TRAINING KIT – CODE1
Secure Coding

EXCELLIUM

Your first call when it comes to IT and Security!

TLP: WHITE

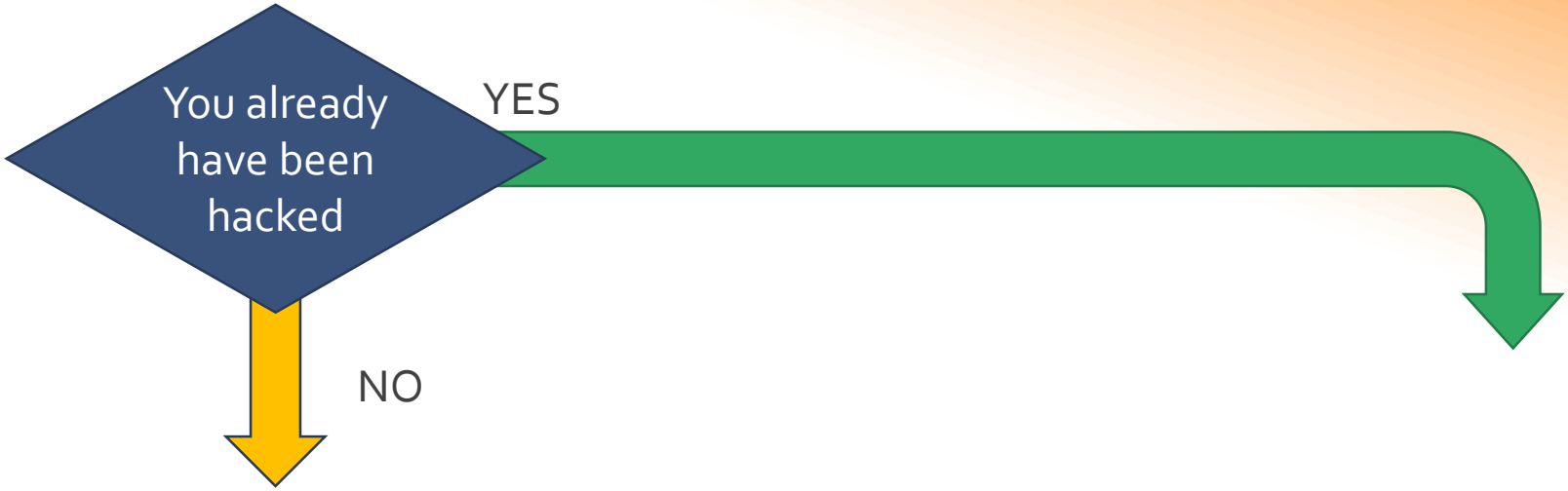
March 16, 2017

Agenda

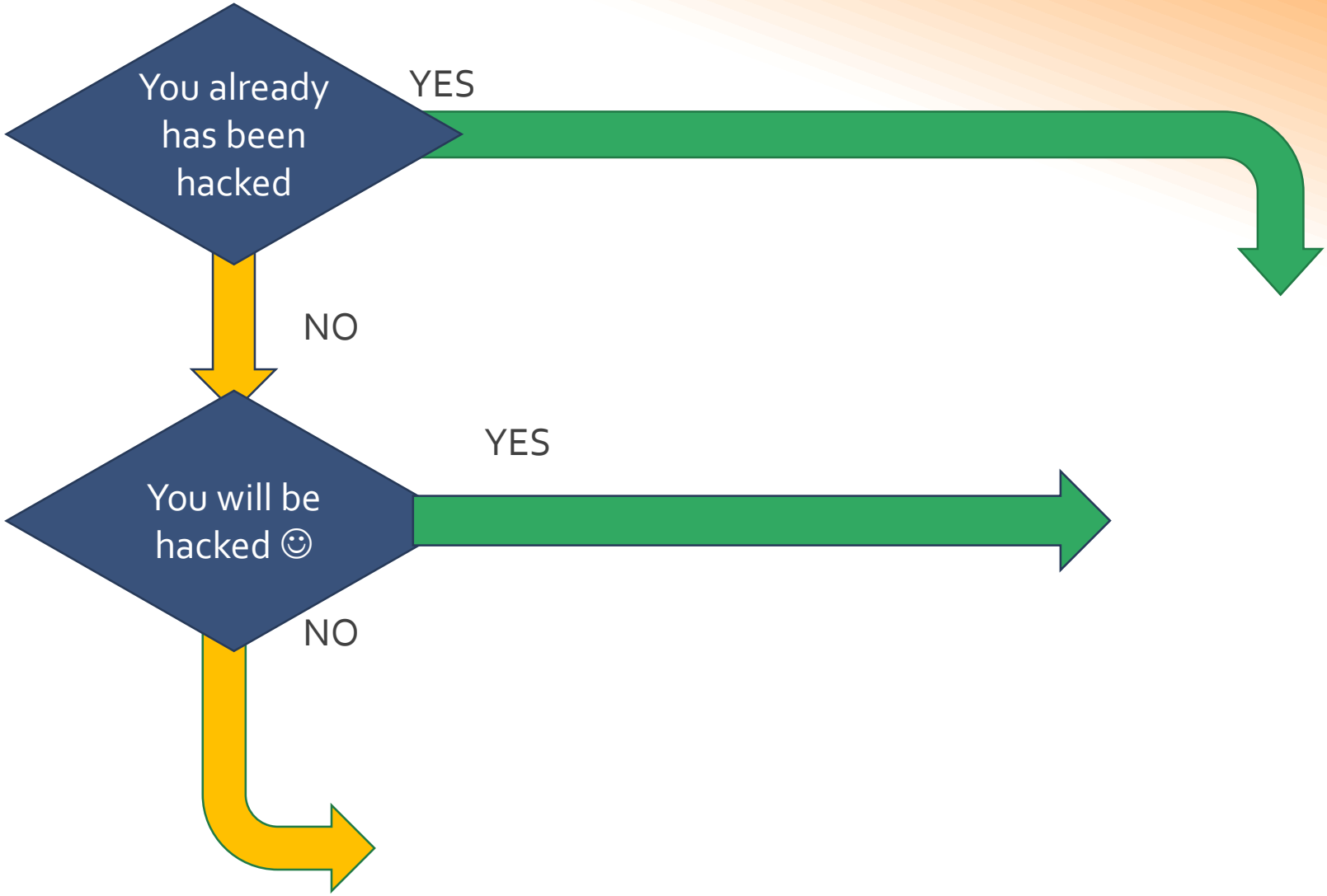
- Introduction
  - Application Security Overview
  - How to improve?
- Application security
  - Common Attacks
  - Common Defenses
- Secure Coding principes
- Interesting links
- Q&A

Introduction

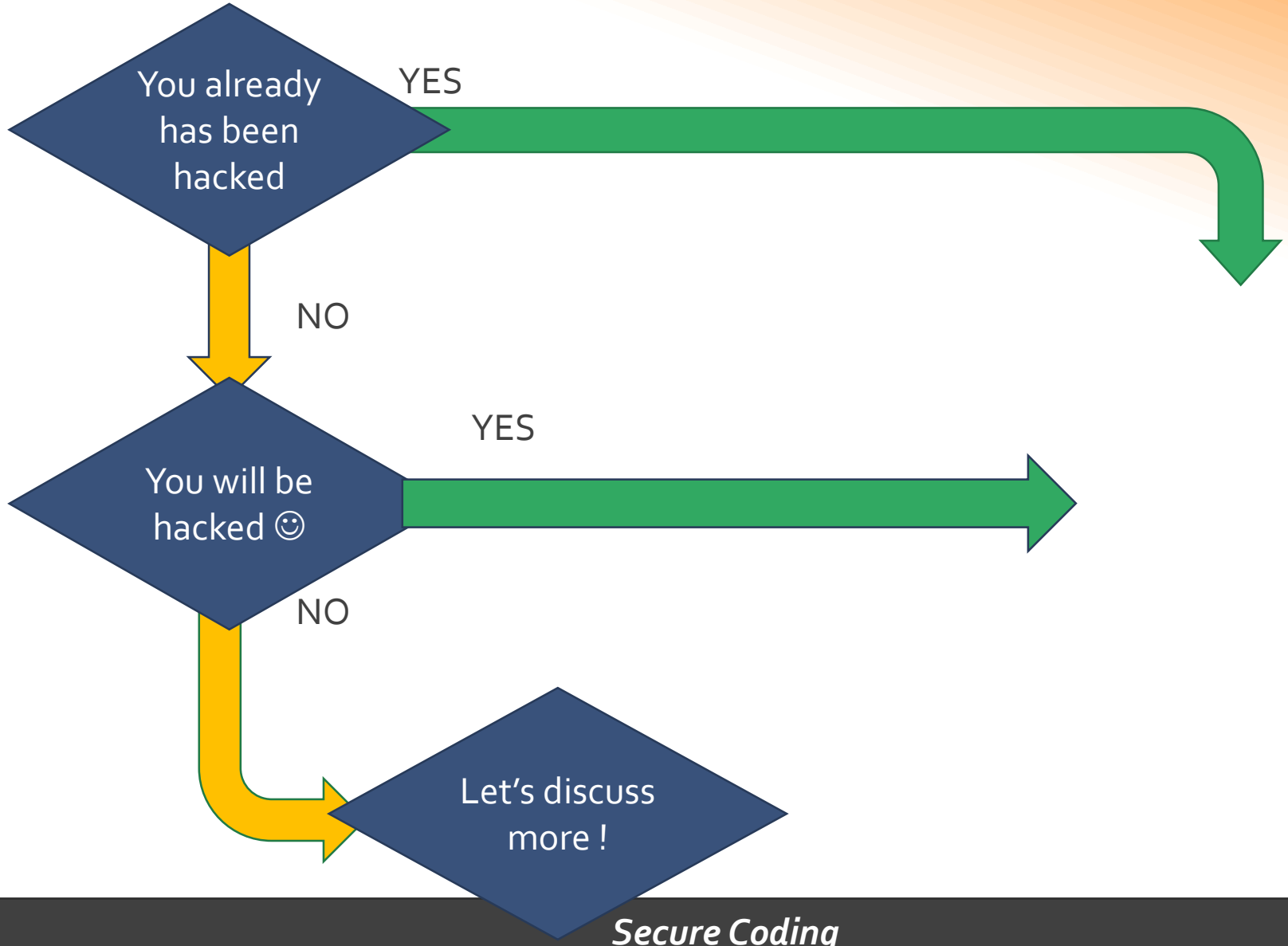
Application Security Overview



Application Security Overview



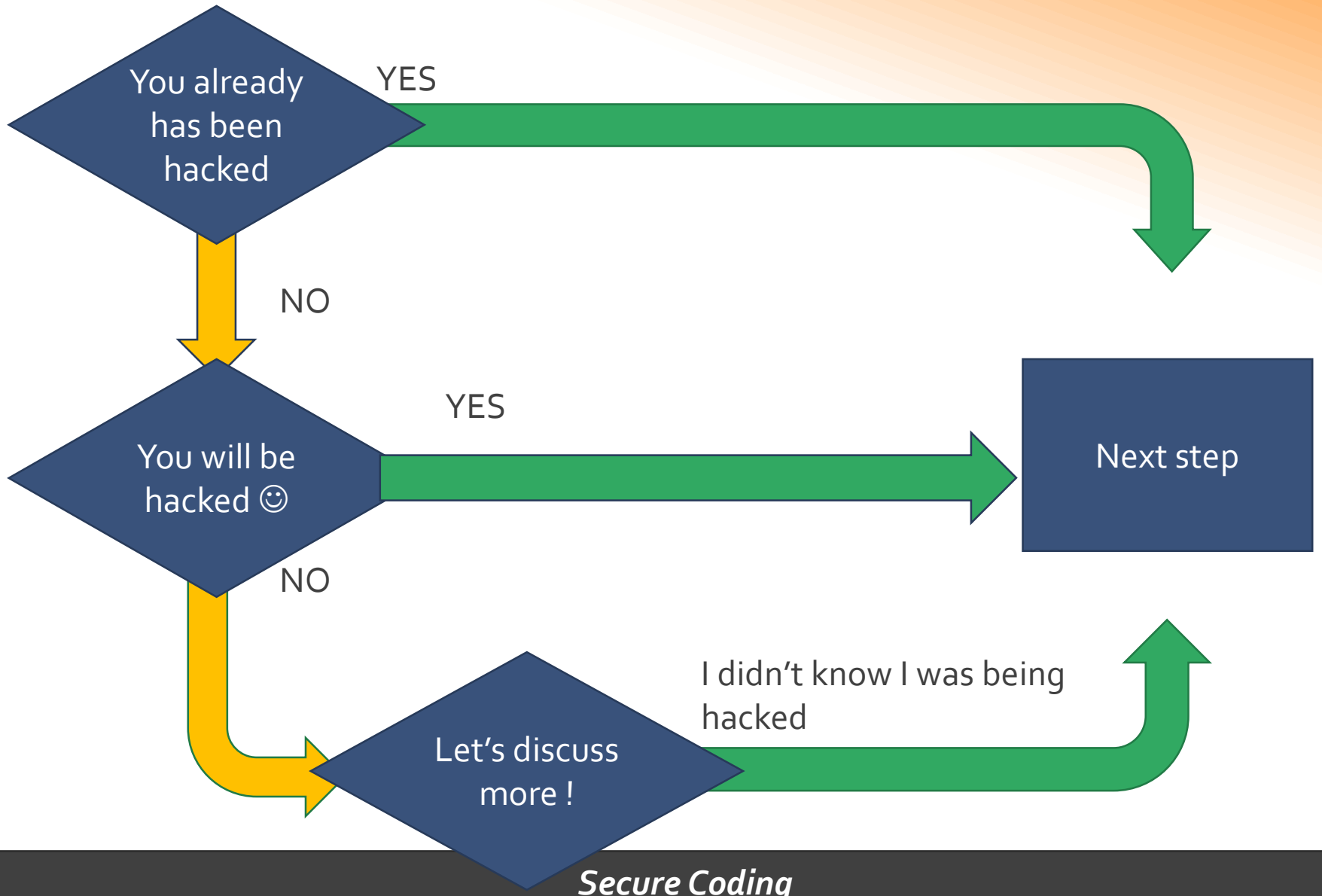
Application Security Overview



Application Security Overview

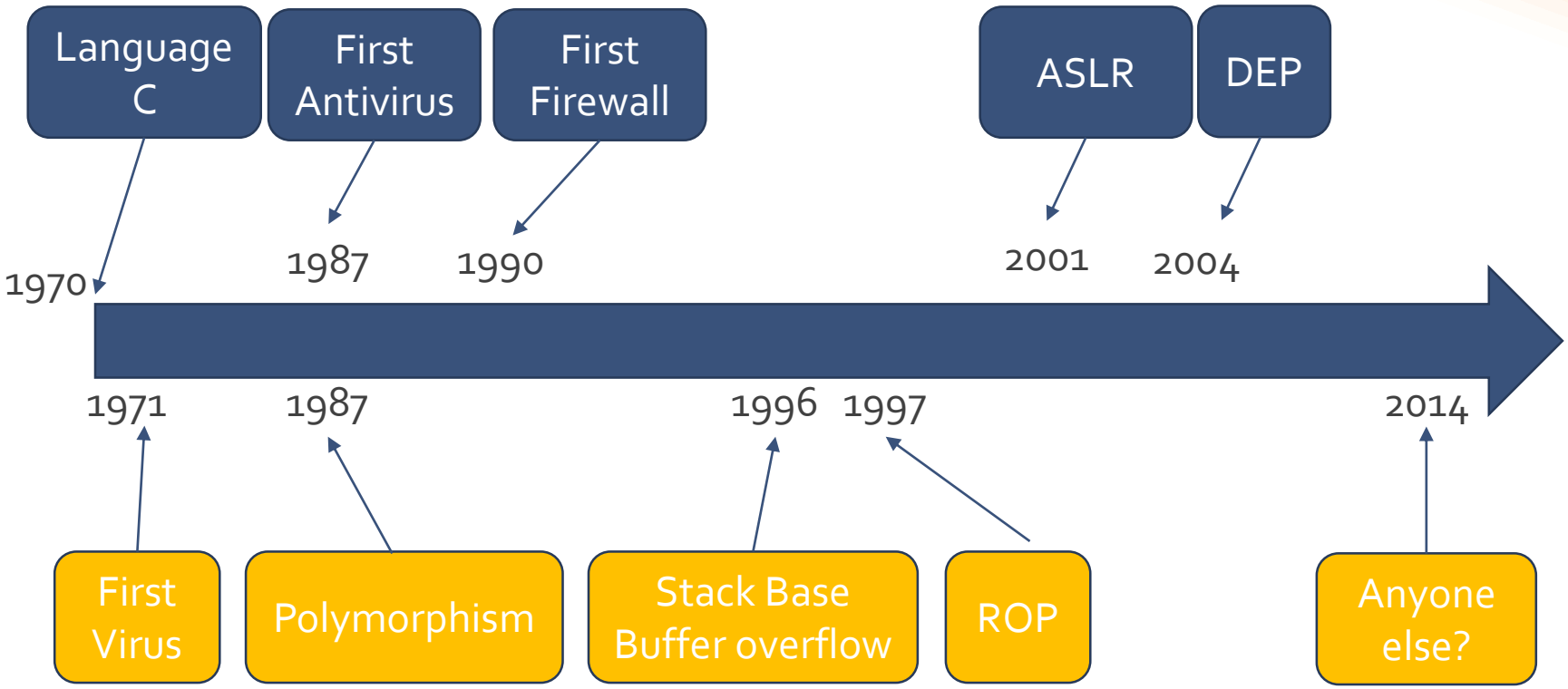
- Who uses prepared statement in SQL? Even for custom search requests?
- Who had to use real crypto in a project? Was it simple?
- Who runs the application server as administrator?
- Who uses a CUSTOM security manager?
- Who uses maven? Who checks for vulnerabilities in installed artefacts?
- Who only checks data coming from user or third-party system for validity from a business point of view?

Application Security Overview



Application Security Overview

Example: Operating System issues



Application Security Overview

- More than 10 years (1971→1987) to find a (weak) countermeasure to viruses
- More than 8 years (1996→2004) to fix the buffer overflow issues.

- But now…
  - The attackers are not working anymore on operating systems, they are on the application level (web & mobile).

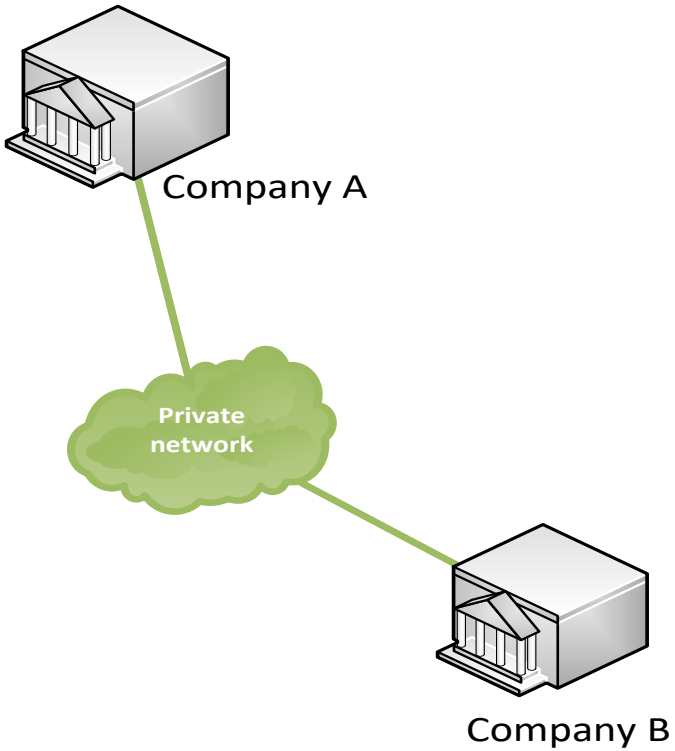
Application Security Overview

And…

- Most of the web sites are vulnerable to attacks
- A lot of components are now using the web (Webmails, online store, eBanking…)
- Mobile application are often:
  - Web mobile application (application on device is just a wrapper)
  - Native mobile application consuming web services exposed by a web backend
- There is no simple way to control what an App is doing on a mobile device.

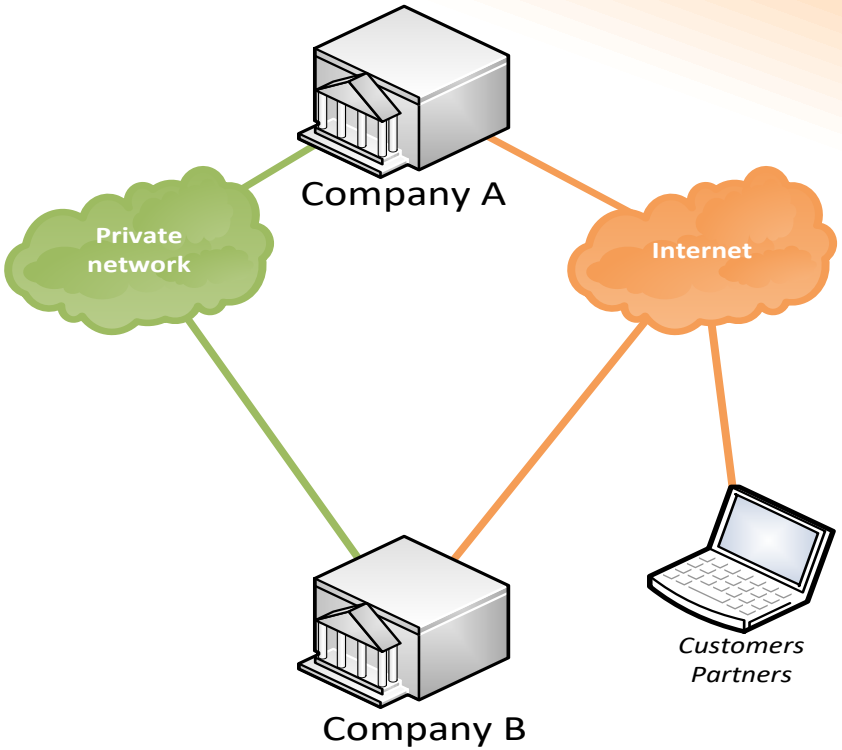
Application Security Overview

One thing that has changed with web applications on the Internet



« In the past »

- **Communication only between companies,**
- **Company information system <u>accessed by is employees</u>.**



Today

- **Communication between companies, customers, partners,**
- **Company information system accessed <u>by employees, partners and customers</u>.**

Application Security Overview

Consequences of a bad security

- Identity theft
- IT downtime
- SLA issues
- Financial loss
- Reputation in media

Application Security Overview

From the Verizon DataBreach 2016 report

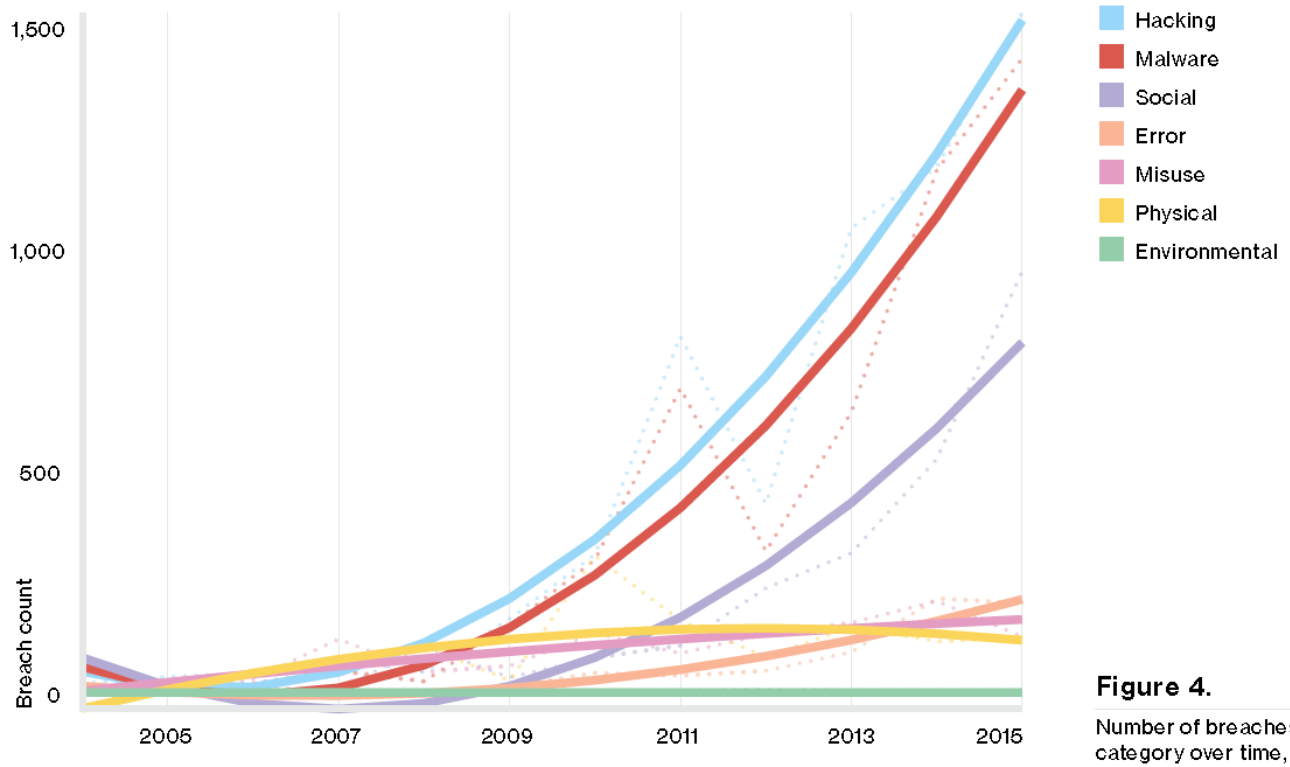


Figure 4.

Number of breaches per threat action category over time, (n=9,009)

Application Security Overview

From the Verizon DataBreach 2016 report

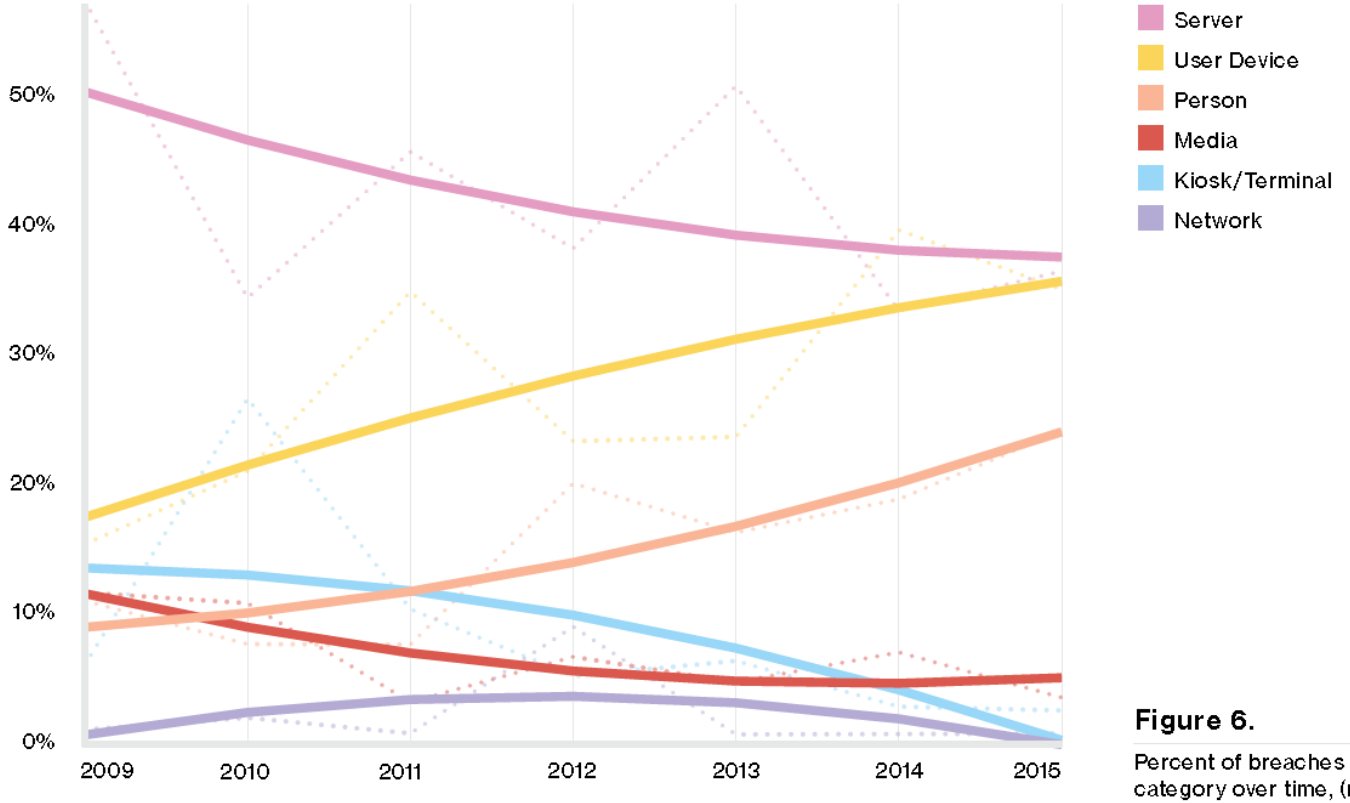


Figure 6.

Percent of breaches per asset category over time, (n=7,736)

Application Security Overview

From the Verizon DataBreach 2016 report

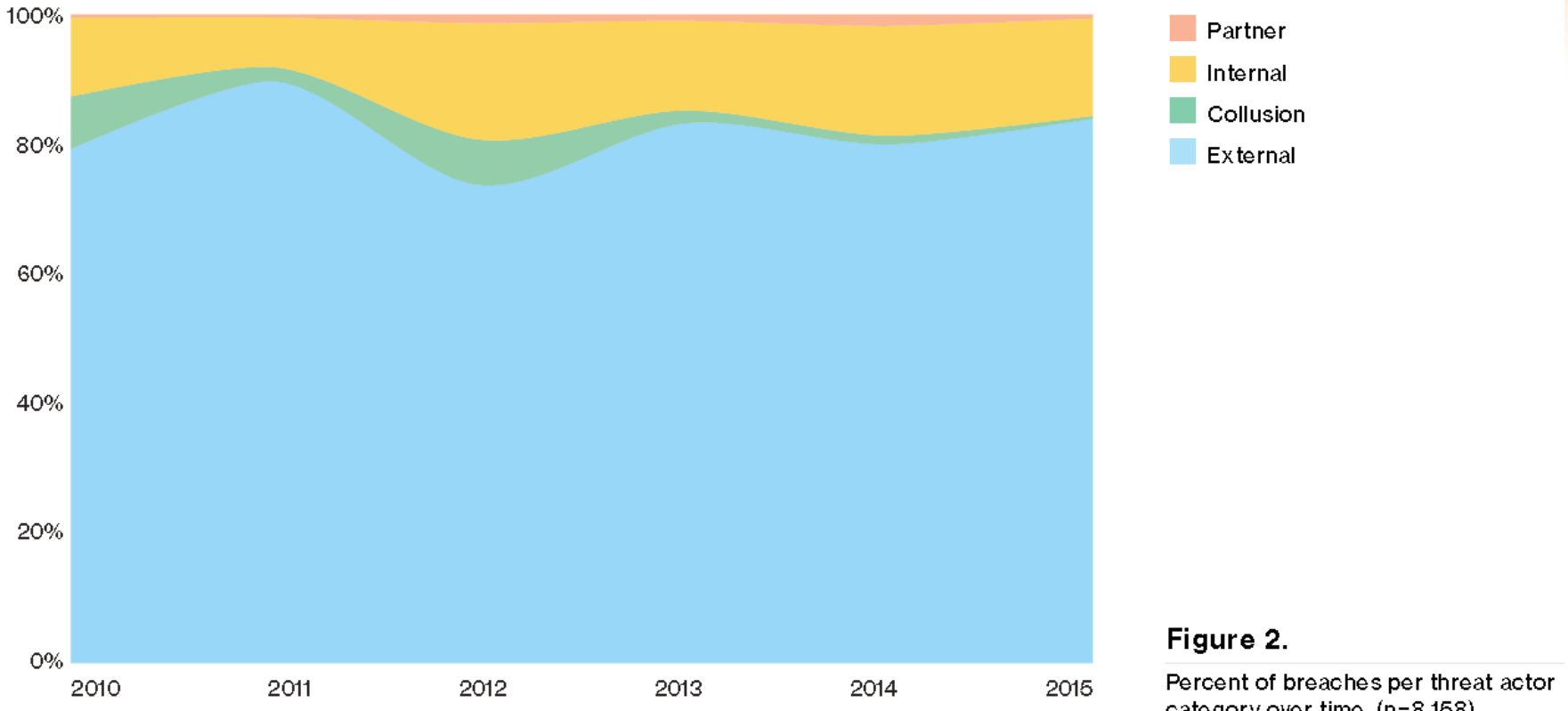


Figure 2.

Percent of breaches per threat actor category over time, (n=8,158)

Application Security Overview

Can the infrastructure protect your application?

- With a firewall?
- With an IPS?
- With a web application firewall?
- From your own users?
- Your internal applications?

Application Security Overview

Can the infrastructure protect your application?

- With a firewall? **NO**
- With an IPS?
- With a web application firewall?
- From your own users?
- Your internal applications?

Application Security Overview

Can the infrastructure protect your application?

- With a firewall? **NO**
- With an IPS? **NO**
- With a web application firewall?
- From your own users?
- Your internal applications?

Application Security Overview

Can the infrastructure protect your application?

- With a firewall? **NO**
- With an IPS? **NO**
- With a web application firewall? It depends
- From your own users?
- Your internal applications?

Application Security Overview

Can the infrastructure protect your application?

- With a firewall? **NO**
- With an IPS? **NO**
- With a web application firewall? It depends
- From your own users? **NO**
- Your internal applications?

Application Security Overview

Can the infrastructure protect your application?

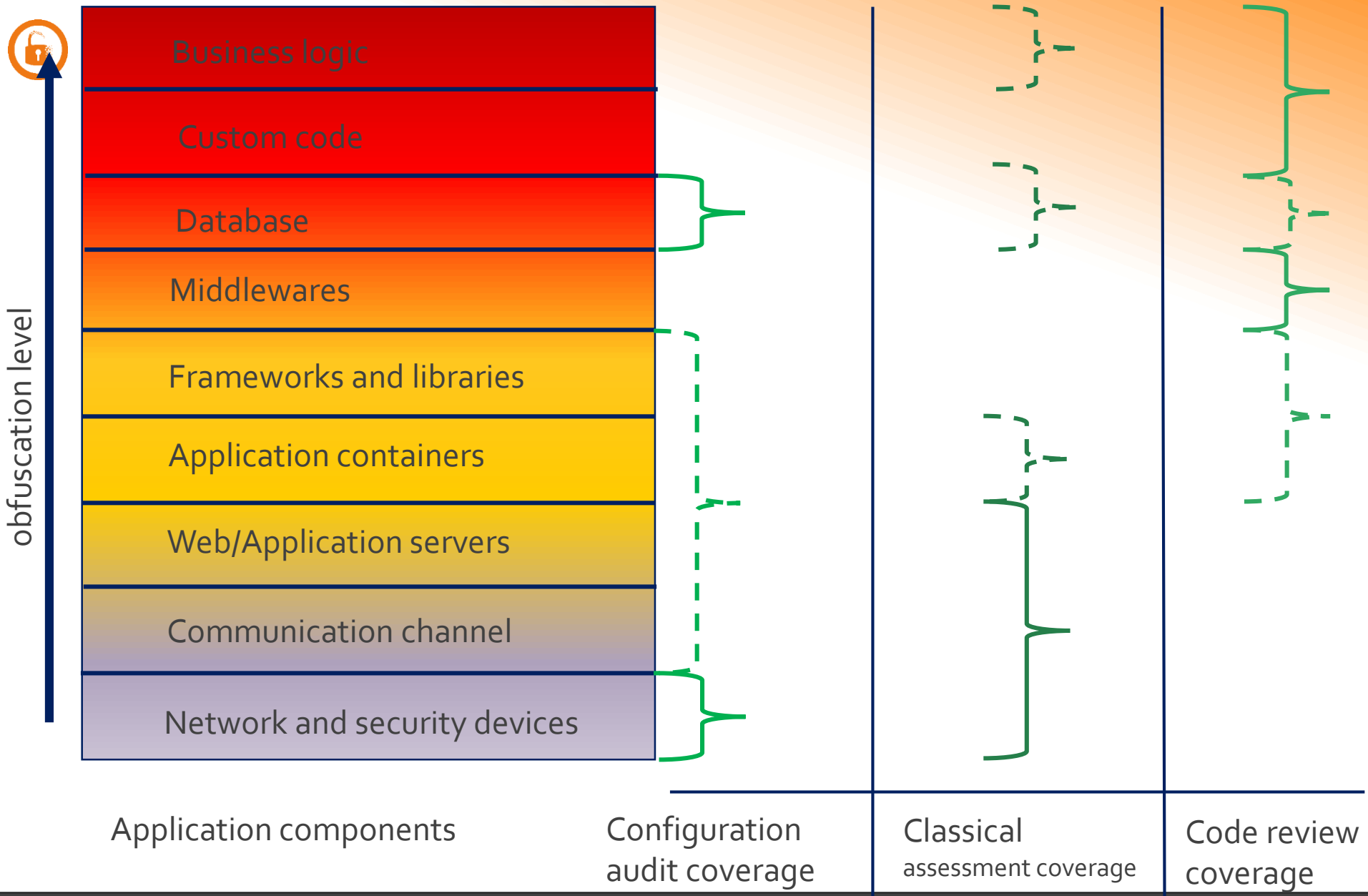
- With a firewall? **NO**
- With an IPS? **NO**
- With a web application firewall? It depends
- From your own users? **NO**
- From another internal applications? **NO**

Application Security Overview

BUT… I'm performing pentests once a year!

Ways to identify threats in applications

EXCELLIUM
Your first call when it comes to IT and Security!



Secure Coding

Agenda

- Introduction
  - Application Security Overview
  - **How to improve?**
- Application security
  - Common Attacks
  - Common Defenses
- Secure Coding principes
- Interesting links
- Questions

How to Improve?

- Know your enemy
  - Learn attacks vectors
  - Learn exploitation steps
- Prepare your defense
  - Include secure SDLC in the contract and in the design
  - Secure coding ☺
- Control your work
  - Add security in your Continous Integration process
  - Perform Code Review during project implementation (in each sprint/milestone)
  - Audit your SDLC

How to Improve ?

OWASP
The Open Web Application Security Project

Learn

Contract

Design

Build

Test

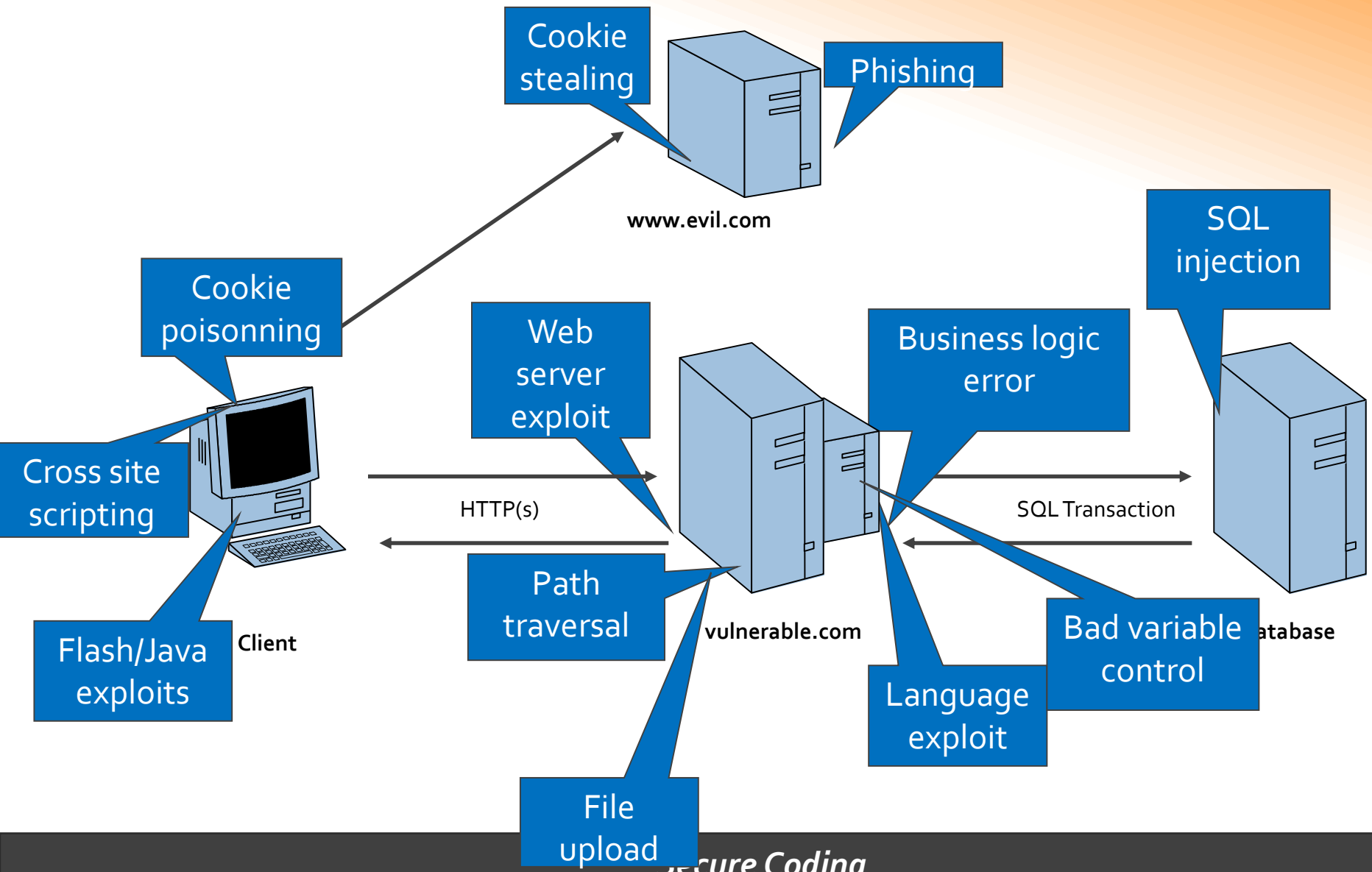
Progress

Application Security

Agenda

- Introduction
  - Application Security Overview
  - How to improve?
- Application security
  - **Common Attacks**
  - Common Defenses
- Secure Coding principes
- Interesting links
- Questions

Common Attacks



Common Attacks

Client side	Session side	Server side	Programming language side	Application side	Data side
XSS ▲Reflective ▲Persistant ▲DOM based ▲CSIT Flash Applets (HTML5 Web Sockets) Clickjacking	Cookie fixation Cookie stealing Cookie guessing CSRF SOP bypass (HTML5)	FingerPrinting Exploit Crowling Path transversal Http methods File Extension Http spliting Http smuggling	Error message Exploit File inclusion Variable control Variable Overwritting Serialization	Error message Business logic Privilege escalation Replay BufferOverFlow Authentication Code injection WSDL discovery SOAP XML DoS XXE	Error message SQL injection SQL Wildcard LDAP injection XML injection XPath injection SMTP header injection

Agenda

- Introduction
  - Application Security Overview
  - How to improve ?
- Application security
  - Common Attacks
  - **Common Defenses**
- Secure Coding principes
- Interesting links
- Questions

Common Defenses

WAF	WAF	Patch management	IPS/WAF	WAF	WAF
Client side	Session side	Server side	Programming language side	Application side	Data side
XSS ▲Reflective ▲Persistant ▲DOM based ▲CSIT Flash Applets (HTML5 Web Sockets) Clickjacking	Cookie fixation Cookie stealing Cookie guessing CSRF SOP bypass (HTML5)	FingerPrinting Exploit Crowling Path transversal http methods File Extension Http spliting Http smuggling	Error message Exploit File inclusion Variable control Variable Overwritting Serialization	Error message Business logic Privilege escalation Replay BufferOverFlow Authentication Code injection WSDL discovery SOAP XML DoS XXE	Error message SQL injection SQL Wildcard LDAP injection XML injection XPath injection SMTP header injection

Common Defenses

WAF	WAF	Patch management	IPS/WAF	WAF	WAF
Client side	Session side	Server side	Programming language side	Application side	Data side
XSS ▲Reflective ▲Persistant ▲DOM based ▲CSIT Flash Applets (HTML5 Web Sockets) Clickjacking	Cookie fixation Cookie stealing Cookie guessing CSRF SOP bypass (HTML5)	FingerPrinting Exploit Crowling Path transversal http methods File Extension Http spliting Http smuggling	Error message Exploit File inclusion Variable control Variable Overwritting Serialization	Error message Business logic Privilege escalation Replay BufferOverFlow Authentication Code injection WSDL discovery SOAP XML DoS XXE	Error message SQL injection SQL Wildcard LDAP injection XML injection XPath injection SMTP header injection

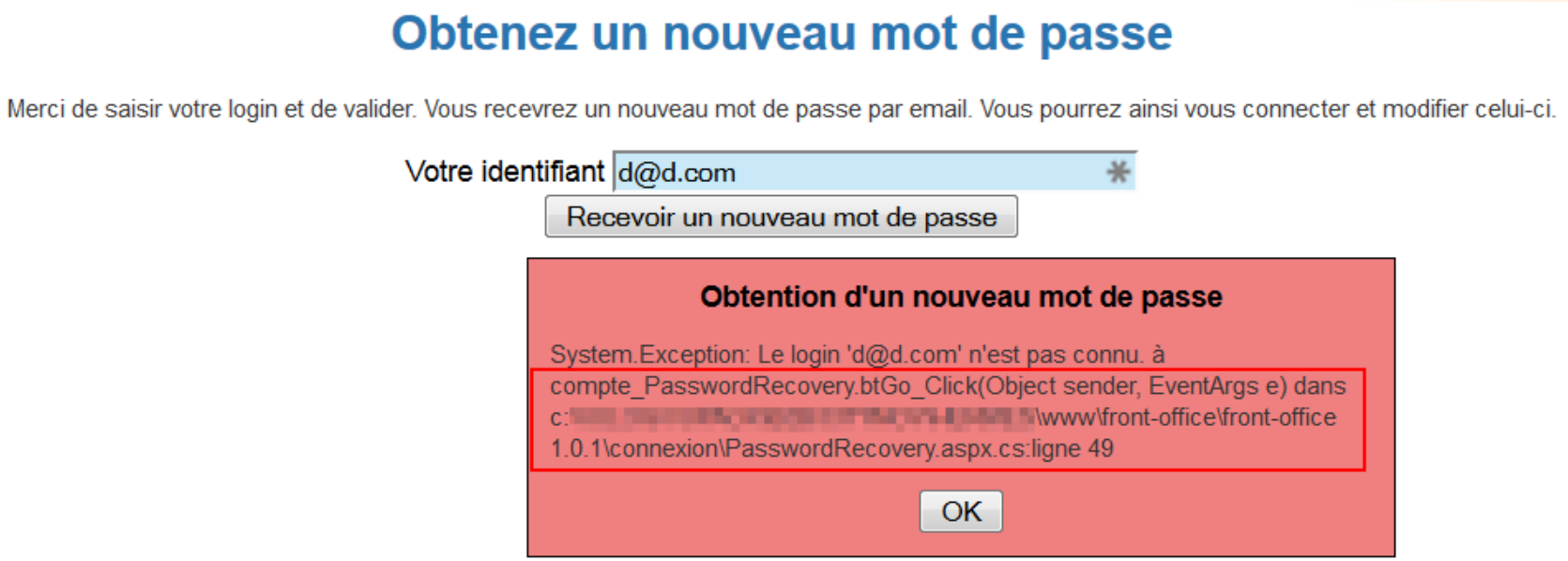
Secure Coding Principles

Secure Coding principles

- Never trust (user) input, use filters
- Never trust… untrusted interfaces
- Deep application defense is mandatory (several layers of defense)
- ALWAYS follow the least privilege principle
- Remember the application will be for users and not security experts
- Log separately application non user related events and user related events

Secure Coding principles

- Do not give information on application failure (crash)



Secure Coding principles

- Clean critical state on exception handler

```
1  public class MyAccessController{
2      public boolean hasAccess(Object securedResource){
3          boolean accessGranted = true;
4
5          try{
6              if(!ESAPI.AccessController().isAuthorizedForData(securedResource)){
7                  accessGranted = false;
8              }
9          }
10         catch(Exception e){
11             log.error(e);
12         }
13
14         return accessGranted;
15     }
16 }
```

Secure Coding principles

- Don't try to hide your code from the reviewers

```
public class SetScorer {
      private int[] gamesWon = {0, 0};

      public void gameWon(int player) {
            gamesWon[player-1]++;
      }

      public String getSetScore() {
            int leader = gamesWon[0] > gamesWon[1] ? 1 : 2;
            int leadersGames = gamesWon[leader -  1];
            int opponentsGames = gamesWon[leader == 1 ? 1 : 0];
            String setScoreMessage = null;
            if ((gamesWon[0] < 6 && gamesWon[1] < 6)
                  || (leadersGames == 6 && opponentsGames == 5)) {
                      setScoreMessage = "Player" + leader + " leads " +
                        leadersGames + " - " + opponentsGames;
            } else if (gamesWon[0] == gamesWon[1]) {
                      setScoreMessage = "Set is tied at " +
                              leadersGames;
            } else if ((leadersGames - opponentsGames >= 2)
                  || (leadersGames == 7)) {
                  setScoreMessage = "Player" + leader +
                        " wins the set " + leadersGames + " - " +
                        opponentsGames;
            }
            return setScoreMessage;
      }
}
```

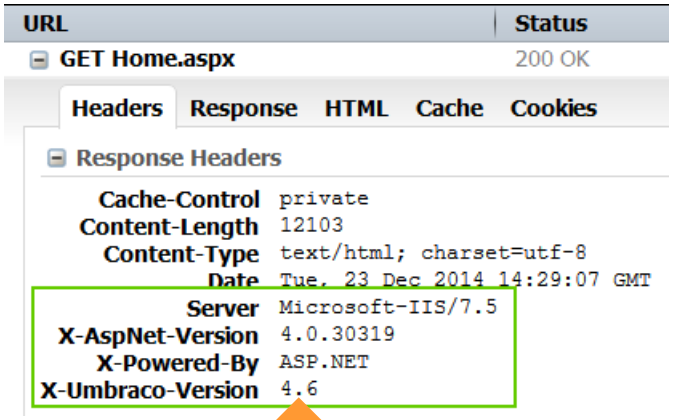
Secure Coding principles

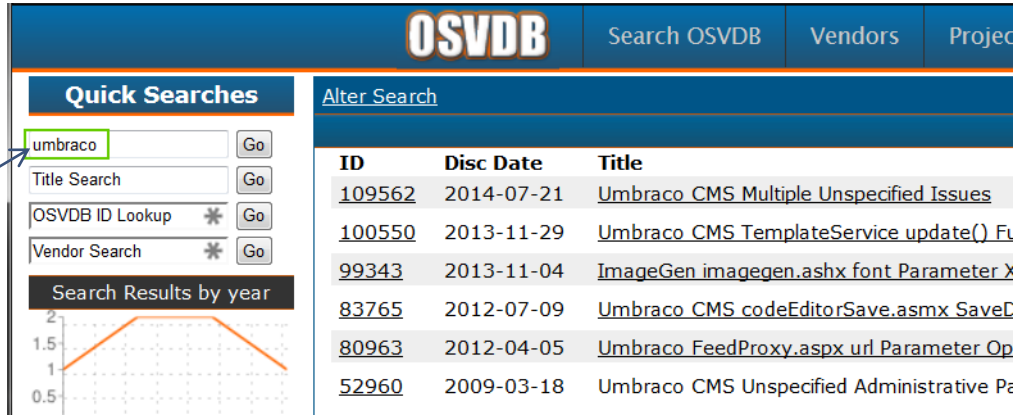
- Comment your contributions but only on server side components in order to avoid to give information to an attacker about your system

```
<!-- =====  Début du contenu ==========================================

<!-- these two div id's are used to setup the main scrolling page area, they sh
<div ID="s4-workspace" class="s4-nosetwidth">  <!-- NOTE: s4-nosetwidth is used
  <div ID="s4-bodyContainer">

      <!-- id="mso_contentdiv" required, helps SharePoint put the web part edit
      <div id="ctl00_MSO_ContentDiv">

        <!-- page editing status bar -->


    <!-- Le Developer dashboard peut être activé par les administrateurs et afficher les infos de debuging et performance -->
    <div id="DeveloperDashboard" class="ms-developerdashboard">

    </div>

  </div><!-- /s4-workspace -->
</div><!-- /s4-bodyContainer -->
</div><!-- /MSO_ContentDiv -->
<script src="/Style Libra[redacted]/pages_initialisation_edit.js" type="text/javascript"></script><!-- JS initialisation page

<!-- ===== fin du contenu ========================================================== -->


<!-- Formdigest est un contrôle de sécurité utilisé pour la validation des formulaires -->
```

Secure Coding principles

- Hide, as much as possible, technical information about your application





Help in attack preparation by:
- Analysing offline source codes and binaries in order to find new vulnerabilities
- Searching for public vulnerabilities

Secure Coding principles

- To protect from the attackers, obfuscation is a possible idea.

  Obfuscation is not the solution because:

  - Obfuscation will not defeat attackers but will make security components uneffective.
  - Obfuscation will just delay the attacker to understand the obfuscated target

    → Uneffective when facing a motivated attacker

- Be evil, test yourself!
- Define a aggressiveness policy used by the application (ex: lock account after a defined number of input validation issues in a business feature…)

Secure Coding principles

Controls

- Controls have to be simple and documented
- Controls have to be done systematically
- Controls have to be functional
- Controls have to use libraries (do not reinvent the wheel) coming from trusted providers (ex: OWASP).

Applications should be able to detect unusual user actions and take measures to defend itself.

Secure Coding principles

Minimum Controls

- Authentication
- Session management
- Authorization
- Input validation and Output encoding
- Error handling and Logging
- Audit Trail
- Trusted network channel/storage:
  - Data protection
  - File system access
  - Cryptography
- Enable defensive features in modern browsers

Secure Coding principles: Authentication

Secure Coding principles

Authentication

- Password length
  - Important: more than 8 chars
  - Critical: more than 10 chars
  - High Critical: more than 14 chars or multi-factor authentication
- Password strength
  - Following the application criticality:
    - A least 1 upper case, 1 lower case, 1 special and 1 digit
    - Must not contains any part of login, username, first name, last name, birthdate from the account owner
  - Do regular dictionary attacks

Secure Coding principles

Authentication: Password cracking stats overview

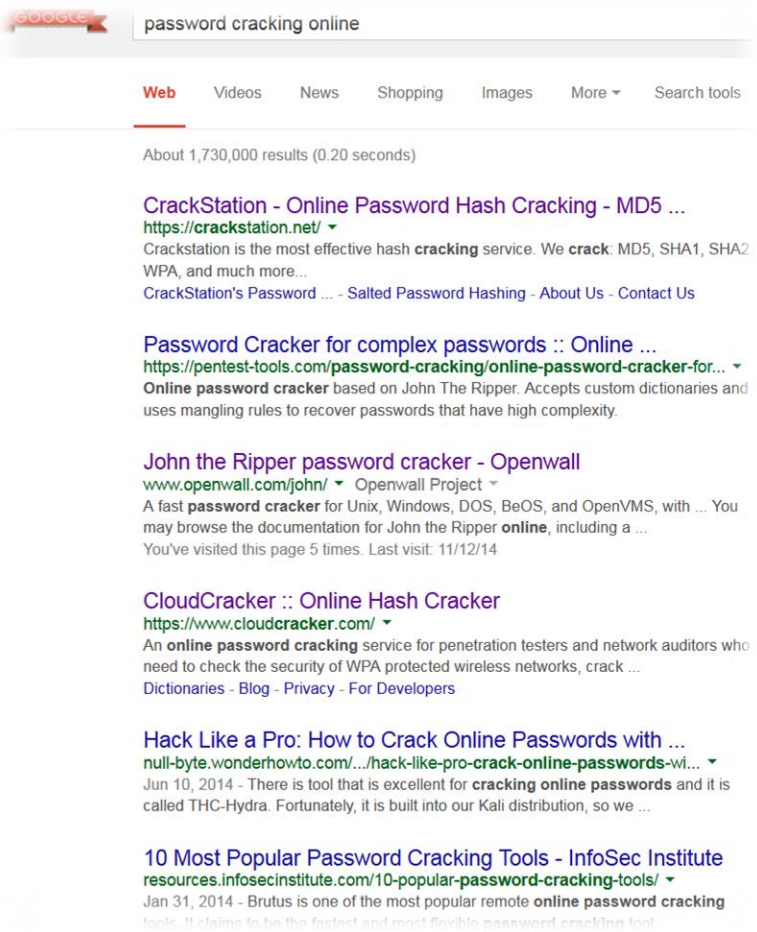
Performance

- PC1: Windows 7, 64 bit · Catalyst 13.8beta1 · 1x AMD hd7970 · stock core clock
- PC2: Windows 7, 64 bit · ForceWare 325.15 · 1x NVidia gtx580 · stock core clock
- PC3: Ubuntu 12.04.1, 64 bit · Catalyst 13.8beta1 · 1x AMD hd6990 · stock core clock
- PC4: Ubuntu 12.04.2, 64 bit · ForceWare 319.37 · 1x NVidia gtx560Ti · stock core clock

Hash Type	PC1	PC2	PC3	PC4
NTLM	7487M c/s	2489M c/s	10935M c/s	1772M c/s
MD5	5144M c/s	1802M c/s	6974M c/s	1363M c/s
SHA1	2030M c/s	785M c/s	3139M c/s	535M c/s
SHA256	1003M c/s	350M c/s	1247M c/s	232M c/s
SHA512	75M c/s	117M c/s	214M c/s	71M c/s
LM	1276M c/s	465M c/s	1004M c/s	242M c/s
phpass P	2071k c/s	789k c/s	2771k c/s	511k c/s
descrypt	63371k c/s	37137k c/s	79100k c/s	18332k c/s
md5crypt 1	3445k c/s	1044k c/s	4425k c/s	648k c/s
bcrypt $2a$	3788 c/s	1583 c/s	3861 c/s	626 c/s
sha512crypt 6	12545 c/s	15153 c/s	34192 c/s	6726 c/s
Password Safe (SHA-256)	495k c/s	158k c/s	648k c/s	106k c/s
IKE-PSK (MD5)	297M c/s	99M c/s	335M c/s	59M c/s
Oracle (DES)	371M c/s	142M c/s	265M c/s	68M c/s
DCC (MD4)	3803M c/s	1181M c/s	5377M c/s	851M c/s
Joomla (MD5)	4609M c/s	1659M c/s	6253M c/s	1172M c/s
MSSQL (SHA1)	1677M c/s	639M c/s	2659M c/s	503M c/s
WPA/WPA2 (PBKDF2)	133k c/s	45k c/s	181k c/s	33k c/s

Secure Coding principles

Authentication: Password cracking cloud service



Because everyone does not have the infrastructure / time / space to build a cracking system, kind guys provide one as SaaS ☺

Secure Coding principles

Authentication

- Passwords are bad because they are static
  - Guessable
  - Forgotten
- Multi-Factor authentication
  - Something the user has (mobile, token)
  - Something the user knows (password, personal detail)

Secure Coding principles

Authentication

- Let the user choose it
- Simple question to ask yourself during architecture phase: "*Is it really necessary to implement the 'I forgot my password' feature?*"

Secure Coding principles

Authentication

- Sign critical transaction with a second authentication
- Force authentication to be in TLS
- Regenerate session once authenticated
- Audit the “change password” functionality
- Enforce account disabling after an established number of invalid login attempts
- Change all vendor-supplied default passwords and user IDs or disable the associated accounts

Secure Coding principles

Authentication

- Use hashing
- Use non predictable Salt (secure random)
- Do not use old MD5,SHA1 (even SHA-256 for password hashing)
- Prefer PBKDF2 / SCRYPT
  - https://www.owasp.org/index.php/Password_Storage_Cheat_Sheet
  - https://cobalt.io/blog/secure-storage-of-passwords-in-your-application-by-jim-manico

Secure Coding principles: Session Management

Secure Coding principles

Session management

- **Protect your cookies**
  - Use the server or framework's session management controls.
  - Do not allow concurrent logins with the same user ID
  - Do not expose session identifiers in URLs, error messages or logs
  - Set the "secure" attribute for cookies transmitted over an TLS connection
  - Set cookies with the "HttpOnly" attribute, unless you specifically require client-side scripts within your application to read or set a cookie's value (*ask yourself why JS is needed to access cookie?)*
  - Set the domain and path for cookies containing authenticated session identifiers to an appropriately restricted value for the site
  - Bind JavaScript events to close session
  - Use JavaScript timers to automatically close sessions

Secure Coding principles

Session management

```
01 <web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
02     xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd"
03     version="3.1">
04
05     <session-config>
06         <session-timeout>15</session-timeout>
07         <cookie-config>
08             <!-- Disable JavaScript access to cookie -->
09             <http-only>true</http-only>
10             <!--  Specify that the cookie will only be sent to secure protocol -->
11             <secure>true</secure>
12         </cookie-config>
13         <!--  Specify that only cookie way will be used to track session -->
14         <tracking-mode>COOKIE</tracking-mode>
15     </session-config>
16
17 </web-app>
```

Secure Coding principles: Authorization

Secure Coding principles

Authorization

- Consider all input as Evil!
- Without Access control, you cannot control the user in the application!

Secure Coding principles

Authorization

- Two levels of authorizations are needed (defense in depth):
  - In the application
  - In the infrastructure
- Control the roles in each sub component

Secure Coding principles

Authorization

- Segregate privileged logic from other application code
- Restrict access to files or other resources, including those outside the application's direct control, to only authorized users
- Restrict access to protected URLs to only authorized users
- Restrict access to protected functions to only authorized users
- Restrict direct object references to only authorized users
- Restrict access to services to only authorized users
- Restrict access to application data to only authorized users
- Restrict access to user and data attributes and policy information used by access controls
- Restrict access security-relevant configuration information to only authorized users
- If state data must be stored on the client, use encryption and integrity checking on the server side to catch state tampering (if possible, associate client IP to is web session in order avoid “replay” attack).

Secure Coding principles

Authorization

- Limit the number of transactions a single user or device can perform in a given period of time.
- If long authenticated sessions are allowed, periodically re-validate a user's authorization.
- Implement account auditing and enforce the disabling of unused accounts
- The application must support disabling of accounts and terminating sessions when authorization are removed.
- Service accounts or accounts supporting connections to, or from external systems should have the least privilege possible

Secure Coding principles

Authorization

- Create an Access Control Policy to document an application's business rules, data types and access authorization criteria and/or processes so that access can be properly provisioned and controlled. This includes identifying access requirements for both the data and system resources
- Apply authorization check before to call any business processing method
- Apply authorization definition using url constraints (mapping) AND method constraints (annotation)

To avoid authorization misconfiguration
the « **Deny by default** » should be used !

Secure Coding principles : Input Validation

Secure Coding principles

Input Validation

- Conduct all data validation on a trusted system (e.g. the server)
- Identify all data sources and classify them into trusted and untrusted.
- Validate all data from untrusted sources (e.g. databases, file streams, etc.)
- There should be a centralized input validation routine/API for the application
- Specify proper character sets, such as UTF-8, for all sources of input

Secure Coding principles

Input Validation

- Consider validating data along all the entry points of the application borders
  - API
  - User input
  - Middleware
  - Database

Secure Coding principles

Input Validation: Whitelist

- Reject unvalidated data
- Validate Data
  - Java Types
  - Numeric Ranges
  - Regular Expression:
    - Document precisely them because everyone is not a master in RegEx and in maintenance phase of an application, it is “easy” to break the initial goal of the RegEx ☺
  - Expected length
  - Expected values
  - Whitelist if possible
  - Character allowed repetition whitelist

Secure Coding principles

Input Validation: Encoding

```
HTML encoding            > can be written as &lt; or &#60;
Javascript encoding      " can be written as \x22
ASCII-7                  <script> becomes +ADw-script+AD4-

No encoding              <IMG SRC=javascript:alert("!")>
UTF-8 encoding           <IMG SRC=&#106;&#97;&#118;&#97;&#115;&#99;&#114;...
UTF-8 econding           <IMG SRC=&#0000106&#0000097&#0000118&#0000097...
Insert white space       <IMG SRC="jav   ascript:alert('!');">
Insert encoded white space <IMG SRC="jav&#x09;ascript:alert('!');">
```

Secure Coding principles

Input Validation: Pattern matching

```
public Boolean validateSSN(String ssnVal) {
    if (ssnVal.matches("^[1|2] [0-9]{2} \\d{2} \\d{2} \\d{6} \\d{2}$")) {
        return true;
    } else {
        return false;
    }
}
```

```
public boolean validateAmount (String amount) {
    if (validate(amount, "^\\d+$")){
        if (Long.parseLong(amount) < 10000L){
            return true;
        }
    }
    return false;
}
```

Secure Coding principles

Input Validation example: SQL context

```
class Login {
    public Connection getConnection() throws SQLException {
        DriverManager.registerDriver(new com.microsoft.sqlserver.jdbc.SQLServerDriver());
        String dbConnection = PropertyManager.getProperty("db.connection");
        // can hold some value like
        // "jdbc:microsoft:sqlserver://<HOST>:1433,<UID>,<PWD>"
        return DriverManager.getConnection(dbConnection);
    }
    String hashPassword(char[] password) {
        // create hash of password
        return ("Hash");
    }

    public void doPrivilegedAction(String username, char[] password) throws SQLException {
        Connection connection = getConnection();
        if (connection == null) {
            // handle error
        }
        try {
            String pwd = hashPassword(password);
            String sqlString = "SELECT * FROM db_user WHERE username = '" + username +"' AND password = '" + pwd + "'";
            Statement stmt = connection.createStatement();
            ResultSet rs = stmt.executeQuery(sqlString);
            if (!rs.next()) {
                throw new SecurityException("User name or password incorrect" );
            }
            // Authenticated; proceed
            } finally {
                try {
                    connection.close();
                    } catch (SQLException x) {
                        // forward to handler
                    }
                }
        }
    }
```

125

Secure Coding principles

Input Validation example: SQL context

```
class Login {
    public Connection getConnection() throws SQLException {
        DriverManager.registerDriver(new com.microsoft.sqlserver.jdbc.SQLServerDriver());
        String dbConnection = PropertyManager.getProperty("db.connection");
        // can hold some value like
        // "jdbc:microsoft:sqlserver://<HOST>:1433,<UID>,<PWD>"
        return DriverManager.getConnection(dbConnection);
    }
    String hashPassword(char[] password) {
        // create hash of password
        return ("Hash");
    }

    public void doPrivilegedAction(String username, char[] password) throws SQLException {
        Connection connection = getConnection();
        if (connection == null) {
            // handle error
        }
        try {
            String pwd = hashPassword(password);
            String sqlString = "SELECT * FROM db_user WHERE username = '" + username +"' AND password = '" + pwd + "'";
            Statement stmt = connection.createStatement();
            ResultSet rs = stmt.executeQuery(sqlString);
            if (!rs.next()) {
                throw new SecurityException("User name or password incorrect" );
            }
            // Authenticated; proceed
            } finally {
                try {
                    connection.close();
                    } catch (SQLException x) {
                        // forward to handler
                    }
                }
        }
    }
```

What is wrong here ?

125

Secure Coding principles

Input Validation example : SQL context

```
class Login {
    public Connection getConnection() throws SQLException {
        DriverManager.registerDriver(new com.microsoft.sqlserver.jdbc.SQLServerDriver());
        String dbConnection = PropertyManager.getProperty("db.connection");
        // can hold some value like
        // "jdbc:microsoft:sqlserver://<HOST>:1433,<UID>,<PWD>"
        return DriverManager.getConnection(dbConnection);
    }
    String hashPassword(char[] password) {
        // create hash of password
        return ("Hash");
    }

    public void doPrivilegedAction(Stri      rname, char[] pass      throws SQLException {
        Connection connection = getCon
        if (connection == null) {
            // handle error
        }
        try {
            String pwd = hashPassword(passwo
            String sqlString = "SELECT * FR      me = '" + username +"' AND password = '" + pwd + "'";
            Statement stmt = connection.
            ResultSet rs = stmt.execu
            if (!rs.next()) {
                throw new SecurityExc      name or p      rrect" );
            }
            // Authenticated; proceed
            } finally {
                try {
                    connection.close();
                    } catch (SQLException x) {
                        // forward to handler
                    }
                }
        }
    }
```

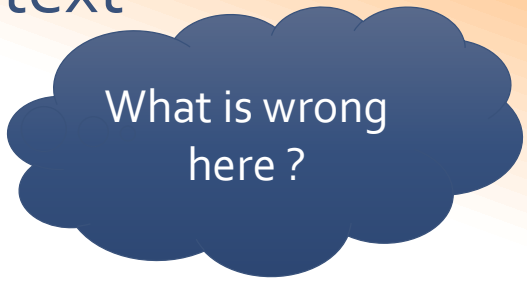
Secure Coding principles

Input Validation example: SQL context

```
Connection connection = getConnection();
if (connection == null) {
    // Handle error
}
try {
    String pwd = hashPassword(password);
    // Ensure that the length of user name is legitimate if ((username.length() > 8) {
    // Handle error }
    String sqlString = "select * from db_user where username=? and password=?";
    PreparedStatement stmt = connection.prepareStatement(sqlString);
    stmt.setString(1, username);
    stmt.setString(2, pwd);
    ResultSet rs = stmt.executeQuery();
    if (!rs.next()) {
        throw new SecurityException("User name or password incorrect");
    }
    // Authenticated, proceed
} finally {
try {
    connection.close();
} catch (SQLException x) {
    // forward to handler
}
```

Secure Coding principles

Input Validation example: JPA/Entity context

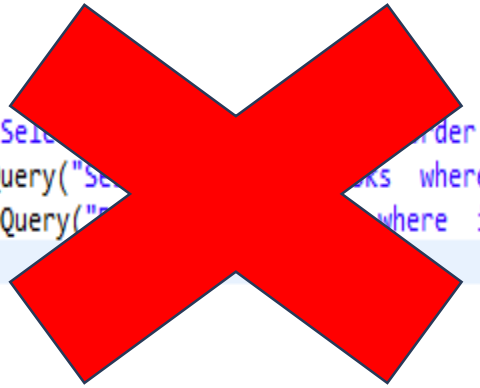



```
List results = entityManager.createQuery("Select order from Orders order where order.id = "+ orderId).getResultList();
List results = entityManager.createNativeQuery("Select * from Books where author = "+ author).getResultList();
int resultCode = entityManager.createNativeQuery("Delete from Cart where itemId = "+ itemId).executeUpdate();
```

Secure Coding principles

Input Validation example: JPA/Entity context

```
List results = entityManager.createQuery("Sel...der where order.id = "+ orderId).getResultList();
List results = entityManager.createNativeQuery("Se...ks where author = "+ author).getResultList();
int resultCode = entityManager.createNativeQuery("...where itemId = "+ itemId).executeUpdate();
```



Secure Coding principles

Input Validation example: JPA/Entity context

```
/* positional parameter in JPQL */
Query jpqlQuery = entityManager.createQuery("Select order from Orders order where order.id = ?1");
List results = jpqlQuery.setParameter(1, "123-TEST-LABEL").getResultList();
```

```
/* Native SQL */
Query sqlQuery = entityManager.createNativeQuery("Select * from Books where author = ?", Book.class);
List results = sqlQuery.setParameter(1, "Charles Dickens").getResultList();
/* named query in JPQL --- Query named "myCart" being "Select c from Cart c where c.itemId = :itemId" */
Query jpqlQuery = entityManager.createNamedQuery("myCart");
List results = jpqlQuery.setParameter("itemId", "item---id---0001").getResultList();
/* named parameter in JPQL */
Query jpqlQuery = entityManager.createQuery("Select emp from Employees emp where emp.incentive > :incentive");
List results = jpqlQuery.setParameter("incentive", new Long(10000)).getResultList();
```

Secure Coding principles

Input Validation example: XML context

What is wrong here ?

```
public class bad {
    private void createXMLStream(BufferedOutputStream outStream, String quantity) throws IOException {
        String xmlString;
        xmlString = "<item>\n<description>Widget</description>\n" +
        "<price>500.0</price>\n" +
        "<quantity>" + quantity + "</quantity></item>"; outStream.write(xmlString.getBytes());
          outStream.flush();
        }
}
```

127

Secure Coding principles

Input Validation example: XML context

Secure Coding principles

Input Validation example: XML context

Check param before to use it

```
public class good {
    private void createXMLStream(BufferedOutputStream outStream, String quantity) throws IOException {
        // Write XML string if quantity contains numbers only.
        // Blacklisting of invalid characters can be performed
        // in conjunction
        if (!Pattern.matches("[0-9]+", quantity)) {
            // Format violation
        }
        String xmlString = "<item>\n<description>Widget</description>\n" + "<price>500</price>\n" +
        "<quantity>" + quantity + "</quantity></item>"; outStream.write(xmlString.getBytes());
          outStream.flush();
        }
```

Secure Coding principles

Input Validation example: XML context

Check using XSD

```
01 <?xml version = "1.0" encoding = "utf-8"?>
02 <xs:schema  xmlns:xs="http://www.w3.org/2001/XMLSchema">
03     <xs:element  name="item">
04         <xs:complexType>
05             <xs:sequence>
06             <xs:element  name="description" type="xs:string"/>
07             <xs:element  name="price" type="xs:decimal"/>
08             <xs:element  name="quantity" type="xs:integer"/>
09             </xs:sequence>
10         </xs:complexType>
11     </xs:element>
12 </xs:schema>
13
```

Secure Coding principles

Input Validation example: XML context

Check using XSD

```
private void createXMLStream(BufferedOutputStream outStream, String quantity)
        throws IOException {
    String xmlString;
    xmlString = "<item>\n<description>Widget</description>\n" + "<price>500.0</price>\n" +"<quantity>" + quantity + "</quantity></item>";
    InputSource xmlStream = new InputSource(new StringReader(xmlString) );

    // Build a validating SAX parser using our schema
    SchemaFactory sf = SchemaFactory.newInstance(XMLConstants.W3C_XML_SCHEMA_NS_URI);
    DefaultHandler defHandler = new DefaultHandler() {
        public void warning(SAXParseException s) throws SAXParseException {throw s;}
        public void error(SAXParseException s) throws SAXParseException {throw s;}
        public void fatalError(SAXParseException s) throws SAXParseException {throw s;}
    };

    StreamSource ss = new StreamSource(new File("schema.xsd"));
    try {
        Schema schema = sf.newSchema(ss);
        SAXParserFactory spf = SAXParserFactory.newInstance();
        spf.setSchema(schema);
        SAXParser saxParser = spf.newSAXParser();
        // To set the custom entity resolver,
        // an XML reader needs to be created
        XMLReader reader = saxParser.getXMLReader();
        reader.setEntityResolver(new CustomResolver());
        saxParser.parse(xmlStream, defHandler);
    } catch (ParserConfigurationException x) {
            throw new IOException("Unable to validate XML", x);
    } catch (SAXException x) {
        throw new IOException("Invalid quantity", x);
    }
    // Our XML is valid, proceed outStream.write(xmlString.getBytes()); outStream.flush();
}
```

Secure Coding principles

Input Validation example: LDAP context

What is wrong here ?

```
DirContext ctx = new InitialDirContext(env);

String managerName = request.getParameter("managerName");

//retrieve all of the employees who report to a manager

String filter = "(manager=" + managerName + ")";

NamingEnumeration employees = ctx.search("ou=People,dc=example,dc=com",
                                         filter);
```

Secure Coding principles

Input Validation example: LDAP context

```
DirContext ctx = new InitialDirContext(env);

String managerName = request.getParameter("managerName");

//retrieve all of the employees who report to a manager

String filter = "(manager=" + managerName + ")";

NamingEnumeration employees = ctx.search("ou=People,dc=example,dc=com",
                                         filter);
```

Secure Coding principles

Input Validation example: LDAP context

Check param before to use it

```
public NamingEnumeration extractEmployees {
    DirContext ctx = new InitialDirContext(env);
    String managerName = request.getParameter("managerName");
    //retrieve all of the employees who report to a manager
    if (validate (managerName, "^\\w$")){
        String filter = "(manager=" + managerName + ")";
        NamingEnumeration employees = ctx.search("ou=People,dc=example,dc=com", filter);
    }
}
public boolean validate (String s, String pat) {
    return (Pattern.matches(pat, s));
}
```

Secure Coding principles

Input Validation example: OWASP ESAPI validators

```
/**
 * Show use of somes validators.<br>
 * Validators RegEx are defined in a file named "validation.properties" located in classpath root.<br>
 * RegEx configuration example: <br>
 * <code>Validator.Email=^[A-Za-z0-9._%-]+@[A-Za-z0-9.-]+\\.[a-zA-Z]{2,4}$</code><br>
 * <code>Validator.AccountName=^[a-zA-Z0-9]{3,20}$</code>
 *
 * @throws Exception
 */
@Test
public void testValidators() throws Exception {
    // Get ref on validator instance
    Validator validator = ESAPI.validator();
    boolean isValid = false;

    // Method signature:
    // isValidInput(java.lang.String context, java.lang.String input, java.lang.String inputType,
    //              int maxLength, boolean allowNull, boolean canonicalize)

    /* Show Email validation */
    isValid = validator.isValidInput("demo", "dominique.righetto@gmail.com", "Email", 100, false, true);
    Assert.assertTrue(isValid);

    /* Show URL validation */
    isValid = validator.isValidInput("demo", "http://www.google.com", "URL", 50, false, true);
    Assert.assertTrue(isValid);

    /* Show IPV4 address validation */
    isValid = validator.isValidInput("demo", "192.168.1.1", "IPAddress", 50, false, true);
    Assert.assertTrue(isValid);

    /* Show custom validator */
    isValid = validator.isValidInput("demo", "righettod", "AccountName", 50, false, true);
    Assert.assertTrue(isValid);

}
```

Secure Coding principles: Output Encoding

Secure Coding principles

Output encoding

- Encode on the server
- Defense in depth principle
- Again, centralize the encoder functions
- Sanitize the data sent to client
  - HTML encode (minimum)
  - HTML purifier

Secure Coding principles

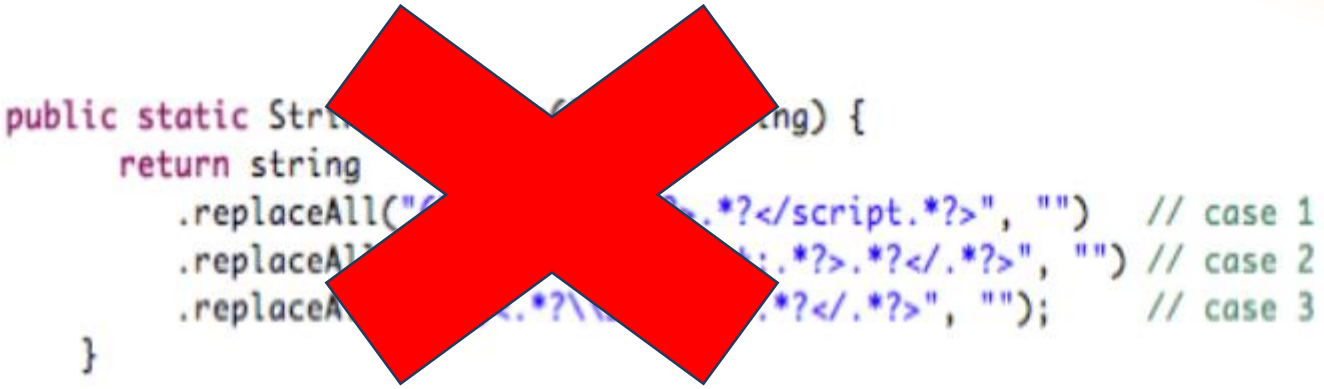
Output encoding: First try

What is wrong here ?

```
public static String sanitize(String string) {
      return string
         .replaceAll("(?i)<script.*?>.*?</script.*?>", "")   // case 1
         .replaceAll("(?i)<.*?javascript:.*?>.*?</.*?>", "") // case 2
         .replaceAll("(?i)<.*?\\s+on.*?>.*?</.*?>", "");     // case 3
   }
```

Secure Coding principles

Output encoding: First try



Secure Coding principles

Output encoding: Second try

```
class HtmlSanitizer
{
    /// A regex that matches things that look like a HTML tag after HtmlEncoding to &#DECIMAL; notation. Microsoft AntiXSS 3.0 can be used to preform this. Splits
    /// chunks that start with &#60; and ends with either end of line or &#62;
    private static readonly Regex _tags = new Regex(@"&\#60;(?!&\#62;).+?(&\#62;|$)", RegexOptions.Singleline | RegexOptions.ExplicitCapture | RegexOptions.Compile
    /// A regex that will match tags on the whitelist, so we can run them through
    /// HttpUtility.HtmlDecode
    /// FIXME - Could be improved, since this might decode &#60; etc in the middle of
    /// an a/link tag (i.e. in the text in between the opening and closing tag)
    private static readonly Regex _whitelist = new Regex(@"^&\#60;(&\#47;)? (a|b(lockquote)?|code|em|h(1|2|3)|i|li|ol|p(re)?|s(ub|up|trong|trike)?|ul)&\#62;$|^&\#6
      RegexOptions.Singleline | RegexOptions.IgnorePatternWhitespace |RegexOptions.ExplicitCapture | RegexOptions.Compiled);
    /// HtmlDecode any potentially safe HTML tags from the provided HtmlEncoded HTML input using
    /// a whitelist based approach, leaving the dangerous tags Encoded HTML tags
    public static string Sanitize(string html)
    {
        Match tag;
        MatchCollection tags = _tags.Matches(html);

        // iterate through all HTML tags in the input
        for (int i = tags.Count - 1; i > -1; i--)
        {
            tag = tags[i];
            string tagname = tag.Value.ToLowerInvariant();

            if (_whitelist.IsMatch(tagname))
            {
                // If we find a tag on the whitelist, run it through
                // HtmlDecode, and re-insert it into the text
                string safeHtml = HttpUtility.HtmlDecode(tag.Value);
                html = html.Remove(tag.Index, tag.Length);
                html = html.Insert(tag.Index, safeHtml);
            }
        }
        return html;
    }
}
```

What is wrong here ?

Secure Coding principles

Output encoding: Second try

```
class HtmlSanitizer
{
    /// A regex that matches things that look like a HTML tag after HtmlEncoding to &#DECIMAL; notation. Microsoft AntiXSS 3.0 can be used to preform this. Splits
    /// chunks that start with &#60; and ends with either end of line or &#62;
    private static readonly Regex _tags = new Regex(@"&\#60;(?!&\#62;).+?(&\#62;|$)", RegexOptions.Singleline | RegexOptions.ExplicitCapture | RegexOptions.Compile
    /// A regex that will match tags on the whitelist, so we  un them through
    /// HttpUtility.HtmlDecode
    /// FIXME - Could be improved, since this might decod  he middl
    /// an a/link tag (i.e. in the text in between the a
    private static readonly Regex _whitelist = new Regex(@"  odelemlh(1|2|3)|i|li|ol|p(re)?|s(ub|up|trong|trike)?|ul)&\#62;$|^&\#6
      RegexOptions.Singleline | RegexOptions.IgnorePatternWhite  tCapture | RegexOptions.Compiled);
    /// HtmlDecode any potentially safe HTML tags from the provide  ut using
    /// a whitelist based approach, leaving the dangerous tags
    public static string Sanitize(string html)
    {
        Match tag;
        MatchCollection tags = _tags.Matches(html);

        // iterate through all HTML tags in the input
        for (int i = tags.Count - 1; i > -1; i--)
        {
            tag = tags[i];
            string tagname = tag.Value.ToLowerInvariant();

            if (_whitelist.IsMatch(tagname))
            {
                // If we find a tag on the whitelist, run it through
                // HtmlDecode, and re-insert it into the text
                string safeHtml = HttpUtility.HtmlDecode(tag.Value);
                html = html.Remove(tag.Index, tag.Length);
                html = html.Insert(tag.Index, safeHtml);
            }
        }
        return html;
    }
}
```

Secure Coding principles

Output encoding: Good solution

```
// AntiSammy
//http://code.google.com/p/owaspantisamy/downloads/list
Policy policy = Policy.getInstance(POLICY_FILE_LOCATION);
AntiSamy as = new AntiSamy();
CleanResults cr = as.scan(dirtyInput, policy);
MyUserDAO.storeUserProfile(cr.getCleanHTML()); // some custom function


// OWASP Java Sanitizer
// http://owasp-java-html-sanitizer.googlecode.com/svn/trunk/distrib/javadoc/org/owasp/html/Sanitizers.html
 PolicyFactory sanitizer = Sanitizers.FORMATTING.and(Sanitizers.BLOCKS);
  String cleanResults = sanitizer.sanitize("<p>Hello, <b>World!</b>");
```

Secure Coding principles: Error Handling & Logging

Secure Coding principles

Error Handling

- The application will crash
- Catch all exceptions without exception (even NullPointerException ☺)
- Use a global handler in order to ensure that any “unexpected error case” will be managed (using, for example, error page configuration feature)
- In case of exception, all layers under the root call layer “pass the exception” above and the exception is managed at caller level
- Clear all sensitive data from exception code
- Don’t give detail, send a generic error message to the user like “A error occur.”
- Logs are sensitive, are they world readable ? World writable ?
- If possible, don’t expose log traces into your application
- Document explicitly any “silent catch clause”, if possible simply avoid them by adding, at least a log trace.

Secure Coding principles

Error Handling

- Log events below non related to user action
  - Exceptions (database connection failure, back end unavailable, out of memory, no more space on disk…)
  - Crypto issues (backend server TLS certificate validation failure…)
  - Session management issues (server cannot create session…)
- Events related to a user (i.e. starting with a action from a user) are sent to the Audit trail log
- Split your logs according to content type (severity level is used to classify the importance of information)
  - Application events
  - Audit trail events (user events)
- Use API like LogBack (use SLF4J if your module must not force the logging API implementation) to manage your logging and events destination abstraction.

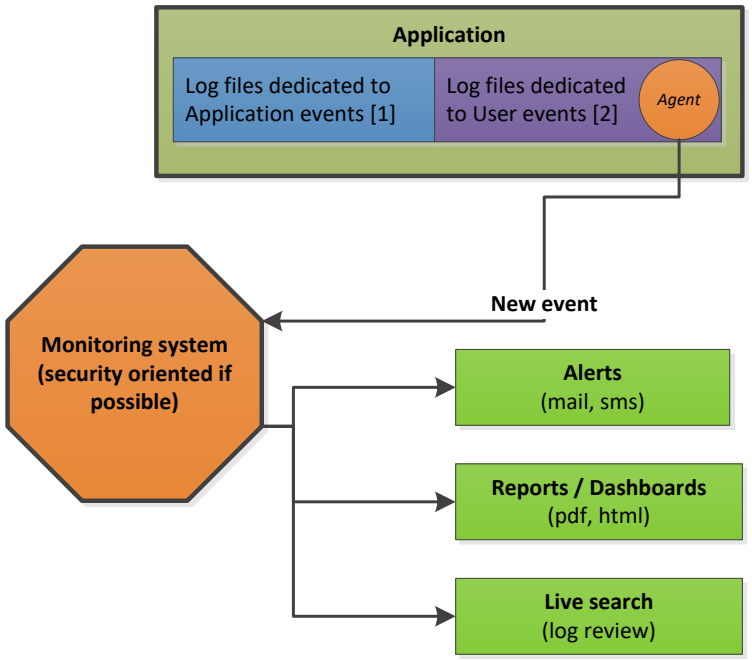
Secure Coding principles

Error Handling: Log distribution approach explanation

- Most of the time application log files:
  - Are not analyzed/reviewed unless in case of technical or business error
  - Have a history up to 2 month max (used to save disk space)
  - Include events generated by application itself and by user
  - Are not archived and not protected against injection/corruption
  - Are not formatted in order to enable log review
- From a security point of view (your security department guys ☺):
  - It's hard to track/detect action performed by a malicious user (directly or through a malware) on a timeframe from 6 month to 1 year or that application is under fingerprinting (attack in preparation)
  - Require to separate application events from users events (or from a specific user) → Make it hard to integrate events into a analysis system and then make it difficult to setup alerting/action rules

Secure Coding principles

Error Handling: Log distribution approach explanation



Implementation example:
→ *Logging system:*
LogBack
→ *Monitoring system:*
Splunk
→ *Agent:*
Splunk Universal Forwarder

[1] Auto clean by rolling on 10 files
[2] Auto clean by rolling on 2 files but files are watched by a agent sending each new event to Monitoring system

Secure Coding principles

Error Handling

What is wrong here ?

```
@Override
protected void doPost(HttpServletRequest request, HttpServletResponse response) throws IOException {
    SensitiveData data = new SensitiveData("123456789");
    Files.write(Paths.get("/secure_store/file.txt"), data.getSsn().getBytes(), StandardOpenOption.WRITE);
    // Do stuff with new file
}
```

```
@Override
protected void doPost(HttpServletRequest request, HttpServletResponse response)  {
    try {
        SensitiveData data = new SensitiveData("123456789");
        Files.write(Paths.get("/secure_store/file.txt"), data.getSsn().getBytes(), StandardOpenOption.WRITE)
        // Do stuff with new file
    }
    catch (IOException e) {}
}
```

```
@Override
protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException {
    try {
        SensitiveData data = new SensitiveData("123456789");
        Files.write(Paths.get("/secure_store/file.txt"), data.getSsn().getBytes(), StandardOpenOption.WRITE);
        // Do stuff with new file
    }
    catch (IOException e) {
        throw new ServletException(e);
    }
}
```

Secure Coding principles

Error Handling

```
@Override
protected void doPost(HttpServletReque                    ttpServletResponse response) throws IOException {
    SensitiveData data = new SensitiveDa                  6789");
    Files.write(Paths.get("/secure_stor                   "), data.getSsn().getBytes(), StandardOpenOption.WRITE);
    // Do stuff with new file
}
```

```
@Override
protected void doPost(HttpServletRequest  equest, HttpServletResponse response)  {
    try {
        SensitiveData data = new Sensit                   3456789");
        Files.write(Paths.get("/secure_st                 .txt"), data.getSsn().getBytes(), StandardOpenOption.WRITE)
        // Do stuff with new file
    }
    catch (IOException e) {}
}
```

```
@Override
protected void doPost(HttpServletRequest  quest, HttpServletResponse response) throws ServletException {
    try {
        SensitiveData data = new Sensiti                  456789");
        Files.write(Paths.get("/secure_st                 txt"), data.getSsn().getBytes(), StandardOpenOption.WRITE);
        // Do stuff with new file
    }
    catch (IOException e) {
        throw new ServletException(e);
    }
}
```

Secure Coding principles

Error Handling: Better

```java
@Override
protected void doPost(HttpServletRequest request, HttpServletResponse response) {
    SensitiveData data = null;
    try {
        data = new SensitiveData("123456789");
        // Use JDK API helper for I/O in order to facilitate release of resources
        Files.write(Paths.get("/secure_store/file.txt"), data.getSsn().getBytes(), StandardOpenOption.WRITE);
        // Do stuff with new file
    }
    catch (IOException | UnsupportedOperationException | SecurityException e) {
        // Don't use sensitive data into your log !
        // Handle error that can be throw according to API methods signatures
        LogEventManagerUtils.report(e, true);
    }
    catch (Exception e) {
        // Don't use sensitive data into your log !
        // Handle other unexpected error case because we are in a top call position before client side
        LogEventManagerUtils.report(e, true);
    }
    finally {
        // Clear senstive data in all case (normal and error) before to exit method
        if (data != null) {
            data.setSsn(null);
        }
    }
}
```

Secure Coding principles

Error Handling: Global handler

```java
@Override
protected void doGet(HttpServletRequest request, HttpServletResponse response) {
    try {
        List<String> lines = Files.readAllLines(Paths.get("myfile.txt"), Charset.forName("utf8"));
        // Do stuff with "lines"
    }
    catch (IOException | SecurityException e) {
        // Delegate management of the exception to a global handler
        // The boolean parameter indicate that the event is associated to a user action
        LogEventManagerUtils.report(e, true);
    }
}
```

Code level

Configuration level

```xml
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd"
    version="3.1">

  <!-- Specify an error page for root exception in order to avoid to let any "Exception"/"Error" reach the client side  -->
  <error-page>
    <exception-type>java.lang.Throwable</exception-type>
    <location>/error.jsp</location>
  </error-page>
</web-app>
```

Secure Coding principles: Audit trail

Secure Coding principles

Audit trail

- **Objective:**
  - Used to track set of actions performed by a user on a period of time.
- **Properties:**
  - Once wrote, a event (trace) cannot be modified or deleted
  - Event should be sent to a centralized monitoring system (ex: Splunk) in order to enable search, reporting and alerting
  - Events must be archived (using the monitoring system) in order to allow investigation about a security issue on application
  - Accesses to events should be restricted to specific people role like application administrator or company security department
  - Event content must be protected against any form of injection (using output encoding) or corruption
  - Event format should use a standard format like “Common Event Format” (objective is to facilitate log parsing by the monitoring system)

Secure Coding principles

Audit trail

- **Type of action tracked for a user:**
  - Authentication success and failure
  - Authorization denied
  - Input validation failure
  - Action on business data (Create, Read, Update, Delete)
  - Administrative actions (admin users are also tracked)
  - Account disabling and enabling
  - Any error related to user action like submitting an expired web session ID, an invalid/unexpected cookie…
  - Logout

Secure Coding principles

Audit trail

- **Information included into event:**
  - Current date & time
  - User identifier (login and, if possible, client IP address)
  - Application unique identifier:
    - Used to identify application in monitoring system
  - Correlation unique identifier (ex: UUID)
    - Used to identify user action in a context of a application composed by several modules (ex: Front End + Back End).
    - This ID is shared in all modules and it has a session lifetime (generate it during login phase and keep it into web session, don't use web session ID as correlation ID ☺)
  - Server name:
    - Used in clustering context in order to identify target node
  - Severity level (INFO / WARN / ERROR)
  - Event detail

Secure Coding principles

Audit trail

- **Note about event detail:**
  - Detail must not contains any sensitive information like web session ID, password, IBAN, credit card ID, Security social number…
  - Technical ID can be used into detail in order to give business context information about event (enable possibility to create specific alert in monitoring system).

- **Example of event detail:**
  - User “righettod” delete customer with reference ID “123”
  - User “righettod” login failed using IP address “x.x.x.x”
  - User “righettod” enter disallowed characters into field “communication” of the “transfer” form

Secure Coding principles: Data Protection

Secure Coding principles

Data Protection

- Centralize sensitive data storage and access point (ex: internal stateless web service) in order to enforce security on data access
- Protect sensitive data : encryption or secure hashing
- Do not cache sensitive data on client side (use cache HTTP headers)
- Clean sensitive data from server side (session/cache/memory) when they are not needed anymore
- Trusted store for trusted data
- HTTP GET with parameter in URL is forbidden (use headers of HTTP GET request to pass parameters to server)
- Simple question to ask yourself during architecture phase: "*Do I really need to store this information into my application or can I simply call the provider each time and use memory caching if necessary ?*"

Secure Coding principles: FileSystem Access

Secure Coding principles

FileSystem Access: path manipulation

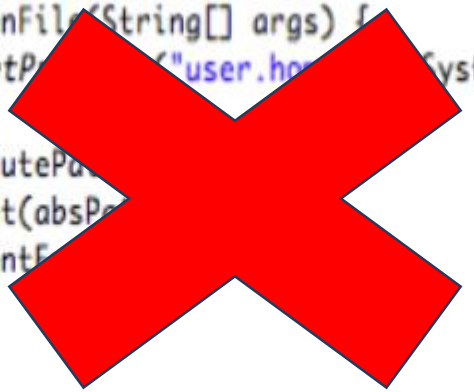
```
public static void badSecureOpenFile(String[] args) {
    File f = new File(System.getProperty("user.home") + System.getProperty("file.separator") + args[0]);

    String absPath = f.getAbsolutePath();
    if (!isInSecureDir(Paths.get(absPath))) {
        throw new IllegalArgumentException();
    }
    if (!validate(absPath)) {
        // Validation
        throw new IllegalArgumentException();
    }
}
```

What is wrong here ?

Secure Coding principles

FileSystem Access: path manipulation

```
public static void badSecureOpenFil(String[] args) {
    File f = new File(System.getP      ("user.ho      ystem.getProperty("file.separator") + args[0]);

    String absPath = f.getAbsolutePa
    if (!isInSecureDir(Paths.get(absP
        throw new IllegalArgumentE
    }
    if (!validate(absPath)) {
        // Validation
        throw new IllegalArgumentException();
    }
}
```



Secure Coding principles

FileSystem Access: path manipulation

```
public static void goodSecureOpenFile (String[] args) throws IOException {
    File f = new File(System.getProperty("user.home") + System.getProperty("file.separator")+ args[0]);

    String canonicalPath = f.getCanonicalPath();
    if (!isInSecureDir(Paths.get(canonicalPath))) {
        throw new IllegalArgumentException();
    }
    if (!validate(canonicalPath)) {
        // Validation
        throw new IllegalArgumentException();
    }
}
```

Secure Coding principles: Crypto

Secure Coding principles

Crypto

- Use TLS v1.2 to secure data transport
- Use certificate revocation lists (CRL) and OCSP server to validate certificates
- Use AES 256 for data ciphering
- Do not become a crypto expert ☺ use recommandation from sources recognized in security community:
  - BetterCrypto project : https://bettercrypto.org/
  - Mozilla: https://wiki.mozilla.org/Security/Server_Side_TLS
- Use client certificates with TLS mutual authentication to authenticate both partners of a communication
- When applicable use certificates and public key pinning
- Failed HTTPS should stop:
  - no certificate validation bypass
  - No fallback to HTTP
- Use HTTP Strict-Transport-Security-Header

Secure Coding principles

Crypto

- Crypto issue example

Source code

Secure Coding principles: Enable defensive measure in modern browsers

Secure Coding principles

Defensive feature in modern browsers

- Modern browsers supports a set of special HTTP response headers that change the way in which the browser process received content.
- Interesting headers are:
  - "*X-Frame-Options*": Indicate whether or not a browser should be allowed to render a page in a <frame>, <iframe> or <object>
  - "*X-XSS-Protection*": Enables the Cross-site scripting (XSS) filter built into most recent web browsers
  - "*X-Content-Type-Options*": Prevents Internet Explorer and Google Chrome from MIME-sniffing a response away from the declared content-type
  - "*Content-Security-Policy*": Define a policy used by the browser to know how it must render the current response
  - "*Strict-Transport-Security*": Enforces secure (HTTP over TLS) connections to the server

Secure Coding principles

Defensive feature in modern browsers

URL	Status	Domain	Size	Remote IP	Timeline
GET www.facebook.com	200 OK	facebook.com	14,5 KB	31.13.90.65:443	280ms

Headers | Response | HTML | Cache | Cookies

Response Headers view source

Cache-Control	private, no-cache, no-store, must-revalidate
Content-Encoding	gzip
Content-Security-Policy	default-src *;script-src https://*.facebook.com http://*.facebook.com https://*.fbcdn.net http://*.fbcdn .net *.facebook.net *.google-analytics.com *.virtualearth.net *.google.com 127.0.0.1:* *.spotilocal.com :* 'unsafe-inline' 'unsafe-eval' https://*.akamaihd.net http://*.akamaihd.net *.atlassolutions.com;style-src * 'unsafe-inline';connect-src https://*.facebook.com http://*.facebook.com https://*.fbcdn.net http ://*.fbcdn.net *.facebook.net *.spotilocal.com:* https://*.akamaihd.net wss://*.facebook.com:* ws:// *.facebook.com:* http://*.akamaihd.net https://fb.scanandcleanlocal.com:* *.atlassolutions.com http: //attachment.fbsbx.com https://attachment.fbsbx.com;
Content-Type	text/html; charset=utf-8
Date	Tue, 23 Dec 2014 05:41:05 GMT
Expires	Sat, 01 Jan 2000 00:00:00 GMT
P3P	CP="Facebook does not have a P3P policy. Learn why here: http://fb.me/p3p"
Pragma	no-cache
Set-Cookie	dpr=deleted; expires=Thu, 01-Jan-1970 00:00:01 GMT; Max-Age=-1419313264; path=/; domain=.facebook.com ; httponly wd=deleted; expires=Thu, 01-Jan-1970 00:00:01 GMT; Max-Age=-1419313264; path=/; domain=.facebook.com ; httponly reg_ext_ref=deleted; expires=Thu, 01-Jan-1970 00:00:01 GMT; Max-Age=0; path=/; domain=.facebook.com
Strict-Transport-Security	max-age=15552000; preload
X-Content-Type-Options	nosniff
X-Firefox-Spdy	3.1
X-Frame-Options	DENY
X-XSS-Protection	0
x-fb-debug	2DWesZoyCywZCIt2omRFR7hkemqknumW6lFzvr+I2JACem8FdDsY4YYq4x41Vpr3I4KNx6q4i6nPNg3mrBvDeg==

Facebook disable browser XSS protection filter ☹

Secure Coding principles

Defensive feature in modern browsers

- Framework like Spring Security provide configuration to automatically add somes of the presented headers into all HTTP responses

```
01 <beans xmlns="http://www.springframework.org/schema/beans"
02        xmlns:security="http://www.springframework.org/schema/security"
03        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
04     xsi:schemaLocation="http://www.springframework.org/schema/beans
05           http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
06           http://www.springframework.org/schema/security
07           http://www.springframework.org/schema/security/spring-security.xsd">
08
09     <security:http>
10         <!-- Add response headers to:
11              - Protect against Clickjacking
12              - Enable XSS protection on browser
13              - Disable content sniffing by IE/Chrome
14              - Define a Content Security Policy
15          -->
16         <security:headers>
17             <security:frame-options policy="DENY" />
18             <security:xss-protection enabled="true" block="true" />
19             <security:content-type-options />
20             <security:header name="Content-Security-Policy" value="default-src 'self'" />
21         </security:headers>
22     </security:http>
23
24 </beans>
25
```

Secure Coding principles

Defensive feature in modern browsers

- If your web framework do not have this type of feature then a simple JEE web Filter can do the job ☺

```
/**
 * Filter used to add browser security headers<br>
 * to all HTTP responses returned by the application
 *
 * @author Dominique Righetto
 *
 */
@WebFilter("/*")
public class BrowserSecurityHeadersFilter implements Filter {

    /**
     * {@inheritDoc}
     *
     * @see javax.servlet.Filter#doFilter(javax.servlet.ServletRequest, javax.servlet.ServletResponse, javax.servlet.FilterChain)
     */
    @Override
    public void doFilter(ServletRequest req, ServletResponse resp, FilterChain chain) throws IOException, ServletException {

        /* Let request reach target */
        chain.doFilter(req, resp);

        /* Add headers */
        HttpServletResponse httpResp = (HttpServletResponse) resp;

        // Protect against Clickjacking
        httpResp.setHeader("X-Frame-Options", "Deny");

        // Enable XSS protection on browser
        httpResp.setHeader("X-XSS-Protection", "1; mode=block");

        // Disable content sniffing by IE/Chrome
        httpResp.setHeader("X-Content-Type-Options", "nosniff");

        // Define a Content Security Policy
        httpResp.setHeader("Content-Security-Policy", "default-src 'self'");

    }
```

Secure Coding principles

Defensive feature in modern browsers

- Enabling theses headers is not a « silver bullet » but it helps to add another layer of defense
- Contribute to « Defense in depth » approach !

Interesting links

Interesting links

- OWASP API:
  - https://github.com/ESAPI/esapi-java
  - https://github.com/owasp/java-html-sanitizer
- Maven plugin to check dependencies for known vulnerabilities:
  - https://github.com/jeremylong/DependencyCheck
- OWASP Guides:
  - http://www.lulu.com/spotlight/owasp
- Password cracking:
  - https://www.owasp.org/images/a/af/2011-Supercharged-Slides-Redman-OWASP-Feb.pdf
  - http://resources.infosecinstitute.com/password-cracking-evolution/


ANY
QUESTIONS
?