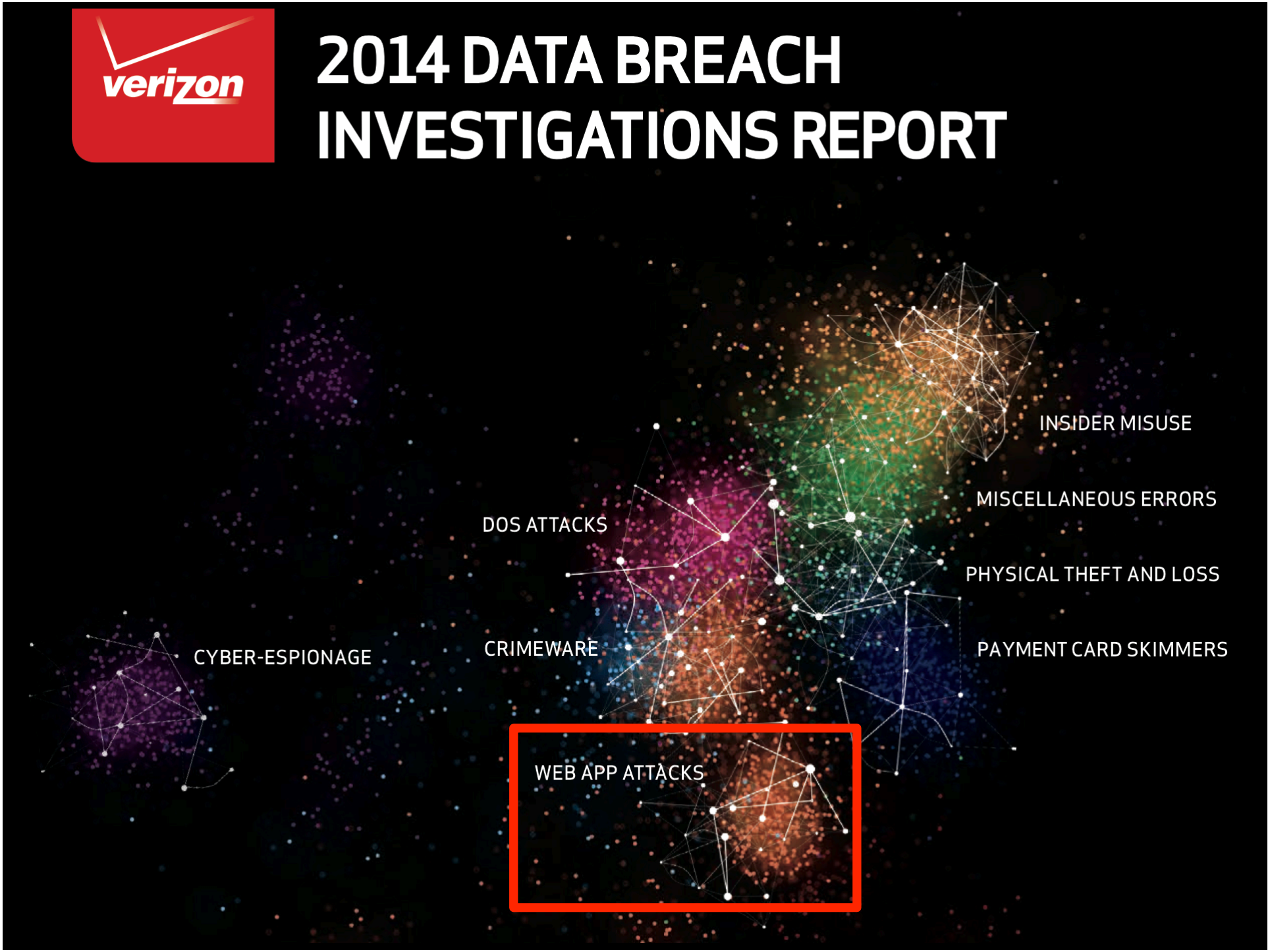
Five Keys for Securing Java Web Apps

JavaOne 2014

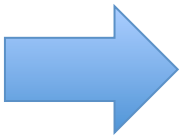
About

- Frank Kim
  - SANS Institute
    - Curriculum Lead, Application Security
    - Author, Secure Coding in Java

2014 DATA BREACH INVESTIGATIONS REPORT



Outline



XSS

- Encryption
- Access Control
- SQL Injection
- CSRF

Cross-Site Scripting (XSS)

- Occurs when malicious data is displayed by a web application
  - Untrusted code is injected into the application
  - This evil code is then rendered without encoding or validation
- Common types of XSS
  - Stored
  - Reflected

Content Security Policy

- Helps mitigate XSS attacks
  - By defining approved sources of content
- Loading resources from the same origin

```
Content-Security-Policy: default-src 'self'
```

- Other directives
  - default-src
  - script-src
  - object-src
  - style-src
  - img-src
  - media-src
  - frame-src
  - font-src
  - connect-src

CSP Examples

1) Only load resources from the same origin

```
X-Content-Security-Policy: default-src 'self'
```

2) Example from mikewest.org

```
x-content-security-policy:
   default-src 'none';
   style-src https://mikewestdotorg.hasacdn.net;
   frame-src
      https://www.youtube.com
      http://www.slideshare.net;
   script-src
      https://mikewestdotorg.hasacdn.net
      https://ssl.google-analytics.com;
   img-src 'self'
      https://mikewestdotorg.hasacdn.net
      https://ssl.google-analytics.com data:;
   font-src https://mikewestdotorg.hasacdn.net
```

Report Only

- Facebook Example

```
x-content-security-policy-report-only:
   allow *;
   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:*;
   options inline-script eval-script;
   report-uri https://www.facebook.com/csp.php
```

XSS Demo

Outline

- XSS
- Encryption
- Access Control
- SQL Injection
- CSRF

Configuring SSL in a Web App

- Enable SSL in web.xml

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

Secure Flag

- Ensures that the Cookie is only transmitted via SSL
- Configure in web.xml since Servlet 3.0

```
<session-config>
  <cookie-config>
    <secure>true</secure>
  </cookie-config>
</session-config>
```

- Programmatically

```
Cookie cookie = new Cookie("mycookie", "test");
cookie.setSecure(true);
```

HTTP Strict-Transport-Security (HSTS)

- Tells browser to only talk to the server via HTTPS
  - First time your site is accessed via HTTPS *and* the header is used the browser stores the certificate info
  - Subsequent requests to HTTP automatically use HTTPS
- Defined in RFC 6797
  - Implemented in Firefox and Chrome

```
Strict-Transport-Security: max-age=seconds
                [; includeSubdomains]
```

Outline

- XSS
- Encryption
- Access Control
- SQL Injection
- CSRF

Declarative Access Control

- Based on HTTP methods

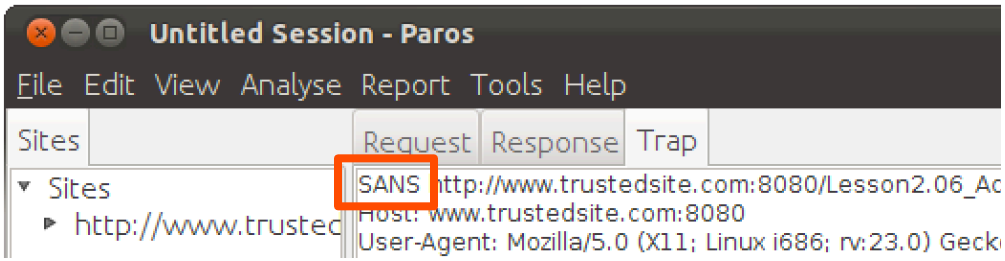
```
<security-constraint>
  <web-resource-collection>
    <url-pattern>/site/*</url-pattern>
    <http-method>GET</http-method>
    <http-method>POST</http-method>
  </web-resource-collection>
  ...
</security-constraint>
```

} Only these methods are protected

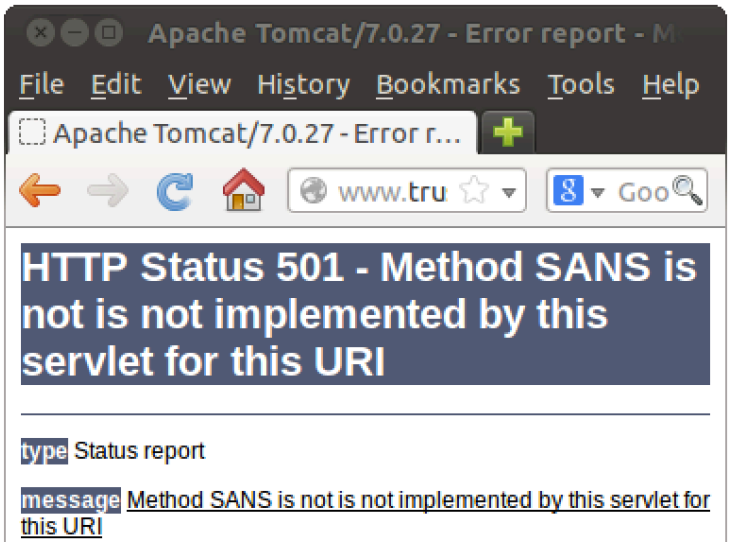
- Security Misconfiguration
  - OWASP Top Ten item

Using Arbitrary HTTP Methods

Change method to SANS



Going to /site/hello gives an error msg



Going to /site/hello.jsp displays the page!

Access Control Bypass Demo

How to Fix

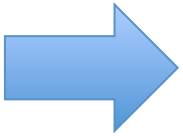
- Do not define any HTTP methods

```
<security-constraint>
  <web-resource-collection>
    <web-resource-name>SimpleAuth</web-resource-name>
    <url-pattern>/site/*</url-pattern>
  </web-resource-collection>
  ...
</security-constraint>
```

<http-method> elements have been removed

Outline

- XSS
- Encryption
- Access Control
- SQL Injection
- CSRF



Preventing SQL Injection

- Safe use of parameterized queries will prevent SQL Injection

```
SELECT * FROM USERS
WHERE ID = ? AND PWD = ?
```

PreparedStatement Gone Wrong

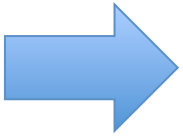
- What is wrong with this code?

```
String query = "SELECT * FROM users
  WHERE age > " + age + "
  AND gender = '" + gender + "'";
PreparedStatement stmt =
  con.prepareStatement(query);
ResultSet rs = stmt.executeQuery();
```

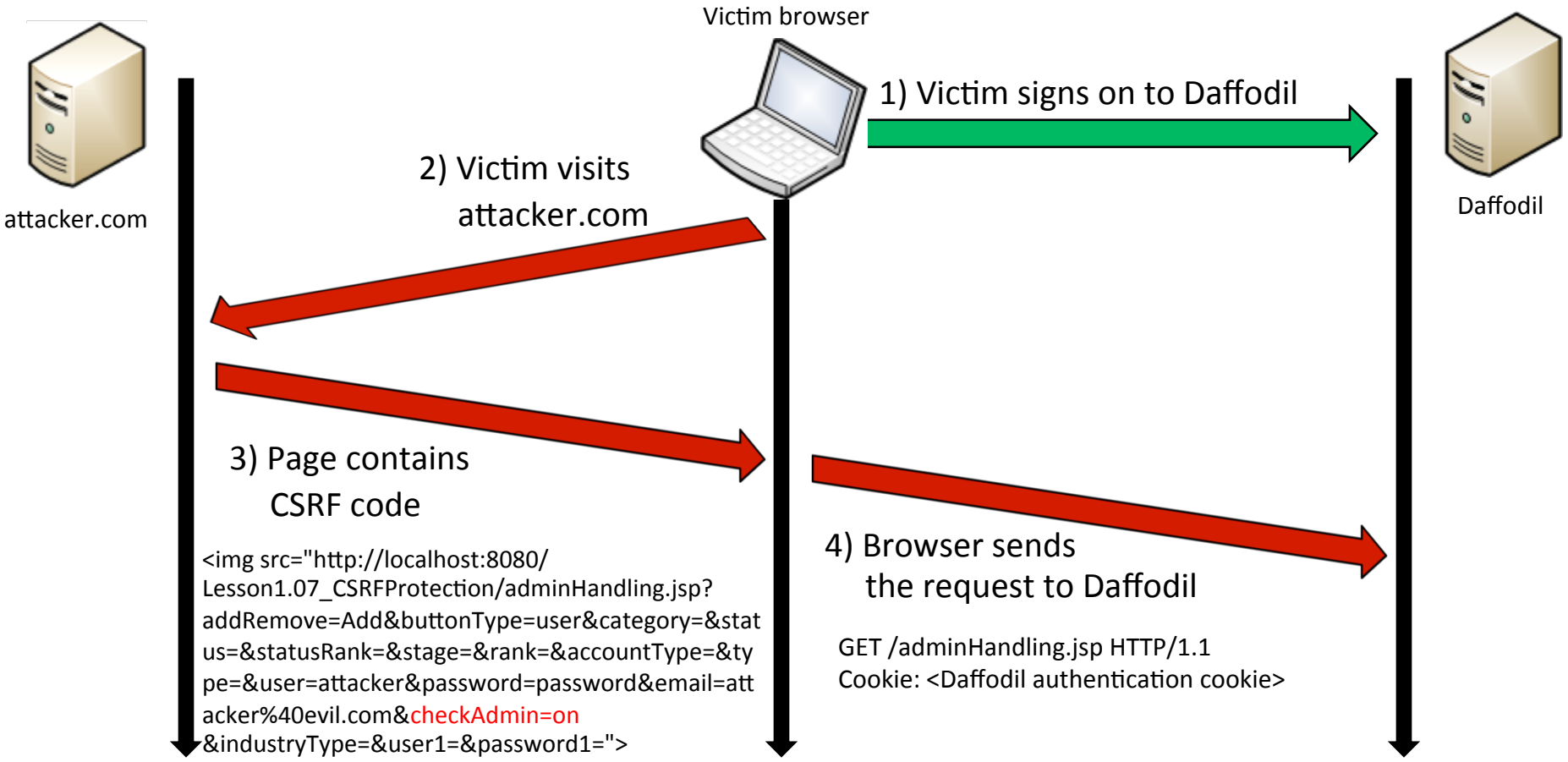
SQL Injection Demo

Outline

- XSS
- Encryption
- Access Control
- SQL Injection
- CSRF



CSRF Attack Overview



CSRF Demo

CSRF Defense

- Must include something random in the request
  - Use an anti-CSRF token
- OWASP CSRFGuard
  - Written by Eric Sheridan @eric_sheridan
  - Can inject anti-CSRF token using
    1) JavaScript DOM manipulation - for automated protection requiring minimal effort
    2) JSP Tag library - for manual, fine grained protection
  - Filter then intercepts request & validates token

CSRFGuard JSP Tags

- Tags for token name and value

```
<form name="test1" action="protect.html">
  <input type="text" name="text" value="text"/>
  <input type="submit" name="submit" value="submit"/>
  <input type="hidden" name="<csrf:token-name/>"
                       value="<csrf:token-value/>"/>
</form>
```

- Tag for name/value pair (delimited with "=")

```
<a href="protect.html?<csrf:token/>">protect.html</a>
```

- Convenience tags for forms and links as well

<csrf:form> and <csrf:a>

Examples from https://www.owasp.org/index.php/CSRFGuard_3_Token_Injection

CSRFGuard DOM Manipulation

- Include JavaScript in every page that needs CSRF protection

```
<script src="/securish/JavaScriptServlet"></script>
```

- JavaScript used to hook the open and send methods

```
XMLHttpRequest.prototype._open = XMLHttpRequest.prototype.open;
XMLHttpRequest.prototype.open = function(method, url, async, user, pass) {
  // store a copy of the target URL
  this.url = url;
  this._open.apply(this, arguments);
}

XMLHttpRequest.prototype._send = XMLHttpRequest.prototype.send;
XMLHttpRequest.prototype.send = function(data) {
  if(this.onsend != null) {
    // call custom onsend method to modify the request
    this.onsend.apply(this, arguments);
  }
  this._send.apply(this, arguments);
}
```

Protecting XHR Requests

- CSRFGuard sends two HTTP headers

```
XMLHttpRequest.prototype.onsend = function(data) {
  if(isValidUrl(this.url)) {
    this.setRequestHeader("X-Requested-With",
         "OWASP CSRFGuard Project")
    this.setRequestHeader("OWASP_CSRFTOKEN",
         "EDTF-U8O6-J91L-RZOW-4X09-KEXB-K9B3-4OIV");
  }
};
```

Summary

- XSS
- Encryption
- Access Control
- SQL Injection
- CSRF

SANS AppSec

APPLICATION & SOFTWARE SECURITY

Website

http://software-security.sans.org

Free resources, white papers, webcasts, and more

Blog

http://software-security.sans.org/blog

Twitter

@sansappsec

Latest news, promos, and other information

Secure Coding Assessment

http://software-security.sans.org/courses/assessment

SANS AppSec CURRICULUM

Core

- **STH.DEVELOPER** Application Security Awareness Modules — CBT
- **DEV522** Defending Web Applications Security Essentials — *GWEB*

Secure Coding

- **DEV541** Secure Coding in Java/JEE — *GSSP-JAVA*
- **DEV544** Secure Coding in .NET — *GSSP-.NET*
- **DEV543** Secure Coding in C/C++

Specialization

- **SEC542** Web App Penetration Testing and Ethical Hacking — *GWAPT*
- **SEC642** Advanced Web App Penetration Testing and Ethical Hacking

Frank Kim
fkim@sans.org

@sansappsec