OWASP Top 10 – 2010
The Top 10 Most Critical Web Application Security Risks

Adrian Hayes
Security Consultant
Security-Assessment.com
Introduction

- OWASP Top 10 Project
  - "The OWASP Top Ten represents a broad consensus about what the most critical web application security flaws are."

- Why are we covering this?
  - Flaws 4, 5 and 6
  - What I see day to day during webapp assessments
  - Widely applicable to .nz businesses

- These slides are heavily entirely based on the work of others
  - See credits at the end
OWASP Top Ten (2010 Edition)

A1: Injection
A2: Cross-Site Request Forgery (CSRF)
A3: Broken Authentication and Session Management
A4: Insecure Direct Object References
A5: Cross-Site Scripting (XSS)
A6: Security Misconfiguration
A7: Failure to Restrict URL Access
A8: Insecure Cryptographic Storage
A9: Insufficient Transport Layer Protection
A10: Unvalidated Redirects and Forwards

http://www.owasp.org/index.php/Top_10
A4 – Insecure Direct Object References

How do you protect access to your data?

• This is part of enforcing proper “Authorization”, along with A7 – Failure to Restrict URL Access

A common mistake ...

• Only listing the ‘authorized’ objects for the current user, or
• Hiding the object references in hidden fields
• ... and then not enforcing these restrictions on the server side
• This is called presentation layer access control, and doesn’t work
• Attacker simply tampers with parameter value

Typical Impact

• Users are able to access unauthorized files or data
Insecure Direct Object References Illustrated

- Attacker notices his acct parameter is 6065?
  ```
  https://www.onlinebank.com/user?acct=6065
  ```
- He modifies it to a nearby number?
  ```
  https://www.onlinebank.com/user?acct=6066
  ```
- Attacker views the victim’s account information
A4 – Avoiding Insecure Direct Object References

- Eliminate the direct object reference
  - Replace them with a temporary mapping value (e.g. 1, 2, 3)
  - ESAPI provides support for numeric & random mappings
    - IntegerAccessReferenceMap & RandomAccessReferenceMap

- Validate the direct object reference
  - Verify the parameter value is properly formatted
  - Verify the user is allowed to access the target object
    - Query constraints work great!
  - Verify the requested mode of access is allowed to the target object (e.g., read, write, delete)
A5 – Cross Site Request Forgery (CSRF)

Cross Site Request Forgery

- An attack where the victim’s browser is tricked into issuing a command to a vulnerable web application
- Vulnerability is caused by browsers automatically including user authentication data (session ID, IP address, Windows domain credentials, ...) with each request

Imagine...

- What if a hacker could steer your mouse and get you to click on links in your online banking application?
- What could they make you do?

Typical Impact

- Initiate transactions (transfer funds, logout user, close account)
- Access sensitive data
- Change account details
CSRF Vulnerability Pattern

■ The Problem
  ▸ Web browsers automatically include most credentials with each request
  ▸ Even for requests caused by a form, script, or image on another site

■ All sites relying solely on automatic credentials are vulnerable!
  ▸ (almost all sites are this way)

■ Automatically Provided Credentials
  ▸ Session cookie
  ▸ Basic authentication header
  ▸ IP address
  ▸ Client side SSL certificates
  ▸ Windows domain authentication
CSRF Illustrated

1. Attacker sets the trap on some website on the internet (or simply via an e-mail)

2. While logged into vulnerable site, victim views attacker site

3. Vulnerable site sees legitimate request from victim and performs the action requested

Hidden `<img>` tag contains attack against vulnerable site

 `<img>` tag loaded by browser – sends GET request (including credentials) to vulnerable site

Application with CSRF vulnerability
A5 – Avoiding CSRF Flaws

- Add a secret, not automatically submitted, token to ALL sensitive requests
  - This makes it impossible for the attacker to spoof the request
    - (unless there’s an XSS hole in your application)
  - Tokens should be cryptographically strong or random

- Options
  - Store a single token in the session and add it to all forms and links
    - **Hidden Field:** `<input name="token" value="687965fdaew87agrde" type="hidden"/>
    - **Single use URL:** /accounts/687965fdaew87agrde
    - **Form Token:** /accounts?auth=687965fdaew87agrde ...
  - Beware exposing the token in a referer header
    - Hidden fields are recommended
  - Can have a unique token for each function
    - Use a hash of function name, session id, and a secret
  - Can require secondary authentication for sensitive functions (e.g., eTrade)

- Don’t allow attackers to store attacks on your site
  - Properly encode all input on the way out
  - This renders all links/requests inert in most interpreters

See the new: [www.owasp.org/index.php/CSRF_Prevention_Cheat_Sheet](http://www.owasp.org/index.php/CSRF_Prevention_Cheat_Sheet) for more details
A6 – Security Misconfiguration

Web applications rely on a secure foundation

- Everywhere from the OS up through the App Server
- Don’t forget all the libraries you are using!!

Is your source code a secret?

- Think of all the places your source code goes
- Security should not require secret source code

CM must extend to all parts of the application

- All credentials should change in production

Typical Impact

- Install backdoor through missing OS or server patch
- XSS flaw exploits due to missing application framework patches
- Unauthorized access to default accounts, application functionality or data, or unused but accessible functionality due to poor server configuration
Security Misconfiguration Illustrated
A6 – Avoiding Security Misconfiguration

■ Verify your system’s configuration management
  ▸ Secure configuration “hardening” guideline
    ▪ Automation is REALLY USEFUL here
  ▸ Must cover entire platform and application
  ▸ **Keep up with patches** for ALL components
    ▪ This includes software libraries, not just OS and Server applications
  ▸ Analyze security effects of changes

■ Can you “dump” the application configuration
  ▸ Build reporting into your process
  ▸ If you can’t verify it, it isn’t secure

■ Verify the implementation
  ▸ Scanning finds generic configuration and missing patch problems
Summary: How do you address these problems?

- **Develop Secure Code**
  - Follow the best practices in OWASP’s Guide to Building Secure Web Applications
  - Use OWASP’s Application Security Verification Standard as a guide to what an application needs to be secure
  - Use standard security components that are a fit for your organization
    - Use OWASP’s ESAPI as a basis for your standard components

- **Review Your Applications**
  - Have an expert team review your applications
  - Review your applications yourselves following OWASPA Guidelines
OWASP (ESAPI)

Custom Enterprise Web Application

OWASP Enterprise Security API


Your Existing Enterprise Services or Libraries

ESAPI Homepage: http://www.owasp.org/index.php/ESAPI
Acknowledgements

We’d like to thank the Primary Project Contributors
  - Aspect Security for sponsoring the project
  - Jeff Williams (Author who conceived of and launched Top 10 in 2003)
  - Dave Wichers (Author and current project lead)

Organizations that contributed vulnerability statistics
  - Aspect Security
  - MITRE
  - Softtek
  - WhiteHat Security

A host of reviewers and contributors, including:
  - Mike Boberski, Juan Carlos Calderon, Michael Coates, Jeremiah Grossman, Jim Manico, Paul Petefish, Eric Sheridan, Neil Smithline, Andrew van der Stock, Colin Watson, OWASP Denmark and Sweden Chapters