OWASP Top 10 – 2010

The Top 10 Most Critical Web Application Security Risks

Nick Freeman
Security Consultant, S-A.com
OWASP NZ Chapter Leader (Auckland)

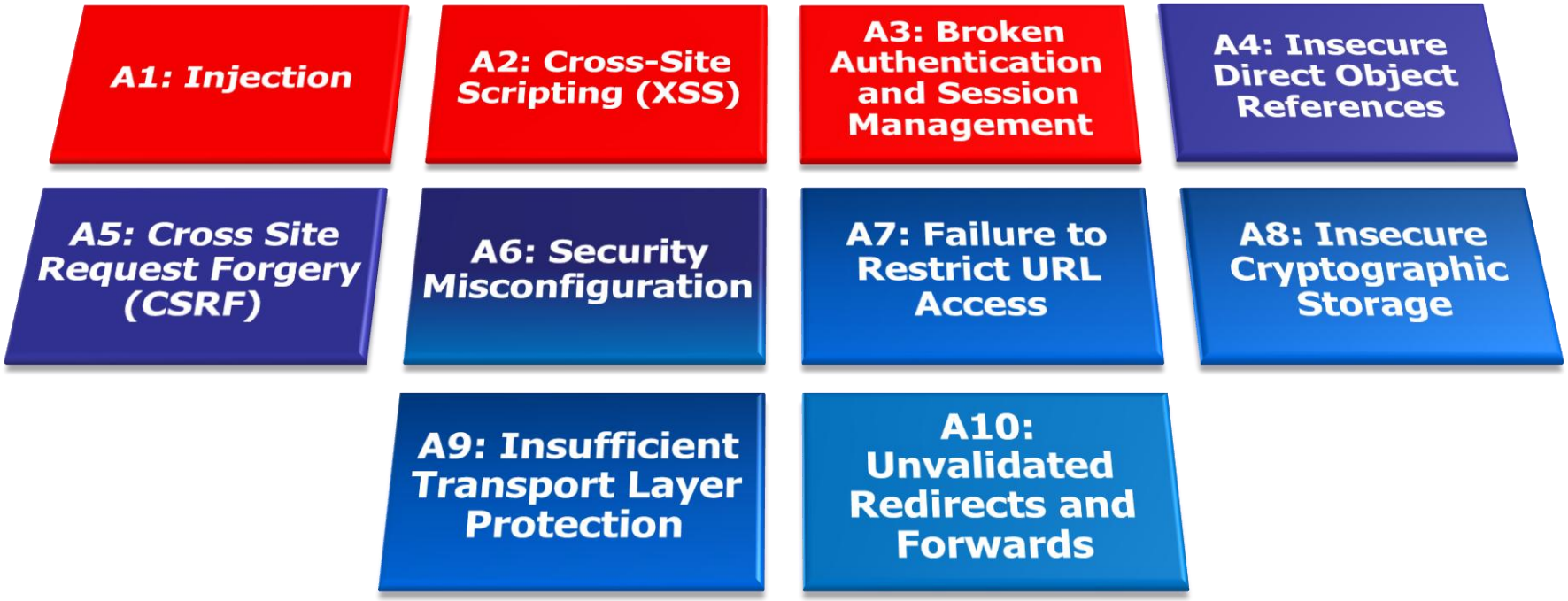
nick.freeman@security-assessment.com
nick.freeman@owasp.org

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?
  - Feedback from OWASP day
  - 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

A1 – Injection

Injection means…

- Tricking an application into including unintended commands in the data sent to an interpreter

Interpreters…

- Take strings and interpret them as commands
- SQL, OS Shell, LDAP, XPath, Hibernate, etc…

Typical Impact

- Usually severe.
  - Entire database can usually be read or modified
  - Could allow read/write of local files
- May also allow OS level access

A1.a – SQL Injection

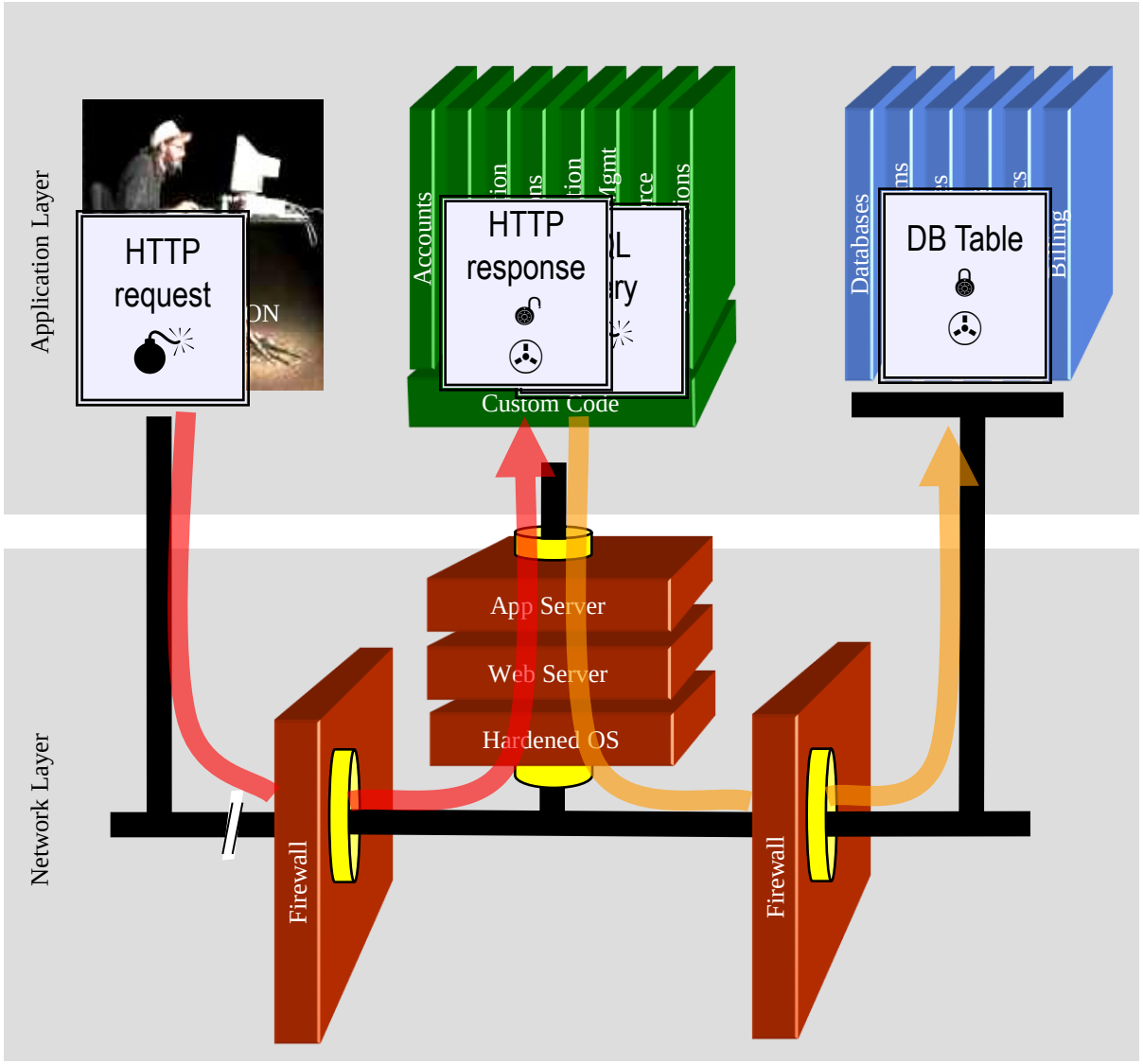
SQL Injection means…

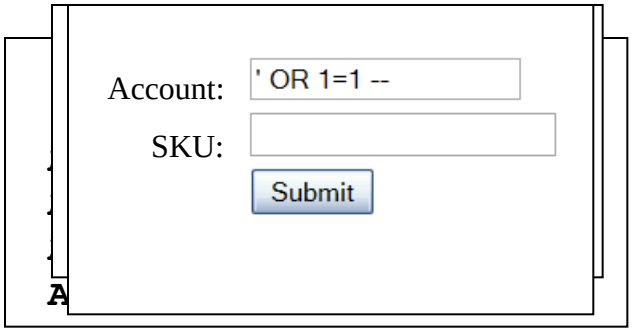
- Tricking a web application into including unintended commands in the data sent to a database driver
- The attacker's injection modifies the application's SQL query to do his own evil bidding

Typical Impact

- Usually severe.
  - Entire database can usually be read or modified
  - Could allow read/write of local files
- May also allow OS level access

SQL Injection – Illustrated





1. Application presents a form to the attacker
2. Attacker sends an attack in the form data
3. Application sends modified SQL query to database
4. Database runs query containing attack and sends encrypted results back to application
5. Application decrypts data as normal and sends results to the user

Demo – Auth Bypass

Demo 1 - Details

- Authentication bypass

```
$query = 'SELECT userid FROM tbl_users WHERE
   username = ' + $username + 'AND password = ' +
   $password;
$db_handler->execute($query);
```

  - In this case, $query ends up being modified to be:

```
SELECT userid FROM tbl_users WHERE username = 'a'
            OR 1=1#' AND password = '';
```


Avoiding SQL Injection Flaws

- Recommendations
  1. Avoid the interpreter entirely, or
  2. Use an interface that supports bind variables (e.g., **prepared statements**, or **stored procedures**),
     - Bind variables allow the interpreter to distinguish between code and data
     - Most all frameworks have ways to do this. There is **NO EXCUSE**!
  3. Encode all user input before passing it to the interpreter
  - Always perform 'white list' input validation on all user supplied input
  - Always minimise database privileges to reduce the impact of a flaw
- References
  - For more details, read the new http://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet

A1.b – OS Command Injection

OS Command Injection means…

- The web application uses your dodgy HTTP parameters as input to craft a command executed by the underlying OS.

Largely unheard of these days

- Few people think letting The Internets run commands on the OS is a good idea

Typical Impact

- Arbitrary remote command execution.

Avoiding OS Command Injection Flaws

- Recommendations
  1. Don’t trust user input to form arguments to a system command
  2. Escape shell meta characters such as pipe (|) semicolons (;) etc.
  3. Always perform ‘white list’ input validation on all user supplied input
  4. Ensure the web server is running with a user with low privileges
  5. Chroot the web server to limit exposure in the event of web server compromise.

A1.c – XML Injection

XML Injection means…

- The web application uses your HTTP parameters as input to create an XML query

XML injection is not common

- Most XML parsers do sensible input validation, and don't allow you to supply extra tags or include external entities

Typical Impact

- Can allow Cross Site Scripting (coming up soon)
- Can allow local file read
- Can allow port scanning of the local network

Example

- External Entity Allows Local File Read

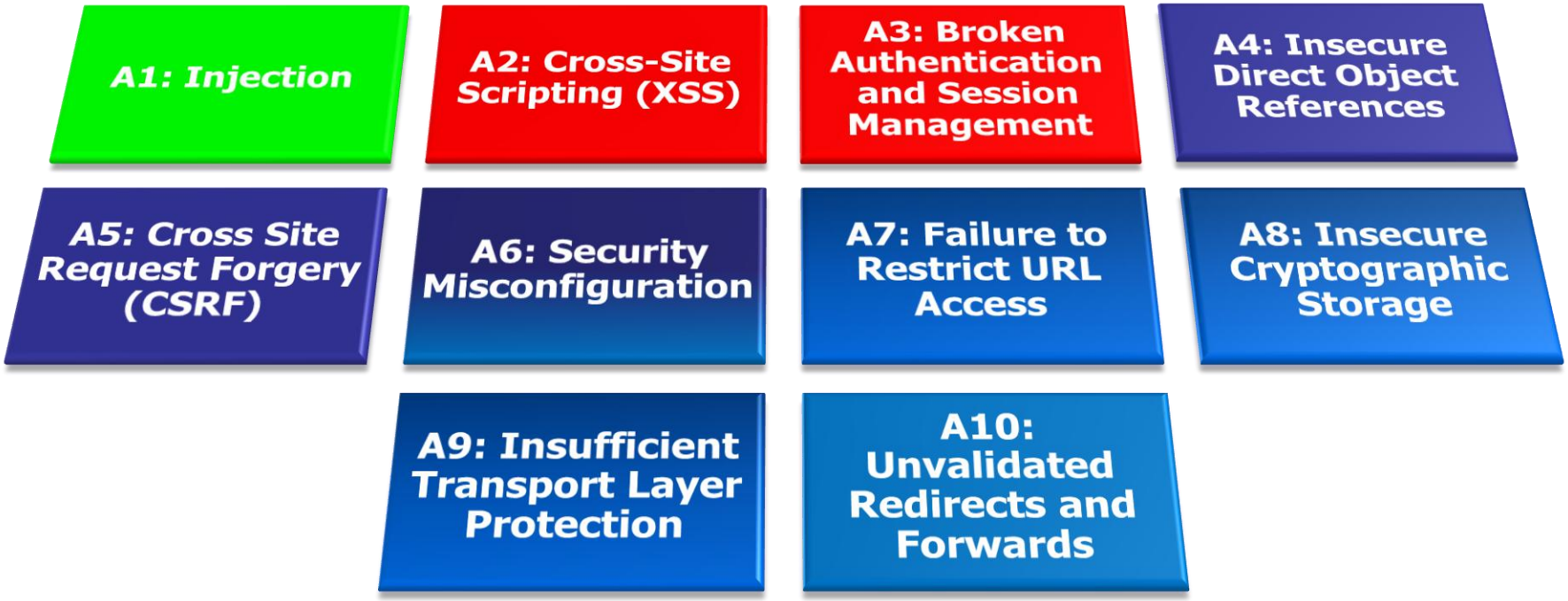
```
<?xml version="1.0" encoding="ISO-8859-1"?>
 <!DOCTYPE foo [
   <!ELEMENT foo ANY >
   <!ENTITY xxe SYSTEM "file:///etc/passwd">
 ]>
 <foo>&xxe;</foo>
```

  - Returns contents of /etc/passwd back to the user

Avoiding XML Injection Flaws

- Recommendations
  - Use a real XML parser. **DON'T** roll your own, they always suck
  - Filter for malicious input (get rid of <>?/& etc)
  - XPath queries should not contain any meta characters (such as ' = * ? // or similar)

OWASP Top Ten (2010 Edition)



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

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

A2 – Cross-Site Scripting (XSS)

Occurs any time…

- Raw data from attacker is sent to an innocent user's browser

Raw data…

- Stored in database
- Reflected from web input (form field, hidden field, URL, etc…)
- Sent directly into rich JavaScript client

Virtually every web application has this problem

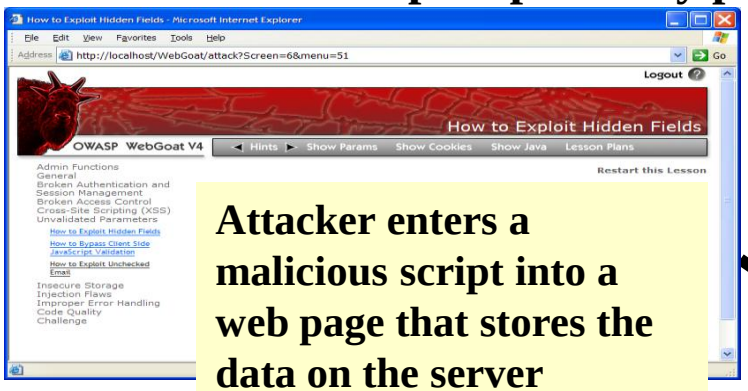
- Many that attempt to fix it don't apply the fix consistently

Typical Impact

- Steal user's session, steal sensitive data, rewrite web page, redirect user to phishing or malware site, enable XSRF exploitation
- Most Severe: Install XSS proxy which allows attacker to observe and direct all user's behavior on vulnerable site and force user to other sites

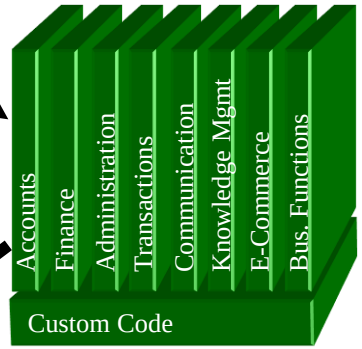
Cross-Site Scripting Illustrated

1 **Attacker sets the trap – update my profile**

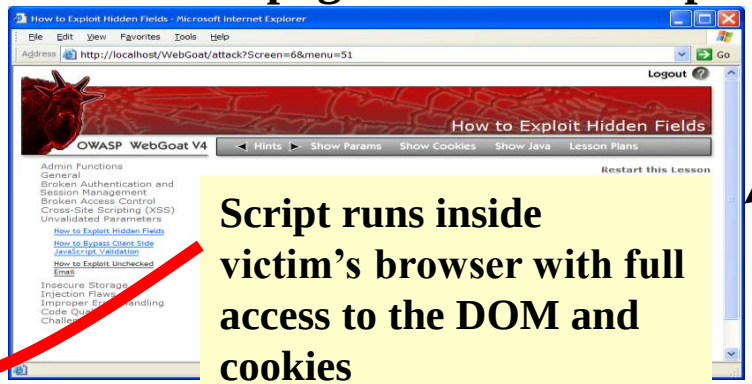


Attacker enters a malicious script into a web page that stores the data on the server

Application with stored XSS vulnerability



2 **Victim views page – sees attacker profile**



Script runs inside victim's browser with full access to the DOM and cookies

3 **Script silently sends attacker Victim's session cookie**

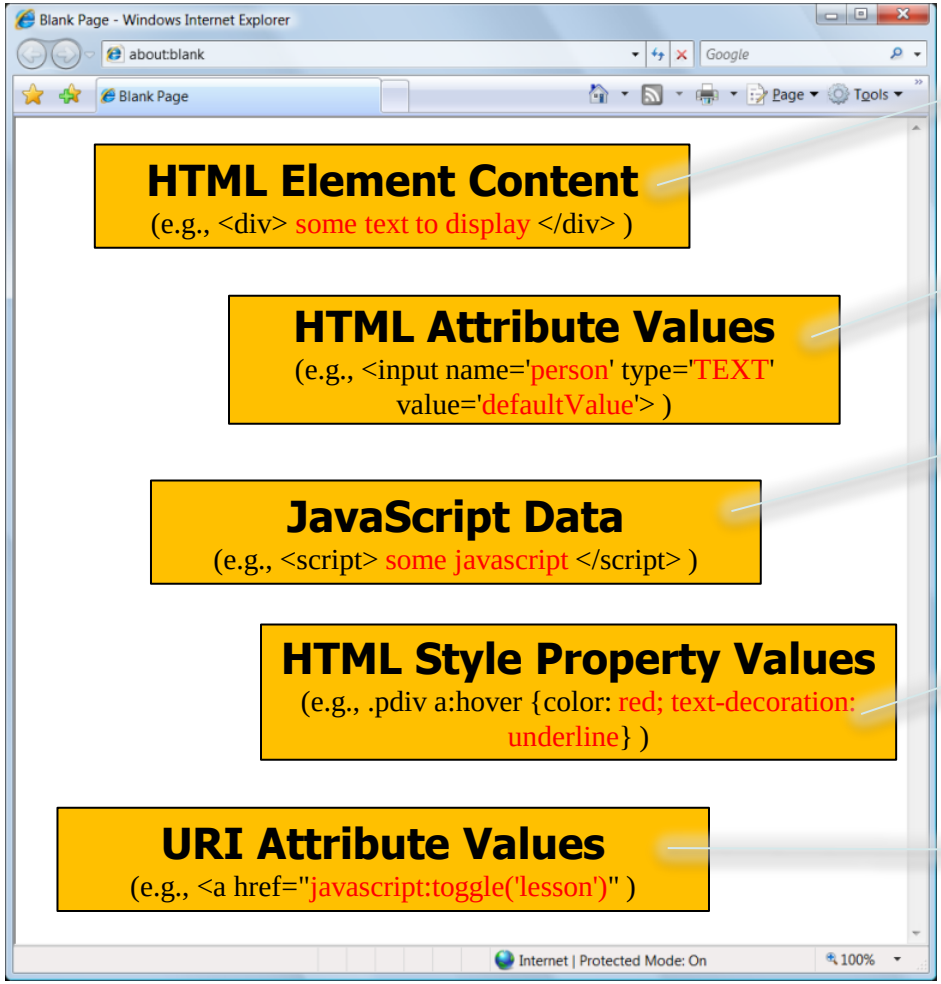
Demo – Cookie Leakage & BeEF

A2 – Avoiding XSS Flaws

- Recommendations
  - Eliminate Flaw
    - Don't include user supplied input in the output page
  - Defend Against the Flaw
    - Primary Recommendation: Output encode all user supplied input (Use OWASP's ESAPI to output encode:
      http://www.owasp.org/index.php/ESAPI
    - Perform 'white list' input validation on all user input to be included in page
    - For large chunks of user supplied HTML, use OWASP's AntiSamy to sanitize this HTML to make it safe
      See: http://www.owasp.org/index.php/AntiSamy
- References
  - For how to output encode properly, read the new http://www.owasp.org/index.php/XSS_(Cross Site Scripting) Prevention Cheat Sheet

(AntiSamy)

Safe Escaping Schemes in Various HTML Execution Contexts



#1: (&, <, >, ") → &entity; (', /) → &#xHH;
ESAPI: encodeForHTML()

#2: All non-alphanumeric < 256 → &#xHH
ESAPI: encodeForHTMLAttribute()

#3: All non-alphanumeric < 256 → \xHH
ESAPI: encodeForJavaScript()

#4: All non-alphanumeric < 256 → \HH
ESAPI: encodeForCSS()

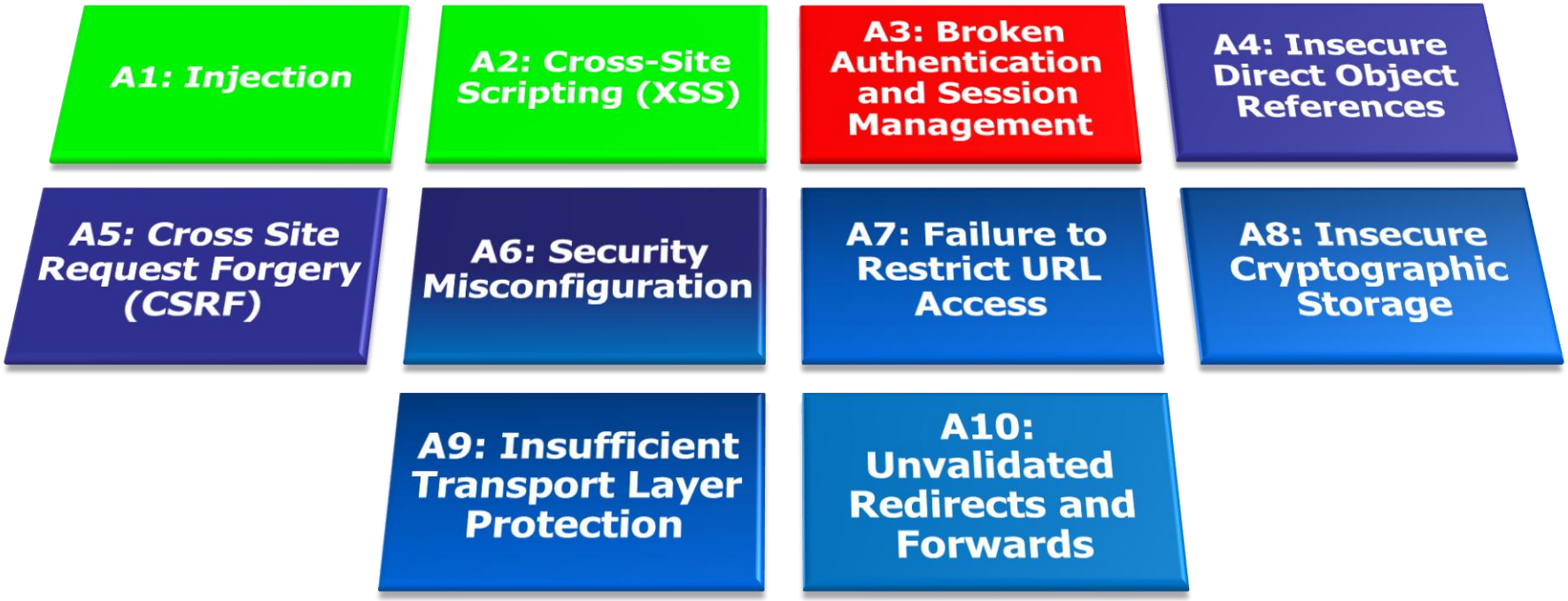
#5: All non-alphanumeric < 256 → %HH
ESAPI: encodeForURL()

ALL other contexts CANNOT include Untrusted Data

Recommendation: Only allow #1 and #2 and disallow all others

See: www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet for more details

OWASP Top Ten (2010 Edition)



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

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

A3 – Broken Authentication and Session Management

HTTP is a "stateless" protocol

- Means credentials have to go with every request
- Should use SSL for everything requiring authentication

Session management flaws

- SESSION ID used to track state since HTTP doesn't
  - and it is just as good as credentials to an attacker
- SESSION ID is typically exposed on the network, in browser, in logs, …

Beware the side-doors

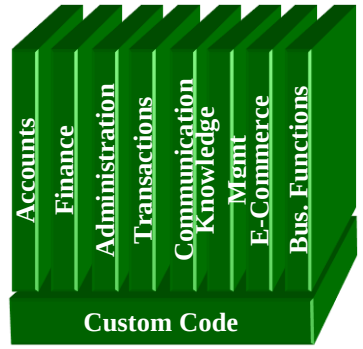
- Change my password, remember my password, forgot my password, secret question, logout, email address, etc…

Typical Impact

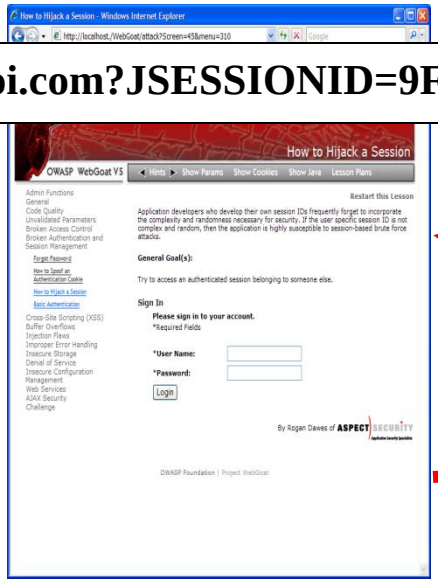
- User accounts compromised or user sessions hijacked

Broken Authentication Illustrated

1 User sends credentials



www.boi.com?JSESSIONID=9FA1DB9EA...



Site uses URL rewriting
(i.e., put session in URL)

2

3 User clicks on a link to http://www.hacker.com from somewhere on www.boi.com, sends HTTP Referer: www.boi.com?JSESSIONID=9FA1DB..

4

Hacker checks referer logs on www.hacker.com and finds user's JSESSIONID

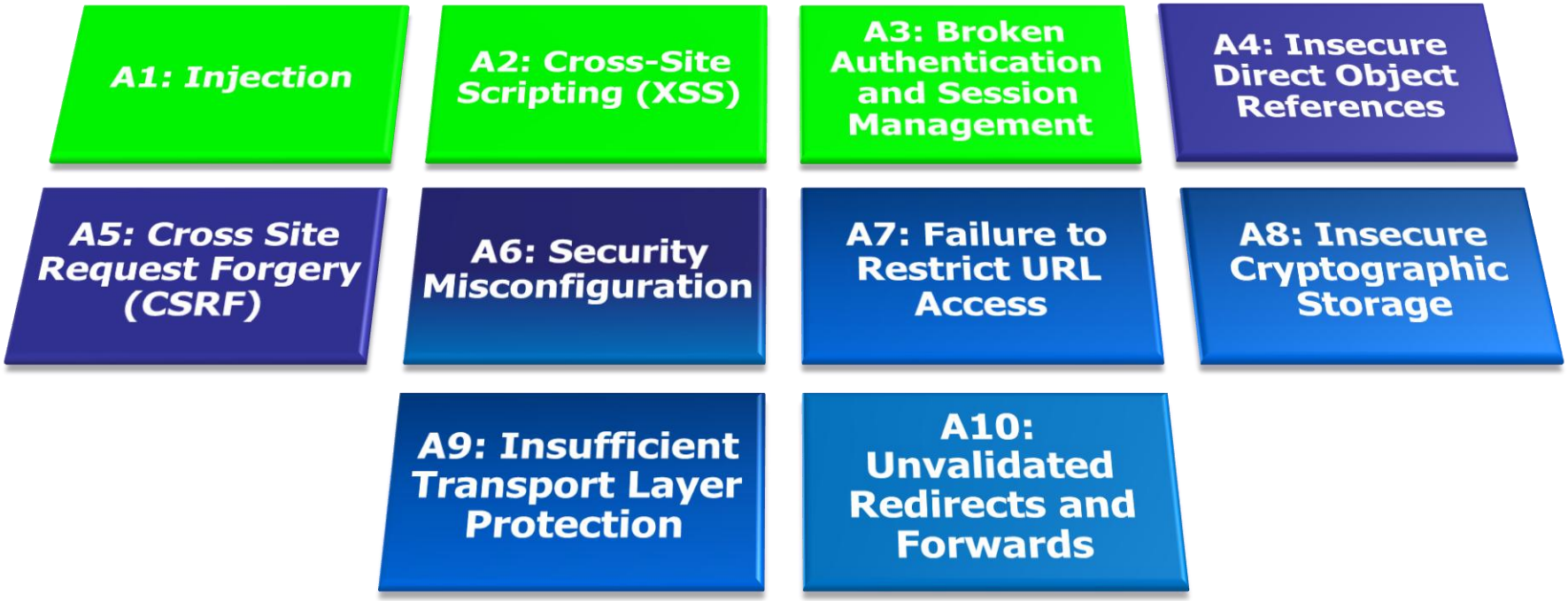
5

Hacker uses JSESSIONID and takes over victim's account

A3 – Avoiding Broken Authentication and Session Management

- Verify your architecture
  - Authentication should be simple, centralized, and standardized
  - Use the standard session id provided by your framework
  - Be sure SSL protects both credentials and session id at all times
- Verify the implementation
  - Forget automated analysis approaches
  - Check your SSL certificate
  - Examine all the authentication-related functions
  - Verify that logoff actually destroys the session
  - Use OWASP's WebScarab to test the implementation
- Follow the guidance from
  - http://www.owasp.org/index.php/Authentication_Cheat_Sheet

OWASP Top Ten (2010 Edition)



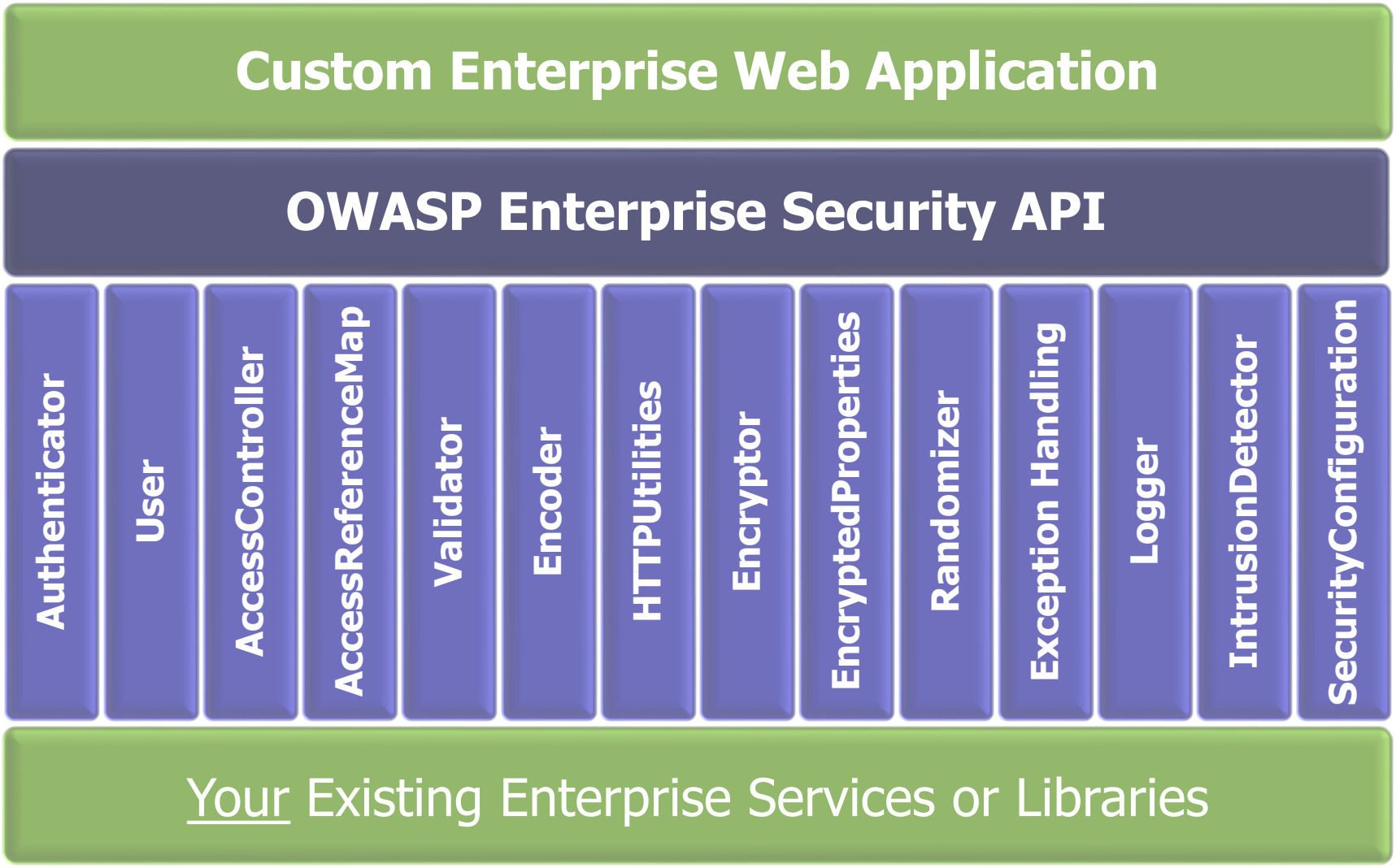
OWASP
The Open Web Application Security Project
http://www.owasp.org

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

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