Using Machine Learning to Improve Security Analysis of Source Code

Mark Sherman
Sept 23, 2020
Cylab Partners Conference

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.

DM20-0770

Research Projects at SEI/ CERT

Source Code as Natural Language

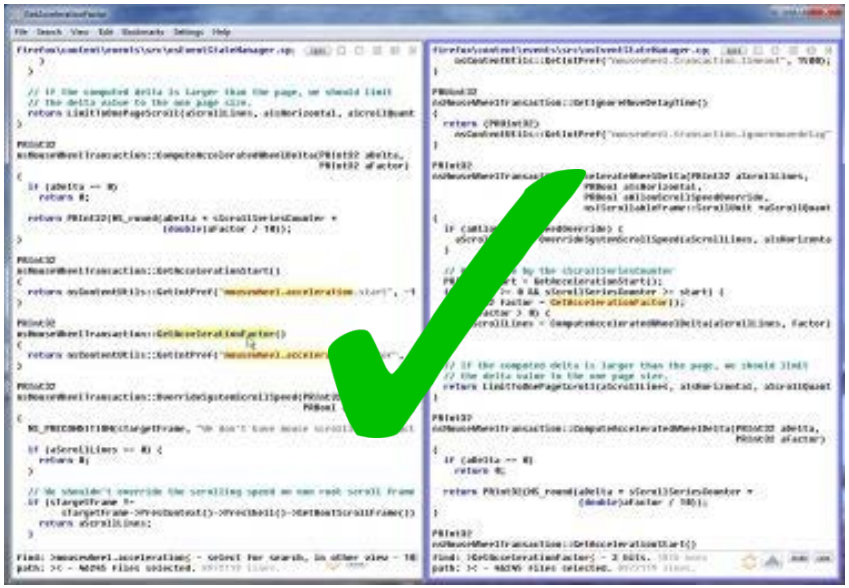
Combining Source Code Analyzers

Automated Source Code Repair

Research Teams

PIs	Team members	Team members (cont)
Lori Flynn	Jennifer Burns	Ebonie McNeil
Will Klieber	Matt Churilla	Richard Qin
Carson Sestili	Zachary Kurtz	Will Snavely
	Jiyeon Lee	Ryan Steele
	Derek Leung	Robert Stoddard
	Guillermo Marce-Santurio	David Svoboda
	Ruben Martins (SCS)	Nathan VanHoudnos
	Mike McCall	Joseph Yankel
		David Zubrow

Finding Code Defects – ML that Considers Source Code as Natural Language

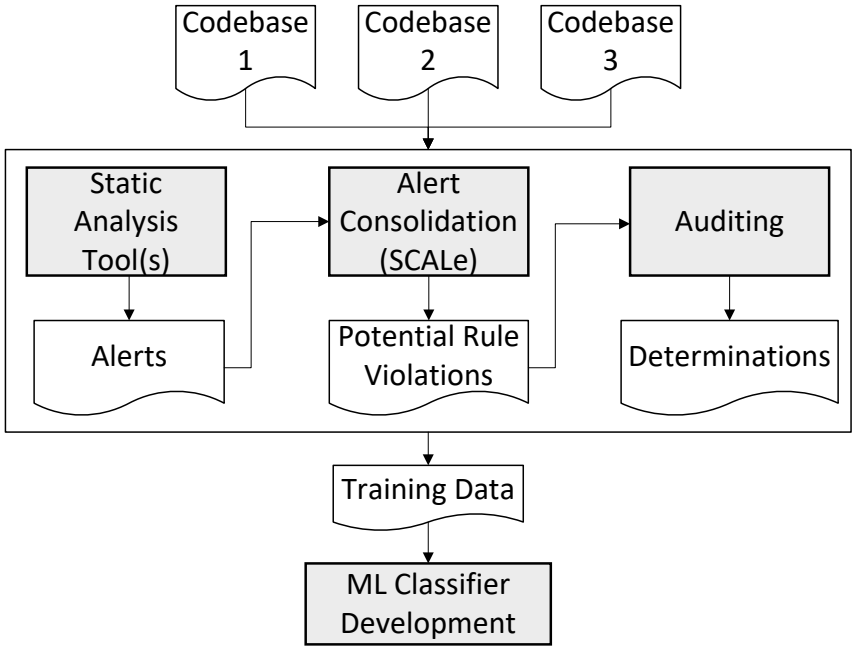


Analyze Source Code for Insecure Coding

- 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, read the article online here.

Finding Code Defects Using Machine Learning



Read the L. Flynn publications at SEI Digital Library here.

Introduction and Integration of Machine Learning

- Combine multiple tool outputs
- Assist analyst with combining outputs and tracking decisions

- Limited tools
- Limited error types
- Limited training data
- Limited features

- Common lexicon and labeling process
- Synthetic generation with test suites
- Increased feature space

- Alternative tools for generating measurements
- Alternative feature extraction
- Alternative ML algorithms, e.g., active learning

- CI/CD data sources
- Rapid feedback

Automated Code Repair (ACR) Tool as a Black Box

Input: Buildable codebase

Output: Repaired source code that is still human-readable and maintainable.

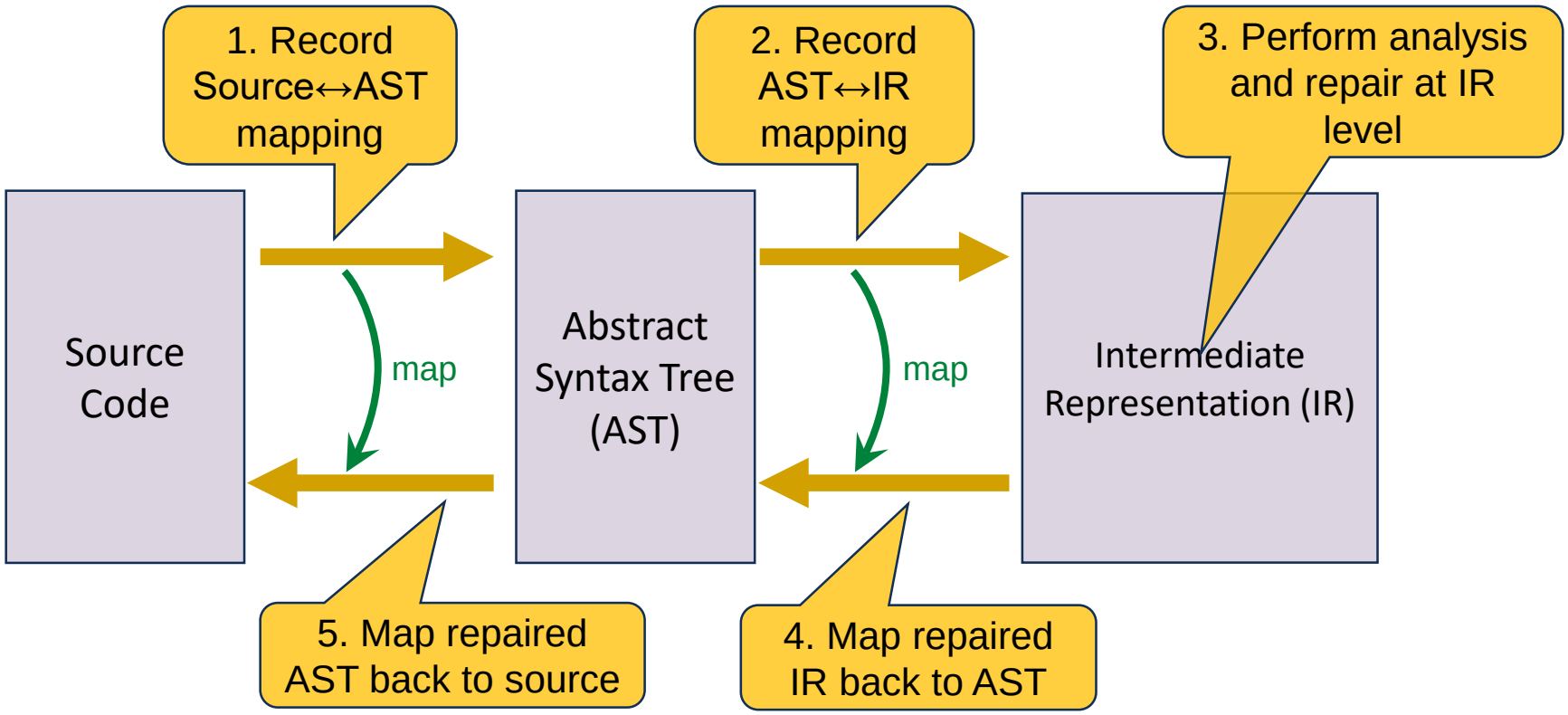
We currently support C code. Support for C++ can likely be added without too much difficulty.

Read Will Klieber, Automated Code Repair to Ensure Memory Safety, Feb 24, 2020, here.

Why repair of source code instead of as a compiler pass?

Repair of source code	Repair as a compiler pass
Easily audited (if desired).	Must trust the tool.
Repairs can easily be tweaked to improve performance, if necessary.	Difficult to remediate performance issues caused by repair.
Changes to source code are frequent and easily handled.	Changes to the build process may be more difficult and error-prone.
Okay to do slow, heavy-weight static analysis; produces a persistent artifact.	Slowing down every test build is not okay.

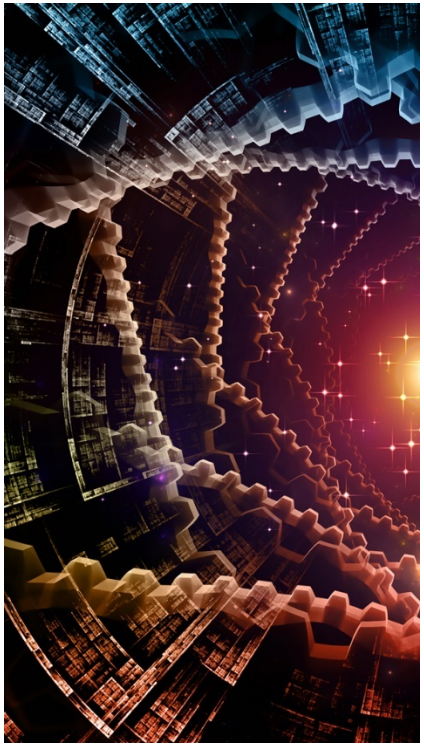
Source Code Repair Pipeline



Challenges

- In translating repairs from AST to source, the C preprocessor is main difficulty.
  - Repairs to macro uses
  - Repairs to `#included` code
  - Conditional-compilation directives (`#ifdef`, `#endif`, etc.) inside expressions
- Considerations of whitespace
- The C preprocessor can conditionally include or exclude pieces of code depending on the configuration chosen at compile time.
  - We repair configurations separately and then merge the results such that the final repaired code is correct under all desired configurations.

Ways to Engage with Us



- Download software and tools
- Explore research and capabilities
- Participate in education offerings
- Attend an event
- Search the digital library
- Read the SEI Year in Review
- Collaborate with the SEI on a new project

Software Engineering Institute
Carnegie Mellon University
4500 Fifth Avenue
Pittsburgh, PA 15213-3890
412-268-5800 - Phone
888-201-4479 - Toll-Free
412-268-5758 - Fax
info@sei.cmu.edu - Email
www.sei.cmu.edu - Web