Three Software Innovations that DoD Needs Now

Jeff Boleng, Sam Procter, Nathan VanHoudnos, Lena Pons, Robert Schiela
Three Software Innovation that DoD Needs Now

© 2018 Carnegie Mellon University

[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 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.

DM18-0334
Virtual Integration: AADL as a “Single Source of Truth”

Model-Based Engineering for System Design
Build models of components, combine them into a unified system, and verify that everything fits before the system is built: Integrate-then-Build

Detect Issues Early, Save Money and Effort
Studies show most system defects are created early – in the design and architecture stages – but fixed late. Defects caught later are more costly in terms of both budget and schedule overruns.

Multiple Analyses, One Model
A single AADL model can support multiple analyses. Custom analyses are easy to implement, or use outputs of preferred tools to annotate AADL model.
AADL Success Stories

Wheel Braking System
- Example used in SAE standardization efforts (ARP 4761 & AIR61160)
- AADL source publically available on github
- Used in ongoing safety research

System Architecture Virtual Integration
- “Incremental Validation, Continuous Integration”
- Pays for itself in commercial development

HACMS: Strong Security
- Secure drone and helicopter developed using AADL, seL4 & other tech
- Resistant to weeks of red-team attacks, even with source code

Image adapted from loonwerks.com
Guided Automated Tradespace Exploration

- Prototype connects AADL tooling to visual exploration software
- Any components that can be specified in AADL can be swapped in and analyzed
- Easily extended to include domain-specific analyses
Machine Learning for the DoD: Malware

Many suspect files. Manual pairwise analysis is expensive.
Machine Learning for the DoD: Malware

Many suspect files.

Statistical visualization lowers costs.
Automated Analysis - Prioritizing Vulnerabilities

Codebases

Analyzer

Analyzer

Analyzer

Alerts

Long-term goal: Automated and accurate statistical classifier, intended to efficiently use analyst effort and to remove code flaws.

Today

Project Goal

Classification algorithm development using CERT- and collaborator-audited data, that accurately classifies most of the diagnostics as:

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

Many alerts left unaudited!

Prioritized, small number of alerts for manual audit

Classification algorithm development using CERT- and collaborator-audited data, that accurately classifies most of the diagnostics as:

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

Prioritized, small number of alerts for manual audit

Classification algorithm development using CERT- and collaborator-audited data, that accurately classifies most of the diagnostics as:

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

Prioritized, small number of alerts for manual audit

Classification algorithm development using CERT- and collaborator-audited data, that accurately classifies most of the diagnostics as:

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

Prioritized, small number of alerts for manual audit

Classification algorithm development using CERT- and collaborator-audited data, that accurately classifies most of the diagnostics as:

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

Prioritized, small number of alerts for manual audit

Classification algorithm development using CERT- and collaborator-audited data, that accurately classifies most of the diagnostics as:

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

Prioritized, small number of alerts for manual audit

Classification algorithm development using CERT- and collaborator-audited data, that accurately classifies most of the diagnostics as:

- Expected True Positive (e-TP)
- Expected False Positive (e-FP)
- and the rest as Indeterminate (I)
Automated Code Repair

Many violations of rules follow a small number of anti-patterns with corresponding patterns for repair

These can be feasibly recognized by static analysis

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

Creating tools to automatically repair these types of defects in source code

• Integer Overflows that lead to memory corruption
• Inferred memory bounds for reading from reused buffers
• Verified memory safety

Constraints

• The patched and unpatched program behave identically over the set of all traces that conform to the rules. (formally proven)
• No trace violates the rules. (formally proven)
• Repair in way that is plausibly acceptable to the developer.
Cyber incident tickets are comprised of semi-structured data containing indicators

Traditional indicators like IP address, filename, file hash, email address can be augmented with concepts & relations
Presenters

Robert Schiela
rschiela@cert.org

Sam Procter
sprocter@sei.cmu.edu

Lena Pons
lepons@cert.org

Nathan VanHoudnos
nmvanhoudnos@cert.org

Jeff Boleng
jlboleng@sei.cmu.edu