Carnegie Mellon University

From Secure Coding to Secure Software

Robert Schiela

Mark Sherman

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

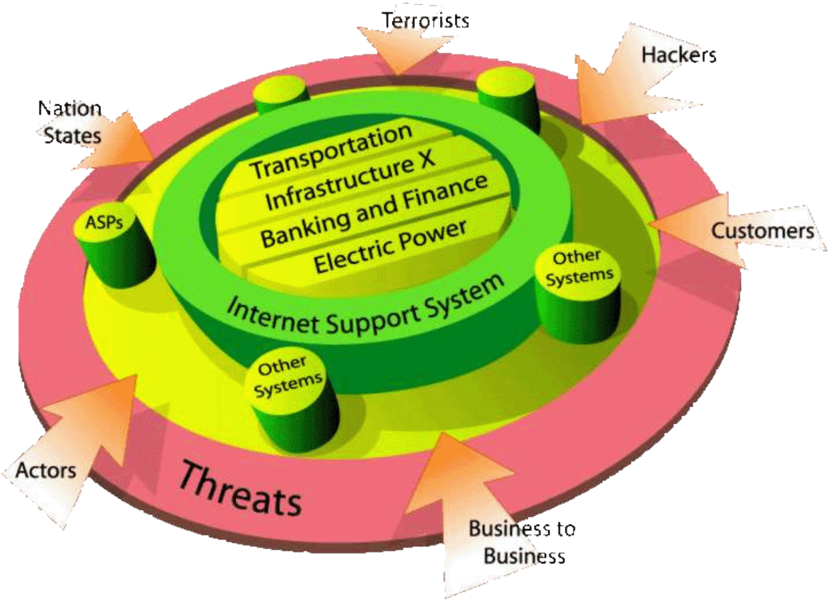
Why Software Security?

Developed nations' economies and defense depend, in large part, on the reliable execution of software

Software is ubiquitous, affecting all aspects of our personal and professional lives.

Software vulnerabilities are equally ubiquitous, jeopardizing:

- personal identities
- intellectual property
- consumer trust
- business services, operations, and continuity
- critical infrastructures & government



Software and security failures are rampant

JAMIE CONDLIFFE · HYBRIDS · 2/12/14 5:15am 6,939 31

Toyota Is Recalling Millions of Prius Hybrids to Fix a Software Bug

Toyota is recalling 1.9 million Prius hybrids to fix a serious bug in the engine control unit which can cause transistors to overheat—and potentially cause the hybrid system to shut down while driving.

Source: Gizmo, Feb 12, 2014

February 25, 2014 7:27 pm

iPhone software security flaws exposed

By Tim Bradshaw and Hannah Kuchler in San Francisco

Apple is facing its biggest security scare in years after flaws in its iPhone software risked exposing its users' communications.

Researchers at FireEye, a cyber security firm, on Monday published a "proof of concept" surveillance app that would allow an attacker to capture every tap on the iPhone's screen or buttons. This came after Apple quietly released a software update on Friday that fixed a serious weakness in its iOS software's encryption technology, which had existed for more than a year.

Source: Financial Times Limited, Feb 25, 2014

INTERNET | FEBRUARY 11, 2014, 9:33 AM | 1 Comments

Daily Report: Software Error Shakes Bitcoin Exchange

By THE NEW YORK TIMES

E-MAIL
FACEBOOK
TWITTER
SAVE
MORE

What was once the world's largest Bitcoin exchange, Mt. Gox, appeared near collapse on Monday, the latest symbol of the woes facing early players in the world of virtual currencies, Nathaniel Popper reports.

Daily Report

Mt. Gox, based in Tokyo, has had a rough ride lately. A few days after cutting off withdrawals for customers, Mt. Gox said on Monday that its problems were a result of a more fundamental flaw in the computer program that underlies Bitcoin.

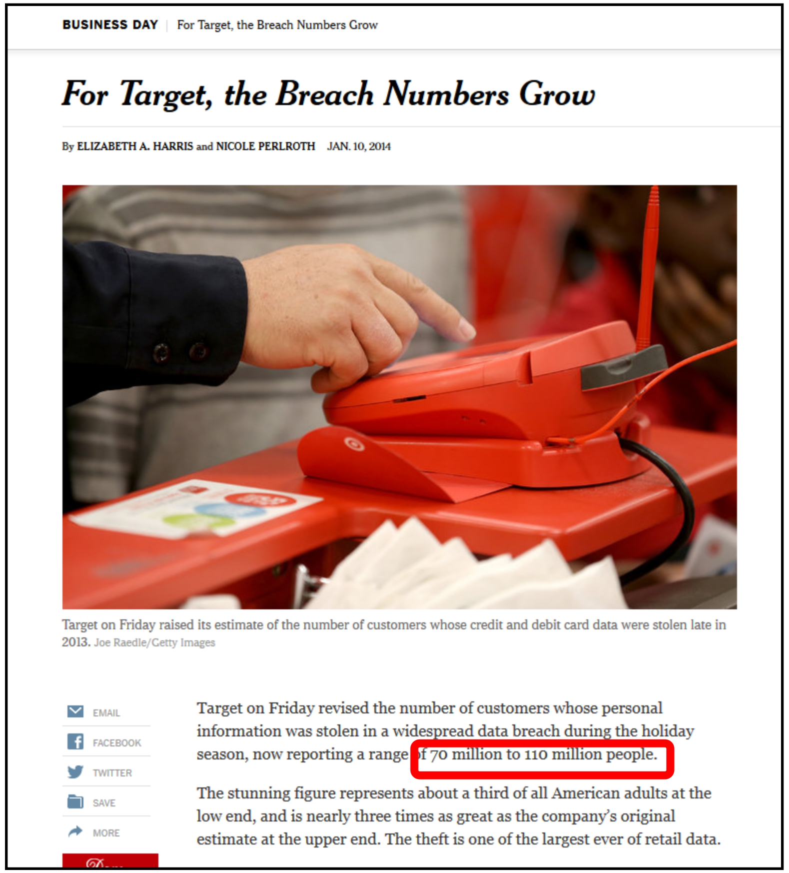
Source: New York Times, Feb 11, 2014

eBay Suffers Massive Security Breach, All Users Must Change Their Passwords

eBay publicly admit[ed] hackers had stolen the names, email and postal addresses, phone numbers and dates of birth of its 233 million users.

Sources: Forbes (online), May 21, 2014;
The Telegraph, May 22, 2104

Software and security failures are expensive



BUSINESS DAY | For Target, the Breach Numbers Grow

For Target, the Breach Numbers Grow

By ELIZABETH A. HARRIS and NICOLE PERLROTH JAN. 10, 2014

Target on Friday raised its estimate of the number of customers whose credit and debit card data were stolen late in 2013. Joe Raedle/Getty Images

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

EARNINGS

Target Earnings Slide 46% After Data Breach

Email Print 6 Comments

ARTICLE FREE PASS
Enjoy your free sample of exclusive subscriber content.
$12 *for* 12 Weeks SUBSCRIBE NOW

By PAUL ZIOBRO CONNECT

Updated Feb. 26, 2014 6:36 p.m. ET

Source: Wall Street Journal, Feb 26, 2014

Average cost in a breach:
$158 per record ($221 in US)

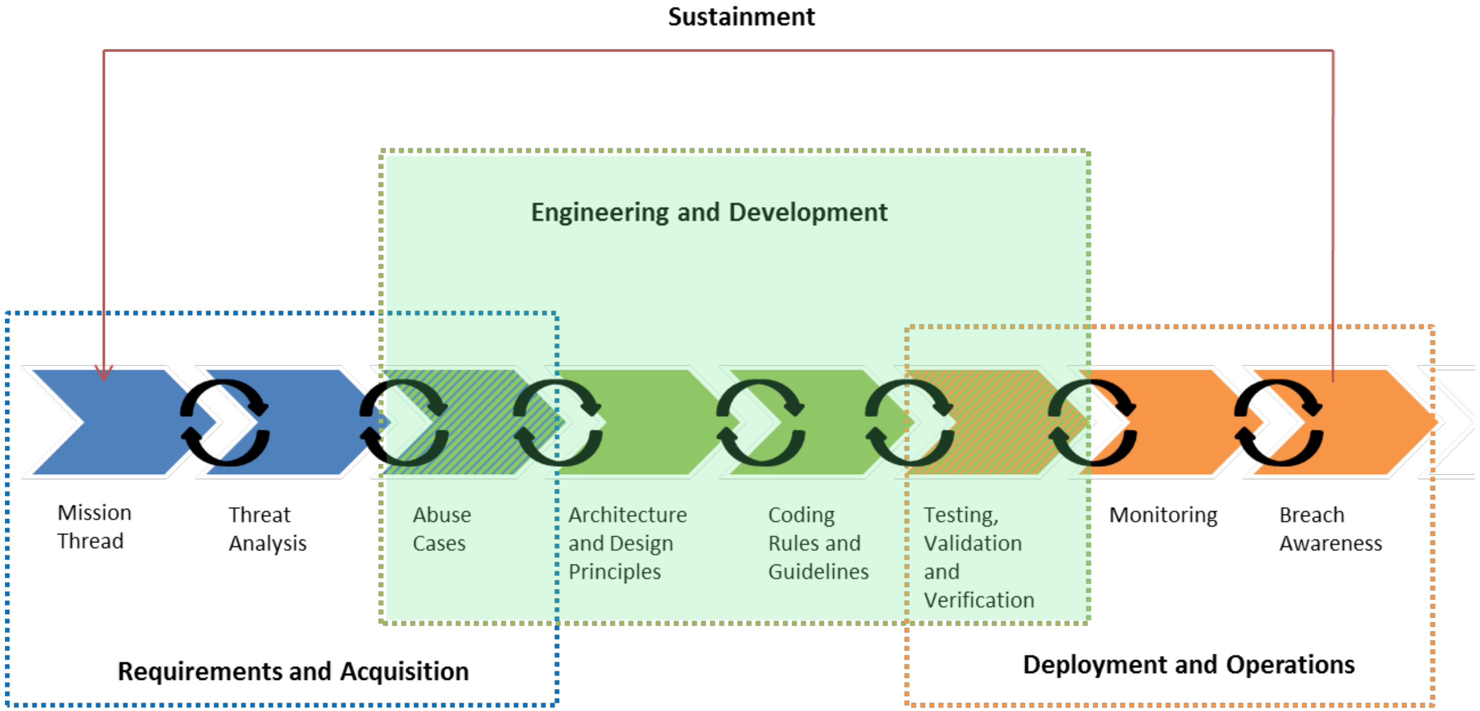
Source: Ponemon Institute, "2016 Cost of Data Breach Study: Global Analysis", June 2016

Polling Question 1

What programming language are you most concerned about using securely?

- Ada
- Assembly
- C
- C++
- C#
- Java
- Java-Script
- Objective-C
- Perl
- PHP
- Python
- PL/SQL or SQL
- Ruby
- Swift
- Visual Basic
- Other
- Little to none developed in-house

Engineering and Development



Most Vulnerabilities Are Caused by Programming Errors

64% of the vulnerabilities in the NIST National Vulnerability Database due to programming errors

- 51% of those were due to classic errors like buffer overflows, cross-site scripting, injection flaws

Top vulnerabilities include

- Integer overflow
- Buffer overflow
- Uncontrolled Format String
- Missing authentication
- Missing or incorrect authorization
- Reliance on untrusted inputs (aka tainted inputs)

Sources: Heffley/Meunier: Can Source Code Auditing Software Identify Common Vulnerabilities and Be Used to Evaluate Software Security?; cwe.mitre.org/top25 Jan 6, 2015

Secure Software Development

Secure software development starts with understanding insecure coding practices, and how these may be exploited.

Insecure designs can lead to “intentional errors”, that is, the code is correctly implemented but the resulting software contains a vulnerability.

Secure designs require an understanding of functional and non-functional software requirements.

Secure coding requires an understanding of implementation specifics.

Sources of Software Insecurity

Absent or minimal consideration of security during all life cycle phases

Complexity, inadequacy, and change

Incorrect or changing assumptions

Not thinking like an attacker

Flawed specifications & designs

Poor implementation of software interfaces

Unintended, unexpected interactions

- with other components
- with the software's execution environment

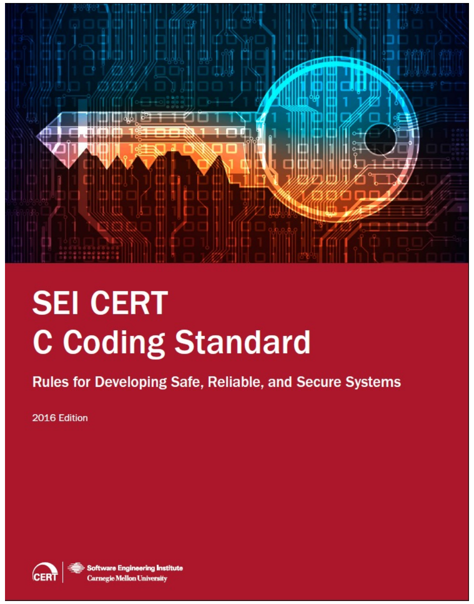
Inadequate knowledge of secure coding practices

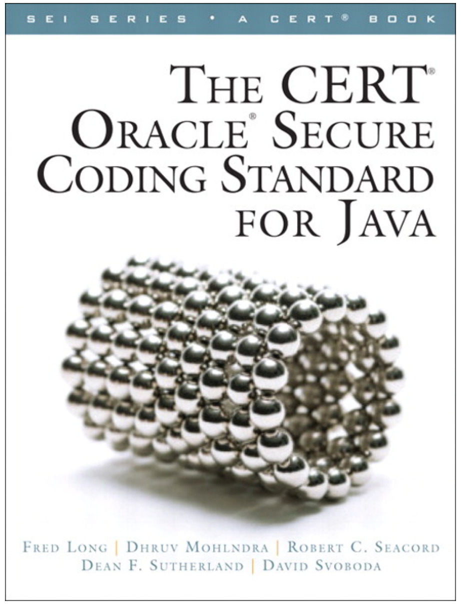
Polling Question 2

Does your organization use a coding standard for security?

- Yes
- No
- Maybe?

Coding rules – 2016 Edition





- Collected wisdom of programmers and tools vendors
  - Fed by community wiki started in Spring 2006
  - Over 1,500 registered contributors
- C Coding Standards
  Available as downloadable report

http://cert.org/secure-coding/products-services/secure-coding-download.cfm

- Java Coding Standards
  Available as book
- C++, Perl, and "Current Standards"
  Available on Secure Coding Wiki

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

CWE Guidance

Common Weakness Enumeration

A Community-Developed Dictionary of Software Weakness Types

CWSS
CWRAF

Home > CWE List > CWE- Individual Dictionary Definition (2.9)   Search by ID: Go

- **CWE List**
  - Full Dictionary View
  - Development View
  - Research View
  - Fault Pattern View
  - Reports
  - Mapping & Navigation
- **About**
  - Sources
  - Process
  - Documents
  - FAQs
- **Community**
  - Use & Citations
  - SwA On-Ramp
  - Discussion List
  - Discussion Archives
  - Contact Us
- **Scoring**
  - Prioritization
  - CWSS
  - CWRAF
  - CWE/SANS Top 25
- **Compatibility**
  - Requirements
  - Coverage Claims Representation
  - Compatible Products
  - Make a Declaration
- **News**
  - Calendar
  - Free Newsletter
- **Search the Site**

Presentation Filter: --None--

CWE-120: Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')

Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')

Weakness ID: 120 *(Weakness Base)*   **Status:** Incomplete

Description

Description Summary

The program copies an input buffer to an output buffer without verifying that the size of the input buffer is less than the size of the output buffer, leading to a buffer overflow.

......

Detection Methods

Automated Static Analysis

This weakness can often be detected using automated static analysis tools. Many modern tools use data flow analysis or constraint-based techniques to minimize the number of false positives.

Automated static analysis generally does not account for environmental considerations when reporting out-of-bounds memory operations. This can make it difficult for users to determine which warnings should be investigated first. For example, an analysis tool might report buffer overflows that originate from command line arguments in a program that is not expected to run with setuid or other special privileges.

Effectiveness: High

Detection techniques for buffer-related errors are more mature than for most other weakness types.

Automated Dynamic Analysis

This weakness can be detected using dynamic tools and techniques that interact with the software using large test suites with many diverse inputs, such as fuzz testing (fuzzing), robustness testing, and fault injection. The software's operation may slow down, but it should not become unstable, crash, or generate incorrect results.

Manual Analysis

Manual analysis can be useful for finding this weakness, but it might not achieve desired code coverage within limited time constraints. This becomes difficult for weaknesses that must be considered for all inputs, since the attack surface can be too large.

Automated Static Analysis - Binary / Bytecode

According to SOAR, the following detection techniques may be useful:

OWASP Guidance

Home
About OWASP
Acknowledgements
Advertising
AppSec Events
Books
Brand Resources
Chapters
Donate to OWASP
Downloads
Funding
Governance
Initiatives
Mailing Lists
Membership
Merchandise
News
Community portal
Presentations
Press
Projects
Video
Volunteer

Page Discussion

Buffer Overflows

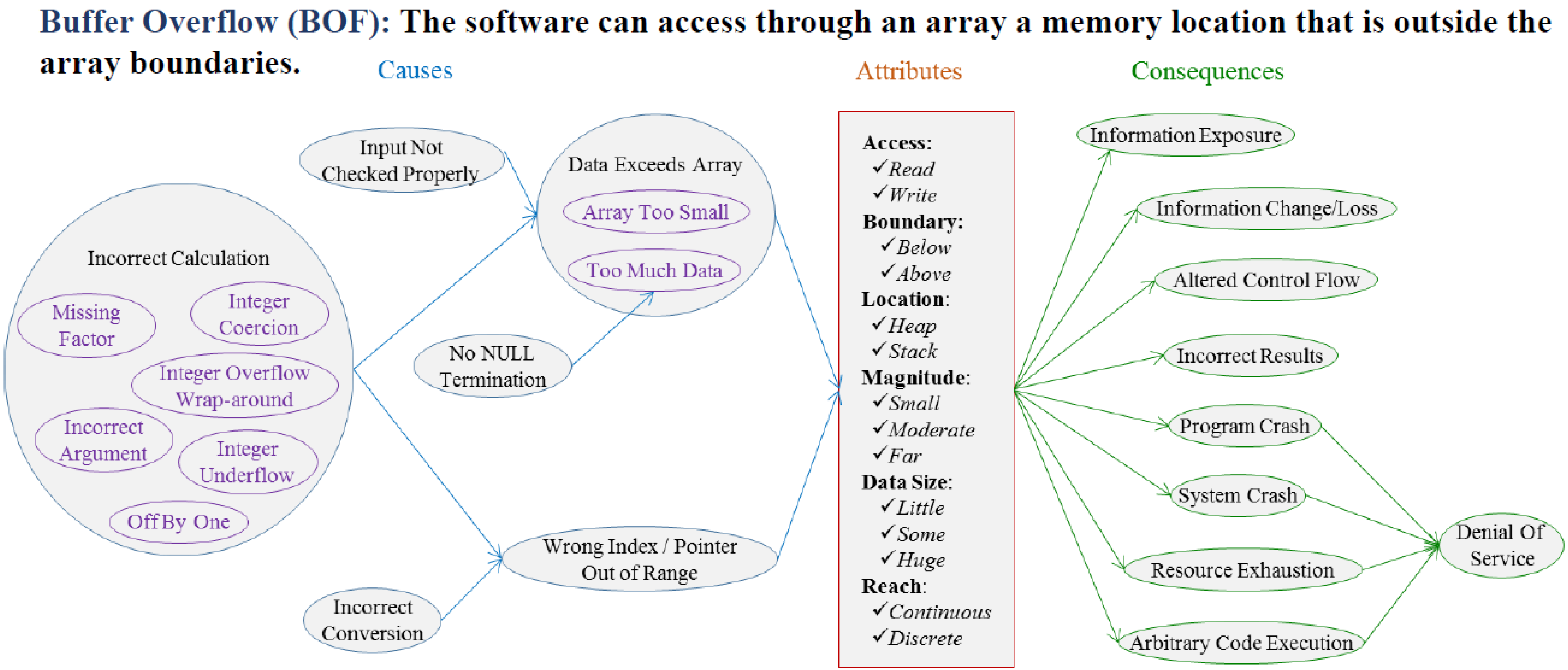
......

General Prevention Techniques

A number of general techniques to prevent buffer overflows include:

- Code auditing (automated or manual)
- Developer training – bounds checking, use of unsafe functions, and group standards
- Non-executable stacks – many operating systems have at least some support for this
- Compiler tools – StackShield, StackGuard, and Libsafe, among others
- Safe functions – use strncat instead of strcat, strncpy instead of strcpy, etc
- Patches – Be sure to keep your web and application servers fully patched, and be aware of bug reports relating to applications upon which your code is dependent.
- Periodically scan your application with one or more of the commonly available scanners that look for buffer overflow flaws in your server products and your custom web applications.

Buffer overflow has many causes



Source: Bojanova, et al, "The Bugs Framework (BF): A Structured, Integrated Framework to Express Software Bugs",2016, http://www.mys5.org/Proceedings/2016/Posters/2016-S5-Posters_Wu.pdf

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

An methodology for rule creation

Exploit language ambiguities

Analyze vulnerable programs

Systematically test the rules

And still consult with experts

Examine language definitions and standards for undefined, unspecified and implementation-defined behavior

3.4.3

1 **undefined behavior**

behavior, upon use of a nonportable or erroneous program construct or of erroneous data, for which this International Standard imposes no requirements

2 NOTE Possible undefined behavior ranges from ignoring the situation completely with unpredictable results, to behaving during translation or program execution in a documented manner characteristic of the environment (with or without the issuance of a diagnostic message), to terminating a translation or execution (with the issuance of a diagnostic message).

3 EXAMPLE An example of undefined behavior is the behavior on integer overflow.

3.4.4

1 **unspecified behavior**

use of an unspecified value, or other behavior where this International Standard provides two or more possibilities and imposes no further requirements on which is chosen in any instance

2 EXAMPLE An example of unspecified behavior is the order in which the arguments to a function are evaluated.

Source: http://www.open-std.org/jtc1/sc22/wg14/www/docs/n1124.pd (ISO 9899 - Programming Languages – C draft)

Examine vulnerable code for patterns

Malware repository with millions of unique, tagged artifacts

CERT Secure Coding Team has evaluated over 100M LOC

CERT | Software Engineering Institute | Carnegie Mellon University

Vulnerability Notes Database

Advisory and mitigation information about software vulnerabilities

CERT Knowledgebase

The CERT Knowledgebase is a collection of internet security information related to incidents and vulnerabilities. The CERT Knowledgebase houses the public Vulnerability Notes Database as well as two restricted-access components:

- Vulnerability Card Catalog contains descriptive and referential information regarding thousands of vulnerabilities reported to the CERT Coordination Center.
- Special Communications Database contains briefs that provide advance warning and important information about vulnerabilities, intruder activity, or other critical security threats.

Implement candidate rules and run against sample code

- Focus rule when possible to
  - maximize true positive of weakness (tag bad code)
  - minimize false negative of weakness (don't tag good code)

- Write program to evaluate source code for particular rule

- Run program against collection of known bad source code and a collection of other (suspected good) code to check sensitivity and specificity of results

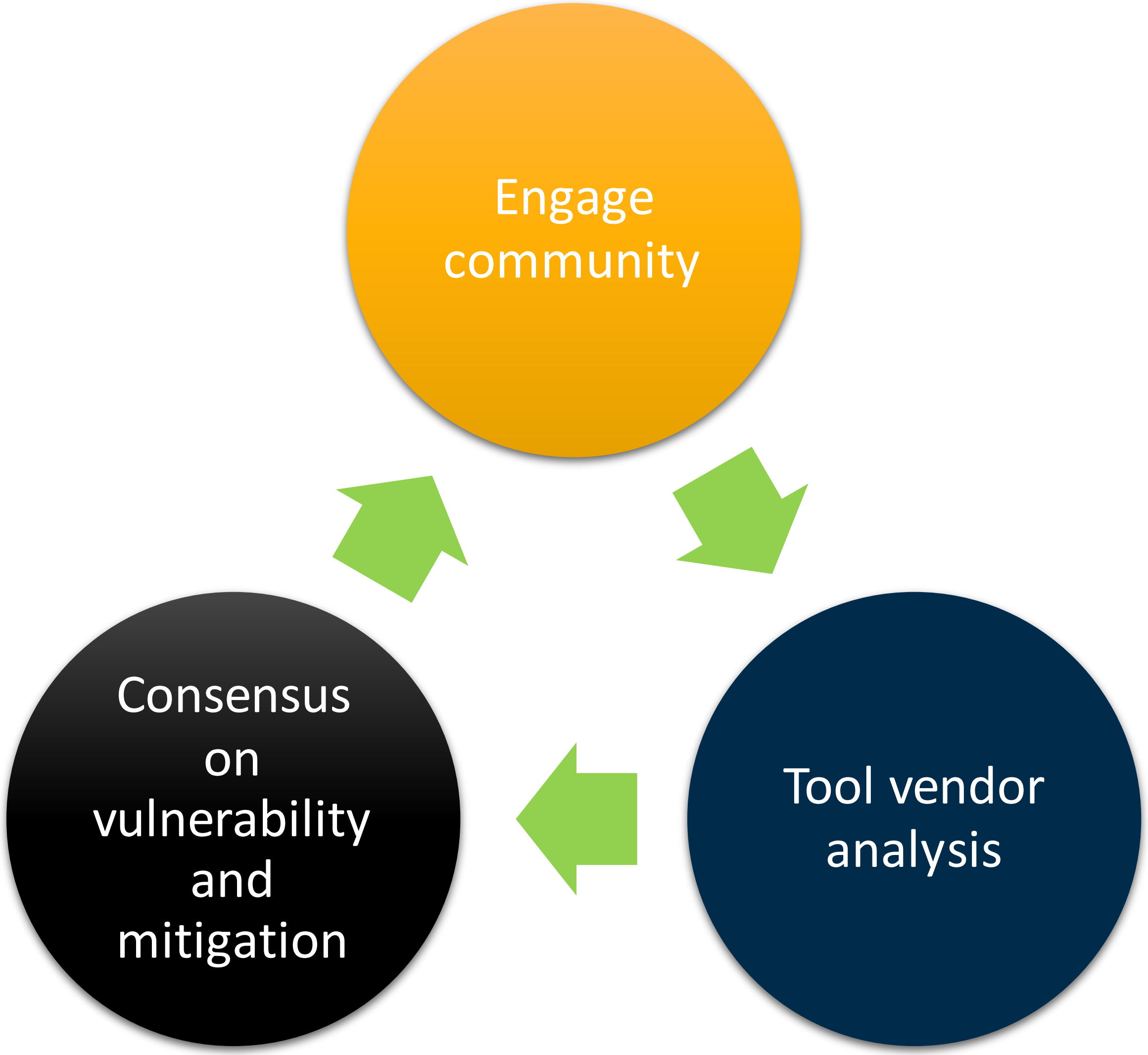
Experience with systematic testing

- Candidate rule typical evaluation
  - 10 iterations of proposed rule and associated checker
    - 7 internal evaluations
    - 3 external evaluations

- Each evaluation iteration carried out against > 10M lines of representative code
  - Variety of domains
  - Variety of code quality

- As part of creating C++ standard, general methodology applied to generate 46 rules and corresponding Clang C++ checkers
  - 19 by CERT researchers, 27 by others

Tapping into expert knowledge for developing CERT coding standards



New Rule Example

EXP46-C – Do not use a bitwise operator with a Boolean-like operand

```
if (!(getuid() & geteuid() == 0)) {
  /* ... */
}
```

```
if (!(getuid() && geteuid() == 0)) {
  /* ... */
}
```

CWE-480, Use of incorrect operator

Updated Rule Example

ARR38-C – Guarantee that library functions do not form invalid pointers

```
if (1 + 2 + payload + 16 > s->s3->rrec.length)
   return 0; /* Silently discard per RFC 6520 */
```

Heartbleed.com

CWE-119, Improper Restriction of Operations within the Bounds of a Memory Buffer

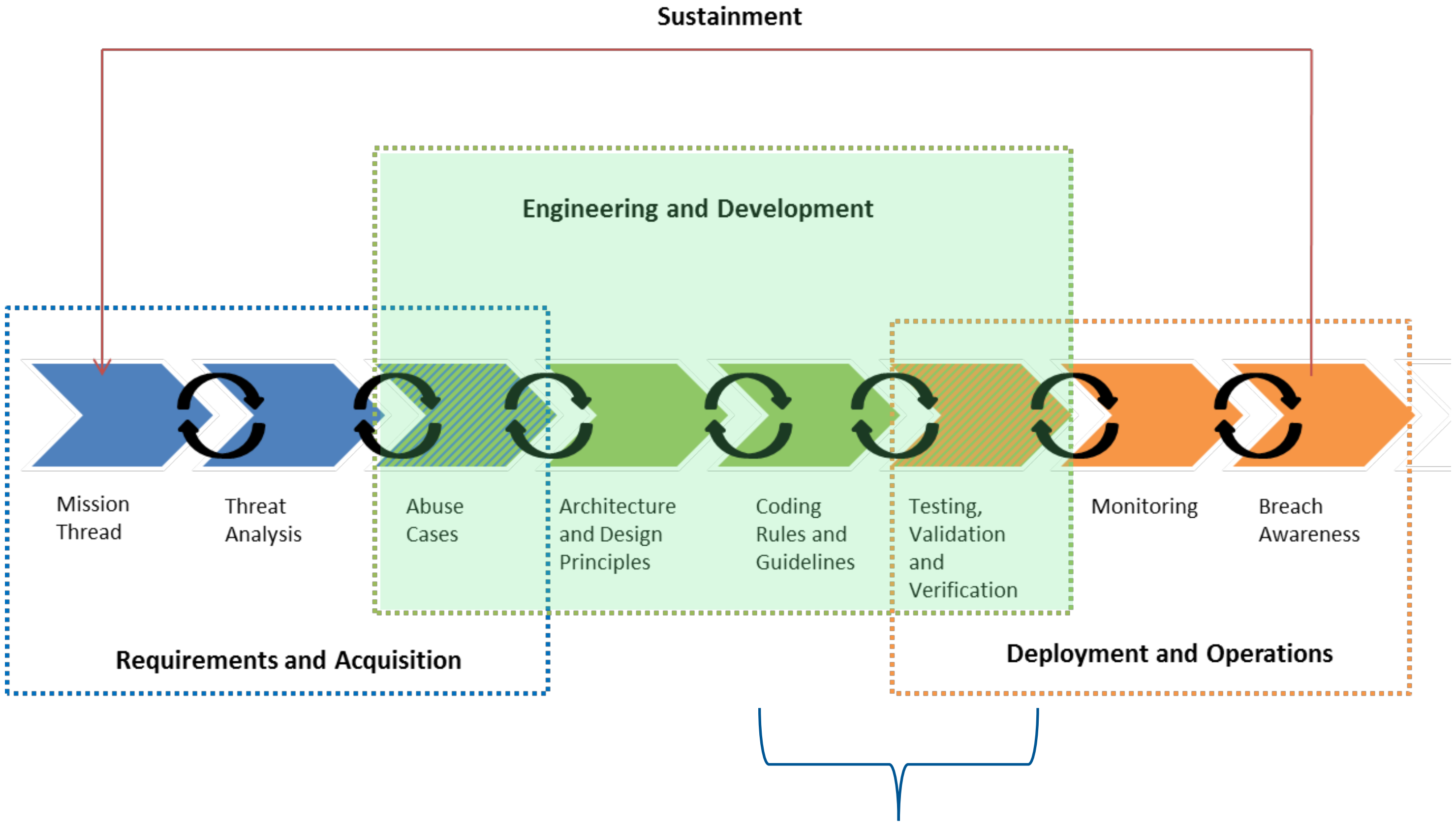
CWE-121, Stack-based Buffer Overflow

CWE-123, Write-what-where Condition

CWE-125, Out-of-bounds Read

CWE-805, Buffer Access with Incorrect Length Value

Development and Verification



DISA STIG Requirements

Application Security STIG Requirements:

- APP3550: CAT I – not vulnerable to integer arithmetic issues
- APP3560: CAT I – does not contain format string vulnerabilities
- APP3570: CAT I – does not allow command injection
- APP3590.1: CAT I – does not have buffer overflows
- APP3590.2: CAT I – does not use functions known to be vulnerable to buffer overflows
- APP2060.1: CAT II – development team follows a set of coding standards
- APP2060.2: CAT II – development team creates a list of unsafe functions to avoid and include in coding standards
- APP2120.3: CAT II – developers are provided with training on secure design and coding practices on at least an annual basis

From Defense Information Systems Agency Application Security and Development Security Technical Implementation Guide, V3 R10 (2015)

Adopting Secure Coding Practices

Secure Coding Infrastructure

- Defining Secure Coding Practices
- Influencing Language Standards
- Influencing Tool Vendors

Processes

- Coding Standards and Security Standards, Testing

Technology

- Tools: IDE's and Analyzers
- Automated transformation and remediation

People

- Workforce Development

Risk Assessment

Risk assessment is performed using failure mode, effects, and criticality analysis.

Severity—How serious are the consequences of the rule being ignored?

Value	Meaning	Examples of Vulnerability
1	low	denial-of-service attack, abnormal termination
2	medium	data integrity violation, unintentional information disclosure
3	high	run arbitrary code

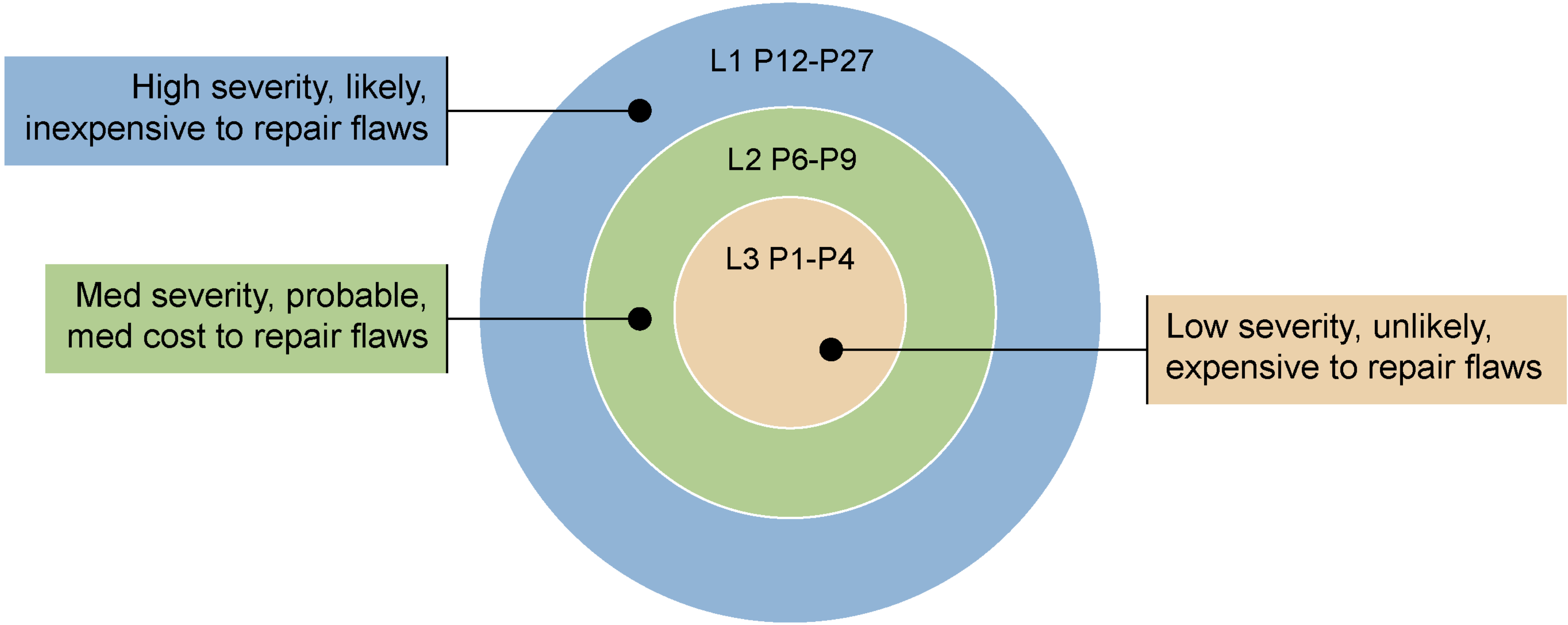
Likelihood—How likely is it that a flaw introduced by ignoring the rule can lead to an exploitable vulnerability?

Value	Meaning
1	unlikely
2	probable
3	likely

Cost—The cost of mitigating the vulnerability.

Value	Meaning	Detection	Correction
1	high	manual	manual
2	medium	automatic	manual
3	low	automatic	automatic

Priorities and Levels



Conformance Testing

The use of secure coding standards defines a proscriptive set of rules and recommendations to which the source code can be evaluated for compliance.

For each secure coding standard, the source code is certified as provably nonconforming, conforming, or provably conforming against each guideline in the standard:

Provably nonconforming	The code is provably nonconforming if one or more violations of a rule are discovered for which no deviation has been allowed.
Conforming	The code is conforming if no violations of a rule can be identified.
Provably conforming	Finally, the code is provably conforming if the code has been verified to adhere to the rule in all possible cases.

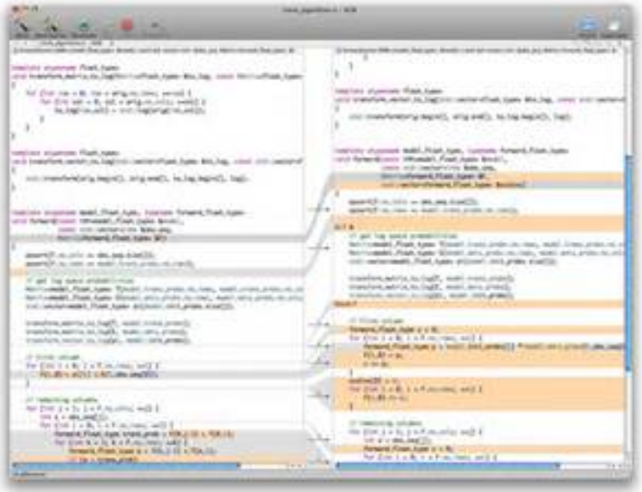
Evaluation violations of a particular rule ends when a “provably nonconforming” violation is discovered.

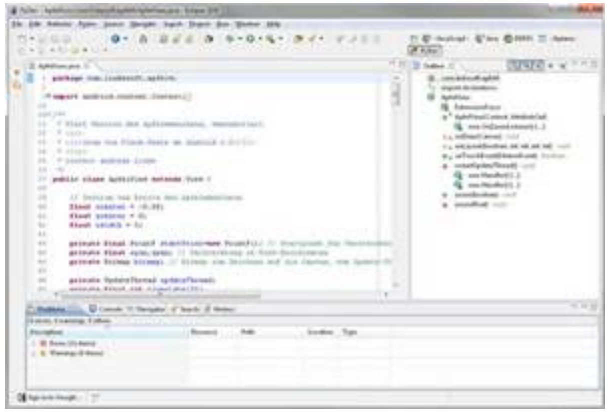
Polling Question 3

What testing does your organization perform on your software?

- Static Analysis
- Dynamic Analysis
- Both
- None

Tools encourage application of secure coding





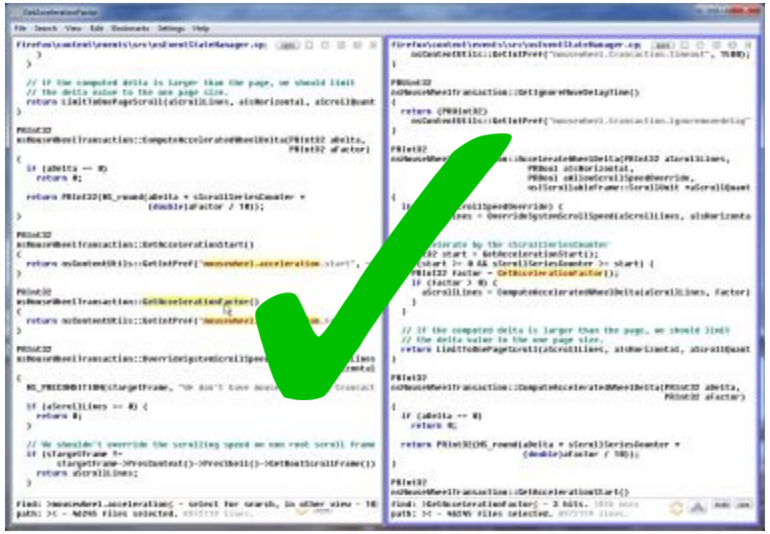
Moving rules into IDEs improves application of secure coding:

- Early feedback corrects errors on introduction.
- Exceptions are understood in context.

Adoption of secure coding IDEs

- help deploy tools
- training on tools
- extend tools to meet targeted needs

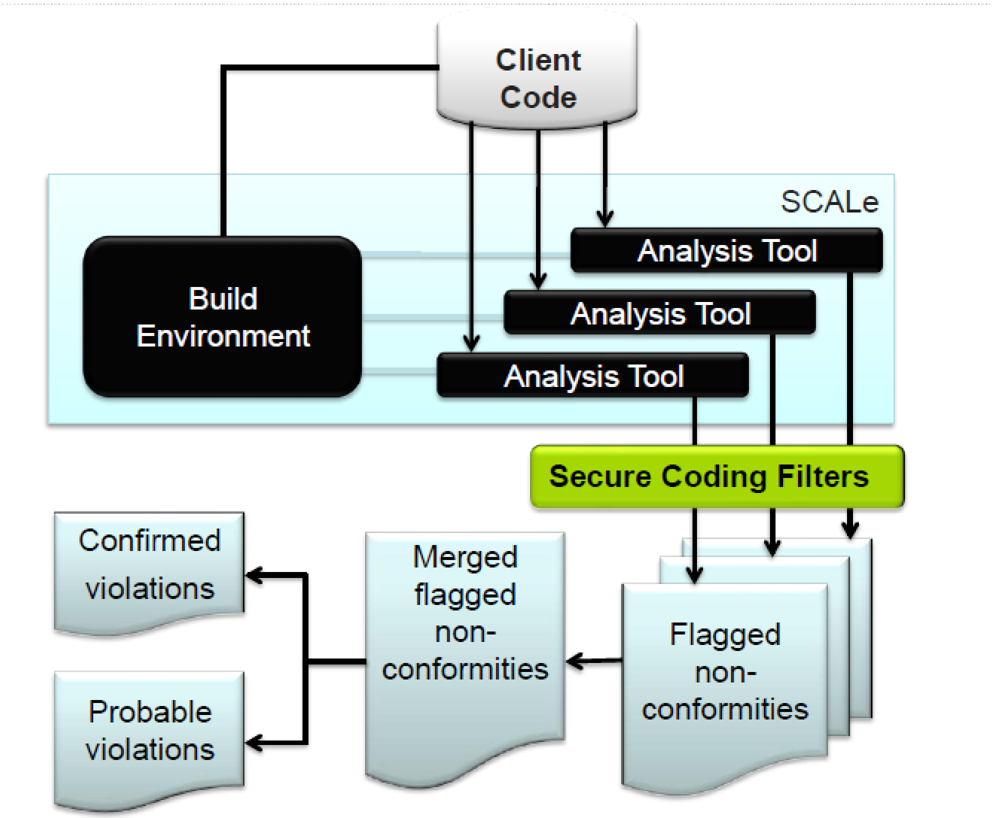
Static Testing – Source code analysis tools



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

Polling Question 4

Do you use multiple static analysis tools?

- Yes, and we use a tool diagnostic aggregator
- Yes, but we review the tool diagnostics separately
- No, we just use one static analysis tool
- No, we don’t use static analysis tools

Select SCALe Assessments

Codebase	Date	Customer	Lang	ksLOC	Rules	Diags	True	Suspect	Diag /KsLOC
A	6/12	Gov1	C++	38.8	12	1,071	52	1,019	27.6
B	3/13	Gov1	C	87.4	28	17,543	86	17,457	200.7
C	10/13	Gov2	C	9,585	18	289	159	130	0.03
D	6/12	Gov3	Java	4.27	18	345	117	228	80.8
E	9/12	Gov2	Java	61.2	33	538	288	250	8.8
F	11/13	Gov2	Java	17.6	21	414	341	73	23.5
G	2/14	Gov4	Java	653	29	8,526	64	8,462	13.1
H	3/14	Gov5	Java	1.51	8	53	53	0	35.1
I	5/14	Mil1	Java	403	27	3114	723	2,391	7.7
J	1/11	Gov3	Perl	93.6	36	6,925	357	6,568	74.0
K	5/14	Gov3	Perl	10.2	10	133	84	49	13.0

Polling Question 5

Have you taken some training on secure coding practices?

- Yes, self-taught
- Yes, through an online-delivered program
- Yes, through an in-person delivered program
- Yes, through my academic education
- No

Secure Coding Professional Certificates

CERT Secure Coding Professional Certificates

Online Courses with Exam and Certificates for C/C++ and Java
2 Courses (Secure Software Concepts & Secure Coding) and Exam
Onsite, instructor-led courses available for groups

SEI Secure Coding in C/C++ Training 1

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
- Integer security
- Formatted output
- File I/O

http://www.sei.cmu.edu/training/p63.cfm

SEI Secure Coding in C/C++ Training 2

Participants gain a working knowledge of common programming errors that lead to software vulnerabilities, how these errors can be exploited, and mitigation strategies to prevent their introduction.

Objectives

- Improve the overall security of any C or C++ application.
- Thwart buffer overflows and stack-smashing attacks that exploit insecure string manipulation logic.
- Avoid vulnerabilities and security flaws resulting from incorrect use of dynamic memory management functions.
- Eliminate integer-related problems: integer overflows, sign errors, and truncation errors.
- Correctly use formatted output functions without introducing format-string vulnerabilities.
- Avoid I/O vulnerabilities, including race conditions.

Java Secure Coding Course

The Java Secure Coding Course is designed to improve the secure use of Java. Designed primarily for Java SE 8 developers, the course is useful to developers using older versions of the platform as well as Java EE and ME developers. Tailored to meet the needs of a development team, the course can cover security aspects of

- Trust and Security Policies
- Validation and Sanitization
- The Java Security Model
- Declarations
- Expressions
- Object Orientation
- Methods
- Vulnerability Analysis Exercise
- Numerical Types in Java
- Exceptional Behavior
- Input/Output
- Serialization
- The Runtime Environment
- Introduction to Concurrency in Java
- Advanced Concurrency Issues

http://www.sei.cmu.edu/training/p118.cfm

Polling Question 6

Are you more concerned about the secure code that you develop or acquire/procure?

- Software we develop
- Source code we acquire/procure
- Third-party libraries we acquire/procure
- Complete software we acquire/procure and integrate
- All of the above

Evolution of software development

Custom development – context:

- Software was limited
  - Size
  - Function
  - Audience
- Each organization employed developers
- Each organization created their own software

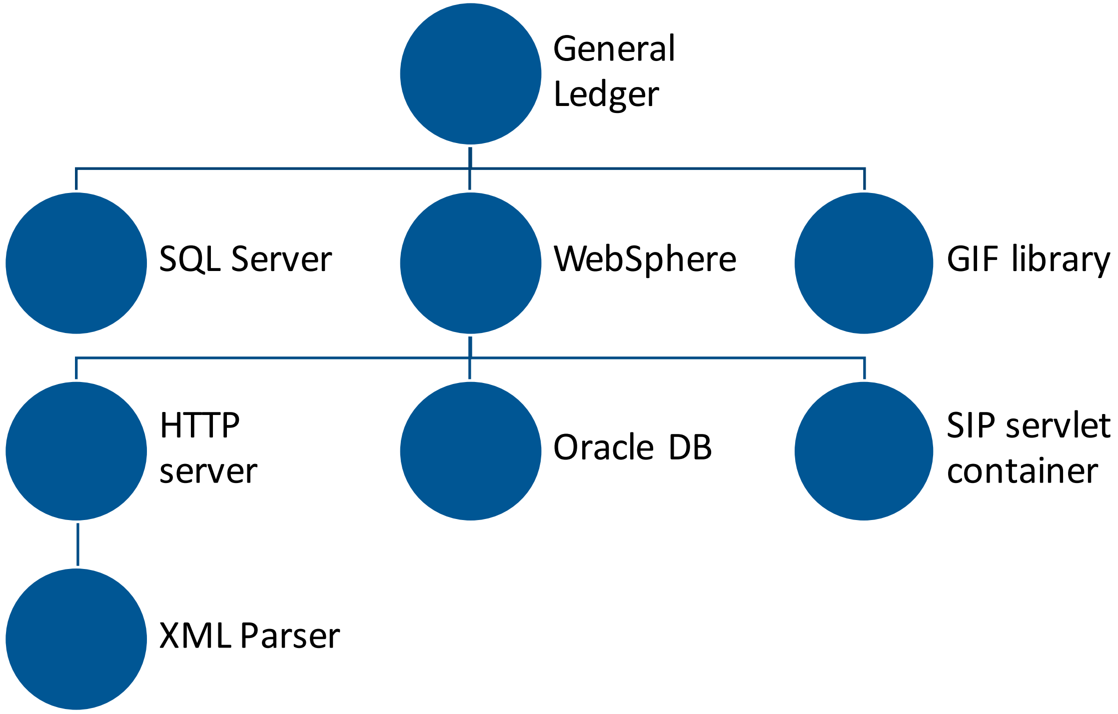
Supply chain: practically none

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: software supplier

Development is now assembly



Note: hypothetical application composition

Collective development – context:

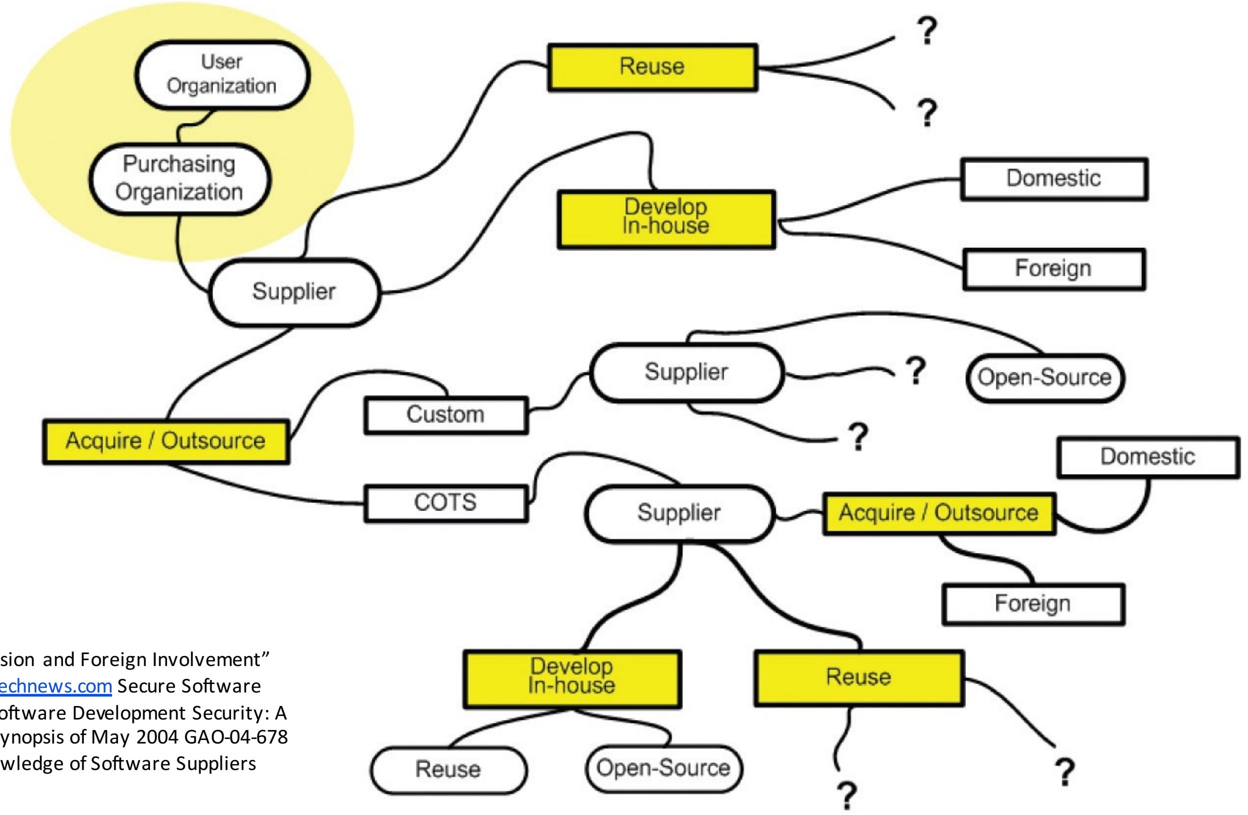
- Too large for single organization
- Too much specialization
- Too little value in individual components

Supply chain: long

Software supply chain for assembled software

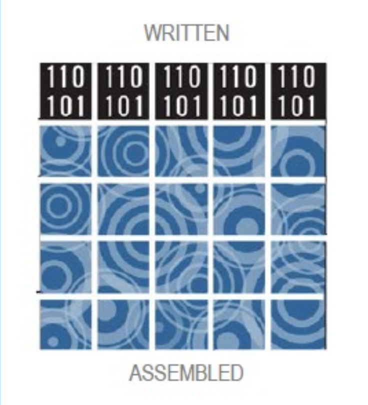
Expanding the scope and complexity of acquisition and deployment

Visibility and direct controls are limited (only in shaded area)



Source: "Scope of Supplier Expansion and Foreign Involvement" graphic in DACS www.softwaretechnews.com Secure Software Engineering, July 2005 article "Software Development Security: A Risk Management Perspective" synopsis of May 2004 GAO-04-678 report "Defense Acquisition: Knowledge of Software Suppliers Needed to Manage Risks"

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

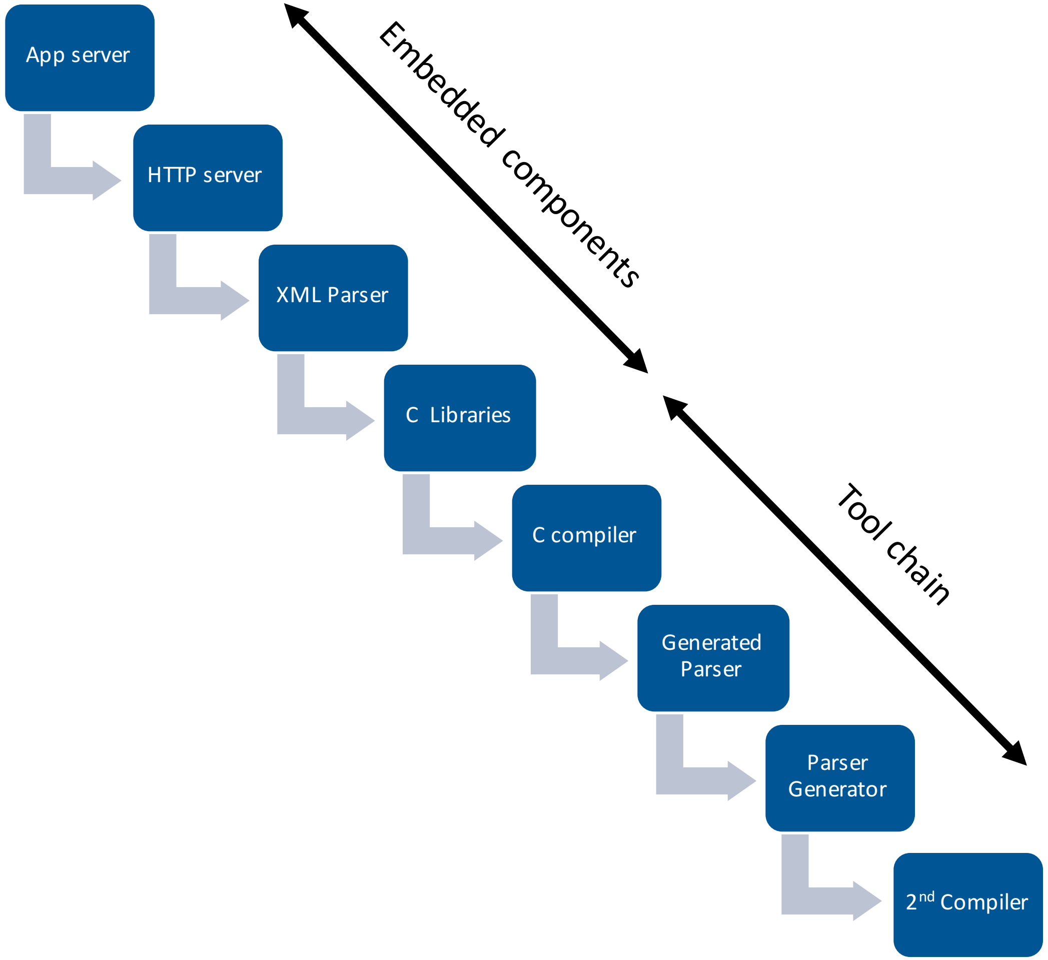
Distributed development – context:

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

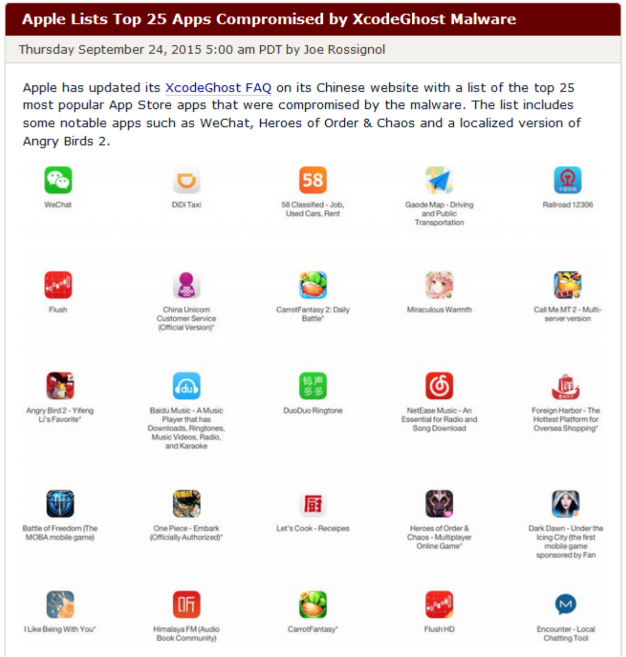
Supply chain: opaque

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 in the tool chain already exists



Apple Lists Top 25 Apps Compromised by XcodeGhost Malware

Thursday September 24, 2015 5:00 am PDT by Joe Rossignol

Apple has updated its XcodeGhost FAQ on its Chinese website with a list of the top 25 most popular App Store apps that were compromised by the malware. The list includes some notable apps such as WeChat, Heroes of Order & Chaos and a localized version of Angry Birds 2.

WeChat
DiDi Taxi
58 Classified - Job, Used Cars, Rent
Gaode Map - Driving and Public Transportation
Railroad 12306
Flush
China Unicom Customer Service (Official Version)*
CarrotFantasy 2: Daily Battle*
Miraculous Warmth
Call Me MT 2 - Multi-server version
Angry Bird 2 - Yifeng Li's Favorite*
Baidu Music - A Music Player that has Downloads, Ringtones, Music Videos, Radio, and Karaoke
DuoDuo Ringtone
NetEase Music - An Essential for Radio and Song Download
Foreign Harbor - The Hottest Platform for Oversea Shopping*
Battle of Freedom (The MOBA mobile game)
One Piece - Embark (Officially Authorized)*
Let's Cook - Receipes
Heroes of Order & Chaos - Multiplayer Online Game*
Dark Dawn - Under the Icing City (the first mobile game sponsored by Fan
I Like Being With You*
Himalaya FM (Audio Book Community)
CarrotFantasy*
Flush HD
Encounter - Local Chatting Tool

Sources: http://www.macrumors.com/2015/09/24/xcodeghost-top-25-apps-apple-list/
http://www.itntoday.com/2015/09/the-85-ios-apps-affected-by-xcodeghost.html

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

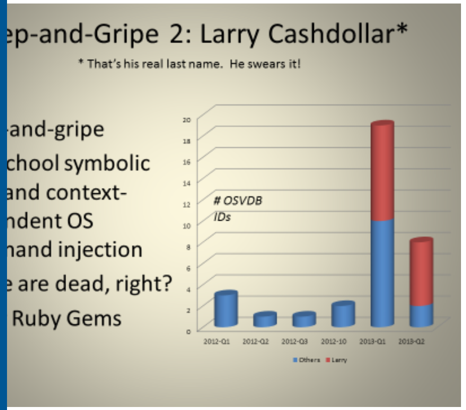
Open source is not secure

Heartbleed and
Shellshock were fou
by exploitation

Other open source
software illustrates
vulnerabilities from cu
inspection

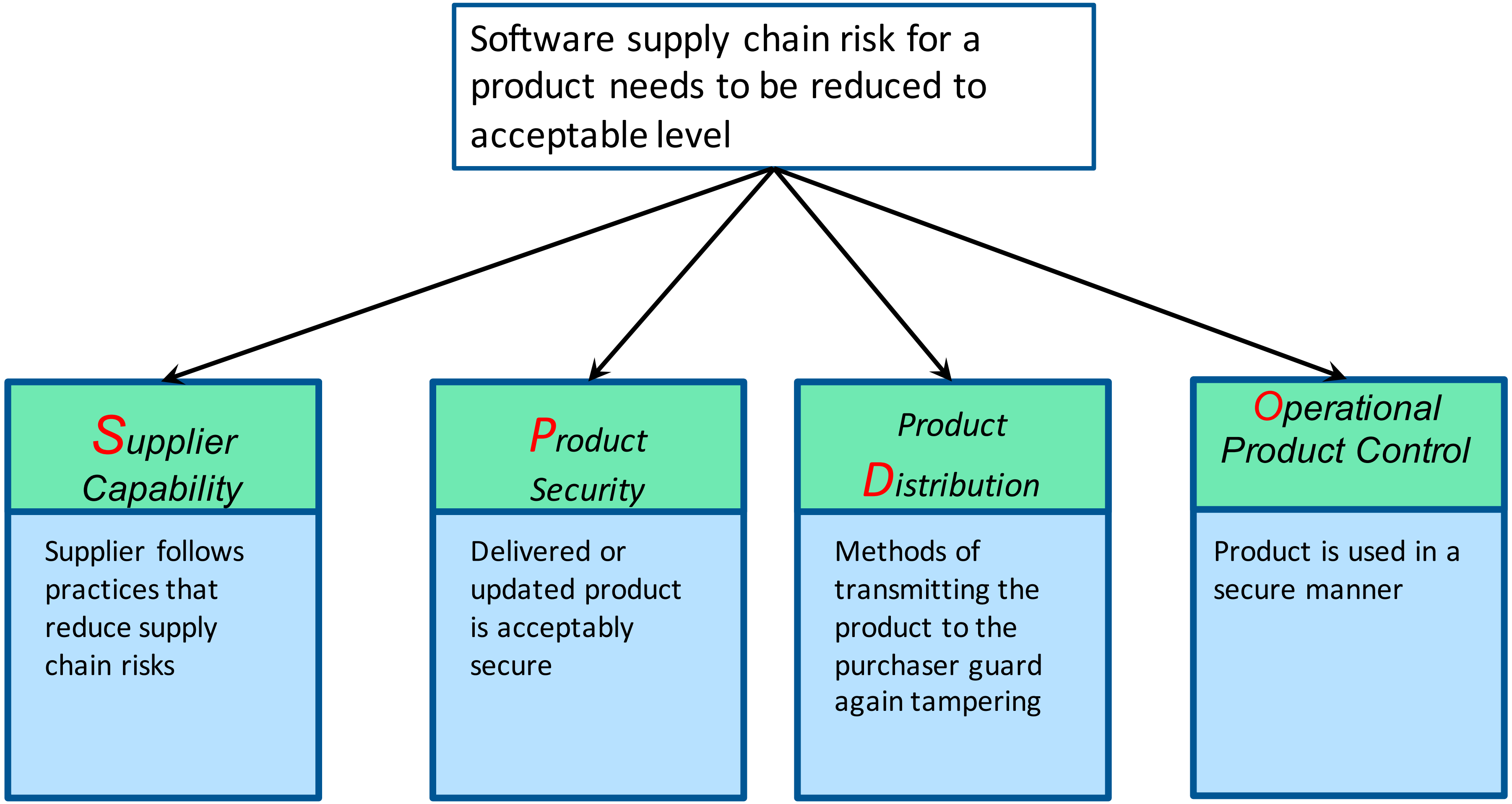
1.8 billion vulnerable open source components downloaded in 2015

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



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

Reducing software supply chain risk factors



Supplier security commitment evidence

Supplier employees are educated as to security engineering practices

- Documentation for each engineer of training and when trained/retrained
- Revision dates for training materials
- Lists of acceptable credentials for instructors
- Names of instructors and their credentials

Supplier follows suitable security design practices

- Documented design guidelines
- Has analyzed attack patterns appropriate to the design such as those that are included in Common Attack Pattern Enumeration and Classification (CAPEC)
- Application of code signing techniques (interest in ISO 17960 – in early draft)

Evaluate a product's threat resistance

What product characteristics minimize opportunities to enter and change the product's security characteristics?

- Attack surface evaluation: Exploitable features have been identified and eliminated where possible
  - Access controls
  - Input/output channels
  - Attack enabling applications – email, Web
- Design and coding weaknesses associated with exploitable features have been identified and mitigated (CWE)
- Independent validation and verification of threat resistance
- Dynamic, Static, Interactive Application Security Testing (DAST, SAST, IAST)
- Delivery in or compatibility with Runtime Application Self Protection (RASP) containers

Establishing good product distribution practices

Recognize that supply chain risks are accumulated

- Subcontractor/COTS-product supply chain risk is inherited by those that use that software, tool, system, etc.

Apply to the acquiring organizations and their suppliers

- Require good security practices by their suppliers
- Assess the security of delivered products
- Address the additional risks associated with using the product in their context

Ideally open source is built with a compiler you trust

Maintain operational attack resistance

Who assumes responsibility for preserving product attack resistance with product deployment?

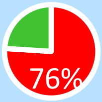
- Maintaining inventory of components
- Patching and version upgrades (component lifecycle management)
- Expanded distribution of usage
- Expanded integration

Usage changes the attack surface and potential attacks for the product

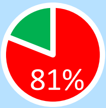
- Change in feature usage or risks
- Are supplier risk mitigations adequate for desired usage?
- Effects of vendor upgrades/patches and local configuration changes
- Effects of integration into operations (system of systems)

Where to start

Anywhere



No meaningful controls over what components are applications



No coordination of security practices in various stages of the development life cycle

No acceptance tests for third-party code

Plenty of models to choose from

BSIMM: Building Security in Maturity Model

CMMI: Capability Maturity Model Integration for Acquisitions

PRM: SwA Forum Processes and Practices Group Process Reference Model

RMM: CERT Resilience Management Model

SAMM: OWASP Open Software Assurance Maturity Model

Sources: Sonatype, 2014 Sonatype Open Source Development and Application Security Survey; Forrester Consulting, “State of Application Security,” January 2011

Contact Information

Robert Schiela
Technical Manager, Secure Coding
Telephone: +1 412.268.3736
Email: rschiela@cert.org

Mark Sherman
Technical Director, Cyber Security Foundations
Telephone: +1 412.268.9223
Email: mssherman@cert.org

Web Resources

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

http://www.cert.org/

http://www.cert.org/secure-coding/

http://securecoding.cert.org/