Automating Static Analysis Alert Handling with Machine Learning

Lori Flynn, PhD

Software Security Researcher

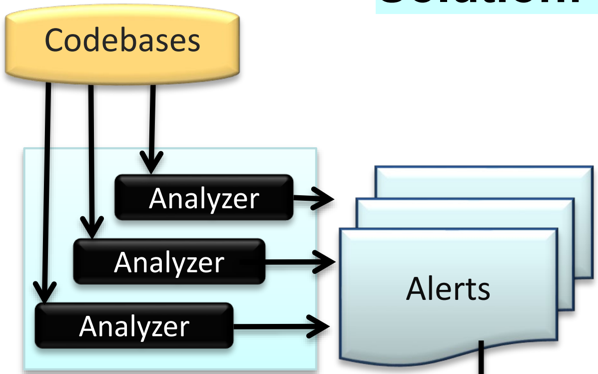
Software Engineering Institute of Carnegie Mellon University

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.

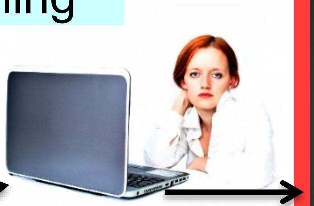
DM18-0727

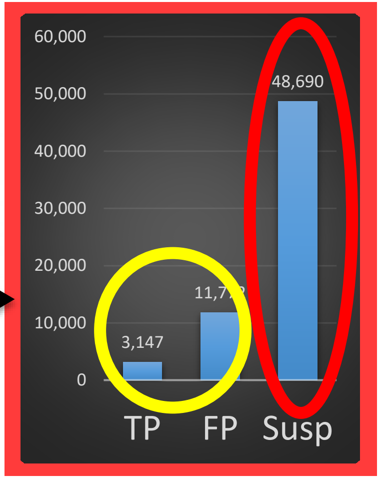
Overview

Problem: too many alerts
Solution: automate handling



Today





Project Goal

Classification algorithm development using “pre-audited” and manually-audited data, that

accurately classifies most of the diagnostics as:

Expected True Positive (e-TP) or
Expected False Positive (e-FP),
and
the rest as Indeterminate (I)

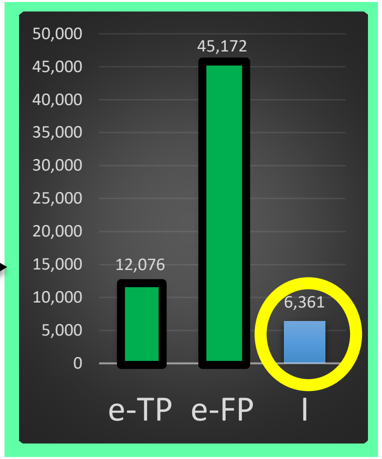
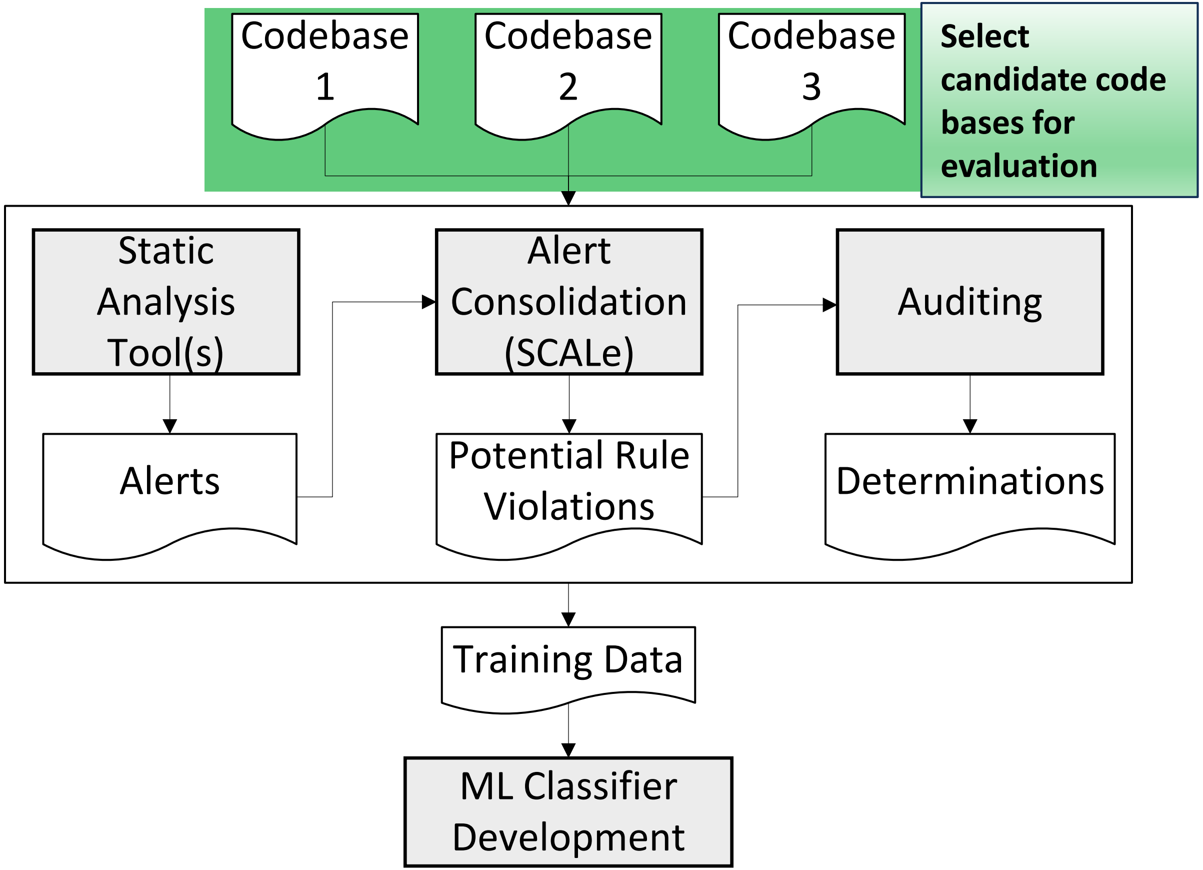
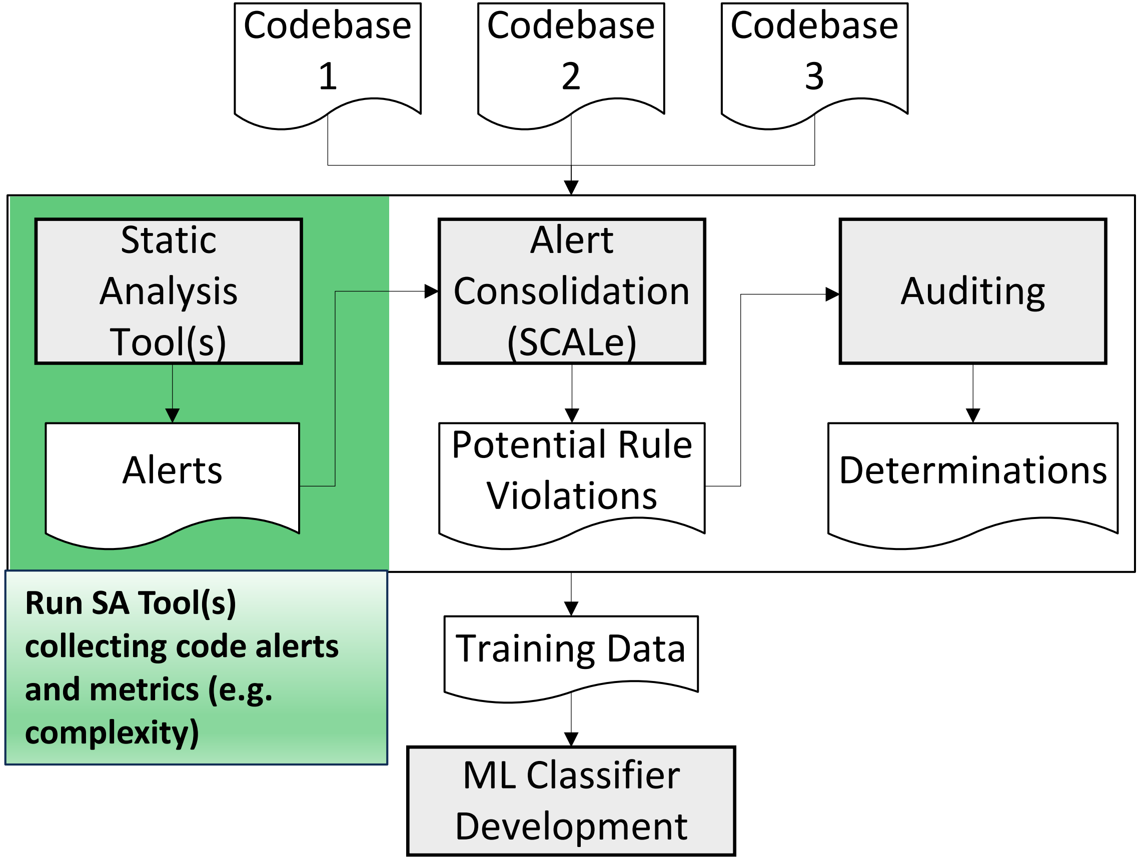


Image of woman and laptop from http://www.publicdomainpictures.net/view-image.php?image=47526&picture=woman-and-laptop “Woman And Laptop”

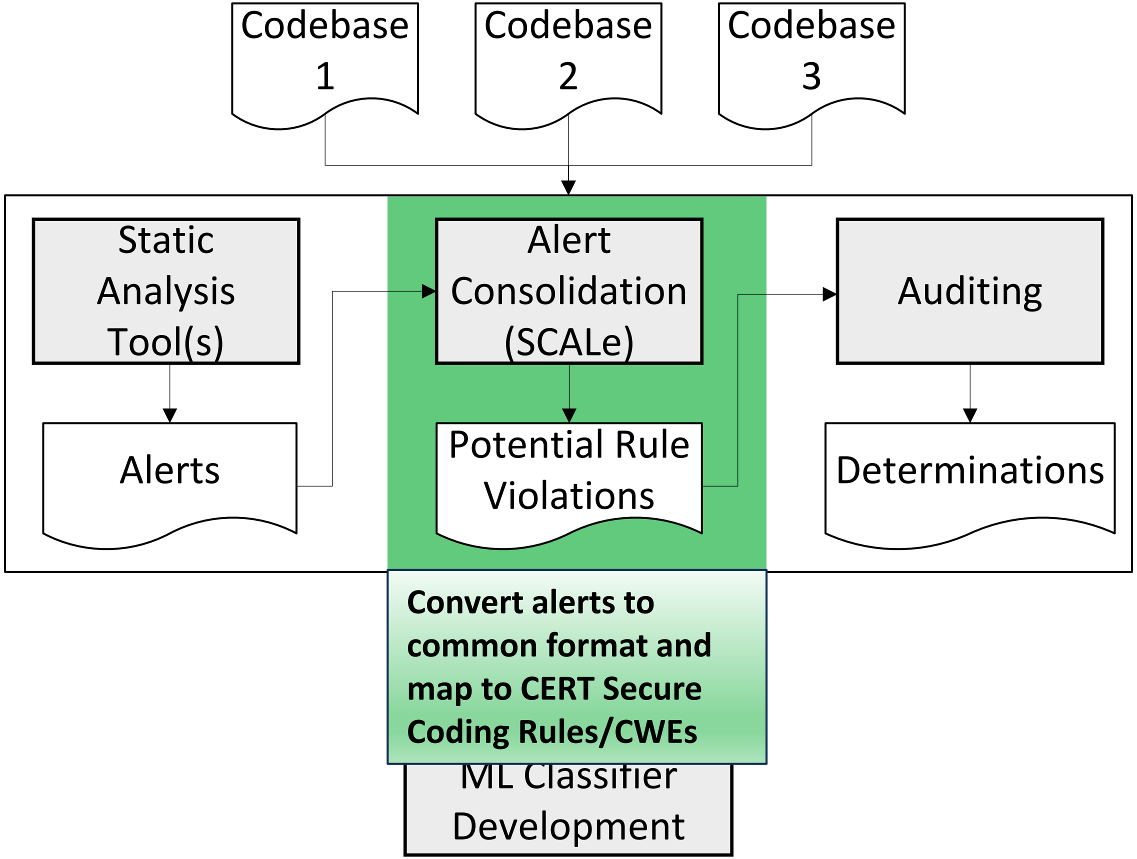
Background: Automatic Alert Classification



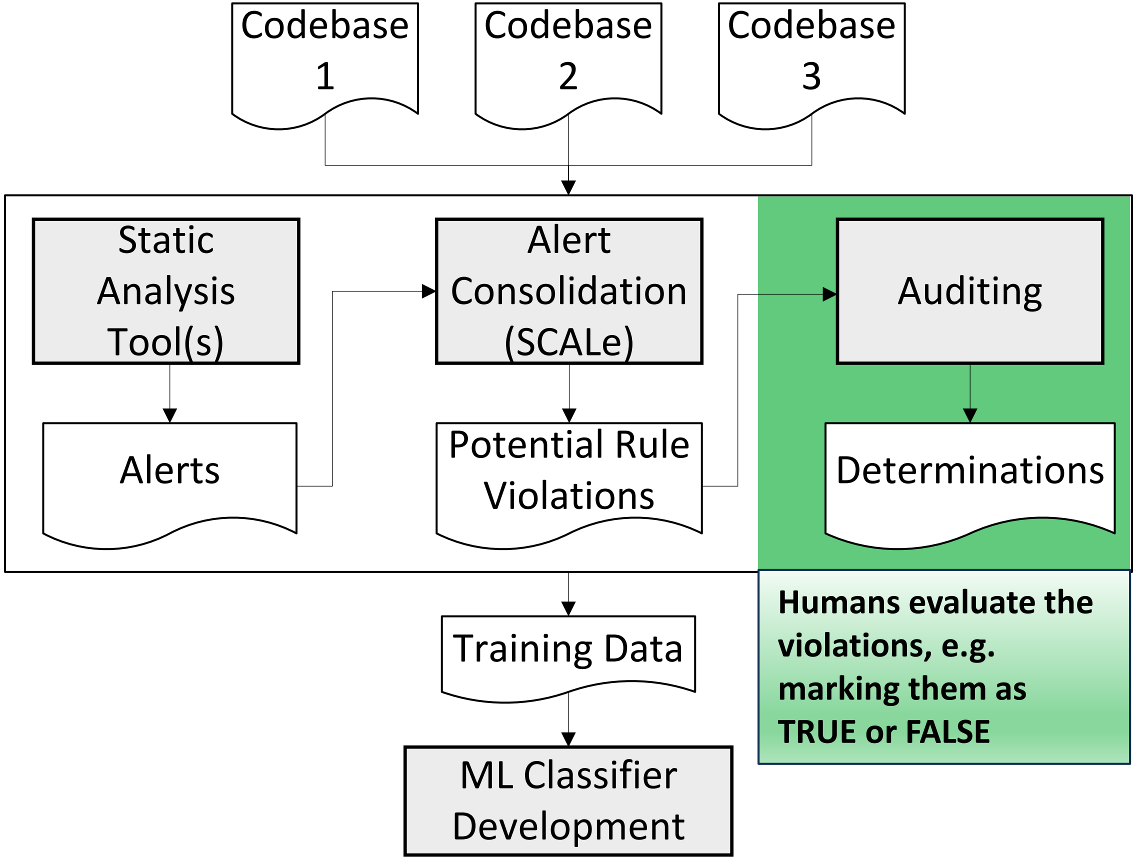
Background: Automatic Alert Classification



Background: Automatic Alert Classification

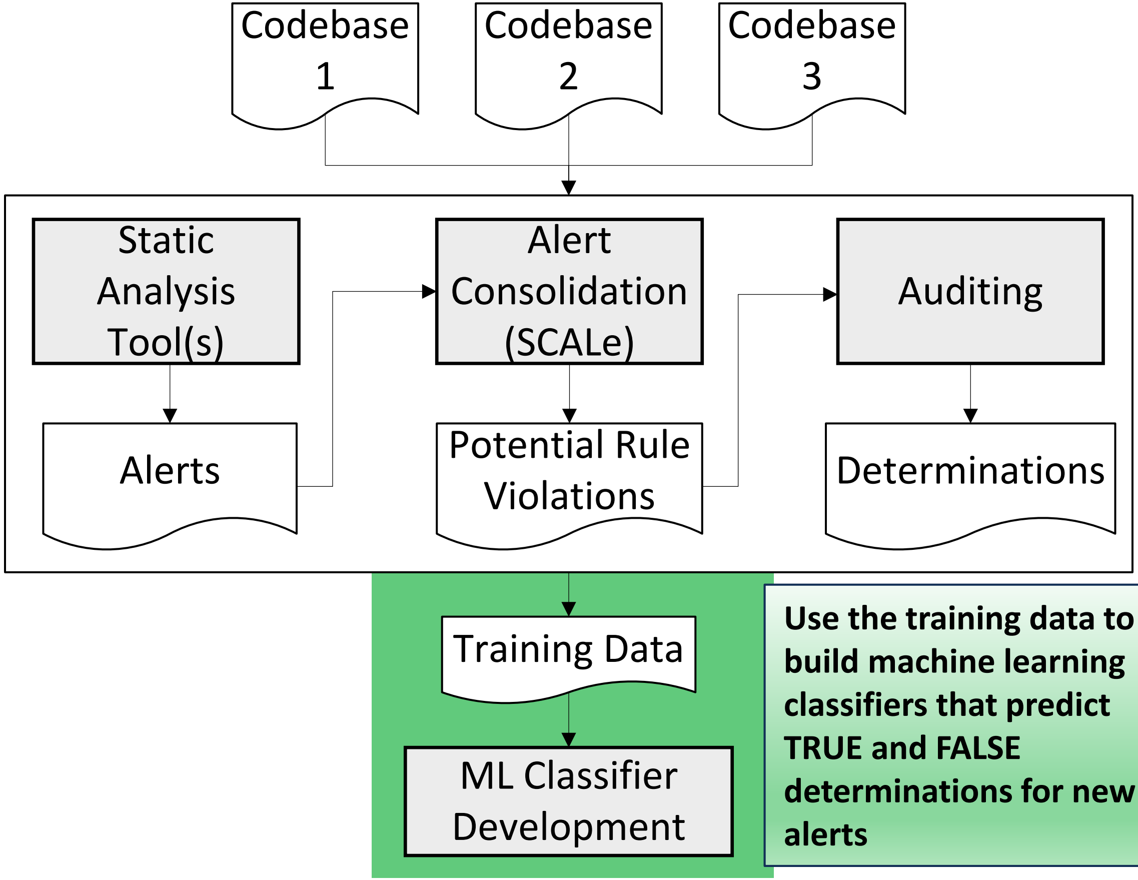


Background: Automatic Alert Classification



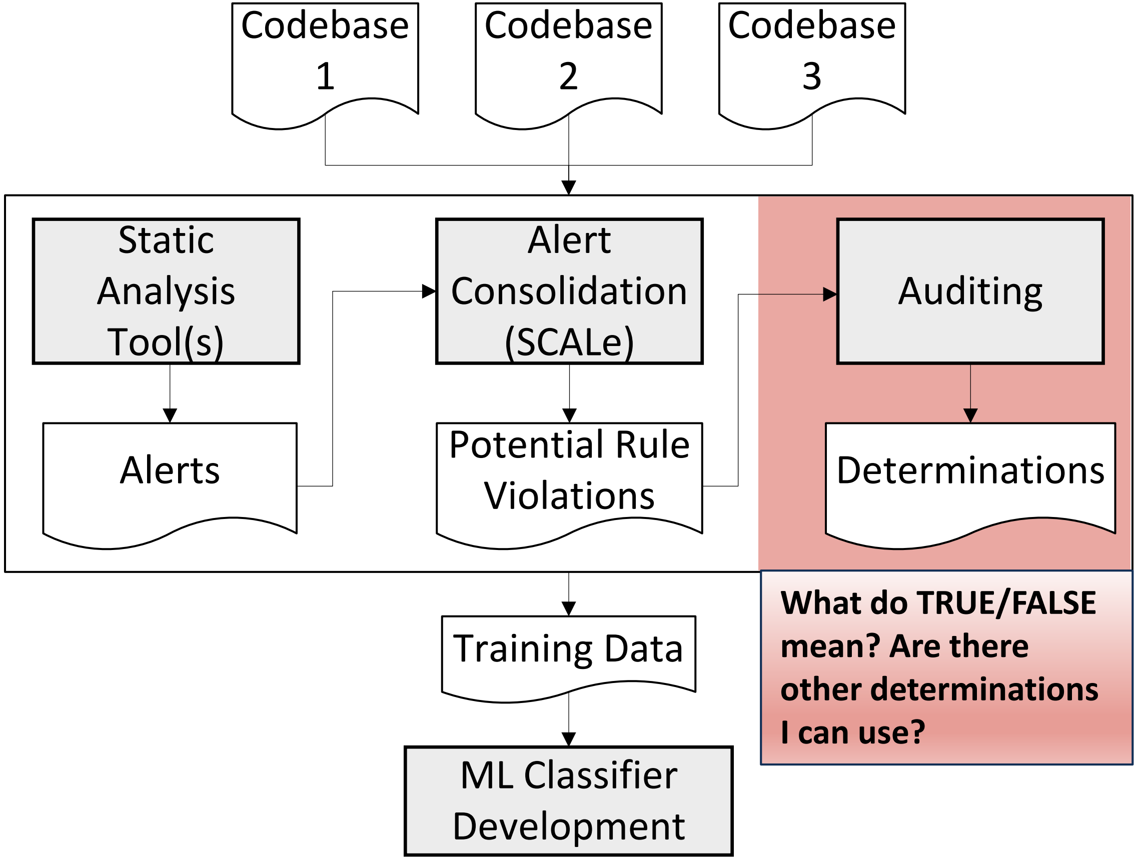
Humans evaluate the violations, e.g. marking them as TRUE or FALSE

Background: Automatic Alert Classification



Use the training data to build machine learning classifiers that predict TRUE and FALSE determinations for new alerts

Background: Automatic Alert Classification

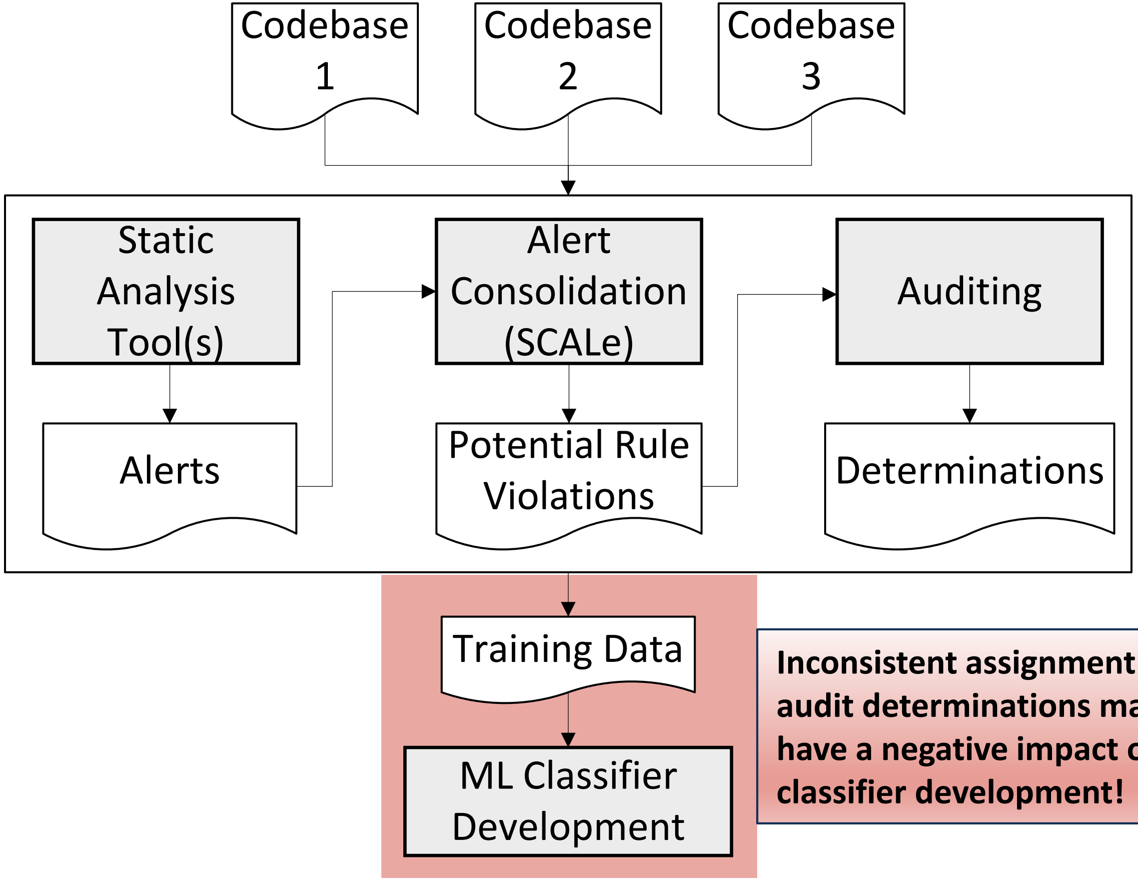


What is truth?

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**!

Background: Automatic Alert Classification



Inconsistent assignment of audit determinations may have a negative impact on classifier development!

Solution: Lexicon And Rules

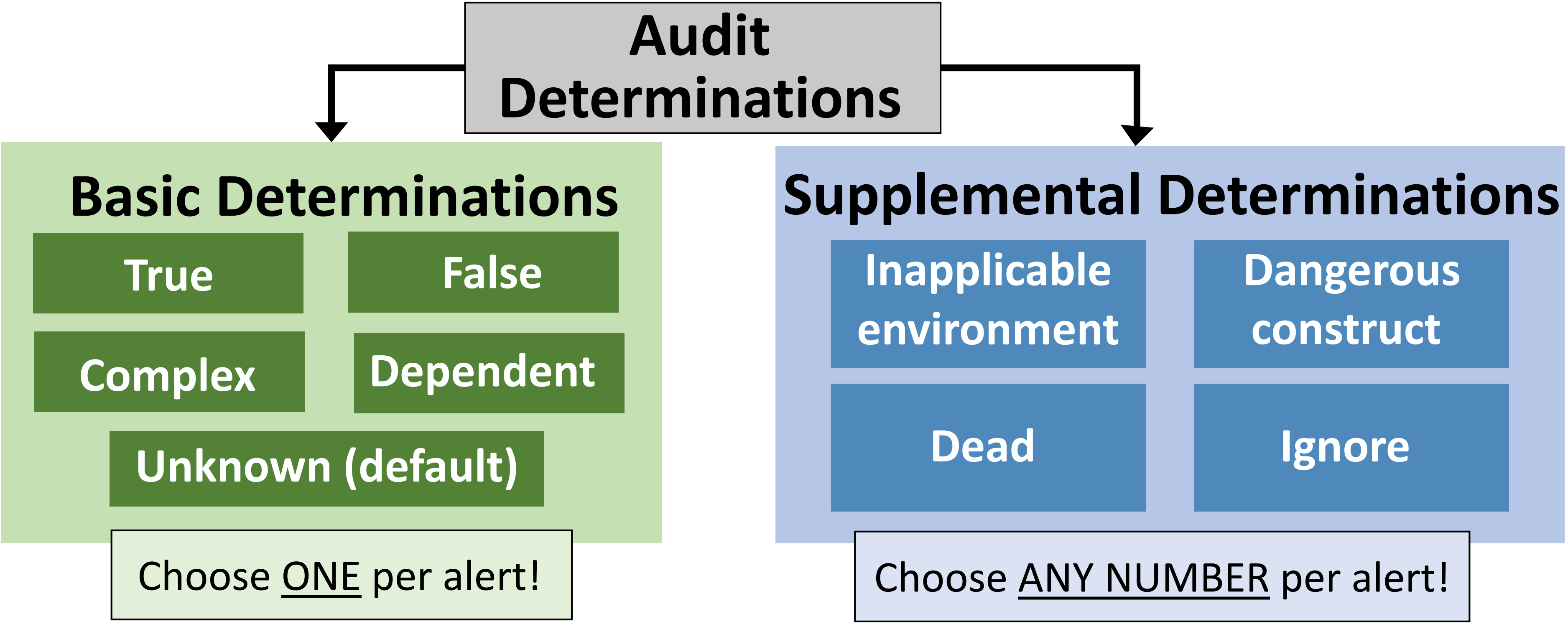
- We developed a **lexicon** and auditing **rule set** for our collaborators
- Includes a standard set of well-defined **determinations** for static analysis alerts
- Includes a set of **auditing rules** to help auditors make consistent decisions in commonly-encountered situations

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**.

Lexicon: Audit Determinations



Audit Rules

Goals

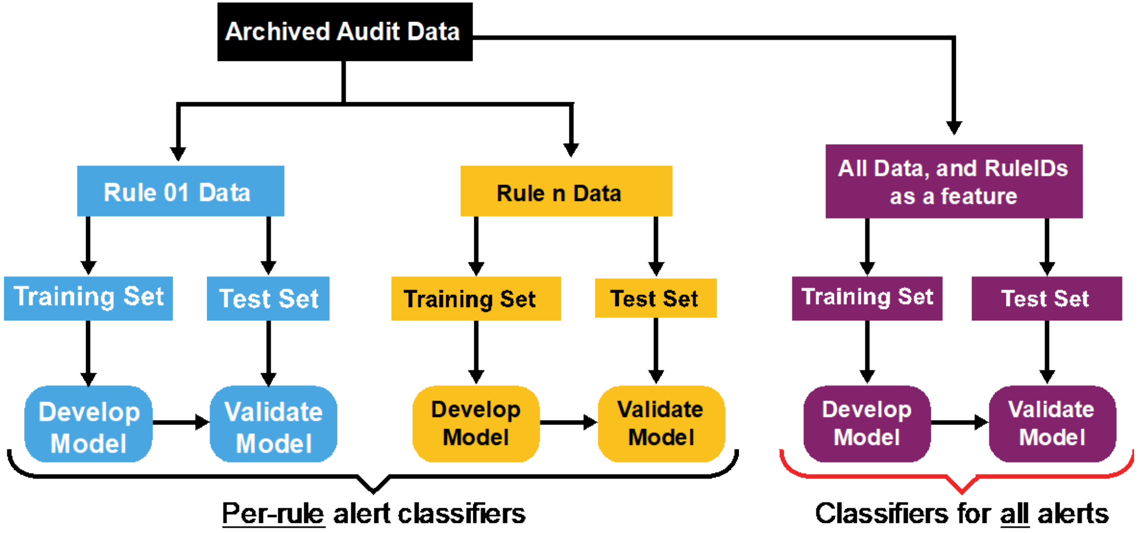
- Clarify **ambiguous or complex** auditing scenarios
- Establish **assumptions** auditors can make
- Overall: help make audit determinations **more consistent**

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

Machine Learning with Static Analysis Audit Archives

Combined use of:
1) multiple analyzers, 2) variety of features,
3) competing classification techniques



Problem: too many alerts
Solution: automate handling

Competing Classifiers to Test
Lasso Logistic Regression
CART (Classification and Regression Trees)
Random Forest
Extreme Gradient Boosting (XGBoost)

Some of the features used (many more)
Analysis tools used
Significant LOC
Complexity
Coupling
Cohesion
SEI coding rule

Data Used for Classifiers

Data used to create and validate classifiers:

- CERT-audited alerts:
  - ~7,500 audited alerts
- 3 collaborators audit their own codebases with our auditing research prototype tool “enhanced SCALe”

We pooled data (CERT + collaborators) and segmented it:

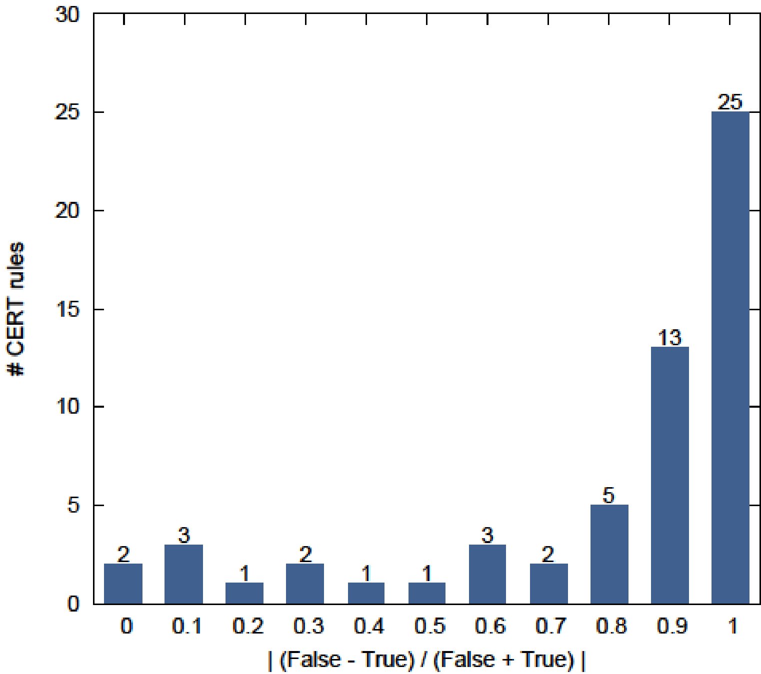
- Segment 1 (70% of data): train model
- Segment 2 (30% of data): testing

Added classifier variations on dataset:

- Per-rule
- Per-language
- With/without tools
- Others

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



Archive sanitizer: enabled collaborator data use

Added data sanitizer to “enhanced SCALe”

- Anonymizes sensitive fields
- SHA-256 hash with salt
- Enables analysis of features correlated with alert confidence

Audit archive for project is in a database

- DB fields may contain sensitive information
- Sanitizing script anonymizes or discards fields
  - Diagnostic message
  - Path, including directories and filename
  - Function name
  - Class name
  - Namespace/package
  - Project filename

Classifier Result Highlights: Data All Sources

Classifiers made from all data, pooled:

All-rules (158) classifier accuracy:
- Lasso Logistic Regression: 88%
- Random Forest: 91%
- CART: 89%
- XGBoost: 91%

Single-rule classifier accuracy:

Rule ID	Lasso LR	Random Forest	CART	XGBoost
INT31-C	98%	97%	98%	97%
EXP01-J	74%	74%	81%	74%
OBJ03-J	73%	86%	86%	83%
FIO04-J*	80%	80%	90%	80%
EXP33-C*	83%	87%	83%	83%
EXP34-C*	67%	72%	79%	72%
DCL36-C*	100%	100%	100%	100%
ERR08-J*	99%	100%	100%	100%
IDS00-J*	96%	96%	96%	96%
ERR01-J*	100%	100%	100%	100%
ERR09-J*	100%	88%	88%	88%

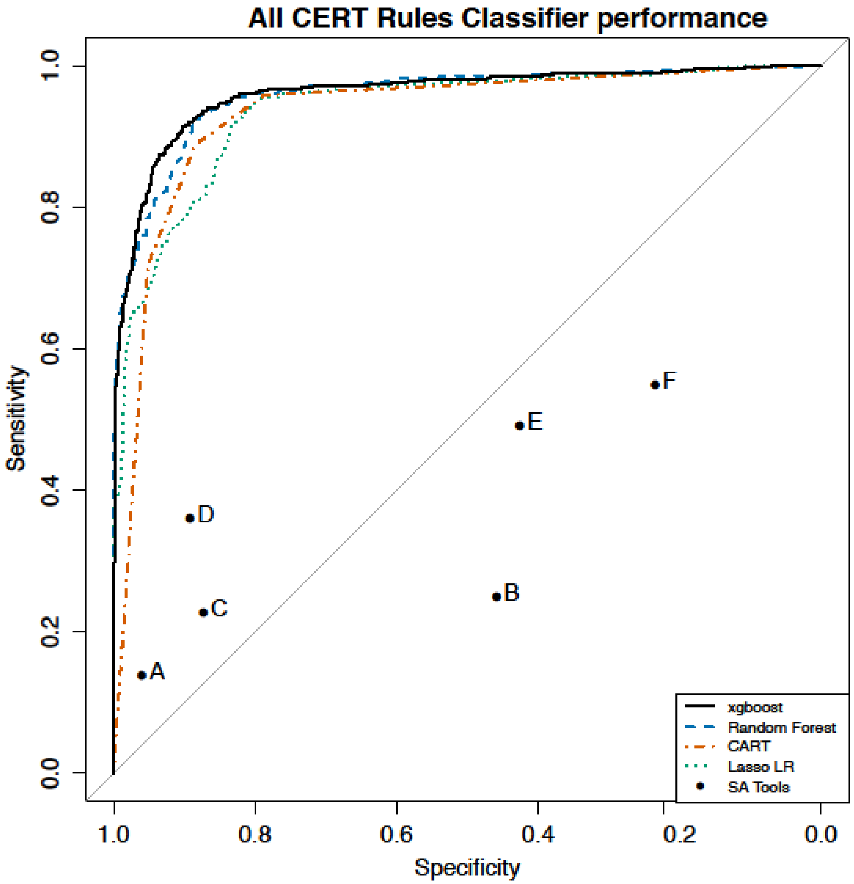
Also, 15 one-way "classifiers".

General results (not true for every test)

- Classifier accuracy rankings for all-pooled test data:
  XGBoost ≈ RF > CART ≈ LR
- Classifier accuracy rankings for collaborator test data:
  LR ≈ RF > XGBoost > CART
- Per-rule classifiers generally not useful (lack data), but 3 rules (INT31-C best) are exceptions.
- With-tools-as-feature classifiers better than without.
- Accuracy of single language vs. all-languages data:
  C > all-combined > Java

* Small quantity of data, results suspect

Tool as Feature Helped



Using toolname as a feature improved classifier performance

Dots show performance of tool alone

Rapid Expansion of Alert Classification

Problem 1: too many alerts
Solution 1: automate handling

Problem 2

Too few manually audited alerts to make classifiers (i.e., to automate!)

Problems 1 & 2: Security-related code flaws detected by static analysis require too much manual effort to triage, plus it takes too long to audit enough alerts to develop classifiers to automate the triage accurately for many types of flaws.

Extension of our previous alert classification work to address challenges:

1. Too few audited alerts for accurate classifiers for many flaw types
2. Manually auditing alerts is expensive

Solution 2

Automate auditing alerts, using test suites

Solution for 1 & 2: Rapid expansion of number of conditions with labeled alerts by using test suites, plus collaborator audits of DoD code.

Approach

1. Automated analysis of test suite programs to label data for many conditions for classifiers

2. Collaboration with MITRE: Systematically map CERT rules to CWE IDs

3. Test classifiers on alerts from real-world code: DoD data

Overview: Method, Approach, Validity

Problem 2: too few manually audited alerts to make accurate classifiers for many flaw types
Solution 2: automate auditing alerts, using test suites

Create alert classifiers trained on many conditions, then use DoD-audited data to validate the classifiers.

Technical methods:

- Use test suites' CWE flaw metadata, to quickly and automatically generate many "audited" alerts.
  - Juliet (NSA CAS) 61,387 C/C++ tests
  - IARPA's STONESOUP: 4,582 C tests
  - Refine test sets for rules: use **mappings, metadata, static analyses**
- Metrics analyses of test suite code, to get feature data
- Use DoD-collaborator SCALe **audits** of their own codebases, to validate classifiers. **Real codebases with more complex structure than most pre-audited code**.

Make Mappings Precise

Problem 2: too few manually audited alerts to make classifiers
Solution 2: automate auditing alerts, using test suites

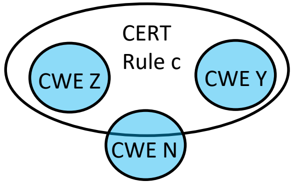
Problem 3: Test suites in different taxonomies (most use CWEs)
Solution 3: Precisely map between taxonomies, then partition tests using precise mappings

Precise mappings: Defines *what kind* of non-null relationship, and if overlapping, *how.*
Enhanced-precision added to "imprecise" mappings.

Imprecise mappings
("*some* relationship")

Precise mappings
(set notation, often more)

2 CWEs subset of CERT rule,
AND partial overlap



Mappings	
Precise	248
Imprecise TODO	364
Total	**612**

Now: all CERT C rules mappings to CWE precise

If a **condition** of a program violates a CERT rule *R* and also exhibits a CWE weakness *W*, that **condition** is in the overlap.

Test Suite Cross-Taxonomy Use

Problem 3: Test suites in different taxonomies (most use CWEs)
Solution 3: Precisely map between taxonomies, **then partition tests with precise mappings**

Partition sets of thousands of tests relatively quickly.

Examine together:

- Precise mapping
- Test suite metadata (structured filenames)
- Rarely examine small bit of code (variable type)

CWE test programs useful to test CERT rules

STONESOUP: **2,608** tests

Juliet: **80,158** tests

Some types of CERT rule violations not tested, in partitioned test suites ("**0**"s).

- Possible coverage in other suites

CERT rule	CWE	Count files that match
ARR38-C	CWE-119	**0**
ARR38-C	CWE-121	6,258
ARR38-C	CWE-122	2,624
ARR38-C	CWE-123	**0**
ARR38-C	CWE-125	**0**
ARR38-C	CWE-805	2,624
INT30-C	CWE-190	1,548
INT30-C	CWE-191	1,548
INT30-C	CWE-680	984
INT32-C	CWE-119	**0**
INT32-C	CWE-125	**0**
INT32-C	CWE-129	**0**
INT32-C	CWE-131	**0**
INT32-C	CWE-190	3,875
INT32-C	CWE-191	3,875
INT32-C	CWE-20	**0**
INT32-C	CWE-606	**0**
INT32-C	CWE-680	984

Process

Generate data for Juliet

Generate data for STONESOUP

Write classifier development and testing scripts

Build classifiers

- Directly for CWEs
- Using partitioned test suite data for CERT rules

Test classifiers

Problem 1: too many alerts

Solution 1: automate handling

Problem 2: too few manually audited alerts to make classifiers accurate for some flaws

Solution 2: automate auditing alerts, using test suites

Problem 3: Test suites in different taxonomies (most use CWEs)

Solution 3: Precisely map between taxonomies, then partition tests using precise mappings

Analysis of Juliet Test Suite: Initial CWE Results

- We automated defect identification of Juliet flaws with location **2 ways**
  - A Juliet program tells about only one type of CWE
  - Exact line defect metadata, for TPs
  - Function line spans, for FPs

Number of “**Bad**” Functions	103,376
Number of “**Good**” Functions	231,476

- Used 8 static analysis tools on Juliet programs
- Automated alert-to-defect matching
- Automated alert-to-alert matching (alerts fused: same line & CWE)

Lots of new data for creating classifiers

Alert Type	**Equivalence Classes:** (EC counts a fused alert once)
TRUE	**13,330**
FALSE	**24,523**

- These are initial metrics (more EC as use more tools, STONESOUP)

Analysis of Juliet Test Suite: Initial CWE Results

Lots of new data for creating classifiers

Alert Type	Equivalence Classes: (EC counts a fused alert once)
TRUE	**13,330**
FALSE	**24,523**

- Big savings: manual audit of 37,853 alerts from non-test-suite programs would take:
  - **Unrealistic minimum: 1,230 hours** (117 seconds per alert audit, Pugh and Ayewah)
  - First 37,853 alert audits wouldn't cover many conditions (and sub-conditions) in the Juliet test suite!
  - Need true and false labels for classifiers
  - Much time and computation to run static analysis tools on many non-test-suite programs
  - **Realistically: enormous amount of manual auditing time**, to develop that much data.
- These are initial metrics (we will have more data as we use more tools and test suites)

Juliet Test Suite Classifiers: Initial Results (Hold-out Data)

Classifier	Accuracy	Precision	Recall	AUROC
rf	0.938	0.893	0.875	0.991
lightgbm	0.942	0.902	0.882	0.992
xgboost	0.932	0.941	0.798	0.987
lasso	0.925	0.886	0.831	0.985

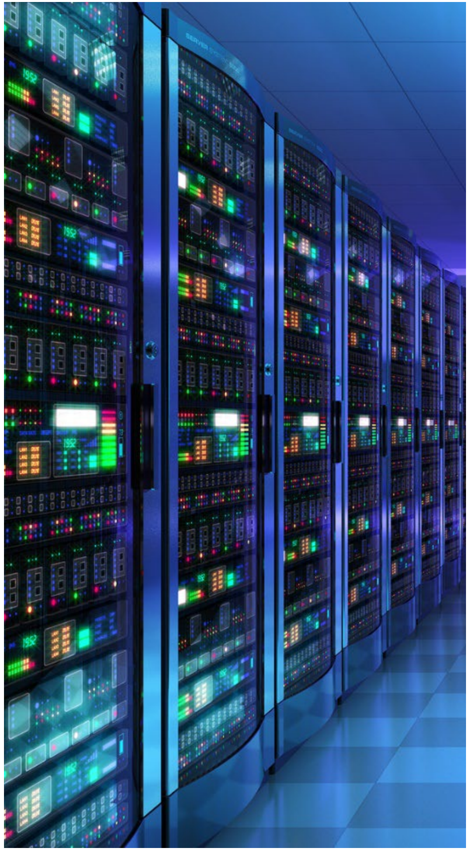
Summary and Future

- Goal: increase automation of static alert auditing, using machine learning
- Developed large archive of labeled alerts
  - For CWEs and CERT rules
- Developed code infrastructure (extensible)
- Developed general method to use test suites across taxonomies
- In-progress:
  - Classifier development and testing in process
  - Major focus: Cross-project and adaptive heuristics
  - Continue to gather data
  - Modified SCALe audit tool for new collaborator testing

<u>Publications:</u>

- IEEE SecDev 2017 “Hands-on Tutorial: Alert Auditing with Lexicon & Rules”
- Research papers (SQUADE’18), others in progress
- New mappings (CWE/CERT rule): CERT Secure Coding C Standard wiki
- SEI blogposts on classifier development

CERT: Applied Machine Learning in Cybersecurity



Automating Static Analysis Alert Handling

Neural Nets for finding coding bugs

Automated malware family classification

Cyberattack forecasting

Security Operations Center optimization

Protection against AI poisoning

Relation of kinetic and cyber actions

Technical debt estimation

Cognitive support for assurance using Watson

Email sentiment analysis

IoT-based search-and-rescue

Contact Information

Presenter / Point(s) of Contact

Lori Flynn (Principal Investigator)

Software Security Researcher

Email: lflynn@cert.org

Telephone: +1 412.268.7886

Additional Contributors

SEI Staff

William Snavely   Zach Kurtz

Ebonie McNeil

David Svoboda

SEI Student Interns

Lucas Bengtson (CMU)

Charisse Haruta (CMU)

Baptiste Vauthey (CMU)

Michael Spece (Pitt)

Christine Baek (CMU)