Copyright 2016 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.

DM-0004178
Software is advancing function and replacing hardware

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

Avionics SLOC

% Airplane Function in Software

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

Software vulnerabilities are ubiquitous
Automotive vulnerabilities were predicted

Emerging Technology Domains Risk Survey

Christopher King
Jonathan Chu
Andrew Mellinger

April 2015

Automotive vulnerabilities were realized

Source: http://www.wired.com/2015/07/hackers-remotely-kill-jeep-highway/
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>

Where Faults are Found

| 3.5% | 16% | 50.5% | 9% | 20.5% |

Nominal Cost Per Fault for Fault Removal

Cost Per Fault for Fault Removal 300–1000x

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


Need to avoid false sense of security

Source: http://xkcd.com/1695/
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.”


Security is a lifecycle issue

Mission thread (Business process)
Threat Analysis

Abuse Cases

Architecture and Design Principles

Coding Rules and Guidelines

Testing, Validation and Verification

Monitoring

Deployment and Operations

Breach Awareness

Requirements and Acquisition

Sustainment
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.

Requirements
Getting the right requirements: desire for backdoors conflicts with secure operations

Helpful capability

Backdoor vulnerability
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: operational and physical, data, workflow, stakeholder, network views
Security Engineering Risk Analysis

1. Establish operational context.
2. Identify risk.
3. Analyze risk.
4. Develop control plan.
Engineering and Development
Architecture Analysis & Design Language (AADL)

AADL Addresses Increasing Interaction Complexity and Mismatched Assumptions

Command & Control

Task & Communication Architecture

Physical system

Physical interface

Distributed Computer Platform

SW Design Architecture
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)
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
- Downloadable 2016 edition

Download link:
http://www.cert.org/secure-coding/products-services/secure-coding-download.cfm
Adoption of secure coding rules

Training

Integrated development environments

Batch analyzers

Automated transformation and remediation
Threat modeling methodologies vary in coverage

Take-aways:
- The “new” approaches perform similarly, finding just about 20% of the relevant threat types on average. Both performed much better than the state of the practice approach, STRIDE.
- STRIDE has the most outliers – leaves more up to the experience of the teams?
- STRIDE has a longer tail above the mean. It seems to get most teams to a certain floor level of performance but allows more teams to do better?


Take-aways:
- PNG misses several classes of threats altogether. However, for the classes it DOES find, teams are consistently more likely to find them.
- SecCards can find a greater variety of threats – but each team is likely to report only a small subset, with lots of variation among teams.
- STRIDE is somewhere in between. Like PNG, it seems to allow more consistent results for the threats that it does find. The distribution is skewed to have a long tail above the mean, meaning that there are some types of threats that truly are found by almost all teams applying STRIDE.
Embedded Systems (ES) represent new classes of vulnerabilities and risk

Characteristics of Embedded Systems

ECU designed for a specific purpose – architecture could be unique for each embedded system
Size, Weight, Power and Latency concerns
  • Watchdog and filtering processes may not fit in operational envelop
Designing ES and code is a special field
  • Subject matter expertise of unique system
  • Autonomous systems have physical resources, navigation needs and Safety-Critical Real-time OS
  • Intermittent communications and multiple command-and-control masters
  • Embedded firmware, unique internal busses & controllers
  • Can require specific skills at the bit and clock cycle level

ES cyber defenses differ from PC/IT network cyber defenses

Network-centric cyber defenses have limited applicability to embedded systems
  • Virus definitions and operating guidelines don’t always apply
  • Centralized account control not possible
  • Network tools and assessment techniques have limited relevance to embedded systems architecture and interfaces

Threat Mitigation for ES and PC/IT are quite different
  • Larger number and more diverse attack surfaces
  • Back doors (maintenance), hardcoded credentials, 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 provides single world-view</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>
Machine-learning based systems increase exposures

Operations are driven by sensor data with large ranges of values

Decision making is based on “trained” models of behaviors

Conventional analysis of code of modest help

Understand the limits of training
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
Cross lifecycle issues

Sustainment

Engineering and Development

Deployment and Operations

Requirements and Acquisition

Automation (DevOps)

Metrics, Models, and Measurement

Building skills (Workforce development)

Procurement / Acquisition (Supply chain)
Evolution of software development

Custom development – context:
- Software was limited
  - Size
  - Function
  - Audience
- Each organization employed developers
- Each organization created their own software

Shared development – ISVs (COTS) – context:
- Function largely understood
  - Automating existing processes
- Grown beyond ability for using organization to develop economically
- Outside of core competitiveness by acquirers

Supply chain: practically none
Supply chain: software supplier
Development is now assembly

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

Supply chain: long

Note: hypothetical application composition
Vulnerabilities emerge in existing code

Defects in functionality found early and in new code

Vulnerabilities found in legacy code and late ("honeymoon effect")

New operating environments are a major cause of vulnerabilities

**Substantial open source contained in supply chain**

- 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

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

**Supply chain: opaque**

Sources: Geer and Corman, “Almost Too Big To Fail,” ;login: (Usenix), Aug 2014; Sonatype, 2014 open source development and application security survey
Open source supply chain has a long path.
Corruption along the supply chain is easy

Unexpected or unintended behaviors in components

Knowledgeable analysts can convert packaged binary into malware in minutes

Sources: Pedro Candel, Deloitte CyberSOC Academy, Deloitte
http://www.8enise.webcastlive.es/webcast.htm?video=08; http://www.microsoft.com/Products/Games/FSInsider/freeflight/PublishingImages/scene.jpg;
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/
Versions of Android illustrate open source fragmentation

Source: http://opensignal.com/reports/fragmentation.php
Open source is not secure

Heartbleed and Shellshock were found by exploitation

Other open source software illustrates vulnerabilities from cursory inspection

Open source is not secure

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

Reducing software supply chain risk factors

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
Assessment: Software Assurance Framework

**What**
- Defines software assurance practices for acquiring and developing assured software products

**Why**
- Improve software assurance practices in acquisition programs
- Enhance software assurance services provided by third parties

**Benefits**
- Establish confidence in a program’s ability to acquire software-reliant systems across the life cycle and supply chain
- Reduce cybersecurity risk of deployed software-reliant systems
Nine Practice Areas

1. Governance Infrastructure Practices

2. Materiel Solution Analysis (MSA) Practices (Alternatives)
3. Technology Development (TD) Practices (Gaps)
4. Engineering and Manufacturing Development (EMD) Practices (Pilot)
5. Production and Deployment (PD) Practices (Production)
6. Operations and Support (O&S) Practices

7. Secure Software Development Practices
8. Secure Software Operation Practices

Focus
- Governance Infrastructure
- Acquisition Lifecycle Assurance
- Software Security
- Software Security Infrastructure
Security is a lifecycle issue
Contact Information

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

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