; http://www.spiral.net/

Using AI For Autocompletion

```
import os
import sys

# Count lines of code in the given directory, separated by file extension
def main(directory):
    line_count = {}
    for filename in os.listdir(directory):
        _, ext = os.path.splitext(filename)
        if ext not in line_count:
            line_count[ext] = 0
        for line in open(os.path.join(directory, filename)):
            line_count[ext] += 1
            line_count[ext] += 1              13%
            line_count[ext               Tab  20%
            line_count[ext] +=             3  14%
            line_count[ext].append(        4   3%
            line                           5  23%


```


Safe, correct code could be written incrementally

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

Sources:
E. Schutte, Autocomplete from StackOverflow, 2016, https://emilschutte.com/stackoverflow-autocomplete/

(Jacob Jackson) TabNine, “Autocompletion with deep learning,” July 18, 2019, https://tabnine.com/blog/deep

L. Tung, “New tool promises to turbo-charge coding in major programming languages,” July 25, 2019, https://www.zdnet.com/article/new-tool-promises-to-turbo-charge-coding-in-major-programming-languages/

Finding Code Defects Using AI: Classifiers & Active Learning

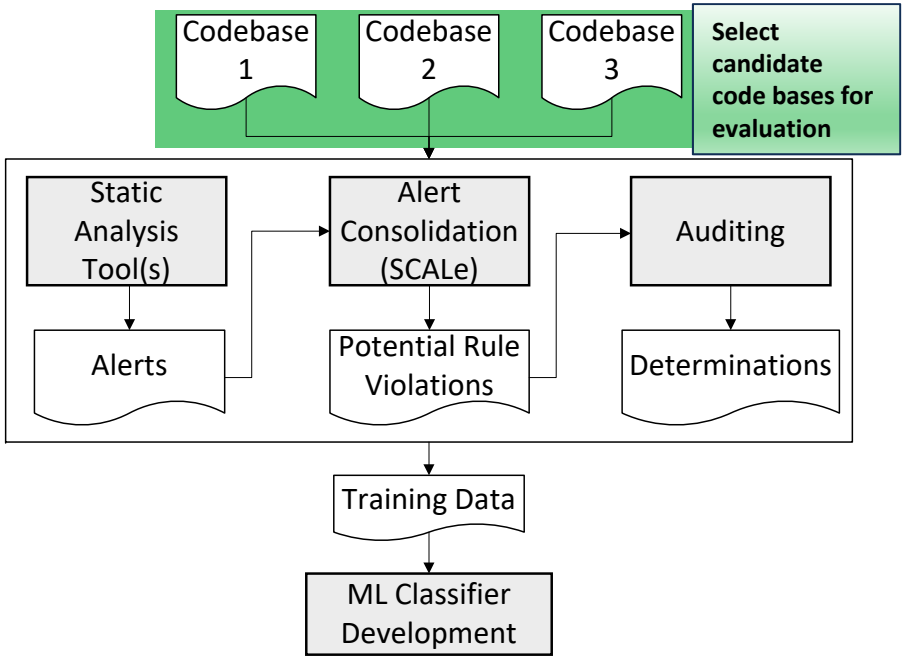
Using classifiers for static analysis alerts

Active learning updates predictions as new data is received.

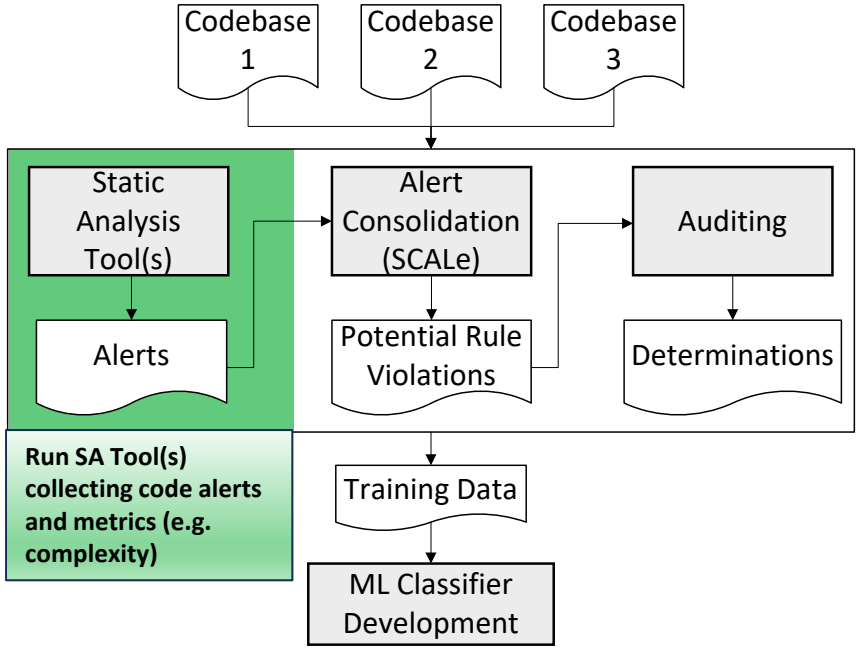
- [Ruthruff]: 85% accurate false positive prediction for FindBugs, Logistic Regression, adaptive using code-fix decisions
- [Heckman] ARM:
  - 81% true positive alerts after investigating only 20% of alerts (vs. avg. of 50 random orderings found 22% after investigating 20%)
  - Code locality and alert 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)

[Heckman] Heckman, Sarah Smith. "Adaptively ranking alerts generated from automated static analysis.“, Crossroads, 2007.
[Heckman B] S. Heckman, L. Williams, On establishing a benchmark for evaluating static analysis alert prioritization and classification techniques, Empirical Software Engineering and Measurement, 2008, pp. 41–50.
[Kremenec] T. Kremenek, K. Ashcraft, J. Yang, D. Engler, Correlation exploitation in error ranking, FSE, 2004, pp.83–93.
[Ruthruff] J. Ruthruff et al. "Predicting accurate and actionable static analysis warnings: an experimental approach." ICSE, 2008.

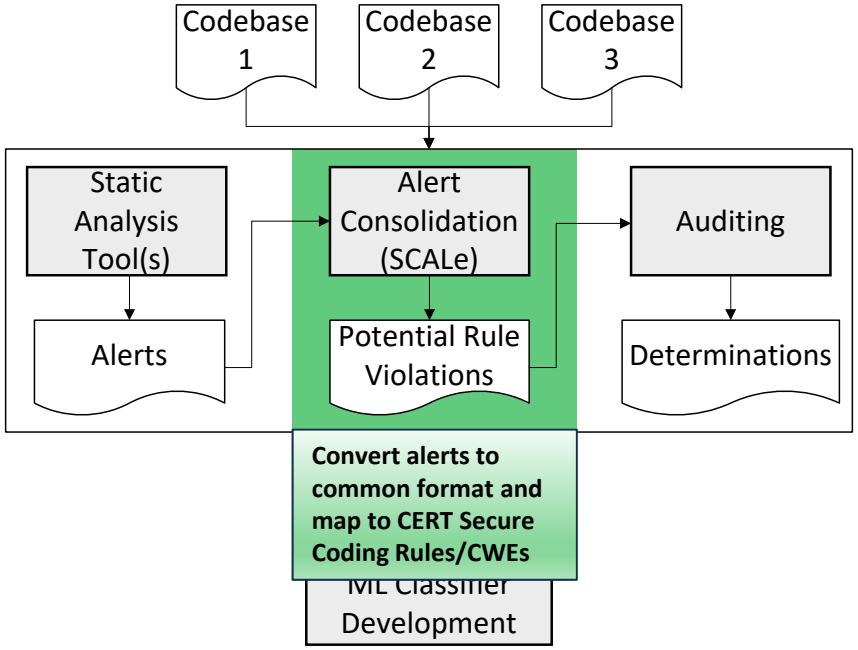
Finding Code Defects Using AI: Data Quality



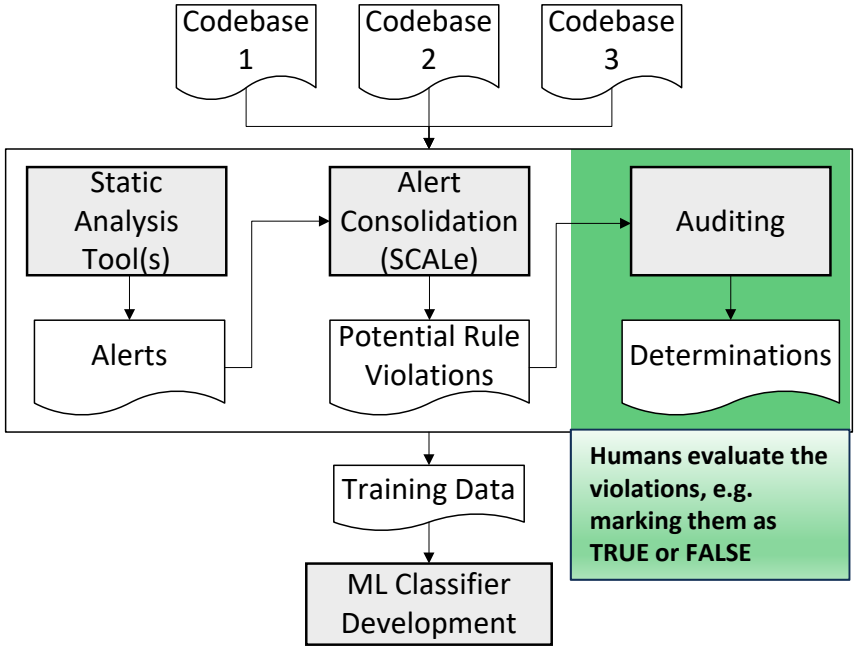
Finding Code Defects Using AI: Data Quality



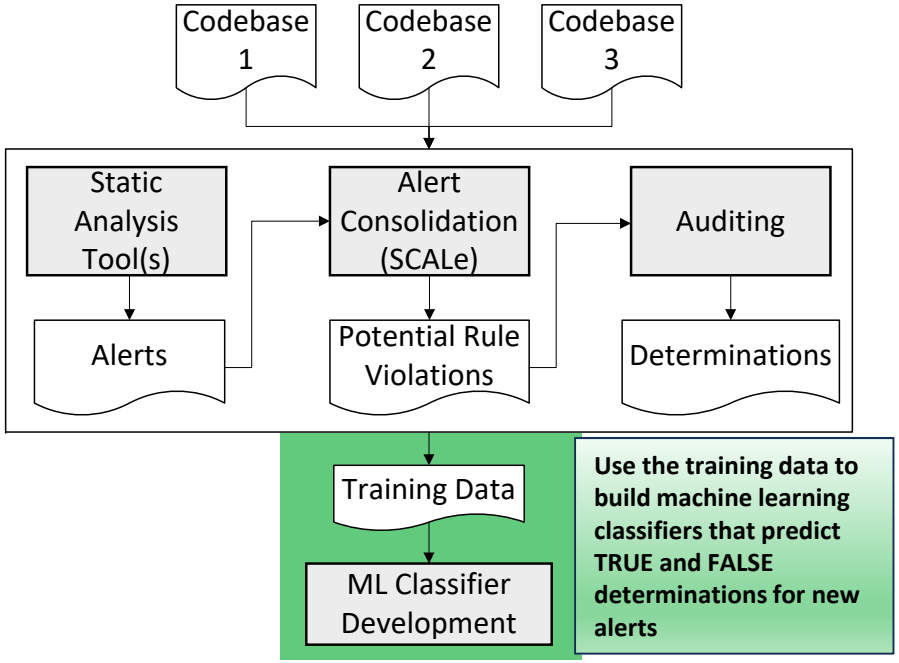
Finding Code Defects Using AI: Data Quality



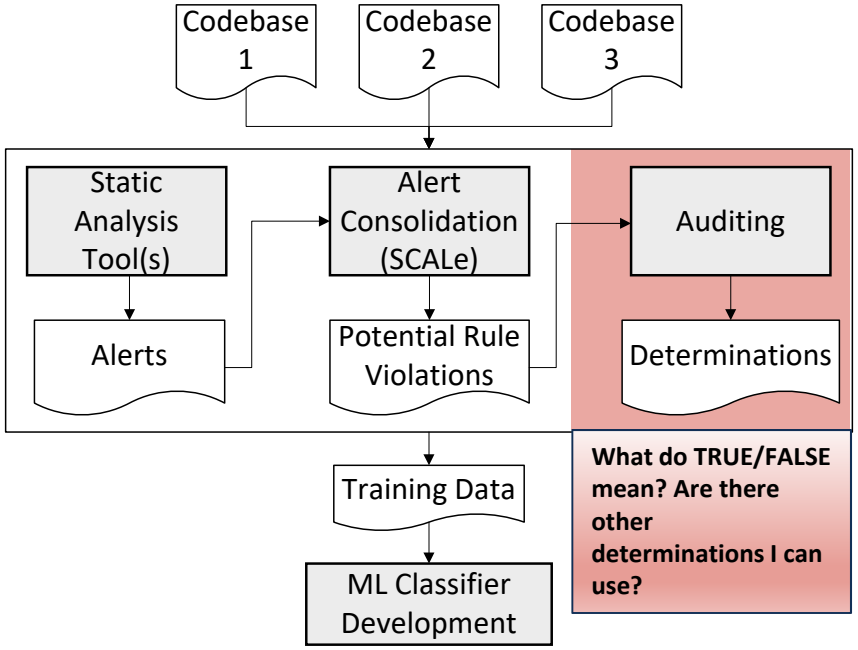
Finding Code Defects Using AI: Data Quality



Finding Code Defects Using AI: Data Quality



Finding Code Defects Using AI: Data Quality



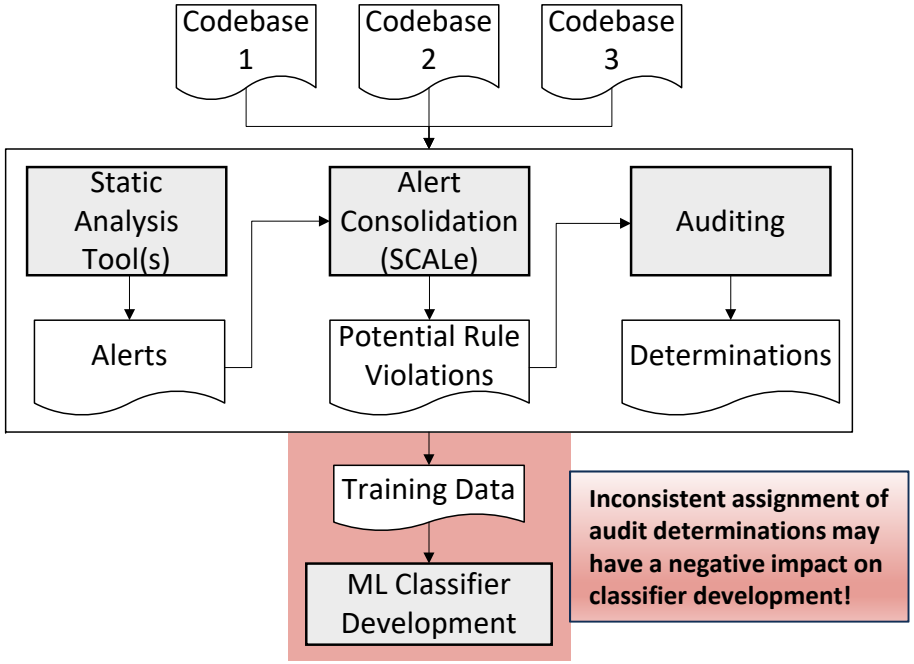
Data Quality: What is Truth?

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**!

D. Svoboda, L. Flynn, and W. Snavely. "Static Analysis Alert Audits: Lexicon & Rules." 2016 IEEE Cybersecurity Development (SecDev)

Finding Code Defects Using AI: Data Quality



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

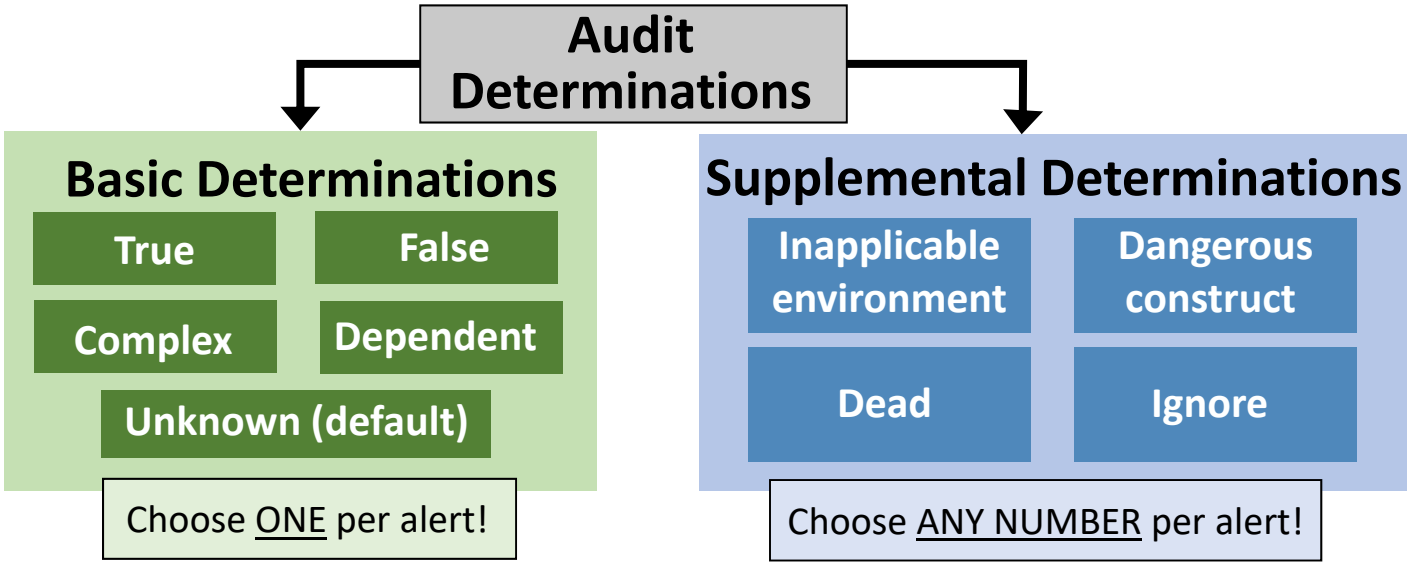
Different auditors should make the **same determination** for a given alert

Improve the **quality and consistency** of audit data for the purpose of building **machine learning classifiers**

Help organizations make **better-informed** decisions about **bug-fixes**, **development**, and **future audits**.

D. Svoboda, L. Flynn, and W. Snavely. "Static Analysis Alert Audits: Lexicon & Rules." 2016 IEEE Cybersecurity Development (SecDev)

Lexicon: Audit Determinations



D. Svoboda, L. Flynn, and W. Snavely. "Static Analysis Alert Audits: Lexicon & Rules." 2016 IEEE Cybersecurity Development (SecDev)

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

D. Svoboda, L. Flynn, and W. Snavely. "Static Analysis Alert Audits: Lexicon & Rules." 2016 IEEE Cybersecurity Development (SecDev)

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

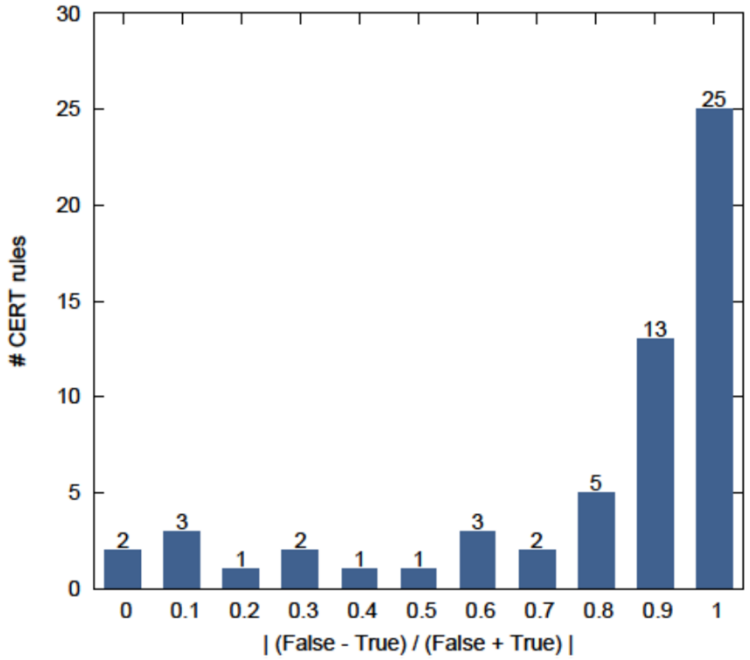
- Issue addressed: **little use of automated alert classifier technology** (requires $$, data, experts)
- Novel approach: **develop extensible architecture with novel test-suite data method**
- Result: **enabled wider use of classifiers (less $$, data, experts)** with extensible architecture, API, software 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



L. Flynn et al. “Prioritizing Alerts from Multiple Static Analysis Tools, Using Classification Models”, Software QUAlities and their Dependencies (SQUADE, ICSE 2018 workshop). https://resources.sei.cmu.edu/asset_files/ConferencePaper/2018_021_001_524697.pdf

Test Suites Used for AI Data Generation: Juliet Initial Data

Alert Type	Labeled Meta-alert (counts a fused alert once)
TRUE	**13,330**
FALSE	**24,523**

Lots of new data for creating classifiers

(37,853 labeled alerts)

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 audit*).

- 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).

- L. Flynn and Z. Kurtz. "Using Test Suites for Static Analysis Alert Classifiers", https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=526030
- *Nathaniel Ayewah and William Pugh, "The Google FindBugs Fixit," Proceedings of the 19th International Symposium on Software Testing and Analysis, ACM, 2010.

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.

Lori Flynn, Ebonie McNeil, and Matt Sisk. "Open Dataset RC_Data for Classifier Research", https://wiki.sei.cmu.edu/confluence/display/seccode/Open+Dataset+RC_Data+for+Classifier+Research
L. Flynn, E. McNeil, J. Yankel. "How to Instantiate SCAIFE API Calls: Using SEI SCAIFE Code, the SCAIFE API, Swagger-Editor, and Developing Your Tool with Auto-Generated Code", https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=644354
Lori Flynn. "Managing Static Analysis Alerts with Efficient Instantiation of the SCAIFE API into Code and an Automatically Classifying System" https://insights.sei.cmu.edu/author/lori-flynn/

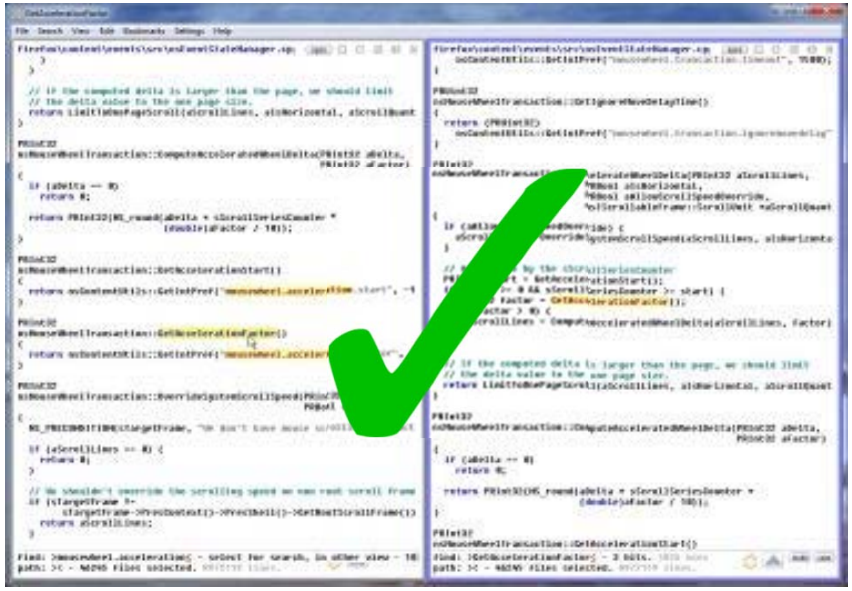
Finding Code Defects Using AI: Cross-Project Prediction

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]

[Baishakhi] Ray, Baishakhi, et al. "On the naturalness of buggy code." ICSE, 2016.
[Flynn] L. Flynn and Z. Kurtz. “Using Test Suites for Static Analysis Alert Classifiers”, https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=526030
[Wang] Wang, Song, Taiyue Liu, and Lin Tan. "Automatically learning semantic features for defect prediction." ICSE, 2016.
[Zhang] Zhang, Feng, et al. "Cross-project defect prediction using a connectivity-based unsupervised classifier." ICSE, 2016.

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

Sources: Carson D. Sestili, William S. Snavely, Nathan M. VanHoudnos, Towards security defect prediction with AI, Sep 12, 2018, https://arxiv.org/abs/1808.09897

Song Wang, Taiyue Liu, and Lin Tan. 2016. Automatically learning semantic features for defect prediction. In *Proceedings of the 38th International Conference on Software Engineering* (ICSE '16). ACM, New York, NY, USA, 297-308. DOI: https://doi.org/10.1145/2884781.2884804

Uri Alon, Meital Zilberstein, Omer Levy, and Eran Yahav. 2019. code2vec: learning distributed representations of code. Proc. ACM Program. Lang. 3, POPL, Article 40 (January 2019), 29 pages. DOI: https://doi.org/10.1145/3290353

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: A. Householder, Announcing CERT Basic Fuzzing Framework Version 2.8, Oct. 5, 2016, https://insights.sei.cmu.edu/cert/2016/10/announcing-cert-basic-fuzzing-framework-bff-28.html

G. Yan, J. Lu, Z. Shu ; Y. Kucuk, "ExploitMeter: Combining Fuzzing with Machine Learning for Automated Evaluation of Software Exploitability," 2017 IEEE Symposium on Privacy-Aware Computing (PAC), 1-4 Aug. 2017, https://doi.org/10.1109/PAC.2017.10

D. She, K. Pei, D. Epstein, J. Yang, B. Ray, S. Jana, "NEUZZ: Efficient Fuzzing with Neural Program Smoothing," 40th IEEE Symposium on Security and Privacy, May 20--22, 2019, San Francisco, CA, USA, https://arxiv.org/pdf/1807.05620.pdf

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

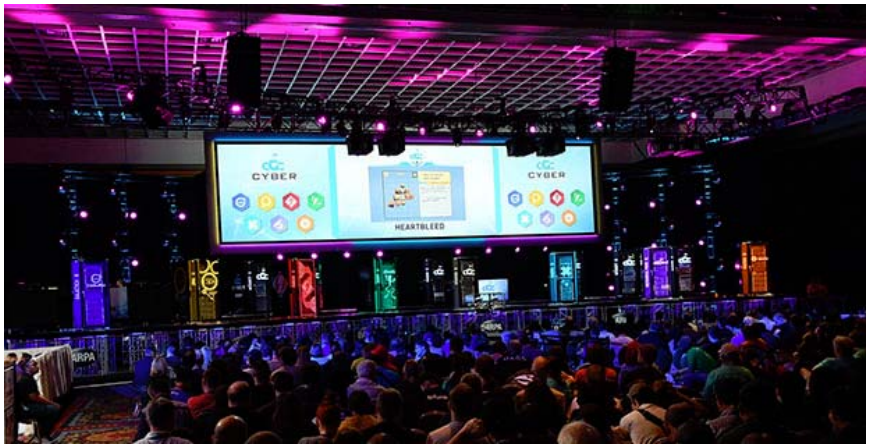
Sources:

K. Durkota and V. Lisy, "Computing Optimal Policies for Attack Graphs with Action Failures and Costs," Conference: Proceedings of the 7th Starting AI Researchers' Symposium (STAIRS), December 2013, https://www.researchgate.net/profile/Karel_Durkota/publication/273640839_Computing_Optimal_Policies_for_Attack_Graphs_with_Action_Failures_and_Costs

C. Sarraute, O. Buffet, and J. Hoffmann, "POMDPs Make Better Hackers: Accounting for Uncertainty in Penetration Testing," AAAI, 2012, https://arxiv.org/pdf/1307.8182

J. Schwartz, "Autonomous Penetration testing using Reinforcement Learning, Nov 16, 2018, https://arxiv.org/ftp/arxiv/papers/1905/1905.05965.pdf

Automated Program Repair – DARPA Cyber Grand Challenge

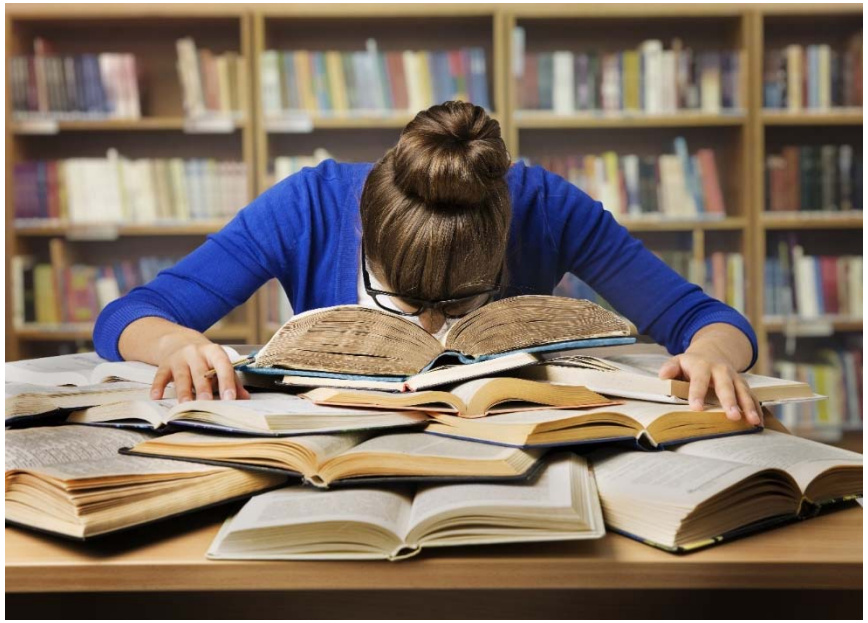


"Mayhem" demonstrated automated cyber defense

- Detect attack on program
- Analyze changes to program
- Deploy updated software

Source: DARPA, "Mayhem" Declared Preliminary Winner of Historic Cyber Grand Challenge, Aug 4, 2016, https://www.darpa.mil/news-events/2016-08-04

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

Source: Mark, Sherman, Verifying Software Assurance with IBM's Watson, https://www.youtube.com/watch?v=aW3497xhypY, Sep 11, 2017

AI Attacks Are Different

Pixel Manipulation

Feature Differentiation

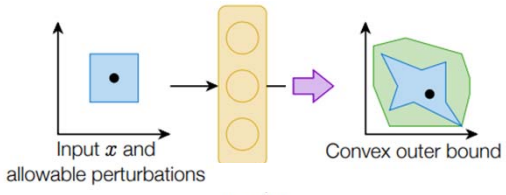
Source: Athalye, A., Engstrom, L., Ilyas, A., & Kwok, K. (2017, July 24). *Synthesizing Robust Adversarial Examples*. *arXiv [cs.CV]*. Retrieved from http://arxiv.org/abs/1707.07397

Source: Mahmood Sharif, Sruti Bhagavatula, Lujo Bauer, and Michael K. Reiter. 2016. Accessorize to a Crime: Real and Stealthy Attacks on State-of-the-Art Face Recognition. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security* (CCS '16). ACM, New York, NY, USA, 1528-1540. DOI: https://doi.org/10.1145/2976749.2978392

Some Technical Approaches for Defending AI Systems

Training Defenses

Wong & Kolter (2017)
output bound



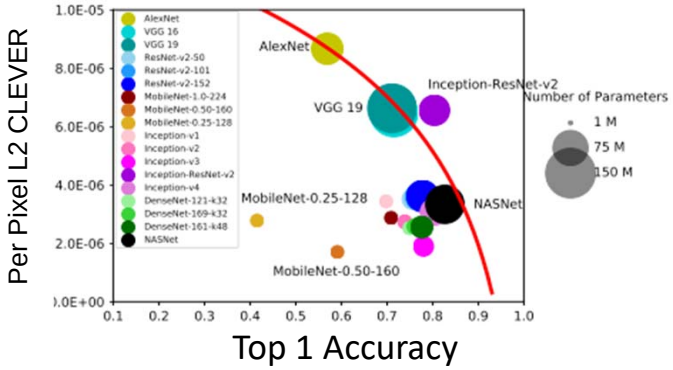
Causal Defenses

Tsipras et al. (2018)
adversarial data augmentation

Turtle → Bird

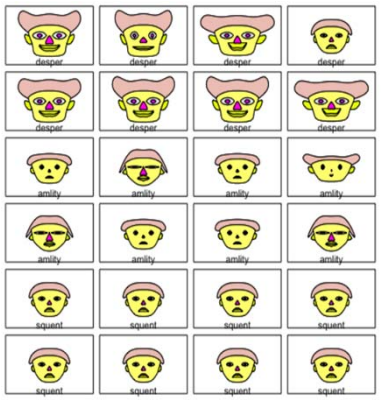
Engineering Defenses

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



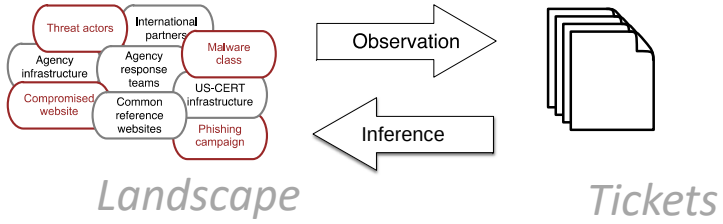
Sources:
Wong, E., & Kolter, J. Z. (2017). Provable defenses against adversarial examples via the convex outer adversarial polytope. ArXiv:1711.00851 [Cs, Math]. Retrieved from http://arxiv.org/abs/1711.00851;
Tsipras, D., Santurkar, S., Engstrom, L., Turner, A., & Madry, A. (2018). Robustness May Be at Odds with Accuracy. ArXiv:1805.12152 [Cs, Stat]. Retrieved from http://arxiv.org/abs/1805.12152;
Su, D., Zhang, H., Chen, H., Yi, J., Chen, P.-Y., & Gao, Y. (2018). Is Robustness the Cost of Accuracy? – A Comprehensive Study on the Robustness of 18 Deep Image Classification Models. ArXiv:1808.01688 [Cs]. Retrieved from http://arxiv.org/abs/1808.01688;
Ling Huang, Anthony D. Joseph, Blaine Nelson, Benjamin I.P. Rubinstein, and J. D. Tygar. 2011. Adversarial machine learning. In Proceedings of the 4th ACM workshop on Security and artificial intelligence (AISec '11). ACM, New York, NY, USA, 43-58. DOI=http://dx.doi.org/10.1145/2046684.2046692

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

Contact Us

Carnegie Mellon University

Software Engineering Institute

4500 Fifth Avenue

Pittsburgh, PA 15213-2612

info@sei.cmu.edu

www.sei.cmu.edu