Secure Coding Frameworks
For the Development of Safe, Secure and Reliable Code

Tim Kertis
October 2015

The 27th Annual IEEE Software Technology Conference
Who Am I?

- Tim Kertis, Software Engineer/Software Architect
- Chief Software Architect, Raytheon IIS, Indianapolis
- Master of Science, Computer & Information Science, Purdue
- Software Architecture Professional through the Software Engineering Institute (SEI), Carnegie-Mellon University (CMU)
- 30 years of diverse Software Engineering Experience
- Advocate for Secure Coding Frameworks (SCFs)
- Author of the JAVA Secure Coding Framework (JSCF)
- Inventor of Cybersecurity thru Lexical And Symbolic Proxy (CLaSP) technology (patent pending)
Top 5 Cybersecurity Concerns …

- 1 - Application Vulnerabilities
- 2 - Malware
- 3 - Configuration Mistakes
- 4 - Mobile Devices
- 5 - Hackers

According to the 2015 ISC(2) Global Information Security Workforce Study (Paresh Rathod)
Number of Software Developers:
- 18,000,000+ (www.infoq.com)

Number of Java Software Developers:
- 9,000,000+ (www.infoq.com)

Software with Vulnerabilities:
- 96% (www.cenzic.com)

Total Cost of Cyber Crime:
- $500B (McAfee)

Cost of Cyber Incidents:
- Low $1.6M
- Average $12.7M
- High $61.0M
  (Ponemon Institute)
Research Conducted

- **SEI Secure Coding Standard**
  - Rules and Recommendations
  - Priorities and Levels
  - Vulnerabilities and Remedies

- **Common Weakness Enumeration (CWE)**
  - Common Software Weaknesses

- **Open Web Application Security Project (OWASP)**
  - Language-Agnostic/ Framework-Agnostic Developer Guide

- **Top 10 Programming Languages**
  - C, C++, C#, Objective-C, Java, JavaScript, Perl, PHP, Python, VB
  - Primitives, Operators and Standard Libraries

- **The Ada Programming Language**
  - Range Constraints
  - Real-Time Constructs
SEI CERT Coding Standards

**Overview**

- This site supports the development of coding standards for commonly used programming languages such as C, C++, Java, and Perl, and the Android™ platform.
- These standards are developed through a broad-based community effort by members of the software development and software security communities.

**Website**

- [https://www.securecoding.cert.org/confluence/display/seccode/SEI+CERT+Coding+Standards](https://www.securecoding.cert.org/confluence/display/seccode/SEI+CERT+Coding+Standards)
SEI Secure Coding Standard for Java

Rules

- 00 Input Validation and Data Sanitization (IDS)
- 01 Declarations and Initialization (DCL)
- 02 Expressions (EXP)
- 03 **Numeric Types and Operations (NUM)**
- 04 Characters and Strings (STR)
- 05 Object Orientation (OBJ)
- 06 Methods (MET)
- 07 Exceptional Behavior (ERR)
- 08 Visibility and Atomicity (VNA)
- 09 Locking (LCK)
- 10 Thread APIs (THI)
- 11 Thread Pools (TPS)
- 12 Thread-Safety Miscellaneous (TSM)
- 13 Input Output (FIO)
- 14 Serialization (SER)
- 15 Platform Security (SEC)
- 16 Runtime Environment (ENV)
- 17 Java Native Interface (JNI)
- 49 Miscellaneous (MSC)
- 50 Android (DRD)

Recommendations

- 00 Input Validation and Data Sanitization (IDS)
- 01 Declarations and Initialization (DCL)
- 02 Expressions (EXP)
- 03 **Numeric Types and Operations (NUM)**
- 04 Characters and Strings (STR)
- 05 Object Orientation (OBJ)
- 06 Methods (MET)
- 07 Exceptional Behavior (ERR)
- 13 Input Output (FIO)
- 15 Platform Security (SEC)
- 18 Concurrency (CON)
- 49 Miscellaneous (MSC)
- AA References
- BB Definitions
- CC Analyzers
Rule 03:
Numeric Types and Operations (NUM)

- **Rules**
  - NUM00-J. Detect or prevent integer overflow
    - NUM01-J. Do not perform bitwise and arithmetic operations on the same data
    - NUM02-J. Ensure that division and remainder operations do not result in divide-by-zero errors
    - NUM03-J. Use integer types that can fully represent the possible range of unsigned data
    - NUM04-J. Do not use floating-point numbers if precise computation is required
    - NUM07-J. Do not attempt comparisons with NaN
    - NUM08-J. Check floating point inputs for exceptional values
    - NUM09-J. Do not use floating point numbers as loop counters
    - NUM10-J. Do not construct BigDecimal objects from floating-point literals
    - NUM11-J. Do not compare or inspect the string representation of floating-point values
    - NUM12-J. Ensure conversions of numeric types to narrower types do not result in lost or misinterpreted data
    - NUM14-J. Use shift operators correctly
Programs must not allow mathematical operations to exceed the integer ranges provided by their primitive integer data types. According to *The Java Language Specification* (JLS), §4.2.2, "Integer Operations" [JLS 2015]:

- The built-in integer operators do not indicate overflow or underflow in any way.
- Integer operators can throw a `NullPointerException` if unboxing conversion of a null reference is required.
- Other than that, the only integer operators that can throw an exception are the integer divide operator `/` and the integer remainder operator `%`, which throw an `ArithmeticException` if the right-hand operand is zero, and the increment and decrement operators `++` and `--` which can throw an `OutOfMemoryError` if boxing conversion is required and there is insufficient memory to perform the conversion.
Root Cause Analysis & Resolution

**Issue**
- Integer Overflow/Underflow is Ignored (in Java)

**Possible Root Cause**
- Java Application Source Code
- Java Programming Language Implementation
- Java Programming Language Specification
- Java Virtual Machine Implementation
- Java Virtual Machine Specification
- Integer Math Processor Unit
- IEEE Standard 754

**Conclusion**
- Integer overflow/underflow indicator bits provided in IEEE 754 are ignored in the Java Programming Language Specification
- Java has flaws in INT primitive and operators +, -, *, /, >>>, >>, <<, etc.

**Resolution(s)**
- Use an infinitely ranged integer
- Raise a run-time constraint violation

SEI provides a full discussion of the Integer vulnerability and remedy in:
*As-If Infinitely Ranged Integer Model, Second Edition, April 2010*
## Top 10 Programming Languages

### TIOBE Index (2015)
1. Java (19.565%)
2. C++ (15.621%)
3. C# (6.782%)
4. Python (3.664%)
5. PHP (2.530%)
6. JavaScript (2.342%)
7. VB .NET (2.062%)
8. Perl (1.899%)
9. Objective-C (1.821%)

### PYPL Index (2014)
1. Java (25.5%)
2. PHP (11.4%)
3. Python (11.1%)
4. C# (9.2%)
5. C++ (7.7%)
6. JavaScript (7.3%)
7. Objective-C (5.3%)
8. VB .NET (2.1%)
9. Perl (1.3%)

http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html  
... based on number of web page references.

https://sites.google.com/site/pydatalog/pypl/python-blog/pythonisthelanguageoftheyear  
... based on number of Google searches.

Secure Coding Frameworks
Conclusions

The following problems were diagnosed as the root cause of the majority of cybersecurity vulnerability types (and safety/reliability issues) in software applications:

- **Programming Language Flaws**
  - Silent integer underflow/overflow in math operations
  - Silent floating point floors/ceilings in math operations
  - Silent loss of magnitude, sign and/or precision in numeric type casts

- **Programming Language Weaknesses**
  - Lack of user-defined range constraints and subsequent bounds checking on numeric data types to support input validation
  - Lack of bounds checking on array indexing resulting in buffer overflow
  - Lack of adequate built-in memory management of primitives to eliminate null pointer dereferencing

- **Standard Library Weaknesses**
  - Lack of specialized strings for filtering and validating characters and sequences in character strings (filenames, database names, SQL, URL, HTTP, LDAP, XSS, etc.)
The Secure Coding Framework

Problem:
- Mainstream programming languages have significant security vulnerabilities and weaknesses and their component libraries also have weaknesses that can be exploited
- To remedy this, developers can apply static analysis tools and rework software in accordance with the SEI Secure Coding Standards
- Developing secure code this way can be prohibitively difficult and expensive
- Mainstream programming languages were not designed for the development of secure applications

Solution:
- Provide developers with a Secure Coding Framework (SCF) protecting against the programming language’s inherent security vulnerabilities and component library flaws and/or misuse
- Replace (by wrapping) primitives and operators with secure classes and methods
- Use the SCF to simplify and expedite the development of safe, secure and reliable code
- Provide developers with a platform for the development of safe, secure and reliable software applications from the ground up

Secure Coding Frameworks
Lexical (Primitives):

- byte, byte[]
- char, char[]
- short, short[]
- int, int[]
- long, long[]
- float, float[]
- double, double[]
- String (class)

Classes:

- SecureByte, SecureByteArray
- SecureCharacter, SecureCharacterArray
- SecureShort, SecureShortArray
- SecureInteger, SecureIntegerArray
- SecureLong, SecureLongArray
- SecureFloat, SecureFloatArray
- SecureDouble, SecureDoubleArray
- SecureString, SecureSQLString, SecureURLString, etc.
Mainstream Programming Language Symbolic vs. JSCF Method Substitute …

<table>
<thead>
<tr>
<th>Symbolic (Operators):</th>
<th>Methods:</th>
</tr>
</thead>
<tbody>
<tr>
<td>▪ =</td>
<td>▪ equal()</td>
</tr>
<tr>
<td>▪ +, -, *, +=, -=, *=</td>
<td>▪ add(), subtract(), multiply(),</td>
</tr>
<tr>
<td>▪ /, %, /=, %=</td>
<td>▪ divide(), modulo()</td>
</tr>
<tr>
<td>▪ ==, !=</td>
<td>▪ equalTo(), EQ(), notEqualTo(), NEQ()</td>
</tr>
<tr>
<td>▪ &lt;, &lt;=, &gt;, &gt;=</td>
<td>▪ lessThan(), LT(),</td>
</tr>
<tr>
<td>▪ &gt;&gt;, &lt;&lt;, &gt;&gt;&gt;</td>
<td>▪ lessThanOrEqualTo(), LTE(),</td>
</tr>
<tr>
<td>▪ ++, --</td>
<td>▪ greaterThan(), GT(),</td>
</tr>
<tr>
<td>▪ &amp;,</td>
<td>, ^</td>
</tr>
</tbody>
</table>

rightShift(), leftShift(), rightShiftZero()

increment(), decrement()

bitwiseAnd(), bitwiseOr(), bitwiseXor()
JSCF Typecasting Methods …

Syntax:
- (byte)
- (char)
- (short)
- (int)
- (long)
- (float)
- (double)

JSCF Methods:
- toByte()
- toCharacter()
- toShort()
- toInteger()
- toLong()
- toFloat()
- toDouble()
Other Useful Methods of JSCF ...

Constructors:
- SecureByte(), SecureByteArray()
- SecureCharacter(), SecureCharacterArray()
- SecureShort(), SecureShortArray()
- SecureInteger(), SecureIntegerArray()
- SecureLong(), SecureLongArray()
- SecureFloat(), SecureFloatArray()
- SecureDouble(), SecureDoubleArray()
- SecureString()

User-Defined Ranges:
- range()
- minimum(), maximum()
- isByte(), isCharacter(), isShort(), isInteger(), isLong(), isFloat(), isDouble()

Interface to Legacy Code:
- value() – returns primitive/literal
- init() – init with primitive/literal
- index() – index with primitive/literal value
### Java

```java
import java.lang.System.out;
...

int inputAngle = 360;
...
public static final int MIN_ANGLE = 0;
public static final int MAX_ANGLE = 359;
int angle = 0;
...
if (inputAngle >= MIN_ANG && inputAngle <= MAX_ANG) {
    angle = inputAngle;
} else {
    System.out.println("Invalid input detected.");
    System.out.print("ANGLE =");
    System.out.println(inputAngle);
}
```

### JSCF

```java
import jscf.SecureInteger;
import jscf.RangeConstraintException;
...
SecureInteger inputAngle = new SecureInteger(360);
...
SecureInteger angle =
    new SecureInteger(0, 359);
...
try {
    angle.setEqualTo(inputAngle);
} catch (RangeConstraintException e) {
    e.printStackTrace();
}
```

---

Secure Coding Frameworks
Exception Handling in JSCF …

Vulnerability:
- Integer Overflow
- Integer Underflow
- Floating Point Floor
- Floating Point Ceiling
- Loss of Sign
- Loss of Magnitude
- Loss of Precision
- Range Constraint
- <etc> …

Exceptions:
- IntegerOverflowException
- IntegerUnderflowException
- <etc> …

Secure Coding Frameworks
Mainstream Programming Language Vulnerability vs. SCF Remedy Tactic ...

Vulnerability:
- Silent Integer Overflow / Underflow in Math Ops
- Silent Floating Point Floor / Ceiling in Math Ops
- Silent Loss of Sign / Magnitude / Precision in Narrowing Implicit / Explicit Type Conversions

Remedy Tactic:
- Exception Thrown / Handling for Integer Overflow / Underflow
- Exception Thrown / Handling for Floating Point Floor / Ceiling
- Exception Thrown / Handling for Loss of Sign / Magnitude / Precision in Narrowing Explicit Type Conversion
- No Implicit Type Conversions

Secure Coding Frameworks
Mainstream Programming Language
Vulnerability vs. SCF Remedy Tactic …

**Vulnerability:**
- Uninitialized Memory
- Memory Leaks
- Arbitrary Code Execution
- Stack Overflow/Overrun
- Heap Overflow/Overrun
- Null Pointer Dereferencing
- Dangling Pointers

**Remedy Tactics:**
- Constructor(s) Initialization
- Destructor/Finally/Other Deallocation
- Array Index Checking
- Array Index Checking
- No Pointers/No Primitives
- No Pointers/No Primitives
- No Pointers/No Primitives

Secure Coding Frameworks
Mainstream Programming Language Vulnerability vs. SCF Remedy Tactic …

- Invalid Parameter Inputs
- Direct Filename References *
- Direct Database References
- Network Functions *
- SQL Injection *
- URL Injection
- HTTP Injection
- LDAP Injection
- Cross-Site Scripting
- Cross-Site Request Forgery

- User-Defined Range Constraint
- Text Filters/Filename String Class
- Text Filters/Database String Class
- Network SSL Functions
- Text Filters/SQL String Class
- Text Filters/URL String Class
- Text Filters/HTTP String Class
- Text Filters/LDAP String Class
- Text Filters/XSS String Class
- Text Filters/HTTP String Class

Secure Coding Frameworks
Mainstream Programming Language  Gap vs. SCF New Feature …

Gaps:
- Vulnerable Primitives and Operators
- Lack of User-Defined Range Constraints on Primitive Values
- Lack of Character Filters on Strings & String Derivatives
- Limitations on Long Integer Values
- Limitations on Double Values
- No Type for Currency
- No Instrumentation

New Features:
- Secure Classes and Methods replace Vulnerable Primitives and Operators
- Exception Handling for User-Defined Range Constraints
- Exception Handling for Violations of Character Filters on Strings
- BigInteger Class with No Limits
- BigDouble Class with No Limits
- Big Decimal Class
- Exception Handling-based Instrumentation Hooks

Secure Coding Frameworks
Exception Handling and Instrumentation Hooks ...

- Trigger Alerts
- Trigger Error Messages
- Trigger Event Log
- Trigger E-Mail
- Trigger IPC Message
- Identify Programmer Errors
- Feedback to Software Vendor

- Support Application Monitoring
- Support Application Monitoring in the Cloud
- Support Application Monitoring across the Enterprise
- Situational Awareness across Product Deployment Area
- Identify Malicious Behavior
- Automated Real-Time Bug Reporting and Patch Management
Cyber security thru Lexical and Symbolic Proxy (CLaSP)

- Encapsulates and Substitutes Lexical Elements (private primitives) with Safe Classes
- Encapsulates and Substitutes Symbolic Elements (public operators) with Safe Methods
- Applies only to Object-Oriented (OO) Programming Languages

CLaSP is the patentable idea that defines the entire process of transforming any general purpose OO programming language (with inherent cyber security vulnerabilities) into a safe, secure and reliable coding platform.

- Security is the #1 priority software quality attribute of the Secure Coding Framework.
- USPO patent is pending.
Benefits of SCF ...

- Assures safe, secure and reliable source code in area of addressed vulnerabilities
- Reduces/eliminates the need for static analysis in area of addressed vulnerabilities
- Easy integration of new SCF code with legacy code
- No need to learn a new programming language
- Easy to learn
- Class and method naming conventions that echo that of the primitives and operators
- SCF source code baseline that conforms to the SEI Secure Coding Standard
- Supports SEI Secure Coding Standards for new development
Secure Coding Frameworks