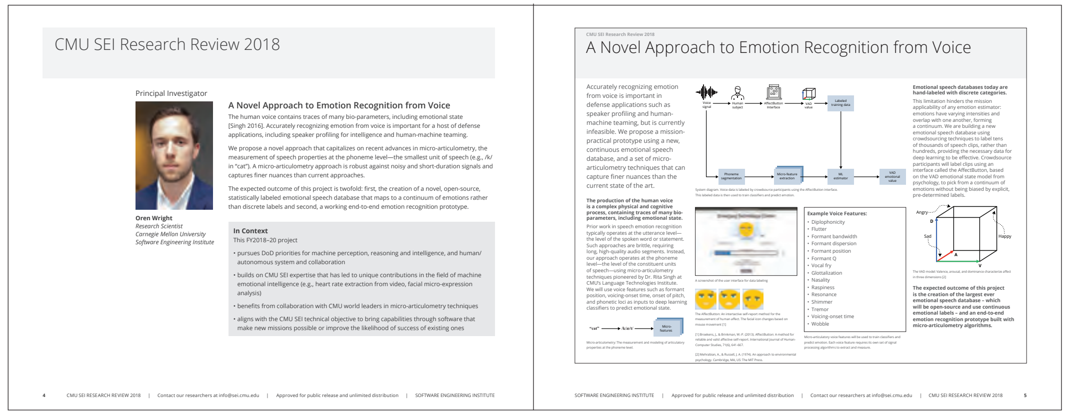
Carnegie Mellon University
Software Engineering Institute

CMU SEI Research Review 2018

OCTOBER 9–10, 2018

Project Summaries and Posters



This booklet contains descriptions of CMU SEI research projects and images of posters related to the research. In each of the sections, you will find a project description on a left-hand (even-numbered) page and a poster image facing it on a (odd-numbered) right-hand page.

CMU SEI Research Review 2018

Roman Danyliw
CTO (Acting)
Carnegie Mellon University
Software Engineering Institute

The ability of the U.S. Department of Defense (DoD) to produce, evolve, and protect software-enabled systems is central to its ability to maintain superiority across domains.

The Carnegie Mellon University Software Engineering Institute (CMU SEI) is the DoD-sponsored federally funded research and development center (FFRDC) for software and cybersecurity. By placing the SEI at CMU, a world-leading computer-science research institution, DoD gains not only this FFRDC's expertise and capacity but also access to cutting-edge basic research through the continual collaboration between CMU SEI technical staff and CMU faculty and students.

In addition, CMU SEI works with other federal agencies, the Intelligence Community, and industry for the ultimate benefit of the DoD. These engagements provide the DoD leverage by cost-sharing solutions to pervasive problems and access to a wider pipeline of adoptable innovative solutions from outside government.

In all its technical work—whether in applied research and development, sponsored engagements, or technology transfer activities—SEI's goal is to enable the DoD to realize advantage through software. CMU SEI work addresses the four enduring challenges for DoD's software-enabled systems:

1. Bring Capabilities that make new missions possible or improve the likelihood of success of existing ones
2. Be Timely so that the cadence of acquisition, delivery, and fielding is responsive to and anticipatory of the operational tempo of DoD warfighters and that the DoD is able to field these new software-enabled systems and their upgrades faster than our adversaries
3. Be Trustworthy in construction and implementation and resilient in the face of operational uncertainties including known and yet unseen adversary capabilities
4. Be Affordable such that the cost of acquisition and operations, despite increased capability, is reduced, predictable and provides a cost advantage over our adversaries

In this booklet, you can read about applied research sponsored by the Under Secretary of Defense, Research and Engineering, that CMU SEI initiated, continued, or concluded in FY2018. I invite you to reach out to our researchers for more information about their ongoing work or to discuss your current and anticipated needs.

Contents

Project Summaries and Posters

A Novel Approach to Emotion Recognition from Voice 4
Advancing Assistance Capabilities for Program Analysts 6
An Integrated Causal Model for Software Cost Prediction & Control (SCOPE) 8
Automated Code Generation for Future-Compatible High-Performance Graph Libraries 10
Automated Code Repair to Ensure Memory Safety 12
Building a COTS Benchmark Baseline for Graph Analytics 14
Certifiable Distributed Runtime Assurance (CDRA) 16
High Assurance Software-Defined IoT Security 18
Infrastructure as Code 20
Integrated Safety and Security Engineering for Mission-Critical Systems (ISSE-MCS) 22
Modeling and Explaining Sequential Behavior 24
Modeling the Operations of the Vulnerability Ecosystem 26
Predicting Security Flaws through Architectural Flaws 28
Rapid Construction of Accurate Automatic Alert Handling 30
Rapid Software Composition by Assessing Untrusted Components 32
Summarizing and Searching Video 34
Technical Debt Analysis through Software Analytics 36
Timing Verification of Undocumented Multicore 38
Towards Security Defect Prediction with AI 40

Project Summaries and Posters

Principal Investigator

Oren Wright
Research Scientist
Carnegie Mellon University
Software Engineering Institute

A Novel Approach to Emotion Recognition from Voice

The human voice contains traces of many bio-parameters, including emotional state [Singh 2016]. Accurately recognizing emotion from voice is important for a host of defense applications, including speaker profiling for intelligence and human-machine teaming.

We propose a novel approach that capitalizes on recent advances in micro-articulometry, the measurement of speech properties at the phoneme level—the smallest unit of speech (e.g., /k/ in “cat”). A micro-articulometry approach is robust against noisy and short-duration signals and captures finer nuances than current approaches.

The expected outcome of this project is twofold: first, the creation of a novel, open-source, statistically labeled emotional speech database that maps to a continuum of emotions rather than discrete labels and second, a working end-to-end emotion recognition prototype.

In Context
This FY2018–20 project

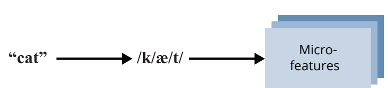
- pursues DoD priorities for machine perception, reasoning and intelligence, and human/autonomous system and collaboration
- builds on CMU SEI expertise that has led to unique contributions in the field of machine emotional intelligence (e.g., heart rate extraction from video, facial micro-expression analysis)
- benefits from collaboration with CMU world leaders in micro-articulometry techniques
- aligns with the CMU SEI technical objective to bring capabilities through software that make new missions possible or improve the likelihood of success of existing ones

A Novel Approach to Emotion Recognition from Voice

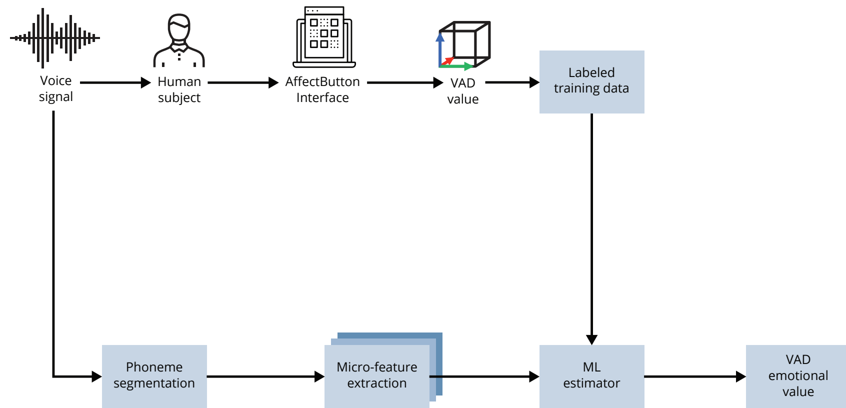
Accurately recognizing emotion from voice is important in defense applications such as speaker profiling and human-machine teaming, but is currently infeasible. We propose a mission-practical prototype using a new, continuous emotional speech database, and a set of micro-articulometry techniques that can capture finer nuances than the current state of the art.

The production of the human voice is a complex physical and cognitive process, containing traces of many bio-parameters, including emotional state.

Prior work in speech emotion recognition typically operates at the utterance level—the level of the spoken word or statement. Such approaches are brittle, requiring long, high-quality audio segments. Instead, our approach operates at the phoneme level—the level of the constituent units of speech—using micro-articulometry techniques pioneered by Dr. Rita Singh at CMU's Language Technologies Institute. We will use voice features such as formant position, voicing-onset time, onset of pitch, and phonetic loci as inputs to deep learning classifiers to predict emotional state.



Micro-articulometry: The measurement and modeling of articulatory properties at the phoneme level.



System diagram. Voice data is labeled by crowdsource participants using the AffectButton interface. This labeled data is then used to train classifiers and predict emotion.

A screenshot of the user interface for data labeling

The AffectButton: An intertactive self-report method for the measurement of human affect. The facial icon changes based on mouse movement [1]

[1] Broekens, J., & Brinkman, W.-P. (2013). AffectButton: A method for reliable and valid affective self-report. International Journal of Human-Computer Studies, 71(6), 641-667.

[2] Mehrabian, A., & Russell, J. A. (1974). An approach to environmental psychology. Cambridge, MA, US: The MIT Press.

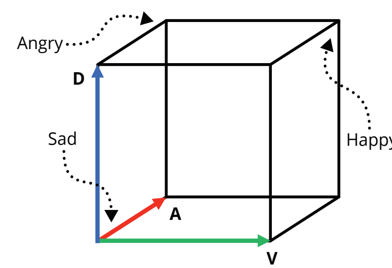
Example Voice Features:

- Diplophonicity
- Flutter
- Formant bandwidth
- Formant dispersion
- Formant position
- Formant Q
- Vocal fry
- Glottalization
- Nasality
- Raspiness
- Resonance
- Shimmer
- Tremor
- Voicing-onset time
- Wobble

Micro-articulatory voice features will be used to train classifiers and predict emotion. Each voice feature requires its own set of signal processing algorithms to extract and measure.

Emotional speech databases today are hand-labeled with discrete categories.

This limitation hinders the mission applicability of any emotion estimator: emotions have varying intensities and overlap with one another, forming a continuum. We are building a new emotional speech database using crowdsourcing techniques to label tens of thousands of speech clips, rather than hundreds, providing the necessary data for deep learning to be effective. Crowdsource participants will label clips using an interface called the AffectButton, based on the VAD emotional state model from psychology, to pick from a continuum of emotions without being biased by explicit, pre-determined labels.



The VAD model: Valence, arousal, and dominance characterize affect in three dimensions [2]

The expected outcome of this project is the creation of the largest ever emotional speech database – which will be open-source and use continuous emotional labels – and an end-to-end emotion recognition prototype built with micro-articulometry algorithms.

Principal Investigator

Cory Cohen
Senior Member of the Technical Staff
Carnegie Mellon University Software Engineering Institute

Advancing Assistance Capabilities for Program Analysts

Highly skilled Department of Defense (DoD) malware and vulnerability analysts currently spend significant amounts of time manually coercing specific portions of executable code to run. Challenges such as determining the unknown input conditions required to trigger the desired behavior, eliminating non-determinism, and coping with missing dependencies complicate this effort.

In this project, we are developing capabilities within CMU SEI's Pharos binary code analysis framework to address the technical problems underlying important cyber challenges. Our proposed research will improve reverse engineers' ability to comprehend complex malware by enabling them to trigger the execution of a specific portion of a program in a debugger or sandbox. This ability will also enable a vulnerability analyst to discover vulnerabilities, which is largely dependent on the same capabilities.

In Context

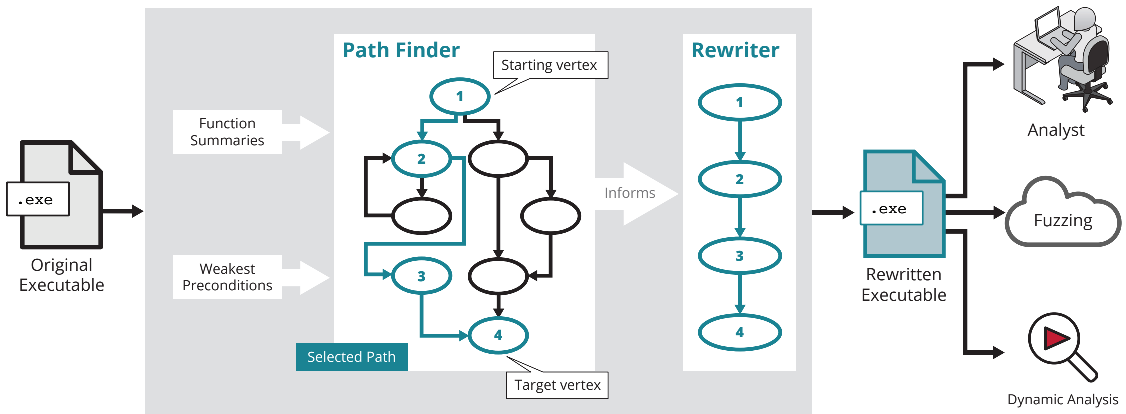
This FY2018–20 project

- extends DoD Line-funded research and tool development for vulnerability and binary code analysis
- contributes to development and transition of Pharos binary code analysis framework
- aligns with the CMU SEI technical objective to make software trustworthy in construction, correct in implementation, and resilient in the face of operational uncertainties including known and yet unseen adversary capabilities

Automatically Understanding Executables

Reducing the cost of manual executable analysis for vulnerability discovery and malware analysis

Automated Executable Path Finding and Rewriting Process



Understanding the conditions that cause an executable to follow specific paths helps DoD analysts identify vulnerabilities and understand malware behavior.

We aim to automate understanding the conditions required to cause an executable to reach a specific point in a control flow graph. This could test a specific feature in a piece of malware or establish whether it is possible to reach a vulnerable condition in software. This mitigates a tedious manual process.

There are potentially an infinite number of execution paths that we must search over. To cope with that complexity, we need multiple approaches.

To date we've implemented two path-finding algorithms with different performance and accuracy tradeoffs. We believe that combining the approaches will yield improved performance compared to either in isolation.

A binary rewriter can then create a new binary file that always follows the desired path when executed.

We're partnering with Dr. Arie Gurfinkel at the University of Waterloo to apply source code reachability techniques to executables.

Property directed reachability (PDR) has proven to be an effective technique for static analysis of source code reachability. Dr. Gurfinkel maintains a PDR implementation as part of Microsoft's open source SMT solver Z3.

This work is partially supported by the Office of Naval Research (ONR).

Automated variable name recovery through large-scale data mining and statistical analysis.

Reverse engineers often read decompiled code to understand the behavior of an executable. Modern decompilers do not attempt to recover meaningful variable names, and instead synthesize names such as *v12*. In this project, we use statistical techniques to learn appropriate variable names.

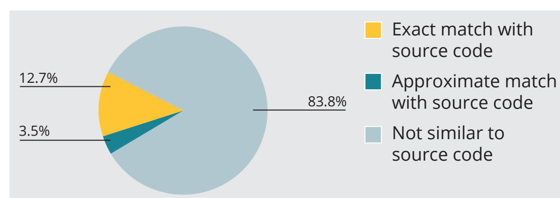
Decompiled C Code with Synthetic Names

```
v14 = &v15;
asxTab(a2 + 1);
for (v13 = asnContents(a1, &v15, 512LL); v13 > 0; --v13)
{
   v9 = (unsignedchar*)(v14++);
   printf(" %02X ", *v9);
}
```

Decompiled C Code with Recovered Names

```
cp = buf;
(void)asxTab(level + 1);
for (n = asnContents(asn, buf, 512); n > 0; n--)
{
   printf(" %02X ", *(cp++));
}
```

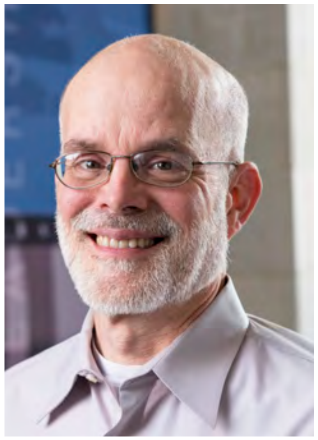
Current results. We evaluated our approach by comparing the variable names recovered by our system with the original variable names in the source code. When our system recovers exactly the same variable name, we call it an exact match. We also measure approximate matches, which occur when our system recovers an abbreviation that consists of at least half the characters of the original. For example, an approximate match would be recovering *buf* for the original variable name of *buffer*.



For more information about Pharos

https://github.com/sei-cmu/pharos

Principal Investigator



Dr. Michael Konrad
Principal Researcher
Carnegie Mellon University
Software Engineering Institute

An Integrated Causal Model for Software Cost Prediction & Control (SCOPE)

Until recently, we did not have a way to obtain or validate causal models from primarily observational data, a challenge shared across nearly all systems and software engineering research, where randomized control trials are nearly impossible. Yet, in search of good practice, systems and software engineering are rife with theories about how best to conduct system and software system development and sustainment. The SCOPE project will apply causal modeling algorithms and tools [Spirtes 2010] to a large volume of project data, so we can identify, measure, and test causality.

In this work, we expect to produce integrated, estimated, and causally based structural equation models (SEMs) that provide a basis for

- calculating the effort, schedule, and quality results of software projects under different scenarios (e.g., traditional vs. agile)
- estimating the results of interventions applied to a project in response to a change in requirements (e.g., a change in mission) or to help bring it back on track toward achieving cost, schedule, and technical requirements

Thus, an immediate benefit of this work is the identification of causal factors that provide a basis for controlling program costs. A longer-term benefit is the use of causal models in negotiating software contracts, designing policy and incentives, and informing could/should cost and affordability efforts.

In Context

This FY2018–20 project

- contributes to a longer term research road map to build causal models for the software developer, software development team, organization, and acquirer
- aligns with the CMU SEI technical objective to make software affordable such that the cost of acquisition and operations, despite increased capability, is reduced and predictable

Causal Models for Software Cost Control (SCOPE)

Recent results from five different studies

How can we better control costs in software development and sustainment? This project is collaborating with systems and software researchers in applying causal learning to program datasets to better understand which factors can reduce costs.

DoD Problem

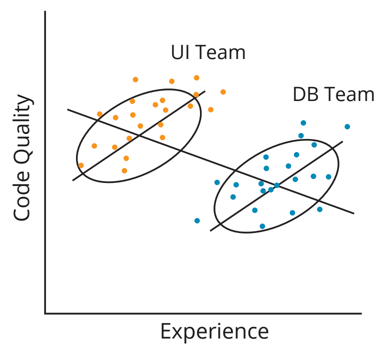
- DoD leadership continues to ask "Why does software cost so much?"
- DoD program offices need to know where to intervene to control software costs

Our Solution

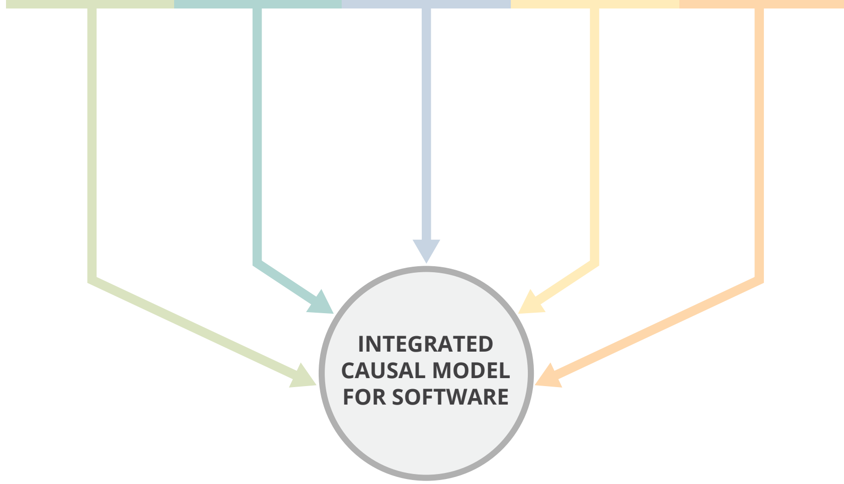
An actionable, full causal model of software cost factors immediately useful to DoD programs and contract negotiators

Causation Vs. Correlation

To reduce costs, what *causes* code quality to be good or bad needs to be understood. Correlations are insufficient. For example, in the figure below, would increasing experience level improve code quality?

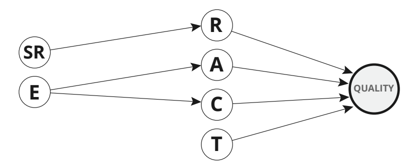


Practitioner	Software Size	Architecture	Complexity	Leadership
Challenge: Which factors affect a programmer's coding effort and quality?	**Challenge:** Which approaches to measuring code size most reflect factors affecting total effort?	**Challenge:** How might a project manager decide which areas of code to prioritize for maintenance?	**Challenge:** Which program and system complexity factors most affect cost, schedule, and performance?	**Challenge:** For action planning, which attributes of teaming and leadership improve team performance?
Approach: Apply Causal Discovery to data from students coding to the same ten requirements specifications.	**Approach:** Apply Causal Discovery to USC's Unified Code Count (UCC) project dataset.	**Approach:** Apply Causal Discovery to the results of a static code and design structure analysis to determine which type of architectural pattern violation most affects code quality.	**Approach:** Apply Causal Discovery to an existing project-survey dataset.	**Approach:** Apply Causal Discovery to results of 18 months of weekly surveys of software engineers from across a DoD organization to determine which factors most affect cost, schedule, and quality.
Results: To achieve precision, software estimation models should include both objective measures of requirements size as well as programmer-specific coding and defect factors.	**Results:** For IT-type systems, only COSMIC Function Points, Programmer Capability, and Documentation-Aligns-with-Lifecycle-Needs repeatedly recur as direct causes of total effort.	**Results:** Cyclic dependency was the single architecture pattern violation affecting code quality.	**Results:** The original analysis identified difficult requirements, stakeholder relationships, and cognitive fog; causal discovery confirmed only cognitive fog.	**Results:** Of the 20+ factors found to be highly correlated with cost, schedule, and quality, direct causal relationships were found for only two: Good Improvement Data and Stress From Overtime.



Technical Approach

Working with collaborators, we will identify and prepare datasets for causal learning to establish key cause-effect relationships among project factors and outcomes. For example, for Quality, we might have this causal graph:



The resulting causal models will then be "stitched" using CMU algorithms to create a universal causal model, but estimated and calibrated for lifecycle and super-domain. These estimated models will be the basis for improved program management.

Collaborative Approach

First, the SEI trains each collaborator to perform causal searches on their own proprietary datasets. The SEI then only needs to be provided with information about what dataset and search parameters were used as well as the resulting causal graph, which is sufficient for integrating into a universal causal model.

Summary

Causal learning has come of age from both a theoretical and tooling standpoint and provides better basis for program control than models based on correlation. Application to cost estimation requires large amounts of quality data. Now is the time to engage the larger community of systems and software researchers in deriving improved cost models that enable improved program control.

CMU SEI Research Review 2018

Principal Investigators

Dr. Scott McMillan
Senior Research Scientist
Carnegie Mellon University
Software Engineering Institute

Dr. Franz Franchetti
Professor, Department of Electrical and Computer Engineering, and Faculty Director of IT Services
Carnegie Mellon University

Automated Code Generation for Future-Compatible High-Performance Graph Libraries

Graph analytics and other data-intensive applications are characterized by unpredictable memory access and low computation intensity. Called irregular algorithms, these applications are difficult to implement efficiently, but they are required by increasingly complex systems.

In this work, we automated code generation of high-performance libraries of graph algorithms, tuned for different hardware architectures such as multi-core CPU and SIMD (Single Instruction, Multiple Data) GPU to establish a foundation for the infrastructure of a wider range of systems.

In addition, in support of our DoD stakeholders, we began collaboration with the University of Maryland—Baltimore County and EMU Technologies to port the GraphBLAS Template Library to a specialized computing platform developed by EMU Technologies.

In Context

This FY2017–18 project

- builds on prior DoD Line-funded research into the patterns and practices for future architectures and graph algorithms on future architectures
- aligns with the CMU SEI technical objective to make software trustworthy in construction, correct in implementation, and resilient in the face of operational uncertainties including known and yet unseen adversary capabilities
- is related to FY2018 Line-funded research in building a COTS benchmark baseline for graph analytics (see page 14)

Automatic Code Generation for Graph Algorithms

Research Problem

Turning mathematical graph algorithms into actual implementations that run at speed is complicated. It requires:

1. **algorithmic design** to identify the appropriate implementable algorithms
2. **tuned implementations** that consider data storage formats and available hardware features

Target Problem

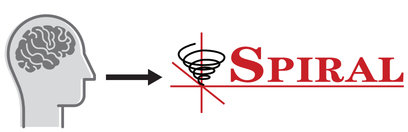
Using triangle counting as an example, we demonstrate our approach to generating graph algorithms from their mathematical specification.

Mathematical Specification

$$\Delta = \frac{1}{6}\Gamma(A^3)$$

Proposed Solution

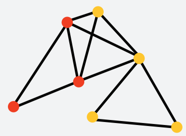
Encode expert knowledge about algorithm design and optimization into an automated system (SPIRAL) to generate tuned implementations automatically. Allow the use of GraphBLAS formulae for providing mathematical specifications.



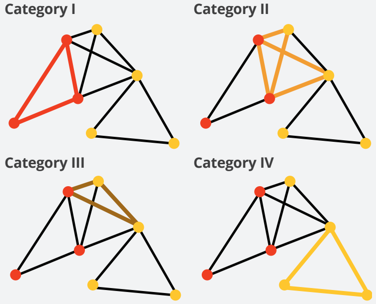
Algorithmic Design

Formally deriving algorithms from the mathematical specification.

Example: Triangle Counting



Original graph split into two subgraphs: processed (red) and unprocessed (yellow)

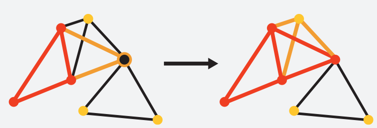


Intuition

Counting different categories of triangles as we iterate over the different vertices yields different algorithms.

Illustration

Counting Category I and II triangles, where red vertices have been processed.



Automatic Code Generation

Formalize the algorithm and implementation techniques into SPIRAL

Program Specification

```
TriangleCount()
```

$$\Delta = \frac{1}{6}\Gamma(A^3)$$

Algorithm Choice

```
Accum_VMV( TriangleCount() )
```

$$L = tril(A)$$
$$C = LL \circ L$$
$$\Delta = Reduce(C)$$

Algorithm Derivation

```
BB(
  Accum(i4, 1, X.N-1,
    Accum_X(i6, [i4, 0], i4,
      Dot([i6, add(i4, V(1)) ], [ i4, add(i4, V(1))],
                    sub(sub(X.N, i4), V(1)))
)))
```

$$\Delta = \Delta + \frac{1}{2}\alpha_{10} A_{00}\alpha_{01}$$

Abstract Code

```
program(
     func(TVoid, “transform”, [ res, IJ ],
          decl([ i6, j131, j1765, j1m31, j231, j2m31, jm32, rf63, rf64 ],
               chain(
                    assign(deref(res), V(0)),
                    loopf(i4, 1, 262110,
                         chain(
                              assign(rf63, V(0)),
                              assign(j1765, add(V(262112), IJ, nth(IJ, i4))),
                              assign(jm32, add(V(262112), IJ, nth(IJ, add(i4, V(1))))),
                              loopw(logic_and(lt(j1765, jm32), lt(deref(j1765), V(0))),
                                   assign(j1765, add(j1765, V(1)))
                              ),
                              loopw(logic_and(lt(j1765, jm32), lt(deref(j1765), i4)),
                                   // dot product
                              ),
                              assign(deref(res), add(deref(res), rf63))
) ) ) ) ) )
```

C Code

```
void tc(int *res, int *IJ) {
  for(...) {
    // VMV product
  }
}
```

Principal Investigator

Dr. Will Klieber
Software Security Engineer
Carnegie Mellon University
Software Engineering Institute

Automated Code Repair to Ensure Memory Safety

A serious limitation in assuring the security of DoD software is the inability to take a codebase and either verify that it is memory-safe or repair potential bugs to make it memory-safe. Existing static analysis tools either report an enormous number of false alarms or fail to report true vulnerabilities.

We propose to design and implement a technique for automatically repairing (in the source code) all potential violations of memory safety so that the program is provably memory-safe. For this, we do not need to solve the challenging problem of distinguishing false alarms from true vulnerabilities: we can simply apply a repair to all potential memory-safety vulnerabilities, at a cost of an often small runtime overhead. Usually only a small percent of a codebase is performance-critical; repairs to this part of the codebase might need to be manually tuned, but with an amount of manual effort much less than that of manually repairing all potential memory-safety vulnerabilities in the codebase.

In Context

This FY2018–20 project

- extends prior DoD Line-funded research in automated repair of code for integer overflow and the inference of memory bounds
- is related to CMU SEI technical work into advancements based on the Pharos static binary analysis framework, vulnerability discovery, and code diversification to avoid detection of vulnerabilities by adversaries
- aligns with the CMU SEI technical objective to make software trustworthy in construction, correct in implementation, and resilient in the face of operational uncertainties including known and yet unseen adversary capabilities

Automated Code Repair to Ensure Memory Safety

Memory-related bugs in C/C++ code are notorious for leading to vulnerabilities. We're developing techniques for automated repair of source code to eliminate such vulnerabilities and enable a proof of memory safety.

What about distinguishing false alarms from true vulnerabilities?

We repair all potential memory-safety vulnerabilities, at a cost of an often small runtime overhead. (Manual tuning might be needed for performance-critical parts.)

Intermediate Representation (IR)

Problem: Static analysis generally works best on a suitable IR, but the repair must be done on the original source code.

Solution: We augment the IR with tags that record how to transform back to source.

- Each abstract syntax tree (AST) node is tagged with a reference to corresponding original (unpreprocessed) source code text.
- We have developed a set of reversible transformations that start with the original AST and transform it to the IR.
- The IR is repaired and then transformed back to source using the tags. If a repair invalidates a tag, then the tag is ignored.
- A macro invocation is preserved if the smallest containing AST node is unchanged; otherwise, it is expanded.
- We consider only a single build config but preserve `#ifdef` s where possible.

Our ACR tool takes buggy code (shown here, a format-string vulnerability) and repairs the code to remove the vulnerability while preserving the desired functionality of the code.

Definition of Memory Safety

We say that a program is *memory-safe* if and only if, on every possible execution of the program, every memory access (read or write) is to a location in a currently allocated region.

Possible executions include those where the compiler leaves gaps of unallocated memory between variables on the stack.

Memory safety is often divided into 2 parts:

- Spatial: Writing or reading beyond the bounds of a memory region (FY18+ work)
- Temporal: Writing or reading to a region after it has been deallocated (FY19+ work)

(Dereferencing NULL is technically a mem violation, but low severity, so we ignore.)

Heuristic

If a program performs arithmetic on a memory address *p1* to obtain a new memory address *p2*, and *p2* is later dereferenced, then *p2* should be in the same allocated memory region as *p1*.

The ISO C standard actually requires compliance with this heuristic (on pain of undefined behavior) for arithmetic on values of pointer type.

Static Analysis and Repair

For each memory access **p*, we generate a precondition ensuring it is within bounds:

$$\text{MemLo}(p) \le p < \text{MemHi}(p)$$

where MemLo and MemHi are functions of the provenance (not value) of *p*:

If the value of *p* (at a particular timepoint in an execution trace) was computed by pointer arithmetic on the result of a memory allocation (e.g., malloc), then MemLo(*p*) and MemHi(*p*) denote the lower bound (incl.) and upper bound (excl.) of this memory region.

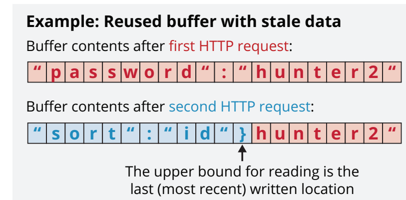
For each precond, do one of the following:

- Prove that it is satisfied.
- Add bounds check with existing variables.
- Modify function signatures and/or structs to include bounds info ("fat pointers").
- Modify program to record info about bounds in global lookup table (as in SoftBound).

In the past, fat pointers were disfavored due to inability to analyze or repair libraries available only in binary form. However, new developments in SEI's Pharos platform will allow us to overcome this limitation by tackling binaries.

Leaks of Sensitive Data via Stale Reads

Consider a web server that stores a received request in a reusable buffer. Once the server is done with the request, the buffer holds *stale* data, which can later be (partially) overwritten by a new request.



We developed a heuristic for identifying such a buffer and what part of it is valid.

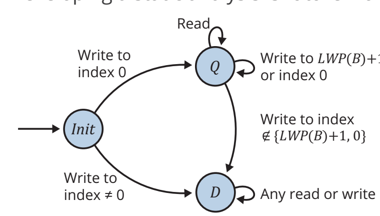
Definition: A buffer *B* is ***qualifying*** if and only if every write is to either index 0 or the successor of the last written position (LWP).

Our *sequential write* heuristic posits that a qualifying array contains valid (non-stale) data up to and including the LWP.

We implemented a dynamic analysis based on this heuristic, targeting C and Java. Our analysis detects JetLeak (CVE-2015-2080) in Jetty and Heartbleed in OpenSSL.

In analyzing GNU coreutils (80k LOC, plus 486k LOC of library), there were 17 alarms (all suspected/confirmed false positives).

Developing a static analysis is future work.



Principal Investigator

Dr. Scott McMillan
Senior Research Scientist
Carnegie Mellon University
Software Engineering Institute

Building a COTS Benchmark Baseline for Graph Analytics

Field-Programmable Gate Array (FPGA) accelerators are used in the DARPA Power Efficiency Revolution for Embedded Computing (PERFECT) program to investigate power-efficient solutions to sparse matrix problems. Because graphs can be represented by sparse matrices, we assert that this work should be expanded upon to address the problems identified in the DARPA Hierarchical Identify Verify Exploit (HIVE) program.

In this work, we propose to build a benchmark baseline based on commercial off-the-shelf (COTS) field-programmable gate array (FPGA) hardware and compete in the DARPA Graph Challenge to demonstrate the FPGA's power and performance advantages for the graph problems targeted by this program.

If successful, the graph primitives identified for FPGA could be transitioned to lower power ASIC designs and could inform chip designers by more concretely defining the measures of success for the new graph processing chip.

In Context

This FY2018 project

- builds on prior DoD Line-funded research into GraphBLAS API specification, graph algorithms on future architectures, and measuring performance of big learning workloads
- aligns with the CMU SEI technical objective to bring capabilities through software that make new missions possible or improve the likelihood of success of existing ones
- is related to FY2018 Line-funded research in Automated Code Generation for Future-Compatible High-Performance Graph Libraries (see page 10)

Building a COTS Benchmark Baseline for Graph Analytics

PageRank acceleration for large graphs with scalable hardware and two-step SpMV

Research Problem

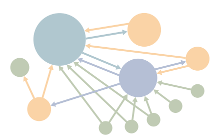
PageRank suffers from poor performance and efficiency due to notorious memory access behavior. More importantly, when graphs become bigger and sparser, PageRank applications are inhibited as most solutions strongly rely on large random access fast memory, which is not scalable.

PageRank vector:

$$x_i = \underbrace{\alpha x_i^T A}_{\text{SpMV}} + (1-\alpha) x_i^T \frac{ee^T}{N}$$

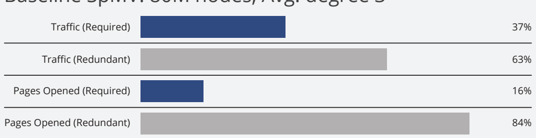
Target Graphs

- Very large (~billion nodes)
- Highly sparse (average degree<10)
- No exploitable non-zero pattern

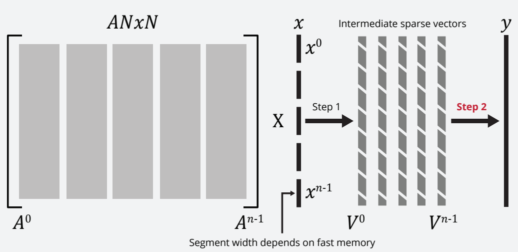


Two-Step SpMV Algorithm

Baseline SpMV: 80M nodes, Avg. degree 3



Two-step algorithm conducts SpMV in two separate steps. It requires blocking of the matrix and the source vector as shown below.

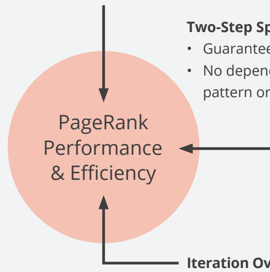


- Guarantees full DRAM streaming access
- Reduces off-chip traffic and enables high-bandwidth utilization
- Requires custom hardware for efficient multi-way merge

Proposed Solution

Custom Hardware with 3D HBM

- Efficient SpMV implementation
- Scalable—less fast memory required



Two-Step SpMV

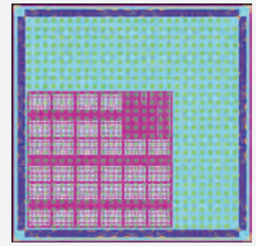
- Guarantees DRAM streaming
- No dependence on non-zero pattern or structure

Iteration Overlapped PageRank

- Saturates HBM bandwidth
- Reduces off-chip DRAM traffic

16nm FinFET ASIC for PageRank

(can also be realized in COTS FPGA)

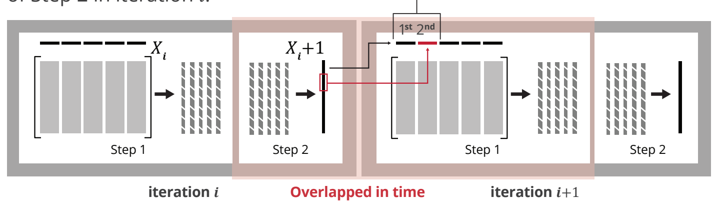


Freq.: 1.4 GHz
Area: 7.5 mm2
Power: 3.11 W

Optimized PageRank by Iteration Overlap (PR_TS_Opt)

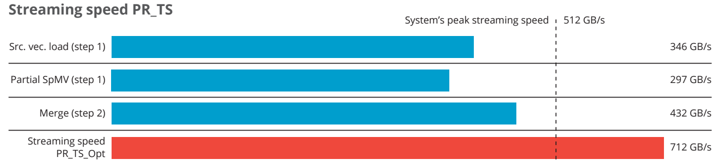
Two source vector segment storages in fast memory are required:

1) for computation of Step1 in iteration $i+1$ and 2) for storing output of Step 2 in iteration i.



- Step 2 of an iteration runs simultaneously with Step 1 of the next iteration
- Reduces off-chip traffic by eliminating DRAM round trip of both vectors
- Simultaneous Step 1 & 2 doubles the throughput and saturates HBM

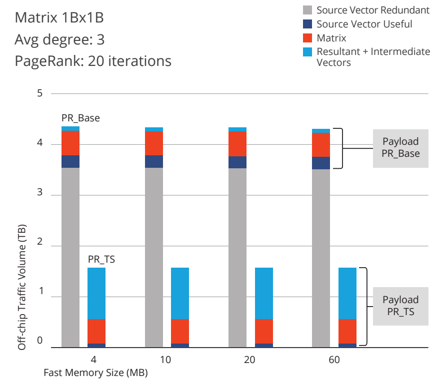
Streaming speed PR_TS

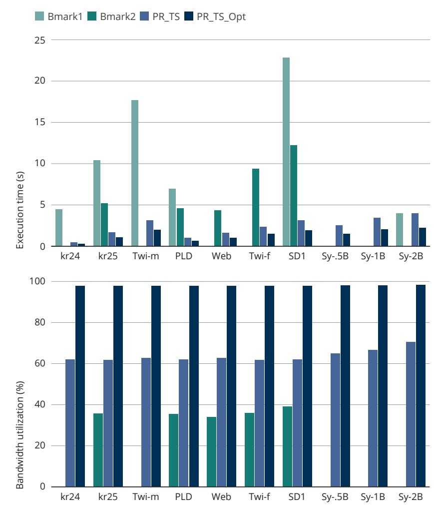


Experimental Results

PageRank Off-chip Traffic Comparison: PR_TS vs Baseline

Matrix 1Bx1B
Avg degree: 3
PageRank: 20 iterations





Reference: F. Sadi, J. Sweeney, S. McMillan, T. Z. Low, J. C. Hoe, L. Pileggi, and F. Franchetti, "PageRank Acceleration for Large Graphs with Scalable Hardware and Two-Step SpMV," at *IEEE High Performance Extreme Computing Conference*, Waltham, MA, September 2018.

Principal Investigator

Dr. Dionisio de Niz
Principal Researcher
Carnegie Mellon University
Software Engineering Institute

Certifiable Distributed Runtime Assurance (CDRA)

Leveraging rising software complexity and use of machine learning (ML) techniques, the DoD is increasingly using complex non-deterministic systems that interact with the physical world (e.g., aircraft). Runtime Assurance (RA) is a promising technique for ensuring safe behavior of those systems, because they cannot be verified satisfactorily prior to deployment. RA relies on the use of an enforcer to monitor the target system and correct (i.e., preserve the safety of) its behavior at runtime. However, current RA techniques are restricted to single domains (i.e., types of analyses); rely on unverified and potentially inconsistent enforcers that can be circumvented easily; and require system-wide re-verification when an enforcer is changed, added, or removed.

In this work, we addressed those challenges in the context of distributed real-time systems (DRTS) by creating tools and techniques to

- express enforceable policies in multiple domains, including logical and timing correctness
- verify correctness of an enforcer implementation against its policy
- combine multiple enforcers and resolve any inconsistencies between their behavior
- verify that enforcers across multiple nodes of DRTS implement a global safety policy
- deploy enforcers so that they cannot be circumvented by a well-defined attacker (i.e., has control of at least one monitored component)
- verify that the enforcers react on time to prevent physical consequences (e.g., aircraft crash)

We are validating our results on DoD-relevant examples.

In Context
This FY2017–18 project

- extends prior DoD Line-funded research in formal methods to verify DRTS
- aligns with the CMU SEI technical objective to make software trustworthy in construction, correct in implementation, and resilient in the face of operational uncertainties

Certifiable Distributed Runtime Assurance

Challenges

- Assure safety of distributed cyber-physical systems
- Unpredictable algorithms (machine learning)
- Multi-vehicle (distributed) coordinating to achieve mission

Solutions

- Add simpler (verifiable) runtime enforcer to make algorithms predictable
- Formally: specify, verify, and compose multiple enforcers
- Enforcer intercepts/replaces unsafe action at right time

Formalization (time-aware logic)

State of system: Variable Values

Statespace & Actions

- $S = \{s\}$, Safe states: $\phi \subseteq S$
- $R_P(\alpha) \subseteq S{\times}S$; $R_P(\alpha,s) = \{s' \mid (s,s') \in R_P(\alpha)\}$

Enforceable states

- $C_\phi = \{s \mid \exists\alpha{:}\ R_P(\alpha,s) \in C_\phi\}$

Safe actions:

- $SafeAct(s) = \{\alpha \mid R_P(\alpha,s) \in C_\phi\}$

Logical Enforcer: $E = (P, C_\phi, \mu)$

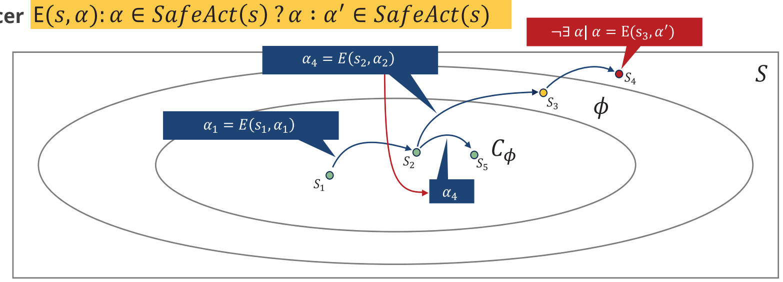
- Set of safe actions:

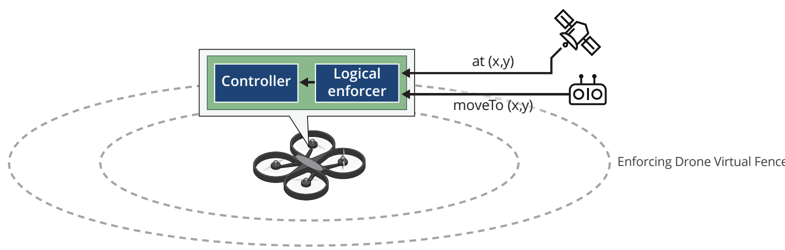
  $\mu(s) \subseteq SafeAct(s)$

- Monitor and enforce safe action:

$$(\alpha=)\begin{cases}\alpha, & \alpha \in \mu(s)\\ pick\,(\mu(s)), & otherwise\end{cases}$$

Enforcer $E(s,\alpha){:}\ \alpha \in SafeAct(s)\ ?\ \alpha : \alpha' \in SafeAct(s)$





Timing Enforcement

- Unverified software may never finish!
- => No action produced to be enforced!

Temporal Enforcer

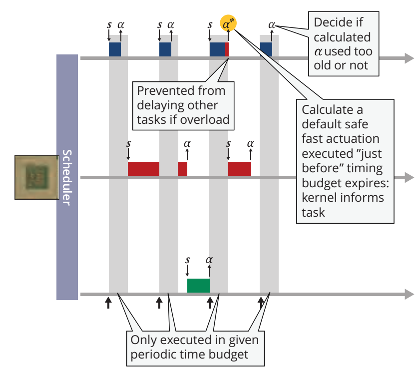
- Protect other tasks from bogus never-ending (or large) executions
- Produce default safe actuation if task takes too long

How

- Each task gets a CPU budgetstop task if budget exceeded
- If task about to exceed budget executesafe action

Timing Guarantees

- Never allow task to exceed budget
- Always execute actuation

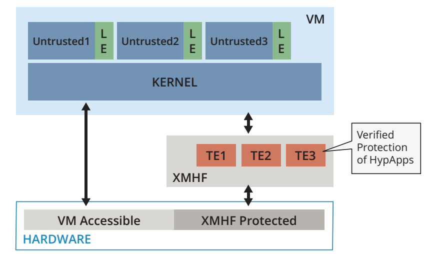


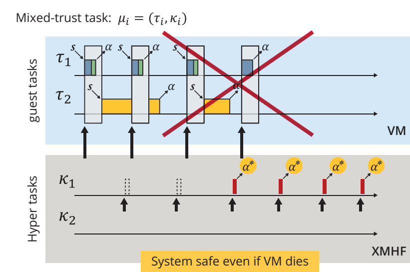
Protecting Enforcer from Untrusted Components

- Unverified software faults may corrupt enforcers

Mixed Trust Computing and Scheduling

- Untrusted/unverified software for quick/cheap fielding
- Trusted enforcer to guarantee safety properties
- Micro-Hypervisor protects enforcers
- Coordinated VM+Hypervisor schedulers guarantees timing





Enforcers Allows Verification of Complex CPS: Autonomous Vehicles

- Limit misbehavior with verifiable enforcers
- Result: Verified whole system

Verified: Logic, Timing, Physics!

KEY for Assured Autonomy

Principal Investigator

Dr. Grace Lewis
Principal Researcher
Carnegie Mellon University
Software Engineering Institute

High Assurance Software-Defined IoT Security

Despite its use of Internet of Things (IoT) devices in Supervisory Control and Data Acquisition (SCADA) systems and its interest in using such devices in tactical systems, the DoD has been slow to adopt IoT. In particular, the DoD is reluctant to use commodity IoT devices, especially in tactical systems, because of untrusted supply chains and a growing amount of reported vulnerabilities in these devices. At the same time, DoD recognizes the rapid pace at which the IoT commercial marketplace is evolving and its urgency to embrace commodity technologies, to match its adversaries.

Our proposed solution moves part of security enforcement to the network to enable the integration of IoT devices into DoD systems, even if the IoT devices are not fully trusted or configurable, by creating an IoT security infrastructure that is provably resilient to a collection of prescribed threats. It uses

- software-defined networking (SDN) and network function virtualization (NFV) to create a highly dynamic IoT security framework
- überSpark (a framework for building secure software stacks) to incrementally develop and verify security properties of elements of the software-defined IoT security infrastructure [Vasudevan 2016]

In Context

This FY2018–20 project

- builds on prior CMU SEI technical work in the mobile communication and computing needs of edge users and the authentication and authorization for IoT devices
- draws from our collaboration with CMU researchers and sponsored engagements to reduce risk through architecture analysis
- aligns with the CMU SEI technical objective to make software trustworthy in construction, correct in implementation, and resilient in the face of operational uncertainties

KalKi: High Assurance Software-Defined IoT Security

The term "KalKi" is of Sanskrit origin and derived from the Sanskrit word "Kala," which means destroyer of filth or malice and bringer of purity, truth, and trust.

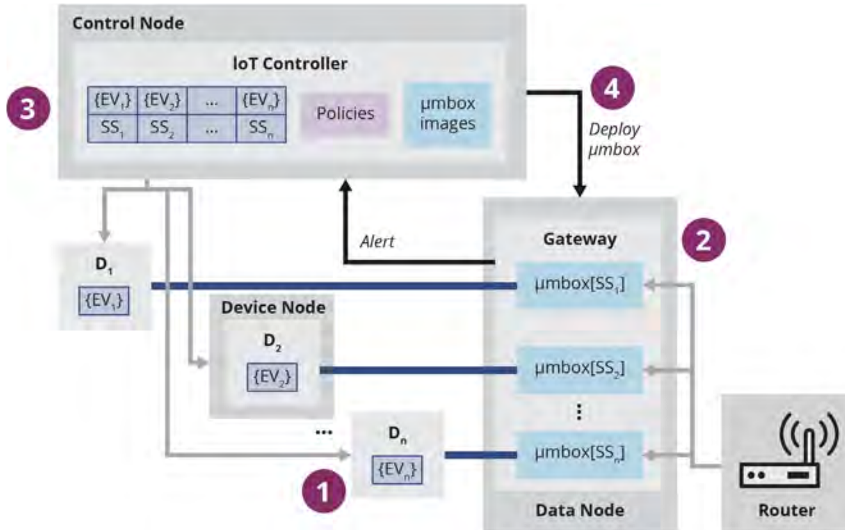
Problem

Despite the DoD's current use of Internet of things (IoT) devices in supervisory control and data acquisition (SCADA) systems, and its interest in using such devices in tactical systems, adoption of IoT has been slow mainly due to security concerns (e.g., reported vulnerabilities, untrusted supply chains).

At the same time, the DoD recognizes the rapid pace at which the IoT commercial marketplace is evolving, and its urgency to embrace commodity technologies to match its adversaries.

Solution

Move part of security enforcement to the network to enable the integration of IoT devices into DoD systems, even if the IoT devices are not fully trusted or configurable, by creating an IoT security infrastructure that is provably resilient to a collection of prescribed threats.



The "Software-Defined" Aspect

Use software-defined networking (SDN) and network function virtualization (NFV) to create a highly dynamic IoT security framework.

1. Each IoT device, D, senses/controls a set of environment variables, EV
2. Network traffic to/from each device is tunneled through µmboxes that implement the desired network defense for the device's current security state
   µmbox[SS_1] = Firewall
   µmbox[SS_2] = IPS, ...
3. IoT controller maintains a shared statespace composed of {EV} and security state (SS) for each device
   SS = {Normal, Suspicious, Attack}
4. Changes in the shared statespace are evaluated by policies and may result in the deployment of new µmboxes

The "High Assurance" Aspect

Use überSpark (a framework for building secure software stacks) to incrementally develop and verify security properties of elements of the software-defined IoT security infrastructure.

Control Node Properties

- Policy data integrity
- µmbox image storage integrity

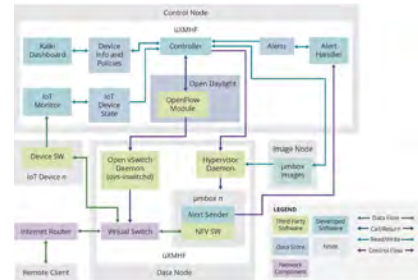
Data Node Properties

- Isolation between µmboxes of different trust levels: trusted, untrusted, and verified
- µmbox deploy-time integrity

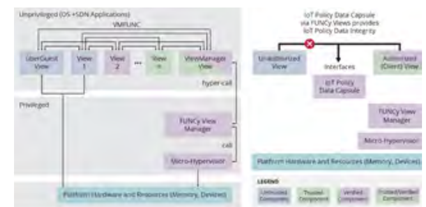
Device Node Properties

- Attestation
- Authenticated channel of communication with the IoT controller

Year 1 Highlights



Initial architecture and prototype of the IoT security framework (focus on control node)



FUNCy views (secure) system architecture: hardware-assisted, low-latency, low-TCB, legacy code compartmentalization on x86 platforms

Security Policy Model

Principal Investigator

Dr. John Klein
Senior Member of the Technical Staff
Carnegie Mellon University
Software Engineering Institute

Infrastructure as Code

DoD sustainment organizations want to adopt agile practices and realize the benefits of DevOps and IaC (Infrastructure as Code, a foundation of DevOps that provides automated deployment to the integration environment). First, however, they must recover the technical baseline for the system's software deployment.

In this project, we prototyped an end-to-end method and tools to recover deployment technical baseline (as infrastructure as code scripts) from an instance of a deployed system, including

- a rule-based analyzer tool to process the inventory of software on each node
- a deployment model definition, expressed using an extensible schema
- a generator tool that uses the schema to create IaC deployment artifacts

In Context

This FY2018 project

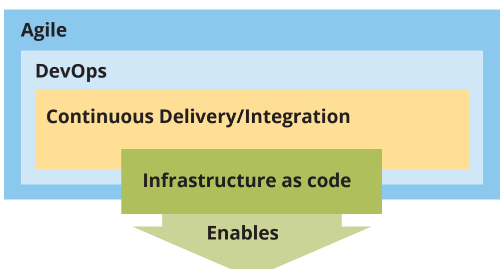
- extends CMU SEI technical work in Agile and DevOps adoption in DoD contexts
- aligns with the CMU SEI technical objective to make software delivery timely so that the cadence of acquisition, delivery, and fielding is responsive to and anticipatory of the operational tempo of DoD warfighters

Infrastructure as Code

Feasibility of recovery of software deployment architecture

DoD sustainment organizations want to adopt agile practices and realize the benefits of DevOps and infrastructure as code (IaC). They must first recover the technical baseline for the software deployment. This project has prototyped technology to automatically recover the deployment baseline and create the needed IaC artifacts, with minimal manual intervention and no specialized knowledge about the design of the deployed system.

IaC is the process and technology to manage and provision computers and networks (physical or virtual) through scripts. IaC is a foundation of integrated development and operations (DevOps) that provides automated deployment to the integration environment and repeatability through immutable infrastructure, enables exploration and experimentation by providing environment versioning and rollback, and ensures parity of test and integration environments across locations and organizations. IaC is usually associated with Agile and DevOps, but it can provide benefits outside of Agile.



Today

- Automated deployment
- Immutable infrastructure
- Versioning and rollback
- Environment parity

Future

- Portability across IaaS
- Assurance evidence
- Moving target defense

Our approach has four elements. We **crawl** through an instance of the deployed system and inspect each node to create an inventory of software. Next, we **analyze** the inventory and "make sense of it" — identify which software is part of the operating system, which other packages are installed, and which is the application software. From this analysis, we populate a **deployment model** of the system. From the deployment model, we **generate** the scripts needed by the infrastructure as code tools, which execute the scripts to create a new deployment of the system

python™

Crawl: Our crawler uses a novel approach to execute a script written in the Python programming language on the source system without installing additional software.

Analyze: We first determine the source repository for each installed package and associate files to installed packages. We then run a set of heuristic rules that uses file patterns to identify configuration files and pattern matching within configuration files to identify directories and files added to the system outside of installed packages.

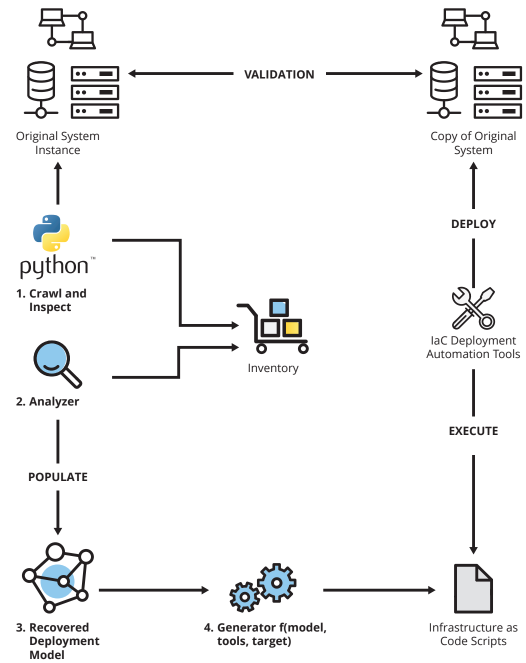
Heuristics classify files by source. The heuristic rules infer identity and source of files that are not installed with a package, such as:

- Content served up by an installed web server
- Scripts or services delivered from an installed web container like nginx or Apache Tomcat
- Configuration and schema definition files for an installed database
- Standalone user services or applications

The **deployment model** is a relational schema that represents all of the facts and inferences.

Generate: Our prototype generates a set of scripts for the open source Ansible automation tool. The prototype can be extended to generate scripts for other tools.

Approach: Crawl, analyze, populate a model, and generate IaC artifacts.



Limitations and future work: Our approach is limited to Linux-based systems. We have demonstrated an initial set of heuristics rules covering a number of inference types and patterns, with extensibility to add new rules to broaden coverage.

Software sustainment organizations can use this tool to quickly understand a system, and create and run automated deployment scripts to enable exploration and evolution.

Principal Investigators

Dr. Peter Feiler
SEI Fellow and Principal Research Scientist
Carnegie Mellon University
Software Engineering Institute

Dr. Samuel Procter
SEI Architecture Researcher
Carnegie Mellon University
Software Engineering Institute

Integrated Safety and Security Engineering for Mission-Critical Systems (ISSE-MCS)

Critical systems must be both safe from inadvertent harm and secure from malicious actors. Historically, safety and security practices have evolved and been applied in isolation. Despite recognition that this disconnect is harmful [Friedberg 2017], there is limited understanding of the interactions between safety and security.

We are overcoming the lack of understanding and knowledge by developing an integrated safety and security engineering approach based on system theory and supported by an AADL-based workbench that

- unifies safety and security analysis through a formalized taxonomy into a single source of truth to more effectively address safety and security concerns
- provides a design framework to combine safety and security mechanisms into a more robust and resilient system architecture through continuous analytic verification
- includes validated metrics to assess the effectiveness of different system designs and residual safety and security risks

In Context

This FY2018–20 project

- extends the safety analysis capability present in OSATE with conditional fault source and propagation capabilities and the requirement specification capability in ALISA, the incremental assurance component of OSATE, to support specification of reusable requirements and verification plans
- aligns with the CMU SEI technical objective to make software trustworthy in construction, correct in implementation, and resilient in the face of operational uncertainties including known and yet unseen adversary capabilities

Integrated Safety and Security Engineering for Mission Critical Systems

Progress and planning in the first year of a three-year project

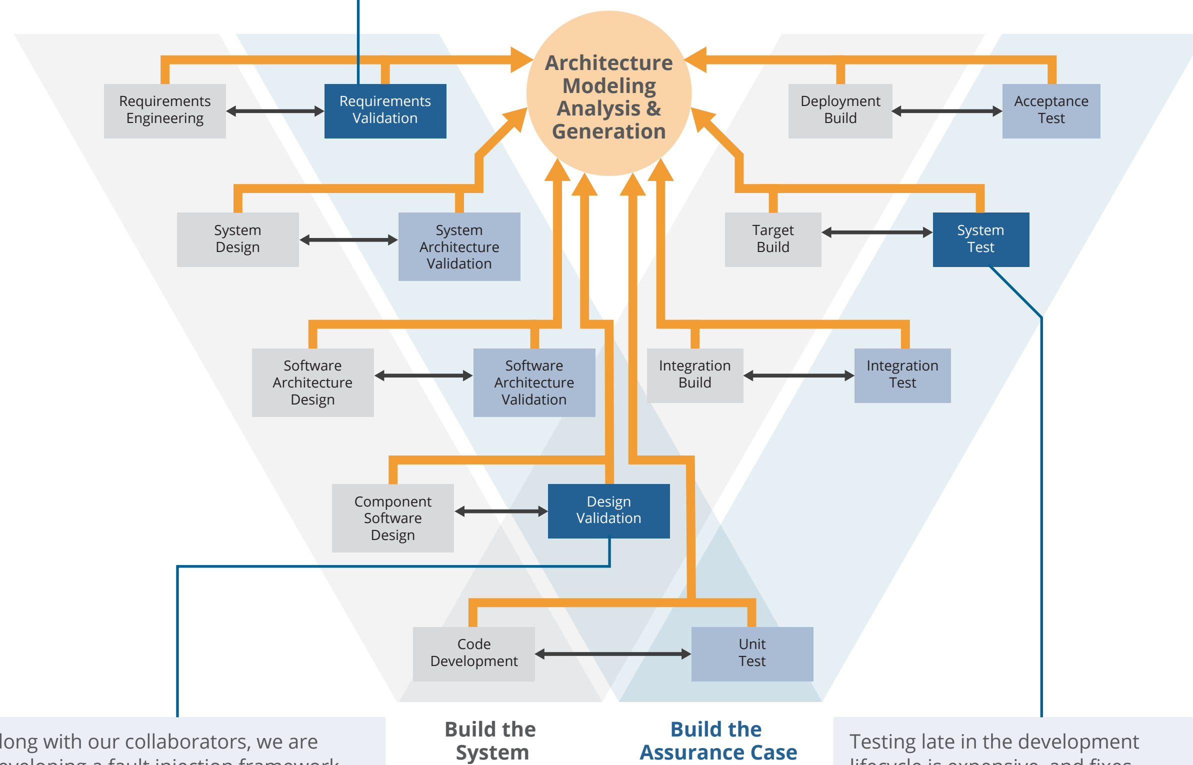
Safety-critical systems, such as airplanes and medical devices, are increasingly connected to the outside world. This adds new capabilities, but also exposes new security risks. We're looking at integrating security engineering techniques with safety processes using a system's architecture.

This work builds on years of successful research with AADL. Previously, the Architecture-Led Incremental System Assurance (ALISA) project established a toolkit and process for reasoning about safety throughout a system's development. Using this technology, we're creating guidance, examples, theory, and new tooling to guide developers of safety-critical systems to also reason accurately about security concerns.

Our development environment has tooling based on state-of-the-art hazard/threat-analysis theory. Previous work, both at the SEI and from the larger research community, has indicated that an effects-focused approach can offer a number of benefits for designing critical systems. Working with our collaborators, we're using this effects-focus to guide updates to our development environment, which is already being used in industry, commerce, and by a number of DoD contractors.

> The end result will be a tool-based, architecture-centric set of guidelines and automated analyses that brings security and safety together early in the system development lifecycle—avoiding costly and time-consuming rework.

We are identifying gaps in current architecture-centric security practices, such as poor documentation of a system's environmental assumptions. We are developing guidance, examples, and tooling to close those gaps. Where those practices conflict with safety guidance, we're documenting the tradeoffs so developers and stakeholders can be more informed.



Along with our collaborators, we are developing a fault injection framework that will let us test a component's error behavior specification. This greatly simplifies testing components in exceptional conditions—currently a very challenging task.

Testing late in the development lifecycle is expensive, and fixes required at this point are similarly costly. This project, like its predecessor ALISA, shifts issues "to the left" so they can be addressed more quickly, cheaply, and—most important— effectively.

Principal Investigator

Dr. Drew Gifford
Data Scientist
Carnegie Mellon University
Software Engineering Institute

Modeling and Explaining Sequential Behavior

A Series of Unlikely Events The DoD and the Intelligence Community (IC) frequently analyze activity based intelligence (ABI) to inform missions about routine patterns of life (POL) and unlikely events that signal important changes. However, current algorithms are typically hand-crafted for particular applications, require labeled anomalous data, and have high false positive rates that require human analysts to verify predictions [Chandola 2009, Chandola 2012, Gupta 2014]. We propose an alternative approach from the field of autonomy that does not require anomalous data to train statistical models of routines; yet it detects anomalies as the absence of routine behavior. Our technique for creating models of routine behavior reduces brittleness compared to hand-crafted detectors. Additionally, the statistical model can also explain why anomalies are detected and could be used by analysts to prioritize the anomalies and retrain models using false positive data.

What Will the Robot Do Next? The DoD, Federal agencies, and industry are increasingly using robots in important tasks such as search and rescue operations. However, when users do not expect or predict that their robots will take a particular set of actions, they believe that those actions reflect imminent failure even when they do not, resulting in a loss of trust and more frequent monitoring of the robot. In this project, we developed algorithms for robots to adapt their behavior proactively during execution to enable users to predict what the robot will do next accurately. Meeting user expectations for robot actions reduces the burden of human-robot communication to explain inappropriate robot behavior while maintaining high trust.

In Context

These projects, executed from FY2017–20

- complement prior DoD Line-funded work in GraphBLAS algorithm specification and algorithms for robots to automatically explain their behaviors
- align with the CMU SEI technical objectives to bring capabilities through software that make new missions possible or improve the likelihood of success of existing ones and to make software trustworthy in construction

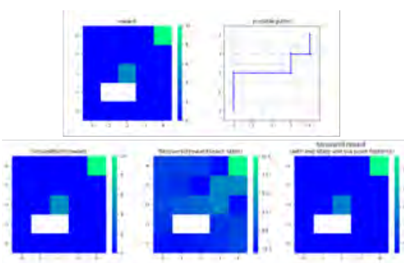
Modeling and Explaining Sequential Behavior

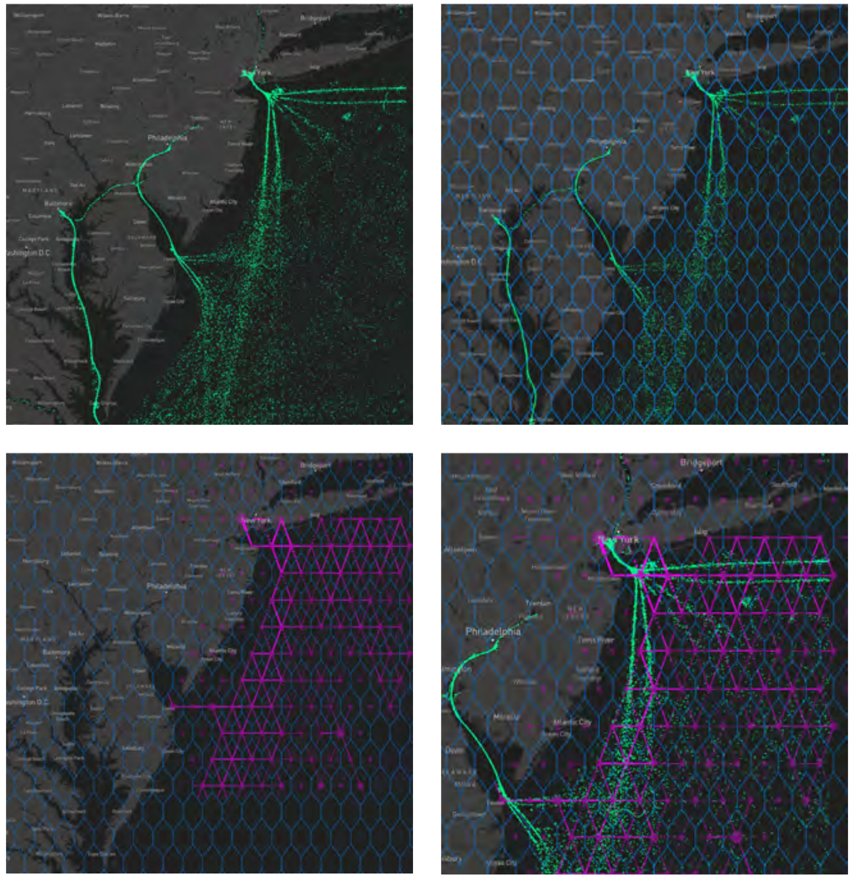
A Series of Unlikely Events and What Will the Robot Do Next?

Understanding sequential behavior is crucial to many defense-related tasks. Why did a drone make a sudden movement away from its destination? Why did a rover choose a certain path? Does a patrolling soldier's route indicate the presence of danger? Two SEI projects offer novel solutions toward modeling and explaining sequential behavior.

Identifying Unlikely Events

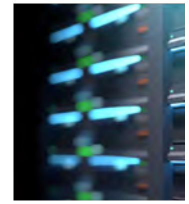
Current methods for identifying unlikely or anomalous events require labeled data about what constitutes an unlikely event and the time of human operators to verify predictions. We are using inverse reinforcement learning, an approach based in machine learning, which learns a statistical model of routine and anomalous actions that are taken from each state.





Modeling Ship Paths

Using publicly available Automatic Identification System (AIS) data collected by the U.S. Coast Guard, we use inverse reinforcement learning to model trajectories of marine vessels into New York Harbor. We can use these models to predict where vessels are going, find anomalous behavior, and potentially classify vessel type based on trajectory.

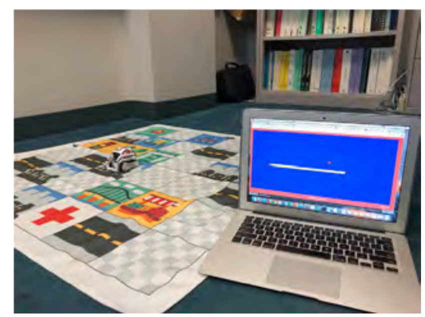


Future Work

In a collaboration with the Carnegie Mellon University Parallel Data Laboratory, we will use inverse reinforcement learning to model behavior of supercomputer users. This collaboration extends our work beyond predicting movements in the physical world and into domains such as cybersecurity, social networks, and more.

Prior Work: Explaining Robot Behavior

For human soldiers working with robot counterparts, being able to predict robot behavior ensures trust and supports human-machine teaming. Our "What Will the Robot Do Next" project has developed algorithms for robots to proactively adapt their behavior to enable users to predict what the robot will do next.



In an ongoing experiment, we are working to predict what people will focus on while performing a dual task: playing a simple video game and observing a Cozmo robot. We will collect dual task data from participants to compare to our predictive models.

Principal Investigator

Allen Householder
Senior Vulnerability & Incident Researcher
Carnegie Mellon University
Software Engineering Institute

Modeling the Operations of the Vulnerability Ecosystem

In November 2016, the Office of the Secretary of Defense launched the DoD Vulnerability Disclosure Program (VDP). CMU SEI's work to implement the VDP has shown that traditional incident and vulnerability management (VM) metrics are inadequate to address the Coordinated Vulnerability Disclosure (CVD) problem space.

Measuring Vulnerability Response (VR) solely by VM metrics underserves defenders, due to inadequate disclosure practices upstream. This inadequacy highlights a deeper problem: while many defenders are familiar with VM practices, they do not recognize the importance of the Coordinated Vulnerability Disclosure (CVD) process that feeds into it.

This work developed models, metrics, datasets, and key performance indicators for VR practices that account for CVD as well as VM.

In Context

This FY2018 project

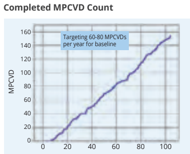
- builds on prior DoD Line-funded research into a gap exposed when defenders focus on measuring VM processes masks actual infrastructure vulnerability
- aligns with the CMU SEI technical objective to make software trustworthy in construction, correct in implementation, and resilient in the face of operational uncertainties including known and yet unseen adversary capabilities

Modeling the Operations of the Vulnerability Ecosystem

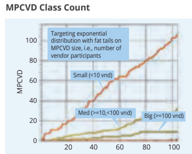
Coordinated Vulnerability Disclosure (CVD) is an emerging capability within DoD. But CVD is known to be difficult and prone to controversy when multiple vendors are involved, as in the case of recent vulnerabilities like Meltdown and Spectre. In this LENS project we modeled the factors affecting cooperation in the multiparty CVD process.

Calibration Target Ranges for Baseline

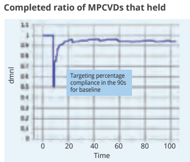
Completed MPCVD Count



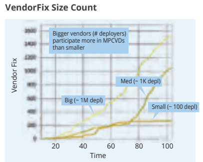
MPCVD Class Count



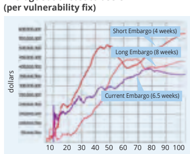
Completed ratio of MPCVDs that held



VendorFix Size Count

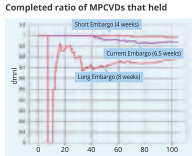


Average social cost to users (per vulnerability fix)



Current embargo (set to about 45 days) to be the lowest cost option per vulnerability fix.

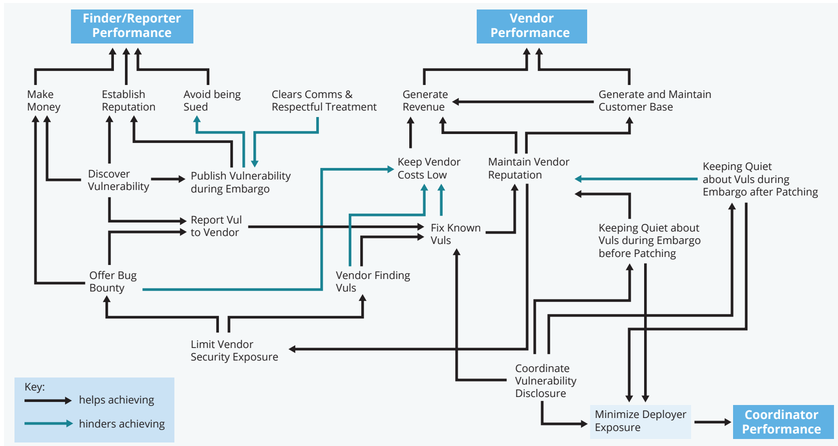
Completed ratio of MPCVDs that held



While the short embargo ensures more MPCVDs hold through the embargo period, as seen in the chart on the left, they are the most costly to users. The current embargo period is a good middle ground to reduce cost to users.

Conclusion: Adjusting the embargo period to increase the likelihood that patches can be developed JUST in time appears to be a good strategy for reducing cost.

Drivers of CVD Player Behaviors



Ventity: A Hybrid Modeling Toolset

Ventity is being developed by Ventana Systems, Inc.

- Modeling and simulation environment supporting two types of modeling
- Agent-based modeling
- System dynamics modeling
- Supports modular construction of socio-technical models for scalable development by independent teams

Used to Model the Multi-Party Coordinated Vulnerability Disclosure (MPCVD) Problem

- Finders, vendors, and MPCVDs are agents
- Simulation runs many MPCVDs over two years to assess management strategies and policies for the coordinator to try out
- Current model under development has been calibrated along several dimensions
- Adjustable model parameters include the number of finders and vendors, size distribution of the MPCVDs and vendors, embargo duration, likelihood of accidental and purposeful disclosure
- Social cost measure includes likelihood of vul exploitation, maximum amount of damage, hacker vul discovery time, attack rate per deployer, amplification of attack rate after disclosure, user workaround costs over time (adapted from Cavusoglu et al., 2007 [1].

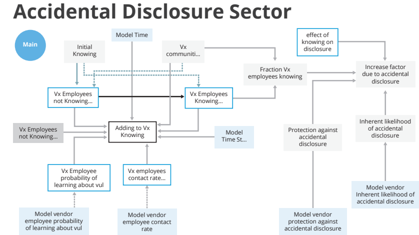
[1] Cavusoglu, H., Cavusoglu, H., & Raghunathan, S. (2007). Efficiency of vulnerability disclosure mechanisms to disseminate vulnerability knowledge. IEEE Transactions on Software Engineering, 33(3), 171-185.

The Ventity Interface

Initial Observations from Non-Validated Model

- The longer after patch development that embargo goes, the greater the chance of reneging
- The more vendors participating in MPCVDs the more early disclosures that occur
- The sooner that patches are distributed the lower the social cost to deployers, whether patch distributed (and vul disclosed) before or after embargo
- Shortening the embargo time leads to lower rates of reneging, but high rates of no patch after embargo
- Assumption: Faster patching is more costly for all vendors.

Accidental Disclosure Sector



Principal Investigators

Robert Schiela
Technical Manager, Cyber Security Foundations
Carnegie Mellon University
Software Engineering Institute

Dr. Rick Kazman
Visiting Scientist
Carnegie Mellon University
Software Engineering Institute

Predicting Security Flaws through Architectural Flaws

Previous analysis of two open source projects (Chromium and OpenSSL) has shown that about 50% of the total effort (in lines of code—LOC) to fix all security issues was spent on fixing only 10% of the security issues [Mo 2015]. That 10% had architectural design flaws—flawed relationships among the source code modules. These design flaws have been shown to be highly correlated with security bugs [Feng 2016].

In this work, we have examined the characteristics of these design flaws, using large open source projects such as Chromium, OpenSSL, and Mozilla as our analysis dataset, as a means of developing statistical predictive models. Using such models allows us to employ automated architecture analysis to identify, prevent, and mitigate security flaws.

In Context

This FY2018 project

- builds on prior DoD Line-funded research and sponsored technical work in software architecture risk evaluation and vulnerability discovery and analysis
- contributes to the CMU SEI research data repository
- aligns with the CMU SEI technical objective to make software trustworthy in construction, correct in implementation, and resilient in the face of operational uncertainties including known and yet unseen adversary capabilities

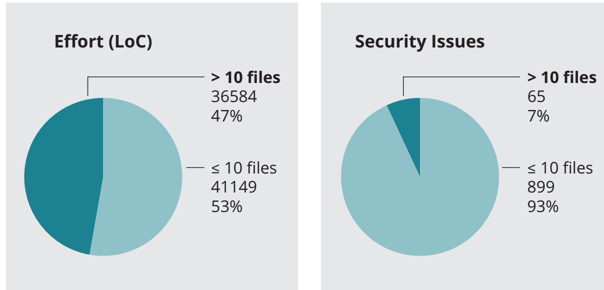
Predicting Security Flaws through Architectural Flaws

Security defects due to implementation and interface dependencies across multiple source code files are difficult and expensive to find and fix. We are evaluating the efficacy of using architectural modular analysis tools to identify security defects and the effect of refactoring on removing security defects.

Our project's goal is to use automated architecture analysis to identify, prevent, and mitigate security flaws in code. We are retrospectively analyzing open source software, with revision history and issue lists that include identified security flaws. With this data, we are identifying correlations and building models between the relationship of architectural flaws and security flaws. Future work could use this approach to identify areas of code to refactor for architectural and security improvements.

Statistical Analysis. We evaluated the correlations of the existence of security flaws with the existence of different architectural flaws. We also evaluated the effect of refactoring for reducing security flaws, based on the existence of security flaws before and after major refactoring.

Our analysis has found that some refactoring has significantly reduced security flaws, and that security flaws are commonly present with some architectural flaw types.



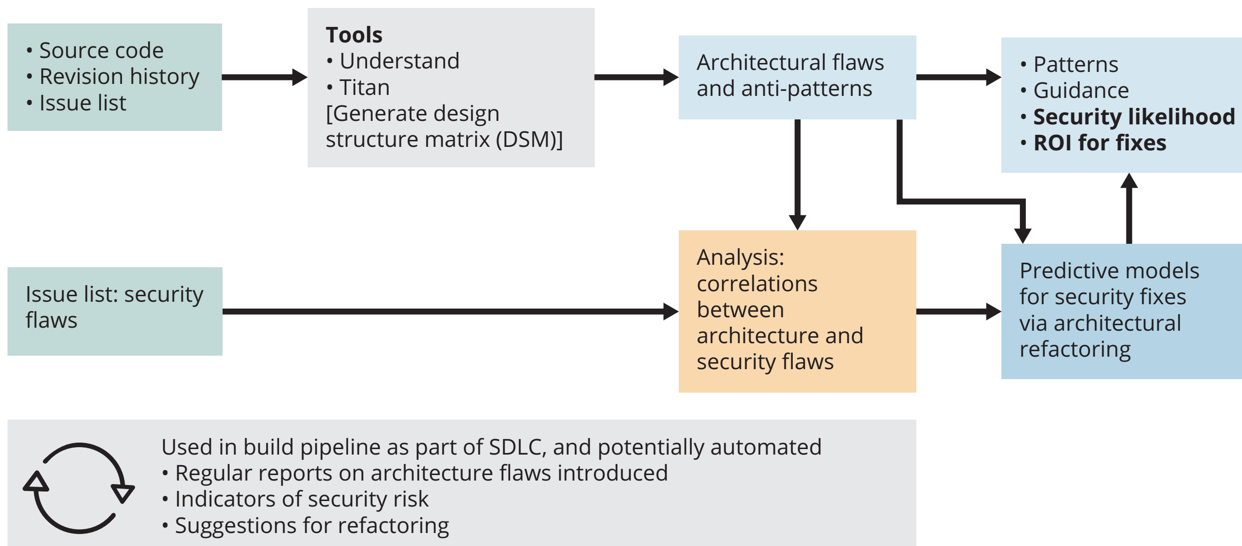
Potential Impact: ~50% of total effort (LoC) to fix security issues came from fixing <10% of the security issues in Chromium.

Improper Interface

		1	2	3	4	5	6	7	8	9	10
1	122654.content.browser.ssl.ssl_error_handler.h	(1)									
2	122654.content.browser.ssl.ssl_manager.h	I,6	(2)								
3	122654.content.browser.renderer_host.resource_dispatcher_host_impl.h	I,Pu,2	,2	(3)							
4	122654.content.browser.ssl.ssl_cert_error_handler.h	I,Pu,5	,12	,2	(4)						
5	122654.content.browser.ssl.ssl_error_handler.cc	U,I,6	,6	,2	C,I,4	(5)					
6	122654.content.browser.ssl.ssl_manager.cc	,4	C,I,11	I,2	C,I,10	C,4	(6)				
7	122654.content.browser.ssl.ssl_cert_error_handler.h	,4	C,U,I,23	I,2	I,10	,12	C,10	(7)			
8	122654.content.browser.rendererHost.socket_stream_dispatcher_host.h	I,Pu	,2		,2				(8)		
9	122654.content.browser.rendererHost.socket_stream_dispatcher_host.cc		I					C	U,I,9	(9)	
10	122654.content.browser.rendererHost.resource_dispatcher_host_impl.cc	,2	I,3	U,I,18	,2	,2	,2	C,3			(10)

C = Call; U = Use; I = Include; T = Type; S = Set; O = Override;
Pu = Public Inherit; ,# = # concurrent check-ins

Design Structure Matrix: Presents Relationships among Modules. This example identifies an Improper Interface, as it has files that are related but co-change frequently (unstable), as well as files that co-change frequently but are not related (implicit relationship).



Process for adding statistical analysis of security flaws and architectural flaws to predict yet-to-be identified security flaws, and provide confidence and a security-ROI for refactoring.

Principal Investigator

Dr. Lori Flynn
Senior Software Security Researcher
Carnegie Mellon University Software Engineering Institute

Rapid Construction of Accurate Automatic Alert Handling

Static analysis (SA) alerts about code flaws require costly human effort to validate (e.g., determine True or False) and repair. Static analysis tools produce many false positives and often generate many more alerts than can be validated manually. Previous research has shown they generate roughly 40 alerts per 1,000 lines of code [Heckman 2008], each requiring approximately 117 seconds to audit [Pugh 2010]—and many of them are false positives [Beller 2016, Delaitre 2015].

Although previous research has developed accurate SA alert classifiers (e.g., 85% accuracy in [Ruthruff 2008], 91% accuracy in our prior results [Flynn 2016], and various rates in many other studies [Heckman 2011]), DoD organizations do not use them because they lack a reference architecture to rapidly and automatically create accurate classifiers [Flynn 2017].

In this project, we are developing a reference architecture and prototype that enables rapid deployment of a method intended to automatically, accurately, and adaptively classify and prioritize alerts. The reference architecture is intended to enable organizations to reduce their validation workload by at least 60% and focus manual validation efforts on true dangerous code flaws within the remaining 40% of alerts.

In Context

This FY2018–19 project

- builds on techniques and tools we developed in prior DoD Line-funded research into prioritizing vulnerabilities from static analysis with classification models and the rapid expansion of classification models to prioritize static analysis alerts for C
- aligns with the CMU SEI technical objective to make software trustworthy in construction, correct in implementation, and resilient in the face of operational uncertainties including known and yet unseen adversary capabilities

Rapid Construction of Accurate Automatic Alert Handling System: Architecture and Prototype

Problem

Static analysis alerts for security-related code flaws require too much manual effort to triage, and **there is little use of automated alert classifier technology because of barriers of cost, expertise, and lack of labeled data.**

Solution

Develop extensible architecture for classification and advanced prioritization, building on novel test-suite data method we developed.

- Implement prototype
- Enable organizations to quickly start using classifiers and advanced prioritization by making API calls from their alert auditing tools
- Develop adaptive heuristics for classifier to adapt as it learns from test suite and "natural program" data

Approach

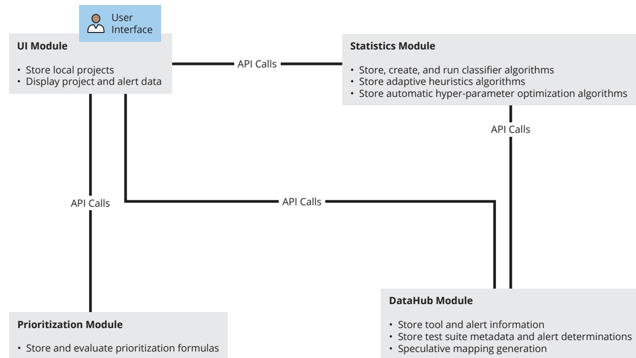
1. Design architecture
2. Develop API definition
3. Implement prototype system
4. Develop adaptive heuristics
5. Test adaptive heuristics with datasets combining test suite and real-world (DoD) data
6. Collaborators test architecture and prototype

Juliet test suite classifiers: initial results (hold-out data)

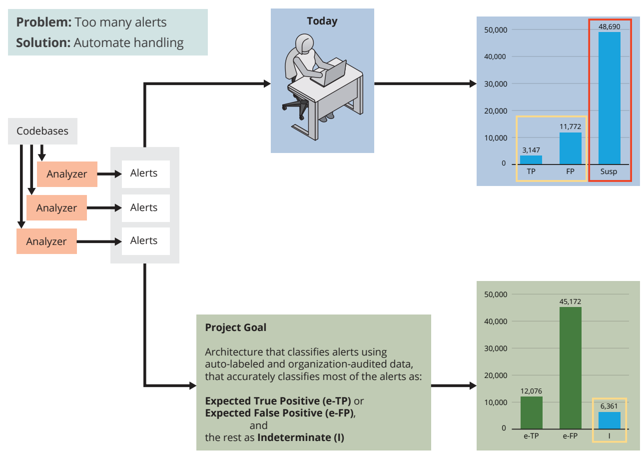
All four classification methods had high accuracy.

CLASSIFIER	ACCURACY	PRECISION	RECALL	AUROC
Random Forest	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

Architecture



Problem and Goal



Artifacts

Code and Test Results

- API definition (swagger, RESTful)
- SCALe v2 static analysis alert auditing tool with new features required for collaborators to generate data (also published on GitHub)
- SCALe v3 released Aug. 2018 (collaborators-only) with advanced prioritization schemes and features for classification
- Code development for prototype system
- Expanded archive of auto-labeled alerts
- Test results from cross-taxonomy test suite classifiers using precise mappings
- Code enabling novel "speculative mapping" method for tools without mappings to test suite metadata's code flaw taxonomy
- Adaptive heuristic development and testing results (in progress)

Non-Code Publications + Papers

Architecture API definition and new SCALe features

- Special Report: "Integration of Automated Static Analysis Alert Classification and Prioritization with Auditing Tools" (August 2018)
- Technical Report: "Integration of Automated Static Analysis Alert Classification and Prioritization with Auditing Tools: Special Focus on SCALe" (September or October 2018)
- SEI Blog Post: "SCALe: A Tool for Managing Output from Static Code Analyzers" (September 2018)

Classifier development research methods and results:

- Paper "Prioritizing Alerts from Multiple Static Analysis Tools, using Classification Models," SQUADE (ICSE workshop)
- SEI Blog Post: "Test Suites as a Source of Training Data for Static Analysis Alert Classifiers" (Apr 2018)
- SEI Podcast (video): "Static Analysis Alert Classification with Test Suites" (September 2018)
- In-progress conference papers (4): precise mapping, architecture for rapid alert classification, test suites for classifier training data, API development

Precise mappings on CERT C Standard wiki

- Metadata for Juliet (created to test CWEs) to test CERT rule coverage
- Per-rule precise CWE mapping

Continuing in FY19

Using test suite data for classifiers, research:

Adaptive heuristics

- How classifiers incorporate new data
- Test suite vs. non-test-suite data
- Weighting recent data

Semantic features for cross-project prediction

- Test suites as different projects

This project developed an architecture and API definition for static analysis alert classification and advanced alert prioritization, plus major parts of a prototype system.

FY16

- Issue addressed: classifier accuracy
- Novel approach: **multiple static analysis tools as features**
- Result: increased accuracy

FY17

- Issue addressed: **too little labeled data for accurate classifiers for some conditions** (CWEs, coding rules)
- Novel approach: **use test suites to automate production of labeled** (True/False) **alert archives for many conditions**
- Result: high accuracy for more conditions

FY18

- Issue addressed: **little use of automated alert classifier technology** (requires $$, data, experts)
- Novel approach: **develop extensible architecture with novel test-suite data method**
- Result: extensible architecture, API definition, software to instantiate architecture, adaptive heuristic research

Principal Investigator

Dr. Rick Kazman
Visiting Scientist
Carnegie Mellon University
Software Engineering Institute

Rapid Software Composition by Assessing Untrusted Components

While third-party components, including open source components, have long been a foundation for DoD software [MITRE 2017], there is a recognition that we may need to adopt greater numbers of such components—and in a more agile fashion. There is likewise a recognition that we may need take on more risk to deliver capabilities more rapidly.

In this research, we provide component scorecards based on project health measures and quality attribute indicators that will enable the automated assessment of external components with greater developer confidence, supporting rapid software delivery [Cervantes 2019].

In Context

This FY2018 project

- builds on CMU SEI technical work and expertise in the role and responsibilities of the software architect
- aligns with the CMU SEI technical objective to make software delivery timely so that the cadence of acquisition, delivery, and fielding is responsive to and anticipatory of the operational tempo of DoD warfighters

Rapid Software Composition by Assessing Untrusted Components

Today no organizations build software-intensive systems from the ground up; everyone builds applications on top of existing platforms, frameworks, components, and tools. Hence today's software development paradigm challenges developers to build trusted systems that include increasing numbers of untrusted components.

The software industry as a whole has increasingly adopted open source and commercial components as fundamental building-blocks of their systems. The U.S. Army has recently created an initiative to deliver capability more quickly—the Rapid Capability Office. While third-party components, including open source components, have long been one of the foundations for DoD software, there is a recognition that we may need to adopt greater numbers of such components, and in a more agile fashion. There is likewise a recognition that, to deliver capabilities more rapidly, we may need to take on more risk.

Our research challenge is: how to speed up the component qualification, analysis, and evaluation process while choosing appropriate levels of risk? Component scorecards, automatically constructed, can provide rapid insight into many important quality attributes and community attributes. These indicators can then be used to determine risk and to plan additional (human-intensive) analyses [1].

[1] H. Cervantes, J. Ryoo, R. Kazman, "Data-driven selection of application frameworks during architectural design," Proceedings of HICSS 52, January 2019

Example Component Scorecard

QUALITY ATTRIBUTE	TOOL	INDICATOR	COMPONENT	
			Dlib 19.10	OpenCV 3.3.1
Performance	Instrumentation	Time (ms)	44,172	55,978
	gperf	Time (ms)	47,480	58,400
	valgrind callgrind	Instructions (billions)	491	272
Memory	Memcheck	Bytes lost	288	17,127
	Memcheck	Heap usage (Mbytes)	4,591	1,093
Modifiability	DV8	Decoupling level	0.51	0.79
	DV8	Propagation cost	0.31	0.14
	Understand	SLOC	276,825	783,344
Security	FlawfinderRaw	Hits 3+	162	676
Community	CodeMaat	Authors >5 commits	13	234
		Total commits	7,191	18,272

In this research we have shown how to increase both the speed and confidence of the component selection process. We have provided component scorecards based on project health measures and quality attribute indicators that enable the automated early assessment of external components with greater developer confidence, supporting rapid software delivery. Our approach is to apply existing automated analysis techniques and tools (e.g., code and software project repository analyses), following the current industry trend towards DevOps, mapping the extracted information to common quality indicators from DoD projects.

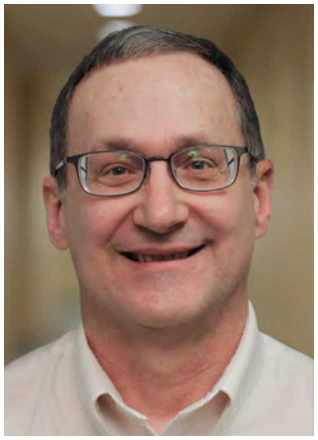
Such scorecards are not the end of analysis, but rather the beginning. They can give rapid insight that allows architects to do triage, quickly and with confidence eliminating some components and providing a context for additional deeper analysis on the remaining components. Raw scores can be aggregated using weighting functions that reflect the importance of each measure to the project, for example

$$\text{Score} = (wM_1 * wM_2) + 2(wP_1 * \log wP_2) + 3(wS_1)$$

Furthermore, by automating the analyses, components can be re-qualified every time they change for relatively low incremental costs. If an indicator changes in a non-trivial way, a deeper analysis can then be performed.

In this way we can balance the needs of agility with the needs of proper component qualification.

Principal Investigators



Edwin Morris
Deputy Technical Director,
Critical System Capabilities
Carnegie Mellon University
Software Engineering Institute

Kevin Pitstick
Member of the Technical
Staff—Engineer
Carnegie Mellon University
Software Engineering Institute

Summarizing and Searching Video

The U.S. relies on surveillance video to determine when activities of interest occur in a surveilled location. Yet, there is a lack of automated tools available to assist analysts in monitoring real-time video or analyzing archived video [Seligman 2016]. As a result, analysts now need to dedicate full attention to video streams in order to avoid missing important information about ongoing activities and patterns of life; and in tactical settings, warfighters miss critical information for improved situational awareness because they cannot stare at a tablet strapped to their chest.

In this work, we are developing

- algorithms for video summarization, activity and pattern of life detection, searching, and alerting
- a prototype architected to exploit new algorithms

In Context

This FY2018–20 project

- builds on prior DoD Line-funded research into the foundations for summarizing and learning latent structure in video, structural multitask transfer learning for improved situational awareness, and generalizing supervised latent Dirichlet allocation (a strategy to provide supervision hints to an unsupervised algorithm)
- draws from sponsored engagements for DoD programs and agencies
- aligns with the CMU SEI technical objective to bring capabilities through software that make new missions possible or improve the likelihood of success of existing ones

Summarizing and Searching Video

The volume of aerial surveillance video is outpacing the current, manual monitoring capabilities. DoD analysts need capabilities that reduce workload by identifying, summarizing, and creating alerts about critical events and patterns.

Initial Approach

Our initial approach (Figure 1) applied object detection, tracking, and stacked sparse LTSM auto encoders to identify unique segments of video for a summary.

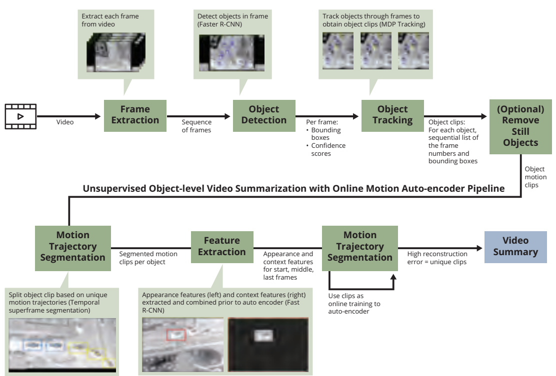


Figure 1: Unsupervised Summarization

This approach proved useful for triaging large volumes of video data to select a subset for further scrutiny, but did not provide useful summaries of the most important events in a video. The result was similar to a movie trailer that provides information useful for deciding whether or not to see the movie, but not for determining the movie's plot.

Based on these results, we revised our approach to better match the needs of an analyst.

Revised Strategy: Pattern of Life Analysis

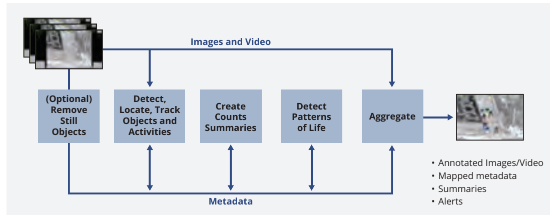


Figure 2: Analysis Pipeline

Our revised strategy focuses on a common surveillance problem of identifying situations of interest relative to a specific area being observed (e.g., a compound). We are:

- selecting and training state-of-the-art classifiers and trackers on images and video representative of DoD aerial surveillance scenarios gathered during these scenarios
- analyzing extracted objects and tracks using statistical and machine learning techniques to summarize data, recognize interactions, and determine patterns of life at the compound

Analysts will be able to:

- set alerts for specific objects, activities, or patterns
- display summaries of detections over time and space
- summaries and alerts will be provided for increasingly complex forms of analysis, from recognition of the signatures of specific objects to prescriptions for suggested courses of action

Example: Anomalous Track Detection

As an initial step toward understanding tracks, we developed a LSTM-Autoencoder algorithm to detect unusual tracks. The algorithm takes as input sequences of spatiotemporal points (tracks) and calculates an anomaly score for each track.

The LSTM-Autoencoder works by learning a compact representation for each track, then attempts to reconstruct that track from the encoding. Behaviors which are under-represented in the training set will be difficult to reproduce and have a high reconstruction error.

Figure 3: Anomalous (Zigzag) Track

To test our algorithm, we created synthetic track data using the SUMO traffic simulator on a set of streets surrounding the SEI (see Figure 3). Each generated track was a best path between a randomly chosen start and stop location. We inserted an additional handed edited track "ZigZag" that was not a best path to see if our algorithm could detect it.

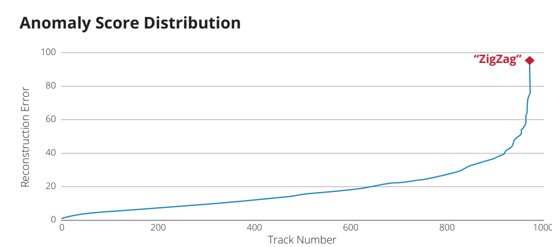
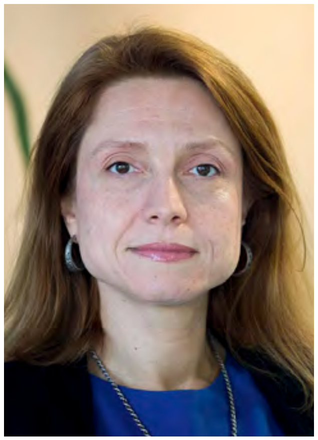


Figure 4: Score Distribution

Figure 4 shows the distribution of score values for all 974 tracks. "ZigZag" track is marked on the graph and had the 2nd highest observed anomaly score. The highest scoring track experienced multiple traffic jams at multiple intersections along its route. Another high scoring (anomalous) track was a vehicle making a u-turn.

Next steps include tuning autoencoder performance and testing against DoD data.

Principal Investigators



Dr. Ipek Ozkaya
Principal Researcher
Carnegie Mellon University
Software Engineering Institute

Dr. Robert Nord
Principal Researcher
Carnegie Mellon University
Software Engineering Institute

Technical Debt Analysis through Software Analytics

DoD and other government acquisition managers need capabilities to assess what kind of technical debt their developers and software contractors are creating through their decisions. Current solutions rely primarily on code analysis and fail to differentiate among design issues that lead to accumulating rework costs. The challenge is to provide the development teams and the government with software analytics capabilities that allow them to analyze the quality of the software and consequences of the design choices made continuously.

In this work, we developed tools that analyze data from multiple, commonly available sources to pinpoint problematic design decisions in a repeatable and reliable way for uncovering technical debt. Improving identification of such issues and quantifying the effect on accumulating rework provides data to help DoD control lifecycle costs, mitigate technical risk, and reduce cycle times.

In Context

This FY2017–18 project

- extends prior DoD Line-funded research on sustainability and technical debt
- is related to DoD Line-funded research and sponsored work engagements in software cost estimation that apply machine learning, causal modeling, and systems thinking
- aligns with the CMU SEI technical objective to make software affordable such that the cost of acquisition and operations, despite increased capability, is reduced and predictable and ensures an acquisition cost advantage over our adversaries

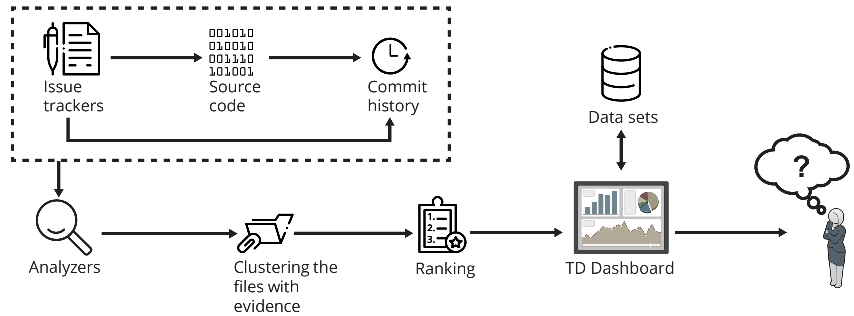
Data-Driven Technical Debt Analysis

Technical debt conceptualizes the tradeoff between the short-term benefits of rapid delivery and the long-term value of developing a software system that is easy to evolve, modify, repair, and sustain. In this work, we extended and developed tools that integrate data from multiple commonly available sources to pinpoint problematic design decisions and quantify their consequences in a repeatable and reliable way for uncovering technical debt. Our results provide a set of analysis approaches for developers' and software managers' technical debt management tool box.

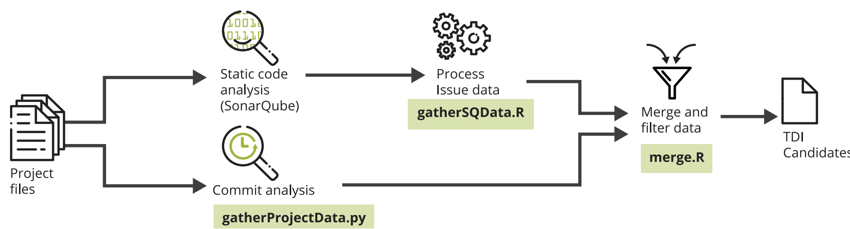
Our outcomes include the following:

- Classifier using gradient-boosting machines that assists in estimating the tickets that contain technical debt discussions by developers
- Design violation view from code quality rules, reducing the space of technical debt investigation by 95%
- Pipeline to extract candidate technical debt items, ranked by amount of evidence, which teams can use for assessing and scoping their technical debt

Technical Debt Analytics

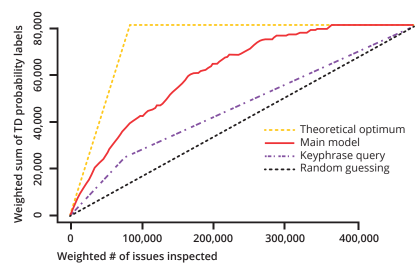


Analysis Pipeline to Generate Candidate Technical Debt Items



Technical Debt Classifier Performance

- First of its kind active-learning pipeline, which resulted in 1,934 labeled technical debt examples
- Feature engineering to combine discussion length, *n*-grams, key phrases, concepts, and document context with guidelines for key phrases that signal technical debt



	Accuracy*	Precision*	Recall*	AUROC*
no TD	0.90	NA	0.00	0.50
keyphrase query	0.83	0.26	0.35	0.62
main model	0.87	0.40	0.62	0.88

Analysis Pipeline to Generate Candidate Technical Debt Items

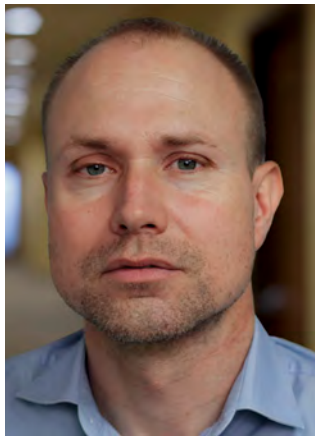
Design problems, frequently the result of optimizing for delivery speed, are a critical part of long-term software costs. We developed an algorithm and analysis pipeline that maps existing static analysis rules to design issues. The goal of the analysis is to help teams focus their attention in areas of the codebase causing extra work earlier in the lifecycle.

Sample Technical Debt Items

TDI candidate	Design paradigm technical debt issue
DFSA***.java	Logging should be centralized to avoid security and data management issues
DFS***s.java	Credentials and IP addresses should not be hard coded Deprecated code should be removed
FS*****.java -F****m.java	Redundant exceptions propagate errors and create vulnerabilities and resource management issues
-F****ry.java -B****.java -F*****p.java	Connected files propagate issues

The SEI team has been a pioneer in advancing the practices and research agenda in managing technical debt. The experiences of the team will culminate in a practitioner book, scheduled to be published in early 2019 by Addison-Wesley. You can find all tools and resources at **sei.cmu.edu/go/technicaldebt**.

Principal Investigator



Dr. Bjorn Andersson
Senior Member of the Technical Staff
Carnegie Mellon University
Software Engineering Institute

Timing Verification of Undocumented Multicore

The lack of timing verification of undocumented multicore is an obstacle for the use of multicore processors in safety-critical systems. Real-time properties cannot be verified on undocumented multicore systems today because

- processor cores typically share memory
- many undocumented shared hardware resources are present in the memory and many of these resources need to be used during a memory access
- the time it takes to execute a memory operation depends on resources that are undocumented
- the time from when a thread is requested to execute until it has finished execution depends on resources that are undocumented

In this project, we developed an abstraction and corresponding analysis that allow timing verification of undocumented hardware [Andersson 2018].

In Context

This FY2018 project

- builds on prior DoD Line-funded research and sponsored work into certifiable distributed runtime assurance, verifying distributed adaptive real-time systems, high-confidence cyber-physical systems, and real-time scheduling for multicore architectures
- aligns with the CMU SEI technical objective to bring capabilities through software that make new missions possible or improve the likelihood of success of existing ones

Timing Verification of Undocumented Multicore

Today, almost all computers use multicore processors. Unfortunately, satisfying hard real-time requirements of software executing on such computers is challenging because the timing depends on how resources in the memory system are shared, and this information is typically not publicly available. This project addresses this problem.

Multicore processors. Today, almost all computers use multicore processors. These computers have many processor cores such that one program can execute on one processor core and another program can execute on another processor core simultaneously (true parallelism). Typically, processor cores share memory. In today's memory system, there is a large number of resources that are used to make memory accesses faster in general but unfortunately also make execution time more unpredictable and dependent on execution of other programs (because these other programs use shared resources in the memory system). A simplified view of a multicore processor with the memory system is shown in Figure 1.

Embedded real-time cyber-physical systems. These systems are pervasive in society in general, as shown by the fact that 99% of all processors produced are used in embedded systems. In many of these systems, computing the correct result is not enough; it is also necessary to compute the correct result at the right time.

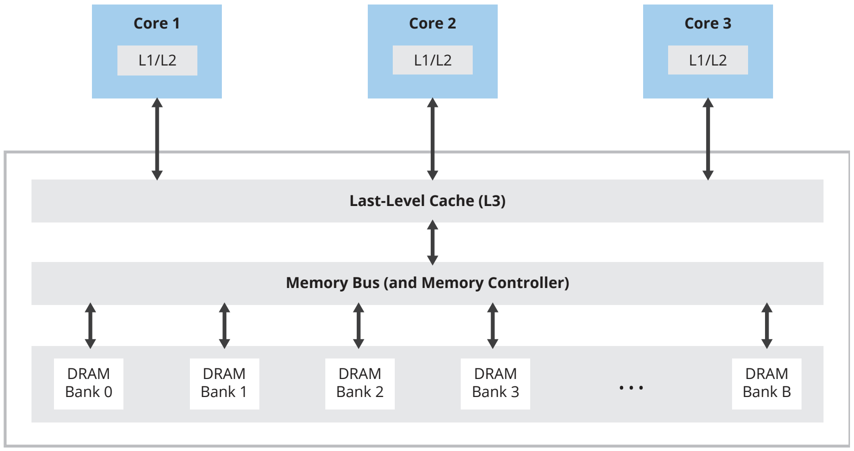


Figure 1: A simplified view of a multicore processor with shared memory

Figure 2: An example of a DoD system that aims to use multicore processors
Photo: CPT Peter Smedberg

Department of Defense (DoD). Embedded real-time cyber-physical systems are pervasive in the DoD. An example is shown in Figure 2. Because of the importance of achieving predictable timing, it is common for practitioners to disable all processor cores except one (hence making a multicore processor behave as a single processor system). The importance of timing was recently stressed by AMRDEC's S3I director, stating [1]:

> *"The trick there, when you're processing flight critical information, it has to be a deterministic environment, meaning we know exactly where a piece of data is going to be exactly when we need to—no room for error," Langhout says. "On a multi-core processor there's a lot of sharing going on across the cores, so right now we're not able to do that."*

Current solutions. Current state-of-the-art makes solutions available for managing contention for resources in the memory system and for analyzing the impact of this contention on timing. These methods assume that one knows the resources in the memory system; unfortunately, most chip vendors do not make this information available.

Problem addressed. In this project, we have addressed the problem of verifying timing of software executing on a multicore processor assuming that we do not know the resources in the memory system.

Results. We have developed a preliminary method—see B. Andersson et al., "Schedulability Analysis of Tasks with Co-Runner-Dependent Execution Times," ACM Transactions on Embedded Computing Systems, 2018.

[1] *"Army still working on multi-core processor for UH-60V,"* May 2017, Available at https://www.flightglobal.com/news/articles/army-still-working-on-multi-core-processor-for-uh-6-436895/

Principal Investigator

Dr. Nathan VanHoudnos
Machine Learning Research Scientist
Carnegie Mellon University
Software Engineering Institute

Towards Security Defect Prediction with AI

Static analysis tools have long been the main means employed to predicting security defects in source code, a need of significant national security interest (e.g., NIST SAMATE project). However, existing static analysis tools have unacceptable performance [Oliveira 2017].

In this project, we compare the state-of-the-art Artificial Intelligence (AI) system to existing static analysis approaches. We find that the AI system can approach the performance of best in class static analyzers; however, it fails to generalize, limiting its use in practice.

Our work identified two ways to improve AI on code: (1) using more expressive representations of source code and (2) using more complex deep learning architectures that allow for arithmetic relationships to be generalized from training data. We also released our dataset sa-bAbI, which will be included in NIST SARD to allow for further development of AI approaches by other researchers.

Please see our pre-print arXiv.org “Towards security defect prediction with AI” for additional information.

In Context

This FY2018 project

- contributes to the improvement of software assurance for the DoD
- is related to CMU SEI technical work in vulnerability discovery and secure coding
- aligns with the CMU SEI technical objective to make software trustworthy in construction, correct in implementation, and resilient in the face of operational uncertainties including known and yet unseen adversary capabilities

Towards Security Defect Prediction with AI

In this study:

- We investigate the limits of the current state-of-the-art AI system for detecting buffer overflows and compare it with current static analysis tools.
- We develop a code generator, sa-bAbI, capable of producing an arbitrarily large number of code samples of controlled complexity.

Static analysis tools considered:

- Frama-C – "A collection of scalable, interoperable, and sound software analyses" for ISO C99 source code. Uses abstract interpretation.
- Clang – Based on symbolic execution and, by default, uses unsound heuristics such as loop unrolling to contend with state space explosion.
- Cppcheck – We believe it also uses unsound heuristics, though little has been published about its specific approach.
- Anonymized commercial tool – Well known to be unsound.

sa-bAbI generator

- Modeled after bAbI from Weston et al. 2015, [1]
- Intentionally very simple
  - Valid C code
  - Conditionals
  - Loops
  - Unknown values such as rand()
- Complements existing software assurance datasets for training AI
- Will be included in NIST SARD

A memory network based on Choi et al., 2016 [2]

Input:

- A program code X $[N \times J]$, consisting of N lines $X_1, \ldots, X_N$, where each line X_i is a list of integer tokens $w_i^1, \ldots, w_i^J$
- A query line q $[1 \times J]$, equal to one of the lines X_i encoding a buffer write

Embedding: We fix an embedding dimension d and establish two learnable embedding matrices E_{val} and E_{addr}, both of dimension $V \times d$. Letting A represent both E_{val} and E_{addr}, we encode each integer token twice, letting Aw_i^j $[1 \times d]$ be the w_i^j-th row of A. For $i = 1, \ldots, N$, define m_i $[1 \times d]$ by

$$m_i = \text{Dropout}_{0.3}\left(\sum_{j=1}^{J} l_j \cdot Aw_i^j\right)$$

$$l_j^k = (1 - j/J) - (k/d)(1 - 2j/J)$$

We store the lines m_i encoded by E_{val} in a matrix M_{val} $[N \times d]$, and store the lines encoded by E_{addr} in a matrix M_{addr}. We embed the query line q by E_{addr} and store the result in u^1 $[1 \times d]$.

Memory search: For each "hop number" $h = 1, \ldots, H$ in a fixed number of "hops" H:

$$p\ [N \times 1] = \text{softmax}(M_{\text{addr}} u^T)$$

$$o\ [1 \times d] = \sum_{i=1}^{N} p_i (M_{\text{val}})_i$$

$$(*)\ r\ [1 \times d] = R_h o$$

$$(*)\ s\ [1 \times d] = \text{Norm}_h(r)$$

$$u^{h+1}\ [1 \times d] = u^h + s$$

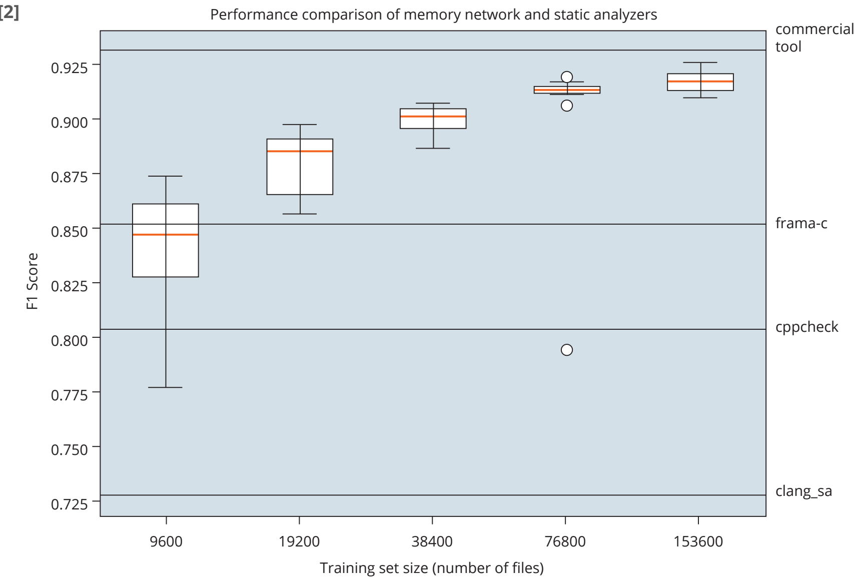
where R_h $[d \times d]$ is an internal learnable weight matrix

Classification:

$$\hat{y}\ [2 \times 1] = \text{softmax}(W(u^H)^T)$$

where W $[2 \times d]$ is a learnable weight matrix.

The forward pass is effectively an iterative inner-product search matching the current query line u^h, which changes with each processing hop, against each line m_i of the stored memory, which remains fixed.



Example Code

```
#include <stdlib.h>              // OTHER
int main()                       // OTHER
{                                // OTHER
    int entity_4;                // BODY
    char entity_8[11];           // BODY
    int entity_3;                // BODY
    int entity_0;                // BODY
    entity_0 = 9;                // BODY
    entity_3 = rand();           // BODY
    entity_4 = 42;               // BODY
    if (entity_3 < entity_0){    // BODY
    } else {                     // BODY
    entity_3 = 69;               // BODY
    }                            // BODY
    while(entity_4 < entity_3){  // BODY
    entity_4++;                  // BODY
    }                            // BODY
    int entity_9;                // BODY
    char entity_7[50];           // BODY
    entity_8[entity_4] = 's';    // BUFWRITE_COND_UNSAFE
    entity_9 = 75;               // BODY
    entity_7[entity_9] = 'S';    // BUFWRITE_TAUT_UNSAFE
    return 0;                    // BODY
}                                // OTHER
```

We found:

- **Static analysis engines have good precision but poor recall on our dataset.**
- **The state-of-the-art AI system can achieve similar performance to the static analysis engines, but it requires an exhaustive amount of training data to do so.**

Our future work:

- **Using representations of code that can capture appropriate scope information.**
- **Using deep learning methods that are able to perform arithmetic operations.**

[1] J. Weston et al., "Towards AI-Complete Question Answering: A Set of Prerequisite Toy Tasks," arXiv:1502.05698 [cs.AI], 19-Feb-2015.

[2] M.-J. Choi, S. Jeong, H. Oh, and J. Choo, "End-to-End Prediction of Buffer Overruns from Raw Source Code via Neural Memory Networks," arXiv:1703.02458 [cs.SE], 07-Mar-2017.

References

[Andersson 2018] Andersson, B., Kim, H, de Niz, D., Klein, M, Rajkumar, R. (Raj) & Lehoczky, J. Schedulability Analysis of Tasks with Co-Runner-Dependent Execution Times. ACM Transactions on Embedded Computing Systems. 17. 3. Article 71.

[Beller 2016] Beller, Moritz, et al. Analyzing the state of static analysis: A large-scale evaluation in open source software. 470-481. Proc. of the 2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER), Vol. 1. March 2016.

[Cervantes 2019] Carvantes, H., Kazman, R & Ryoo, J. Data-driven selection of application frameworks during architectural design. Proc. of the 52nd Hawaii International Conference on System Sciences. January 2019.

[Chandola 2009] Chandola, Varun et al. Anomaly Detection: A Survey. ACM Computing Surveys. 09 2009. 1–72

[Chandola 2012] Chandola, Varun et al. Anomaly detection for discrete sequences: A survey. IEEE Transactions on Knowledge and Data Engineering. 24. 5. 823-839.

[Delaitre 2015] Delaitre, Aurelien et al. Evaluating Bug Finders—Test and Measurement of Static Code Analyzers. Proc. of the 2015 IEEE/ACM 1st International Workshop on Complex Faults and Failures in Large Software Systems (COUFLESS). https://ws680.nist.gov/publication/get_pdf.cfm?pub_id=918370

[Feng 2016] Feng, Q. et al. Towards an Architecture-centric Approach to Security Analysis. 221-230. Proc. of the 13th Working IEEE/IFIP Conference on Software Architecture (WICSA 2016). April 2016.

[Friedberg 2017] I. Friedberg, K. et al. STPA-SafeSec: Safety and Security Analysis for cyber-physical systems. Journal of Information Security and Applications. 34. 183-196. https://pure.qub.ac.uk/portal/files/132972897/Stpa.pdf

[Flynn 2016] Flynn, Lori. Prioritizing Alerts from Static Analysis with Classification Models. SEI Research Review, October 2016. http://resources.sei.cmu.edu/library/asset-view.cfm?assetID=474252

[Flynn 2017] Flynn, Lori. Prioritizing Security Alerts: A DoD Case Study [blog post]. SEI Blog. January 2017. https://insights.sei.cmu.edu/sei_blog/2017/01/prioritizing-security-alerts-a-dod-case-study.html

[Gupta 2014] Gupta, Manish et al. Outlier Detection for Temporal Data: A Survey. IEEE Transactions on Knowledge and Data Engineering.26. 9. 2250-2267.

[Heckman 2008] Heckman, Sarah & Williams, Laurie. On establishing a benchmark for evaluating static analysis alert prioritization and classification techniques. 41-50. Proc. of the Second ACM-IEEE International Symposium on Empirical Software Engineering and Measurement. October 2008.

[Heckman 2011] Heckman, Sarah & Williams, Laurie. A systematic literature review of actionable alert identification techniques for automated static code analysis. Information and Software Technology 53. 4. 363-387.

[MITRE 2017] MITRE. Use of Free and Open-Source Software (FOSS) in the U.S. Department of Defense. http://dodcio.defense.gov/Portals/0/Documents/OSSFAQ/dodfoss_pdf. Retrieved Aug. 30, 2017.

[Mo 2015] Mo, Ran; Cai, Yuanfang; Kazman, Rick; & Xiao, Lu. Hotspot Patterns: The Formal Definition and Automatic Detection of Architecture Smells. 51-60. Proc. of the 12th Working IEEE/IFIP Conference on Software Architecture (WICSA 2015). May 2015.

[Oliveria 2017] Oliveria, D. et al. Improving Software Assurance through Static Analysis Tool Expositions. Journal of Cyber Security and Information Systems. 5. 3. Nov. 2017.

[Pugh 2010] Ayewah, Nathaniel & Pugh, William. The google findbugs fixit. 241-252. Proc. of the 19th International Symposium on Software Testing and Analysis. July 2010.

[Ruthruff 2008] Ruthruff, Joseph R. et al. Predicting accurate and actionable static analysis warnings: an experimental approach. 341-350. Proc. of the 30th International Conference on Software Engineering. May 2008. https://ai.google/research/pubs/pub33330.pdf

[Seligman 2016] Seligman, Lara. Interview: Air Force Chief Scientist Dr. Greg Zacharias. Defense News. February 20, 2016. http://www.defensenews.com/story/defense/policy-budget/leaders/interviews/2016/02/20/interview-air-force-chief-scientist-dr-greg-zacharias/80424570/

[Singh 2016] Singh, R. et al. Profiling hoax callers. 1-6. IEEE International Symposium on Technologies for Homeland Security. May 2016

[Spirtes 2010] Spirtes, Peter. Introduction to causal inference. Journal of Machine Learning Research 11. 1643-1662.

[Vasudevan 2016] Vasudevan, Amit; Chaki, Sagar; Maniatis, Petros; Jia, Limin; & Datta, Anupam. überspark: Enforcing verifiable object abstractions for automated compositional security analysis of a hypervisor. 87-104. Proc. of the 25th USENIX Security Symposium. August 2016.

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.

DM18-1101

About Us

The Software Engineering Institute is a federally funded research and development center (FFRDC) that works with defense and government organizations, industry, and academia to advance the state of the art in software engineering and cybersecurity to benefit the public interest. Part of Carnegie Mellon University, the SEI is a national resource in pioneering emerging technologies, cybersecurity, software acquisition, and software lifecycle assurance.

Contact Us

CARNEGIE MELLON UNIVERSITY
SOFTWARE ENGINEERING INSTITUTE
4500 FIFTH AVENUE, PITTSBURGH, PA 15213-2612

412.268.5800 | 888.201.4479
sei.cmu.edu | info@sei.cmu.edu