OWASP Top 10 – 2010

The Top 10 Most Critical Web Application Security Risks

Adrian Hayes
Security Consultant
Security-Assessment.com

The OWASP Foundation
http://www.owasp.org/

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 7, 8, 9 and 10
  - What I see day to day during webapp assessments
  - Widely applicable to .nz businesses
- These slides are heavily based on the work of others
  - See credits at the end

OWASP Top Ten (2010 Edition)

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

http://www.owasp.org/index.php/Top_10

A7 – Insecure Cryptographic Storage

Storing sensitive data insecurely

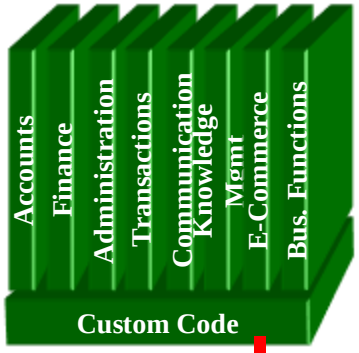
- Failure to identify all sensitive data
- Failure to identify all the places that this sensitive data gets stored
  - Databases, files, directories, log files, backups, etc.
- Failure to properly protect this data in every location

Typical Impact

- Attackers access or modify confidential or private information
  - e.g, credit cards, health care records, financial data (yours or your customers)
- Attackers extract secrets to use in additional attacks
- Company embarrassment, customer dissatisfaction, and loss of trust
- Expense of cleaning up the incident, such as forensics, sending apology letters, reissuing thousands of credit cards, providing identity theft insurance
- Business gets sued and/or fined

Insecure Cryptographic Storage Illustrated

1 Victim enters username and password to login



2 Error handler logs login credentials because SSO gateway is unavailable

3 Logs spewed all over the network by syslog.

4 Malicious hacker with network access (hacked wifi, WPS anyone?)

A7 – Avoiding Insecure Cryptographic Storage

- Verify your architecture
  - Identify all sensitive data
  - Identify all the places that data is stored
  - Ensure threat model accounts for possible attacks
  - Use cryptography to counter the threats, don't just 'encrypt' the data
- Protect with appropriate mechanisms
  - File encryption, database encryption, data element encryption
  - Hashing, Public Key Crypto, Symmetric Crypto
- Use the mechanisms correctly
  - Use standard strong algorithms
  - Generate, distribute, and protect keys properly
  - Be prepared for key change
- Verify the implementation
  - A standard strong algorithm is used, and it's the proper algorithm for this situation
  - All keys, certificates, and passwords are properly stored and protected
  - Safe key distribution and an effective plan for key change are in place
  - Analyze encryption code for common flaws

A8 – Failure to Restrict URL Access

How do you protect access to URLs (pages)?

- This is part of enforcing proper "authorization", along with A4 – Insecure Direct Object References

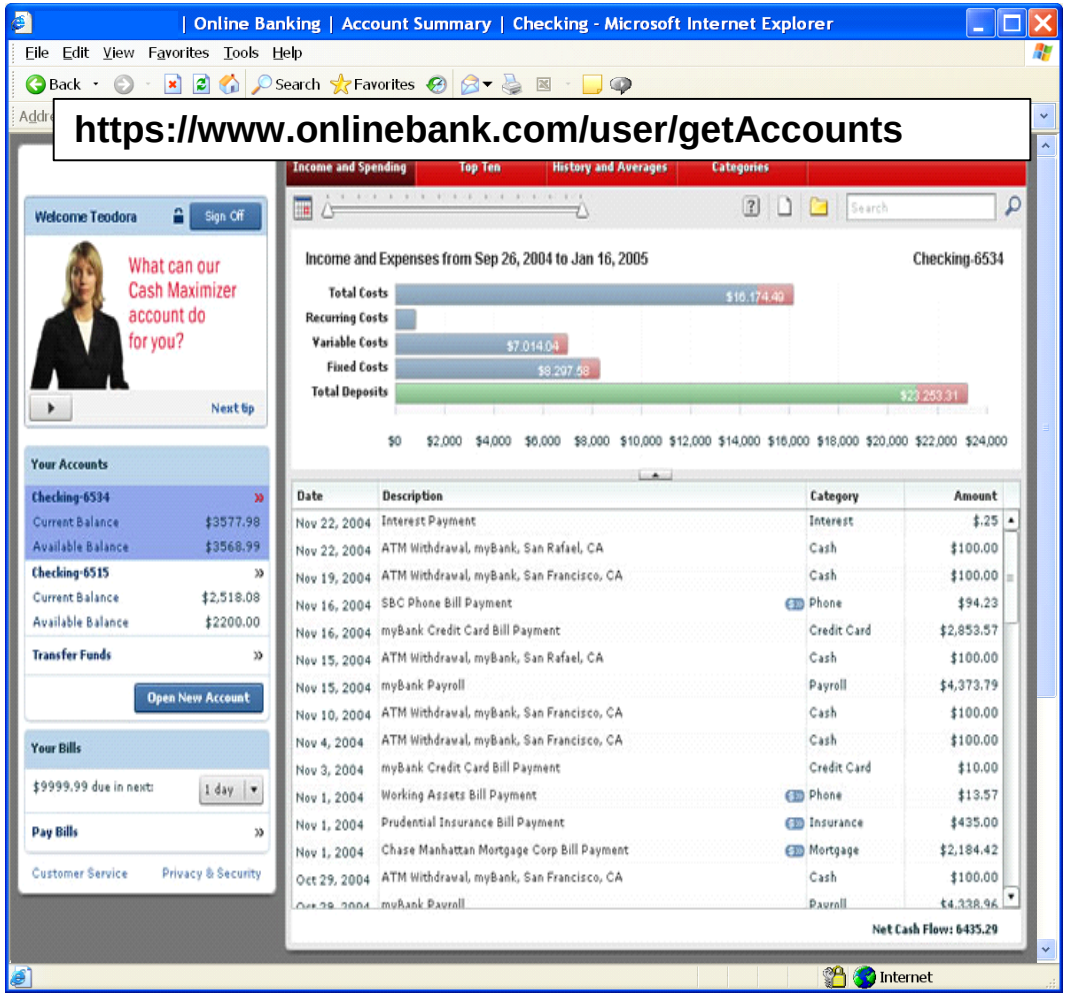
A common mistake …

- Displaying only authorized links and menu choices
- This is called presentation layer access control, and doesn't work
- Attacker simply forges direct access to 'unauthorized' pages

Typical Impact

- Attackers invoke functions and services they're not authorized for
- Access other user's accounts and data
- Perform privileged actions

Failure to Restrict URL Access Illustrated



- Attacker notices the URL indicates his role
  /user/getAccounts
- He modifies it to another directory (role)
  /admin/getAccounts, or
  /manager/getAccounts
- Attacker views more accounts than just their own

A8 – Avoiding URL Access Control Flaws

- For each URL, a site needs to do 3 things
  - Restrict access to authenticated users (if not public)
  - Enforce any user or role based permissions (if private)
  - Completely disallow requests to unauthorized page types (e.g., config files, log files, source files, etc.)
- Verify your architecture
  - Use a simple, positive model at every layer
  - Be sure you actually have a mechanism at every layer
- Verify the implementation
  - Forget automated analysis approaches
  - Verify that each URL in your application is protected by either
    - An external filter, like Java EE web.xml or a commercial product
    - Or internal checks in YOUR code – Use ESAPI's isAuthorizedForURL() method
  - Verify the server configuration disallows requests to unauthorized file types
  - Use your browser to forge unauthorized requests

A9 – Insufficient Transport Layer Protection

Transmitting sensitive data insecurely

- Failure to identify all sensitive data
- Failure to identify all the places that this sensitive data is sent
  - On the web, to backend databases, to business partners, internal communications
- Failure to properly protect this data in every location

Typical Impact

- Attackers access or modify confidential or private information
  - e.g, credit cards, health care records, financial data (yours or your customers)
- Attackers extract secrets to use in additional attacks
- Company embarrassment, customer dissatisfaction, and loss of trust
- Expense of cleaning up the incident
- Business gets sued and/or fined

Insufficient Transport Layer Protection Illustrated

External Victim

Custom Code

Backend Systems

Business Partners

Employees

1

External attacker steals credentials and data off network

External Attacker

2

Internal attacker steals credentials and data from internal network

Internal Attacker

A9 – Avoiding Insufficient Transport Layer Protection

- Protect with appropriate mechanisms
  - Use TLS on all connections with sensitive data
  - Individually encrypt messages before transmission
    - E.g., XML-Encryption
  - Sign messages before transmission
    - E.g., XML-Signature
- Use the mechanisms correctly
  - Use standard strong algorithms (disable old SSL algorithms)
  - Manage keys/certificates properly
  - Verify SSL certificates before using them
  - Use proven mechanisms when sufficient
    - E.g., SSL vs. XML-Encryption
- See: http://www.owasp.org/index.php/Transport_Layer_Protection_Cheat_Sheet for more details

A10 – Unvalidated Redirects and Forwards

Web application redirects are very common

- And frequently include user supplied parameters in the destination URL
- If they aren't validated, attacker can send victim to a site of their choice

Forwards (aka Server.Transfer in .NET) are common too

- They internally send the request to a new page in the same application
- Sometimes parameters define the target page
- If not validated, attacker may be able to use unvalidated forward to bypass authentication or authorization checks

Typical Impact

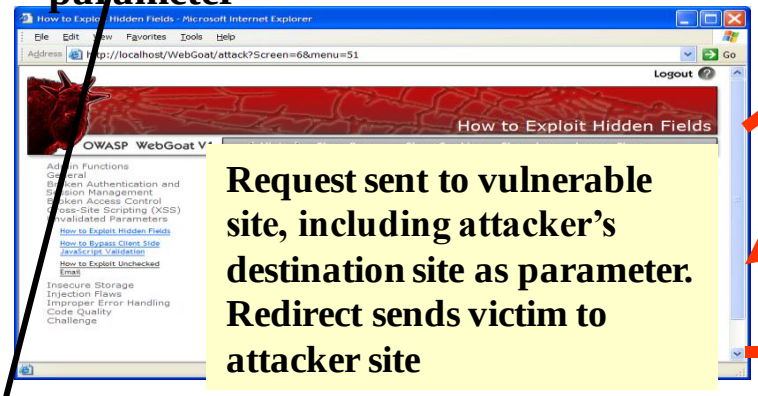
- Redirect victim to phishing or malware site
- Attacker's request is forwarded past security checks, allowing unauthorized function or data access
- Forward to URL handlers, javascript:// or skype://

Unvalidated Redirect Illustrated

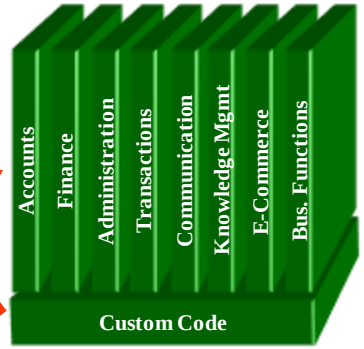
1 Attacker sends attack to victim via email or webpage

From: IRD
Subject: Your Unclaimed Tax Refund
Our records show you have an unclaimed tax refund. Please click here to initiate your claim.

2 Victim clicks link containing unvalidated parameter



3 Application redirects victim to attacker's site

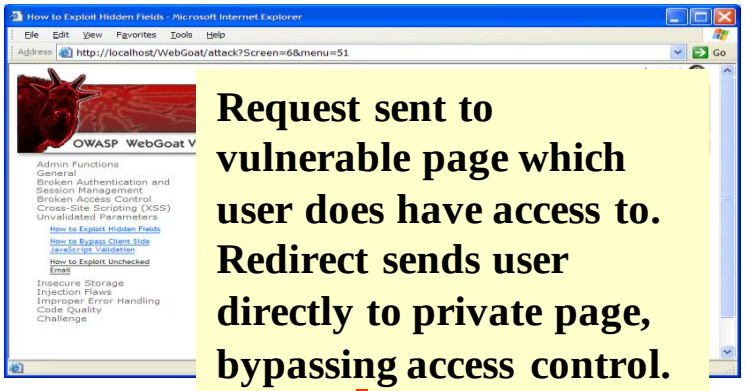


4 Evil site installs malware on victim, or phish's for private information

http://www.irs.gov/taxrefund/claim.jsp?year=2006 & … &dest=www.evilsite.com

Unvalidated Forward Illustrated

1 **Attacker sends attack to vulnerable page they have access to**



Request sent to vulnerable page which user does have access to. Redirect sends user directly to private page, bypassing access control.

2 **Application authorizes request, which continues to vulnerable page**

Filter

```
public void doPost( HttpServletRequest request,
HttpServletResponse response) {
    try {
        String target = request.getParameter( "dest" ) );
        ...
        request.getRequestDispatcher( target
        ).forward(request, response);
    }
    catch ( ...
```

```
public void sensitiveMethod(
HttpServletRequest request,
HttpServletResponse response) {
    try {
          // Do sensitive stuff here.
          ...
    }
    catch ( ...
```

3 **Forwarding page fails to validate parameter, sending attacker to unauthorized page, bypassing access control**

A10 – Avoiding Unvalidated Redirects and Forwards

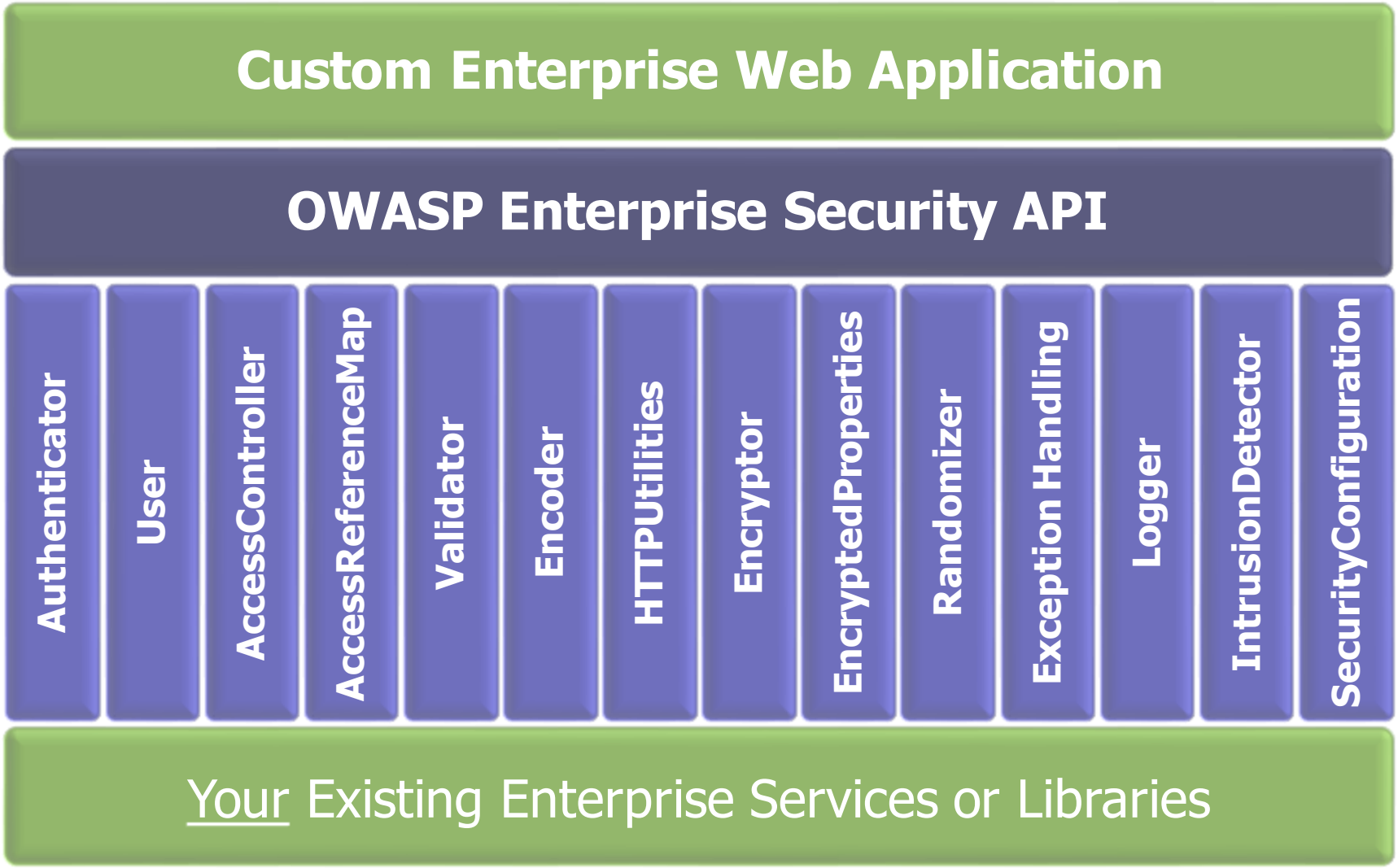
- There are a number of options
  1. Avoid using redirects and forwards as much as you can
  2. If used, don't involve user parameters in defining the target URL
  3. If you 'must' involve user parameters, then either
     a) Validate each parameter to ensure its valid and authorized for the current user, or
     b) (preferred) – Use server side mapping to translate choice provided to user with actual target page
  - Defense in depth: For redirects, validate the target URL after it is calculated to make sure it goes to an authorized external site
  - ESAPI can do this for you!!
    - See: SecurityWrapperResponse.sendRedirect(URL)

- Some thoughts about protecting Forwards
  - Ideally, you'd call the access controller to make sure the user is authorized before you perform the forward (with ESAPI, this is easy)
  - Don't let the user control where the forward goes
  - Don't forward across privilege boundaries (showRates.jsp -> updateRates.jsp)

Summary: How do you address these problems?

- Develop Secure Code
  - Follow the best practices in OWASP's Guide to Building Secure Web Applications
    - http://www.owasp.org/index.php/Guide
  - Use OWASP's Application Security Verification Standard as a guide to what an application needs to be secure
    - http://www.owasp.org/index.php/ASVS
  - Use standard security components that are a fit for your organization
    - Use OWASP's ESAPI as a basis for your standard components
    - http://www.owasp.org/index.php/ESAPI
- Review Your Applications
  - Have an expert team review your applications
  - Review your applications yourselves following OWASP Guidelines
    - OWASP Code Review Guide: http://www.owasp.org/index.php/Code_Review_Guide
    - OWASP Testing Guide: http://www.owasp.org/index.php/Testing_Guide

OWASP (ESAPI)



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