Introduction to Secure Web Coding

Presented by Gavin Porter

Introductions

FAQ for Today

- Toilets
  - Breaks and labs are good times to visit!
- Fire Alarms
- Food
- Questions

Common Standards

- Payment Card Industry (PCI) Data Security Standard (DSS)
- Australian Government Information Security Manual (AG ISM)
- New Zealand Information Security Manual (NZISM)
- ISO 27002 - commercial (non-public) standard of security controls for the design & maintenance of IT systems.

PCI Standards

PCI-DSS

Two really simple principles to remember:

- Never touch the full credit card number
- Redirect users to a dedicated payment page from a PCI DSS certified payment gateway.

What do we need to do? - ASDISM / NZISM

Key coding sections:

- Software Application Development -> Secure Programming
- Access Control -> Identification and Authentication
- Access Control -> Event Logging and Auditing
- Web Application Development

Secure Programming

Software developers should use secure programming practices when writing code, including:

- designing software to use the lowest privilege level needed to achieve its task;
- denying access by default;
- checking return values of all system calls; and
- validating all inputs.

User Identification and Authentication

- Users must be uniquely identifiable (no shared accounts);
- System user authentication data to be protected when in transit on networks;
- Minimum password complexity.

User Identification and Authentication (CONT.)

- Password management policy;
  - e.g. don't store passwords in the clear;
- Account lockout policy;
  - e.g. lock user accounts after three failed logon attempts
- Displaying when a user last logged in.

Event Logging and Auditing

Log at least the following events for all software components:

- user login;
- privileged operations, e.g. new users, role changes, permission changes;
- failed attempts to elevate privileges;
- failed attempts to access files, functions or services;
- security related alerts and failures.

Event Logging and Auditing (CONT.)

“ Who did what, where and when? ”

The *Where* needs to include the device that generated the event and the source IP of the web request.

Web Applications

Use a robust web application framework

It should assist with complex components such as session management, input handling and cryptographic operations.

Input Handling

“ It is essential that web applications do not trust any input such as the URL and its parameters, HTML form data, cookie values and HTTP request headers without validating and/or sanitising it. ”

Output Handling

Output encoding, e.g. HTML or URL, is essential on all output produced by a web application.

> *“ It is particularly useful where external data sources, which may not be subject to the same level of input filtering, are output to users. ”*

Browser-based security controls

> *Browser-based security controls such as Content Security Policy, HTTP Strict Transport Security and Frame Options can be leveraged by web applications to help protect the web application and its users.*

OWASP

For web application development, agencies should follow the Open Web Application Security Project (OWASP) guide to building secure Web applications and Web services.

OWASP

- Top Ten List of Security Weaknesses
- Cheat Sheets (lots)
- Secure Coding Quick Reference Guide
- Zed Attack Proxy (ZAP)

OWASP Top Ten

Aimed at web applications (and web APIs).

The rest of the day will be covering each vulnerability along with interactive LAB sessions using a framework called bWAPP.

New version released in November 2017.

OWASP Top Ten 2017

- A1 Injection [was #1]
- A2 Broken Authentication [was #2]
- A3 Sensitive Data Exposure [was #6]
- A4 XML External Entities (XXE) [new]
- A5 Broken Access Control [was #4 and #7]
- A6 Security Misconfiguration [was #5]

OWASP Top Ten 2017

- A7 Cross-Site Scripting [was #3]
- A8 Insecure Deserialization [new]
- A9 Using Components with Known Vulnerabilities [was #9]
- A10 Insufficient Logging & Monitoring [new]
- Old A8 - Cross-Site Request Forgery

A1 - Injection

Injection flaws occur when an application sends untrusted data to an interpreter.

Examples:

- SQL;
- NoSQL;
- LDAP;
- XML;
- OS commands.

A1 - Injection

SQL

```
mysql_query("select * from users where name = '" + $name + "'");
```

```
$name = "mike o'connor";
```

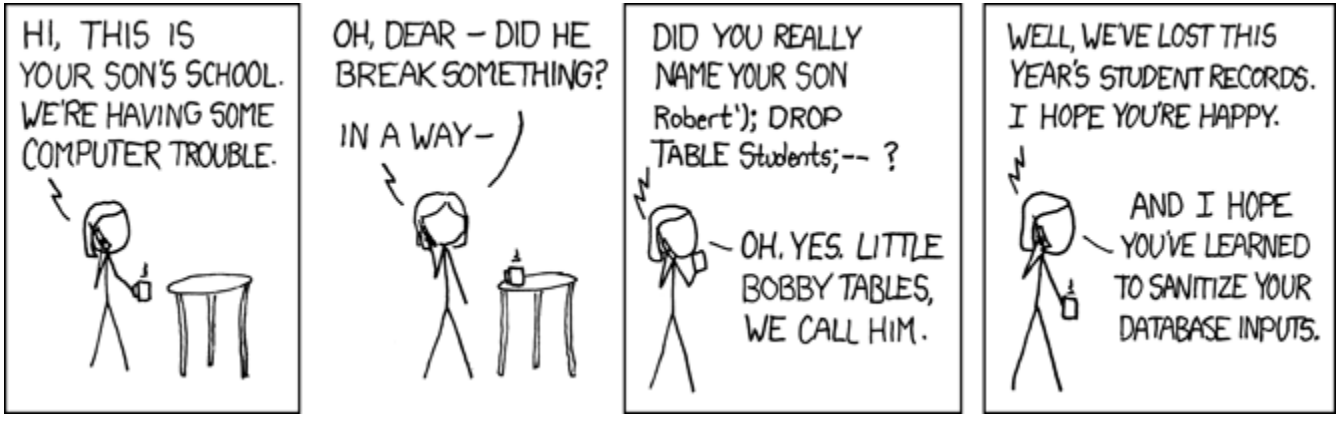
```
ERROR:  syntax error at or near "connor"
LINE 1: select * from users where name = 'mike o'connor';
```

```
$name = "bobby';drop table users;--";
```

```
mysql_query("select * from users where name = 'bobby';
  drop table users;--'");
```

A1 - Injection

SQL



A1 - Injection

OS commands

```
system("ping -c 3 $hostname");
```

```
$hostname = "google.com;cat /etc/passwd"
```

```
system("ping -c 3 google.com;cat /etc/passwd");
```

A1 - Injection

OS command example

Netgear R7000 - Real Command Injection (Dec 2016)

https://www.exploit-db.com/exploits/40889/

Open a telnet service on port 45:

```
http://router_ip/;telnetd$IFS-p$IFS'45'
```

A1 - Injection

Prevention

- Input validation;
- Escape special characters;
- Separate untrusted data from commands.

A1 - Injection

Prevention | Separating data

- Parameterized Queries;
  - Insert placeholders in queries;
  - Supply data separately.
- Use a 'Safe' API.
  - A database ORM to provide commonly needed functionality.

A2 - Broken Authentication

- Credentials not protected when stored, or transmitted;
- Credentials can be guessed or overwritten;
  - e.g. brute-force on a recover password page,
  - or, knowledge-based security answers
- Session IDs...
  - exposed in URLs;
  - not changed after login;
  - not invalidated after logout or timeout.

A2 - Broken Authentication

Prevention

- Use multi-factor authentication;
- Disable default accounts / change default passwords;
- Enforce a 'sensible' password policy;
- Check for weak passwords;
- Limit or increasingly delay failed login attempts;
- Detect / prevent account enumeration attacks against registration and recovery pages.

A2 - Broken Authentication

Prevention

- Use random session cookies;
- Use 'secure' and 'HttpOnly' for session cookies;

```
SetCookie: sessionid=kp0l2g67g7ki9mbjb1rgf1a3f5; httpOnly; secure
```

- Use HTTPS including local assets and third-party scripts;
- Use an authentication and session management framework.

A2 - Broken Authentication

Auto-complete

Tell browsers not to cache information in an input field:

```
autocomplete="off"
```

- Great for sensitive information such as credit card number.
- Historically often recommended for password fields.
  - but, blocks password managers.
  - HTML5 allows password managers to override page instructions.

A2 - Broken Authentication

Auto-complete | Browser Support

Browsers started ignoring autocomplete for password fields:

- Internet Explorer - version 11 (November 2013)
- Firefox - version 30 (February 2014)
- Safari - version 7.02 (February 2014)
- Chrome - version 34 (April 2014)

A3 - Sensitive Data Exposure

was A6

What counts as 'Sensitive'?

- Passwords;
- Personal information;
- Health information;
- Financial information, such as credit cards.

A3 - Sensitive Data Exposure

How can it be exposed?

- Is it transmitted in clear text across a network?
- Is it stored encrypted? backups?
- Is it stored protected by an access control list?
- Are the crypto algorithms strong?

A3 - Sensitive Data Exposure

Prevention

- Use encryption!
- Use access controls!
- Use **strong** encryption!
- Destroy data when no longer required.
- Don't cache sensitive data at the client.
- Store encryption keys sensibly (private means private).

Government 'Paranoid' Approved Crypto

- Approved Symmetric Encryption Algorithm:
  - AES with a key length of at least 256 bits.
- Approved Hashing Algorithm:
  - Secure Hashing Algorithm 2 (i.e. SHA-256, SHA-384 and SHA-512)
- Approved Asymmetric/Public Key Algorithms:
  - Elliptic Curve Diffie Hellman (ECDH) - used for agreeing on encryption session keys;
  - Elliptic Curve Digital Signature Algorithms (ECDSA) - used for digital signatures.

NZISM Approved TLS Cryptographic Algorithms

This is what we should be using for any government system doing HTTPS:

```
$ openssl ciphers 'EECDH+AES256+AESGCM+ECDSA!SHA1'
```

```
ECDHE-ECDSA-AES256-GCM-SHA384
```

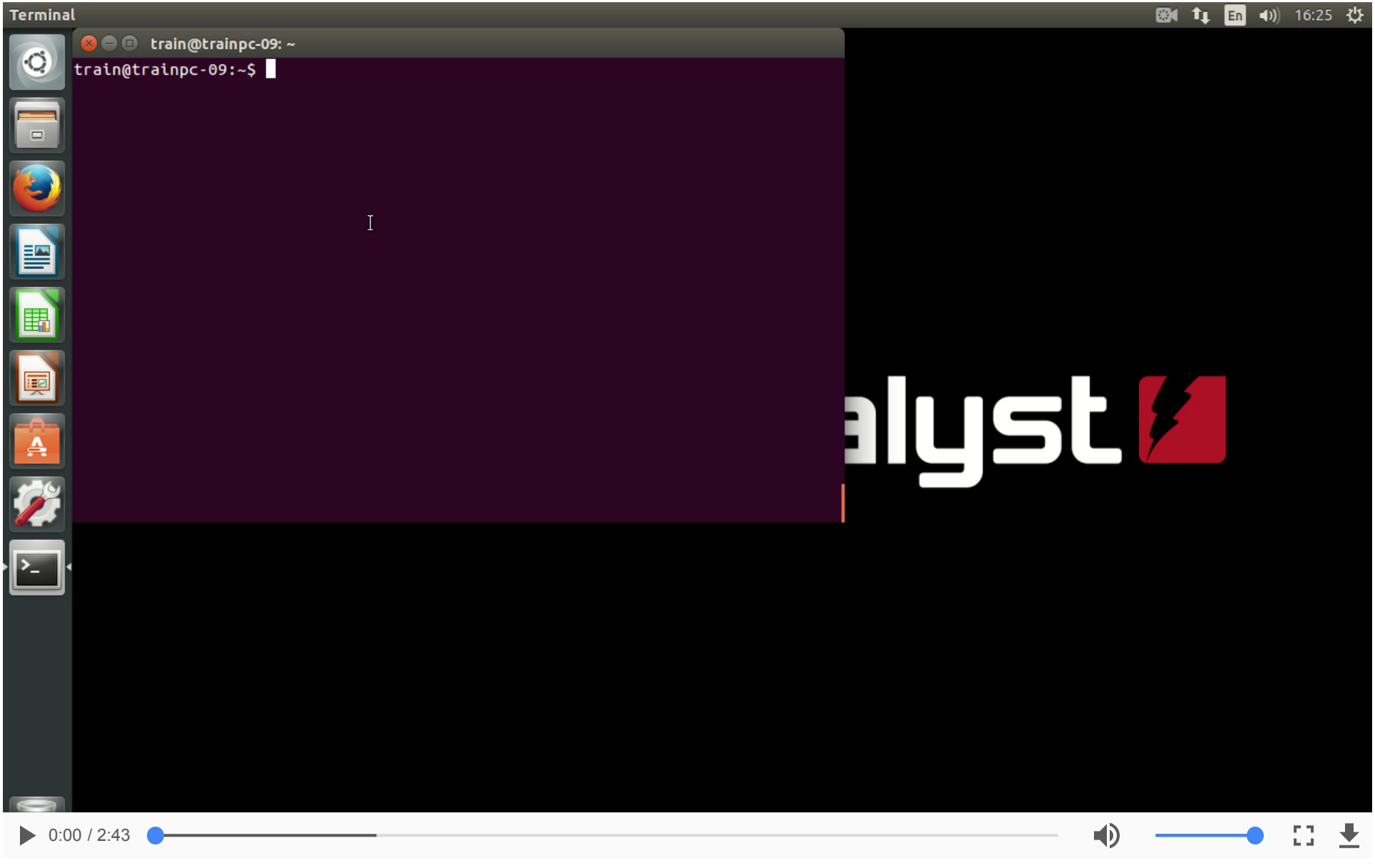
Doesn't work with many/all web browsers.

https://en.wikipedia.org/wiki/Transport_Layer_Security#Web_br

SSL Configuration Generator

https://mozilla.github.io/server-side-tls/ssl-config-generator/

Video | Wireshark



bWAPP, a buggy web application

- An open-source deliberately insecure web application;
- For learning about how to find, and how to prevent vulnerabilities;
- Based on OWASP Top Ten 2013;
- Lots of labs.

LAB #1 | bWAPP

Go to http://[a.b.c.d]/

Select your 'student number', then login as **bee**, with password **bug** and security setting as "low".

Work through your worksheets for SQL Injection and Command Injection.

A4 - XML External Entities (XXE)

The 'Entity' statement defines entities in the XML document type definition (DTD).

Format is:

```
<!ENTITY [%] name [SYSTEM|PUBLIC publicID] resource [NDATA notation] >
```

Resource can be a URL to reference an external entity.

A4 - XML External Entities (XXE)

Attacks occur when...

XML input containing a reference to an external entity is processed by a weakly configured XML parser.

This attack may lead to:

- disclosure of confidential data
- server side request forgery
- port scanning from parser machine
- ...and other system impacts.

A4 - XML External Entities (XXE) Example

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

A4 - XML External Entities (XXE)

Common sources

- SOAP
- SAML
- Any untrusted XML that is processed

A4 - XML External Entities (XXE) Prevention

- Use a simpler data format such as JSON
- Use a standard XML processing library and upgrade it
- Use whitelist input validation on the XML data
  - and use XSD validation
- Disable XML external entity and DTD processing
  - Look at OWASP 'XXE Prevention' cheatsheet

https://www.owasp.org/index.php/XML_External_Entity_%28XXE%29_Prevention_Cheat_Sheet

A5 - Broken Access Control

Was A4 - Insecure Direct Object References

Access to an object, such as a file, or a REST API call, is permitted without the user going through the proper access control and authorisation checks.

or access to a public object under embargo.

or internal document posted to AWS data store without access controls.

A5 - Broken Access Control

Insecure Direct Object References | Examples

- http://store.example.com/invoices/12345
- http://fileserver.example.com/getFile?
  file=../../../../../../../../../etc/passwd

A5 - Broken Access Control

Insecure Direct Object References | Examples

- Always perform an access control for every request;
- Do not use direct object references:
  - Use per user indirect object references;
  - Use per session indirect object references.
  - Use UUIDs instead of plain integers for ID numbers
- Do not allow a user to use ../ to access a resource

A5 - Broken Access Control

Was A7 - Missing Function Level Access Control

Inadequate access control allows attackers to access unauthorised functionality, particularly administrative functionality.

A5 - Broken Access Control

Missing Function Level Access Control | Examples

- http://bank.example.com/transfer?from=12345&to=67890&amount=10000000
- http://hr.example.com/admin/setPay.php?employee=1234&salary=9999999

A5 - Broken Access Control

Missing Function Level Access Control | Prevention

- Enforce access control on the server, not just in the client
- Enforce access control using data that the user cannot tamper with
- Use a principle of 'Deny by default'

LAB #2

Go to http://[a.b.c.d]/

Work through the access controls worksheet.

A6 - Security Misconfiguration

was A5

Security misconfiguration can happen at any level of an application stack, including the platform, web server, application server, database, framework, and custom code.

A6 - Security Misconfiguration

Prevention

- Have a defined configuration / hardening process for each component of the tech stack;
  - Disable unnecessary and default features and accounts;
  - Disable stack traces and detailed error messages in production.
- Use security scanning tools, e.g. OpenVAS, Nessus, Qualys.

Intercepting proxies

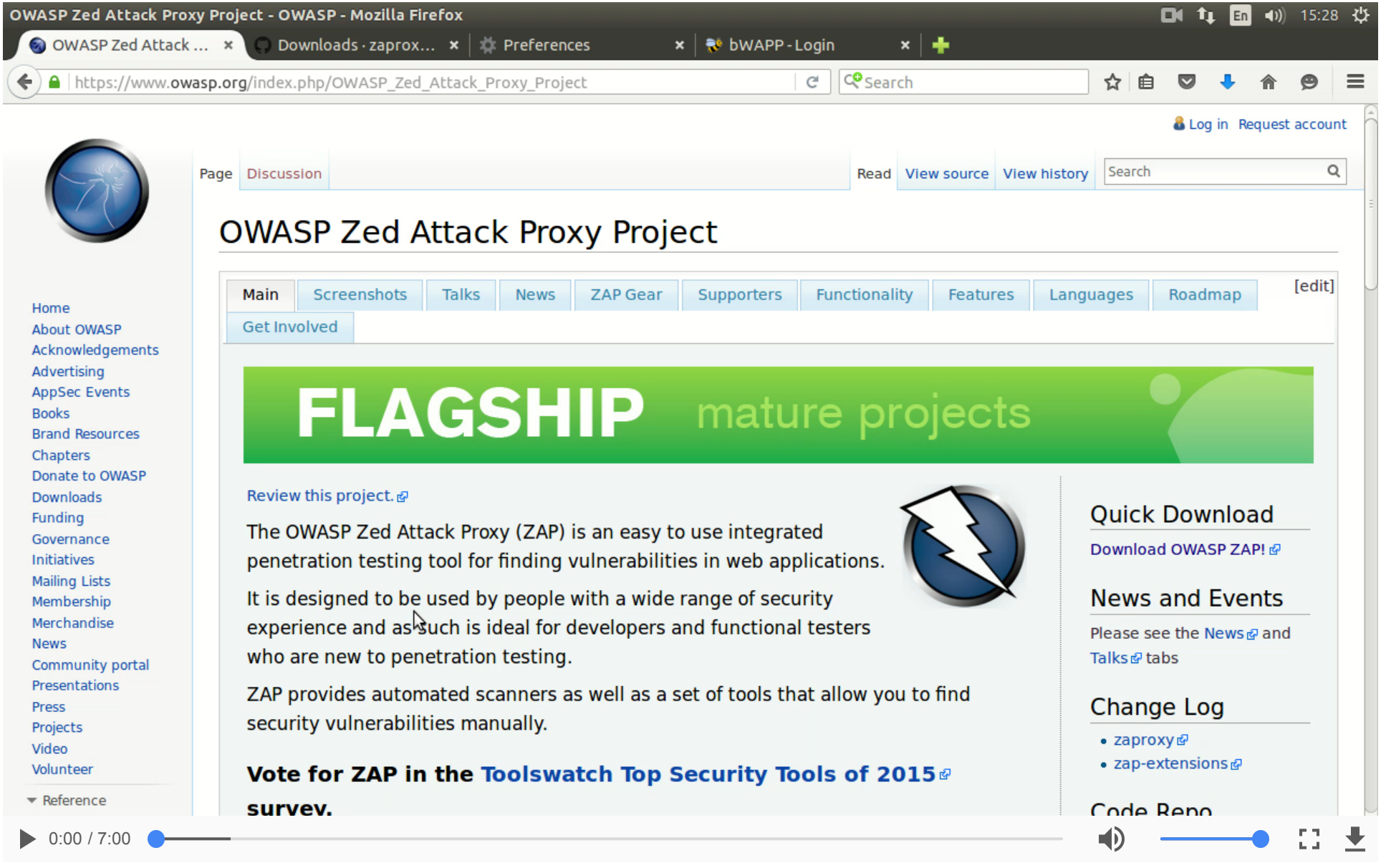
Proxies that allow interception and modification of the request and response of web requests.

Some examples:

- OWASP ZAP (open source)
- Burp (commercial)

Some proxies can also scan a site for potential vulnerabilities.

Video | ZAP proxy



OWASP ZAP | Proxy setup

Change proxy settings in Firefox:

- *Preferences* -> *Advanced* -> *Network* -> *Settings*
- Select **Manual proxy configuration**
- Set **HTTP Proxy** to 127.0.0.1, **Port** to 8080
- Set **No Proxy for** to blank

For Chrome you need to use the command-line:

```
google-chrome --proxy-server=host:port
```

A7 - Cross-Site Scripting (XSS)

Was A3

XSS flaws occur when an application includes users supplied data in a page sent to a browser without properly validating or escaping that content.

A7 - Cross-Site Scripting (XSS)

What can you do with XSS?

- Hijack a user's session;
- Hijack a user's browser;
- Deface websites.

A7 - Cross-Site Scripting (XSS)

Two types of XSS:

- Stored;
- Reflected.

A7 - Cross-Site Scripting (XSS)

Example

```
< script>alert(document.cookie);< /script>
```

A7 - Cross-Site Scripting (XSS)

Prevention

- Escape or encode *all* untrusted data;
- Whitelist input validation;
- Use quotes on HTML tag attributes;
- Use a Content Security Policy (CSP) HTTP header.

A7 - Cross-Site Scripting (XSS)

Content Security Policy

- Uses an HTTP header, `Content-Security-Policy`.

```
Content-Security-Policy: default-src 'self'
```

Used by facebook (2014)

```
content-security-policy:  default-src *;script-src https://*.facebook.cc
*.google-analytics.com *.virtualearth.net *.google.com 127.0.0.1:* *.spc
*.atlassolutions.com chrome-extension://lifbcibllhkdhoafpjfnlhfpfgnpldfl
https://*.fbcdn.net http://*.fbcdn.net *.facebook.net *.spotilocal.com:*
https://fb.scanandcleanlocal.com:* *.atlassolutions.com http://attachmer
```

A7 - Cross-Site Scripting (XSS)

Example Tool: BeEF

http://beefproject.com/

- The Browser Exploitation Framework (BeEF) is a testing tool that works through an XSS vulnerability.
- The XSS injects a hook for the client to use. After the client is hooked, the attacker can use the BeEF interface to perform more attacks on the client.

LAB #3

Go to http://[a.b.c.d]/

Work through the Cross-site scripting(XSS) worksheet.

A8 - Insecure Deserialization

Bad news: Insecure deserialization can often lead to remote code execution!

Good news: Often requires a custom attack for successful exploitation.

A8 - Insecure Deserialization

What is serialization?

The process of translating data structures or object state into a format that can be stored or transmitted and reconstructed later.

Processing the serialized version allows creation of a semantically identical clone of the original object.

A8 - Insecure Deserialization

Where is serialization used?

- Remote- and inter-process communications (RPC/IPC)
- Wire protocols, web services, messages brokers
- Caching/Persistence
- Database, cache servers, file systems
- HTTP cookies and form tokens
- API authentication tokens

A8 - Insecure Deserialization

Simple example:

A PHP forum uses PHP object serialization to save a "super" cookie, containing user's ID, role, password hash, etc...

```
a:4:{i:0;i:132;i:1;s:7:"Mallory";i:2;s:4:"user";
  i:3;s:32:"b6a8b3bea87fe0e05022f8f3c88bc960";}
```

Attacker changes it to give themselves admin privileges:

```
a:4:{i:0;i:1;i:1;s:5:"Alice";i:2;s:5:"admin";
  i:3;s:32:"b6a8b3bea87fe0e05022f8f3c88bc960";}
```

A8 - Insecure Deserialization

Prevention | Simple Approach

- Don't use serialization from untrusted sources
- Use serialization mediums that only permit primitive data types
  - protects against remote code execution but not value change

A8 - Insecure Deserialization

Prevention | If you really need to support it

- Do cryptographic integrity checks
- Isolate and run code that deserializes in the lowest privilege environment possible
- Log deserialization exceptions and failures
- Alert repeated deserialization attempts
- Restrict network connectivity from containers or servers that deserialize

A9 - Using Components With Known Vulnerabilities

Need to ensure that we have a complete list of all software components and the ability to determine the install version.

A9 - Using Components With Known Vulnerabilities

Examples

- Operating system?
- Web server?
- Database server?
- Language? (PHP/Python/Perl/Java/etc)
- Framework?
- Application Modules?
- Javascript libraries?

A10 - Insufficient Logging & Monitoring

- Monitoring logs may allow you to discover that you have been hacked.
- If you're not actively monitoring them you may need historical logs after a breach.
- Most breaches are not detected for over six months!

A10 - Insufficient Logging & Monitoring Prevention

- Ensure logs can be digested by a central log management system
  - Provide logs to a central log management system
- Ensure security sensitive things are logged with sufficient user context for analysis, e.g.
  - login and access control failures
  - server-side input validation failures
- Monitor the logs

OWASP Top Ten 2013 - removed in 2017

A8 - Cross-Site Request Forgery (CSRF)

Browsers automatically send credentials like session cookies with any request.

- Significant problem for Windows web sites that support transparent SSO or NTLM authentication

If attacker can persuade a victim's browser to load a particular page, it will be automatically authenticated.

This forged request might transfer money, or change an email-address.

Cross-Site Request Forgery (CSRF)

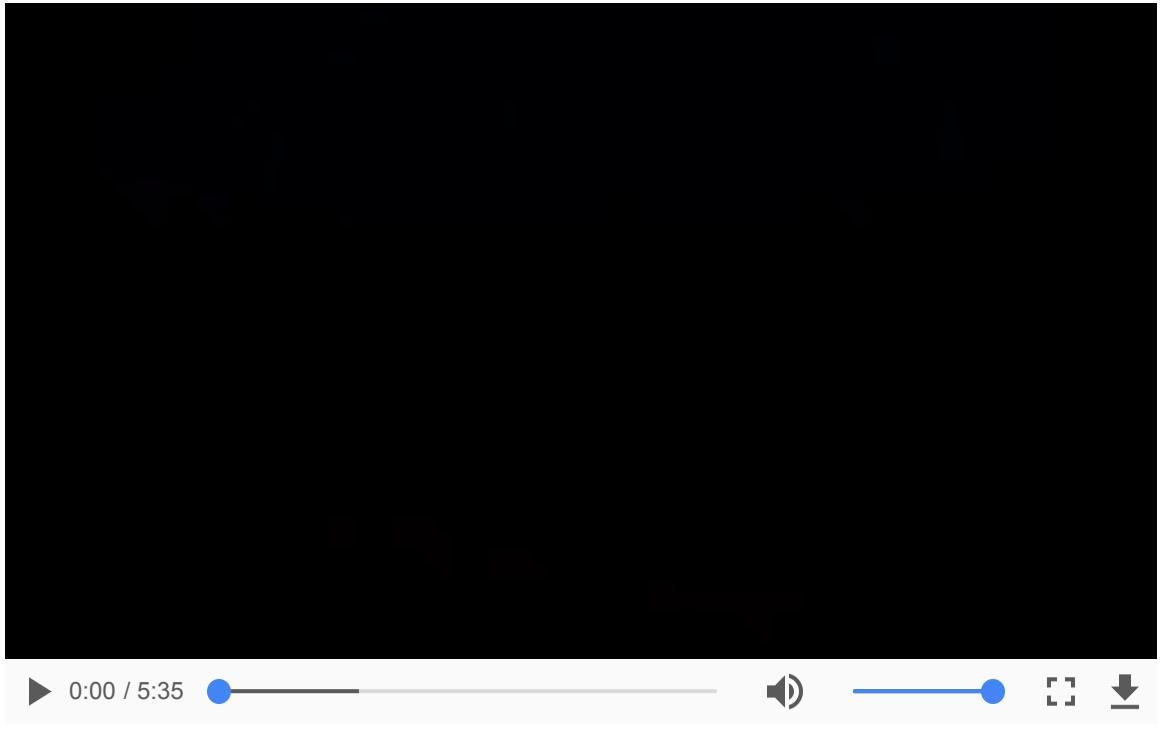
Examples

Add one of the following to http://evil-but-very-cute-kittens.com/

```
<img src="http://otherdomain.bank.com/transfer?to=drevil&amount=100000"
  width="2px" height="2px" style="display:none"/>
```

```
<img src="http://otherdomain.social.com/changepassword?
  newpassword=hackme&confirmpassword=hackme"
  width="2px" height="2px" style="display:none"/>
```

Demo



Cross-Site Request Forgery (CSRF)

Prevention

- Preventing CSRF by making HTTP requests include an unpredictable token, such as in a hidden form field.
  - Token should be a one-time use, or short lived to prevent reuse attacks.
- Confirm the request has come from a human by using a CAPTCHA.
- Ask the user to re-authenticate for particularly sensitive requests.

Cross-Site Request Forgery (CSRF)

Prevention

Most *good* frameworks will have built-in support for CSRF prevention.

Cross-Site Request Forgery (CSRF)

Prevention

Modern *really easy* approach:

Set the 'SameSite' cookie attribute on session cookies.

```
Set-Cookie: sess=abc123; path=/; SameSite
```

The attribute tells the browser not to include the cookie in any cross-origin requests.

Works with Chrome 55+ and related web browsers, such as Opera, and Firefox 58+.

http://caniuse.com/#search=SameSite

Cross-Site Request Forgery (CSRF)

Prevention

Two modes (default is Strict)

```
Set-Cookie: sess=abc123; path=/; SameSite=Strict
Set-Cookie: sess=abc123; path=/; SameSite=Lax
```

The problem with Strict mode is that cross-origin includes changes initiated in the browser, such as changing the URL or opening a new tab and choosing a bookmark.

Lax mode has an exception to allow cookies to be attached to top-level browser navigations using safe HTTP methods (GET, HEAD, OPTIONS and TRACE).

More info: https://scotthelme.co.uk/csrf-is-dead/

LAB #4

Go to http://[a.b.c.d]/

Work through the A9 - PHP Eval worksheet.

Static Website Generators

Lots of options - look at https://www.staticgen.com/

https://wiki.python.org/moin/StaticSiteGenerator

Suitable for lots of public read-only websites.

Dynamic content from APIs called by web browser.

Static Website & OWASP Top Ten (1)

- A1 Injection
  - Nothing to inject against in the website
  - Only third-party APIs
- A2 Broken Authentication
  - No users
  - No authentication
  - No session management

Static Website & OWASP Top Ten (2)

- A3 Sensitive Data Exposure
  - No logins. Should be no sensitive data to be exposed.
  - Can still use HTTPS
- A4 XML External Entity
  - Unlikely to be processing any XML
- A5 Broken Access Controls
  - Could easily host content that should not be public
  - However, access controls not expected so content should not be there

Static Website & OWASP Top Ten (3)

- A6 Security Misconfiguration
  - No dynamic application server to misconfigure
- A7 Cross-Site Scripting
  - No stored cross-site scripting
  - Reflected cross-site scripting only from third-party APIs
- A8 Insecure Deserialization
  - Unlikely to be doing any

Static Website & OWASP Top Ten (4)

- A9 Using Components with Known Vulnerabilities
  - Less components -> less vulnerabilities
- A10 Insufficient Logging & Monitoring
  - Still need it but less
- Old A8 Cross-Site Request Forgery
  - Only a problem with third-party APIs
  - Use a CAPTCHA

HTTP Security Headers

- Strict Transport Security
- Content Security Policy
- X-Frame-Options
- X-XSS-Protection
- X-Content-Type-Options
- Information leakage, such as X-Powered-By
- Caching

Strict-Transport-Security Header

HTTP Strict-Transport-Security (HSTS) enforces secure (HTTP over SSL/TLS) connections to the server.

- HSTS defends against Man-in-the-Middle attacks.
- HSTS disables the ability for a user to ignore SSL negotiation warnings.
- When active, browser automatically changes HTTP links for a site to HTTPS.

HSTS - Supported Browsers

- Chrome/Chromium 4.0.211.0+ (2010)
- Firefox 4+ (2011)
- Opera 12+ (2012)
- Safari OS X Mavericks+ (2013)
- Internet Explorer 11+ (2015)

http://caniuse.com/#feat=stricttransportsecurity

HSTS - Typical implementation

Applies for six months (recommended value) to the domain of the web server, and its sub-domains:

```
Strict-Transport-Security: max-age=15768000; includeSubDomains
```

Six months to two years is recommended.

HSTS - Preloading

Firefox, Opera, Safari, IE11+ all have HSTS preload lists based on the Chrome list. These sites are only available over HTTPS including all sub-domains.

The 'preload' attribute indicates that the domain owner consents to preloading, i.e. this site should already be on the Chrome HSTS preload list.

```
Strict-Transport-Security: max-age=3153600; includeSubDomains; preload
```

https://hstspreload.appspot.com/

Content-Security-Policy

Whitelists allowed sources for web page resources, such as scripts, styles and images.

Specification and use has evolved over time:

- X-Content-Security-Policy - older Firefox and IE versions
- X-Webkit-CSP - older Chrome and Safari versions
- Content-Security-Policy - newer browsers

Simple example:

```
Content-Security-Policy: script-src 'self'
```

Content-Security-Policy - Supported Browsers

- Chrome/Chromium 25+
  - Chrome 4-24 used X-Webkit-CSP
- Firefox 4+
- Internet Explorer Edge+
  - IE 10 and 11 used X-Content-Security-Policy
- Opera 15+
- Safari 7+
  - Safari 6 used X-Webkit-CSP

Content-Security-Policy

Generator

- https://report-uri.com/home/generate
- Enable 'Report only' mode for testing. Send reports to this site.

Clickjacking Protection

Clickjacking typically occurs by loading a web page in an HTML frame or iframe.

The **X-Frame-Options** header tells the web browser where the frame is allowed to come from.

It has been obsoleted by the 'frame-ancestors' directive in CSP 2.

X-Frame-Options Examples

No frames supported:

```
X-Frame-Options: DENY
```

Frames only from the same server:

```
X-Frame-Options: SAMEORIGIN
```

Frames supported from a specified third-party domain:

```
X-Frame-Options: ALLOW-FROM https://example.com/
```

Cross-Site Scripting Protection

Internet Explorer, and some WebKit based browsers, contain cross-site scripting protection that can be disabled!

Override, the disable with:

```
X-XSS-Protection: 1; mode=block
```

The mode can either attempt to sanitise the XSS or block the web browser from rendering the page.

Disable MIME-sniffing

When downloading files from a website, IE and Chrome can ignore the specified content-type.

They will attempt to guess the correct type by sniffing its MIME type, and then launching a suitable handler.

A user may download an executable file and run it without realising.

```
X-Content-Type-Options: nosniff
```

Information Leakage

Common development mode default:

```
Server: Apache/2.2.23 (Unix) mod_ssl/2.2.23 OpenSSL/0.9.8e-fips-rhel5
 mod_auth_passthrough/2.1 mod_bwlimited/1.4 FrontPage/5.0.2.2635
X-Powered-By: PHP/5.2.13
```

Page Caching

We do not want to cache pages from an HTTPS site containing sensitive data.

Lots of HTTP headers that can be set:

```
Cache-Control: no-cache, no-store, must-revalidate
Pragma: no-cache
Expires: -1
```

Evil User Stories

> *“A user story is an objective a person should be able to achieve when using the system.”*

Used in testing. But how can we test what the bad guys do?

Evil User Stories | Examples

- As a hacker, I can send bad data in **URLs**, so that I can access data and functions for which I am not authorized.
- As a hacker, I can send bad data in the **content of requests**, so I can access data and functions for which I am not authorized.
- As a hacker, I can send bad data in **HTTP headers**, so I can access data and functions for which I am not authorized.
- As a hacker, I can read and even modify all data that is **input and output** by your application.

Evil User Stories | Testing

We need to test every sprint.

We need complete coverage of every variable supported to the server, not just the ones we think the user has control over.

We need to test for unusual values (e.g. integer range, string size, unusual characters)

```
< > ( ) & ' " ` \ ; =
```

Evil User Stories | Example

```
SELECT * FROM users WHERE username = '<username>'
 and password = '<password>'
```

```
username: admin
password: ' OR 1=1;--
```

```
SELECT * FROM users WHERE username = 'admin'
 and password = ' ' or 1=1
```

freedom to innovate

open source technologists