Bad Smells That Lead to Bad Security
aka Developing your Security Spidey Senses

Andy Prow – Aura InfoSec
Kirk Jackson - Xero
Intro

Andy Prow
Managing Director
Aura Information Security Ltd

Kirk Jackson
Security Officer
Xero
Spidey Senses

“A vague but strong sense of something being wrong, dangerous or suspicious”

- Should trigger almost immediately.
- Suggests there is something to be wary of or something that needs investigating
- Good hackers have great ones.
- If smart users have one, they run…
Bad smells

- Unlocked Vault – Fishy
- Too Trusting – Sickly sweet
- Spilling your secrets – Vomit
- Back door is open – Garbage
- DIY – Oily rag
- Dodgy foundations – Sewers
- Risky Business - Fear
The Unlocked Vault
Something Smells Fishy…

Our vault contains rotten fish (and heaps of cash) but I shouldn’t be able to smell it outside…

Senses tell me:
Basic security steps are missing entirely.
Examples

- Site mixes HTTP and HTTPS
- Length and character limits for passwords
- Missing security HTTP headers
- Files that expose content (robots.txt, sitemap.xml, VCS etc)
- Cached search engine results
Something Smells Too Sickly Sweet...

My security guard is way too smiley and seems to think his job is let everyone in...

Senses tell me:
Information and actions going into the system are being trusted too openly.
Examples

- Entering [10] invalid passwords, can still log in
- Change password without entering existing password
- Certain characters give errors, or display incorrectly: < ' " ; 😊
- Client side validation of user input
- User content displays directly on screen (e.g. reflected XSS)
- Request contains SQL "select x from y where z"
- No random tokens in the form data (CSRF)
- URLs are in the query string
Spilling your secrets
Something Smells of Vomit...

The security guard has spilt his guts...

Senses tell me:
Information is being leaked out of the system.
Examples

- Signup or password reset email contains a password
- Change password form shows current password
- Site lists characters that are banned in text fields
- Customer id (or similar object reference) in the URL
- A number in the URL that increases by one (or is an obvious format)
- You can tell the underlying technology (e.g. LDAP, SQL, XML)
Back door is open
Something Smells of Garbage...

I’ve been to the back of the bank, and the security stinks...

Senses tell me:
Front-door is good but someone’s left alternative avenues into the system.
Examples

- Sensitive data is unencrypted
- Authorisation - by URL or menu
- Admin site available to the world
- Production data in a test environment
Something Smells of Oily Rags…

This does not smell like a professional job…

Senses tell me:
Some core security features are not using proven techniques but have been home-grown.
Examples

- Doesn't use out of the box authentication
- Custom single-sign-on
- Homebuilt encryption
- Cookie flags, scope (domain + path), expiration
Dodgy Foundations

By Capt' Gorgeous
Something Smells of Sewers…

The bank looks great, but it’s built on a crumbling sewer.

Senses tell me:
There is a poor underlying architecture or technology beneath this system.
Examples

- Multiple apps, including third party
- A mix of technologies (e.g. PHP + ASP.NET)
- Cross-domain javascript / CSS
- Rich Client component (e.g. Flash, Silverlight, JS)
- Old OS, server, ASP.NET, PHP, Java version
- Writing directly to disk
Something Smells of Fear…

Smells like someone’s taken a risk and implemented something dangerous.

Senses tell me:
Features requiring strong security have not been implemented securely enough.
Examples

- Application accepts file uploads
- Parsing XML, HTML or other file formats
- Credit card payments
- Mobile app talking to API
Billion Laughs

&m;
Billion Laughs

&l;&l;&l;&l;&l;&l;&l;&l;&l;
Billion Laughs
Billion Laughs
Billion Laughs

Boom!
Bad smells

- Unlocked Vault – Fishy
- Too Trusting – Sickly sweet
- Spilling your secrets – Vomit
- Back door is open – Garbage
- DIY – Oily rag
- Dodgy foundations – Sewers
- Risky Business - Fear
<table>
<thead>
<tr>
<th>A1 - Injection</th>
<th>Injection flaws, such as SQL, ODBC, and LDAP injection occur when untrusted data is sent to an interpreter or as part of a command or query. The attacker's hostile data can trick the interpreter into executing unintended commands or accessing data without proper authorization.</th>
</tr>
</thead>
<tbody>
<tr>
<td>A2 - Broken Authentication and Session Management</td>
<td>Application functions related to authentication and session management are often not implemented correctly, allowing attackers to compromise passwords, keys, or session tokens, or to exploit other implementation flaws to assume other users' identities.</td>
</tr>
<tr>
<td>A3 - Cross-Site Scripting (XSS)</td>
<td>XSS flaws occur whenever an application sends untrusted data and sends it to a web browser without proper validation or escaping. XSS attacks allow attackers to execute scripts in the victim's browser which can hijack user sessions, deface web sites, or redirect the user to malicious sites.</td>
</tr>
<tr>
<td>A4 - Insecure Direct Object References</td>
<td>A direct object reference occurs when a developer exposes a reference to an internal implementation object, such as a file, directory, or database key. Without an access control check or other protection, attackers can manipulate these references to access unauthorized data.</td>
</tr>
<tr>
<td>A5 - Security Misconfiguration</td>
<td>Good security requires having a secure configuration defined and deployed for the application, framework, application server, web server, database server, and platform. Secure settings should be defined, implemented, and maintained, as defaults are often insecure. Additionally, software should be kept up to date.</td>
</tr>
<tr>
<td>A6 - Sensitive Data Exposure</td>
<td>Many applications and APIs do not properly protect sensitive data, such as credit cards, tax IDs, and authentication credentials. Attackers may steal or modify such weakly protected data to conduct credit card fraud, identity theft, or other crimes. Sensitive data deserves extra protection such as encryption at rest or in transit, as well as special precautions when exchanged with the browser.</td>
</tr>
<tr>
<td>A7 - Missing Function Level Access Control</td>
<td>Most web applications verify function level access rights before granting that functionality visible in the UI. However, applications need to perform the same access control checks on the server when each function is accessed. If requests are not verified, attackers will be able to forge requests in access functionality without proper authorization.</td>
</tr>
<tr>
<td>A8 - Cross-Site Request Forgery (CSRF)</td>
<td>A CSRF attack forces a logged-on victim's browser to send a forged HTTP request, including the victim's session cookie and any other automatically included authentication information, to a vulnerable web application. This allows the attacker to force the victim's browser to generate requests to the vulnerable application that appear to originate from the victim.</td>
</tr>
<tr>
<td>A9 - Using Components with Known Vulnerabilities</td>
<td>Components, such as libraries, frameworks, and other software modules, almost always run with full privileges. If a vulnerable component is exploited, such an attack can facilitate various data theft or server takeover. Applications using components with known security vulnerabilities may undermine application defenses and enable a range of possible attacks and reports.</td>
</tr>
<tr>
<td>A10 - Unvalidated Redirects and Forwards</td>
<td>Web applications frequently redirect and forward users to other pages and websites, and use untrusted data to determine the destination pages. Without proper validation, attackers can redirect victims to phishing or malware sites, or use forwards to access unauthorized pages.</td>
</tr>
</tbody>
</table>
Summary

- Develop your own Spidey-Senses
- OWASP Top 10 2013 owasp.org
- Publications on www.aurainfosec.com
- View our other talks at hack-ed.com
Q&A...