Web security: OWASP project, CSRF threat and solutions.

Fabio Lombardi
About me

• Since 2013
  – Research & Development Engineer @Bonitasoft

• Past:
  – 2 years as Web Penetration Tester Consultant
  – Post-graduate studies in IT Security and Software engineering

BPM open source world leader
Security checklist

- Anti-SQL-injection protection
- SSL and OpenSSL up to date
- AES encryption on sensitive data
- Passwords hashed with salt
- Multi-factor authentication on the back-office
- Preventing the PM from sending the whole unencrypted database by email

CommitStrip.com
DID YOU KNOW?
In the US

$5,400,000
Average organization cost of a data breach

$277
Average cost per stolen record

~ 10,000
Cyber attacks attempts against the US Navy since this talk started

From a 2013 Fonemon institute study: Cost of Data Breach Study: Global Analysis
... and in Europe

From a 2013 Fonemon institute study: Cost of Data Breach Study: Global Analysis
Seeing the financial consequences of a security breach

How do the costs of a breach add up across six categories?

- 29% Reputation and brand damage
- 21% Lost productivity
- 19% Lost Revenue
- 12% Forensics
- 10% Technical support
- 8% Compliance/Regulatory

It’s not just about money...

- eBay: 233 million users
- Montana Department of Public Health & Human Services: 1 million people
- Orange: 800,000 users
- iCloud: Personal photos leakage
- pf.chang's China Bistro: Thousands credit cards
- SnapSave: 200,000 Snapchats photos
but why?
Security has to be taken into account
How to change things?

You are not alone.
The reference for web application security

OWASP
The Open Web Application Security Project
https://www.owasp.org

https://www.youtube.com/user/OWASPGLOBAL

LAPSE+ plugin for code source analysis
Example of a vulnerability description

Ref: https://www.owasp.org/index.php/Top_10_2013-A8-Cross-Site_Request_Forgery_(CSRF)
The OWASP Top 10 Application Security Risks

1. Injection
2. Cross-Site Scripting (XSS)
3. Broken Authentication and Session Management
4. Insecure Direct Object References
5. Cross-Site Request Forgery (CSRF)
6. Insecure Cryptographic Storage
7. Insufficient Transport Layer Protection
8. Failure to Restrict URL Access
9. Insecure Deserialization
10. Unvalidated Redirects and Forwards

May be outside the developer’s control

Ref: https://www.veracode.com/blog/2012/06/building-secure-web-applications-infographic
CSRF (Cross-Site request forgery)

- Not well known (w.r.t. XSS or SQL Injection)

- Impacts
  - Malicious money transfers
  - User creation
  - Privilege escalation
  - Compromise end user data
  - The entire web application can be compromised
  - ...
CSRF: The attack

User

My-site.com

Attacker web page

data
CSRF: The attack

1. The user logs-in to My-site.com

My-site.com

User

Attacker web page

The user logs in to My-site.com.
CSRF: The attack

1. The user logs-in to My-site.com
2. My-site.com sends back a session cookie

My-site.com

User

Attacker web page
CSRF: The attack

1. The user logs-in to My-site.com
2. My-site.com sends back a session cookie
3. The user is authorized to make API calls
CSRF: The attack

1. The user logs-in to My-site.com
2. My-site.com sends back a session cookie
3. The user is free to make API calls
4. The user simply browses the malicious web page

My-site.com

User

Attacker web page
CSRF: The attack

1. The user logs-in to My-site.com
2. My-site.com sends back a session cookie
3. The user is free to make API calls
4. The user simply browses the malicious web page
5. The malicious web page makes blind API calls on the user behalf on my-site.com
CSRF: The attack

1. The user logs-in to My-site.com
2. My-site.com sends back a session cookie
3. The user is free to make API calls
4. The user simply browses the malicious web page
5. The malicious web page makes **blind API calls** on the user behalf on my-site.com
6. Data is compromised
The Attacker web page

- Example:
  - A popular web site clone (phishing tools) + hidden:
    `<iframe src=./attack_payload.html width=0 height=0> </iframe>`

- attack_payload.html:

  ```html
  <form id="adduser" action="http://my-site.com/API/users/" method="POST" enctype="text/plain">
    <input name='{"password_confirm":"password","name":"newuser2","Firstname":"test2","password":"password", "ignore_me":""'} type='hidden'>
    <input type="submit" value="CSRF me !">  
  </form>
  <script type="text/JavaScript">document.getElementById("adduser").submit(); </script>
  ```

- action="http://my-site.com/API/users" works using the session cookie obtained from my-site.com page
  - Generates a **HTTP POST** call to my-site.com on user behalf
  - A new user is created
Why the attack works?

- It's due to a web browser feature

- The session is kept alive for multiple browser tabs
The attack weaknesses

- The attacker only attempts **blind attacks**
  - Cannot read the HTTP response
  - Cannot read the session cookie
CSRF: Solution

My-site.com

User

Attacker web page

©2014 Bonitasoft
CSRF: Solution

1. The user logs-in to My-site.com

User

Attacker web page

My-site.com

data
CSRF: Solution

1. The user logs-in to My-site.com

2. My-site.com sends back a session cookie + X-API-Token in the **response header**
CSRF: Solution

1. The user logs-in to My-site.com
2. My-site.com sends back a session cookie + X-API-Token in the response header
3. The user has to resend the X-API-Token in the request header of further API calls
CSRF: Solution

1. The user logs-in to My-site.com

2. My-site.com sends back a session cookie + X-API-Token in the response header

3. The user has to resend the X-API-Token in the request header of further API calls

4. The user simply browses the malicious web page
CSRF: Solution

1. The user logs-in to My-site.com
2. My-site.com sends back a session cookie + X-API-Token in the response header
3. The user has to resend the X-API-Token in the request header of further API calls
4. The user simply browses the malicious web page
5. The malicious web page blindly attempts to make the API call but without knowing the X-API-Token
CSRF: Solution

1. The User logs in My-site.com

2. My-site.com sends back a session cookie + X-API-Token in the response header

3. The user has to resend the X-API-Token in the request header of further API calls

4. The user simply browses the malicious web page

5. The malicious web page blindly attempts to make the API call but without knowing the X-API-Token

The call is rejected as unauthorized
Solution: server side

- Token generation

```java
public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain) throws IOException, ServletException {
    final HttpServletRequest req = (HttpServletRequest) request;
    final HttpServletResponse res = (HttpServletResponse) response;

    // Create
    Object apiTokenFromClient = req.getSession().getAttribute("api_token");
    if (apiTokenFromClient == null) {
        apiTokenFromClient = new APIToken().getToken();
        req.getSession().setAttribute("api_token", apiTokenFromClient);
    }

    res.addHeader("X-API-Token", apiTokenFromClient.toString());
    chain.doFilter(req, res);
}
```

- Token check

```java
boolean checkValidCondition(HttpServletRequest httpRequest, HttpServletResponse httpResponse) {
    String headerFromRequest = httpRequest.getHeader("X-API-Token");
    String apiToken = (String) httpRequest.getSession().getAttribute("api_token");

    if (headerFromRequest == null || !headerFromRequest.equals(apiToken)) {
        httpResponse.setStatusCode(HttpServletResponse.SC_UNAUTHORIZED);
        return false;
    }
    return true;
}
```
Solution: client side

- Session initialization

```java
protected void initSession(final Action callback) {
    @Override
    public void onSuccess(final int httpStatusCode, final String response, final Map<String, String> headers) {
        if (headers.get("X-API-Token") != null) {
            UserSessionVariables.addUserVariable(UserSessionVariables.API_TOKEN, headers.get("X-API-Token"));
        }
    }
}
```

- API Requests

```java
public void send(final Method method, final String url, final String datas, final String contentType, final HttpCallback callback) {
    final RequestBuilder builder = new RequestBuilder(method, url);
    if (datas != null) {
        builder.setRequestData(datas);
    }
    if (UserSessionVariables.getUserVariable(UserSessionVariables.API_TOKEN) != null) {
        builder.setHeader("X-API-Token", UserSessionVariables.getUserVariable(UserSessionVariables.API_TOKEN));
    }
    builder.setTimeoutMillis(30000);
    builder.setCallback(callback);
    Request request = null;
    try {
        request = builder.send();
    } catch (final RequestException e) {
        callback.onError(request, e);
    }
}
```
OWASP Eclipse plugin

Static code analysis for detecting the some OWASP vulnerabilities. Three steps:

Vulnerability Source, Vulnerability Sink and Provenance Tracker

https://www.owasp.org/index.php/OWASP_LAPSE_Project

https://code.google.com/p/lapse-plus/

connection = DriverManager.getConnection(DataURL, LOGIN, PASSWORD);  
} catch (SQLException e) {  
  // TODO Auto-generated catch block  
  e.printStackTrace();  
}

String Username = request.getParameter("USER"); // From HTTP request  
String Password = request.getParameter("PASSWORD"); // From HTTP request  
int UserID = -1;  
String loggedUser = "";  
String sel = "SELECT User_id, Username FROM USERS WHERE Username = " + Username + " AND Password = " + Password + "";  
Statement selectStatement = null;  
try {  
  selectStatement = (Statement) connection.createStatement();  
} catch (SQLException e) {  
  // TODO Auto-generated catch block  
}
Thank you

@lomba_fabio

fr.linkedin.com/pub/fabio-lombardi/27/833/660/