Software Solutions Symposium 2017

March 20–23, 2017

Building Secure Software for Mission Critical Systems

Mark Sherman, PhD

Technical Director, CERT

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

Agenda

- **State of software**
- **Building software: the Secure Software Development Lifecycle**
  - Requirements
  - Development
  - Operations
- **Review**

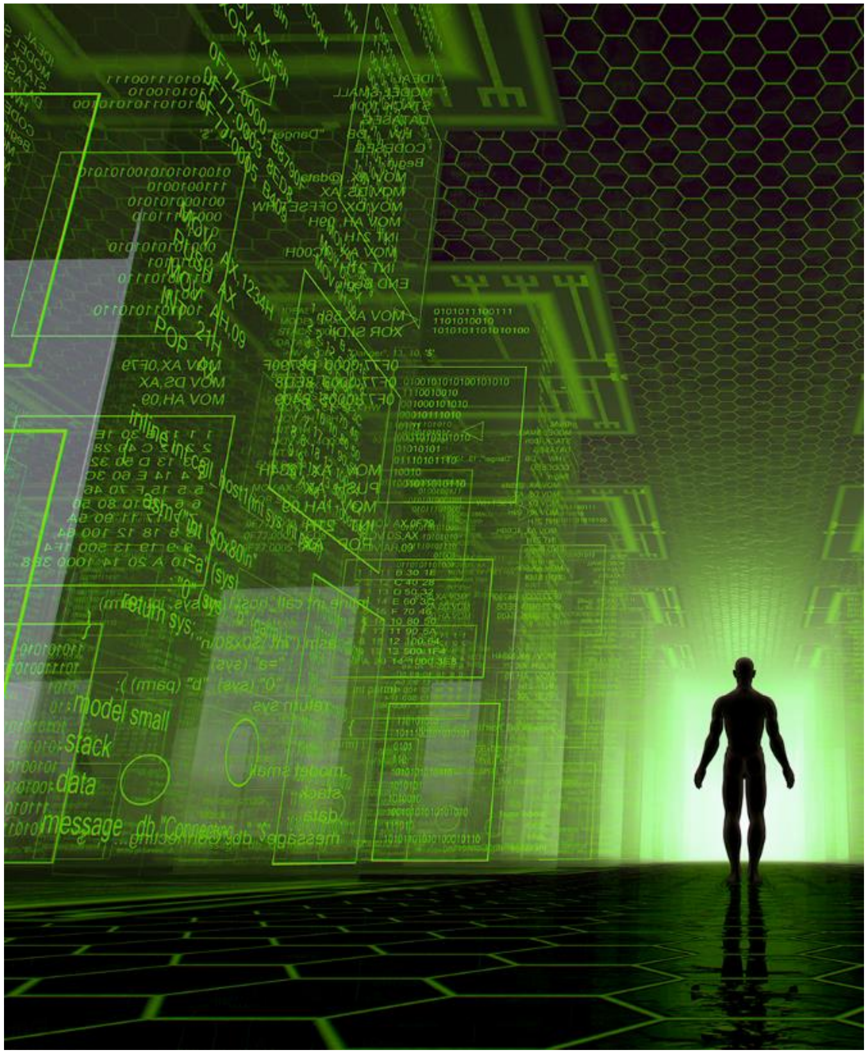
"Software is eating the world"

Marc Andreessen
Wall Street Journal
Aug 20, 2011

Software is the new Hardware

Source: http://www.wsj.com/articles/SB10001424053111903480904576512250915629460

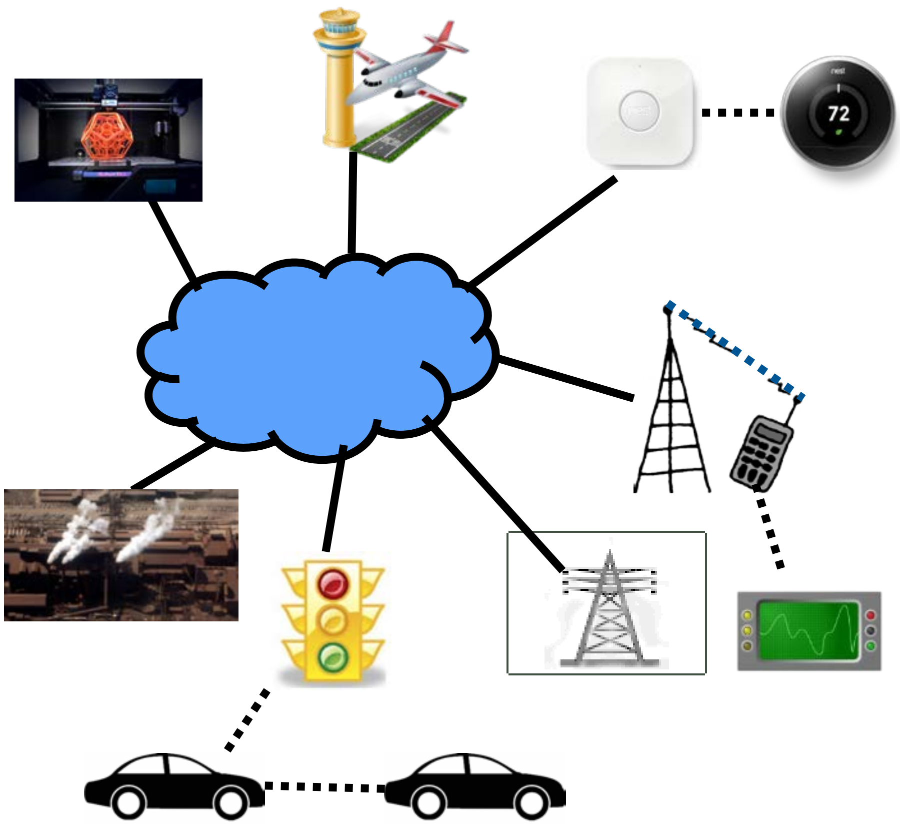
Software is the new hardware – IT



IT moving from specialized hardware to software, virtualized as

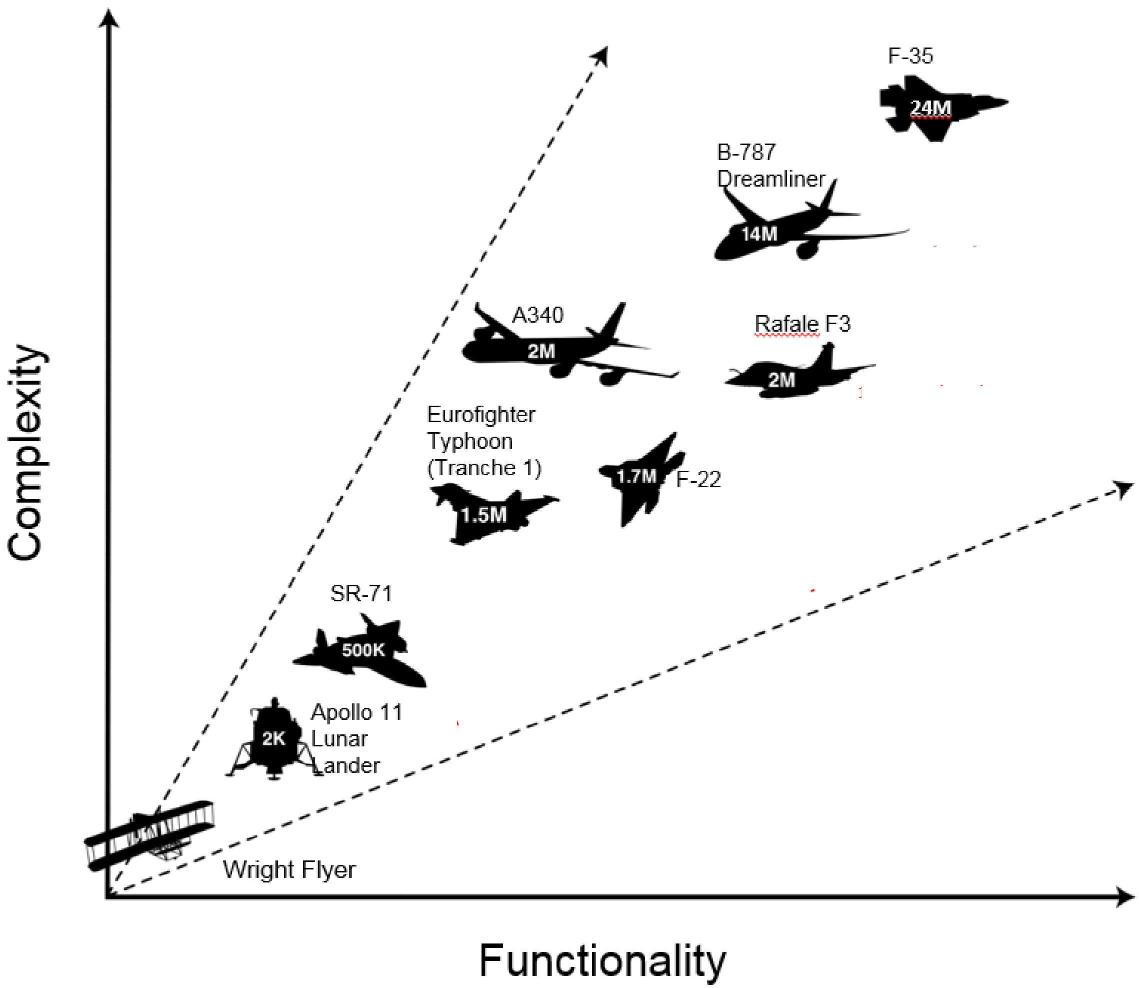
- Servers: virtual CPUs
- Storage: SANs
- Switches: Soft switches
- Networks: Software defined networks

Software is the new hardware – cyber physical



- Cellular
  - Main processor
  - Graphics processor
  - Base band processor (SDR)
  - Secure element (SIM)
- Automotive
  - Autonomous vehicles
  - Vehicle to infrastructure (V2I)
  - Vehicle to vehicle (V2V)
- Industrial and home automation
  - 3D printing (additive manufacturing)
  - Autonomous robots
  - Interconnected SCADA
- Aviation
  - Next Gen air traffic control
- Smart grid
  - Smart electric meters
  - Smart metering infrastructure
- Embedded medical devices

Mission function is increasingly delivered in software

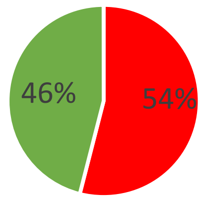


"The [F-35] aircraft relies on more than 20 million lines of code to "fuze" information from the JSF's radar, infrared cameras, jamming gear, and even other planes and ground stations to help it hunt down and hide from opponents, as well as break through enemy lines to blow up targets on the ground. …. But if the computer doesn't work, the F-35's greatest advertised advantages over existing rivals and future threats would suddenly become moot."
The Week, 2016

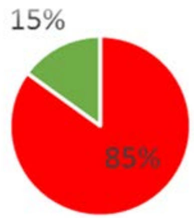
Source: Joseph Trevithick, http://theweek.com/articles/605165/f35-still-horribly-broken. Feb 26, 2016

Software vulnerabilities are ubiquitous

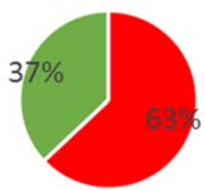
Existing Customer Premise Equipment (SOHO) typically vulnerable



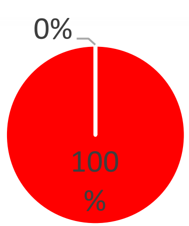
54% of tested routers are vulnerable to cross-site request forgery (CSRF)



85% of tested routers use non-unique default credentials



63% of tested routers are vulnerable to DNS spoofing attacks



100% of router firmware use BusyBox versions from 2011 or earlier and embedded Linux kernel versions from 2010 or earlier

Source: Land, J. "Systemic Vulnerabilities in Customer-Premises Equipment Routers," unpublished white paper, 2015

Steel furnaces have been successfully attacked

"Steelworks compromise causes massive damage to furnace.

One of the most concerning was a targeted APT attack on a German steelworks which ended in the attackers gaining access to the business systems and through them to the production network (including SCADA). The effect was that the attackers gained control of a steel furnace and this caused massive damages to the plant."

Source: Sources: https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/Lageberichte/Lagebericht2014.pdf?__blob=publicationFile; http://www.resilienceoutcomes.com/state-ict-security/

Electric grid under attack

BlackEnergy trojan strikes again: Attacks Ukrainian electric power industry

BY ROBERT LIPOVSKY IN COOPERATION WITH ANTON CHEREPANOV POSTED 4 JAN 2016 - 12:49PM

CYBERCRIME TAGS BLACKENERGY UKRAINE

On December 23rd, 2015, around half of the homes in the Ivano-Frankivsk region in Ukraine (population around 1.4 million) were left without electricity for a few hours. According to the Ukrainian news media outlet TSN, the cause of the power outage was a “hacker attack” utilizing a “virus”.

Source: http://www.welivesecurity.com/2016/01/04/blackenergy-trojan-strikes-again-attacks-ukrainian-electric-power-industry/

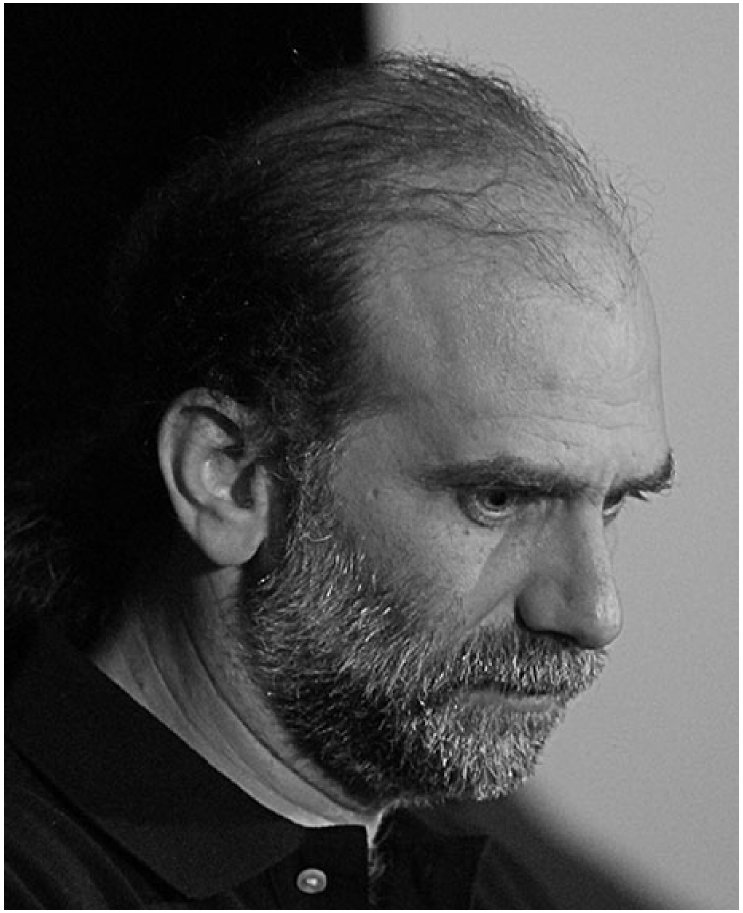
Weapons platforms potential cyber attack targets

"The [Joint Strike Fighter] aircraft relies on more than 20 million lines of code … In November 2015, the Pentagon canceled a cyber test because of worries it would, unsurprisingly, damage [the Autonomic Logistics Information System that identifies broken parts and other faults]."

The Week, 2016

Sources: https://www.dvidshub.net/image/935698/aerial-refueling-f-35-lightning-ii-joint-strike-fighters-eglin-afb-fla;
Joseph Trevithick, http://theweek.com/articles/605165/f35-still-horribly-broken. Feb 26, 2016

An ounce of prevention ….



"We wouldn't have to spend so much time, money, and effort on network security if we didn't have such bad software security."

Bruce Schneier in Viega and McGraw, "Building Secure Software," 2001

Source: Washington Post, March 19, 2014, http://www.washingtonpost.com/business/economy/toyota-reaches-12-billion-settlement-to-end-criminal-probe/2014/03/19/5738a3c4-af69-11e3-9627-c65021d6d572_story.html; http://www.greene-broillet.com/Articles/Toyotasuddenacceleration.shtml

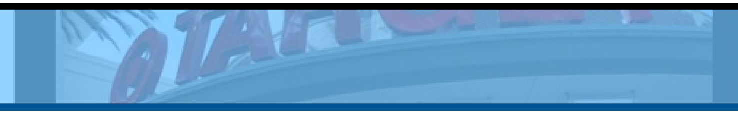
Software and security failures are expensive

Sections

The Washington Post

Business

Toyota reaches $1.2 billion settlement to end probe of accelerator problems



[GREENE BROILLET & WHEELER, LLP]
WHERE SUCCESS IS A TRADITION®

Toyota Sudden Acceleration Defect Case: $1.1 Billion Settlement

EMAIL
FACEBOOK
TWITTER
SAVE
MORE

Target on Friday revised the number of customers whose personal information was stolen in a widespread data breach during the holiday season, now reporting a range of 70 million to 110 million people.

The stunning figure represents about a third of all American adults at the low end, and is nearly three times as great as the company's original estimate at the upper end. The theft is one of the largest ever of retail data.

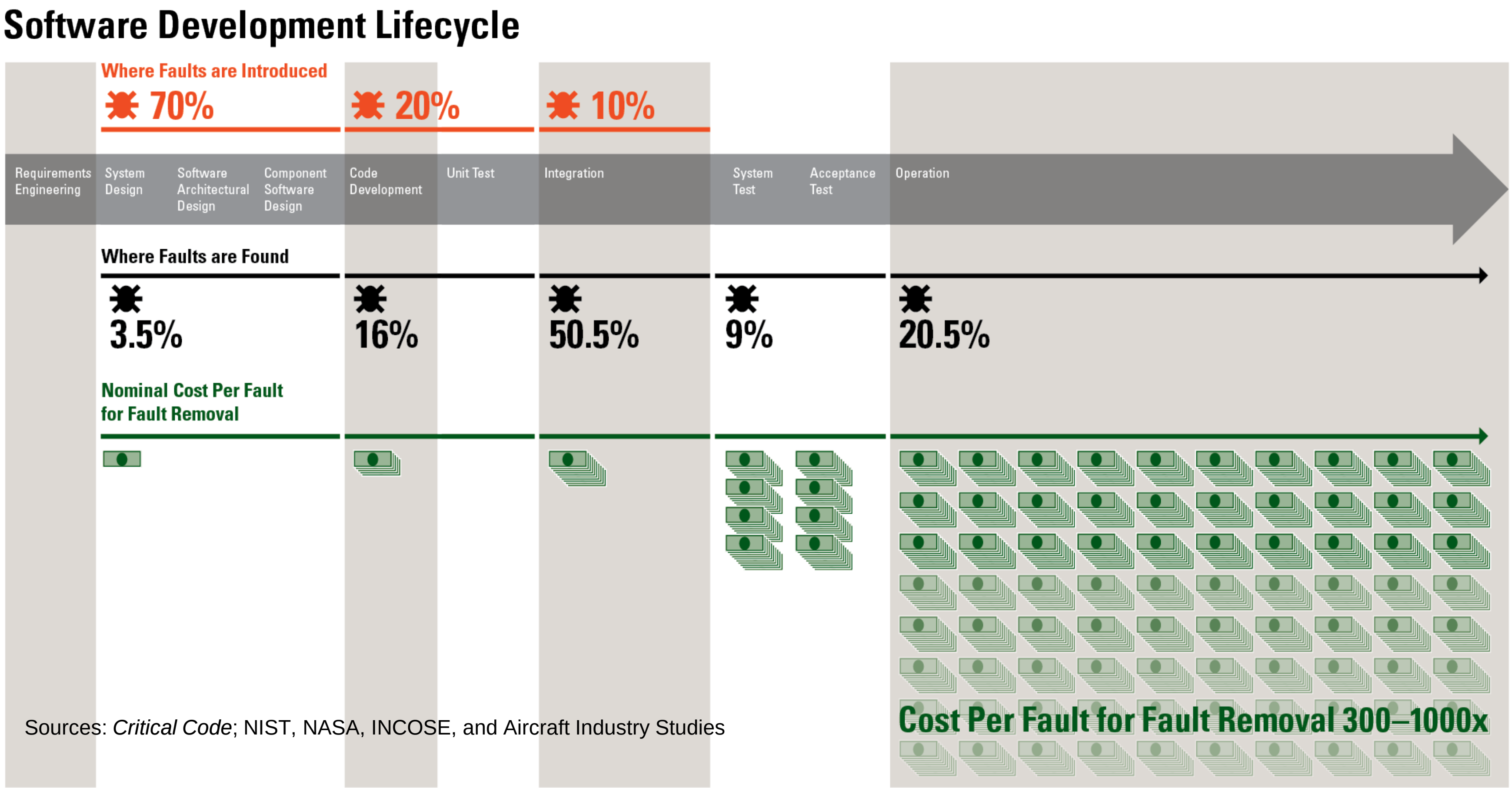
Source: New York Times, Jan 10, 2014

Average cost in a breach:
US$188 per record

Source: Ponemon Institute, "2013 Cost of Data Breach Study: Global Analysis", May 2013

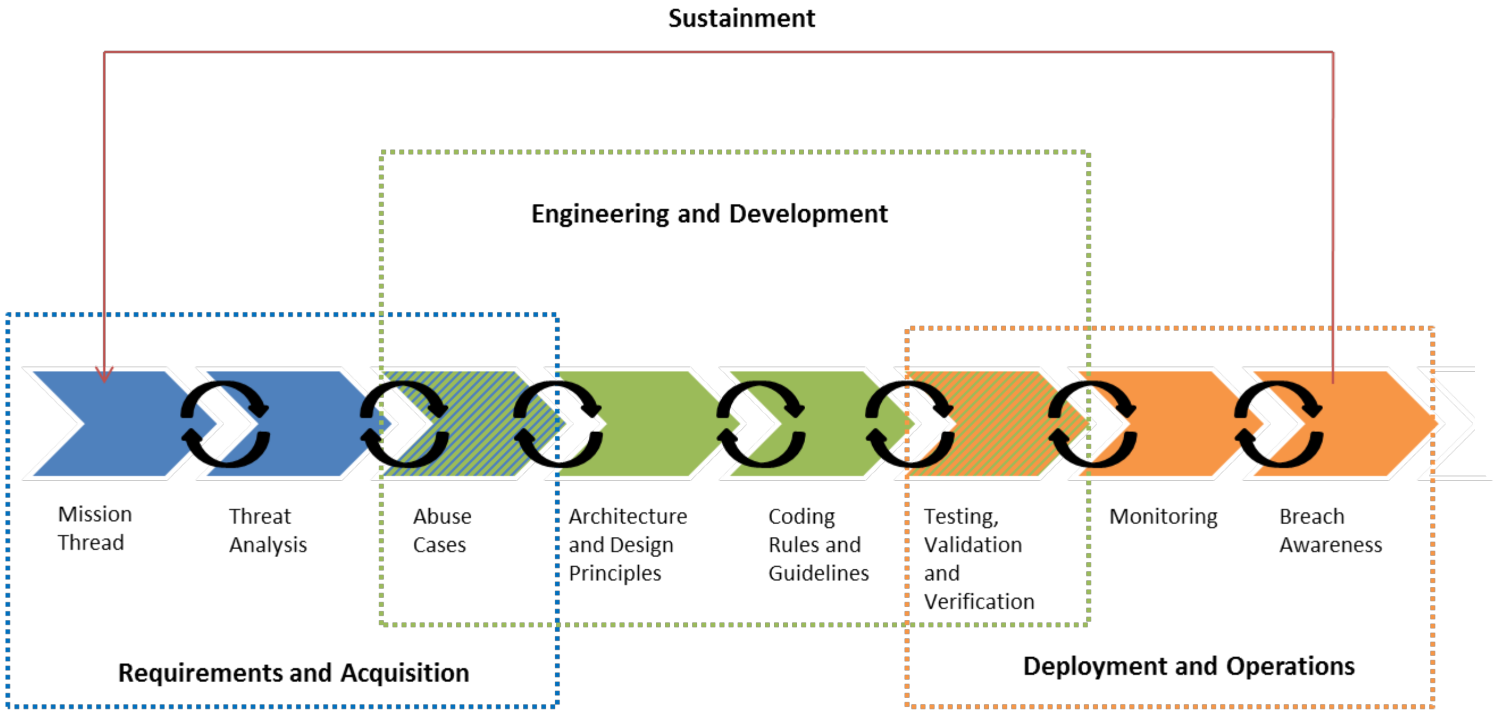
Catching software faults early saves money

Faults accounts for 30–50% percent of total software project costs

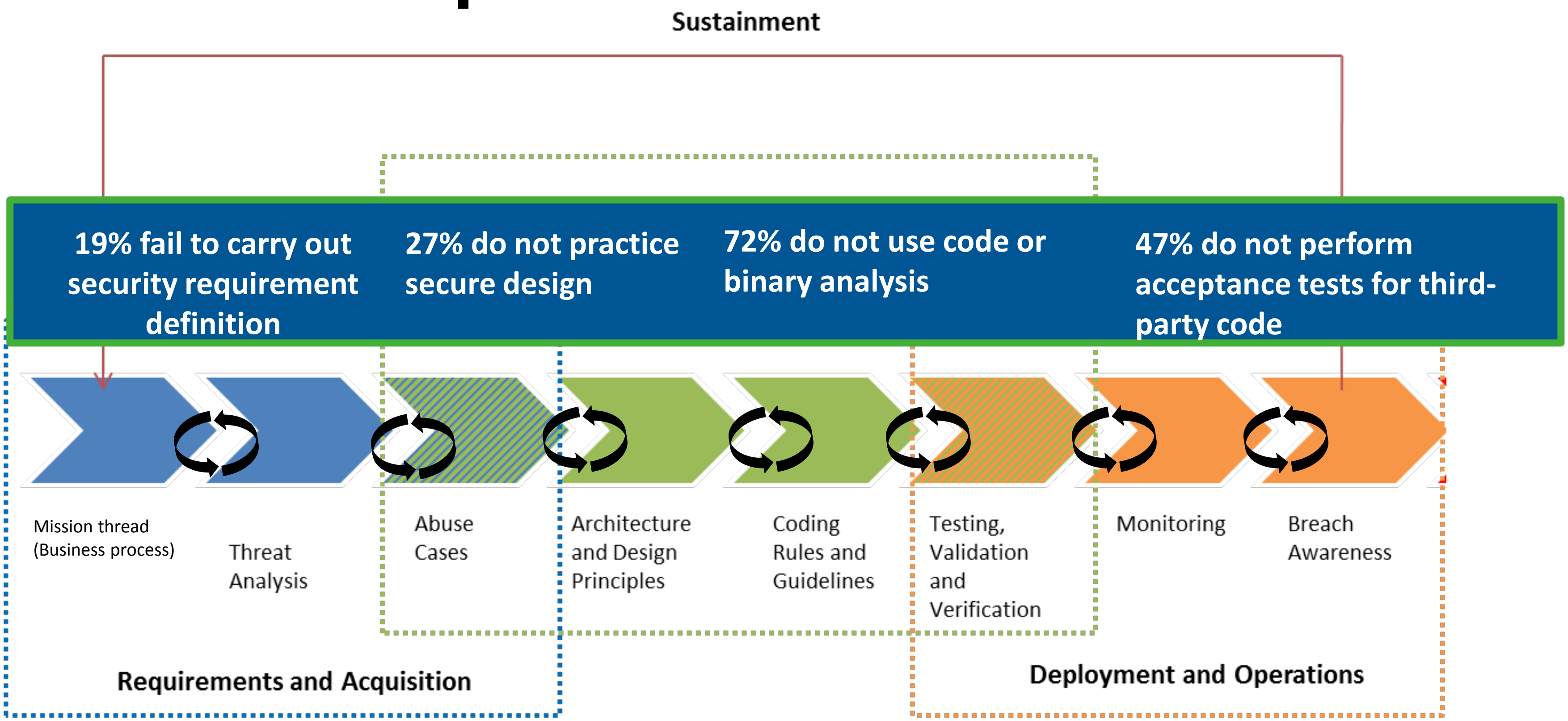


Sources: *Critical Code*; NIST, NASA, INCOSE, and Aircraft Industry Studies

Security is a lifecycle issue



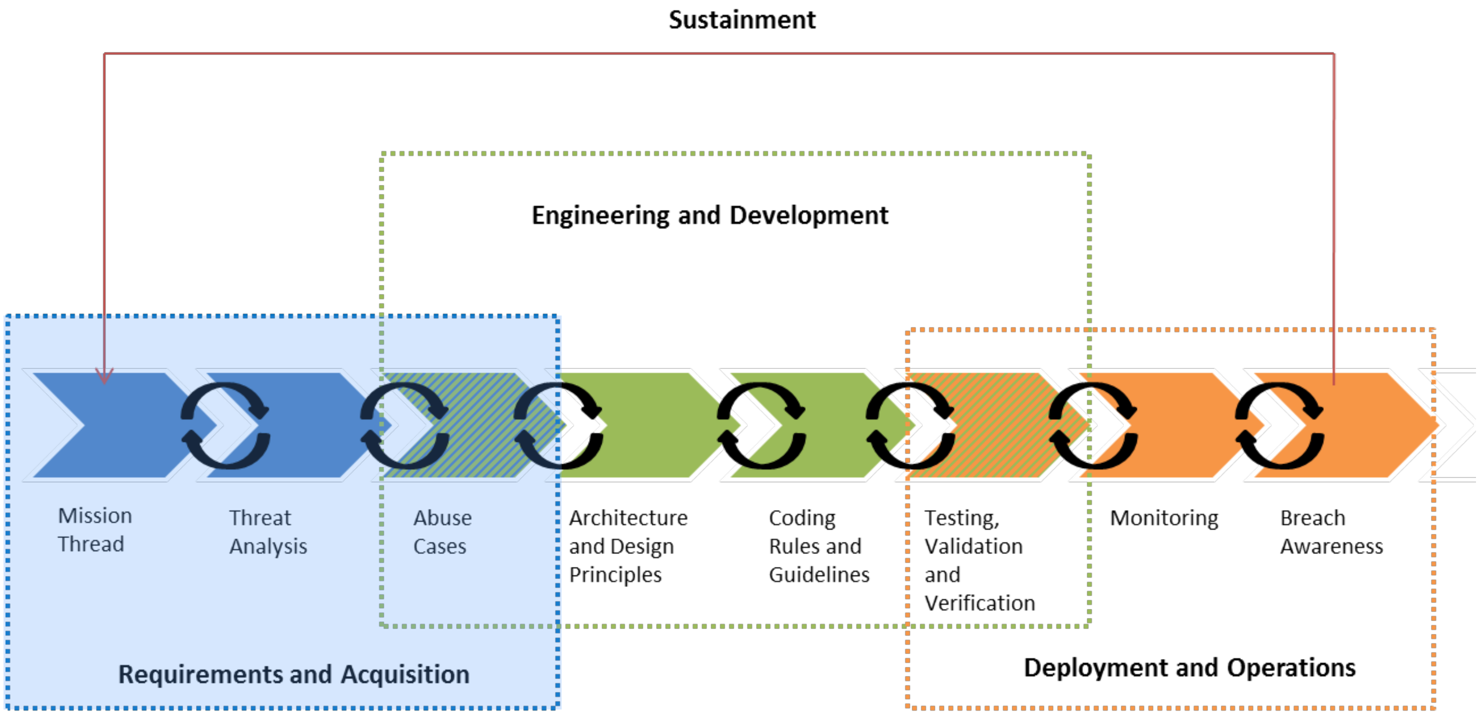
Room for improvement



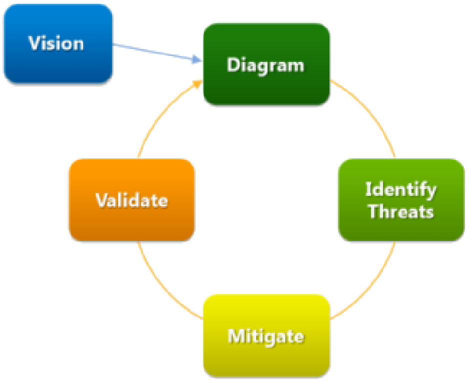
More than 81% do not coordinate their security practices in various stages of the development life cycle.

Sources: Forrester Consulting, “State of Application Security,” January 2011; Wendy Nather, Research Director, 451 Research, “Dynamic testing: Why Tools Alone Aren't Enough, March 25, 2015”

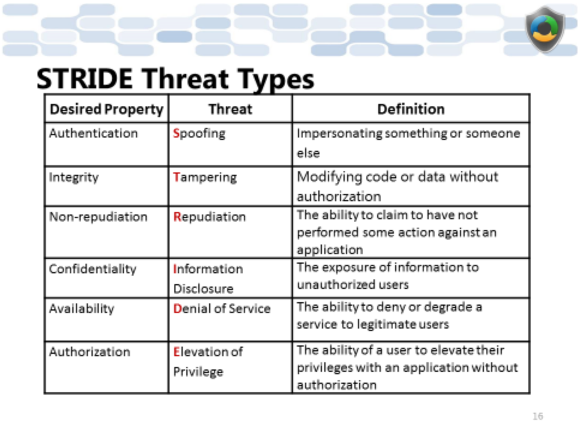
Requirements



Threat analysis tools help derive abuse and misuse cases



Microsoft SDL Threat Modeling Tool

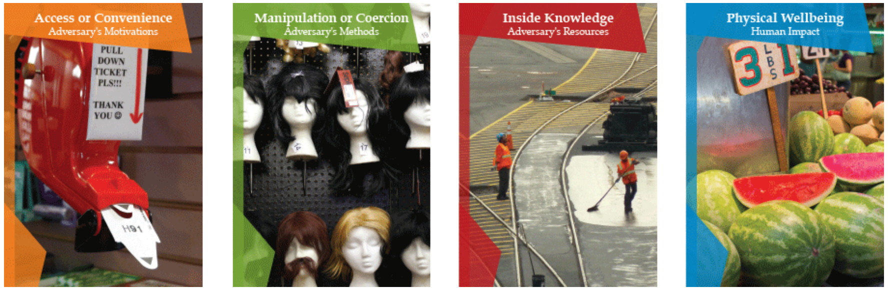


STRIDE Threat Types

Desired Property	Threat	Definition
Authentication	Spoofing	Impersonating something or someone else
Integrity	Tampering	Modifying code or data without authorization
Non-repudiation	Repudiation	The ability to claim to have not performed some action against an application
Confidentiality	Information Disclosure	The exposure of information to unauthorized users
Availability	Denial of Service	The ability to deny or degrade a service to legitimate users
Authorization	Elevation of Privilege	The ability of a user to elevate their privileges with an application without authorization

16

Microsoft STRIDE Threat Types



Denning, Friedman, Kohno
The Security Cards: Security Threat Brainstorming Toolkit

Jane Cleland-Huang's Persona non Grata
http://www.infoq.com/articles/personae-non-gratae

Embedded systems represent new classes of vulnerabilities

Embedded systems have different characteristics than IT systems

More and varied attack surfaces

- Sensors
- Multiple command-and-control masters
- Embedded firmware, FPGAs, ASICs
- Unique internal busses & controllers

Size, weight, power and latency demands tradeoff against defense-in-depth

Timing demands offer potential side channels

- Bit and clock cycle level operations
- Physical resources with real time sensors
- Safety-Critical Real-time OS

Confusion between failure resilience and attack

- Intermittent communications

Security approaches for IT systems do not cover embedded system security

Virus definitions and operating guidelines do not always apply

Firewalls and IDS/IPS of limited value

Centralized account control not possible

Network tools and assessment techniques unaware of embedded systems architecture and interfaces

- Unique and insecure protocols
- Maintenance backdoors
- Hardcoded credentials
- Unique architectures of embedded controllers

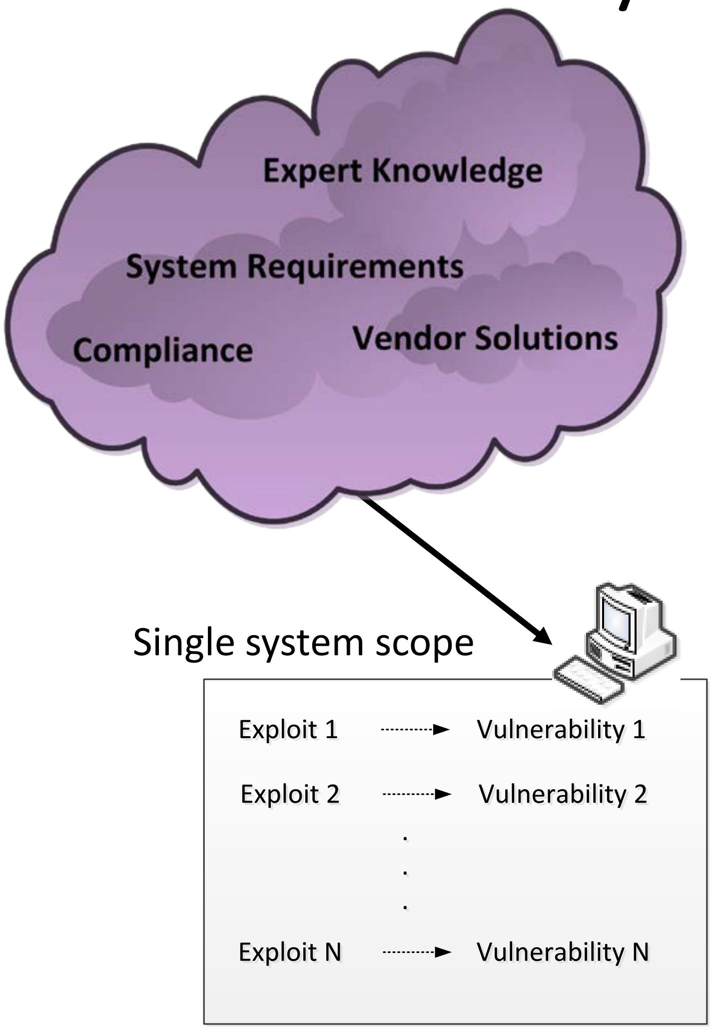
Unplanned connectivity and upgrades

Developers are not trained in software engineering

Programming for security is not the same as programming for safety

Safety strategy	Security view
Rely on physical models in fault trees	Attackers do not obey the laws of physics
Redundancy mitigates single failures	Attackers are not independent events
Fault trees collectively exhaustive	Attack trees depend on adversaries' creativity
Steady state behavior indicator of proper operation	APT (Advanced persistent threats) hide in steady state behavior
Deteriorating performance predicts maintenance for safety	Attackers cover their tracks
Microcontrollers and air gaps implement boundaries	Side channels open vulnerabilities

Need for multisystem risk analysis



Risk analysis is focused on a single system

- Standalone (i.e., single system) models have been developed
- Risk analysis considers the exploit of an individual vulnerability within a single system

Security risk identification techniques do not consider:

- Compositions of multiple vulnerabilities
- Cross-system security events/risks
- Impacts beyond the exploit of a single system (to the intended service and organization)

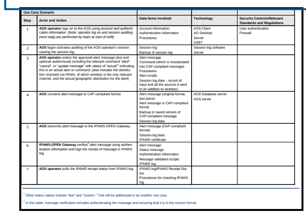
Need for systematic, multiple system evaluations

- Notation for expressing a security events and risks
- Take into account all context

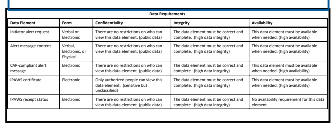
Security Engineering Risk Analysis approach

Comprehensive context

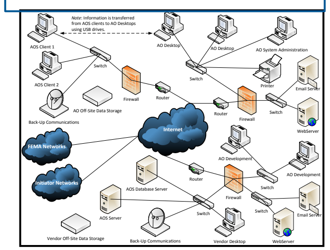
Use-Case View



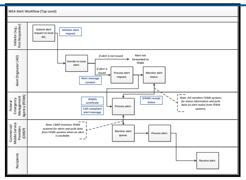
Data View



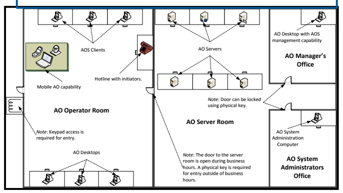
Network View



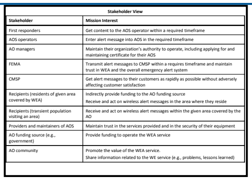
Workflow View



Physical View



Stakeholder View

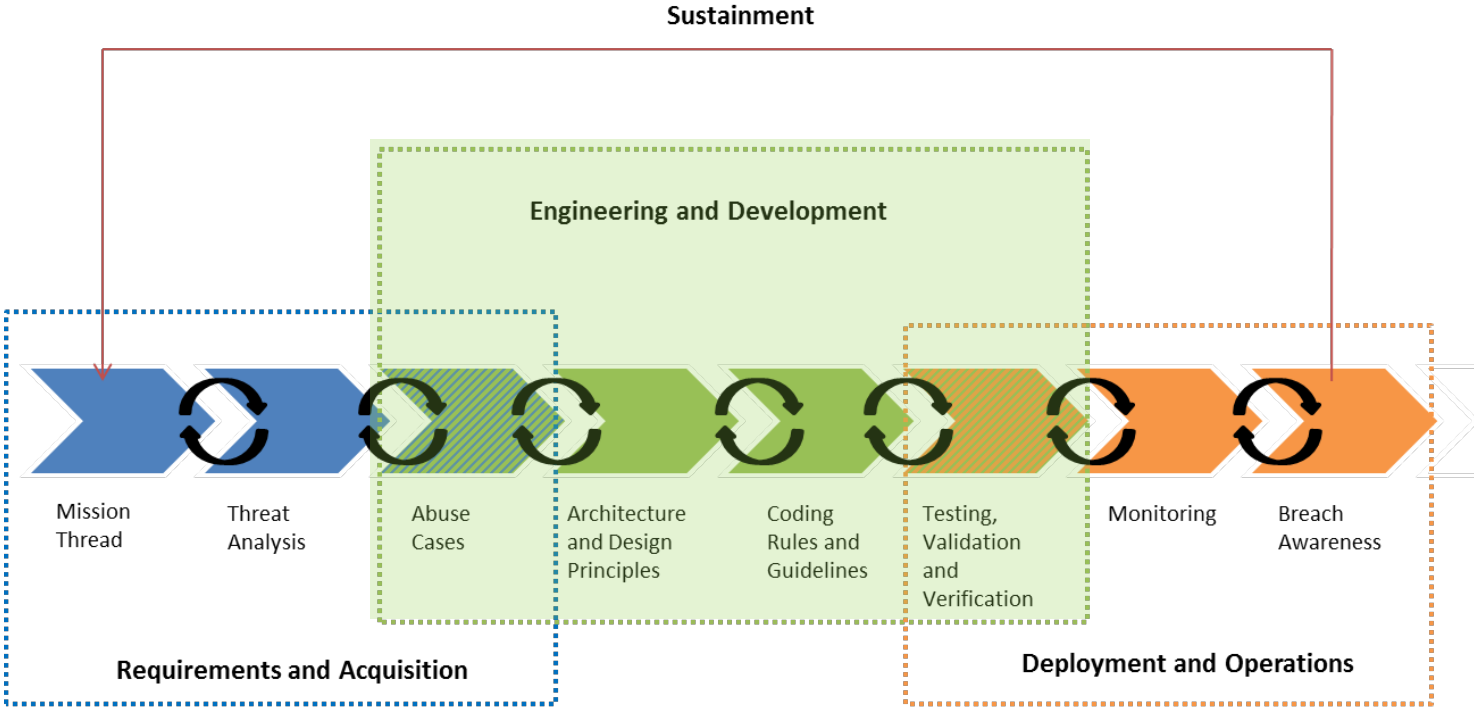


Determining actions

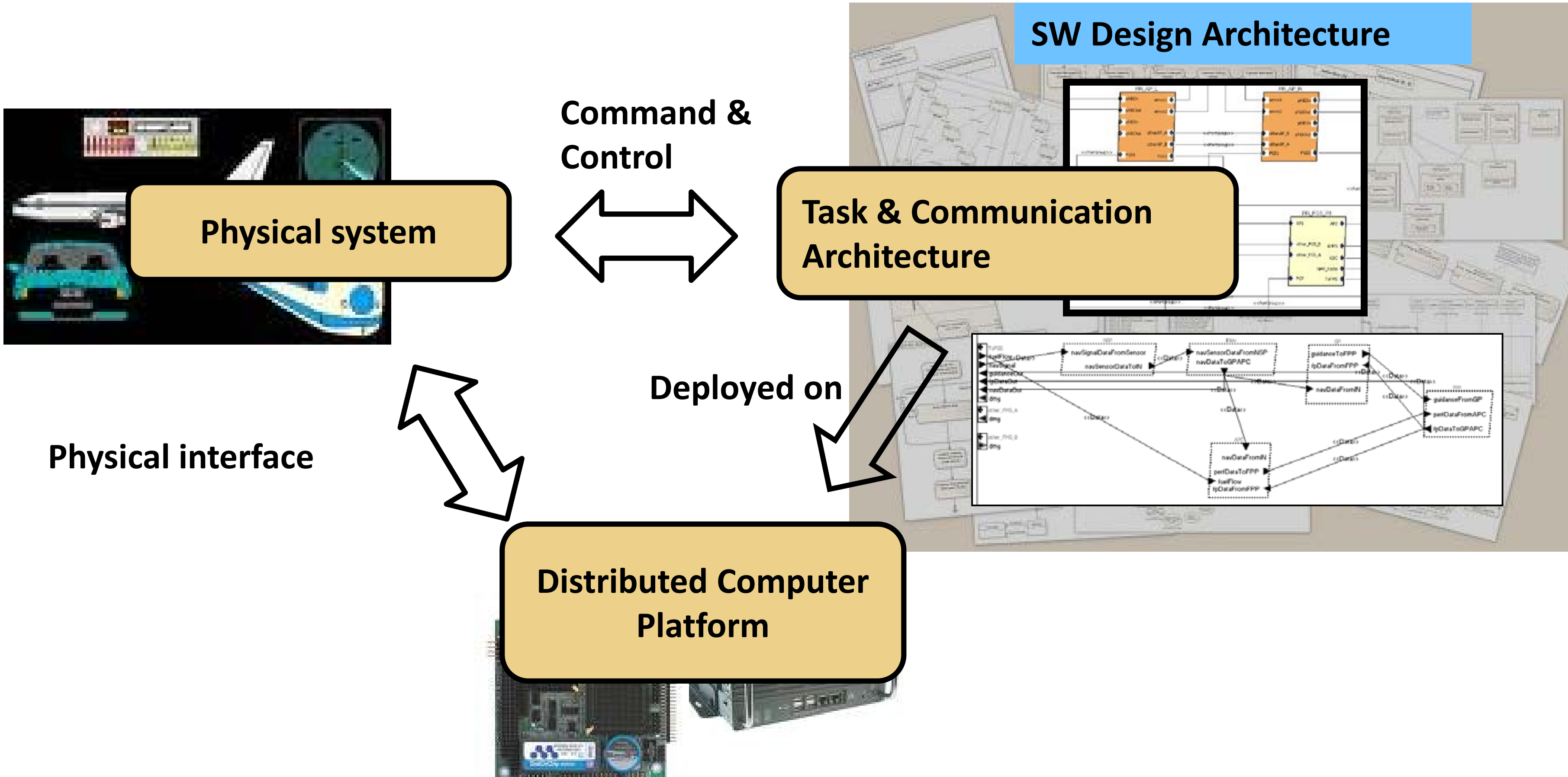
- Establish threat model
- Determine common system view
- Inspect connections between systems
- Evaluate
  - Consequences
  - Likelihood
  - Risk

http://resources.sei.cmu.edu/library/asset-view.cfm?assetid=427321

Development



Architecture Analysis & Design Language (AADL)



AADL Addresses Increasing Interaction Complexity and Mismatched Assumptions

Team Software Process

TSP is an agile, team-focused process for software and systems development.

The TSP strategy improves software engineering from the bottom up.

- Instills engineering discipline in software developers
- Builds high-performance trusted teams

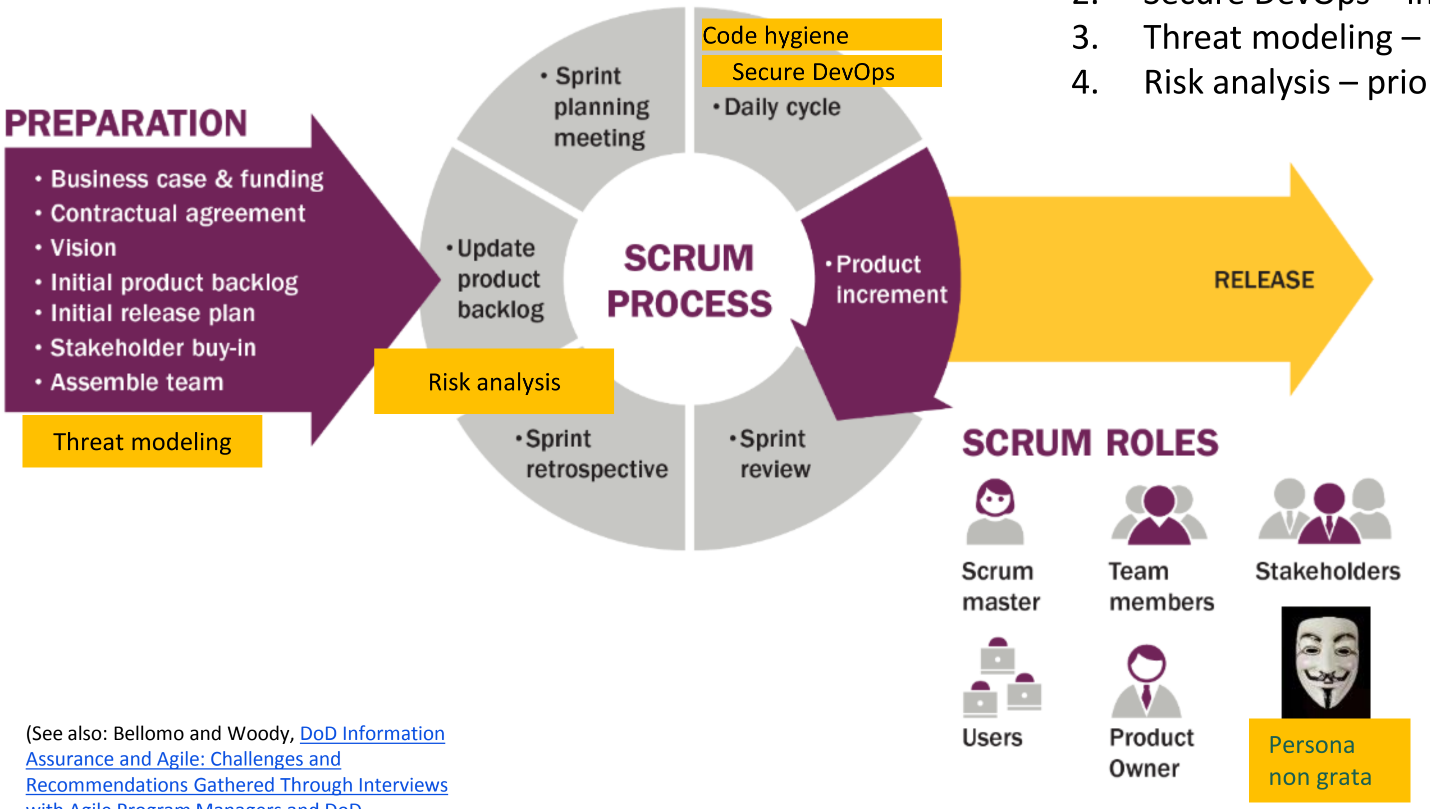
TSP works in practice

Performance Category	Typical TSP Result	Typical Industry Result
Effort estimation error	<10%	>30%
Schedule estimation error	<10%	>30%
Product quality (defects/KLOC)	0.01 to 0.5	1.0 to 7.0

Extending TSP with security

- Adding secure design
  - Minimize attack surfaces
  - Defense in depth for software development
- Adding secure coding
  - Adopting secure coding practices
- Tooling support for automated conformance checking
- Tracking security defects
  - Monitoring results of tests with respect to security

Integrating security into Agile (Scrum) development



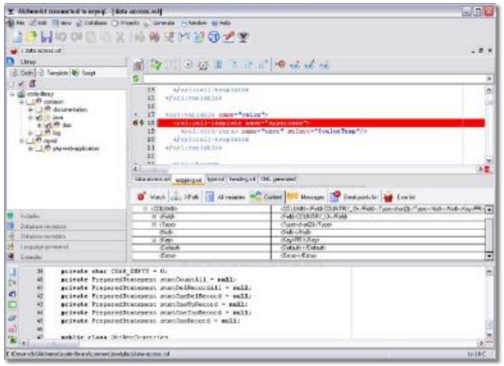
1. Code hygiene – introduce secure coding
2. Secure DevOps – include security tools
3. Threat modeling – represent a new role
4. Risk analysis – prioritize in backlog

(See also: Bellomo and Woody, DoD Information Assurance and Agile: Challenges and Recommendations Gathered Through Interviews with Agile Program Managers and DoD Accreditation Reviewers (http://repository.cmu.edu/cgi/viewcontent.cgi?article=1674&context=sei)

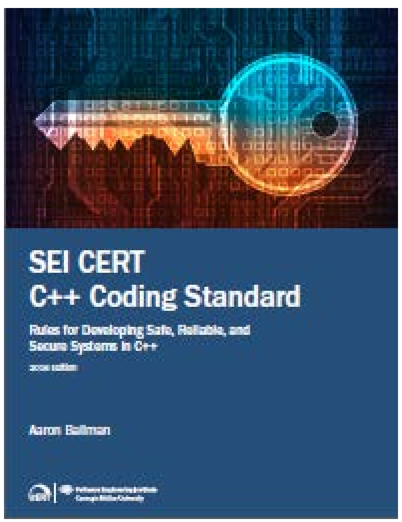
Adoption of secure coding rules

Training

Integrated development environments



CERT Secure Coding Standards



Collected wisdom from thousands of contributors on community wiki since Spring 2006

SEI CERT C Coding Standard

- Free PDF download:
  http://cert.org/secure-coding/products-services/secure-coding-download.cfm
- Basis for ISO TS 17961 C Secure Coding Rules

SEI CERT C++ Coding Standard

- Free PDF download (Released March 2017):
  http://cert.org/secure-coding/products-services/secure-coding-cpp-download-2016.cfm

CERT Oracle Secure Coding Standard for Java

“Current” guidelines available on CERT Secure Coding wiki

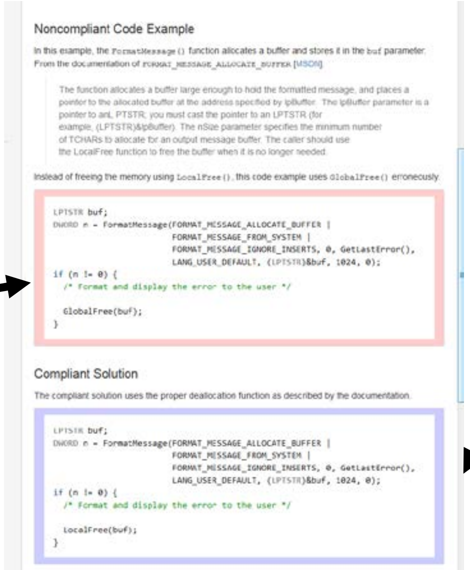
- https://www.securecoding.cert.org

Learning from rules and recommendations

Rules and recommendations in the secure coding standards focus to improve behavior

The “Ah ha” moment:

Noncompliant code examples or antipatterns in a pink frame—do not copy and paste into your code



Noncompliant Code Example

In this example, the `FormatMessage()` function allocates a buffer and stores it in the `buf` parameter. From the documentation of `FORMAT_MESSAGE_ALLOCATE_BUFFER` [MSDN]

> The function allocates a buffer large enough to hold the formatted message, and places a pointer to the allocated buffer at the address specified by lpBuffer. The lpBuffer parameter is a pointer to anL PTSTR; you must cast the pointer to an LPTSTR (for example, (LPTSTR)&lpBuffer). The nSize parameter specifies the minimum number of TCHARs to allocate for an output message buffer. The caller should use the LocalFree function to free the buffer when it is no longer needed.

Instead of freeing the memory using `LocalFree()`, this code example uses `GlobalFree()` erroneously.

```
LPTSTR buf;
DWORD n = FormatMessage(FORMAT_MESSAGE_ALLOCATE_BUFFER |
                        FORMAT_MESSAGE_FROM_SYSTEM |
                        FORMAT_MESSAGE_IGNORE_INSERTS, 0, GetLastError(),
                        LANG_USER_DEFAULT, (LPTSTR)&buf, 1024, 0);
if (n != 0) {
  /* Format and display the error to the user */

  GlobalFree(buf);
}
```

Compliant Solution

The compliant solution uses the proper deallocation function as described by the documentation.

```
LPTSTR buf;
DWORD n = FormatMessage(FORMAT_MESSAGE_ALLOCATE_BUFFER |
                        FORMAT_MESSAGE_FROM_SYSTEM |
                        FORMAT_MESSAGE_IGNORE_INSERTS, 0, GetLastError(),
                        LANG_USER_DEFAULT, (LPTSTR)&buf, 1024, 0);
if (n != 0) {
  /* Format and display the error to the user */

  LocalFree(buf);
}
```

Compliant solutions in a blue frame that conform with all rules and can be reused in your code

Secure Coding in C/C++ Training

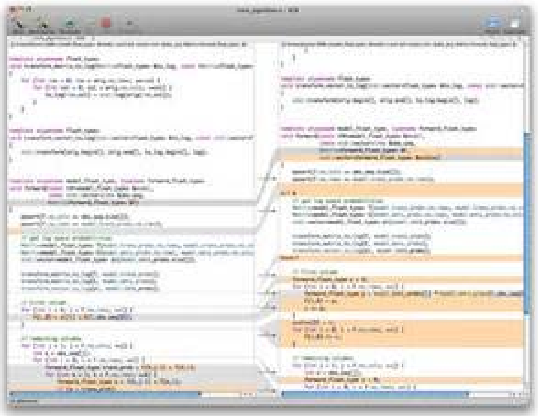
The Secure Coding course is designed for C and C++ developers. It encourages programmers to adopt security best practices and develop a security mindset that can help protect software from tomorrow's attacks, not just today's.

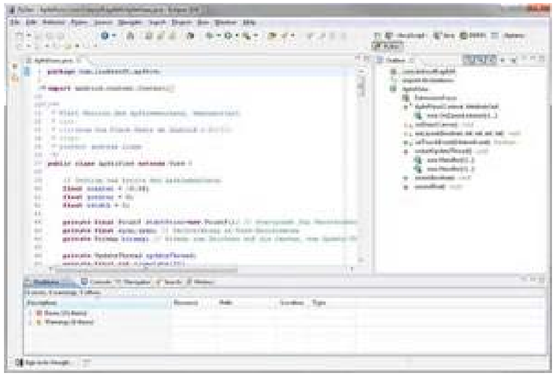
Topics

- String management
- Dynamic memory management
- Integral security
- Formatted output
- File I/O

Additional information at ttp://www.sei.cmu.edu/training/p63.cfm

Tools encourage application of secure coding





Moving rules into IDE improves application of secure coding

- Early feedback corrects errors on introduction
- Exceptions are understood in context
- Feedback improves developer skill

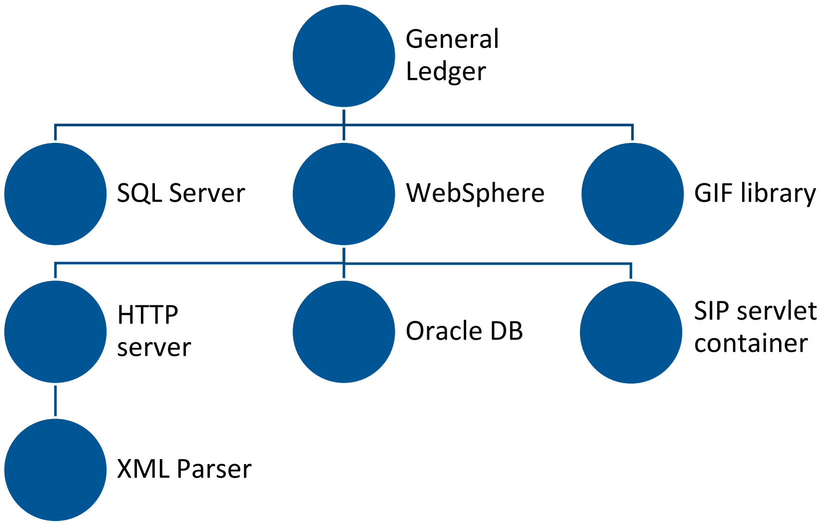
Target Clang static analyzer (C based languages)

- Widely used open source front end for popular compilers
- Integrated into Apple's Xcode IDE

Target FindBugs (Java)

- Integrated into Eclipse and JDeveloper

Software is more assembled than built

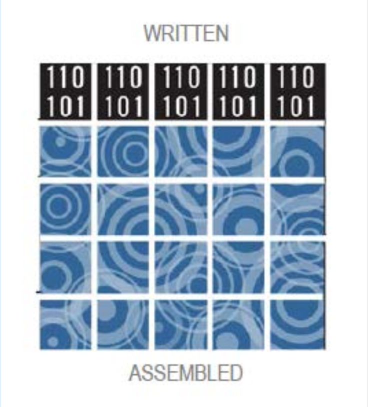


"Development" is now "assembly" using collective development

- Too large for single organization
- Too much specialization
- Too little value in individual components

Note: hypothetical application composition

The rise of open source



- 90% of modern applications are assembled from 3rd party components
- Most applications are now assembled from hundreds of open source components, often reflecting as much as 90% of an application
- At least 75% of organizations rely on open source as the foundation of their applications

Distributed development – context:

- Amortize expense
- Outsource non-differential features
- Lower acquisition (CapEx) expense

Sources: Geer and Corman, “Almost Too Big To Fail,” ;login: (Usenix), Aug 2014; Sonatype, 2014 open source development and application security survey

The rise of open source

- 90% of
  assem
  - At le
    as th
- Most applications are now assembled from hundreds of open source components, often reflecting as much as 90% of an application

Distributed development – context:

ntial

Ex)

“Developers are gorging themselves on an ever expanding supply of open source components”

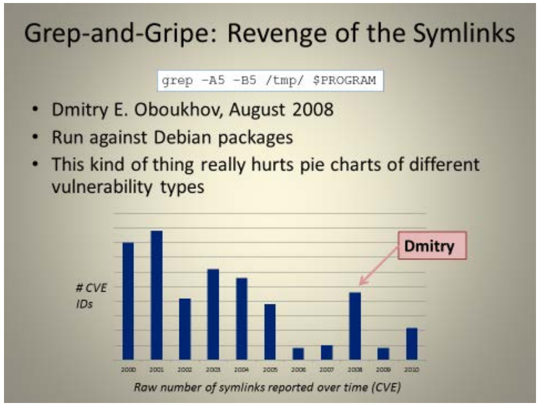
Sonatype, “2016 State of the Software Supply Chain”

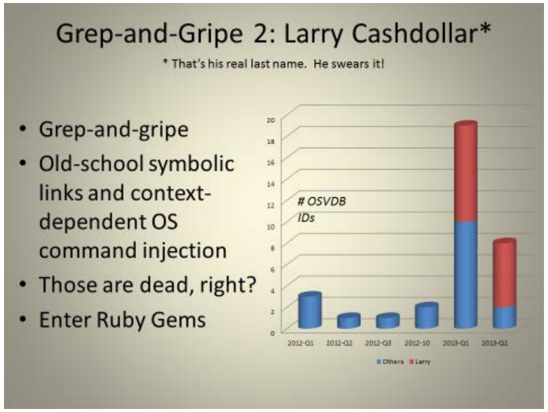
Sources: Geer and Corman, “Almost Too Big To Fail,” ;login: (Usenix), Aug 2014; Sonatype, 2014 open source development and application security survey

Open source is not secure

Heartbleed and Shellshock were found by exploitation

Other open source software illustrates vulnerabilities from cursory inspection





Sources: Steve Christey (MITRE) & Brian Martin (OSF), Buying Into the Bias: Why Vulnerability Statistics Suck, https://media.blackhat.com/us-13/US-13-Martin-Buying-Into-The-Bias-Why-Vulnerability-Statistics-Suck-Slides.pdf; Sonatype, Sonatype Open Source Development and Application Security Survey; Sonatype, 2016 State of the Software Supply Chain; Aspect Software "The Unfortunate Reality of Insecure Libraries," March 2012

Open sourc

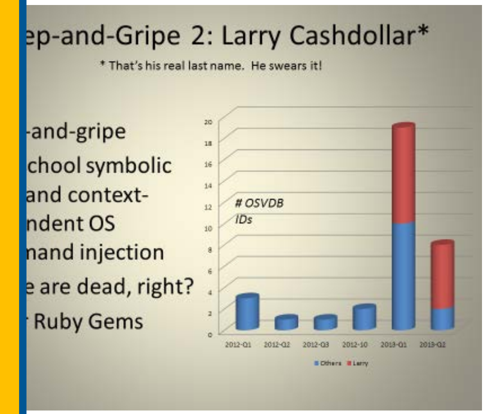
Heartbleed and
Shellshock were fou
by exploitation

Other open source
software illustrates
vulnerabilities from cu
inspection

1.8 billion vulnerable open source components downloaded in 2015

26% of the most common open source components have high risk vulnerabilities

On average, applications have 22.5 open source vulnerabilities



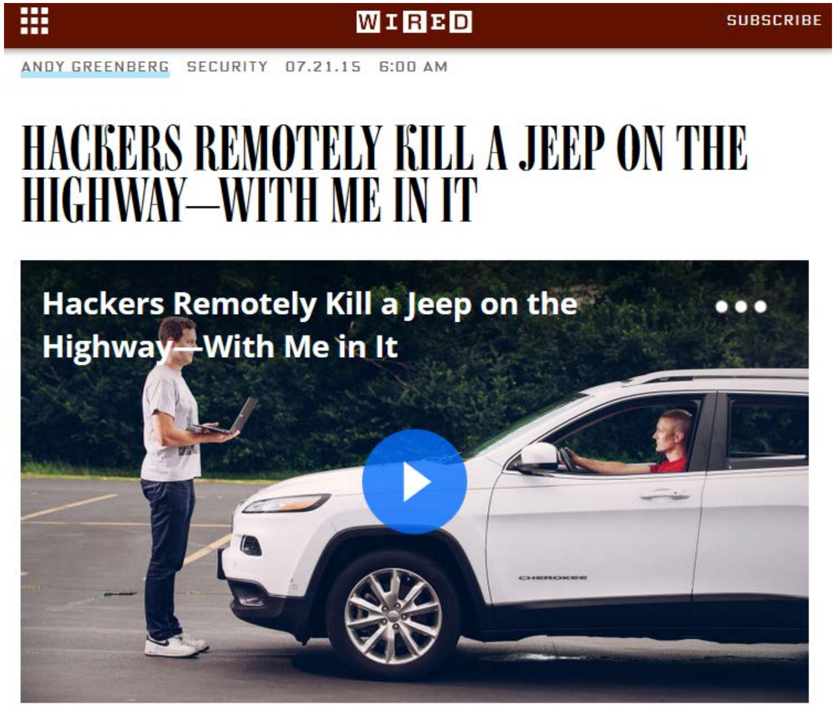
s Suck, https://media.blackhat.com/us-13/US-13-
ource Development and Application Security Survey;
ure Libraries," March 2012, Mike Pittenger, Black

Reducing software supply chain risk factors

Software supply chain risk for a product needs to be reduced to acceptable level

Supplier Capability	Product Security	Product Distribution	Operational Product Control
Supplier follows practices that reduce supply chain risks	Delivered or updated product is acceptably secure	Methods of transmitting the product to the purchaser guard again tampering	Product is used in a secure manner

Connecting automotive systems to internet opens system to attack



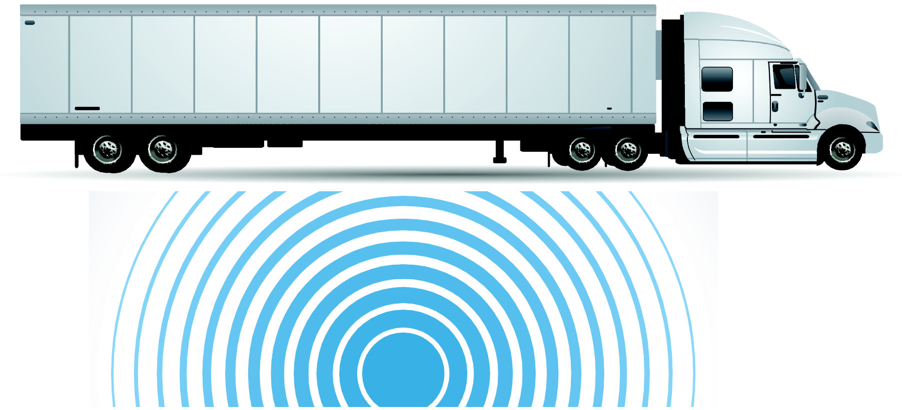
Extending systems opens vulnerabilities not anticipated

- Optimizations performed assuming one attack method
- Assumptions no longer hold with additional integrations

Studies suggest that new operational environments are a leading cause for introducing new vulnerabilities in existing systems.

Source: http://www.wired.com/2015/07/hackers-remotely-kill-jeep-highway/
Clark, Frei, Blaze, Smith, "Familiarity Breeds Contempt: The Honeymoon Effect and the Role of Legacy Code in Zero-Day Vulnerabilities," ACSAC '10 Dec. 6-10, 2010, p. 251-260."

Machine-learning based systems increase exposures



“the [Tesla] car's driverless technology failed to detect the white side of the tractor-trailer against a brightly lit sky, so the brake wasn't activated.”
-ABC7News, July 1, 2016

Operations are driven by high volume, high velocity sensor data

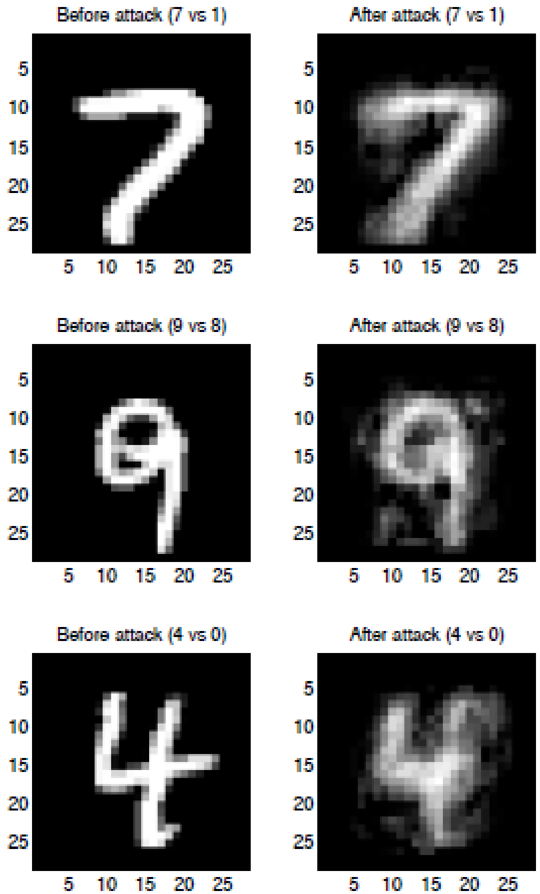
Decision making is based on “trained” models of behaviors

Conventional code development techniques of modest help

Understand the limits of training

Source: http://abc7news.com/automotive/tesla-self-driving-car-fails-to-detect-truck-in-fatal-crash/1410042/

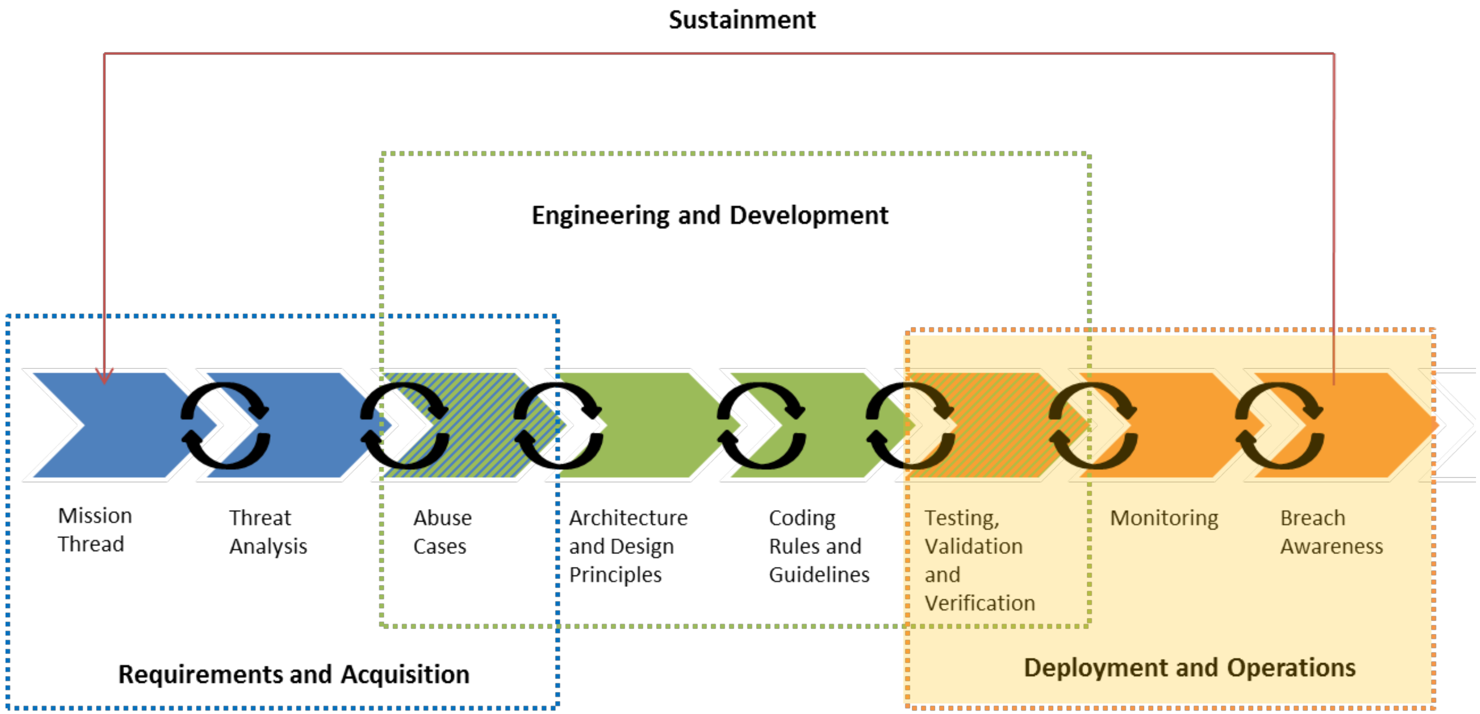
Recognizing and recovering poisoned systems



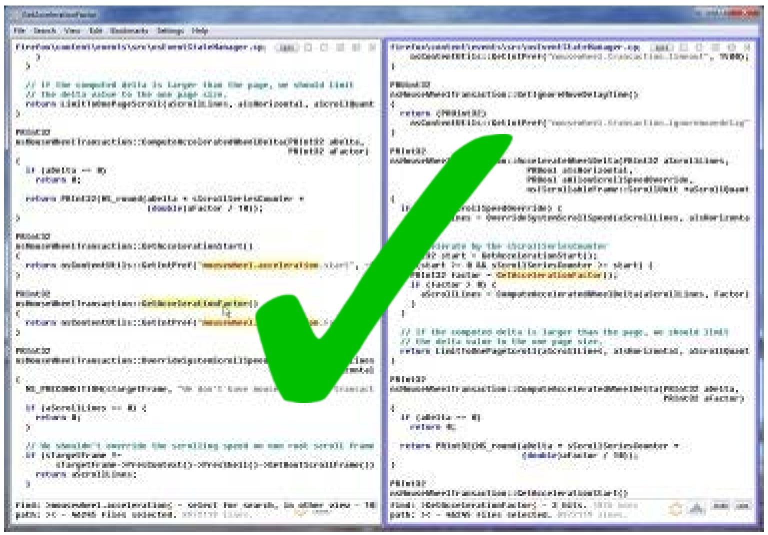
- “Chaff” and “noise” can emerge as vulnerabilities
- Defensive strategy based on “it is difficult to lie at scale”
- Tactics include consistency checks, such as
  - Multiple models in a single unit
  - Coordination among units
  - Coordination with environment

Source: Battista Biggio, Blaine Nelson, Pavel Laskov, Poisoning Attacks against Support Vector Machines, 2012, arxiv.org/abs/1206.6389

Deployment and operations



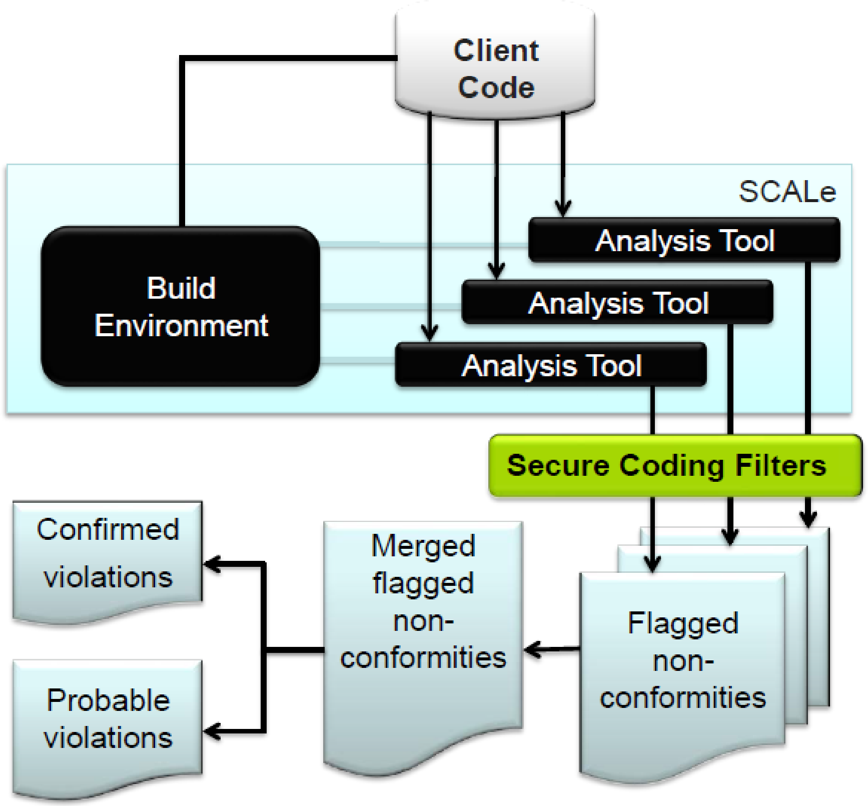
Static Testing – Source code analysis tools



Secure Code Analysis Laboratory (SCALe)

- C, C++, Java, PERL, Python, Android rule conformance checking
- Thread safety analysis
- Information flows across Android applications
- Operating system call flows

SCALe Multitool evaluation



Improve expert review productivity by focusing on high priority violations

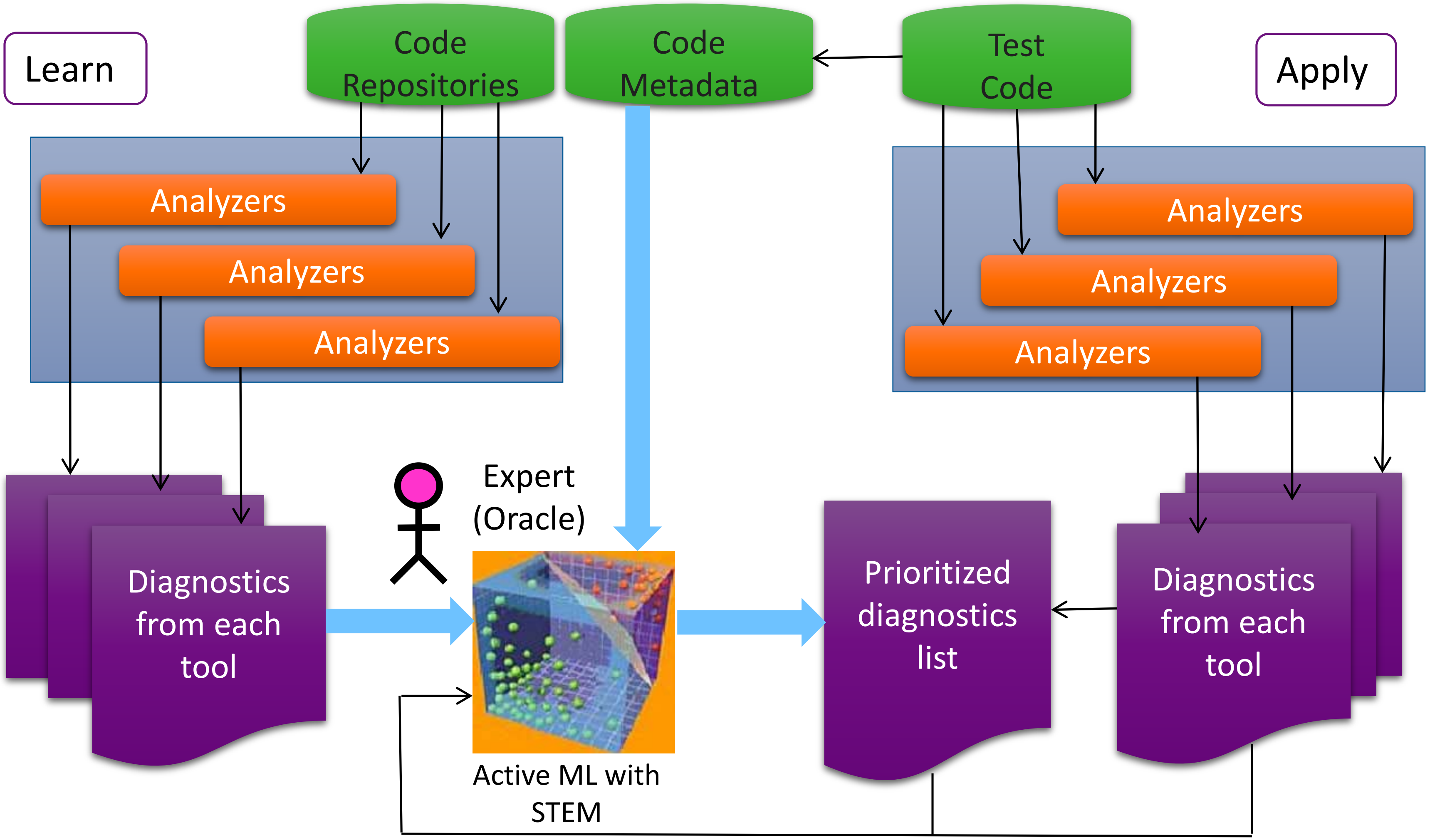
Filter select secure coding rule violations

- Eliminate irrelevant diagnostics
- Convert to common CERT Secure Coding rule labeling

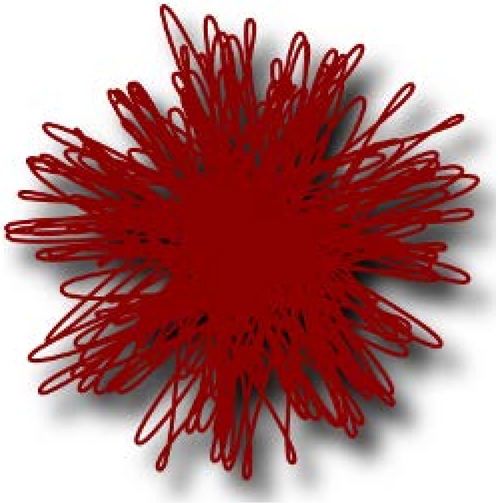
Single view into code and all diagnostics

Maintain record of decisions

Optimizing multitool evaluations



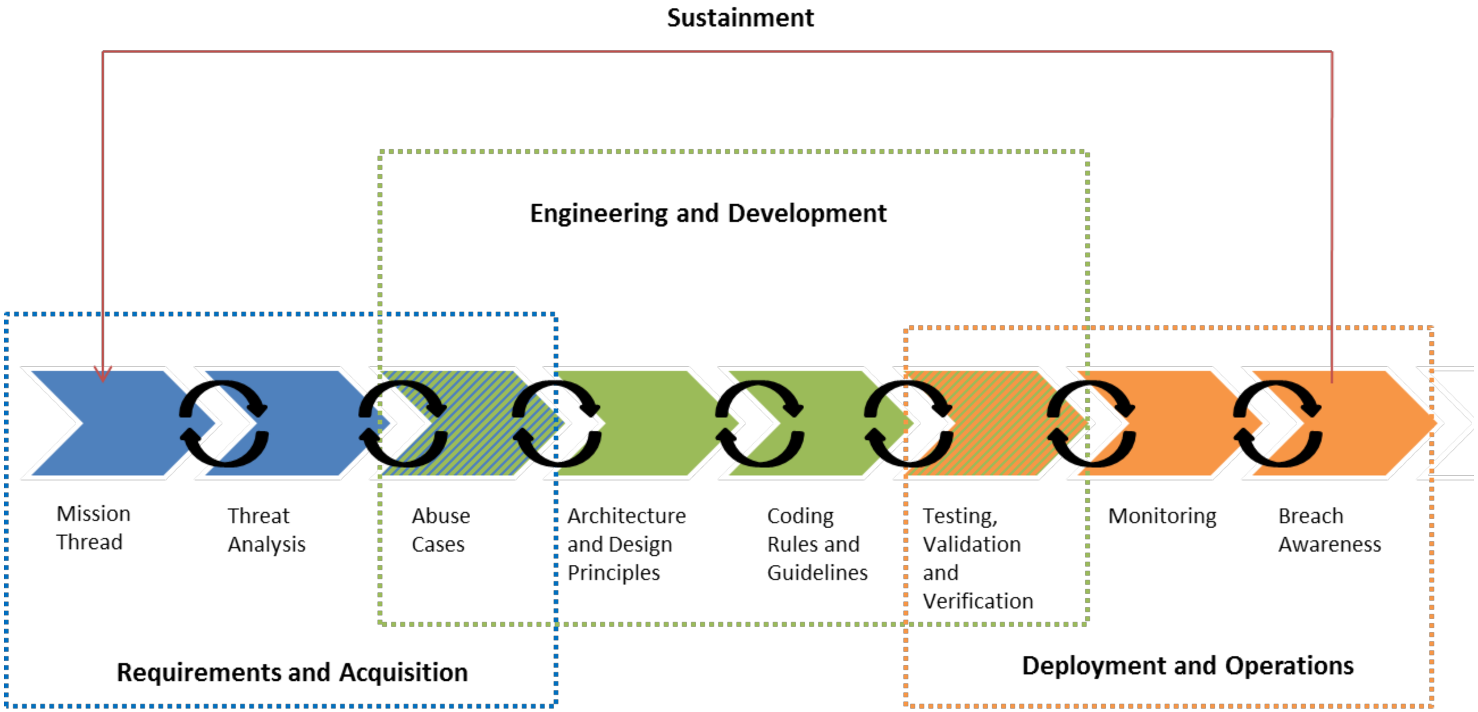
Dynamic testing and evaluation – fuzzing



Fuzz testing of attack surfaces

- Based on techniques used in CERT's Basic Fuzzing Framework (BFF)
- mutational fuzzing
- machine learning and evolutionary computing techniques
- adjusts its configuration parameters based on what it finds (or does not find) over the course of a fuzzing campaign

Review: Secure Software Development Lifecycle



Automation; Acquisition (Supply chain); Building skills (Workforce development); Metrics, Models, and Measurement

Software Assurance Framework

Mission Ready Diagnostics; Threat Modeling; SQUARE; Security Engineering Risk Analysis

Architecture Analysis & Design Language

Team Software Process; Secure TSP; Secure Agile; Secure Coding; SCALe

Run time support; Vulnerability Analysis

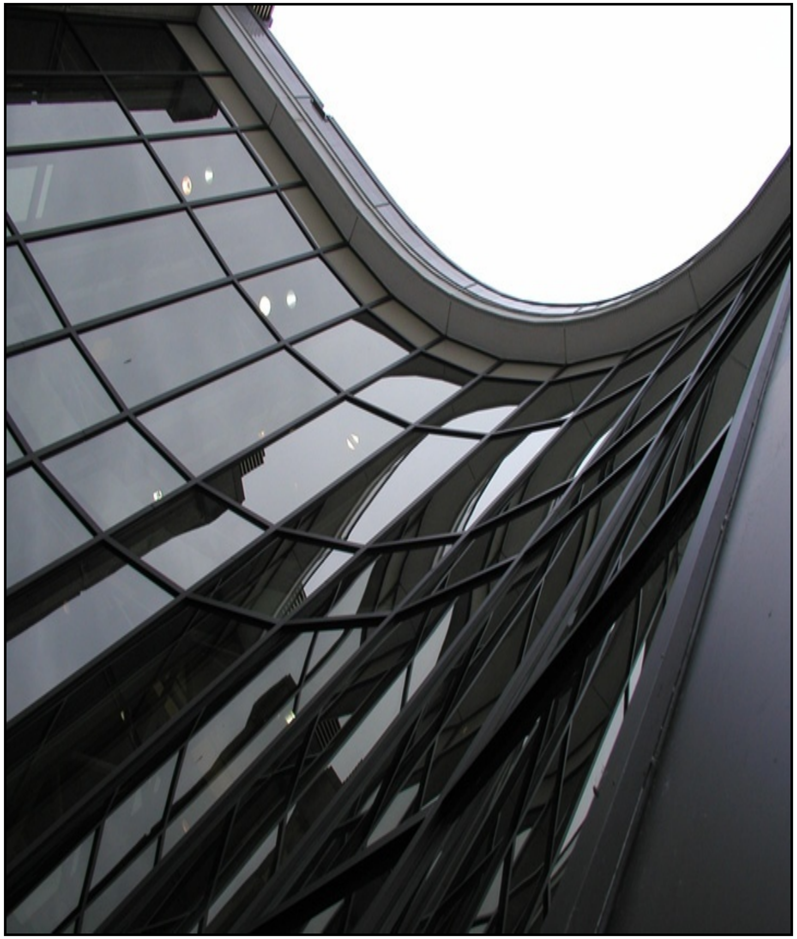
Forensic Operations & Investigations

Contact Information

Mark Sherman

(412) 268-9223

mssherman@sei.cmu.edu



Web Resources (CERT/SEI)

http://www.cert.org/

http://www.sei.cmu.edu/