Building Secure Software for Mission Critical Systems
Mark Sherman, PhD
Technical Director, CERT

Software Solutions Conference 2015
November 16–18, 2015
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
  - 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
Complex software is business and mission critical

Evolution of avionics size and function from F-4A (1960) to F-35 (2000)

Sources: Final Report, NASA Study on Flight Software Complexity
Vehicle technology following the same path

2010 Jeep Cherokee
(12 ECUs)

2014 Jeep Cherokee
(32 ECUs)

Common assertion that modern high end vehicles have

- Over 50 antennas
- Over 80 ECUs
- Over 100M lines of code

https://en.wikipedia.org/wiki/Electronic_control_unit
Software vulnerabilities are ubiquitous
New slide before 14 -- what's the external view why we need to get better and fast
Mark S. Sherman, 7/13/2015
Cyber attacks on physical systems

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.

Dragonfly attacks a dozen companies
The Dragonfly hacker group attacked a number of companies' SCADA systems and installed the malware 'Havex'. This was used to gather information about the systems. No damage was done, because the compromise was detected and removed before the hackers had completed the observation and intelligence gathering phase.

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


Average cost in a breach:

US$188 per record

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

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

Toyota Sudden Acceleration Defect Case: $1.1 Billion Settlement
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.”


Catching software faults early saves money
Faults accounts for 30–50% percent of total software project costs

Software Development Lifecycle

<table>
<thead>
<tr>
<th>Where Faults are Introduced</th>
<th>70%</th>
<th>20%</th>
<th>10%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirements Engineering</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>System Design</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Software Architectural Design</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Component Software Design</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Code Development</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Unit Test</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Integration</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>System Test</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Acceptance Test</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Operation</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Where Faults are Found</th>
<th>3.5%</th>
<th>16%</th>
<th>50.5%</th>
<th>9%</th>
<th>20.5%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nominal Cost Per Fault for Fault Removal</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Cost Per Fault for Fault Removal: 300–1000x

Sources: Critical Code; NIST, NASA, INCOSE, and Aircraft Industry Studies
Security is a lifecycle issue
Room for improvement

19% fail to carry out security requirement definition
27% do not practice secure design
72% do not use code or binary analysis
47% do not perform acceptance tests for third-party code

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

There is a wide range of application security quality

Source: CERT sample of evaluated programs
Requirements

Sustainment

Engineering and Development

Deployment and Operations

Requirements and Acquisition

Mission Thread

Threat Analysis

Abuse Cases

Architecture and Design Principles

Coding Rules and Guidelines

Testing, Validation and Verification

Monitoring

Breach Awareness

Software Engineering Institute | Carnegie Mellon University
Threat analysis tools help derive abuse and misuse cases

Microsoft SDL Threat Modeling Tool

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

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

<table>
<thead>
<tr>
<th>STRIDE Threat Types</th>
<th>Threat</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Authentication</td>
<td>Spoofing</td>
<td>Impersonating something or someone else</td>
</tr>
<tr>
<td>Integrity</td>
<td>Tampering</td>
<td>Modifying code or data without authorization</td>
</tr>
<tr>
<td>Non-repudiation</td>
<td>Repudiation</td>
<td>The ability to claim not to have performed an action against an application</td>
</tr>
<tr>
<td>Confidentiality</td>
<td>Disclosure</td>
<td>The exposure of information to unauthorized users</td>
</tr>
<tr>
<td>Availability</td>
<td>Denial of Service</td>
<td>The ability to deny or degrade a service to legitimate users</td>
</tr>
<tr>
<td>Authorization</td>
<td>Elevation of Privilege</td>
<td>The ability of a user to elevate their privileges with an application without authorization</td>
</tr>
</tbody>
</table>
Embedded systems represent new classes of vulnerabilities

Embedded systems have different characteristics than IT systems

- Size, weight, power and latency concerns
- Unique architectures of embedded controllers
- Bit and clock cycle level operations
- Physical resources with real time sensors
- Safety-Critical Real-time OS
- Intermittent communications
- Multiple command-and-control masters
- Embedded firmware,
- Unique internal busses & controllers
- Developers are engineers, not software developers
Security approaches for IT systems do not cover embedded system security

Responses to attack surfaces and threat models not generally reusable

- 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
- Larger number of attack surfaces
- More diverse attack surfaces
- Maintenance backdoors
- Hardcoded credentials
- Unique and insecure protocols
- Unplanned connectivity and upgrades
Programming for security is not the same as programming for safety

<table>
<thead>
<tr>
<th>Safety strategy</th>
<th>Security view</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rely on physical models in fault trees</td>
<td>Attackers do not obey the laws of physics</td>
</tr>
<tr>
<td>Redundancy mitigates single failures</td>
<td>Attackers are not independent events</td>
</tr>
<tr>
<td>Shared, global state improves behavior</td>
<td>Attackers use leaked information beyond intended purposes</td>
</tr>
<tr>
<td>Shared service containers to meet space, power and weight constraints</td>
<td>Coupled services enable denial of service attacks</td>
</tr>
<tr>
<td>Microcontrollers and air gaps implement boundaries</td>
<td>Side channels open vulnerabilities</td>
</tr>
</tbody>
</table>
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

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
Integrating security into Agile 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
Coding rules

- Collected wisdom of programmers and tools vendors
  - Fed by community wiki started in Spring 2006
  - 1,576 registered contributors
- Basis for ISO Standard
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.

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 depends on a supply chain of components

Development is largely assembly

Collective development – context:
• Too large for single organization
• Too much specialization
• Too little value in individual components

Note: hypothetical application composition
Substantial open source contained in supply chain

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

Distributed development – context:
- Amortize expense
- Outsource non-differential features
- Lower acquisition (CapEx) expense

Source: Sonatype, 2014 Sonatype 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

46 million vulnerable open source components downloaded annually

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

Open source supply chain has a long path
Corruption in the tool chain already exists

- XcodeGhost corrupted Apple's development environment
  - Major programs affected
    - WeChat
    - Badu Music
    - Angry Birds 2
    - Heroes of Order & Chaos
    - iOBD2

Sources: http://www.macrumors.com/2015/09/24/xcodeghost-top-25-apps-apple-list/
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

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.

Need to manage software supply chain

Software Supply Chain risk for a product needs to be reduced to acceptable level

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

Static testing optimization
- SCALe set up
- SCALe filters and visualizer
- Tool conformance and capability testing
- Multitool integration and statistical optimizer
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
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 & Investigations

Forensic Operations & Investigations

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

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

Software Assurance Framework
Contact Information

Mark Sherman
(412) 268-9223
mssherman@sei.cmu.edu

Web Resources (CERT/SEI)
http://www.cert.org/
http://www.sei.cmu.edu/