Tools for Vulnerability Analysis and Automated Code Repair

Robert Schiela

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

Document Markings

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.

DM23-0051

Context

Analysis Types

- Source Code Analysis – mostly static
- Program Analysis – static and dynamic

Process

- Identify
- Analyze
- Mitigate
- Measure (findings, risk)

Recent Work: Automating the Processes…

Progress

Our foundational work:

- Secure Coding Security Standards: language specific guides
- Source Code Analysis Lab (SCALe) Static Code Analysis

Recent Developments:

- Combine with broader Architectural profile: Code Risk Estimation Worksheet (CREW)

Recent Research:

- Automatically improve false-positive rates (Type I errors) of static analysis findings
- Automatically repair code
  - High confidence it is a problem
  - High confidence the change will not introduce a negative impact to the program
  - Ideally: high confidence we fix the problem
- Detecting malicious injects in code

Secure Coding Research

Rapid Adjudication of Static Analysis Alerts During Continuous Integration

- Automated predictions to improve prioritization of source code analysis using machine learning and information from multiple analysis tools over multiple iterations.

Automated Code Repair (ACR)

- Fixing code based on anti-patterns and patterns for repair, rather than just alerting developers and testers to a potential defect.
- Applying source code analysis techniques to binary code for analysis and repair

Semantic Equivalence Checker for Binary Static Analysis

- Evaluating the effectiveness of using source static analysis tools on decompiled binary.

Automated Repair of Static Analysis Alerts (FY23-24)

- Automatically change code (or propose code changes) to resolve simple findings from Static Analysis tools, even if false positives. (reduce manual analysis when no cost)

Detecting Inserted Malicious Code with Information Flows (FY23-24)

- Develop tool to detect exfiltration of sensitive data and sensitive operations driven by external input

Additional Research

Automated Support (ML) for Program Understanding
- Reverse Engineering support
- Defect Identification
- ChatGPT…

Binary Analysis
- Pharos
- Ghidra

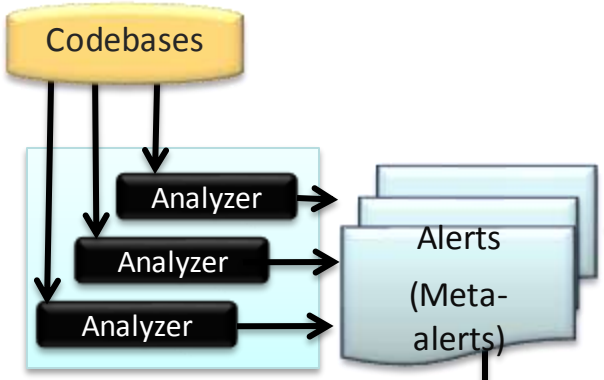
Lower System Level Code Analysis (Firmware, UEFI…)

Software Understanding for National Security (SUNS 2023) Workshop

Backup

Prioritizing Alerts

Problem: too many (meta-)alerts
Solution: automate handling



Today

Project Goal

Classification algorithm development for CI systems, that **precisely and with high recall, classifies at least as many manually-adjudicated meta-alerts as:**

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

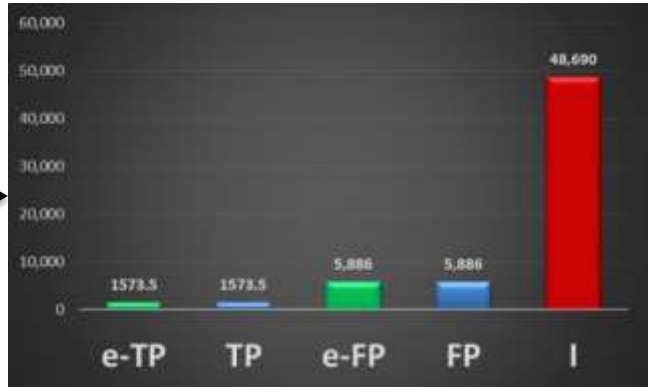


Image of woman and laptop from http://www.publicdomainpictures.net/view-image.php?image=47526&picture=woman-and-laptop "Woman And Laptop"

Prioritizing Alerts

Research progression:

- FY16 – Prioritizing Vulnerabilities from Static Analysis with Classification Models: Proof of concept of Machine Learning for Prioritizing Static Analysis Results
- FY17 – Rapid Expansion of Classification Models to Prioritize Static Analysis Alerts: Adding open source test suite code for data
- FY18-19 – Rapid Construction of Accurate Automatic Alert Handling System: Reference Model & Reference Prototype: Source Code Analysis Integrated Framework Environment (SCAIFE)
- FY20-21 - Rapid Adjudication of Static Analysis Alerts During Continuous Integration: Support for Continuous Integration (automated triggers & iterative data)

Automated Code Repair (ACR) tool as a black box

Input: Buildable codebase

Output: Repaired source code, suitable for committing to repository

We support C and plan to have limited support for C++

Envisioned use of tool

- Use before every release build
- Use occasionally for debugging builds
- Intended for ordinary developers
- Can be a tool in the DevOps toolchain
- Can be used for legacy code and for new code

Automated Code Repair

Hypothesis: Many violations of rules follow a small number of anti-patterns with corresponding patterns for repair, and these can be feasibly recognized by static analysis.

- `printf(attacker_string)` → `printf("%s", attacker_string)`

Research progression:

- FY16 Integer Overflow: (start + i) → UADD (start, i)
- FY17 Inference of Memory Bounds (out of bounds reads like HeartBleed): abort or warning
- F18-20 - Memory Safety: Fat Pointers
- FY21 - Combined Analysis for Source Code and Binaries for Software Assurance: Semantic Equivalence Checking of Decompiled Code (LLVM Focus)
- FY22 - Decompilation for Software Assurance: Analysis and Repair of Correctly Decompiled Functions (Ghidra Focus)

Source Code Repair Pipeline