Challenges and Progress: Automating Static Analysis Alert Handling with Machine Learning

Lori Flynn, PhD
Software Security Researcher
Software Engineering Institute of Carnegie Mellon University
Copyright 2018 Carnegie Mellon University. All Rights Reserved.

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.

The view, opinions, and/or findings contained in this material are those of the author(s) and should not be construed as an official Government position, policy, or decision, unless designated by other documentation.

NO WARRANTY. THIS CARNEGIE MELLON UNIVERSITY AND SOFTWARE ENGINEERING INSTITUTE MATERIAL IS FURNISHED ON AN "AS-IS" BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT.

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution. Please see Copyright notice for non-US Government use and distribution.

This material may be reproduced in its entirety, without modification, and freely distributed in written or electronic form without requesting formal permission. Permission is required for any other use. Requests for permission should be directed to the Software Engineering Institute at permission@sei.cmu.edu.


DM18-0547
Overview

Problem: too many alerts
Solution: automate handling

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)
Background: Automatic Alert Classification

- Static Analysis Tool(s)
- Alert Consolidation (SCALe)
- Auditing

- Alerts
- Potential Rule Violations
- Determinations

- Training Data
- ML Classifier Development

Select candidate code bases for evaluation
Background: Automatic Alert Classification

- Run SA Tool(s) collecting code alerts and metrics (e.g. complexity)
  - Static Analysis Tool(s)
  - Alerts
  - Alert Consolidation (SCALE)
  - Potential Rule Violations
  - Auditing
  - Determinations
  - Training Data
  - ML Classifier Development
Background: Automatic Alert Classification

Static Analysis Tool(s) → Alerts

Alert Consolidation (SCALe) → Potential Rule Violations

Codebase 1 → Codebase 2 → Codebase 3

Auditing → Determinations

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

ML Classifier Development
Background: Automatic Alert Classification

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

Training Data → ML Classifier Development

Humans evaluate the violations, e.g. marking them as TRUE or FALSE
Background: Automatic Alert Classification

- **Static Analysis Tool(s)**
- **Alerts**
- **Alert Consolidation (SCALe)**
- **Potential Rule Violations**
- **Auditing**
- **Determinations**

**Training Data**

Use the training data to build machine learning classifiers that predict TRUE and FALSE determinations for new alerts.
Background: Automatic Alert Classification

Static Analysis Tool(s) → Alerts

Alert Consolidation (SCALE) → Potential Rule Violations

Training Data → ML Classifier Development

Auditing → Determinations

What do TRUE/FALSE mean? Are there other determinations I can use?
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**. |
Audit Lexicon And Rules

Lexicon
Lexicon: Audit Determinations

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

Choose ONE Per Alert!

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

Choose ANY NUMBER Per Alert!
Lexicon: Basic Determinations

True

• The code in question violates the condition indicated by the alert.

• A condition is a constraint or property of validity.
  - E.g. A valid program should not deference NULL pointers.

• The condition can be determined from the definition of the alert itself, or from the coding taxonomy the alert corresponds to.
  - CERT Secure Coding Rules
  - CWEs
Lexicon: Basic Determinations
True Example

```c
char *build_array(size_t size, char first) {
    if(size == 0) {
        return NULL;
    }

    char *array = malloc(size * sizeof(char));
    array[0] = first;
    return array;
}
```

**Alert:** Do not dereference NULL pointers!

**Determination:** TRUE
Lexicon: Basic Determinations

False

• The code in question does **not** violate the **condition** indicated by the alert.

```c
char *build_array(int size, char first) {
    if(size == 0) {
        return NULL;
    }

    char *array = malloc(size * sizeof(char));
    if(array == NULL) {
        abort();
    }
    array[0] = first;
    return array;
}
```

**Alert:** Do not dereference NULL pointers!

**Determination:** FALSE
Lexicon: Basic Determinations

Complex
• The alert is too difficult to judge in a reasonable amount of time and effort
  • “Reasonable” is defined by the individual organization.

Dependent
• The alert is related to a True alert that occurs earlier in the code.
  • Intuition: fixing the first alert would implicitly fix the second one.

Unknown
• None of the above. This is the default determination.
Lexicon: Basic Determinations

Dependent Example

```c
char *build_array(size_t size, char first, char last) {
    if(size == 0) {
        return NULL;
    }
    char *array = malloc(size * sizeof(char));
    array[0] = first;
    array[size - 1] = last;
    return array;
}
```

**Alert**: Do not dereference NULL pointers!

**Determination**: TRUE

**Alert**: Do not dereference NULL pointers!

**Determination**: DEPENDENT
Lexicon: Supplemental Determinations

Dangerous Construct
• The alert refers to a piece of code that poses risk if it is not modified.
• Risk level is specified as High, Medium, or Low
• Independent of whether the alert is true or false!

Dead
• The code in question not reachable at runtime.

Inapplicable Environment
• The alert does not apply to the current environments where the software runs (OS, CPU, etc.)
• If a new environment were added in the future, the alert may apply.

Ignore
• The code in question does not require mitigation.
Lexicon: Supplemental Determinations
Dangerous Construct Example

```c
#define BUF_MAX 128

void create_file(const char *base_name) {
    // Add the .txt extension!
    char filename[BUF_MAX];
    snprintf(filename, 128, "%s.txt", base_name);

    // Create the file, etc...
}

ALERT: potential buffer overrun!

Seems ok...but why not use BUF_MAX instead of 128?

Determination: False + Dangerous Construct
```
Audit Lexicon And Rules

Rules
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
• In the following slides, we will inspect three of the rules in more detail.
Example Rule: Assume external inputs to the program are malicious

An auditor should assume that inputs to a program module (e.g. function parameters, command line arguments, etc.) may have arbitrary, potentially malicious, values.

• Unless they have a strong guarantee to the contrary

Example from recent history: Java Deserialization

• Suppose an alert is raised for a call to readObject, citing a violation of the CERT Secure Coding Rule SER12-J, Prevent deserialization of untrusted data
• An auditor can assume that external data passed to the readObject method may be malicious, and mark this alert as True
  - Assuming there are no other mitigations in place in the code
Audit Rules
External Inputs Example

```java
import java.io.*;

class DeserializeExample {
    public static Object deserialize(byte[] buffer) throws Exception {
        ByteArrayInputStream bais;
        ObjectInputStream ois;
        bais = new ByteArrayInputStream(buffer);
        ois = new ObjectInputStream(bais);
        return ois.readObject();
    }
}
```

**ALERT**: Don’t deserialize untrusted data!

Without strong evidence to the contrary, assume the buffer could be malicious!

**Determination**: TRUE
Example Rule: Unless instructed otherwise, assume code must be portable.

When auditing alerts for a code base where the target platform is not specified, the auditor should err on the side of portability.

If a diagnosed segment of code malfunctions on certain platforms, and in doing so violates a condition, this is suitable justification for marking the alert True.
Audit Rules
Portability Example

```c
int strcmp(const char *str1, const char *str2) {
    while(*str1 == *str2) {
        if(*str1 == '\0') {
            return 0;
        }
        str1++;
        str2++;
    }
    if(*str1 < *str2) {
        return -1;
    } else {
        return 1;
    }
}
```

**ALERT**: Cast to unsigned char before comparing!

This code would be safe on a platform where chars are unsigned, but that hasn’t been guaranteed!

Determination: **TRUE**
Example Rule: Handle an alert in unreachable code depending on whether it is exportable.

Certain code segments may be **unreachable** at runtime. Also called **dead code**.

A static analysis tool might not be able to realize this, and still **mark alerts** in code that cannot be executed.

The **Dead** supplementary determination can be applied to these alerts.

However, an auditor should **take care** when deciding if a piece of code is truly dead.

In particular: just because a given program module (function, class) is not used does **not** mean it is dead. The module might be exported as a **public interface**, for use by another application.

This rule was developed as a result of a scenario encountered by one of our collaborators!
Scientific Approach

Build on novel (in FY16) 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
Rapid Expansion of Alert Classification

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 classification models by using “pre-audited” code, plus collaborator audits of DoD code.

Approach
1. Automated analysis of “pre-audited” (not by SEI) tests to gather sufficient code & alert feature info for classifiers
2. Collaboration with MITRE: Systematically map CERT rules to CWE IDs in subsets of “pre-audited” test code (known true or false for CWE)
3. Modify SCALe research tool to integrate CWE (MITRE’s Common Weakness Enumeration)
4. Test classifiers on alerts from real-world code: DoD data

Problem 1: too many alerts
Solution 1: automate handling
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

Rapidly create many coding-rule-level classifiers for static analysis alerts, 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 enhanced-SCALe audits of their own codebases, to validate classifiers. Real codebases with more complex structure than most pre-audited code.

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

Make Mappings Precise

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

Imprecise mappings ("some relationship")  Precise mappings (set notation, often more)

2 CWEs subset of CERT rule, AND partial overlap

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

<table>
<thead>
<tr>
<th>Mapping Type</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Precise</td>
<td>248</td>
</tr>
<tr>
<td>Imprecise TODO</td>
<td>364</td>
</tr>
<tr>
<td>Total</td>
<td>612</td>
</tr>
</tbody>
</table>

Now: all CERT C rules mappings to CWE precise
Test Suite Cross-Taxonomy Use

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
  • Test set partitioning incomplete (32% left)

Some types of CERT rule violations not tested, in partitioned test suites (“0”s).
  - Possible coverage in other suites
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
We automated defect identification of Juliet flaws with location 2 ways
- A Juliet program tells about only one type of CWE
- Bad functions definitely have that flaw
- Good functions definitely don’t have that flaw
- Function line spans, for FPs
- Exact line defect metadata, for TPs

We used static analysis tools on Juliet programs
- We automated alert-to-defect matching
  - Ignore unrelated alerts (other CWEs) for program
  - Alerts give line number

We automated alert-to-alert matching (alerts fused: same line & CWE)

**Analysis of Juliet Test Suite: Initial CWE Results**

<table>
<thead>
<tr>
<th>Tool</th>
<th>“Pre-audited” TRUE</th>
<th>“Pre-audited” FALSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tool A</td>
<td>1,655</td>
<td>8,539</td>
</tr>
<tr>
<td>Tool B</td>
<td>162</td>
<td>3,279</td>
</tr>
<tr>
<td>Tool C</td>
<td>7,225</td>
<td>2,394</td>
</tr>
<tr>
<td>Tool D</td>
<td>16,958</td>
<td>23,475</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>26,000</strong></td>
<td><strong>37,687</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Alert Type</th>
<th>Equivalence Classes: (EC counts a fused alert once)</th>
<th>Number of Alerts Fused (from different tools)</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>22,885</td>
<td>3,115</td>
</tr>
<tr>
<td>FALSE</td>
<td>29,507</td>
<td>8,180</td>
</tr>
</tbody>
</table>

- A Juliet program tells about only one type of CWE
- Bad functions definitely have that flaw
- Good functions definitely don’t have that flaw
- Function line spans, for FPs
- Exact line defect metadata, for TPs
- Ignore unrelated alerts (other CWEs) for program
- Alerts give line number

Lots of new data for creating classifiers!

- These are initial metrics (more EC as use more tools, STONESOUP)
Juliet: Data from 4 Tools, per CWE

35 CWEs with **at least** 5 HCFPs and 45 HCTPs

More data to be added

- **Tools**
- **STONESOUP**

Classifier development requires **True and False**

Successfully generated lots of data for classifiers

**The 35 CWEs**

- 457
- 680
- 252
- 843
- 483
- 195
- 404
- 369
- 377
- 126
- 197
- 415
- 606
- 398
- 835
- 134
- 665
- 122
- 196
- 758
- 191
- 121
- 468
- 194
- 761
- 681
- 469
- 190
- 127
- 476
- 688
- 401
- 563
- 775
- 587

The 35 CWEs successfully generated lots of data for classifiers.
Classifiers: Accuracy, #Alerts, AUROC

<table>
<thead>
<tr>
<th>Rule</th>
<th>Accuracy</th>
<th># Alerts</th>
<th>AUROC</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARR30-C</td>
<td>96.9%</td>
<td>483</td>
<td>99.8%</td>
</tr>
<tr>
<td>ARR32-C</td>
<td>100.0%</td>
<td>947</td>
<td>100.0%</td>
</tr>
<tr>
<td>ARR36-C</td>
<td>63.3%</td>
<td>30</td>
<td>50.0%</td>
</tr>
<tr>
<td>ARR37-C</td>
<td>74.0%</td>
<td>77</td>
<td>83.6%</td>
</tr>
<tr>
<td>ARR38-C</td>
<td>94.0%</td>
<td>397</td>
<td>98.0%</td>
</tr>
<tr>
<td>ARR39-C</td>
<td>67.7%</td>
<td>31</td>
<td>50.0%</td>
</tr>
<tr>
<td>CON33-C</td>
<td>100.0%</td>
<td>88</td>
<td>100.0%</td>
</tr>
<tr>
<td>ERR33-C</td>
<td>91.2%</td>
<td>376</td>
<td>94.9%</td>
</tr>
<tr>
<td>ERR34-C</td>
<td>100.0%</td>
<td>947</td>
<td>100.0%</td>
</tr>
<tr>
<td>EXP30-C</td>
<td>100.0%</td>
<td>947</td>
<td>100.0%</td>
</tr>
<tr>
<td>EXP33-C</td>
<td>89.5%</td>
<td>5214</td>
<td>96.3%</td>
</tr>
<tr>
<td>EXP34-C</td>
<td>91.8%</td>
<td>546</td>
<td>95.4%</td>
</tr>
<tr>
<td>EXP39-C</td>
<td>70.7%</td>
<td>116</td>
<td>83.1%</td>
</tr>
<tr>
<td>EXP46-C</td>
<td>82.5%</td>
<td>143</td>
<td>87.8%</td>
</tr>
<tr>
<td>FIO30-C</td>
<td>86.5%</td>
<td>1065</td>
<td>95.1%</td>
</tr>
<tr>
<td>FIO34-C</td>
<td>72.5%</td>
<td>1132</td>
<td>78.5%</td>
</tr>
<tr>
<td>FIO42-C</td>
<td>83.9%</td>
<td>933</td>
<td>93.2%</td>
</tr>
<tr>
<td>FIO46-C</td>
<td>100.0%</td>
<td>947</td>
<td>100.0%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Rule</th>
<th>Accuracy</th>
<th># Alerts</th>
<th>AUROC</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIO47-C</td>
<td>86.4%</td>
<td>1070</td>
<td>95.4%</td>
</tr>
<tr>
<td>FLP32-C</td>
<td>100.0%</td>
<td>947</td>
<td>100.0%</td>
</tr>
<tr>
<td>FLP34-C</td>
<td>70.5%</td>
<td>3619</td>
<td>78.0%</td>
</tr>
<tr>
<td>INT30-C</td>
<td>63.7%</td>
<td>1365</td>
<td>66.4%</td>
</tr>
<tr>
<td>INT31-C</td>
<td>68.7%</td>
<td>5139</td>
<td>77.5%</td>
</tr>
<tr>
<td>INT32-C</td>
<td>69.9%</td>
<td>1599</td>
<td>75.7%</td>
</tr>
<tr>
<td>INT33-C</td>
<td>79.8%</td>
<td>228</td>
<td>86.3%</td>
</tr>
<tr>
<td>INT34-C</td>
<td>100.0%</td>
<td>947</td>
<td>100.0%</td>
</tr>
<tr>
<td>INT35-C</td>
<td>64.3%</td>
<td>622</td>
<td>72.2%</td>
</tr>
<tr>
<td>INT36-C</td>
<td>100.0%</td>
<td>967</td>
<td>100.0%</td>
</tr>
<tr>
<td>MEM30-C</td>
<td>94.5%</td>
<td>1461</td>
<td>99.3%</td>
</tr>
<tr>
<td>MEM31-C</td>
<td>83.9%</td>
<td>933</td>
<td>93.2%</td>
</tr>
<tr>
<td>MEM35-C</td>
<td>66.7%</td>
<td>2514</td>
<td>76.0%</td>
</tr>
<tr>
<td>MSC37-C</td>
<td>100.0%</td>
<td>947</td>
<td>100.0%</td>
</tr>
<tr>
<td>POS54-C</td>
<td>90.0%</td>
<td>239</td>
<td>94.5%</td>
</tr>
<tr>
<td>PRE31-C</td>
<td>97.8%</td>
<td>46</td>
<td>99.1%</td>
</tr>
<tr>
<td>STR31-C</td>
<td>94.0%</td>
<td>397</td>
<td>98.0%</td>
</tr>
<tr>
<td>WIN30-C</td>
<td>95.6%</td>
<td>1465</td>
<td>97.8%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Model</th>
<th>Accuracy</th>
<th>AUROC</th>
</tr>
</thead>
<tbody>
<tr>
<td>lighgbm</td>
<td>83.7%</td>
<td>93.8%</td>
</tr>
<tr>
<td>xgboost</td>
<td>82.4%</td>
<td>92.5%</td>
</tr>
<tr>
<td>rf</td>
<td>78.6%</td>
<td>86.3%</td>
</tr>
<tr>
<td>lasso</td>
<td>82.5%</td>
<td>92.5%</td>
</tr>
</tbody>
</table>

**Improvement:** 67 per-rule classifiers (and more coming) vs. only 3 in FY16

**All-data CWE classifiers**

**Lasso per-CERT-rule classifiers (36)**

<table>
<thead>
<tr>
<th>Rule</th>
<th>Accuracy</th>
<th># Alerts</th>
<th>AUROC</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARR30-C</td>
<td>96.9%</td>
<td>483</td>
<td>99.8%</td>
</tr>
<tr>
<td>ARR32-C</td>
<td>100.0%</td>
<td>947</td>
<td>100.0%</td>
</tr>
<tr>
<td>ARR36-C</td>
<td>63.3%</td>
<td>30</td>
<td>50.0%</td>
</tr>
<tr>
<td>ARR37-C</td>
<td>74.0%</td>
<td>77</td>
<td>83.6%</td>
</tr>
<tr>
<td>ARR38-C</td>
<td>94.0%</td>
<td>397</td>
<td>98.0%</td>
</tr>
<tr>
<td>ARR39-C</td>
<td>67.7%</td>
<td>31</td>
<td>50.0%</td>
</tr>
<tr>
<td>CON33-C</td>
<td>100.0%</td>
<td>88</td>
<td>100.0%</td>
</tr>
<tr>
<td>ERR33-C</td>
<td>91.2%</td>
<td>376</td>
<td>94.9%</td>
</tr>
<tr>
<td>ERR34-C</td>
<td>100.0%</td>
<td>947</td>
<td>100.0%</td>
</tr>
<tr>
<td>EXP30-C</td>
<td>100.0%</td>
<td>947</td>
<td>100.0%</td>
</tr>
<tr>
<td>EXP33-C</td>
<td>89.5%</td>
<td>5214</td>
<td>96.3%</td>
</tr>
<tr>
<td>EXP34-C</td>
<td>91.8%</td>
<td>546</td>
<td>95.4%</td>
</tr>
<tr>
<td>EXP39-C</td>
<td>70.7%</td>
<td>116</td>
<td>83.1%</td>
</tr>
<tr>
<td>EXP46-C</td>
<td>82.5%</td>
<td>143</td>
<td>87.8%</td>
</tr>
<tr>
<td>FIO30-C</td>
<td>86.5%</td>
<td>1065</td>
<td>95.1%</td>
</tr>
<tr>
<td>FIO34-C</td>
<td>72.5%</td>
<td>1132</td>
<td>78.5%</td>
</tr>
<tr>
<td>FIO42-C</td>
<td>83.9%</td>
<td>933</td>
<td>93.2%</td>
</tr>
<tr>
<td>FIO46-C</td>
<td>100.0%</td>
<td>947</td>
<td>100.0%</td>
</tr>
</tbody>
</table>

**Lasso per-CWE-ID classifiers (31)**

<table>
<thead>
<tr>
<th>Rule</th>
<th>Accuracy</th>
<th># Alerts</th>
<th>AUROC</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARR30-C</td>
<td>96.9%</td>
<td>483</td>
<td>99.8%</td>
</tr>
<tr>
<td>ARR32-C</td>
<td>100.0%</td>
<td>947</td>
<td>100.0%</td>
</tr>
<tr>
<td>ARR36-C</td>
<td>63.3%</td>
<td>30</td>
<td>50.0%</td>
</tr>
<tr>
<td>ARR37-C</td>
<td>74.0%</td>
<td>77</td>
<td>83.6%</td>
</tr>
<tr>
<td>ARR38-C</td>
<td>94.0%</td>
<td>397</td>
<td>98.0%</td>
</tr>
<tr>
<td>ARR39-C</td>
<td>67.7%</td>
<td>31</td>
<td>50.0%</td>
</tr>
<tr>
<td>CON33-C</td>
<td>100.0%</td>
<td>88</td>
<td>100.0%</td>
</tr>
<tr>
<td>ERR33-C</td>
<td>91.2%</td>
<td>376</td>
<td>94.9%</td>
</tr>
<tr>
<td>ERR34-C</td>
<td>100.0%</td>
<td>947</td>
<td>100.0%</td>
</tr>
<tr>
<td>EXP30-C</td>
<td>100.0%</td>
<td>947</td>
<td>100.0%</td>
</tr>
<tr>
<td>EXP33-C</td>
<td>89.5%</td>
<td>5214</td>
<td>96.3%</td>
</tr>
<tr>
<td>EXP34-C</td>
<td>91.8%</td>
<td>546</td>
<td>95.4%</td>
</tr>
<tr>
<td>EXP39-C</td>
<td>70.7%</td>
<td>116</td>
<td>83.1%</td>
</tr>
<tr>
<td>EXP46-C</td>
<td>82.5%</td>
<td>143</td>
<td>87.8%</td>
</tr>
<tr>
<td>FIO30-C</td>
<td>86.5%</td>
<td>1065</td>
<td>95.1%</td>
</tr>
<tr>
<td>FIO34-C</td>
<td>72.5%</td>
<td>1132</td>
<td>78.5%</td>
</tr>
<tr>
<td>FIO42-C</td>
<td>83.9%</td>
<td>933</td>
<td>93.2%</td>
</tr>
<tr>
<td>FIO46-C</td>
<td>100.0%</td>
<td>947</td>
<td>100.0%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Model</th>
<th>Accuracy</th>
<th>AUROC</th>
</tr>
</thead>
<tbody>
<tr>
<td>lighgbm</td>
<td>83.7%</td>
<td>93.8%</td>
</tr>
<tr>
<td>xgboost</td>
<td>82.4%</td>
<td>92.5%</td>
</tr>
<tr>
<td>rf</td>
<td>78.6%</td>
<td>86.3%</td>
</tr>
<tr>
<td>lasso</td>
<td>82.5%</td>
<td>92.5%</td>
</tr>
</tbody>
</table>

**Similar for other classifier methods**

<table>
<thead>
<tr>
<th>Avg. accuracy</th>
<th>Count accuracy 95+%</th>
<th>Count accuracy 85-94.9%</th>
<th>Count accuracy 0-84.9%</th>
</tr>
</thead>
<tbody>
<tr>
<td>85.8%</td>
<td>12</td>
<td>9</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td>99.2%</td>
<td>90.9%</td>
<td>72.1%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Avg. accuracy</th>
<th>Count accuracy 95+%</th>
<th>Count accuracy 85-94.9%</th>
<th>Count accuracy 0-84.9%</th>
</tr>
</thead>
<tbody>
<tr>
<td>81.8%</td>
<td>7</td>
<td>10</td>
<td>14</td>
</tr>
<tr>
<td></td>
<td>98.4%</td>
<td>89.6%</td>
<td>67.9%</td>
</tr>
</tbody>
</table>

**Similar for other classifier methods**
Summary and Future

FY17 Line “Rapid Classifiers” built on the FY16 LENS “Prioritizing vulnerabilities”.
- Developed widely useful general method to use test suites across taxonomies
- Developed large archive of “pre-audited” alerts
  - Overcame challenge to classifier development
  - For CWEs and CERT rules
- Developed code infrastructure (extensible)
- In-progress:
  - Classifier development and testing in process
  - Continue to gather data
  - Enhanced SCALe audit tool for collaborator testing: distribute to collaborators soon
- FY18-19 plan: architecture for rapid deployment of classifiers in varied systems
- Goal: improve automation of static alert auditing (and other code analysis and repair)

Publications:
- New mappings (CWE/CERT rule): MITRE and CERT websites
- SEI blogposts on classifier development
- Research papers (SQUADE’18), others in progress
Ideas for collaboration welcome

Collaborative work topics might include:

• Continuous integration:
  - Optimizing alert analysis of developing project over time
  - Modifications to previously-developed techniques
• Enhancements to algorithms/architecture, to enable more widespread use
• ??
Contact Information

Presenter / Point(s) of Contact
Lori Flynn (Principal Investigator)
Software Security Researcher

Email: lflynn@cert.org
Telephone: +1 412.268.7886

Contributors

SEI Staff
William Snavely
David Svoboda
Zach Kurtz

SEI Student Interns
Lucas Bengtson (CMU)
Charisse Haruta (CMU)
Baptiste Vauthey (CMU)
Michael Spece (Pitt)
Christine Baek (CMU)