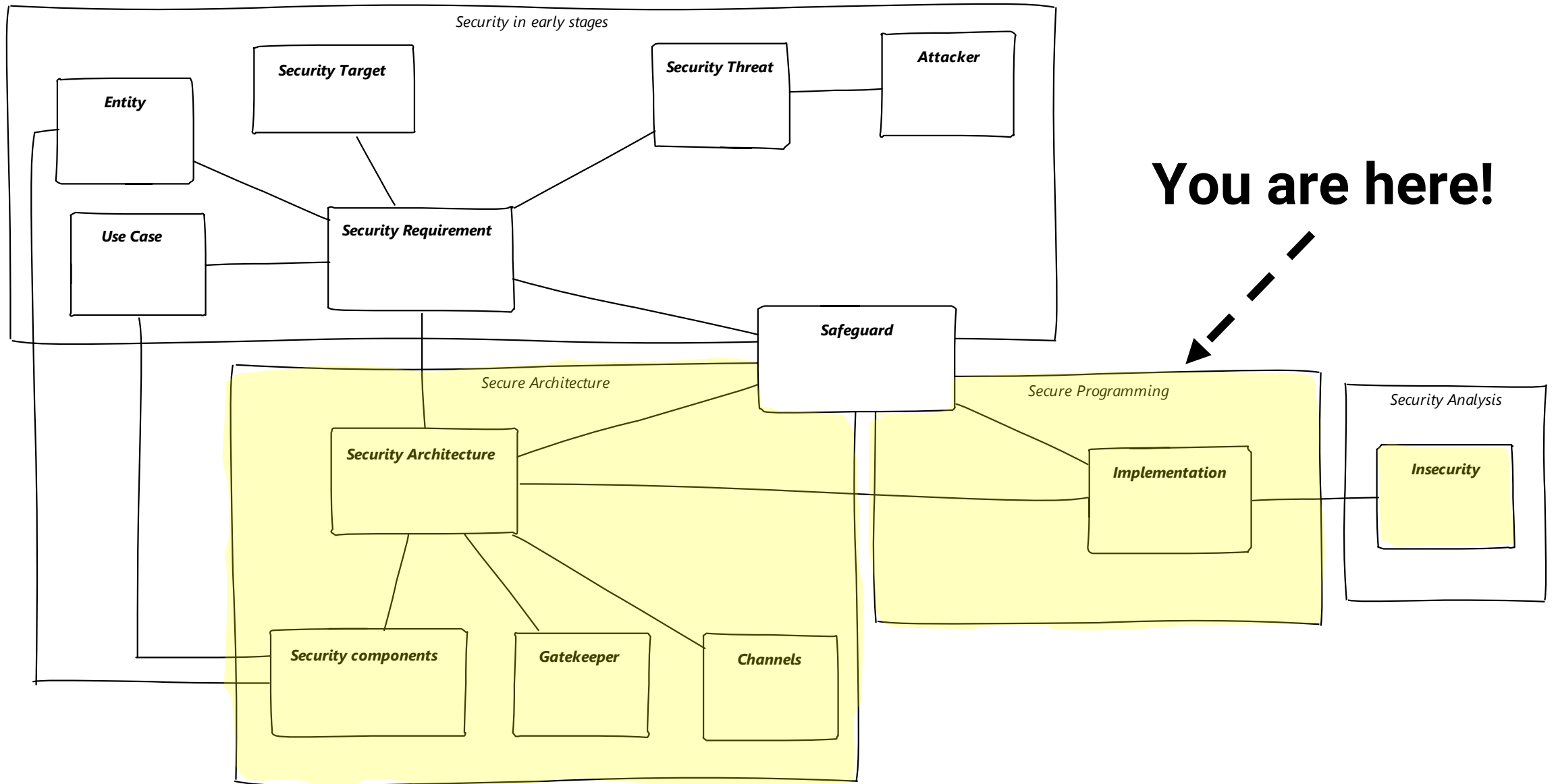


Secure JEE Architecture and Programming 101

Mario-Leander Reimer, Chief Technologist
Wednesday, Oct 28 @ JavaOne 2015

Security seems to be the
most underrated non functional
requirement in software
engineering.

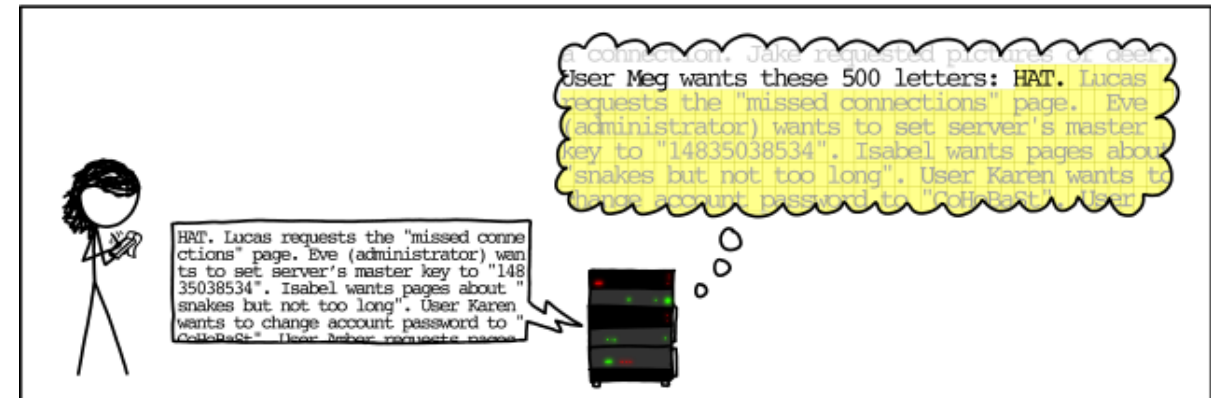
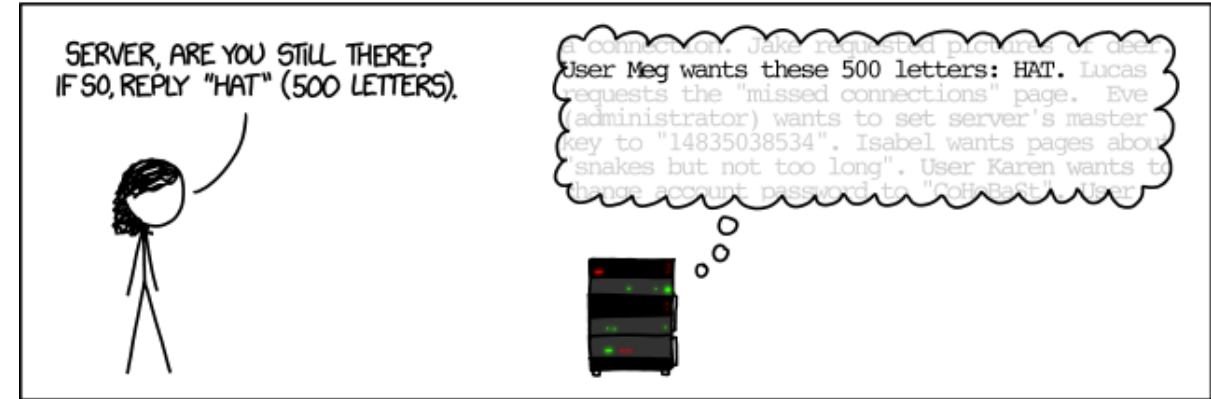
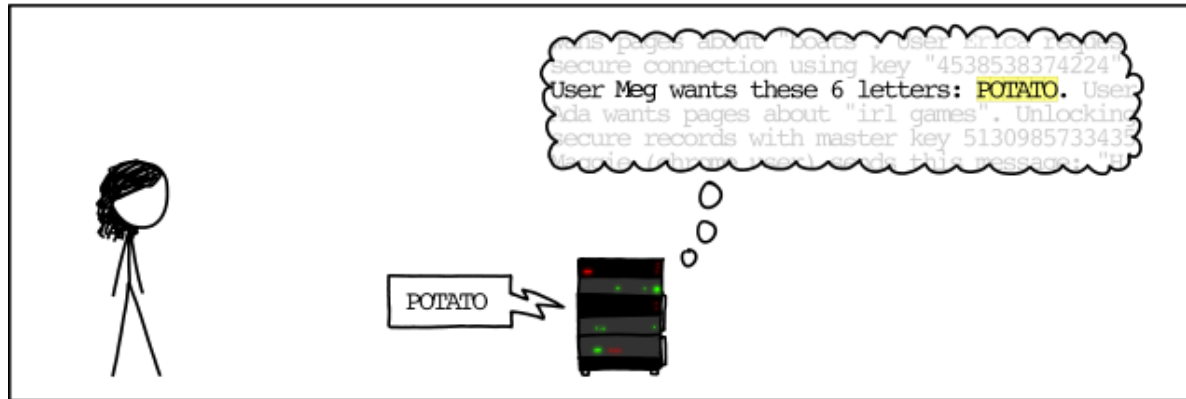
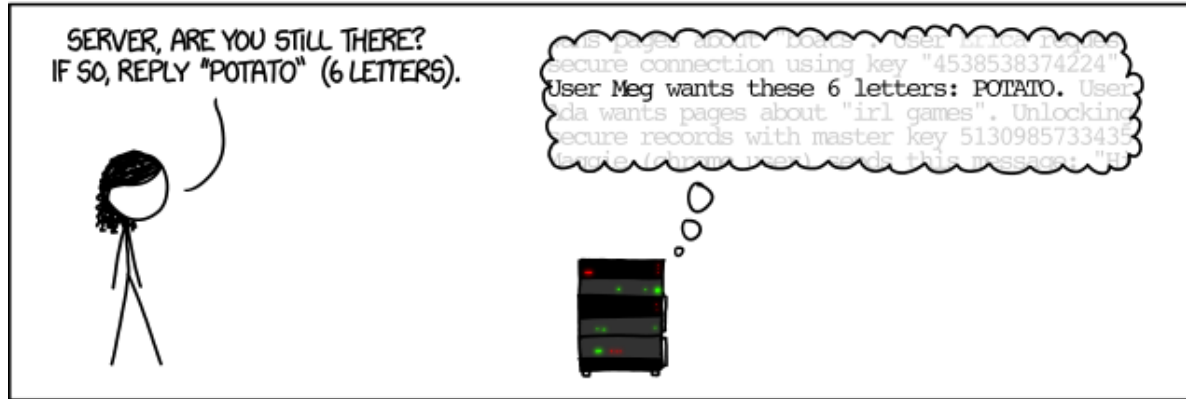


So what's on the agenda?

- The anatomy of two prominent security vulnerabilities
- Java as a secure programming language and platform
- Security Analysis: attacking an insecure JEE webapp
- Secure Programming Awareness: 221 rules for more secure code
- Secure Architecture: concepts and basic JEE features

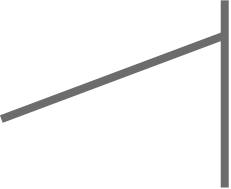
HOW THE HEARTBLEED BUG WORKS:

<http://xkcd.com/1354/>



**The Java exploit for Heartbleed only had 186 lines of code.
The patch for Heartbleed only added 8 lines of code.**

```
-    /* Read type and payload length first */
-    hbtype = *p++;
-    n2s(p, payload);
-    pl = p;
-
-    if (s->msg_callback)
-        s->msg_callback(0, s->version, TLS1_RT_HEARTBEAT,
-            &s->s3->rrec.data[0], s->s3->rrec.length,
-            s, s->msg_callback_arg);
+
+    /* Read type and payload length first */
+    if (1 + 2 + 16 > s->s3->rrec.length)
+        return 0; /* silently discard */
+    hbtype = *p++;
+    n2s(p, payload);
+    if (1 + 2 + payload + 16 > s->s3->rrec.length)
+        return 0; /* silently discard per RFC 6520 sec. 4 */
+    pl = p;
+
```



Bounds check for the
correct record length

Apple's SSL bug: goto fail;

```
static OSStatus
SSLVerifySignedServerKeyExchange(SSLContext *ctx, bool isRsa, SSLBuffer signedParams,
                                uint8_t *signature, UInt16 signatureLen)
{
    OSStatus err;
    SSLBuffer hashOut, hashCtx, clientRandom, serverRandom;
    uint8_t hashes[SSL_SHA1_DIGEST_LEN + SSL_MD5_DIGEST_LEN];
    SSLBuffer signedHashes;
    uint8_t *dataToSign;
    size_t dataToSignLen;

    signedHashes.data = 0;
    hashCtx.data = 0;

    clientRandom.data = ctx->clientRandom;
    clientRandom.length = SSL_CLIENT_SRVR_RANDOM_SIZE;
    serverRandom.data = ctx->serverRandom;
    serverRandom.length = SSL_CLIENT_SRVR_RANDOM_SIZE;

    if(isRsa) {
        /* skip this if signing with DSA */
        dataToSign = hashes;
        dataToSignLen = SSL_SHA1_DIGEST_LEN + SSL_MD5_DIGEST_LEN;
        hashOut.data = hashes;
        hashOut.length = SSL_MD5_DIGEST_LEN;

        if ((err = ReadyHash(&SSLHashMD5, &hashCtx)) != 0)
            goto fail;
        if ((err = SSLHashMD5.update(&hashCtx, &clientRandom)) != 0)
            goto fail;
    }
}
```


Apple's SSL bug: goto fail;

```
if ((err = ReadyHash(&SSLHashSHA1, &hashCtx)) != 0)
    goto fail;
if ((err = SSLHashSHA1.update(&hashCtx, &clientRandom)) != 0)
    goto fail;
if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
    goto fail;
if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
    goto fail;
goto fail;
if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
    goto fail;
```

Always goto fail;

Never called.

```
err = sslRawVerify(ctx,
                  ctx->peerPubKey,
                  dataToSign,
                  dataToSignLen,
                  signature,
                  signatureLen);
/* plaintext */
/* plaintext length */

if(err) {
    sslErrorLog("SSLDecodeSignedServerKeyExchange: sslRawVerify "
               "returned %d\n", (int)err);
    goto fail;
}
```

```
fail:
    SSLFreeBuffer(&signedHashes);
    SSLFreeBuffer(&hashCtx);
    return err;
```

Success!? Not really what you would expect.

Probably all security
vulnerabilities are caused by
poor, negligent or just plain
unsafe programming!


Java CPU and PSU Releases Explained.

- Java SE Critical Patch Updates (CPU)

- Odd version numbers: 8u31, 8u05, 7u71, 7u65, 7u45, ...
- Fixes for known security vulnerabilities
- Further severe bug fixes
- **Recommendation:** upgrade as soon as possible after it has been released

- Java SE Patch Set Updates (PSU)

- Even version numbers: 8u40, 8u20, 7u72, 7u60, ...
- All fixes of the CPU release
- Further non-critical fixes and enhancements
- **Recommendation:** only upgrade if non-critical fix is required

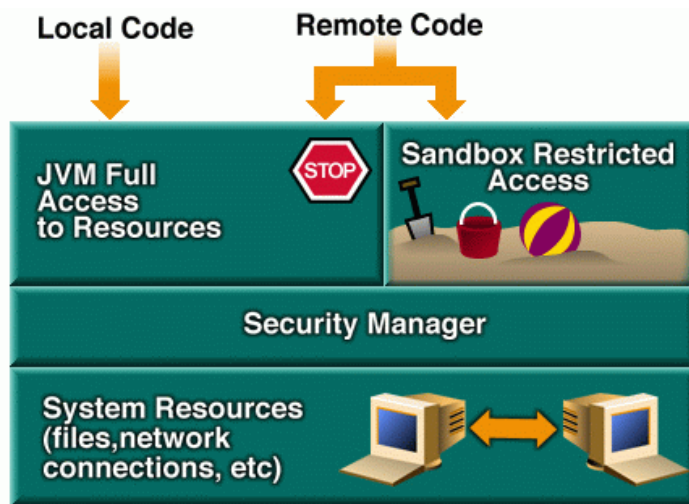


**Subscribe to
Critical Patch
Update Alert
Emails Today**

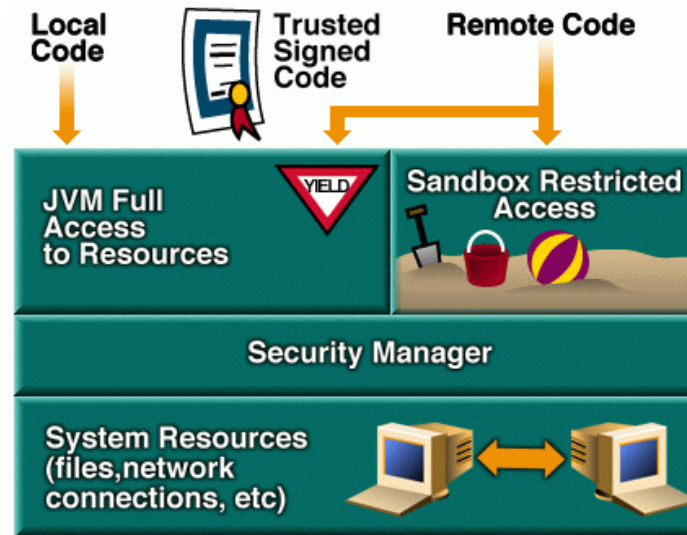
Java has been designed with security in mind from the start. Java is a secure programming language and platform.

- The JVM and the Java language provide several features and APIs for secure programming
 - Bytecode verification, memory management, sandbox model, security manager, ...
 - The `java.security` package in JDK8 contains 15 interfaces, 54 classes, 3 enums, 16 exceptions
 - Configurable, fine-grained access control
 - cryptographic operations such as message digest and signature generation
 - Support for generation and storage of cryptographic public keys
- The security features are constantly improved and developed, such as resource consumption management, object-level protection, arbitrary permission grouping, ...

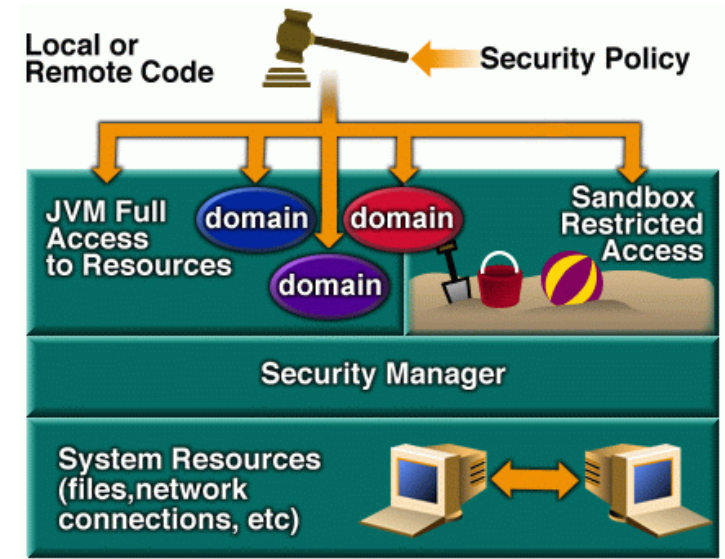
The evolution of the Java security model. It hasn't changed much since.



JDK 1.0 Security Model
(1996)



JDK 1.1 Security Model
(1997)



Java 2 Security Model
(1998)

The default Java security policy file is very restrictive. But ...

\$JAVA_HOME/
jre/lib/security/java.policy

```
// Standard extensions get all permissions by default
grant codeBase "file:${java.ext.dirs}/*" {
    permission java.security.AllPermission;
};

// default permissions granted to all domains
grant {
    // allows anyone to listen on dynamic ports
    permission java.net.SocketPermission "localhost:0", "listen";
    // permission for standard RMI registry port
    permission java.net.SocketPermission "localhost:1099", "listen";

    // "standard" properties that can be read by anyone
    permission java.util.PropertyPermission "java.version", "read";
    permission java.util.PropertyPermission "java.vendor", "read";
    permission java.util.PropertyPermission "java.vendor.url", "read";
    permission java.util.PropertyPermission "java.class.version", "read";
    permission java.util.PropertyPermission "os.name", "read";
    permission java.util.PropertyPermission "os.version", "read";
    permission java.util.PropertyPermission "os.arch", "read";
    permission java.util.PropertyPermission "file.separator", "read";
    permission java.util.PropertyPermission "path.separator", "read";
    permission java.util.PropertyPermission "line.separator", "read";

    permission java.util.PropertyPermission "java.specification.version", "read";
    permission java.util.PropertyPermission "java.specification.vendor", "read";
    permission java.util.PropertyPermission "java.specification.name", "read";

    permission java.util.PropertyPermission "java.vm.specification.version", "read";
    permission java.util.PropertyPermission "java.vm.specification.vendor", "read";
    permission java.util.PropertyPermission "java.vm.specification.name", "read";
    permission java.util.PropertyPermission "java.vm.version", "read";
    permission java.util.PropertyPermission "java.vm.vendor", "read";
    permission java.util.PropertyPermission "java.vm.name", "read";
};
```

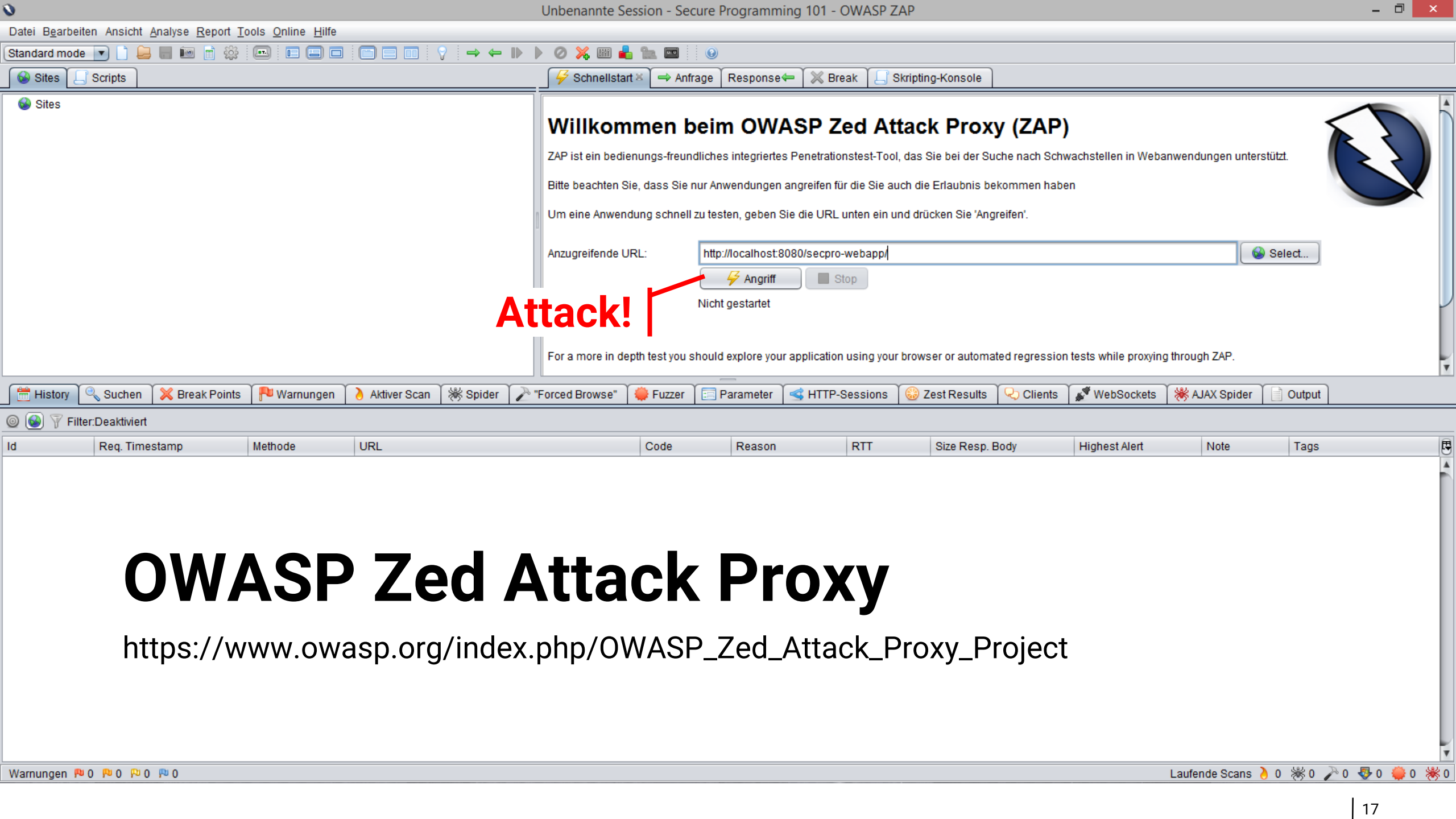
... if you allow everything and don't pay attention, don't blame others.



No magic provided!

It is up to us developers and
architects to use and apply the
Java security features.

How do I know my web application
has security vulnerabilities?



Attack!

OWASP Zed Attack Proxy

https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project

Willkommen beim OWASP Zed Attack Proxy (ZAP)

ZAP ist ein bedienungs-freundliches integriertes Penetrationstest-Tool, das Sie bei der Suche nach Schwachstellen in Webanwendungen unterstützt.

Bitte beachten Sie, dass Sie nur Anwendungen angreifen für die Sie auch die Erlaubnis bekommen haben

Um eine Anwendung schnell zu testen, geben Sie die URL unten ein und drücken Sie 'Angreifen'.

Anzugreifende URL:

Fortschritt:

Angriff abgeschlossen - Sehen Sie in den Alert-Reiter für Details von gefundenen Problemen

For a more in depth test you should explore your application using your browser or automated regression tests while proxying through ZAP.

Vollständige Details aller ausgewählter Warnungen werden hier angezeigt.

Separate manuelle Warnungen können ausserdem über das Rechtsklick-Menü auf jedem beliebigen URL hinzugefügt werden.

Bestehende Warnungen können mittels Doppelklick bearbeitet werden.

One inconsiderate line of code can be the root of all evil ...

```
@WebServlet(name = "DownloadServlet", urlPatterns = "/download")
public class DownloadServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
        throws ServletException, IOException {

        // translate src parameter to full file system path
        String src = req.getParameter("src");
        File file = new File(getServletContext().getRealPath("/"), "/" + src);

        if (file.exists() && file.canRead() && file.isFile()) {
            // copy file contents to servlet output stream
            Files.copy(file.toPath(), resp.getOutputStream());
        } else {
            // the file does not exist
            resp.sendError(404);
        }
    }
}
```

Usage of raw request parameter

How can we do better?

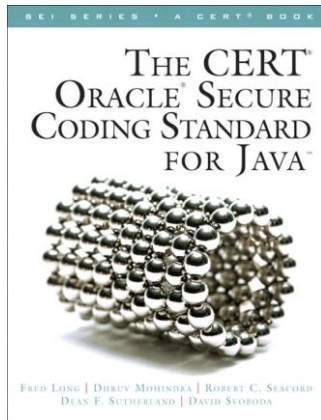
Only 3 sources and 221 rules for more secure and better code.

ORACLE®

Secure Coding Guidelines for Java SE

Updated for Java SE 8, Version: 5.0, Last updated: 25 September 2014

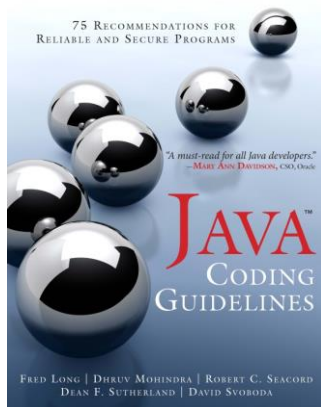
<http://www.oracle.com/technetwork/java/seccodeguide-139067.html>



The CERT™ Oracle™ Secure Coding Standard for Java

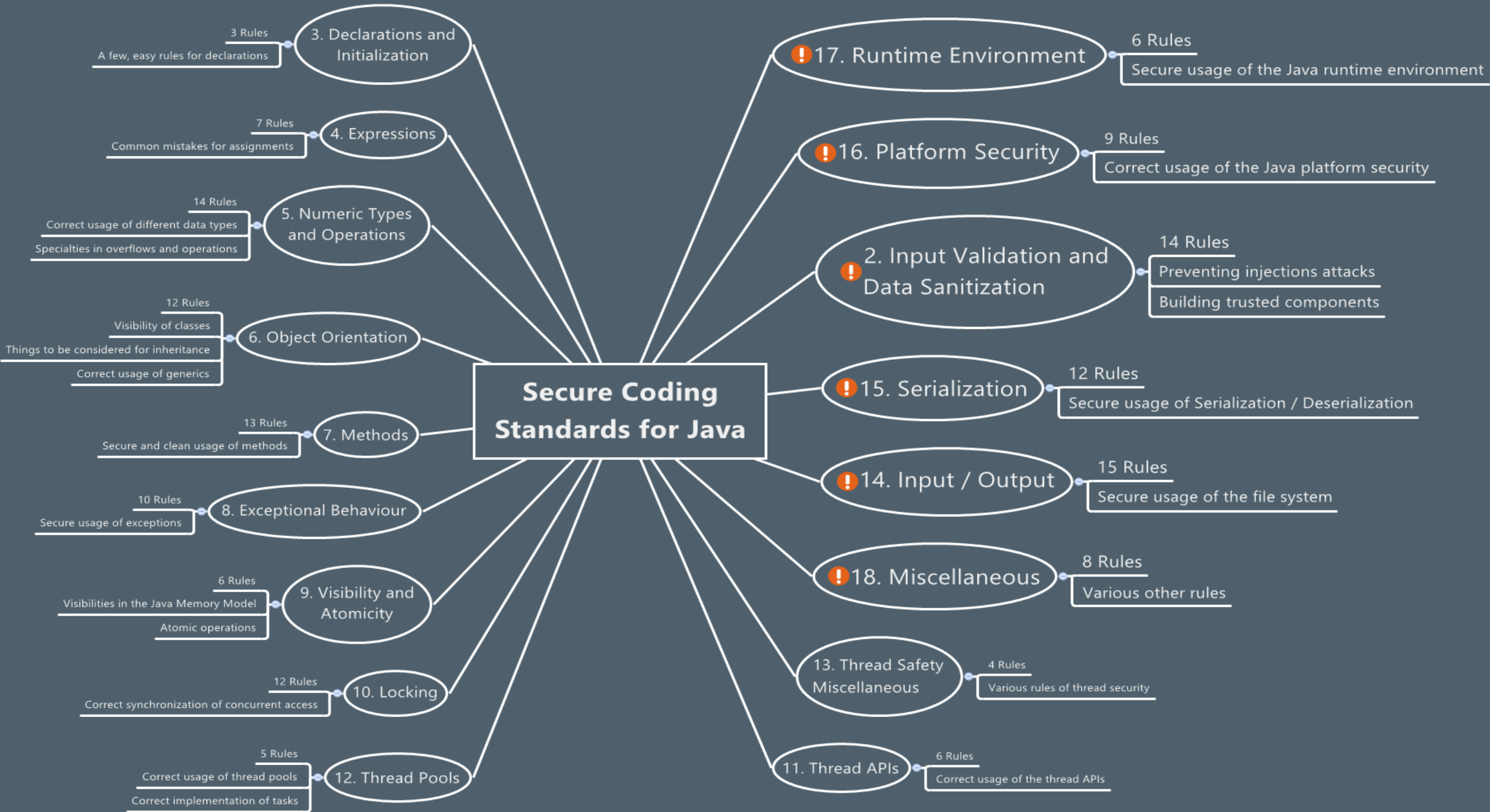
Fred Long, Dhruv Mohindra, Robert C. Seacord,
Dean F. Sutherland, David Svoboda

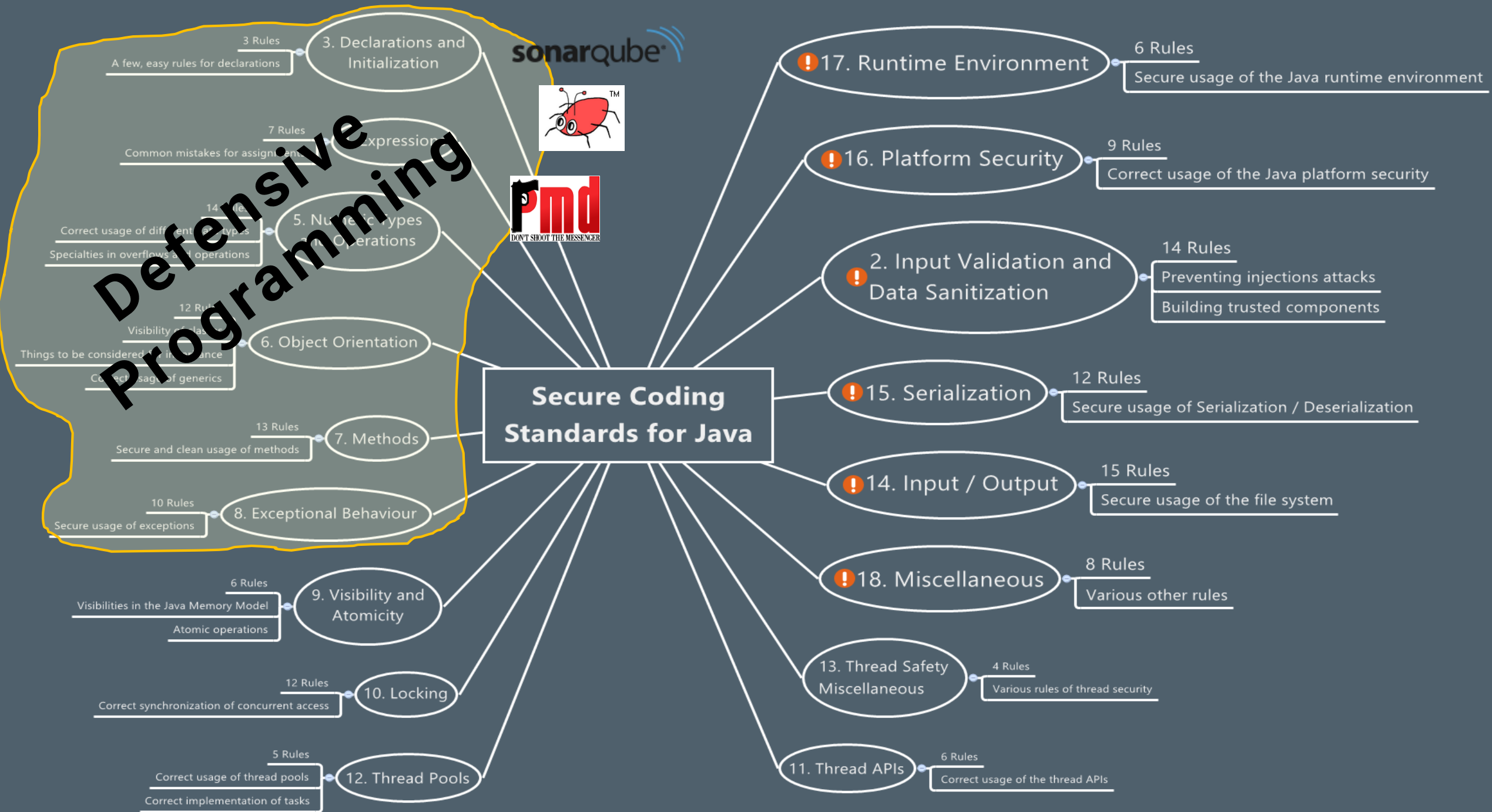
Rules available online at www.securecoding.cert.org

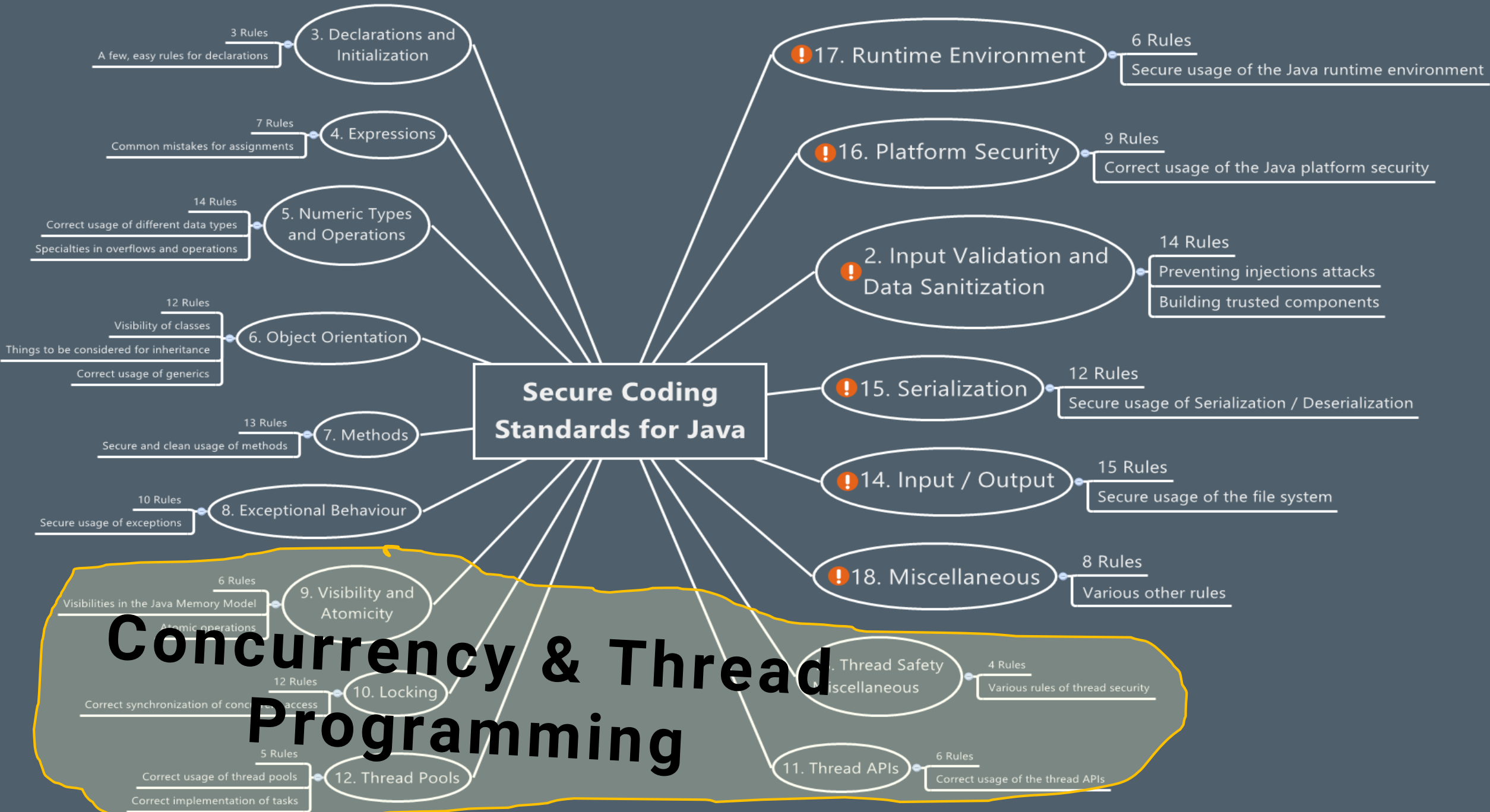


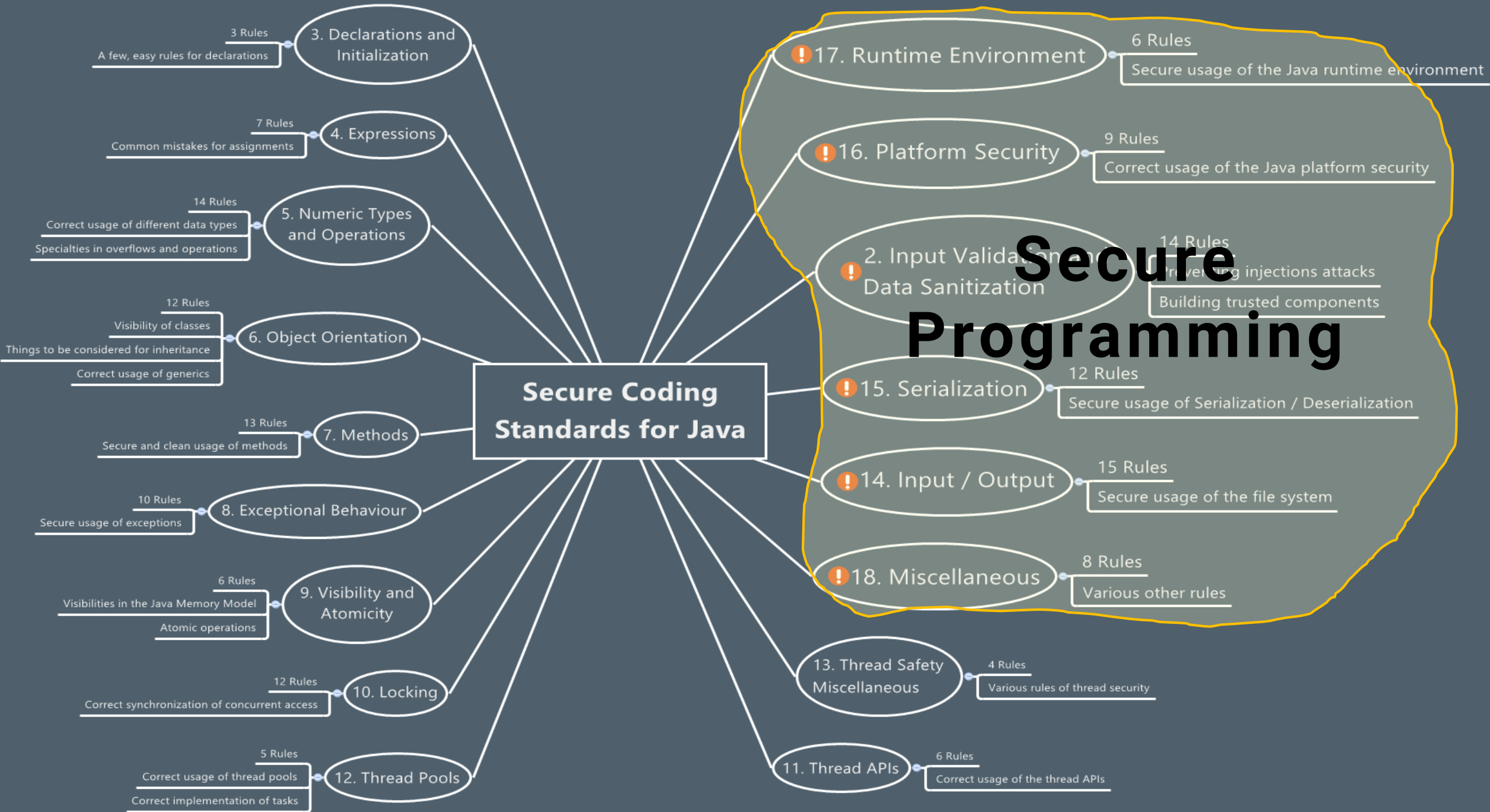
Java Coding Guidelines

Fred Long, Dhruv Mohindra, Robert C. Seacord,
Dean F. Sutherland, David Svoboda









MSC03-J. Never hard code sensitive information.

What's the problem?

Sensitive information should never be hard coded. If the system is compromised, this information can be easily retrieved. Access to further resources may be possible.

How can you exploit the code?

Simply by disassembling the relevant code, using tools like *javap*, *JAD*, *dirtyJOE*.

How can we do better?

- Obtain information from a secure configuration file, system property or environment var.
- Use infrastructure security features such as password aliases in Glassfish.

A very very very ... bad example of a login component.

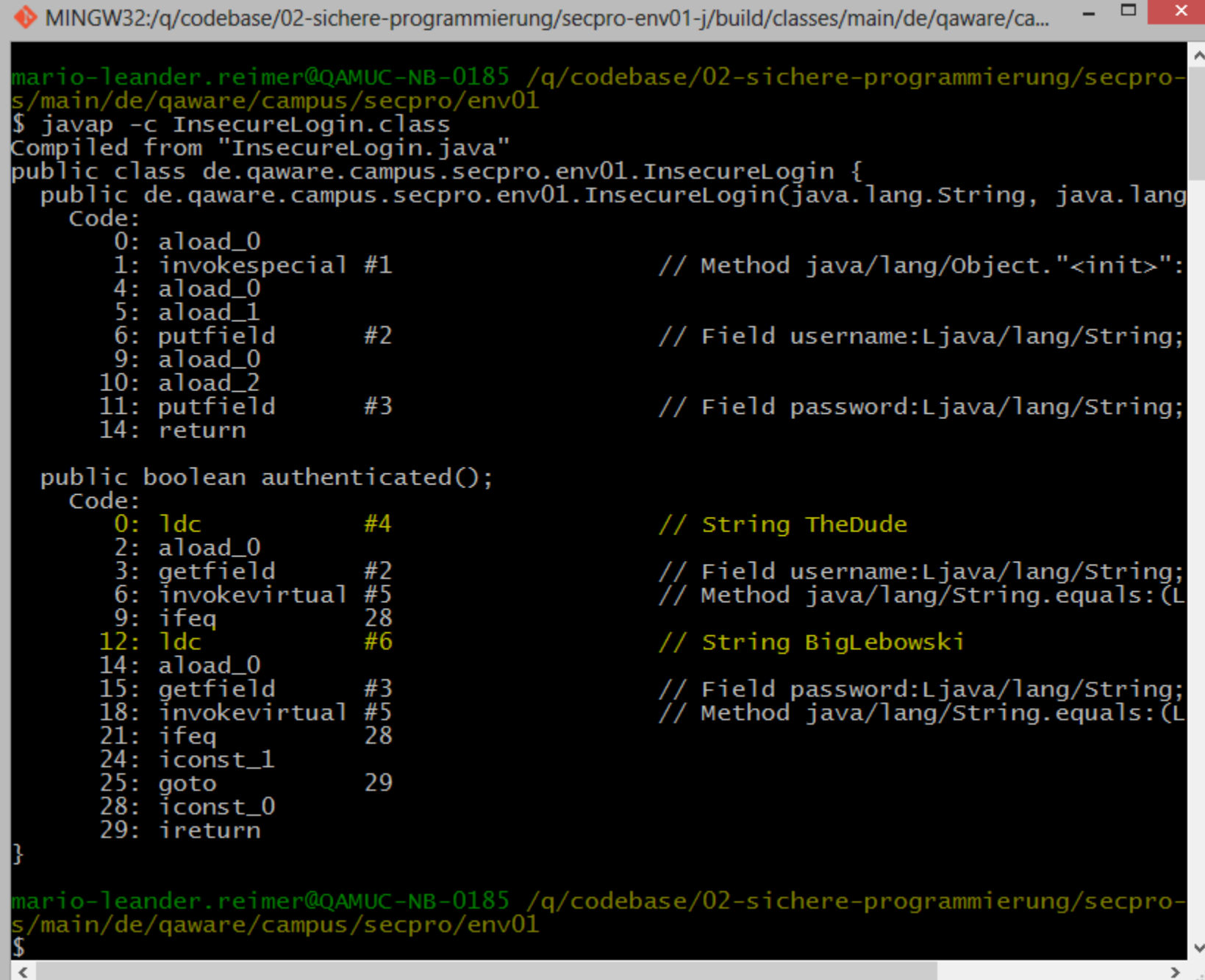
```
public class InsecureLogin {  
  
    private static final String USERNAME = "TheDude";  
    private static final String PASSWORD = "BigLebowski";  
  
    private final String username;  
    private final String password;  
  
    public InsecureLogin(String username, String password) {  
        this.username = username;  
        this.password = password;  
    }  
  
    public boolean authenticated() {  
        return USERNAME.equals(username)  
            && PASSWORD.equals(password);  
    }  
}
```



Please don't do this!

javap -c InsecureLogin.class

```
public class InsecureLogin {  
  
    private static final String USERNAME;  
    private static final String PASSWORD;  
  
    private final String username;  
    private final String password;  
  
    public InsecureLogin(String username,  
        String password) {  
        this.username = username;  
        this.password = password;  
    }  
  
    public boolean authenticated() {  
        return USERNAME.equals(username)  
            && PASSWORD.equals(password);  
    }  
}
```



```
MINGW32:/q/codebase/02-sichere-programmierung/secpro-env01-j/build/classes/main/de/qaware/ca...  
mario-leander.reimer@QAMUC-NB-0185 /q/codebase/02-sichere-programmierung/secpro-  
s/main/de/qaware/campus/secpro/env01  
$ javap -c InsecureLogin.class  
Compiled from "InsecureLogin.java"  
public class de.qaware.campus.secpro.env01.InsecureLogin {  
    public de.qaware.campus.secpro.env01.InsecureLogin(java.lang.String, java.lang.  
        Code:  
        0: aload_0  
        1: invokespecial #1           // Method java/lang/Object."<init>":  
        4: aload_0  
        5: aload_1  
        6: putfield      #2           // Field username:Ljava/lang/String;  
        9: aload_0  
       10: aload_2  
       11: putfield      #3           // Field password:Ljava/lang/String;  
       14: return  
  
    public boolean authenticated();  
        Code:  
        0: ldc           #4           // String TheDude  
        2: aload_0  
        3: getfield      #2           // Field username:Ljava/lang/String;  
        6: invokevirtual #5           // Method java/lang/String.equals:(L  
        9: ifeq          28  
       12: ldc           #6           // String BigLebowski  
       14: aload_0  
       15: getfield      #3           // Field password:Ljava/lang/String;  
       18: invokevirtual #5           // Method java/lang/String.equals:(L  
       21: ifeq          28  
       24: iconst_1  
       25: goto          29  
       28: iconst_0  
       29: ireturn  
  
}
```

javap -c MoreSecureLogin.class

```
public class MoreSecureLogin {  
  
    private static final char[] USERNAME;  
    private static final char[] PASSWORD;  
  
    private final String username;  
    private final char[] password;  
  
    public MoreSecureLogin(String username,  
        char[] password) {  
        this.username = username;  
        this.password = password;  
    }  
  
    public boolean authenticated() {  
        return Arrays.equals(USERNAME, password)  
            && Arrays.equals(PASSWORD, password);  
    }  
}
```

MINGW32:/q/codebase/02-sichere-programmierung/secpro-env01-j/build/classes/main/de/qaware/ca...

```
mario-leander.reimer@QAMUC-NB-0185 /q/codebase/02-sichere-programmierung/secpro-  
s/main/de/qaware/campus/secpro/env01  
$ javap -c MoreSecureLogin.class  
Compiled from "MoreSecureLogin.java"  
public class de.qaware.campus.secpro.env01.MoreSecureLogin {  
    public de.qaware.campus.secpro.env01.MoreSecureLogin(java.lang.String, char[])  
        Code:  
        0: aload_0  
        1: invokespecial #1           // Method java/lang/Object."<init>":  
        4: aload_0  
        5: aload_1  
        6: putfield      #2           // Field username:Ljava/lang/String;  
        9: aload_0  
       10: aload_2  
       11: putfield      #3           // Field password:[C  
       14: return  
  
    public boolean authenticated();  
        Code:  
        0: getstatic     #4           // Field USERNAME:[C  
        3: aload_0  
        4: getfield      #2           // Field username:Ljava/lang/String;  
        7: invokevirtual #5           // Method java/lang/String.toCharArray():[C  
       10: invokestatic  #6           // Method java/util/Arrays.equals:([C[C  
       13: ifeq         33  
       16: getstatic     #7           // Field PASSWORD:[C  
       19: aload_0  
       20: getfield      #3           // Field password:[C  
       23: invokestatic  #6           // Method java/util/Arrays.equals:([C[C  
       26: ifeq         33  
       29: iconst_1  
       30: goto         34  
       33: iconst_0  
       34: ireturn  
  
    static {};  
        Code:  
        0: bipush       7
```


Using password aliases is a much more secure option. And Java EE Security API 1.0 (JSR 375) is on it's way.

```
asadmin> create-password-alias
Enter the value for the aliasname operand> secpro_password_alias
Enter the alias password> qwertz123
Enter the alias password again> qwertz123
Command create-password-alias executed successfully.
```

This will be replaced by the container automatically.

```
-Dmaster.password=${ALIAS=secpro_password_alias}
```

PBKDF2WithHmacSHA1

```
secure.password=tvvtCEwfdmUAzXaKKlYQM6XYIjgQHxCZHZG/8SbdBQ+Vk9yH7PDK+x0aIgSZ2pvfWbC0avXyF30w+tWleYlnideYwXpyJXrkHv+DRdQthEmM=
```

```
secure.password.Production=r7mCJogt0VUI8s3UKJ1IHgCJ65pllW8q8uZ39+KjsvT910/iBppLt/8gNTGok/w1wscS7E24zLQKCOBbBZTU9A==
```

MLR01-J. Limit lifetime and visibility of sensitive information.

What's the problem?

Application scoped security information also ends up in your heap memory. The garbage collection only frees unreachable objects.

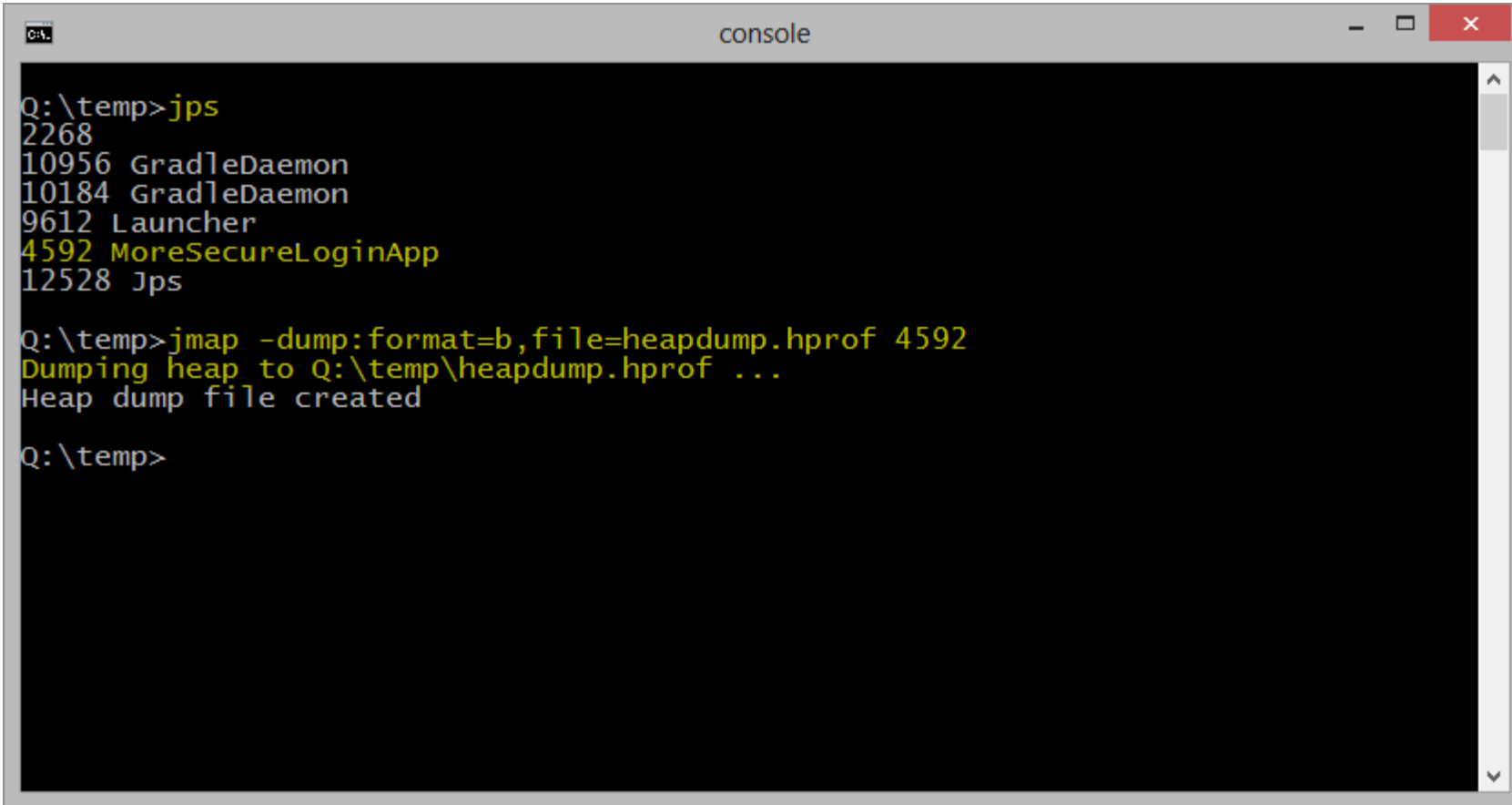
How can you exploit the code?

By taking a heap dump and analysing it, using tools like *jps + jmap*, *VisualVM*, *Eclipse MAT*

How can we do better?

- Use security sensitive information only method locally (parameters, variables)
- Clear or overwrite sensitive information after usage, e.g. `Arrays.fill(chars, \0);`

Taking heap dumps with JDK tools is simple. Use the command line or tools like Java VisualVM.



```
Q:\temp>jps
2268
10956 GradleDaemon
10184 GradleDaemon
9612 Launcher
4592 MoreSecureLoginApp
12528 Jps

Q:\temp>jmap -dump:format=b,file=heapdump.hprof 4592
Dumping heap to Q:\temp\heapdump.hprof ...
Heap dump file created

Q:\temp>
```

Heap Dump Analysis.

```
C:\>
Q:\temp>jps
2268
10956 GradleDaemon
10184 GradleDaemon
9612 Launcher
4592 MoreSecureLoginApp
12528 Jps

Q:\temp>jmap -dump:format=b
Dumping heap to Q:\temp\heap
Heap dump file created

Q:\temp>
```

Java VisualVM

File Applications View Tools Window Help

Start Page [heapdump] heapdump.hprof

[heapdump] heapdump.hprof

Heap Dump

Summary Classes Instances OQL Console

de.qaware.campus.secpo.env01.MoreSecureLogin Instances: 1 | Instance size: 32 | Total size: 32 | Compute Retained Sizes

Instances

Fields

References

Instance	Field	Type	Value
#1	this	MoreSecureLogin	#1
	password	char[]	#2201 ("BigLebowski")
	username	String	#1666
	hash32	int	0
	hash	int	0
	value	char[]	#2198 ("TheDude")
	HASHING_SEED	int	-361586141
	CASE_INSENSITIVE_ORDER	String\$CaseInsensitiveComparator	#1
	serialPersistentFields	ObjectStreamField[]	#1 (0 items)
	serialVersionUID	long	-6849794470754667710
	<classLoader>	<object>	null
	PASSWORD	char[]	#2235 ("BigLebowski")
	[0]	char	B
	[1]	char	i
	[2]	char	g
	[3]	char	L
	[4]	char	e
	[5]	char	b
	[6]	char	o
	[7]	char	w
	[8]	char	s
	[9]	char	k
	[10]	char	i
	USERNAME	char[]	#2234 ("TheDude")

Field	Type	Value
this (Java frame)	MoreSecureLogin	#1
<no references>	<none>	<none>

Array type | Object type | Primitive type | Static field | GC Root | Loop

Clear sensitive information after usage.

```
public final class ImprovedLoginApp {  
  
    public static void main(String[] args) {  
        Console console = System.console();  
  
        String username = console.readLine("Enter username: ");  
        char[] password = console.readPassword("Enter password: ");  
  
        ImprovedLogin login = new ImprovedLogin();  
        boolean authenticated = login.authenticated(username, password);  
  
        obfuscate(username);  
        obfuscate(password);  
  
        if (!authenticated) {  
            console.printf("Authentication failure. Exit.");  
            System.exit(1);  
        }  
  
        String command = null;  
        while (!"exit".equalsIgnoreCase(command)) {  
            command = console.readLine("$ ");  
        }  
    }  
}
```

Limited lifetime of sensitive information: method parameters.

Magic happens here!
Sensitive information is replaced with junk data.

Heap Dump Analysis.

```
C:\>
Q:\temp>jps
2268
10956 GradleDaemon
10184 GradleDaemon
11484 Main
14096 ImprovedLoginApp
9612 Launcher
5156 Jps

Q:\temp>jmap -dump:format=b
Dumping heap to Q:\temp\heap
Heap dump file created

Q:\temp>
```

The screenshot displays the Java VisualVM application window. The title bar reads "Java VisualVM". The menu bar includes "File", "Applications", "View", "Tools", "Window", and "Help". The toolbar contains icons for "Start Page", "Heap Dump", "Summary", "Classes", "Instances", and "OQL Console". The main pane is titled "[heapdump] heapdump2.hprof" and shows a "Heap Dump" view with a "Query Results" table. The table lists two entries: "java.lang.String#1666" and "java.lang.String#1751". Below the main pane is a "Query Editor" window with the following SQL query: `select s from java.lang.String s where s.toString() == '???????'`. An arrow points from the text "select s from java.lang.String s where s.toString() == '???????' " to the "Query Editor" window. To the right of the "Query Editor" is a "Saved Queries" panel showing a tree structure with folders "Custom", "Samples", and "PermGen Analysis". The "Custom" folder is expanded, showing sub-items: "InsecureLogin", "InsecureLogin Referrers", "Find TheDude", and "MoreSecureLogin". The "Find TheDude" item is selected. At the bottom of the "Saved Queries" panel are buttons for "Properties", "Delete", and "Open". The "Query Editor" window has a "Save" button at the bottom left and an "Execute" button at the bottom right.

Java VisualVM

File Applications View Tools Window Help

Start Page x [heapdump] heapdump.hprof x [heapdump] heapdump2.hprof x

[heapdump] heapdump2.hprof

Heap Dump

Summary Classes Instances OQL Console

Query Results

java.lang.String#1666

java.lang.String#1751

select s from java.lang.String s where s.toString() == '???????'

Query Editor

Saved Queries

Custom

- InsecureLogin
- InsecureLogin Referrers
- Find TheDude
- MoreSecureLogin

Samples

PermGen Analysis

Save Execute Properties Delete Open

ENV01-J. Place all security-sensitive code in a single JAR and sign and seal it.

What's the problem?

Without additional protection a JAR can be modified by an attacker. Any ***package*** or ***package private*** visibility can be circumvented in open packages.

How can you exploit the code?

- Exchange of classes, direct manipulation of byte code or important configuration files.
- Malicious inheritance with package and class definitions in foreign JAR files.

How can we do better?

Sign the relevant JARs to detect modification. Seal the JAR to prevent malicious inheritance.

USERNAME.equals(username) &&
Arrays.*equals*(***PASSWORD***, password)

```
00000000 : ldc          "SomeUsername"
00000002 : aload_1
00000003 : invokevirtual boolean java.lang.String.equals(java.lang.Object)
00000006 : ifeq         pos.00000017
00000009 : getstatic   char[] de.qaware.campus.secpro.env01.CrackedLogin.PASSWORD
0000000C : aload_2
0000000D : invokestatic boolean java.util.Arrays.equals(char[], char[])
00000010 : ifeq         pos.00000017
00000013 : iconst_1
00000014 : goto        pos.00000018
00000017 : iconst_0
00000018 : ireturn
```

! *USERNAME*.equals(username) &&
!Arrays.equals(*PASSWORD*, password)

```
00000000 : ldc          "SomeUsername"
00000002 : aload_1
00000003 : invokevirtual boolean java.lang.String.equals(java.lang.Object)
00000006 : ifne         pos.00000017
00000009 : getstatic   char[] de.qaware.campus.secpro.env01.CrackedLogin.PASSWORD
0000000C : aload_2
0000000D : invokestatic boolean java.util.Arrays.equals(char[], char[])
00000010 : ifne         pos.00000017
00000013 : iconst_1
00000014 : goto        pos.00000018
00000017 : iconst_0
00000018 : ireturn
```

ifeq
99 00 11



ifne
9A 00 11

Example MANIFEST.MF for a signed and sealed JAR.

Manifest-Version: 1.0
Sealed: true

A sealed JAR specifies that all packages defined by that JAR are sealed.

Name: de/qaware/campus/secpro/env01/DefaultDudeLogin.class
SHA-256-Digest: rrkKKGa0bjV3Hq6eIPIUr6CiQ54pXVB1D1pMzSLUdYg=

Name: de/qaware/campus/secpro/env01/LoginFactory.class
SHA-256-Digest: iDsZpu6sIvcTi5L907wemTARzWqNPQn2B07eqxqXclI=

Name: META-INF/services/de.qaware.campus.secpro.env01.Login
SHA-256-Digest: lyA9+HaJRTIKzn1vqCi/IQ7jhMHTZNGrf3aAznYQYss=

Name: de/qaware/campus/secpro/env01/UnmodifiableLoginApp.class
SHA-256-Digest: FUBXb0usoxcn4fSfLLYhDDLYABZbPJuDvypkXcek9dI=

Name: de/qaware/campus/secpro/env01/Login.class
SHA-256-Digest: lHXni+e9baryeHHrAesGd3CwHCUGfQ4mcV9AF0mtLmM=

Each file in the archive is given a digest entry in the archive's manifest.

More info: <http://docs.oracle.com/javase/tutorial/deployment/jar/intro.html>

Example MANIFEST.MF for a signed and sealed JAR.

Manifest-Version: 1.0

Sealed: true

Name: de/qaware/campus/secpro/er

SHA-256-Digest: rrkKKGa0bjV3Hq6e

Name: de/qaware/campus/secpro/er

SHA-256-Digest: iDsZpu6sIvcTi5L9

Name: META-INF/services/de.qawar

SHA-256-Digest: lyA9+HaJRTIKzn1v

Name: de/qaware/campus/secpro/er

SHA-256-Digest: FUBXb0usoxcn4fSf

Name: de/qaware/campus/secpro/er

SHA-256-Digest: lHXni+e9baryeHHr

```
console
Q:\codebase\02-sichere-programmierung\secpro-env01-j\build\libs>java -classpath
secpro-env01-j-signed.jar de.qaware.campus.secpro.env01.UnmodifiableLoginApp
Enter username:
Enter password:
Exception in thread "main" java.lang.SecurityException: SHA-256 digest error for
META-INF/services/de.qaware.campus.secpro.env01.Login
    at sun.security.util.ManifestEntryVerifier.verify(ManifestEntryVerifier.
java:220)
    at java.util.jar.JarVerifier.processEntry(JarVerifier.java:241)
    at java.util.jar.JarVerifier.update(JarVerifier.java:228)
    at java.util.jar.JarVerifier$VerifierStream.read(JarVerifier.java:483)
    at java.io.FilterInputStream.read(FilterInputStream.java:133)
    at sun.nio.cs.StreamDecoder.readBytes(StreamDecoder.java:283)
    at sun.nio.cs.StreamDecoder.implRead(StreamDecoder.java:325)
    at sun.nio.cs.StreamDecoder.read(StreamDecoder.java:177)
    at java.io.InputStreamReader.read(InputStreamReader.java:184)
    at java.io.BufferedReader.fill(BufferedReader.java:154)
    at java.io.BufferedReader.readLine(BufferedReader.java:317)
    at java.io.BufferedReader.readLine(BufferedReader.java:382)
    at java.util.ServiceLoader.parseLine(ServiceLoader.java:247)
    at java.util.ServiceLoader.parse(ServiceLoader.java:299)
    at java.util.ServiceLoader.access$200(ServiceLoader.java:181)
    at java.util.ServiceLoader$LazyIterator.hasNext(ServiceLoader.java:349)
    at java.util.ServiceLoader$LazyIterator.next(ServiceLoader.java:356)
    at java.util.ServiceLoader$1.next(ServiceLoader.java:445)
    at de.qaware.campus.secpro.env01.LoginFactory.getInstance(LoginFactory.j
ava:45)
    at de.qaware.campus.secpro.env01.UnmodifiableLoginApp.main(UnmodifiableL
oginApp.java:46)
Q:\codebase\02-sichere-programmierung\secpro-env01-j\build\libs>
```

Verify the signer certificate of a given class against a known and secured keystore.

```
public void verify(Class<?> clazz) throws SecurityException {
    Certificate[] certificates = clazz.getProtectionDomain().getCodeSource().getCertificates();
    if (certificates == null || certificates.length == 0) {
        LOGGER.info("No certificates found. Code source is not signed.");
        return;
    }

    try {
        KeyStore ks = KeyStore.getInstance("JKS");
        BufferedInputStream inputStream = new BufferedInputStream(clazz.getResourceAsStream("/keystore.jks"));

        ks.load(inputStream, securePasswords.decrypt(keystorePassword).toCharArray());
        boolean verified = certificates[0].equals(ks.getCertificate(keystoreAlias));
        if (!verified) {
            throw new SecurityException("The signer certificate could not be verified.");
        }
    } catch (KeyStoreException | CertificateException | NoSuchAlgorithmException | IOException e) {
        throw new SecurityException(e);
    }
}
```

MLR02-J. Obfuscate all security-sensitive code.

What's the problem?

Clean Code. Good programming style. Debugging symbols. Basically, everything that helps us developers is also helpful to the attacker.

How can you exploit the code?

Simply by disassembling the relevant code, using tools like *javap*, *JAD*, *dirtyJOE*.

How can we do better?

Obfuscate the security sensitive code with tools like *ProGuard*, *yGuard*, et.al.

Obfuscation leads to reduced readability, cryptic variable names, inlining of method calls, misleading branches.

```
public static void main(String[] var0) {  
    Console var4;  
    String var1 = (var4 = System.console()).readLine("Enter username: ", new Object[0]);  
    char[] var2 = var4.readPassword("Enter password: ", new Object[0]);  
    new b();  
    new a();  
    boolean var10000;  
    switch(var1.concat(":".concat(new String(var2)).hashCode())) {  
    case -660170103:  
        var10000 = true;  
        break;  
    default:  
        var10000 = false;  
    }  
  
    if(!var10000) {  
        var4.printf("Authentication failure. Exit.", new Object[0]);  
        System.exit(1);  
    }  
}
```

Only up to 10% of the bytecode
instructions in modern JEE
applications are your code!!!

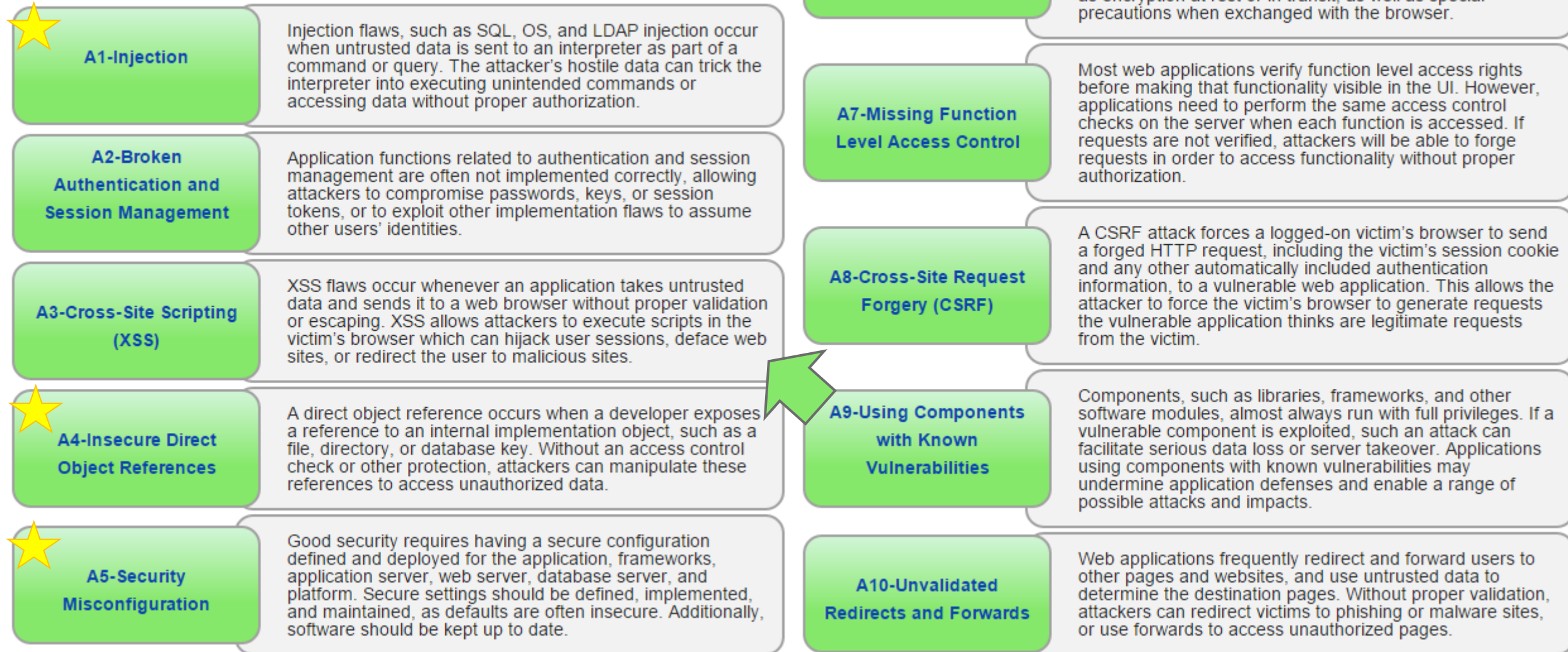
At least 90% of your application
pose a potential security risk!

About 26% of the downloaded
libraries on Maven Central
contain known vulnerabilities!

https://www.owasp.org/index.php/OWASP_AppSec_DC_2012/The_Unfortunate_Reality_of_Insecure_Libraries

OWASP Top 10 2013

A9 should be in the Top 3.



Know your dependencies. The secure usage of open source components and frameworks is key to application security.

- But how do I secure my application against security issues in open source software?
 - ⊖ ~~Option a) Do not use open source software. Write everything yourself! → **Not very realistic!**~~
 - *Option b)* Have clear guidelines and rules for the responsible usage of open source software.
- Upgrading your dependencies to the latest versions is crucial. Urgent security fixes are usually only applied to the latest release.
- Monitor security issues of used frameworks in public databases (CVE, NVD) and mailing lists.
- Implement **security decorators** to disable or secure weak and unused framework functionality.

mvn versions:display-dependency-updates

[INFO] The following dependencies in Dependencies have newer versions:

```
[INFO] com.sun.faces:jsf-api ..... 2.1.10 -> 2.2.12
[INFO] com.sun.jersey:jersey-client ..... 1.9.1 -> 1.19
[INFO] commons-fileupload:commons-fileupload ..... 1.2.1 -> 1.3.1
[INFO] org.apache.httpcomponents:httpclient ..... 4.2.1 -> 4.5.1
[INFO] org.apache.solr:solr-core ..... 4.6.1 -> 5.3.1
```

```
<plugin>
  <groupId>org.codehaus.mojo</groupId>
  <artifactId>versions-maven-plugin</artifactId>
  <version>2.2</version>

  <!-- generate reports when running mvn site -->
  <reportSets>
    <reportSet>
      <reports>
        <report>dependency-updates-report</report>
        <report>plugin-updates-report</report>
      </reports>
    </reportSet>
  </reportSets>
</plugin>
```

secpro-owasp-check

secpro-owasp-check

Last Published: 2015-10-21 | Version: 3.0.0-SNAPSHOT

Project Documentation
Project Information
Project Reports
Dependency-check
Dependency Updates Report
Plugin Update Report

Overview

This report summarizes newer versions that may be available for your project's various dependencies.

🚩	# of dependencies using the latest version available	0
🚩	# of dependencies where the next version available is smaller than an incremental version update	0
🚩	# of dependencies where the next version available is an incremental version update	4
🚩	# of dependencies where the next version available is a minor version update	1
🚩	# of dependencies where the next version available is a major version update	0

Dependency Management

This project does not declare any dependencies in a dependencyManagement section.

Dependencies

Status	Group Id	Artifact Id	Current Version	Scope	Classifier	Type	Next Version	Next Incremental	Next Minor	Next Major
🚩	com.sun.faces	jsf-api	2.1.10	runtime		jar	2.1.11	2.2.0		
🚩	com.sun.jersey	jersey-client	1.9.1	runtime		jar	1.10-b01	1.10		
🚩	commons-fileupload	commons-fileupload	1.2.1	runtime		jar	1.2.2	1.3		
🚩	org.apache.httpcomponents	httpClient	4.2.1	runtime		jar	4.2.2	4.3		
🚩	org.apache.solr	solr-core	4.6.1	runtime		jar	4.7.0	5.0.0		

Dependency Updates

com.sun.faces:jsf-api

Status	🚩 There is at least one newer incremental version available. Incremental updates are typically passive.
Group Id	com.sun.faces
Artifact Id	jsf-api
Current Version	2.1.10
Scope	runtime
Classifier	
Type	jar
Newer versions	2.1.11 Next Incremental 2.1.12 2.1.13 2.1.14 2.1.15 2.1.16 2.1.17



mvn org.owasp:dependency-check-maven:check

DEPENDENCY-CHECK

Dependency-Check is an open source tool performing a best effort analysis of 3rd party dependencies; false positives and false negatives may exist in the analysis performed by the tool. Use of the tool and the reporting provided constitutes acceptance for use in an AS IS condition, and there are NO warranties, implied or otherwise, with regard to the analysis or its use. Any use of the tool and the reporting provided is at the user's risk. In no event shall the copyright holder or OWASP be held liable for any damages whatsoever arising out of or in connection with the use of this tool, the analysis performed, or the resulting report.

Project: secpro-owasp-check

Scan Information ([show all](#)):

- *dependency-check version:* 1.3.1
- *Report Generated On:* Okt 21, 2015 at 01:06:13 MESZ
- *Dependencies Scanned:* 49
- *Vulnerable Dependencies:* 6
- *Vulnerabilities Found:* 8
- *Vulnerabilities Suppressed:* 0
- ...

- 49 scanned dependencies
- 6 vulnerable dependencies
- 8 found vulnerabilities

Display: [Showing Vulnerable Dependencies \(click to show all\)](#)

Dependency	CPE	GAV	Highest Severity	CVE Count	CPE Confidence	Evidence Count
commons-fileupload-1.2.1.jar	cpe:/a:apache:commons_fileupload:1.2.1	commons-fileupload:commons-fileupload:1.2.1	Medium	2	HIGHEST	26
hadoop-auth-2.2.0.jar	cpe:/a:apache:hadoop:2.2.0	org.apache.hadoop:hadoop-auth:2.2.0	Medium	1	HIGHEST	17
httpclient-4.2.1.jar	cpe:/a:apache:httpclient:4.2.1	org.apache.httpcomponents:httpclient:4.2.1	Medium	1	HIGHEST	25
solr-core-4.6.1.jar	cpe:/a:apache:solr:4.6.1	org.apache.solr:solr-core:4.6.1	Medium	1	HIGHEST	23
zookeeper-3.4.5.jar	cpe:/a:apache:zookeeper:3.4.5	org.apache.zookeeper:zookeeper:3.4.5	Low	1	LOW	20
org.restlet-2.1.1.jar	cpe:/a:restlet:restlet:2.1.1	org.restlet.jee:org.restlet:2.1.1	High	2	HIGHEST	13



mvn org.owasp:dependency-check-maven:check

DEPENDENCY-CHECK

Dependency-Check is an open source tool performing a best effort analysis of the dependencies of your project. It is not a scanner, it does not scan your code. It is a tool, it does not have a mind, it does not have a conscience. It is a tool, it does not have a conscience. It is a tool, it does not have a conscience.

Project: secpro-owasp-che

Scan Information (show all):

- dependency-check version: 1.3.1
- Report Generated On: Okt 21, 2014
- Dependencies Scanned: 49
- Vulnerable Dependencies: 6
- Vulnerabilities Found: 8
- Vulnerabilities Suppressed: 0
- ...

Display: Showing Vulnerable Depend

Dependency

commons-fileupload-1.2.1.jar cpe:/a:apache:commons-fileupload:1.2.1

hadoop-auth-2.2.0.jar cpe:/a:apache:hadoop-auth:2.2.0

httpclient-4.2.1.jar cpe:/a:apache:httpclient:4.2.1

solr-core-4.6.1.jar cpe:/a:apache:solr:4.6.1

zookeeper-3.4.5.jar cpe:/a:apache:zookeeper:3.4.5

org.restlet-2.1.1.jar cpe:/a:restlet:restlet:2.1.1

Published Vulnerabilities

[CVE-2014-0050](#)

Severity: Medium

CVSS Score: 5.0 (AV:N/AC:L/Au:N/C:N/I:N/A:P)

CWE: CWE-264 Permissions, Privileges, and Access Controls

MultipartStream.java in Apache Commons FileUpload before 1.3.1, as used in Apache Tomcat, JBoss Web, and other products, allows remote attackers to cause a denial of service (infinite loop and CPU consumption) via a crafted Content-Type header that bypasses a loop's intended exit conditions.

- BID - [65400](#)
- BUGTRAQ - [20140625 NEW VMSA-2014-0007 - VMware product updates address security vulnerabilities in Apache Struts library](#)
- BUGTRAQ - [20141205 NEW: VMSA-2014-0012 - VMware vSphere product updates address security vulnerabilities](#)
- BUGTRAQ - [20150402 NEW: VMSA-2015-0003 VMware product updates address critical information disclosure issue in JRE](#)
- CONFIRM - <http://advisories.mageia.org/MGASA-2014-0110.html>
- CONFIRM - <http://svn.apache.org/r1565143>
- CONFIRM - <http://tomcat.apache.org/security-7.html>
- CONFIRM - <http://tomcat.apache.org/security-8.html>
- CONFIRM - <http://www-01.ibm.com/support/docview.wss?uid=swg21669554>
- CONFIRM - <http://www-01.ibm.com/support/docview.wss?uid=swg21675432>
- CONFIRM - <http://www-01.ibm.com/support/docview.wss?uid=swg21676091>
- CONFIRM - <http://www-01.ibm.com/support/docview.wss?uid=swg21676092>
- CONFIRM - <http://www-01.ibm.com/support/docview.wss?uid=swg21676401>
- CONFIRM - <http://www-01.ibm.com/support/docview.wss?uid=swg21676403>
- CONFIRM - <http://www-01.ibm.com/support/docview.wss?uid=swg21676405>
- CONFIRM - <http://www-01.ibm.com/support/docview.wss?uid=swg21676410>
- CONFIRM - <http://www-01.ibm.com/support/docview.wss?uid=swg21676656>
- CONFIRM - <http://www-01.ibm.com/support/docview.wss?uid=swg21676853>

org.apache.httpcomponents:httpclient:4.2.1 Medium 1 HIGHEST 25

org.apache.solr:solr-core:4.6.1 Medium 1 HIGHEST 23

org.apache.zookeeper:zookeeper:3.4.5 Low 1 LOW 20

org.restlet.jee:org.restlet:2.1.1 High 2 HIGHEST 13



mvn org.owasp:dependency-check-maven:check

DEPENDENCY-CHECK

Dependency-Check is an open source tool performing a best effort analysis of the dependencies of your project. It is not a scanner, it does not scan your code. It is a tool, it does not have any opinions, it only reports what it finds. It is not a security tool, it only reports what it finds.

Project: secpro-owasp-che

Scan Information (show all):

- dependency-check version: 1.3.1
- Report Generated On: Okt 21, 2017
- Dependencies Scanned: 49
- Vulnerable Dependencies: 6
- Vulnerabilities Found: 8
- Vulnerabilities Suppressed: 0
- ...

Display: Showing Vulnerable Depend

Dependency

commons-fileupload-1.2.1.jar

hadoop-auth-2.2.0.jar

httpclient-4.2.1.jar

solr-core-4.6.1.jar

zookeeper-3.4.5.jar

org.restlet-2.1.1.jar

Published Vulnerabilities

[CVE-2014-3577](#)

Severity: Medium

CVSS Score: 5.8 (AV:N/AC:M/Au:N/C:P/I:P/A:N)

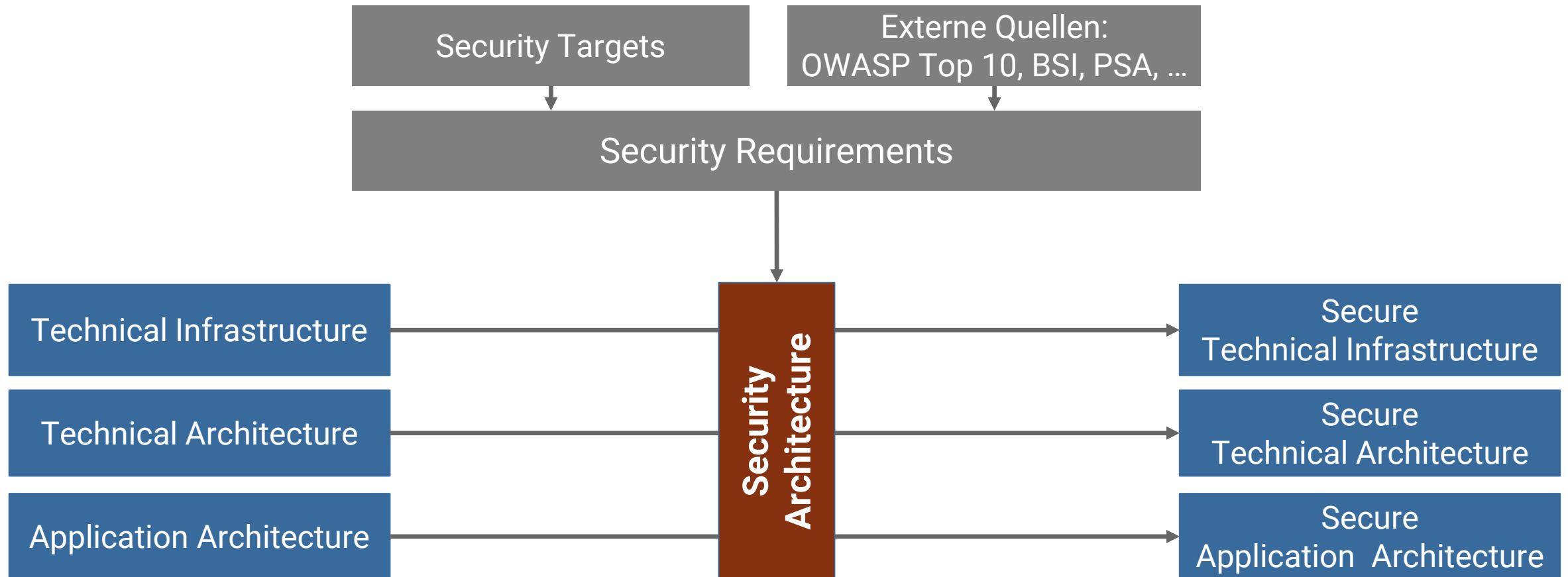
org.apache.http.conn.ssl.AbstractVerifier in Apache HttpComponents HttpClient before 4.3.5 and HttpAsyncClient before 4.0.2 does not properly verify that the server hostname matches a domain name in the subject's Common Name (CN) or subjectAltName field of the X.509 certificate, which allows man-in-the-middle attackers to spoof SSL servers via a "CN=" string in a field in the distinguished name (DN) of a certificate, as demonstrated by the "foo,CN=www.apache.org" string in the O field.

- CONFIRM - <https://access.redhat.com/solutions/1165533>
- FULLDISC - [20140818 CVE-2014-3577: Apache HttpComponents client: Hostname verification susceptible to MITM attack](#)
- MISC - <http://packetstormsecurity.com/files/127913/Apache-HttpComponents-Man-In-The-Middle.html>
- REDHAT - [RHSA-2014:1146](#)
- REDHAT - [RHSA-2014:1166](#)
- REDHAT - [RHSA-2014:1833](#)
- REDHAT - [RHSA-2014:1834](#)
- REDHAT - [RHSA-2014:1835](#)
- REDHAT - [RHSA-2014:1836](#)
- REDHAT - [RHSA-2014:1891](#)
- REDHAT - [RHSA-2014:1892](#)
- REDHAT - [RHSA-2015:0125](#)
- REDHAT - [RHSA-2015:0158](#)
- REDHAT - [RHSA-2015:0675](#)
- REDHAT - [RHSA-2015:0720](#)
- REDHAT - [RHSA-2015:0765](#)
- REDHAT - [RHSA-2015:0850](#)

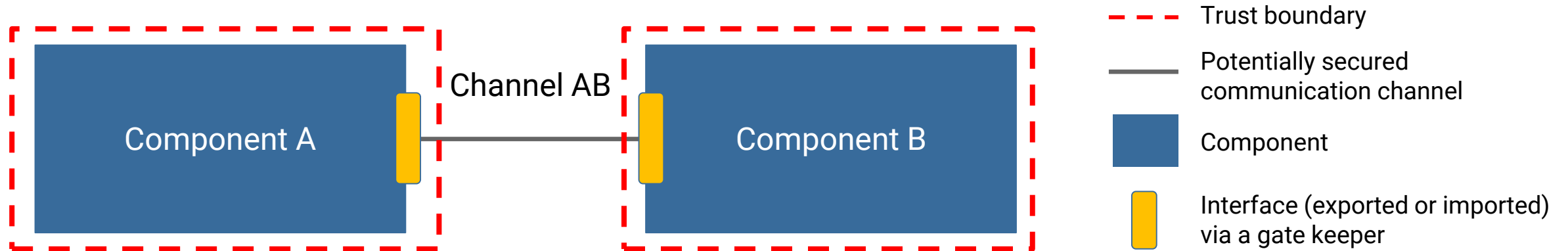
httpclient-4.2.1.jar	cpe:/a:apache:httpclient:4.2.1	org.apache.httpcomponents:httpclient:4.2.1	Medium	1	HIGHEST	25
solr-core-4.6.1.jar	cpe:/a:apache:solr:4.6.1	org.apache.solr:solr-core:4.6.1	Medium	1	HIGHEST	23
zookeeper-3.4.5.jar	cpe:/a:apache:zookeeper:3.4.5	org.apache.zookeeper:zookeeper:3.4.5	Low	1	LOW	20
org.restlet-2.1.1.jar	cpe:/a:restlet:restlet:2.1.1	org.restlet.jee:org.restlet:2.1.1	High	2	HIGHEST	13

Perform the OWASP dependency
check in a dedicated security
build in your CI environment.

The security architecture of a systems describes how the normal architecture is secured at different levels.

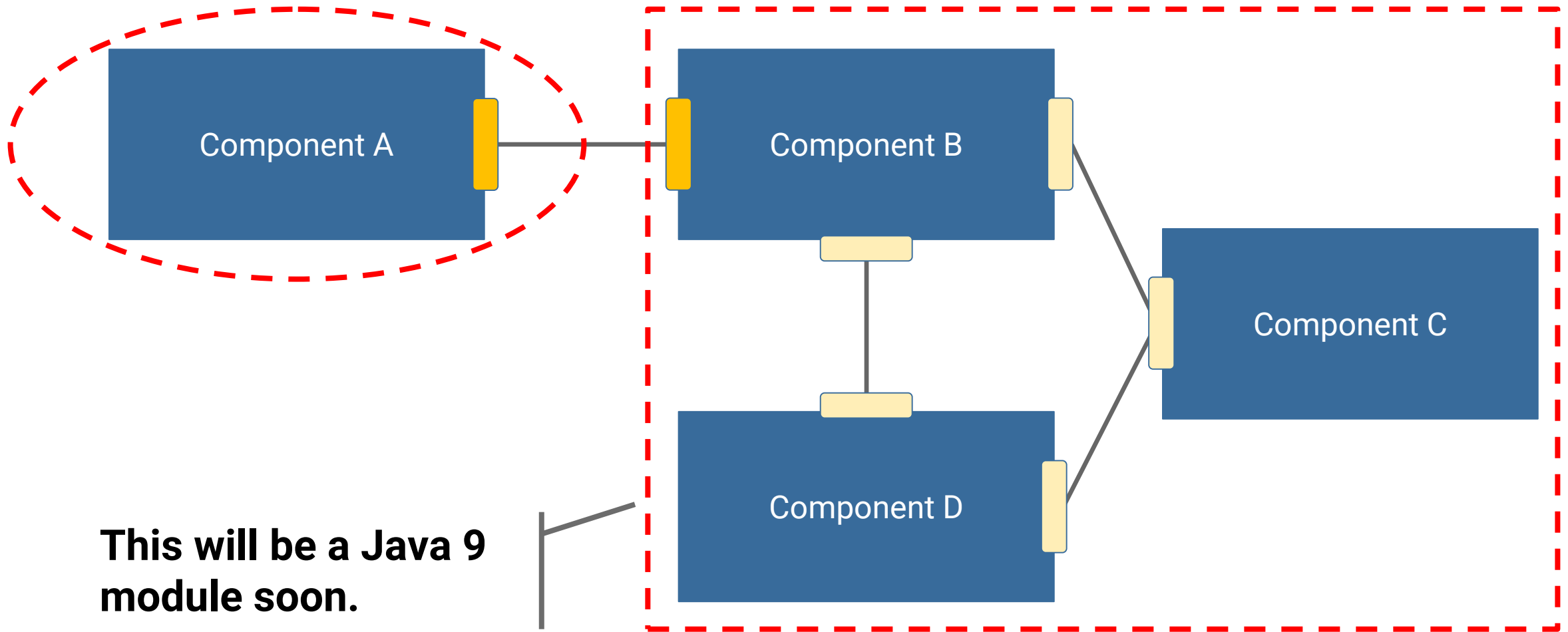


The security architecture consists of security components and communication channels that may need to be secured.

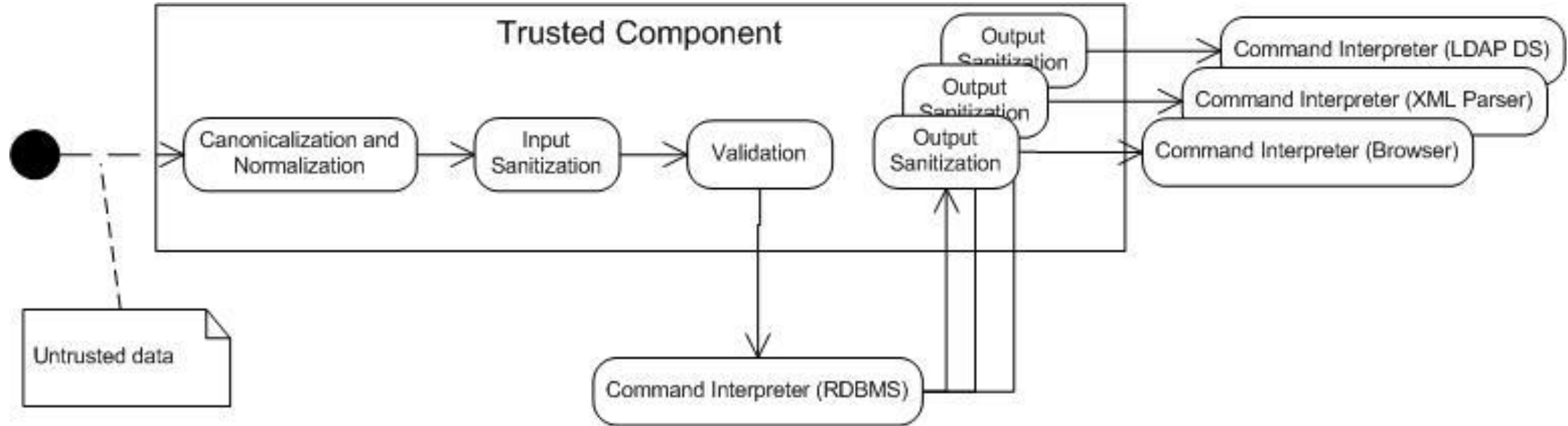


- Each system consists of security components that are connected by channels
 - Different abstractions: data centers, hardware units, VMs, app servers, databases, *software components*, ...
 - Each security component is owned by somebody. This somebody may be trust worthy or not.
- Each security component has a defined security - from very secure to insecure:
 - How exhaustive and elaborate must the gate keeper be at the entries and exits? Fort Knox or access to everyone?
- Each channel has a defined security – from very secure to insecure:
 - How robust is a channel and the used protocol against typical attacks?

Security components can form security communities, with hard boarder controls and loose inner security.



The internal design of secure components is influenced by security concerns. But the business logic should stay clean.



○ Canonicalization

- Lossless reduction to the most simple representation.

○ Normalization

- Lossy reduction to the most simple representation

○ Sanitization

- Ensure data hygiene
- Prevent information disclosure and leakage

○ Validation

- Expected types and value ranges
- Validate if input satisfies the expected patterns

Security is a cross cutting concern. Interceptors are a perfect match to implement security functionality.

```
@Interceptor  
@Sanitized
```

Interceptor + Binding annotations

```
public class SanitizedInterceptor {
```

```
@AroundInvoke
```

```
public Object invoke(InvocationContext ctx) throws Exception {  
    Sanitized sanitization = getSanitizedAnnotation(ctx.getMethod());  
  
    // apply the sanitization function  
    Object[] sanitized = Arrays.stream(ctx.getParameters()).map(sanitization.type()).toArray();  
    ctx.setParameters(sanitized);  
  
    return ctx.proceed();  
}
```

Sanitize parameters and continue

```
private Sanitized getSanitizedAnnotation(Method m) {...}  
}
```

Get annotation from method
or it's declaring class

```
<interceptors>
```

Activate in beans.xml

```
    <class>de.qaware-campus.secpro.web.security.SanitizedInterceptor</class>  
</interceptors>
```

The interceptor binding annotation defines relevant types and their sanitization functions.

```
@Retention(RetentionPolicy.RUNTIME)
@Target({TYPE, METHOD})
@InterceptorBinding
public @interface Sanitized {
```

Interceptor binding annotation can be applied to methods and classes

```
/**
 * Specific types with their sanitization functions.
 */
```

```
enum Type implements Function<Object, Object> {
```

```
    ECMA_SCRIPT {
```

```
        @Override
```

```
        public Object apply(Object o) {
```

```
            if (o instanceof String) {
```

```
                return StringEscapeUtils.ESCAPE_JAVASCRIPT.translate(o.toString());
```

```
            }
```

```
            return o;
```

```
        }
```

```
    }, SQL {...}
```

```
}
```

The sanitization function

```
/**
```

```
 * Pass additional information to the interceptor.
```

```
 */
```

```
@Nonbinding Type type() default Type.ECMA_SCRIPT;
```

```
}
```

Non-binding sanitization type value

Use CDI decorators for component specific security features.

```
@Decorator
public class NoGreetingToAttackersDecorator implements Greeting {

    @Inject
    @Delegate
    private Greeting greeter;

    @Override
    public String getMessage(@Size(min = 3) String name) {
        if ("attacker".equalsIgnoreCase(name)) {
            throw new SecurityException("No greeting for evil attackers.");
        }

        // do some additional specific security checks
        // maybe use a javax.validation.Validator for this

        // continue and delegate
        return greeter.getMessage(name);
    }
}
```

Inject the delegate instance

Do any additional security check that may be required

```
<decorators>
    <class>de.qaware.campus.secpro.web.hello.NoGreetingToAttackersDecorator</class>
</decorators>
```

Activate in beans.xml

Apply Design by Contract (DbC) to your gate keeper and security components using the method validation API.

```
public interface Greeting {  
    /**  
     * Returns a Greeting for the given name. The actual greeting  
     * may depend on implementation.  
     *  
     * @param name the name, at least 3 characters  
     * @return the greeting message, never null  
     */  
    @NotNull  
    String getMessage(@Size(min = 3) String name);  
}
```

The interface is the contract.

It defines the pre and post conditions of methods using javax.validation annotations.

```
@ApplicationScoped  
public class DefaultGreeting implements Greeting {  
    @Override  
    @NotNull  
    public String getMessage(@Size(min = 3) String name) {  
        return format("Hello %s!", name);  
    }  
}
```

There is no 100% security.

It's up to us developers to
write secure applications!

Incorporate security into your
daily development process.

Pay your employees well! Cater
for a good work environment.



Mario-Leander Reimer

Chief Technologist, QAware GmbH

mario-leander.reimer@qaware.de

<https://slideshare.net/MarioLeanderReimer/>

<https://speakerdeck.com/lreimer/>

<https://github.com/lreimer/secure-programming-101/>

<https://twitter.com/leanderreimer/>



SOFTWARE ENGINEERING