Welcome to the OWASP TOP 10

Secure Development for Java Developers

Dominik Schadow

03/20/2012

BASEL BERN LAUSANNE ZÜRICH DÜSSELDORF FRANKFURT A.M. FREIBURG I.BR. HAMBURG MÜNCHEN STUTTGART WIEN

2012 © Trivadis

trivadis makes IT easier.

AGENDA

1. OWASP and the top 10 project
2. The top 10 in detail – more or less
3. Are we there yet?

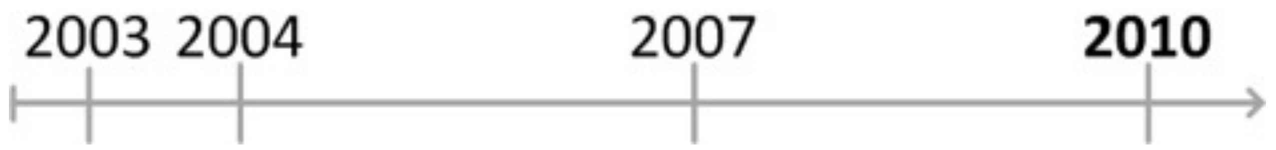
We need way more secure software

- Every developer needs secure programming know how
  - Project leads/ architects need security requirements awareness
- Applications must be protected from the beginning
  - A security fix does not bring back stolen data
  - The problem may be caused by the architecture
    - Not fixable with a couple of simple code changes
- 100% secure software will never exist
  - But we can stop making it that easy for attackers
  - Secure software is not developed accidentally

About the Open Web Application Security Project (OWASP)

- Not-for-profit worldwide charitable organization since 2001
- Improves the security of (web) application software
- All material is available for free
  - Tools and processes
    - The OWASP Enterprise Security API (ESAPI)
    - Comprehensive, Lightweight Application Security Process
    - Application Security Verification Standard
  - Documentation
    - Cheat Sheets to avoid most of the top 10 risks
    - Development Guide

The OWASP TOP 10 project



- Lists the 10 most critical web application security risks
  - Focus changed from weaknesses/ vulnerabilities to risks in 2010
  - Consider the top 10 list as a starter
- There are more than 10 risks for web applications out there
  - Focus on secure development first and train your developers
  - Document secure coding conventions
  - Think about a Software Development Lifecycle (SDLC) later

The Enterprise Security API (ESAPI)

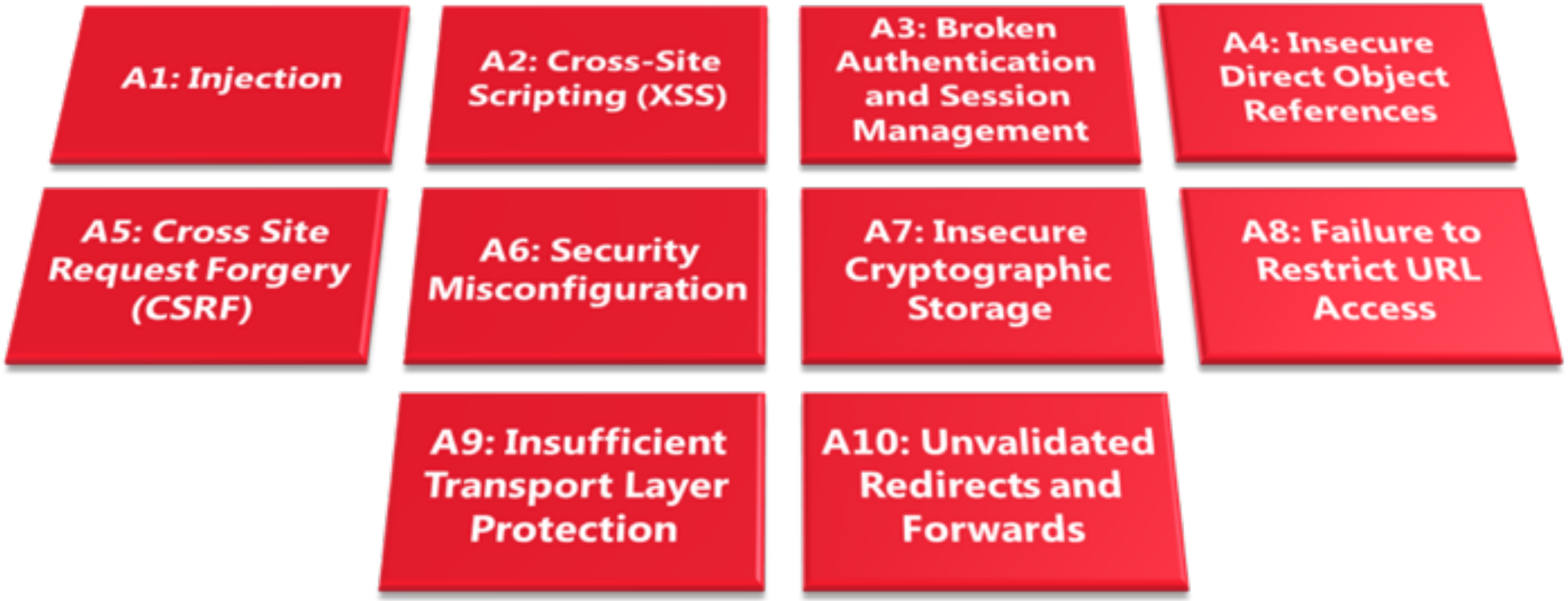
- Addresses the OWASP Top 10 risks
  - Good Java library, but project is not really active
- Easy to use open source web application security library
  - Collection of security building blocks, not a framework
  - Centralized access to all security related functionality
    - One access point for all security functionality
    - Much easier for developers
- Provides authentication, access control, input validation, output escaping, encryption, random numbers, ...

https://www.owasp.org/index.php/Category:OWASP_Enterprise_Security_API

AGENDA

1. OWASP and the top 10 project
2. The top 10 in detail – more or less
3. Are we there yet?

Top 10 2010



A1: Injection

A2: Cross-Site Scripting (XSS)

A3: Broken Authentication and Session Management

A4: Insecure Direct Object References

A5: Cross Site Request Forgery (CSRF)

A6: Security Misconfiguration

A7: Insecure Cryptographic Storage

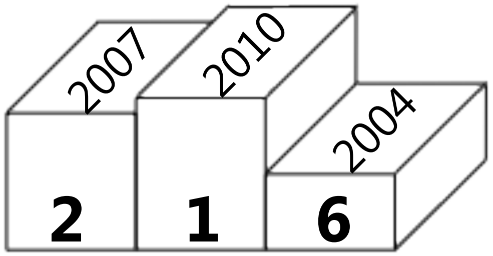
A8: Failure to Restrict URL Access

A9: Insufficient Transport Layer Protection

A10: Unvalidated Redirects and Forwards

Source http://owasptop10.googlecode.com/files/OWASP_Top_10_-_2010%20Presentation.pptx

A1 – Injection



- The famous (and least necessary) **SQL injection**
  - Simple to avoid with prepared statements
  - Use an OR-Mapper like Hibernate, myBatis, …
    - Even Spring JDBCTemplate provides protection
    - Dynamic queries may still be misused and made vulnerable
  - Limit the database user permissions
- **Other injections** (like LDAP injection, XPath injection)
  - **White list validation** for all user supplied input

Always validate in
front- and backend

A2 – Cross Site Scripting (XSS)

- Execute code in victim’s browser
  - Steal users’ session, sensitive data, redirect to phishing sites, …
- Attacker can take over control of victim’s browser
  - **XSS proxy**
    - Observe and control victim’s browser
- Different XSS types
  - Stored
  - Reflected
  - DOM based

The well-known XSS attacks – stored and reflected

- Stored
  - Injected code stored permanently on target servers
    - Databases, forum, guestbook, comment field
  - Victim retrieves malicious code (script) during visit
- Reflected
  - Injected code is reflected off the web server
    - Search results, error messages, or other response which contains (parts of) the input

1. Victim clicks manipulated (email) link
2. Link injects code on vulnerable server (page)
3. Server reflects the attack back to the browser
4. Browser executes the code (coming from the same server)

The lesser-known DOM based XSS attack

- HTTP response from server does not contain attacker's payload
  - Reflected and Stored XSS are server side execution issues
  - DOM based XSS is a client side execution issue

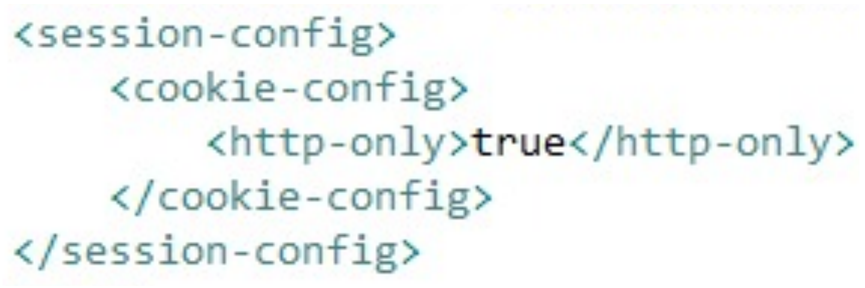
1. Victim clicks on link with malicious code (script)
2. Browser sends a request
3. Server responds with the page containing the malicious script
4. Browser creates the DOM object for the page, renders it and executes the attacker's script

A2 – Cross Site Scripting (XSS) (cont'd.)

- Every time an application accepts user input
  - **Validate** all user supplied input with a white list
  - Output **encode** all user supplied input

Always validate in front- and backend

- **Basic protection for reflective XSS** in browser
  - Internet Explorer 8 detects some patterns (X-XSS-Protection)
  - Firefox (NoScript), Chrome/ Safari (WebKit)
- Prevent scripts from accessing cookie: **http-only**



```
<session-config>
    <cookie-config>
        <http-only>true</http-only>
    </cookie-config>
</session-config>
```

A3 – Broken Authentication and Session Management

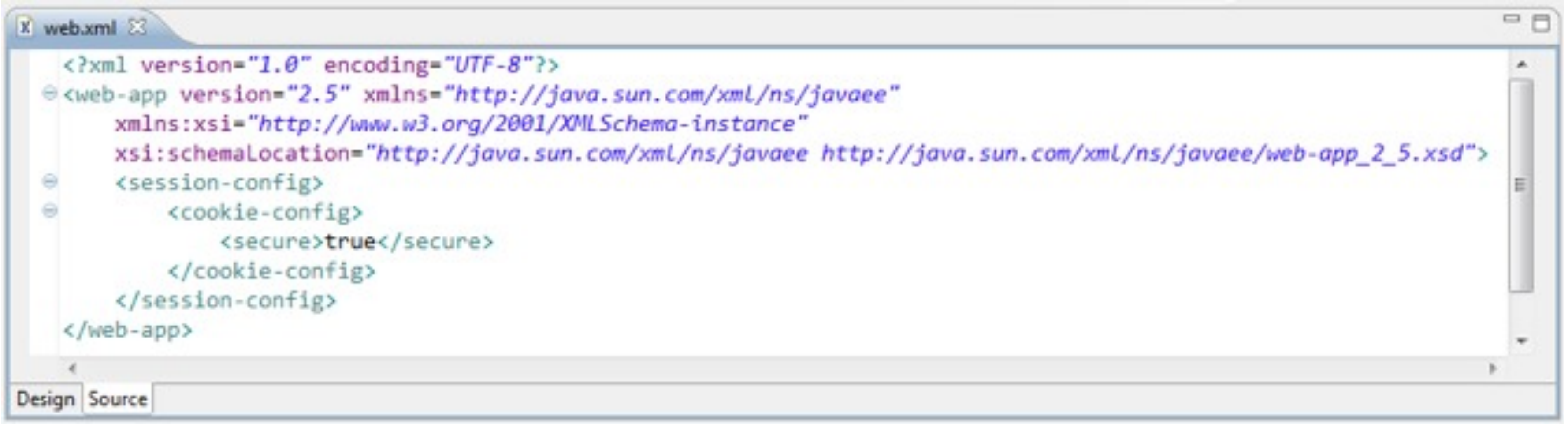
- One of the most complicated parts to develop
  - Simply: Don't invent it again, use existing frameworks
    - Spring Security http://static.springsource.org/spring-security/site
    - Apache Shiro http://shiro.apache.org
  - Centralize in one place and reuse code application wide
    - Try to use one library only
    - Know exactly how to use it
- Remember to store passwords encrypted or hashed
  - See **A7 (Insecure Cryptographic Storage)**
- HTTP is a stateless protocol
  - Credentials (session id) are included in every request

A3 – Broken Authentication and Session Management (cont'd.)

- Protect all connections with authentication data with TLS
  - Session id and credentials must be protected at all times
    - Unprotected connection does expose the session id
    - Session id is as valuable as username and password
  - Don't include any session information (like session id) in URLs
    - Shows up in the referrer logs on the next page
    - Included in copied links (send via email, twitter, …)
- Use the standard session id provided by your container
  - Configure a session timeout
  - Make sure that a logoff actually destroys the session
  - Set the **no-cache control header**
    - https should normally not be cached, but anyway…

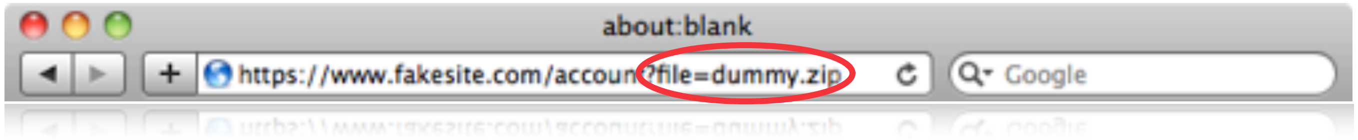
Set the secure-flag to protect the (session) cookie

- Browser may not send (session) cookie unprotected
  - Set **secure** to **true** in web.xml
    - Combine it with **http-only**
  - This is only one part in securing your session management



```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
    <session-config>
        <cookie-config>
            <secure>true</secure>
        </cookie-config>
    </session-config>
</web-app>
```

A4 – Insecure Direct Object References



- Presentation layer access control
  - User notices a direct reference in the URL
    - e.g. a file, account, database record, …
  - No enforcement of these restrictions on server side

1. User 57894 logs in with username/ password
   URL is https://www.myfakewebsite.com/account?**no=57894**
2. User experiments with URL *no* parameter, e.g. 57895
   URL is https://www.myfakewebsite.com/account?**no=57895**
3. User can view/ change other accounts

trivadis makes IT easier.

Reference map samples with ESAPI

```
private Set<Object> fileSet;
private File fileA = new File("/temp/dummyA.txt");
private File fileB = new File("/temp/dummyB.txt");
private File fileC = new File("/temp/dummyC.txt");
private File fileD = new File("/temp/dummyD.txt");

public FileService() {
    fileSet = new HashSet<Object>();

    fileSet.add(fileA);
    fileSet.add(fileB);
    fileSet.add(fileC);
    fileSet.add(fileD);
}
```

```
public void accessMap() throws AccessControlException {
    IntegerAccessReferenceMap map = new IntegerAccessReferenceMap(fileSet);
    String indRef = map.getIndirectReference(fileB);

    System.out.println("indRef " + indRef);

    String mapRef = indRef; // e.g. accessed via request parameter
    File file = (File) map.getDirectReference(mapRef);

    System.out.println("file " + file.getAbsolutePath());
}
```

```
indRef 3
file C:\temp\dummyB.txt
```

```
public void accessRandomMap() throws AccessControlException {
    RandomAccessReferenceMap map = new RandomAccessReferenceMap(fileSet);
    String indRef = map.getIndirectReference(fileA);

    System.out.println("indRef " + indRef);

    String mapRef = indRef; // e.g. accessed via request parameter
    File file = (File) map.getDirectReference(mapRef);

    System.out.println("file " + file.getAbsolutePath());
}
```

```
indRef hUDXFM
file C:\temp\dummyA.txt
```

A4 – Insecure Direct Object References (cont'd.)

- Replace the direct object references with an **access reference map** (indirect object references)
  - Replace account no with *no=1*, *no=2*, … for the logged in user
  - Mapping reference <-> real object on server **for this user**
    - Map is stored somewhere safe, e.g. session
  - No way for an attacker to break out
    - Using no=100 results in an error
    - Only resources in this map are accessible
- Use random numbers for more protection
- Useable for files, database records, accounts, …

ESAPI only

A5 – Cross Site Request Forgery (CSRF)

- Victim's browser is tricked into issuing commands to a vulnerable web application
  - Most credentials are automatically submitted by browser after logging in
  - Attacker creates a request (caused by image, form or script), forces browser **with authenticated user** to send credentials

1. Attacker prepares website with hidden img tag containing attack (script) against vulnerable site
2. While logged into vulnerable site, victim visits attacker's site
3. The img tag forces the browser to send request with credentials and malicious script to vulnerable site
4. Vulnerable site processes the authorized request

A5 – Cross Site Request Forgery (CSRF) (cont'd.)

- Calculate a **random secret token** (at beginning of session)
  - May not be automatically submitted like session cookie
  - Don't forget to check the token
  - Switch from get to post requests
- Add this token to **all forms and links (all requests)**
  - Hidden field (recommended)
    ```
    <input name="token" value="abekdil873843944"
    type="hidden"/>
    ```
  - Form token
    ```
    /accounts?auth=abekdil873843944
    ```

A6 – Security Misconfiguration

- Not really the developer's job
  - Patches for app-/web-server, databases, operating system, ...
  - App-/web-server/ database configuration, firewall, user rights
    - Turn off unnecessary features, disable ports, services, ...
    - **Work together with administrators**: Inform them about your needs (document them), about available (Java) patches, ...
- Developer's job
  - Configure logging, exception handling
    - No technical errors in frontend
    - Never serve log over web application in a production environment
  - Framework security configuration
    - Security related settings in all used frameworks
    - Security updates, new library versions

DevOps

A7 – Insecure Cryptographic Storage

- Most of the time, the problem is not the algorithm
  - The data isn't protected at all
  - The real threats are not identified
    - DB encryption protects data from DBA/ stolen disks, not SQL injection
- Identify and protect **all** sensitive data
  - Identify all places where this data gets stored
    - Make sure the data is protected in all locations
  - Never log any sensitive data
- Store key(s) and data in different locations
  - Prepare key exchange/ revocation
  - Change the keys periodically

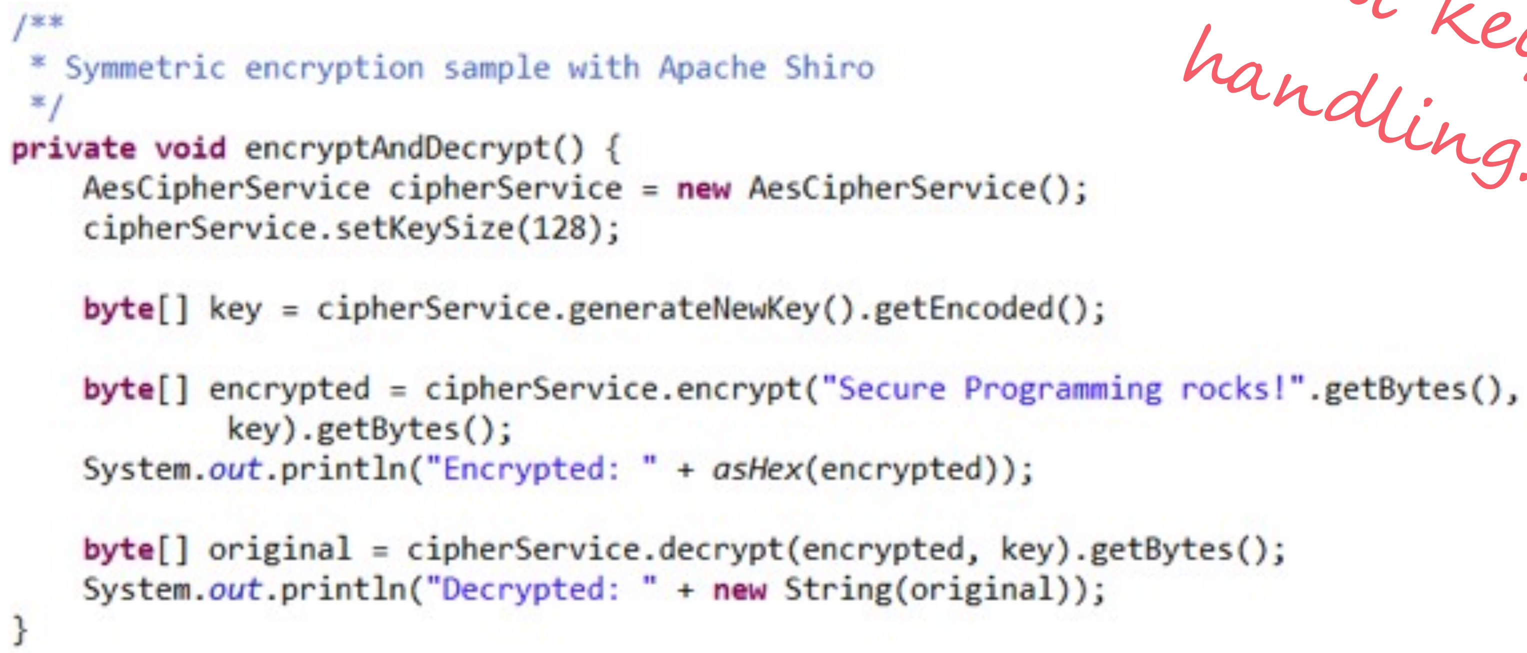
How do I select a strong algorithm?

- Never invent your own algorithms
- There is more than just the algorithm name
  - Size, padding, mode, and don't forget the salt
    - Symmetric
      AES/CBC/PKCS5Padding with 192 bit, Blowfish, 3DES
    - Asymmetric
      RSA, DSA with > 1024 bit
    - Hash
      SHA-256, RIPEMD-160
- Follow the news, replace broken or weak algorithms

if in doubt, choose the stronger key (negative impact on performance)

Encryption does not have to be complicated…

But key handling…

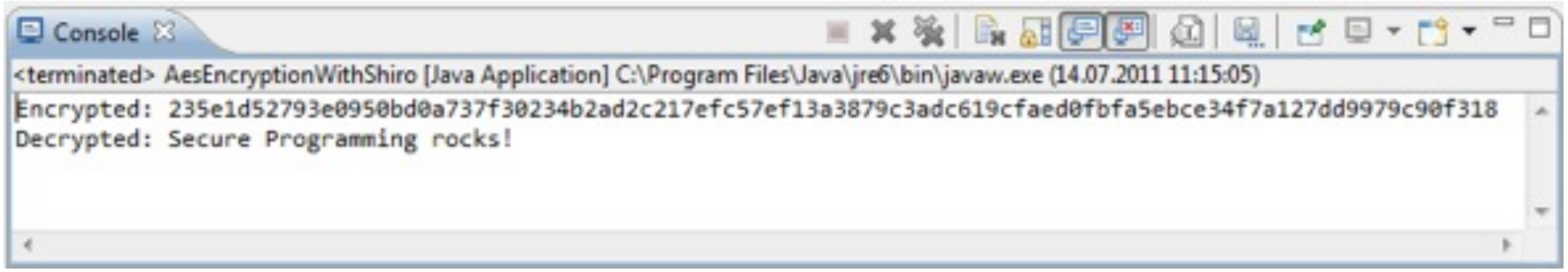


```java
/**
 * Symmetric encryption sample with Apache Shiro
 */
private void encryptAndDecrypt() {
    AesCipherService cipherService = new AesCipherService();
    cipherService.setKeySize(128);

    byte[] key = cipherService.generateNewKey().getEncoded();

    byte[] encrypted = cipherService.encrypt("Secure Programming rocks!".getBytes(),
            key).getBytes();
    System.out.println("Encrypted: " + asHex(encrypted));

    byte[] original = cipherService.decrypt(encrypted, key).getBytes();
    System.out.println("Decrypted: " + new String(original));
}
```



```
<terminated> AesEncryptionWithShiro [Java Application] C:\Program Files\Java\jre6\bin\javaw.exe (14.07.2011 11:15:05)
Encrypted: 235e1d52793e0950bd0a737f30234b2ad2c217efc57ef13a3879c3adc619cfaed0fbfa5ebce34f7a127dd9979c90f318
Decrypted: Secure Programming rocks!
```

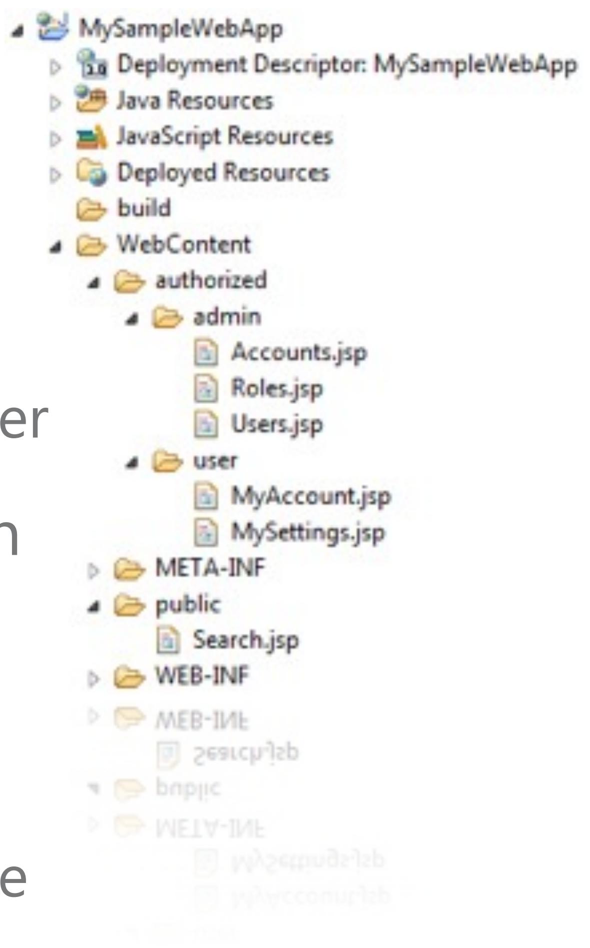
A8 – Failure to Restrict URL Access

- Presentation layer access control
  - GUI only shows authorized buttons/ links/ …
  - User notices his role in the URL and changes it
    - e.g. user, editor, admin, …
  - No enforcement of these restrictions on server side

1. User 57894 logs in with username/ password
   URL is https://www.myfakewebsite.com/**user**/account
2. User experiments with role part in URL, e.g. admin
   URL is https://www.myfakewebsite.com/**admin**/account
3. User has access to other accounts

A8 – Failure to Restrict URL Access (cont'd.)

- Enforce all restrictions on server side
  - Access for authorized users only
- **Think about roles from the beginning**
  - Store view files (JSP, JSF, …) in different folders based on their roles
  - Makes role/ filter configuration much easier
- Avoid combining user and admin roles in one application
  - Public application with user role only accessible via internet
  - Separate admin application only accessible in the intranet



```
MySampleWebApp
  Deployment Descriptor: MySampleWebApp
  Java Resources
  JavaScript Resources
  Deployed Resources
  build
  WebContent
    authorized
      admin
        Accounts.jsp
        Roles.jsp
        Users.jsp
      user
        MyAccount.jsp
        MySettings.jsp
    META-INF
    public
      Search.jsp
    WEB-INF
```

A9 – Insufficient Transport Layer Protection

- Identify all routes where this data is broadcasted
- Select the appropriate protection mechanism
  - Transport based security with SSL/TLS
  - Information based security
    - e.g. XML Encryption and XML Signatures
- Protect all ~~(or nothing)~~
  - Don't mix protected with unprotected content
  - Secure the input form with log-in credentials
  - Secure the (session) cookie

less vulnerable for Man-in-the-Middle attacks

- Remember the **Insecure Crypto Storage** recommendations

Some Secure Sockets Layer and Transport Layer Security basics

- **SSL v2** is insecure and must not be used
  - Disable it
- **SSL v3** and **TLS v1.0** are most common
  - Do not have any major security flaws up to now
  - TLS v1.0 is sometimes referred to as SSL v3.1
- **TLS v1.1** and **TLS v1.2** are the best selection
  - Do not have any security flaws up to now
  - Widely unsupported, choose in case the server supports it
    - Older clients will automatically fall back to TLS v1.0

Use SSL Server Test on https://www.ssllabs.com

trivadis makes IT easier.

Set the HTTP Strict Transport Security (HSTS) header

- HTTP Strict Transport Security is currently an IETF draft
- Application forces browser to only use HTTPS when visiting
  - For specified time, renewed with every response
  - Browser should not send data if communication is insecure
    - Invalid certificate results into error page, not a strange certificate warning dialog
- Browser support required, no backwards compatibility issues
  - Supported in Firefox and Chrome

```
HttpServletResponse response ...;
response.setHeader("Strict-Transport-Security",
  "max-age=8640000; includeSubdomains");
```

http://tools.ietf.org/html/draft-ietf-websec-strict-transport-sec

A10 – Unvalidated Redirects and Forwards

- **Redirects** may include user supplied parameters in destination URL
  - Phishing and malware installation
    - Victim thinks he is still on the save company/ bank/ … page
    - But attacker can send victim to any page
- **Forwards** send request to a new page in same application
  - May process parameters too
  - Attacker may bypass authentication/ authorization checks

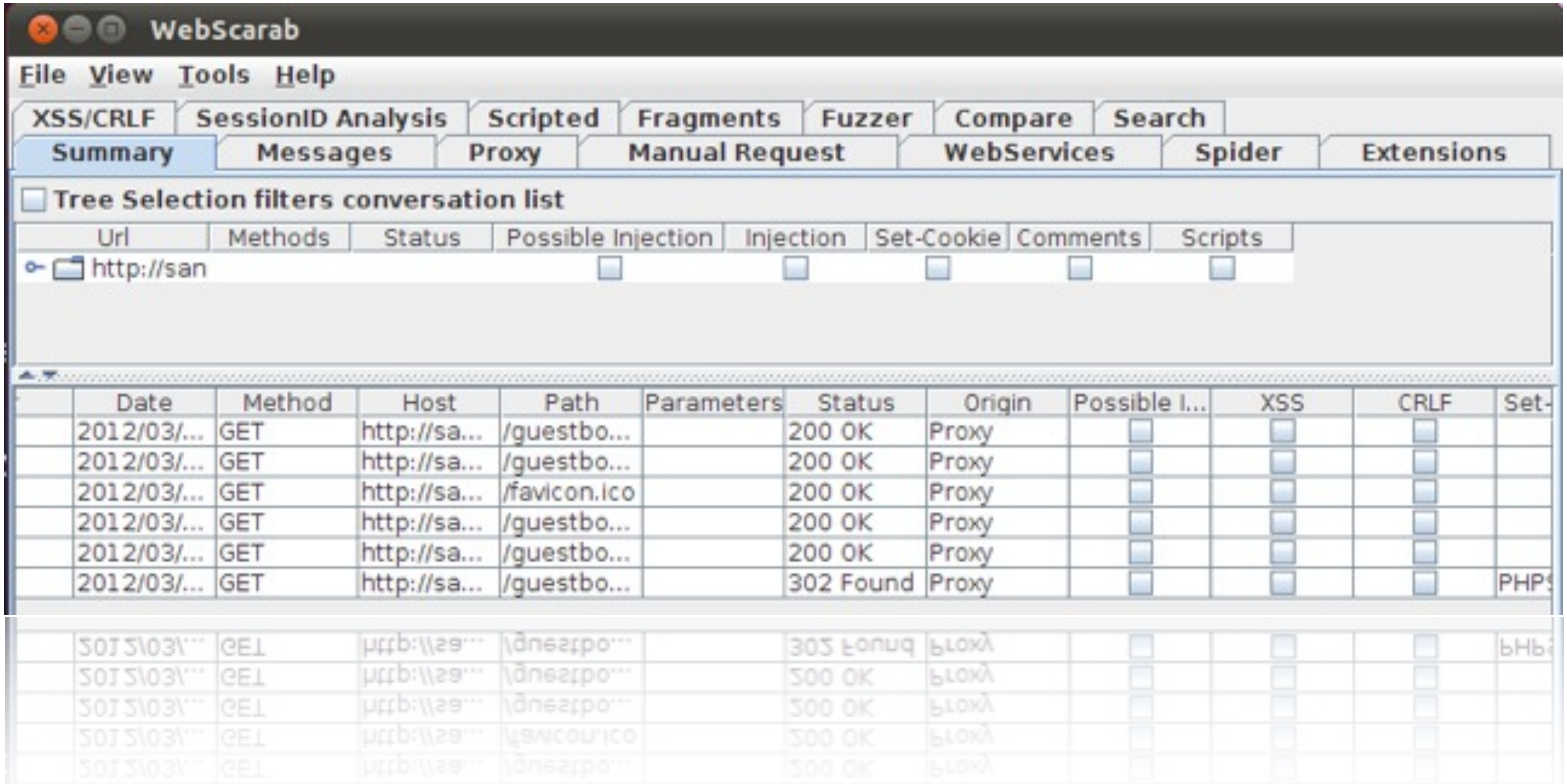
A10 – Unvalidated Redirects and Forwards (cont'd.)

- Avoid redirects and forwards wherever possible
- Don't allow user parameters for the target URL
- In case you need parameters in the target URL
  - Use a server side mapping to translate the values shown to the user into valid URL parts
  - Validate the final target URL
- Call the access controller for any forward

AGENDA

1. OWASP and the top 10 project
2. The top 10 in detail – more or less
3. Are we there yet?

Use tools to examine/ manipulate your web application (data)


WebScarab
File View Tools Help
XSS/CRLF | SessionID Analysis | Scripted | Fragments | Fuzzer | Compare | Search
Summary | Messages | Proxy | Manual Request | WebServices | Spider | Extensions
Tree Selection filters conversation list

Url	Methods	Status	Possible Injection	Injection	Set-Cookie	Comments	Scripts
http://san			☐	☐	☐	☐	☐

	Date	Method	Host	Path	Parameters	Status	Origin	Possible I...	XSS	CRLF	Set-
	2012/03/...	GET	http://sa...	/guestbo...		200 OK	Proxy	☐	☐	☐	
	2012/03/...	GET	http://sa...	/guestbo...		200 OK	Proxy	☐	☐	☐	
	2012/03/...	GET	http://sa...	/favicon.ico		200 OK	Proxy	☐	☐	☐	
	2012/03/...	GET	http://sa...	/guestbo...		200 OK	Proxy	☐	☐	☐	
	2012/03/...	GET	http://sa...	/guestbo...		200 OK	Proxy	☐	☐	☐	
	2012/03/...	GET	http://sa...	/guestbo...		302 Found	Proxy	☐	☐	☐	PHP

https://www.owasp.org/index.php/Category:OWASP_WebScarab_Project
with Firefox QuickProxy https://addons.mozilla.org/de/firefox/addon/quickproxy

There are many more tools available, use them (the hacker will...)

One security aware developer is not enough

- Developing with security awareness is a good start
  - Make sure the environment is configured properly
  - Inform the administrators about your requirements
- Design security in from the beginning
  - Think about security needs before starting to code
  - Much harder/ more expensive to secure an existing application

Security must be a natural part of the development process

THANK YOU.

Trivadis GmbH
Dominik Schadow

Industriestrasse 4
D-70565 Stuttgart

Phone +49-711-903 63 230
Fax +49-711-903 63 259

info@trivadis.com
www.trivadis.com

BASEL BERN LAUSANNE ZÜRICH DÜSSELDORF FRANKFURT A.M. FREIBURG I.BR. HAMBURG MÜNCHEN STUTTGART WIEN

trivadis
makes IT easier.

Resources

- OWASP www.owasp.org
  - Developer's Guide, Testing Guide, Code Review Guide
  - **Cheat Sheets**
- OWASP Guide Project https://www.owasp.org/index.php/Category:OWASP_Guide_Project
- ESAPI http://esapi.org
- Java Secure Coding Guidelines http://www.oracle.com/technetwork/java/seccodeguide-139067.html
- Qualys SSL Labs https://www.ssllabs.com

trivadis makes IT easier.