This booklet contains descriptions of 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.
Delivering Capabilities that Assure System Security and Performance

Gaining assurance (confidence and trust) in the behavior of large software-based systems requires understanding and overcoming the effects of software complexity and risk.

The Carnegie Mellon University (CMU) Software Engineering Institute (SEI) delivers capabilities to its sponsor, the U.S. Department of Defense (DoD), and other government agencies, which assure the security and performance of large-scale, complex, software-based systems.

SEI research and development (R&D) produces analysis, tools, techniques, prototypes, and practices that deliver confidence throughout a system’s life—from specifying cybersecurity and other requirements, to estimating cost and schedule in acquisition, to developing software functionality, and to evaluating and ensuring the performance of desirable behaviors during system operations (i.e., non-functional requirements such as reliability, sustainability, and availability).

In addition, as a federally funded research and development center (FFRDC), the SEI serves as a trusted and value-added broker of R&D by working with members of the software community in government, academia (in particular, CMU), and industry to customize, develop, and adapt software and cybersecurity technologies and related methods for the measurable benefit of the U.S. government.

With the access afforded by its DoD affiliation and a conflict-free status as an FFRDC, SEI has a unique ability to undertake technical work—line-funded research and sponsored engagements—ranging from fundamental research with widespread publication to support of sensitive government programs. We invite you to explore the details in this report of the 2016 SEI line-funded, fundamental research portfolio.
## Contents

Delivering Capabilities that Assure System Security and Performance | 1

### Assuring Critical Systems
- Incremental Lifecycle Assurance of Critical Systems | 4
- Automated Assurance of Security Policy Enforcement | 6
- Verifying Distributed Adaptive Real-Time (DART) Systems | 8
- Auto-Active Verification of Software with Timers and Clocks (STAC) | 10
- Property-Directed Test Generation | 12
- Using Technical Debt to Improve Software Sustainability and Find Software Vulnerabilities | 14
- Evaluation of Threat Modeling Methodologies | 16

### Assuring Missions
- Tactical Analytics | 20
- Semiconductor Foundry Verification | 22
- Tactical Computing and Communications | 24
- Enabling Evidence-Based Modernization | 26

### Assuring Software
- Vulnerability Discovery | 30
- Prioritizing Alerts from Static Analysis with Classification Models | 32
- Establishing Coding Requirements for Non-Safety-Critical C++ | 34
- Automated Code Repair | 36

### Assuring Autonomy and Human-Machine Interactions
- Why did the robot do that? Explaining Robot Behavior to Improve Trust in Autonomy | 40
- Human-Computer Decision Systems for Cybersecurity | 42
- Multi-Agent Decentralized Planning for Adversarial Robotic Teams (MADPARTS) | 44
- Statistical Model Checking of Swarm Algorithms | 46
- Experiences Developing an IBM Watson Cognitive Processing Application to Support Q&A of Application Security (Software Assurance) Diagnostics | 48
- GraphBLAS: A Programming Specification for Graph Analysis | 50
- The Critical Role of Positive, Intrinsic Incentives in Reducing Insider Threat | 52
- Workplace Violence/IT Sabotage: Two Sides of the Same Coin? | 54
- Data Validation for Large-Scale Analytics | 56
- Supporting Software Engineering Best Practices in Additive Manufacturing | 58

### Assuring Cyber Workforce Readiness
- Utilizing Serious Games to Assist Motivation and Education | 62
- Generalized Automated Cyber-Readiness Evaluation (ACE) | 64
Assuring Critical Systems
Incremental Lifecycle Assurance of Critical Systems

The current lifecycle practice of build-then-test for software-reliant (safety and mission) critical systems results in rapidly increasing verification-related rework costs, because 70% of defects are related to poor quality requirements and 80% of defects detected only after the unit test phase.

In this research, we produced a workbench of tools that demonstrate a measurable reduction in the cost of verifying system implementations against requirements, including:

- an Excel-based prototype implementation of Spotlight, which integrates requirement coverage, verification plan coverage, and multi-valued verification result metrics
- an Open Source AADL Tool Environment (OSATE) release that includes support for architecture-led requirement specification in ReqSpec (a textual requirement specification language)
Incremental Lifecycle Assurance of Critical Systems

Critical System Assurance Challenge
The traditional development lifecycle using existing methods of system engineering result in:
- High percentage of operator workarounds for software fixes
- Labor-intensive system safety analysis without addressing software as major hazard source
- High percentage of operator workarounds for software fixes due to high recertification cost

NIST Study
Current requirement engineering practice relies on stakeholders traceability and document reviews resulting in high rate of requirement change

Rolls Royce Study
Managed awareness of requirement uncertainty can lead to 50% reduction in requirement changes

U Minnesota Study
Requirements often span multiple architecture layers

Incremental Lifecycle Assurance Goals
- Improve requirement quality through coverage and managed uncertainty
- Improve evidence quality through compositional analytical verification
- Measurably reduce certification related rework cost through virtual integration and verification automation

Assurance & Qualification Improvement Strategy
- Measurement-driven Assurance Cost and Confidence Improvement
- Architecture-centric Virtual System Integration (ACVIP)
- Incremental Lifecycle Assurance (ALISA)

Microsoft Requirements for a Patient Therapy System
3. When piston stop is received, the system shall stop piston movement within 0.01 seconds.

<table>
<thead>
<tr>
<th>Textual Requirements for a Patient Therapy System</th>
<th>Incremental Improvement Thresholds</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Measurably reduce certification related rework cost through virtual integration and verification automation</td>
<td></td>
</tr>
</tbody>
</table>

Impact and Alignment
- AMRDEC Joint Multi-Role (JMR) Tech Demo: maturation of ACVIP for Future Vertical Lift (FVL)
- Aerospace industry System Architecture Virtual Integration (SAVI) multi-year initiative
- Standards: SAES 5/A (AADL Requirements, Constraints), SAE S18 (ARP4761 System Safety)
- Regulatory agencies: NRC, FDA, AAMI/UL
Automated Assurance of Security Policy Enforcement

Work on this project will span FY2016 and FY2017

As mission and safety-critical systems become increasingly connected, exposure due to security infractions is likewise increasing. This project aims at developing techniques to detect vulnerabilities early in the lifecycle in architecture models.

The SEI focuses on producing tools to reduce the cost of and improve the quality of system security assurance by

- detecting security policy violations early by verifying the enforcement of mission system security policies against a MILS-based (multiple independent levels of security) runtime architecture using a formalized set of consistency rules
- assuring that the system implementation enforces the policies by complementing model-based verification with system-level security tests that are generated from the architectural security policy specification
- assuring that no security risks are introduced by the runtime architecture decisions by reducing the attack surface in the runtime architecture
- improving the efficiency of the security assurance process by automating the execution of security assurance plans throughout the development lifecycle

The SEI has worked on techniques to auto-detect vulnerabilities in architectural models (developed using the Architecture Analysis and Design Language) and generate security reports such as Attack Impact or Attack Tree. Tools produced in this project have been released under an open-source license and are available on the SEI Github code repository (github.com/cmu-sei/AASPE).
Automated Assurance of Security Policy Enforcement
Detecting and fixing architecture-related vulnerabilities early in the lifecycle

Safety-critical systems are now extremely software-reliant, which increases their attack surface. In recent years, security vulnerabilities of critical systems have enabled threats on our lives. Our project uses architecture models to find security vulnerabilities early in the development lifecycle.

After the Jeep hack in 2015, Fiat-Chrysler issued a massive recall of 1.4 million cars. In the medical domain, the FDA advised hospitals to stop operating the Symbiq Infusion System due to potential tampering. With estimates targeting more than 20 billion connected devices by the end of 2020, the number of vulnerabilities, and their impact, will continue to grow. Vulnerabilities are no longer only a matter of code but strongly related to the system architecture.

The SEI team is working on solutions using the semantics of the Architecture Analysis & Design Language (AADL) and its extensions to detect vulnerabilities in software architectures. We are developing an AADL extension to capture security concerns in software architecture as well as new analysis tools that produce security reports from an AADL architecture.

What vulnerabilities can we detect?
The latest reports show that vulnerabilities are no longer related only to code (e.g., buffer overflow, semantic code) but are tightly coupled to the architecture: in component connections (e.g., use of encryption), shared resources (e.g., processing or memory), or configuration directives (e.g., use of encryption). We extended the AADL core language to provide the capability to detect common architecture-related vulnerabilities. With security expertise from the SEI CERT Division, we identified AADL modeling patterns for architecture-related vulnerabilities. We also identified patterns to capture and recognize Common Vulnerabilities and Exposures (CVE) in AADL architecture models.

A collaborative effort for safer systems
We initiated collaboration with the following projects or standardization bodies: SAE AS-2C: As the technical lead of the AADL standard, the SEI team collaborates with the standardization committee and will propose a new security annex for the standard.

The Open Group: The SEI is working with the Open Group and its Real-Time Embedded Systems Forum on a MILS standard for developing secure systems.

The MITRE CVE: With security knowledge from the CERT Division, the SEI team mapped architecture-related CVEs into AADL to detect security vulnerabilities in architecture models.

How are vulnerabilities reported?
The SEI research team developed AADL architecture analysis tools to detect vulnerabilities and show their impact. The tools currently generate two analysis reports from AADL models:

Attack Impact: This comprehensive report provides the architecture vulnerabilities for each component and shows how they are propagated using connections and shared resources. This analysis method is similar to Failure Modes and Effects Analysis.

Attack Tree: A hierarchical tree represents the relationships between contributors (architecture elements and vulnerabilities) of a compromised component. This analysis method is similar to Fault-Tree Analysis.

These tools are integrated with the AADL modeling tool OSATE and are available under the open source Eclipse Public License for download.

Making an impact
We have demonstrated our approach through case studies from the automotive and avionics domains. We retro-engineered automotive architectures to show how our approach and tools can detect security issues such as the one reported in the Jeep hack. For the avionics domain, we demonstrated our approach in the System Architecture Virtual Integration (SAVI) consortium and showed how attacks against the Automotive Dependent Surveillance-Broadcast (ADS-B) protocol could impact airplanes and ground station security.
Verifying Distributed Adaptive Real-Time (DART) Systems

Work on this project spanned FY2015 and FY2016

DART systems (such as autonomous multi-unmanned-air-system missions) are key to Department of Defense (DoD) capability. However, verifying DART systems has proven to be intractable.

In response, we have developed and validated assurance techniques for DART systems. We created these techniques through

• a new domain-specific language, called DMPL, to program DART systems. DMPL has been integrated with the Architecture Analysis and Design Language standard as an annex.

• new temporal isolation mechanisms to protect high-critical threads from low-critical ones across multiple processors

• new compositional model checking algorithms to verify high-critical properties of distributed software

• new proactive self-adaptation approaches to achieve low-critical properties under uncertainty—assuring them via statistical model checking

Principal Investigators

Dr. Sagar Chaki  
Principal Researcher  
Lead, Cyber-Physical and ULS Systems Initiative

For more information: sei.cmu.edu/about/people/profile.cfm?id=chaki_13495

Dr. Dionisio de Niz  
Principal Researcher  
Deputy Lead, Cyber-Physical and ULS Systems Initiative
Verifying Distributed Adaptive Real (DART) Systems

**Pipelined ZSRM Scheduling**
- Reduces pipeline over single-resource scheduling
- Avoids assuming worst alignment in all stages
But need to deal with transitive interferences due to zero-slack
Ongoing work: theory worked out, implementing scheduler in Linux

**Functional Verification**
Prove application-controller contract for unbounded time
- Previously limited to bounded verification only
Prove controller-platform contract via hybrid reachability analysis
- Done by AFRL
Working on automation and analysis
- Reduces pipeline

**DART Vision**
A sound engineering approach based on the judicious use of precise semantics, formal analysis and design constraints leads to assured behavior of (DART) systems while accounting for:
- Required guarantees
- Probabilistic requirements
- Uncertain environments
- Necessary coordination
- Assurance at source code level

**DART Process**
1. Define computational and requirement-specific verification.
2. Use proactive self-adaptation and mixed criticality to cope with uncertainty and changing context.
3. Monitor for runtime assurance.

**DMPL: DART Modeling and Programming Language**
- C-like language that can express distributed, real-time systems
- Supports formal assertions usable for model checking and probabilistic model checking
- Physical and logical concurrency can be expressed in sufficient detail to perform timing analysis
- Can call external libraries
- Generates compileable C++
- Developed syntax, semantics, and compiler (dmplc)

**Example: Self-Adaptive and Coordinated UAS Protection**
- Adapts to changing context
- High-Critical Threads (HCTs)
- Low-Critical Threads (LCTs)
- Actuators
- Sensors & Environment

**Distributed Statistical Model Checking**
Batch Log and Analyze

<table>
<thead>
<tr>
<th>SMC Client</th>
<th>SMC Aggregator</th>
</tr>
</thead>
<tbody>
<tr>
<td>log client</td>
<td>log analyzer</td>
</tr>
</tbody>
</table>

Result:
- Decide if adaptation is acceptable?
- Update Result and RE

Future Work: Importance sampling to reduce number of simulations needed for “rare” events.
Each run of log-generator and log-analyzer occurs on a VM. Multiple VMs run in parallel on HPC platform. Clients added and removed on-the-fly.
Auto-Active Verification of Software with Timers and Clocks (STAC)

The inability to assure STACs at the source code level cost-effectively impedes their certification and adoption.

The project produced

- formal clocked semantics of STACs
- verification condition (VC) generation algorithm for sequential and distributed STACs
- prototype auto-active verifier for STACs
- evaluation of the tool on an implementation of the zero-slack rate monotonic (ZSRM) scheduler as a Linux kernel module that uses timers and clocks to enforce thread CPU budgets and mixed-criticality scheduling guarantees

Principal Investigators

Dr. Sagar Chaki
Principal Researcher
Lead, Cyber-Physical and ULS Systems Initiative

Dr. Dionisio de Niz
Principal Researcher
Deputy Lead, Cyber-Physical and ULS Systems Initiative

For more information:
sei.cmu.edu/about/people/profile.cfm?id=chaki_13495
Auto-Active Verification of Software with Timers and Clocks

Motivation

STAC = software that accesses the system clock, exchanges clock values, and uses these values to set timers and perform computation

- Key to real-time and cyber-physical systems
- Essential to keep software in sync with the physical world
- Examples = thread schedulers and time budget enforcers, distributed protocols (e.g., plug-and-play medical devices)

Goal: Formally verify STACs at the source code level using deductive (aka auto-active) verification

- Target: ZSRM mixed-criticality scheduler
  - Performs thread CPU allocation and time budget enforcement
  - Available as Linux kernel module implemented in C
  - Currently we focus on ZSRM budget enforcement only

Execution & Thread CPU Usage

Time is Global "Newtonian" clock

- Flows monotonically, dense real-time

\[ C(t, t) = \text{total cpu usage by thread } t \text{ over execution } \pi \]
- Add up durations of all the transitions labeled by \( t \)

\[ C(\pi, t_1) = C(t) - C(t_1) \]
\[ C(\pi, t_2) = C(t) - C(t_2) \]
\[ C(\pi, t_3) = C(t) - C(t_3) \]

\( C(\pi, t) \) can never be measured precisely

But can be over-approximated!

Verifying \( \text{Timer}(\tau) \) on source code

Started with ZSRM implementation as Linux kernel module

Expressed \( \text{Timer}(\tau) \) as ACU annotations and verified with Frama-C

Why use Auto-Active Verification?

Soundness

Language expressivity

- Pointers, recursion, loops

Rich specification

- Quantifiers
- Separation

Tool maturity

- Frama-C
  - Multiple backend SMT solvers

Why Verify Source Code?

Push assurance closer to executable level

- Use verified compilers (e.g., CompCERT) to close the final gap
  - Don’t need to sacrifice performance

- This is a problem when we verify models
- And is a no-go for low-level system software

Easier to integrate with existing systems

- Linux kernel module means anyone using Linux can use it
- Can be modified to work with other OSes, such as SEL4

- What You Verify Is What You Execute!

Technical results

Theorem 1. For any execution \( \pi = \alpha_1 \cdots \alpha_{n-1} \alpha_n \), and thread \( \tau \), the following four conditions hold:

\( C(\pi, t) \leq C(\pi, t) \text{ for all } t \in \pi \)

We assume that \( \text{now()} \) returns a value that is within the time boundary of the transition in which it is executed

Measuring current time

System calls and timer handlers use a special function \( \text{now()} \) to measure current time

We assume that \( \text{now()} \) returns a value that is within the time boundary of the transition in which it is executed

\[ a_2 \leq \text{now}() < a_3 \]

\[ b_2 \leq \text{now}() < b_3 \]

We assume that multiple calls to \( \text{now}() \) return strictly increasing values

- Implemented using hardware timestamp counter

Complete source code with ACSL annotations publicly available

- https://github.com/cps-sei/stac
- Compiles on recent Linux distributions
- Tested to demonstrate good performance

- Verifiers with Frama-C Aluminium

To our knowledge, the first formally verified and performant timing enforcer

Results extended to periodic threads as well
Property-Directed Test Generation
We are developing an automated, property-directed, executable, test-case-generation technique that combines the strengths of software model checking and symbolic execution. Our tool takes a declarative description of a behavior (e.g., an execution with a buffer overflow, an execution reaching a dangerous function call, or leaking sensitive information through a low-security interface) and automatically generates an executable test harness that executes it.

Dr. Edward Schwartz
Research Scientist
Vulnerability Analysis Team,
Threat and Vulnerability Analysis Initiative
Property Directed Test-case Generation

Manually finding inputs to trigger a behavior of interest in a program is complex and time consuming. In this project, we repurpose existing formal methods techniques to help automate this problem. We use counter examples produced by SEI’s Seahorn model checker to create executable harnesses that demonstrate how the behavior of interest can be reached.

Verifying Linux Device Drivers
A common problem when model checking software is understanding the results that the model checker yields. For example, a small discrepancy in modeling can result in a complicated counter-example that is difficult to understand. We applied PDTG to model checking instances of Linux Device Drivers where the model check failed, and automatically produced an executable harness that showed the problematic execution. The final harness can be executed in a debugger and reviewed step by step, which makes correcting the problem much easier.

Reverse-engineering Malware
We also used PDTG to assist in reverse-engineering malware. We start with a sequence of API calls that may indicate malicious or interesting behavior. For example, enumerating processes on Windows requires calls to CreateToolhelp32-Snapshot, Process32First, and Process32-Next in sequence. PDTG can construct a harness that forces the program to execute these calls, and thus display the malicious behavior for an analyst. We tested this technique on the Gh0st RAT variant.

<table>
<thead>
<tr>
<th>Property Directed Test-case Generation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Manually finding inputs to trigger a behavior of interest in a program is complex and time consuming. In this project, we repurpose existing formal methods techniques to help automate this problem. We use counter examples produced by SEI’s Seahorn model checker to create executable harnesses that demonstrate how the behavior of interest can be reached.</td>
</tr>
</tbody>
</table>

**Verifying Linux Device Drivers**
A common problem when model checking software is understanding the results that the model checker yields. For example, a small discrepancy in modeling can result in a complicated counter-example that is difficult to understand. We applied PDTG to model checking instances of Linux Device Drivers where the model check failed, and automatically produced an executable harness that showed the problematic execution. The final harness can be executed in a debugger and reviewed step by step, which makes correcting the problem much easier.

**Reverse-engineering Malware**
We also used PDTG to assist in reverse-engineering malware. We start with a sequence of API calls that may indicate malicious or interesting behavior. For example, enumerating processes on Windows requires calls to CreateToolhelp32-Snapshot, Process32First, and Process32-Next in sequence. PDTG can construct a harness that forces the program to execute these calls, and thus display the malicious behavior for an analyst. We tested this technique on the Gh0st RAT variant.
Using Technical Debt to Improve Software Sustainability and Find Software Vulnerabilities

Technical debt is a metaphor that conceptualizes the tradeoff between short-term and long-term value. Managing technical debt is an increasingly critical aspect of producing cost-effective, timely, and high-quality software products.

Improving Software Sustainability through Data-Driven Technical Debt Management

Work on this project spanned FY2015 and FY2016.

Budget constraints and the need to accelerate capability delivery have resulted in the DoD’s adoption of incremental system development approaches and a shift from new system acquisition to more cost-effective system evolution and sustainment of existing systems. We developed a suite of tools and techniques that detect technical debt and analyze its causes and effects.

Finding Software Vulnerabilities Early by Correlating with Technical Debt

Our technical results for this project include a dataset correlating relationships between vulnerabilities and known sources of security-related technical debt such as design flaws.
Using Technical Debt to Improve Software Sustainability and Find Software Vulnerabilities

**Technical debt** is a term that 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 an effort to manage budget constraints the DoD is increasingly searching for tool-supported approaches to manage technical debt. The goal of this project is to develop a suite of tools and techniques for detecting and visualizing technical debt and provide exemplar data sets.

**Our approach included:**
1. **Codify known architectural sources of technical debt that are not addressed adequately by today’s code-oriented tools** (e.g., safety-critical testing partitioning, unbalanced modules, dependency violations)
2. **Identify architecture indicators through abstractions (e.g., interfaces, restrict compositional dependencies) and anti-patterns that are correlated with technical debt, and that can be automatically identified by analyzing source code and other project artifacts.**
3. **Integrate these architectural indicators with code indicators in an experimental workbench.**
4. **Conduct empirical studies over multiple releases of at least two systems to correlate the identified indicators with observable project measures such as cost to fix, cost to implement new features, and defects.**

**Technical debt analytics vision and the timeline:**
1: time technical debt is incurred; 2: time technical debt is recognized; 3: time to plan and re-architect; 4: time until debt is actually paid-off

**Finding:** Tagging technical debt explicitly in issue trackers improves its management.

**Crash:** WebCore: TransparencyWin::initializeNewContext()

*“We could just fix off negative numbers near the crash site or we can dig deeper and find out how this -10000 is happening.”*

*“Time permitting, I’m inclined to want to know the root cause. My sense is that if we patch it here, it will pop-up somewhere else later.”*

*“There have been 28 reports from 7 clients... 18 reports from 6 clients.”*

*“Hmm... reopening. The test case crashes a debug build, but not the production build. I have confirmed that the original source code does crash the production build, so there must be multiple things going on here.”*

**Finding:** Correlations between vulnerabilities and technical debt demonstrate areas of key improvement.

<table>
<thead>
<tr>
<th># Types of Design Flaws</th>
<th>Non-val files</th>
<th>Val files</th>
<th>% have val.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>8544</td>
<td>47</td>
<td>0.5%</td>
</tr>
<tr>
<td>2</td>
<td>1247</td>
<td>91</td>
<td>4%</td>
</tr>
<tr>
<td>3</td>
<td>194</td>
<td>10</td>
<td>5%</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>0</td>
<td>0%</td>
</tr>
</tbody>
</table>

**Technical debt:**
- Exists in an executable system artifact, such as code, build scripts, data model, automated test suites;
- Is traced to several locations in the system, implying issues are not isolated but propagate throughout the system;
- Has a quantifiable effect on system attributes of interest to developers.

**The technical debt metaphor** is widely used to encapsulate numerous software quality problems. In a survey of 1831 participants, primarily software engineers and architects working in long-lived, software-intensive projects from three large organizations, we found that architectural decisions are the most important source of technical debt. Our research has shown that technical debt detection improves when source code analysis is complimented with an architecture focus.

The SEI Architecture Practices team has been a pioneer in advancing the research agenda in analyzing technical debt. Our ongoing work is focused on combining multiple artifacts, such as source code, issue trackers, commit histories and augmenting analysis with machine learning driven approaches to locate and manage technical debt. You can engage with us by:
- collaborating on an in-depth analysis of your project and sharing your data
- contributing your technical debt examples
Evaluation of Threat Modeling Methodologies

Failure to sufficiently identify computer security threats leads to missing security requirements and poor architectural decisions, resulting in vulnerabilities in cyber and cyber-physical systems. This research compares practical threat modeling methods (TMMs) that proactively identify cyber-threats, leading to software requirements and architectural decisions that address the needs of the DoD. The primary result of this project is a set of tested principles that can help programs select the most appropriate TMMs. Using the most appropriate TMMs will result in confidence in the cyber-threats identified, accompanied by evidence of the conditions under which the TMMs are most effective.
Evaluation of Threat Modeling Methodologies

Motivation
Failure to sufficiently identify computer security threats leads to missing security requirements and poor architectural decisions, resulting in vulnerabilities in cyber and cyber-physical systems.

This research compares 3 practical threat modeling methods (TMMs) that pro-actively identify cyber-threats, leading to software requirements and architectural decisions that address the needs of the DoD. Its primary result is a set of tested principles which can help programs select the most appropriate TMMs, accompanied by evidence of the conditions under which each technique is most effective. These principles can be applied to better assess the confidence that can be had in cyber threat analysis.

The Study
Evaluate three exemplar Threat Modeling Methods, designed on different principles, to understand strengths and weaknesses of each.

Results
We identified characteristic differences among the TMMs that affect the confidence to be had in their application on programs. Our data show substantial tradeoffs among threat types detected, number of threats missed, and number of potential false positives reported—and that no one TMM optimizes on all dimensions.

Future Work: Creating a training course of tested threat modeling principles & practices. Looking for transition partners for case studies on DoD programs.

Long term: Our vision is to support dynamic threat models that can trace changes in the threat environment to needed impacts on system requirements, design, and code.

“...engineers have not had sufficient training nor been encouraged to have a mind-set that considers how an adversary might thwart their system... the R&D community has not given engineers the tools they need.”

—Greg Shannon, SEI/CERT Chief Scientist
IEEE Institute, March 2015

Key results:
- STRIDE: Greatest variability in terms of how frequently it leads to types of threats.
- Security Cards: Able to find the most threat types but also substantial variability across teams.
- PnG: Was the most focused TMM (teams found only a subset of threat types), but showed the most consistent behavior across teams.

Union of Threat Types

<table>
<thead>
<tr>
<th>TYPES DETECTED</th>
<th>TYPES UNDETECTED</th>
<th>NOT IN REFERENCE LIST</th>
</tr>
</thead>
<tbody>
<tr>
<td>STRIDE (13 teams)</td>
<td>Sec.Cards (23 teams)</td>
<td>PnG (17 teams)</td>
</tr>
</tbody>
</table>

Resources:
OSD(AT&L) Working Group on Cyber Threat Modeling brings together practitioners and researchers for quarterly meetings. Ask for details.
Architecture Modeling Helps Joint Multi-Role (JMR) Effort

The SEI with collaborator Adventium Labs used Architecture-Centric Virtual Integration Practice (ACVIP) to discover potential software and system integration issues early in the development process. The JMR program manager recommended that contractors use this technology in next-phase demonstrations.
Assuring Missions
Tactical Analytics

Work on these projects spanned FY2015 and FY2016

This work encompasses two projects: Tactical Analytics and Structural Multi-Task Transfer Learning for Improved Situational Awareness. In general, this work supports analysis of data in timeframes sufficient for tactical planning (i.e., typically, less than 72 hours prior to the mission) and during tactical operations (i.e., analysis of data gathered from data streams during the execution of the mission).

In these projects, we developed

- prototypes to demonstrate new capabilities for script learning (i.e., patterns of life) and credibility scoring for social media
- generalized machine-learning techniques for data classification and exploration that enable analysts to understand emerging situations quickly

Edwin Morris
Senior Member of the Technical Staff
Lead, Advanced Mobile Systems Initiative

For more information:
sei.cmu.edu/about/people/profile.cfm?id=morris_13107
Tactical Analytics
Recognizing Patterns of Life and Determining Credibility of Textual Data

Structural Multi-Task Transfer Learning
To support analysis of real-time streaming data for situational awareness, we created methods for recognition patterns in textual data and determining credibility of textual data.

Patterns of Life: To recognize patterns of life in textual data we use the concept of “scripts”. A script is a sequence of ordered, related events that describe a stereotypical pattern that adversaries follow during military and other activities. Scripts allow analysts to recognize these patterns and make predictions about emerging events. This year’s work was focused on automatically identifying scripts from streaming data, accounting for multiple pathways through the script.

Comparing Sequence Z against Script X:

1. Scripts can be learned from streaming data
2. Constraints are necessary to avoid obviously invalid pathways
3. Even a simple test case is very complicated

Measures of Similarity

\[ s(X,Z) = 1 - \arg \min_{\alpha, \beta} \sum_{x \in X} \delta(x,Z) \]

\[ \delta(x,Z) = \begin{cases} 0 & \text{if a match B.T.} \\ \alpha & \text{if insertion} \\ \beta & \text{if deletion} \end{cases} \]

\[ \beta = 1 \text{ unless specified otherwise by the user} \]

Challenges:
1. State-of-the-art event recognition algorithms proved insufficient for our task. Solution: We used data from baseball box scores that allowed easy event extraction. FY17 work will extend DARPA algorithms for single & multiple sentence event recognition. Script recognition will ultimately require recognizing events across multiple dissimilar documents.
2. Establishing event relationships must be improved. Solution: FY16 work involved creating constraints for order and uniqueness. FY17 work will extend this work.

Future Work: We need to remove more noise from the social media data in step #1 of the analytics pipeline. Step #2 must be improved to generalize to more event types. Step #3 requires external sources to improve the credibility assessment of the entities providing information.

Lessons Learned:
1. Stance determination is essential
2. Noise is difficult to filter; we need accurate event recognition

Determining Credibility Scores of Streaming Social Media Data

Credibility Analytics Pipeline:

- Data Stream
- Filter Stream
- Identify Author’s Stance
- Tweet + Stance
- Calculate Aggregate Score
- Credibility Score + Chain of Reasoning

This work depends on accurate event detection. As a proxy, we used 3 celebrity death events and 80 diverse events from Twitter.*

Twitter Events

- True: Castaic Earthquake
- False: Whitney Houston Died
- False: Paul McCartney Died

Rumor: Obama Bans Sprinkles

Future Work: We need to remove more noise from the social media data in step #1 of the analytics pipeline. Step #2 must be improved to generalize to more event types. Step #3 requires external sources to improve the credibility assessment of the entities providing information.

Semiconductor Foundry Verification

Unknown and counterfeit electronic components pose risk to secure operations in critical infrastructure systems. The project produced methodology to verify the history of chip design and manufacturing. The results of this research can substantially cut the effort required to validate a supply chain.

Dr. Alexander Volynkin
Research Scientist
Forensic Operations and Investigations Team, Monitoring and Response Initiative
Semiconductor Foundry Verification
Detecting Counterfeit Electronics

Motivation
- Project aims at verifying history of chip design and manufacturing used in critical infrastructure.
- Unknown electronic components possess risk to secure operations.
- Analysis is done at the integrated circuit (IC) level. Verified information includes foundry info, design specifics, sources of 3rd party circuitry.
- Algorithms detect attribution with minimal human intervention.

Research Goals
- Well-established algorithmic approach to circuit component recognition based on behavioral matching of an unknown sub-circuit against a library of abstract components
- Leverage available component/foundry information to study the attribution impact and extract samples of sub-circuits.
- Measure logic gate density, metal layer routing, collections of logic gates.
- Analyze numerous different ICs for differentiating factors.
- Verify results on another relatively large set of various ICs.

Main Idea
- Semi-automated image processing to detect chip features
- Each layer is photographed and processed
- Relevant features extracted and checked against rules
- Fabrication facilities have design and fabrication requirements and tolerances

Experimental Results

Counterfeit Example. These two chips appear to be identical. The one on the left is counterfeit, the one on the right is authentic.

Integrated Circuit Fabrication
- Doping agents, glasses, or metals on silicon
- Individual components nowadays are on the order of 100nm~10nm
- Chips are multi-layered - Bottom layer is transistors, other silicon features
- Layers above alternate:
  - Metal interconnects (copper/aluminum)
  - Vias (same material as metal)
  - Glass (Silicon Dioxide) between all of this, isolating the layers
- Topmost layer contains pads for connecting to packaging and an encapsulation layer

Project Outcomes

Automated Analysis Framework. Square Area Density Based Spatial Cluster Analysis with Noise (SADBCAAN)

- Method of cluster analysis specifically designed for segmentation and area differentiation in images
- Weights the geographical difference as more important and mark these objects as different clusters
- Queries different regions separately and efficiently
- Calculates simple Euclidian distance of color values
- Combines clusters of pixels based not only on color similarities but also the "geographic" location
- Accurate feature detection with high speed parallel processing (10-15 minutes on 1GB image)
- Various additional analytical image processing and feature extraction methods implemented in plugins
Tactical Computing and Communications

Work on this project spanned FY2015 and FY2016

This project worked toward a goal of developing architectures and technologies to provide efficient and secure computing and communications for teams operating in tactical environments, in particular

- **Trusted Identities in Disconnected Environments** for securing communication between mobile devices and cloudlets operating in tactical environments
- **Secure VM Migration** for enabling secure migration of capabilities between cloudlets in tactical environments
- **Delay-Tolerant Data Sharing** for efficient information sharing between nodes in tactical (DIL) environments

Results of this work include reference architectures, demos, prototypes, and source code that validate and incorporate research results. Code for tactical cloudlets is available as open source at https://github.com/SEI-AMS/pycloud.
Tactical Computing and Communications (TCC)
Secure and Efficient Computing and Communications at the Edge

Previous Work
Tactical Cloudlets

Features:
- Pre-Provisioned Cloudlets w/ App Store
- Standard Packaging of Service VMs
- Optimal Cloudlet Selection
- Cloudlet Management Console
- Cloudlet Handoff/Migration
- Secure Key Generation and Exchange

Device Credential Revocation
- Automatic due to timeout: Bootstrapping requires setting up mission length
- Manual due to known loss or compromise: Cloudlet Manager component has revocation option

Secure Service VM Migration

Device Credential Generation
- Cloudlet A discovers and connects to Cloudlet B using exchanged credentials
- Cloudlet B generates new credentials for Device
- Cloudlet B sends credentials to Device via Cloudlet A

Device Connection
- Device connects to Cloudlet B using new credentials
- Client App on Device connects to Service VM running on Cloudlet B

Delay-Tolerant Networking (DTN)

Metadata
- Time and location
- Priority
- Type of payload (image, voice, video, text, …)
- Set of tags describing payload content (building, crowd, fire, injured person, …)

Extensions to the existing DTN standard for priorities, staleness, replacement, and redundancy monitoring to increase bandwidth efficiency in DIL environments

Step 1: Bootstrapping
- Generation of Server Credentials usingIBE (Identity-Based Encryption)
- Setup of RADIUS Server with Server Credentials

Step 2: Pairing
- Generation of Device Credentials usingIBE
- Transfer to device using Bluetooth or USB, plus visual confirmation
- Transfer to RADIUS Server

Delay-Tolerant Data Sharing

Subscription
- User B subscribes to the Content Delivery Network (CDN) for files tagged “map” or “IED”
- CDN uses RSS to discover files

Publication
- User A sends file X tagged with “map” to the CDN
- CDN uses RSS to publish files

File Download using BitTorrent
- CDN Node 2 downloads file X using BitTorrent
- CDN Node 2 pushes file X to User B’s device

Device connects to the migrated Service VM on Cloudlet B

Copyright © 2016 Software Engineering Institute

Contact: Grace Lewis | glewis@sei.cmu.edu

Delay-Tolerant Networking (DTN)

Maintain shared group context
Make best use of available bandwidth
Pre-fetch data likely to be relevant later in the mission
Delay transmission of non-critical data

Applications continue to function
Predict state where possible
Predict location of teams based on mission plan
Provide connectivity map to help the user reconnect

Prioritize synchronization of critical messages
Eliminate redundant messages

Device connects to router, single hop network, multiple devices
Forward-deployed, discoverable, virtual machine (VM) based cloudlets that can be hosted on vehicles or other platforms
- computation offload
- forward data-staging
- filtering of data intended for mobile devices
- collection points for data heading for enterprise repositories
Enabling Evidence-Based Modernization

Business system modernization continues to be problematic for the DoD. It appears on the General Accounting Office High Risk List again in 2015. The project is producing a prototype of a decision support tool that incorporates stakeholder solution preferences and analyzes the alternative decisions to find solutions that best meet the preferences.

John Klein
Senior Member of the Technical Staff
Architecture Practices Initiative

For more information:
sei.cmu.edu/about/people/profile.cfm?id=klein_14435
Enabling Evidence-Based Modernization (EEBM)

The GAO reports that most DoD business system modernization projects fail to establish a baseline within 2 years. These are not unprecedented systems – viable solutions exist, but choosing a solution involves stakeholders agreeing about the architecture approach and delivery sequence. We’ve found that in many cases, only a few decisions affect the solution cost and benefit, and we have developed a method and tool to help find those decisions that matter.

**Softgoal Modeling** is a lightweight approach to capture the structure of the decisions to be made as a network. Softgoals allow representation of subjective, qualitative desires about the system.

**Analytic Hierarchy Process (AHP)** collects stakeholder preferences about the softgoal decisions. AHP is time-efficient for stakeholders, using pairwise comparisons to rank alternatives.

**LOOPHOLE** is a search-based tool that uses differential evolution to efficiently find optimal solutions – the combinations of decisions that best satisfy preferences and other constraints. LOOPHOLE then uses Bayesian inference to identify the decisions that contribute to the best solutions – the **Key Decisions** that have the most influence over the quality of the solution.

This approach scales to large decision models, and is fast enough to provide real-time collaboration support. By focusing on the decisions that matter, programs can focus attention, establish baselines, and make faster progress.

**Softgoal Model**

**Decision Structure**
- (Softgoal Model)

**Decision Cost/Benefit**
- (Existing Technology)

**Stakeholder Preferences**
- (Analytical Hierarchy Process)

**KEY DECISIONS**
- Identify the decisions that contribute to the best solutions – the combinations of decisions that best satisfy preferences and constraints.

**LOOPHOLE**
- Uses Bayesian inference to identify the decisions that contribute to the best solutions.
- Key Decisions (the ones that matter)

**Analytic Hierarchy Process (AHP)**
- Ranking by pairwise comparisons

** LOOPHOLE Performance and Scalability**
- **Model**
- **Nodes**
- **Edges**
- **Runtime(s)**

<table>
<thead>
<tr>
<th>Model</th>
<th>Nodes</th>
<th>Edges</th>
<th>Runtime(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>XXXX</td>
<td>114</td>
<td>267</td>
<td>0.099</td>
</tr>
<tr>
<td>XX</td>
<td>101</td>
<td>239</td>
<td>0.096</td>
</tr>
<tr>
<td>XXX</td>
<td>86</td>
<td>216</td>
<td>0.095</td>
</tr>
<tr>
<td>XXXXX</td>
<td>82</td>
<td>214</td>
<td>0.094</td>
</tr>
<tr>
<td>XXXXXX</td>
<td>78</td>
<td>210</td>
<td>0.094</td>
</tr>
<tr>
<td>XXXXXXX</td>
<td>74</td>
<td>206</td>
<td>0.092</td>
</tr>
<tr>
<td>XXXXXXXX</td>
<td>69</td>
<td>202</td>
<td>0.091</td>
</tr>
<tr>
<td>XXXXXXXXX</td>
<td>65</td>
<td>198</td>
<td>0.093</td>
</tr>
<tr>
<td>XXXXXXXXXX</td>
<td>60</td>
<td>194</td>
<td>0.093</td>
</tr>
<tr>
<td>XXXXXXXXX</td>
<td>55</td>
<td>190</td>
<td>0.092</td>
</tr>
<tr>
<td>XXXXXXXXXX</td>
<td>50</td>
<td>186</td>
<td>0.092</td>
</tr>
<tr>
<td>XXXXXXXXX</td>
<td>45</td>
<td>182</td>
<td>0.091</td>
</tr>
<tr>
<td>XXXXXXXXXX</td>
<td>40</td>
<td>178</td>
<td>0.091</td>
</tr>
<tr>
<td>XXXXXXXXX</td>
<td>35</td>
<td>174</td>
<td>0.090</td>
</tr>
<tr>
<td>XXXXXXXXXX</td>
<td>30</td>
<td>170</td>
<td>0.090</td>
</tr>
<tr>
<td>XXXXXXXXX</td>
<td>25</td>
<td>166</td>
<td>0.089</td>
</tr>
<tr>
<td>XXXXXXXXXX</td>
<td>20</td>
<td>162</td>
<td>0.089</td>
</tr>
<tr>
<td>XXXXXXXXX</td>
<td>15</td>
<td>158</td>
<td>0.088</td>
</tr>
<tr>
<td>XXXXXXXXXX</td>
<td>10</td>
<td>154</td>
<td>0.088</td>
</tr>
<tr>
<td>XXXXXXXXX</td>
<td>5</td>
<td>150</td>
<td>0.087</td>
</tr>
</tbody>
</table>

**Support**
- Note: Support varies depending on collaboration baseline reading
Providing Computation and Data at the Tactical Edge

The SEI developed KD-Cloudlet, a software solution that enables the quick deployment of tactical cloudlets—forward-deployed, discoverable, virtual-machine-based cloudlets that can be hosted on vehicles or other platforms and provide secure computation offload and data staging capabilities for soldiers in the field. KD-Cloudlet is available on GitHub.
Vulnerability Discovery

Work on this project spanned FY2015 and FY2016

Vulnerabilities are pervasive in software-based systems, both in traditional IT networks and networks that support critical U.S. infrastructure.

In this project, we focused on automated and sound vulnerability discovery and prioritization in both traditional and non-traditional (i.e., mobile) computing platforms and on vulnerability discovery and correlation in emerging networked technologies. Our results include prototype tools for

- uniqueness determination, to show which vulnerabilities are triggered by a crashing test case
- the automatic discovery of vulnerabilities in binary programs by combining mutational fuzzing and concolic execution
Vulnerability Discovery

Current vulnerability discovery techniques such as black-box fuzz testing and concolic testing are so effective that they routinely find hundreds of thousands of crashers, which crash the target program. We created a new methodology for precisely and naturally defining vulnerabilities through the creation of patches. We use our methodology to debunk three commonly held beliefs in fuzzing practice.

Experiment setup.

We fuzzed Flasm, ImageMagick, Jasper, and OpenJpeg for a week under various configurations, which yielded hundreds of thousands of crashes. We patched each crash using our methodology, which yielded vulnerabilities for each program. We used this data to debunk the following beliefs shown on the right:

<table>
<thead>
<tr>
<th>Program</th>
<th># Vuls</th>
<th>UC</th>
<th>% Error</th>
<th>OC</th>
<th>% Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flasm</td>
<td>6</td>
<td>1.8</td>
<td>29%</td>
<td>410.9</td>
<td>6,848%</td>
</tr>
<tr>
<td>ImageMagick</td>
<td>31</td>
<td>1.9</td>
<td>6%</td>
<td>67.9</td>
<td>219%</td>
</tr>
<tr>
<td>Jasper</td>
<td>12</td>
<td>0.0</td>
<td>0%</td>
<td>226.4</td>
<td>1,887%</td>
</tr>
<tr>
<td>OpenJpeg</td>
<td>36</td>
<td>0.1</td>
<td>0%</td>
<td>267.5</td>
<td>743%</td>
</tr>
</tbody>
</table>

Misbelief 1: Stack backtrace hashing always accurately counts vulnerabilities

- # Vuls: Number of vulnerabilities as counted by our methodology
- UC (Undercount): Average number of vuls missed due to stack backtrace hashing
- OC (Overcount): Average number of vuls counted more than once by stack backtrace hashings

Misbelief 2: Sanitization never harms fuzzing performance

Misbelief 3: The AFL fuzzer always finds more vulnerabilities than non-guided fuzzers
Prioritizing Alerts from Static Analysis with Classification Models

Triaging the number of alerts about possible security-related code flaws detected by static analysis currently requires an unacceptable level of manual effort.

The project created alert classification models using features derived from multiple static analysis tools, code base metrics, and archived audit determinations. The results are accurate predictors of alert validity, intended for use in automatic prioritization of alerts from static analysis tools that minimizes the number of alerts needing human assessment.
Prioritizing Alerts from Static Analysis with Classification Models

Problem
The number of security-related code flaws detected by static analysis requires too much effort to triage.

Significance
- Code flaws and vulnerabilities remain
- Scarce resources are used inefficiently

Project goals
Classification algorithm development using CERT- and collaborator-audited data, to accurately estimate the probability of true & false positives, intended to reduce analyst effort.

Scientific Approach
Novel combined use of:
1) multiple analyzers, 2) variety of features, 3) competing classification techniques!

Data Used for Classifiers
Data used to create and validate classifiers:
- CERT-audited alerts:
  - ~7,500 audited alerts
- 3 DoD collaborators audit their own codebases with enhanced-SCALE

We pooled data (CERT + collaborators) and segmented it:
- Segment 1 (70% of data): train model
- Segment 2 (30% of data): testing

Added classifier variations on dataset:
- Per-rule
- Per-language
- With/without tools
- Others

CERT-audited data

<table>
<thead>
<tr>
<th>Rule ID</th>
<th>Lasso LR</th>
<th>Random Forest</th>
<th>CART</th>
<th>XGBoost</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT31C</td>
<td>98%</td>
<td>97%</td>
<td>98%</td>
<td>97%</td>
</tr>
<tr>
<td>EXP01J</td>
<td>74%</td>
<td>74%</td>
<td>81%</td>
<td>74%</td>
</tr>
<tr>
<td>OBJ03J</td>
<td>73%</td>
<td>86%</td>
<td>86%</td>
<td>83%</td>
</tr>
<tr>
<td>TID04J</td>
<td>80%</td>
<td>80%</td>
<td>90%</td>
<td>80%</td>
</tr>
<tr>
<td>EXP3C</td>
<td>83%</td>
<td>87%</td>
<td>83%</td>
<td>83%</td>
</tr>
<tr>
<td>EXP34C*</td>
<td>67%</td>
<td>72%</td>
<td>79%</td>
<td>72%</td>
</tr>
<tr>
<td>ERR08J*</td>
<td>99%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>IOS00-J*</td>
<td>96%</td>
<td>96%</td>
<td>96%</td>
<td>96%</td>
</tr>
<tr>
<td>ERR01J*</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>ERR09-J*</td>
<td>100%</td>
<td>88%</td>
<td>88%</td>
<td>88%</td>
</tr>
</tbody>
</table>

*Single-rule IDs with asterisk: small quantity of data, results suspect

General results (not true for every test)
- Classifier accuracy rankings for all-pooled test data: XGBoost ≈ RF > CART > LR
- Classifier accuracy rankings for collaborator test data: LR > RF > XGBoost > CART
- Per-rule classifiers generally not useful (lack data), but 3 rules are exceptions.
- With-tools-as-feature classifiers better than without.
- Accuracy of single language vs. all-languages data: C > all-combined > Java

288 Classifiers Developed
- 15 featureless classifiers (20 or more audits, 100% True or False)
- 201 classifiers for 11 with mixed determinations
  - True/False ratio & count combination insufficient for classifiers, for some rules
- 72 all-rules classifiers name used as feature
  - 44 per-language classifiers

Results with DoD Transition Value
Software and paper: Classifier-development
- Code for developing classifiers in R
- Paper on classifier project [1]
Software: Enhanced-SCALE Tool (multi-tool alert auditing framework)
- Added data collection
- Archive sanitizer
- Alert fusion
- Offline SCALE installs and first VM training to ensure high-quality data
- SEI CERT coding rules
- Auditing rules [2]
- Enhanced-SCALE use
- Auditor quality test
- Test audit skill: mentor-expert designation

Conference/workshop papers from project:

Future work
Goal: improve accuracy
- Try different classification techniques
- Add features:
  - Semantic features (ICSE 2016)
  - Dynamic analysis tool results
- More audit archive data needed
- Additional data welcome! Potential collaborators, please contact me
- FY17 project focuses on rapid expansion of per-rule classifiers
Establishing Coding Requirements for Non-Safety-Critical C++

C++ is used extensively throughout the DoD, including major weapons systems such as the Joint Strike Fighter. Existing C++ coding standards fail to address security, subset the language (e.g., MISRA C++:2008), or are outdated and unprofessional (e.g., C++ Coding Standard referenced in DISA’s Application Security and Development STIG).

This project has resulted in

- acceptance by the Clang (a compiler front-end for C++ and other programming languages) community of a flag for enabling all Clang-tidy checkers that map to CERT secure coding guidelines

- 16 new C++ rules

- 15 new checkers to the Clang trunk

- two new C++ defect reports
Establishing Coding Requirements for Non-Safety-Critical C++ Systems

Writing secure C++ code is hard and existing coding standards are insufficient. Our research focuses on educating developers about C++ security issues through quality coding rules and alerting developers of security-related deficiencies in their source code through automated checkers.

The CERT C++ Coding Standard comprises 83 C++-specific rules spread over 11 broad categories of language constructs. Additionally, the Standard references 79 (out of the 102) rules from the CERT C Coding Standard that also apply to C++. Each rule has a title, an introduction & normative text, followed by a series of noncompliant code examples and their accompanying compliant solutions. Each rule also guides the user to the risks of failing to comply with the rule, what kind of automated detection mechanisms exist, what real-world vulnerabilities have resulted from failing to comply with the rule, and citations & related material.

Modified 137 C++-related rules and created an additional 16 rules on our public Wiki, engaging an average of 2000 unique visits per month. Contributed 15 checkers to the Clang open source C/C++ compiler, available by default for 10s of millions of programmers.

Research the kernel of a security-focused rule
Rule creation follows an iterative process involving multiple parties: hackers, authors, the C++ committee, and the C++ Standard itself. Hackers help form the kernel of a rule. External collaborators such as compiler writers and users help iterate the rule concept and checker behavior until it is solid and applicable to real-world code. The results are a more compelling rule and automated detection capabilities.

Our Results: Rules

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>C++ Rules</td>
<td>CERT C++ Coding Standard Rules</td>
</tr>
<tr>
<td>Old</td>
<td>New (FY16)</td>
</tr>
</tbody>
</table>

Our Results: Sections

1. Declarations and Initialization (DCL)
2. Expressions (EXP)
3. Integers (INT)
4. Containers (CTR)
5. Characters and Strings (STR)
6. Memory Management (MEM)
7. Input Output (FIO)
8. Exceptions and Error Handling (ERR)
9. Object Oriented Programming (OOP)
10. Concurrency (CON)
11. Miscellaneous (MSC)

Example Rule

DCL22-CPP. Functions declared with [[noreturn]] must return void

As described in MSC55-CPP: Do not return from a function declared [[noreturn]], because [[noreturn]] implies that the function returns a value to the caller even though it would result in undefined behavior.

Noncompliant Code Example

In this noncompliant code example, the function declared with [[noreturn]] claims to return an int.

```cpp
#include <cstdlib>

[[noreturn]] [[setc]] f() {
  int x = 5;
  return x;
}
```

This example does not violate MSC55-CPP: Do not return from a function declared [[noreturn]] because [[noreturn]] is declared [[noreturn]], so the return () statement can never be executed.

Compliant Solution

Because the function is declared [[noreturn]], and no code paths in the function allow for a return in order to comply with MSC55-CPP: Do not return from a function declared [[noreturn]], the compliant solution declares the function as returning void and adds the explicit return statement:

```cpp
#include <cstdlib>

[[noreturn]] void f() {
  std::exit(4);
}
```

Risk Assessment

A function declared with a non-void return type and declared with [[noreturn]] is confusing to consumers of the function because the two declarations are conflicting. In turn, it can result in misuse of the API by the consumer or can indicate an implementation bug by the producer.

<table>
<thead>
<tr>
<th>Rule</th>
<th>Severity</th>
<th>Likelihood</th>
<th>Remediation Effort</th>
<th>Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>DCL22-CPP</td>
<td>Low</td>
<td>Unlikely Low</td>
<td>P3</td>
<td>L3</td>
</tr>
</tbody>
</table>

Automated Detection

<table>
<thead>
<tr>
<th>Tool</th>
<th>Version</th>
<th>Checker</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clang</td>
<td>3.9</td>
<td>-[[noreturn]]</td>
<td></td>
</tr>
</tbody>
</table>

Related Vulnerabilities

Search for vulnerabilities resulting from the violation of this rule on the CERT website.

Related Guidelines

SEI CERT C++ Coding Standard
MSC54-CPP. Value-returning functions must return a value from all exit paths

Bibliography

[SEI-1486-2014] Subclause 7.3.1, "Nonreturn Attribute"
Automated Code Repair

Experience from CERT and DoD source code analysis labs shows that most software contains numerous vulnerabilities, largely arising from common coding errors. Automated code repair reduces a system’s attack surface and improves its ability to withstand cyber-attacks.

This project focused on integer overflow in calculations of how much memory to allocate and calculations related to array bounds. Through this work, we will reduce a typical number of unhandled violations to a number small enough for a development team to mitigate all of them.

Dr. Will Klieber
Software Security Engineer
Secure Coding Team,
Cybersecurity Foundations Initiative
Automated Code Repair

Integer overflow in calculations related to array bounds or indices is almost always a bug. We have developed and implemented an automated technique for repairing such bugs so that the program behaves as likely desired.

Experience from source code analysis labs at CERT and DoD shows that most software contains numerous vulnerabilities. A majority arise from common coding errors.

Static analysis tools help, but typically they produce an enormous number of warnings. The volume of just the true positives can overwhelm the ability of the development team to fix the code. Consequently, the team eliminates only a small percentage of the vulnerabilities.

Our work on automated repair is based on three premises:

1. Many security bugs follow common patterns. E.g., one common bug pattern is “p = malloc(n * sizeof(T))” where n is attacker-controlled. If n is very large, integer overflow occurs, and too little memory is allocated. This sets the stage for a buffer overflow later on.

2. By recognizing such a pattern, it is possible to make a reasonable guess of the developer’s intention (inferred specification). E.g., “Try to allocate enough memory for a objects of type T.”

3. It is possible to repair the code to satisfy this inferred specification. Example of repair: Insert code to check if overflow occurs and, if it does, to simulate malloc failing with ENOMEM.

Integer Overflow

Integers in C are stored in a fixed number of bits N (e.g., 32 or 64). Overflow occurs when the result cannot fit in N bits. In modular arithmetic, only the least significant N bits are kept.

This past year (FY16), we focused on integer overflow that leads to memory corruption. E.g.:

- Memory allocation: malloc(n), where the calculation of n can overflow.
- Integer overflow in array bounds check.

Example: Android Stagefright vul (July 2015) had both of the above types of overflows.

Repair: Emulate normal arithmetic

For non-negative integers with only addition or multiplication (no subtraction or division), the value is monotonically non-decreasing (except for multiplication by zero).

In this case, unlimited-bitwidth arithmetic can be emulated by using saturation arithmetic: Replace an overflowed value with the greatest representable value.

Experimental Results

<table>
<thead>
<tr>
<th></th>
<th>OpenSSL</th>
<th>Jasper</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overflows*</td>
<td>969</td>
<td>481</td>
</tr>
<tr>
<td>Overflows that are sensitive</td>
<td>233</td>
<td>101</td>
</tr>
<tr>
<td>Overflows fully repaired</td>
<td>180</td>
<td>53</td>
</tr>
<tr>
<td>Semi-repair</td>
<td>28</td>
<td>32</td>
</tr>
<tr>
<td>Unrepaired</td>
<td>25</td>
<td>16</td>
</tr>
</tbody>
</table>

An overflow is sensitive if it involves variables that are associated with array indices or bounds.

Conclusion

Automated code repair (ACR) reduces a system’s attack surface and improves its ability to withstand cyber-attacks.

ACR is suitable for problems where many security bugs follow a common pattern and have a corresponding pattern for repair.

In FY16, we focused on integer overflows involving memory bounds/indices.

A difficulty we encountered was the Source<->IR mapping problem.

- Code is most readily analyzed and repaired on an intermediate representation (IR). But actual repair must be on the source.
- Transformations on the IR aren’t unambiguously mappable to the source.
- Macros and #ifdefs are a further difficulty.
- We are continuing to investigate these issues in FY17.
Saving Malware Analysts Time

The SEI is developing an automated tool to dramatically reduce the time human analysts need to gather data for malware comparisons. The researchers are extending existing SEI automated analysis capability built in the ROSE open source compiler infrastructure to provide the required data. The ROSE infrastructure was developed at Lawrence Livermore National Laboratory.
Assuring Autonomy and Human-Machine Interactions
Why did the robot do that? Explaining Robot Behavior to Improve Trust in Autonomy

Work on this project will span FY2016 and FY2017

Government and industry are increasingly using robots in important tasks such as search and rescue operations. However, because robot behaviors can be hard to distinguish and understand, users mistrust and often abandon these very useful tools.

In this work, we hypothesize that having robots automatically explain their behavior using natural language will improve users’ trust and acceptance of them. To that end, we are developing algorithms to explain robot actions automatically.
Why did the robot do that?

Robots are increasingly being utilized in important tasks such as search and rescue operations. However, their behaviors are often hard to distinguish and understand, leading to users’ mistrust and often abandonment of very useful tools. We are developing algorithms for robots to automatically explain their behaviors to users and are demonstrating that these explanations improve users’ trust and acceptance of them compared to robots that do not explain themselves.

How can we capture diverse sets of user preferences for what the robot explains?

Representing preferences: We have developed a set of parameters that allow us to capture preferences such as level of abstraction and length and automatically generate different explanations based on those preferences.

User Interaction: The user can query the robot for more or different information if their preferences change or they want to dig deeper into the explanation.

How can we generate explanations of a diverse set of robots, sensors, actions, and tasks?

Our Methodology

We first poll many people to capture many different ways to explain example robot behaviors.

Then, we poll a new set of people to measure which words and explanations are best.

By analyzing the ranked explanations we can capture patterns of language that the robots should use in their explanations.

Order of importance:
1. Describing action
2. Describing immediate scene
3. Describing surrounding scene
4. Describing uncertainty in scene

Why is trust important?

Prior research has found that users often try to take control of their robots and limit autonomy when they lose trust in them, taking the user focus off the task at hand. Especially in time-sensitive applications like search and rescue in which robots can be utilized to perform dangerous tasks or speed up search tasks, we cannot afford to have first-responders lose trust or even stop using robots.
Human-Computer Decision Systems for Cybersecurity

Work on this project spanned FY2015 and FY2016

The DoD faces the challenges of securing deployed systems against malware and responding quickly enough when a security intrusion has been detected. Many in our field continue to ask the whether these processes can be completely automated, or whether machine learning has failed and these are tasks that intrinsically required human analysts. We assert that both human experts and machine learning (ML) play important roles in network defense. A system using only human experts cannot scale; pure ML systems are susceptible to structured attack by adversaries and have unsatisfactory performance on their own. In order for ML to become an effective tool for cyber defense, we must improve the collaboration between experts and automation.

In this work, we studied multiple facts of human-ML collaboration, using both real malware classification problems and a model problem based on malware classification. We investigated methods using both supervised (active) and unsupervised learning to augment the abilities of analysts. We also discovered a surprising result regarding the potential for non-experts to perform malware family analysis using low-dimensional visualizations.
Security decision systems aim to distinguish malicious activity from benign and often use a combination of human expert and automated analysis, including machine learning (ML). Systems using only human experts scale badly; pure ML systems are susceptible to structured attack by adversaries and, in most cases, have unsatisfactory performance on their own.

• Many operational security problems depend on a small number of skilled analysts to process a large and growing firehose of potentially malicious data.
• Traditional active learning tries to address this situation by suggesting allocation of limited analysis resources that optimize the convergence of a machine learning classifier.

How good is your cheap feature?

Cheap can be noisy... a different IAT hash

• 20k observations of 545 mnemonic counts reduced to two dimensions.
• Red points are a specific IAT hash of interest.
• This IAT hash (cheap) is well localized in t-SNE space (expensive)
• Knowing this IAT hash is likely good enough to define this family.
• Expert analysis concludes this is a single family.

Result: t-SNE-based visualizations paired with IAT section hashes greatly reduce the number of manual binary analyses required to understand new groups of binaries

• The human-computer collaboration model will improve upon traditional active learning by optimizing not simply for convergence of the ML component, but also for future performance of the overall system, including mutable human analysts.
• We test the performance of new models not only through simulation, but also through human-subject experiments.
• Because conducting these experiments using real security analysts performing their normal tasks would be prohibitively expensive, we instead developed a proxy problem of identifying fictional creatures and leveraged non-experts on Amazon’s Mechanical Turk platform. The process of generating the fictional creatures adheres to the statistical distributions of real malware classes.

Future work includes joint optimization of classifier and analyst objectives, extension of the experimentation software to support multi-session and team experimental trials, and a test of transferability of the model problem results to the target domain.

To keep pace with adaptive adversaries, our cybersecurity defenses must take advantage of both machine learning and human analyst strengths. Future solutions should optimize for success of the overall system.
Multi-Agent Decentralized Planning for Adversarial Robotic Teams (MADPARTS)

Effective wireless control of groups of collaborative robotic systems remains a problem, and a DoD-centered mission often involves an adversary or, at the very least, planning for an adversary potentially being present (e.g., in convoy, patrol, ISR, or force protection scenarios). There is a need in DoD for a more scalable, robust artificial intelligence that can learn and respond with dynamic group planning and control despite an intelligent, changing adversary, whether single individual agents or multiple collaborative enemy agents.

To meet the challenge of distributed autonomy in real-world robotics, we have created decentralized, multi-agent planning techniques, middleware, and algorithms that take into account a potentially changing adversary model in both simulations and real-world demonstrations with robotic unmanned surface vehicles.
Multi-Agent Decentralized Planning for Adversarial Robotic Teams

For the past four years, the SSD CPS-ULS group has been working on technologies to enable one human operator to control and interact with a team of autonomous, unmanned systems. In FY16 MADPARTS, we focused on defensive algorithms that protect a human operator or an important asset from a mobile adversary. We demonstrated our line-of-sight prevention algorithms in simulated quadcopters and in real-world demonstrations with unmanned surface vehicles in lakes near Pittsburgh. The algorithms resulted in line-of-sight prevention at over 99% success rates in simulations against mobile adversaries.

Defensive Schemes

- We took some inspiration from American football and robot soccer
- Zone defense: Protector agents move to assigned zones between a vip and the enemy
- Useful for holonomic robots like quadcopters
- Onion defense: Protector agents layer a defense between vip and enemy
- Useful for non-holonomic robots like fixed-wing planes and boats that drift

Our Autonomy Process

- Users write an application in C++ or Java
- Developers read and write to knowledge handled by the underlying middleware
- Platforms have standardized interfaces that algorithms interact with
- No interaction with message queues (handled under the hood)
- Users only have to focus on their algorithm or platform
- Built-in translations between simulation and real-world
- Pose system (Cartesian to GPS and vice-versa)
- High consistency, predictability and QoS
- Important for verification

Simulation Results

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Initial Defense</th>
<th>Defect Range</th>
<th>Failure</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scenario 1</td>
<td>Loose</td>
<td>Long</td>
<td>11.1%</td>
<td>200,000</td>
</tr>
<tr>
<td>Scenario 2</td>
<td>Loose</td>
<td>Short</td>
<td>0.5%</td>
<td>114,912</td>
</tr>
<tr>
<td>Scenario 3</td>
<td>Tight</td>
<td>Short</td>
<td>0.2%</td>
<td>114,504</td>
</tr>
<tr>
<td>Scenario 4</td>
<td>Tight</td>
<td>Long</td>
<td>0.0%</td>
<td>400,000+</td>
</tr>
</tbody>
</table>

Transition (ALW)

- PWP in place for AFRL Autonomy of the Loyal Wingman FY17-FY18
- Core software candidate for autonomous F-16 wingmen for a human pilot
- Algorithm creation for target defense and prosecution

Transition (NATO)

- Invitation to participate in NATO CMRE REP17-Atlantic exercise
- REP17 is a joint exercise between Portuguese Navy, NATO CMRE, and the University of Porto
- Current plan is for our autonomous boats to participate in the joint exercises

Transition (Multi-Planetary Smart Tile)

- GAMS and MADARA are core software architecture for the Keck Institute for Space Studies’ Phase 1 Multi-Planetary Smart Tile
- Hardware prototyped by GE GRC and Biovericom
- Separate offers to launch into LEO by United Launch Alliance and NASA
- Phase 1 is expected to perform simple autonomy experiments in low-earth orbit for up to 1 year
- Goal of project is to create a distributed, renewable power infrastructure for solar system that scales to tens of thousands of interacting robotic systems
Statistical Model Checking of Swarm Algorithms

The DoD is increasingly interested in using swarms (or ensembles) of autonomous systems against adversaries. However, the software and systems engineering communities lack methods to evaluate probability of mission success involving autonomous system swarms.

The project produced source code, prototype tools, and experimental results that validate the approach of applying adaptive sampling and input attribution toward (1) statistical model checking and (2) attribution of failure conditions.

Dr. Jeffery Hansen
Senior Researcher
Cyber-Physical and ULS Systems Initiative

For more information: sei.cmu.edu/about/people/profile.cfm?id=hansen_17141
Input Attribution – The “Why” of SMC

Statistical Model Checking (SMC) provides an estimate on the probability $P[\text{M satisfies } \Phi]$ that a predicate $\Phi$ in a model $\mathcal{M}$ is satisfied, but does not address why a particular result was obtained. The goal of Input Attribution (IA) is to use machine learning techniques to synthesize an explanation for an SMC result in terms of the inputs. IA for SMC can be thought of as analogous to the counter-example in traditional model checking.

A good Input Attribution has the following properties:
1. Describes relationship that actually exists in data
2. Is presented in a way that is quantitative and understandable
3. Gives investor new insights
4. Is resilient to randomness in the system

Example Scenario
Let $(x,y)$ and $(x',y')$ be random initial positions for a pursuer and an evader, respectively. The goal of the evader is to make it to one of several designated safe zones before it is caught by the pursuer. The SMC problem is to calculate the percentage of the “odds” that the evader will escape. Intuitively, the probability that the evader will escape increases as target relative error decreases.

Validation
Even though LR analysis may indicate statistical significance on one or more variables, the overall model must have a good fit to the data before an input attribution can be accepted. The AUC (Area Under Curve) of the logistic regression is a metric. AUC of 0.5 indicates the model is no better than guessing. An AUC of 1.0 is a perfect model.

Experimental Results
We conducted SMC trials of the pursuer/evader scenario shown above using the V-REP simulation environment. Trials were conducted on a set if six 20-core blade servers. A target relative error of 0.01 was used which resulted in 39,960 trials. The resulting “mission success” probability for the evader was 0.214. The LR model is a linear function of $x$ and $y$.

Conclusion
We applied SMC with Input Attribution to a pursuer/evader scenario. Intuitively we expected an Input Attribution indicating that the increased initial distance between pursuer and evader should be correlated with improved chance of escape for the evader.
Experiences Developing an IBM Watson Cognitive Processing Application to Support Q&A of Application Security (Software Assurance) Diagnostics

Contracting officers and program managers often cannot find assurance information in acquisition documents and artifacts or relate it to changes in risks and software.

This project provides the experiences of a team of computer scientists in building a cognitive processing application using IBM Watson, as a way to meet the needs of contracting officers and program managers. Both the process for building IBM Watson applications and the lessons learned are described.

The team represents a typical application team in that they are familiar with a technical domain—application security and software assurance—and are not experts in artificial intelligence, natural language processing, or cognitive computing.
Developing and IBM Watson Cognitive Processing Application
Supporting Application Security (Software Assurance)

IBM Watson made an impressive introduction. In 2011, Watson competed on one of America’s leading question and answer shows against former winners Brad Rutter and Ken Jennings. Watson received the first place prize of $1 million.*

Watson is a question answering computer system capable of answering questions posed in natural language, developed in IBM’s DeepQA project by a research team led by principal investigator David Ferrucci. Watson was named after IBM’s first CEO and industrialist Thomas J. Watson. The computer system was specifically developed to answer questions on one of America’s leading question and answer shows.

Application development timeline

**Example original document: CERT INT33-C**

**Rule - Parts**

- IBM Watson works on Solr document
- Each rule or CWE resulted in about 11 Solr documents
- Whole rule or CWE is a Solr document
- Key sections are Solr documents
- Many different formats within document
- Corpus held about 15,000 documents

**Watson’s interfaces for cognitive querying evolved over time**

Organization of technology rapidly evolved
- Splitting some components into distinct services
- Combining some services into usable chunks
- Ease-of-use interfaces delivered in open source (out of product cycle)

**Project focused on using “Retrieve and Rank” on BlueMix**

- Available support from IBM
- Combined Watson Pathways for Concept Expansion, Concept Insights and Question-and-Answer

**Application performance**

Better Recall and Precision: Example: “What is the risk of INT33-C”

<table>
<thead>
<tr>
<th><strong>INT33-C</strong></th>
<th><strong>Risk Overview</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Question and Answer (QAAPI)</strong></td>
<td><strong>with Local Infrastructure</strong></td>
</tr>
<tr>
<td><strong>Retrieve and Rank (R&amp;R)</strong></td>
<td><strong>with BlueMix infrastructure</strong></td>
</tr>
</tbody>
</table>

**Lessons learned from project**

**Theory**

Automated natural language comprehension

**Practice**

SME-driven Q&A training

**Disposition of materials**

Government use rights apply. IBM Watson software (and any dependencies) must be licensed from IBM.

SparkCognition is an IBM Watson business partner (independent software vendor) and has licensed the project materials from CMU for use in their products.

We want to thank and acknowledge collaborators

**SparkSecure team at SparkCognition**

IBM Watson team at IBM

Prof. Eric Nyberg, Language Technologies Institute, School of Computer Science, CMU

And our student interns: Christine Baek, Anire Bowman, Skye Toor and Myles Blodnick

---

*https://en.wikipedia.org/wiki/Watson_(computer)
GraphBLAS: A Programming Specification for Graph Analysis

Graph algorithms are in wide use in DoD software applications, including intelligence analysis, autonomous systems, cyber intelligence and security, and logistics optimizations. However, graph algorithms are difficult and costly to implement efficiently on hardware systems. As the size of graphs and the pace at which new hardware is being developed increase, the complexity of developing high performance graph libraries becomes a prohibitive barrier to the work of analyzing the deluge of information.

To address this problem, we are working with both leading graph analytics experts and high-performance computing experts from government, academia, and industry—the GraphBLAS forum—to derive an “interface” that represents a separation of concerns between lower-level implementations for specific hardware architectures and higher-level graph analytics concepts. By treating graphs as matrices and identifying primitives in terms of operations on these matrices, our approach is similar to what the scientific computing community accomplished with NIST’s Basic Linear Algebra Subprograms (BLAS) specification.
Graph BLAS
A Programming Specification for Graph Analysis

Graph algorithms are in wide use in DoD software applications, including intelligence analysis, autonomous systems, cyber intelligence and security, and logistics optimizations. However, graph algorithms are difficult and costly to implement efficiently on hardware systems. As the size of graphs and the pace at which new hardware is being developed increase, the complexity of developing high performance graph libraries becomes a prohibitive barrier to the work of analyzing the deluge of information.

Currently deep expertise is needed in graph algorithms and hardware tuning to achieve good performance on targeted hardware. It is rare to find this in individuals or even on teams within one organization.

The GraphBLAS Forum – a government, academic and industry consortium – has defined a set of graph primitive objects and operations and is nearing completion of the C Application Programming Interface (API) specification that is able to separate the concerns between:

- the graph expertise needed to develop advanced graph analytics (writing code using the API)
- the hardware expertise is needed to achieve high levels of performance (implementing efficient versions of the API for specific hardware).

For more information on the GraphBLAS Forum: http://graphblas.org

Graph Expertise

Separation of Concerns: GraphBLAS Application Programming Interface (API)

Hardware Expertise

GOAL: write once, run everywhere (with help from hardware experts).

The table above lists all of the primitive operations supported by the GraphBLAS API along with their mathematical description. These mathematical “requirements” are being captured in a C API Specification as shown for matrix multiplication (mmx) below.

```
GrB_Info GrB_mxm(GrB_Matrix *C,
const GrB_Matrix A,
const GrB_Matrix B,
const GrB_Semiring op,
const GrB_BinaryFunction accum,
const GrB_Mask Mask,
[const Descriptor desc]);
```
The Critical Role of Positive, Intrinsic Incentives in Reducing Insider Threat

Traditional guidance regarding how to defend against insider threat focuses primarily on practices that constrain employee behavior or that detect and punish misbehavior. However, excessive use of such negative incentives can result in counterproductive constraints on employees’ actions, overreliance on after-the-fact responses that fail to prevent damage, and alienation of staff that can actually exacerbate the threat that they are intended to mitigate. The objective of this project is to assess the potential for positive incentives to complement traditional cybersecurity practices in a way that provides a better balance for organizations’ insider threat programs.

We investigated the following dimensions along which to align an employee’s intrinsic incentive to act consistently with his or her employer’s interests: job engagement, perceived organizational support, and connectedness at work. Through insider-incident-case analyses and an organizational survey, we gained insight into the influence of positive incentives on insider threat risk. We developed a system dynamics model to capture the discovered relationships and explore how positive incentives can reduce operational costs as well as the insider threat. We expect that the evidence gathered will support a business case for organizations to complement traditional practices with positive intrinsic incentives as a win-win strategy to improve both employee satisfaction and organizational performance.
Empirical analysis shows insider alienation and the potential of positive incentives for reducing insider threat baseline. A simulation model illustrates benefits in terms of fewer incidents and lower costs. Balanced deterrence is key!

An Emerging Physics of Employee Dissatisfaction and Insider Threat
- System Dynamics model of how flow of dissatisfaction translates into incidents
- Empirical analysis providing structural validation of model
- Annual data on USG employee attitudes grounds simulation model
- Sensitivity simulation captures uncertainty

Preliminary Analysis Conducted:
- Case analysis shows organization support foundational
- Insider threat program survey shows negative correlation between organization support, insider threat

Positive Incentive-Based Workforce Management Practice Areas
- Positive incentives encouraging participation, acknowledgment, and recognition
- Performance-based rewards and recognition
- Environment and Employee Communications
- Employee and Supervisor Expectations
- Communication

Future Research and Transition:
- Theory: Experiment to determine cause-effect relationship between positive incentives, threat
- Adoption: Transition model for organization to go from current state to state with appropriate mix of positive and negative incentives
- Technology: Detection of insider alienation by identifying at-risk behaviors and indicative changes in networks of coworker relations

Balanced Deterrence: Extending the Traditional Security Paradigm

Security Through Positive Incentives
- Engagement Feedback
  - Engagement
  - Correctedness
  - Organizational Supportiveness
  - Supported Employees

Positive Deterrence
- Fewer unintended consequences
- Satisfaction, performance, retention

Balanced Deterrence
- Fewer insider incidents and misbehaviors
- Lower investigative costs, productivity loss

Negative Deterrence
- Lower rates of insider incidents, misbehaviors, and attacks

Simulation Controls
- 0 | 2 | 5 percent disgruntled starting to attack
- 0 | 50 | 100 percent satisfaction improvement

Insider Threat Incidents
- Number of Insider Threats After 20 Years

Future Research and Transition:
- Theory: Experiment to determine cause-effect relationship between positive incentives, threat
- Adoption: Transition model for organization to go from current state to state with appropriate mix of positive and negative incentives
- Technology: Detection of insider alienation by identifying at-risk behaviors and indicative changes in networks of coworker relations
Workplace Violence/IT Sabotage: Two Sides of the Same Coin?

It is difficult to have a coherent, integrated means for mitigating diverse threats from disgruntled insiders. To address the challenge, this project compared incidents of Information Technology Sabotage (ITS) to existing cases of workplace violence (WPV) and workplace aggression (WPA) in the DoD/Intelligence Community.

We identified the observable predispositions, stressors, and concerning behaviors that were common to both types of crimes as well as those that were found in only one type of crime. From these findings, it should be possible to develop common indicators that apply to both types of crime in a more coherent and integrated way.
Workplace Violence and IT Sabotage: Two Sides of the Same Coin?

We set out to determine if coherent, integrated, and validated indicators for Insider Workplace Violence (WPV) and Insider Cyber Sabotage (ICS) can be identified.

**Reason:** If there are common indicators organizations may be able to develop socio-technical controls that prevent, detect, and help respond to both threats without identifying which crime will eventually be committed.

**Approach:** Collect, code, and analyze cases of WPV and compare them to cases of ICS in the CERT Insider Threat Center's corpus.

**Coding & Analysis:** We coded WPV & ICS cases for personal predispositions, stressors, concerning behaviors, problematic organizational responses, and the hostile act to identify a common incident pathway.

**WPV & ICS Incident Pathway**

**WPV and ICS Pathways**

The pathways were most common in areas of predispositions and stressors. Concerning behaviors was usually the earliest point where organizations might be able to determine if a hostile act might manifest as WPV or ICS.

**Coding for Stressors.**

The clearest commonality between all the coding factors were the categories of stressors that the perpetrators experienced. These were coded into six major categories: personal, financial, mental health, work, relationship, and work relationship. Two areas that were significant in both WPV and ICS were work and work relationship stressors; two areas that organizations could have the greatest influence over.
Data Validation for Large-Scale Analytics

Large-scale analytics have wide application across the DoD and the Intelligence Community, but the process of constructing data analytics is iterative and incremental and is rife with challenges. Concerns about data quality, validating assumptions, and understanding anomalies and errors permeate the process.

We are building automated data sampling and visualization tools to help data scientists inspect and understand their large-scale data. With our collaborators at Carnegie Mellon University, we are demonstrating through user studies that these tools increase the quality of data analysis. The tools are available for download.
Data Validation for Large-Scale Analytics
Building Tools to Support Data Sampling and Visualization

Large-scale analytics hold great promise for government and industry, and data validation is essential to ensure that those analytics make accurate predictions. We studied practitioners in the field, built data validation tools to support data sampling and visualization, and found that our tools help practitioners generate a diverse set of insights about their data.

Why is Data Validation Important?
Data analysts agree that their biggest challenges are data quality, validating assumptions, and understanding anomalies and errors throughout the process. These challenges are not about the correctness of their code but rather the validation of data analysts’ assumptions about their data and subsequent analytics. Without valid data, data science practitioners cannot be sure that their resulting machine learning algorithms are making accurate predictions using relevant features and correct labels.

Today’s Data Validation Practices
The state of the art solution to data validation today is human experts who manually sort through predictions and confirm assumptions. However, the process of understanding even a subset of data points is extremely tedious and error prone, especially as the number of data points and features grows.

Data scientists today only have a few data sampling techniques available to them to give them insight into the distribution, common values, and anomalies of their data and they do not sample large datasets efficiently.

Implementing and Visualizing Multiple Sampling Techniques
We hypothesized that using many data sampling techniques would allow practitioners to learn more about their data compared to their current practice. We implemented four data sampling techniques not available on current platforms today to allow data scientists to process and sample large scale datasets:
- Random Sampling
- Uncertainty Sampling
- Density Sampling
- Query By Committee

Each sampling technique selects different subsets of the data, allowing data scientists to capture multiple views of their data more effectively than they could with current tools.
Supporting Software Engineering Best Practices in Additive Manufacturing

Additive manufacturing (or 3D printing) provides new opportunities for the DoD to lower the cost and reduce the time needed to create replacement or customized parts and components. However, 3D printing communities—like software development communities from decades ago—do not have tools to modularize their 3D models for reuse in other applications.

In this project, we have developed a framework to support scalable production and customization of 3D models by enabling modelers to decompose objects into functional parts that they can reason about and interchange independently.
3D printing, also called additive manufacturing, is a powerful medium to use to prototype and design objects. However, current tools for fabrication do not take advantage of basic concepts such as modularity and abstraction that have made it possible to develop highly complex and re-usable software systems and tools. We propose the Parameterizable, Abstractions of Reusable Things (PARTs) Framework, a parallel to object-oriented software classes, to support the validation and integration of 3D models using a combination of geometry and logic.

Modular Design
Elements are designed separately & recombined at any time.

With today’s 3D modeling software, modelers can create modular models containing multiple 3D geometric surfaces and objects, and it is up to them how the objects integrate together. Additionally, they must manually check their assumptions about how those parts can be combined rather than depending on the software to validate those assumptions automatically.

In PARTs, 3D models are created the same way. However, programmers can assign assertions to the geometry to allow the software to identify when their assumptions are not met. Similarly, they can create integrators to ensure that their object is combined with others in particular ways that they specify. As a result, 3D models can be reused and integrated modularly.

Modelers combine geometry and logic to define PARTs as a set of assertions and integrators. Shown are two parts of a smartphone bike mount. With PARTs, we can develop the phone holder and clamp individually, then iteratively combine them until their assertions and integration rules are met. Finally, we can integrate the PARTs together into a single geometry to print.
Driving Control Standards for Unmanned Systems

The lack of a common architecture for control among the Unmanned Aircraft Systems (UASs) limited their mission capabilities. SEI experts were key contributors to the development of an architecture-focused standard for the UAS Control Segment Working Group (UCS-WG).
Assuring Cyber Workforce Readiness
Utilizing Serious Games to Assist Motivation and Education

To make the best use of the DoD’s extremely limited time for continuation training, we are testing and measuring the benefit of gamification and serious games on participant motivation and attainment of educational goals.

To this end, we are integrating a battlefield simulator with the existing CERT Simulation, Training, and Exercise Platform (STEP). The resulting system allows for the effects of operations in the kinetic domain to propagate into the cyber domain and, similarly, for effects in the cyber domain to propagate into the kinetic domain.
Utilizing Serious Games to Assist Motivation & Education
Leveraging: Cyber Kinetic Effects Integration (CKEI)

In an increasingly interconnected world, DoD is tasked with completing missions requiring cyber operator support. DoD has limited resources for continuing training. The cyber operator community is largely driven by outliers, experts creating new capabilities usable across the community. Our program aims to stimulate the creation of experts by bringing together the cyber and kinetic domains to create a highly motivational training experience.

**Approach**
Integration of realistic kinetic simulations with our existing cyber simulation capabilities can be used to create a gamified training experience that simulates the complex realities of a cyber-physical environment and also captures the attention of participants to drive emotional investment in the mission.

**Tactical Resources Defense:** Friendly assets are modeled in game and attackable. Failure to defend cyber terrain may result in loss of communications or the crashing of intel drones.

**User Testing:** Events conducted as part of ISC2 High-school Summer Cyber Challenge to gauge effectiveness.

**Combined Landscape:** A fully modeled cyber-physical environment allows participants to explore and develop new strategies for completing their missions.

**Shared World:** Special Operators, Drone Operators, and Cyber Operators must work together cohesively to complete missions within the environment.

**STEP Technology:** Mature range capability provided by STEP technology modified to allow seamless connection to kinetic simulations.
Generalized Automated Cyber-Readiness Evaluation (ACE)

Work on this project spanned FY2015 and FY2016

It is important for the DoD gain the capability provided by a scalable, objective assessment capability that it can use to validate the hands-on, technical knowledge and skills of its cyber workforce.

In this project, we have developed the first generation of the Automated Cyber-Readiness Evaluator—a system designed to automatically interpret the actions a user performs on a computer screen and objectively measure that user’s competence within a defined knowledge and skill set.

Rotem Guttman
Cybersecurity Exercise Developer and Trainer
Workforce Development Initiative

For more information:
sei.cmu.edu/about/people/profile.cfm?id=guttman_18232
Assessing the mission readiness of all DoD cyber operators is a daunting task that is not achievable using individual one-on-one evaluation techniques. Our project utilizes advanced computer vision and machine learning techniques to evaluate the activity of cyber operators in a realistic scenario in order to determine their mission readiness.

**Mission Readiness Assessment**
The DoD must assess the capability and capacity of its cyber workforce to support operations conducted in the cyberspace domain and this assessment capability is a key determinant of operational mission readiness. However, because cyber is a relatively new domain for the DoD, it does not yet have a scalable, objective assessment capability that it can use to validate the hands-on, technical knowledge and skills of its cyber workforce.

**ACE Philosophy**
Current evaluation methods involve checklists of prompted activities or individual assessments. These methods are not reliable, not uniform, and not scalable to DoD requirements. The ACE philosophy is that true mission readiness assessments can only be performed in a realistic environment. ACE users are placed in an environment that mimics their real work environment. Our automated system then observes and understands the actions performed within this environment as users attempt to complete a mission. Based on their activities, our system assesses their knowledge, skills, and abilities.

**ACE Capture**
ACE evaluation scenarios are conducted in the CERT® Simulation, Training, and Exercise Platform (STEP). This platform allows us to push out realistic simulations of real DoD networks through a web browser. The ACE-Capture module has been integrated into the STEP platform, allowing unattended background recording of participants within an evaluation scenario. This recording is performed on the backend servers and consists only of the views we provide to the end users – thus avoiding the possibility of accidentally collecting any personal information that may exist on their personal workstation. Our recording system is highly scalable. It allows us to simultaneously record dozens of users per allocated machine and natively scales with available hardware.

**ACE Vision**
Video recorded by the ACE-Capture system is processed by a dedicated vision engine that detects a wide array of GUI elements, as well as a set of relevant console commands. These detections (and their associated confidence measures) are generated utilizing a highly optimized, parallelizable algorithm that takes advantage of the unique conditions available within our simulation environment.

**ACE Eval**
The detections generated within the ACE-Vision system provide the data for evaluation by ACE-Eval. This system is composed of two layers. Layer 1 maps groups of detection events with associated higher level activities such as “Opened file examiner_notes.txt for editing in gedit” or “Mounted the evidence drive”. Layer 2 maps these activities to specific activities identified as critical for a given job role.

**Looking Forward**
- Merger with existing sponsored work
- Addition of multiple job roles
- Customer-driven assessment creation

**Additional use-cases**
- Stand-Alone operation
- Insider Threat Analyst Support
- Dynamic Workstation Monitor
- User Study Data Collection
- Template Generation Utility
- Assessment creation
- Compatilble with user simulation (GUS)

By utilizing the automated generation of reliable skill reports, commanders may easily assess the capabilities of their troops, at scale, and with the resources already available.
Technology and Know-How for Critical Cyber Exercises

USCYBERCOM uses SEI learning technologies to support Cyber Flag and Cyber Guard tactical exercises and as the basis for its Persistent Training Environment. In addition, AFCYBER, ARCYBER, MARFORCYBER, and others choose SEI platforms for their cyber capabilities exercises.
About Us
The Software Engineering Institute (SEI) is a not-for-profit Federally Funded Research and Development Center (FFRDC) at Carnegie Mellon University, specifically established by the U.S. Department of Defense (DoD) to focus on software and cybersecurity. As an FFRDC, the SEI fills voids where in-house and private sector research and development centers are unable to meet DoD core technology needs.