Building Secure Software for Mission Critical Systems

Mark Sherman, PhD
Technical Director, CERT

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213
Copyright 2017 Carnegie Mellon University

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the United States Department of Defense.

References herein to any specific commercial product, process, or service by trade name, trade mark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by Carnegie Mellon University or its Software Engineering Institute.

NO WARRANTY. THIS CARNEGIE MELLON UNIVERSITY AND SOFTWARE ENGINEERING INSTITUTE MATERIAL IS FURNISHED ON AN "AS-IS" BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT.

[Distribution Statement A] This material has been approved for public release and unlimited distribution. Please see Copyright notice for non-US Government use and distribution.

This material may be reproduced in its entirety, without modification, and freely distributed in written or electronic form without requesting formal permission. Permission is required for any other use. Requests for permission should be directed to the Software Engineering Institute at permission@sei.cmu.edu.

Carnegie Mellon® and CERT® are registered marks of Carnegie Mellon University.

Team Software Process℠ and TSP℠ are service marks of Carnegie Mellon University.

DM-0004576
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

Software vulnerabilities are ubiquitous
## Existing Customer Premise Equipment (SOHO) typically vulnerable

<table>
<thead>
<tr>
<th>Vulnerability</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cross-site request forgery (CSRF)</td>
<td>54%</td>
</tr>
<tr>
<td>Non-unique default credentials</td>
<td>85%</td>
</tr>
<tr>
<td>DNS spoofing attacks</td>
<td>63%</td>
</tr>
<tr>
<td>Router firmware versions</td>
<td>100%</td>
</tr>
</tbody>
</table>

- 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.

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: 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

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

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.”
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

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.”


Software and security failures are expensive

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


**Average cost in a breach:**

US$188 per record

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


**Source:** New York Times, Jan 10, 2014

**Source:** Ponemon Institute, “2013 Cost of Data Breach Study: Global Analysis”, May 2013
Catching software faults early saves money

Faults account 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>

| Where Faults are Found      | 3.5% | 16% | 50.5% | 9% | 20.5% |

Nominal Cost Per Fault for Fault Removal

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

Sustainment

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

Mission thread (Business process)
Threat Analysis
Abuse Cases
Architecture and Design Principles
Coding Rules and Guidelines
Testing, Validation and Verification
Monitoring
Breach Awareness

Requirements and Acquisition
Deployment and Operations

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

Requirements
Threat analysis tools help derive abuse and misuse cases

**Microsoft SDL Threat Modeling Tool**

<table>
<thead>
<tr>
<th>STRIDE Threat Types</th>
<th>Desired Property</th>
<th>Threat</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Authentication</td>
<td>Spoofing</td>
<td>Impersonating someone or something else</td>
<td></td>
</tr>
<tr>
<td>Integrity</td>
<td>Tampering</td>
<td>Modifying code or data without authorization</td>
<td></td>
</tr>
<tr>
<td>Non-repudiation</td>
<td>Repudiation</td>
<td>The ability to claim to have not performed some action against an application</td>
<td></td>
</tr>
<tr>
<td>Confidentiality</td>
<td>Information Disclosure</td>
<td>The exposure of information to unauthorized users</td>
<td></td>
</tr>
<tr>
<td>Availability</td>
<td>Denial of Service</td>
<td>The ability to deny or degrade a service to legitimate users</td>
<td></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>
<td></td>
</tr>
</tbody>
</table>

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

<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>Fault trees collectively exhaustive</td>
<td>Attack trees depend on adversaries’ creativity</td>
</tr>
<tr>
<td>Steady state behavior indicator of proper operation</td>
<td>APT (Advanced persistent threats) hide in steady state behavior</td>
</tr>
<tr>
<td>Deteriorating performance predicts maintenance for</td>
<td>Attackers cover their tracks</td>
</tr>
<tr>
<td>safety</td>
<td></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

[Link to resource: http://resources.sei.cmu.edu/library/asset-view.cfm?assetid=427321]
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

<table>
<thead>
<tr>
<th>Performance Category</th>
<th>Typical TSP Result</th>
<th>Typical Industry Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Effort estimation error</td>
<td>&lt;10%</td>
<td>&gt;30%</td>
</tr>
<tr>
<td>Schedule estimation error</td>
<td>&lt;10%</td>
<td>&gt;30%</td>
</tr>
<tr>
<td>Product quality (defects/KLOC)</td>
<td>0.01 to 0.5</td>
<td>1.0 to 7.0</td>
</tr>
</tbody>
</table>
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):

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

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 http://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

Distributed development – context:
- 90% of modern applications are assembled from 3rd party components
- 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

“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

Open source

Heartbleed and Shellshock were found by exploitation

Other open source software illustrates vulnerabilities from cursory 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

Reducing software supply chain risk factors

Software supply chain risk for a product needs to be reduced to an 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
Connecting automotive systems to internet opens system to attack

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


Machine-learning based systems increase exposures

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

“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

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

Deployment and operations

Sustainment

Engineering and Development

Requirements and Acquisition

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

Learn

- Code Repositories
  - Analyzers
  - Analyzers
  - Analyzers

- Code Metadata

- Test Code
  - Analyzers
  - Analyzers
  - Analyzers

Apply

- Expert (Oracle)

- Active ML with STEM

- Prioritized diagnostics list

- Diagnostics from each tool

- Diagnostics from each tool
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; Forensic Operations & Investigations

Sustainment

Engineering and Development

Requirements and Acquisition

Deployment and Operations

- Mission Thread
- Threat Analysis
- Abuse Cases
- Architecture and Design Principles
- Coding Rules and Guidelines
- Testing, Validation and Verification
- Monitoring
- Breach Awareness

- Architecture Analysis & Design Language
- Team Software Process; Secure TSP; Secure Agile; Secure Coding; SCALe
- Run time support; 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/