David Rook

The Principles of Secure Development

OWASP Ireland Conference, Dublin

Friday, 18 September 2009
if (slide == introduction)
    System.out.println("I’m David Rook");

• Security Analyst, Realex Payments, Ireland
  CISSP, CISA, GCIH and many other acronyms

• Security Ninja (www.securityninja.co.uk)

• Speaker at security events (national and international)

• IIA Web Development Working Group

• Facebook hacker and published security author (insecure magazine, bloginforesec etc)
Agenda

- It is broken so let's fix it
- The current approach
- The Principles of Secure Development
- An example of a real world implementation
It is broken so let's fix it

• Cross Site Scripting, 10 years old?

• SQL Injection, 11 years old?

33% of all vulnerabilities in 2008 and 2009 (so far) are XSS or SQL Injection (Based on CVE numbers)


Friday, 18 September 2009
It is broken so let's fix it
It is broken so let's fix it.
It is broken so let's fix it

SQLi & XSS = 32.24%
Give a man a fish and you feed him for a day, teach him to fish and you feed him for a lifetime.
Give a man a fish and you feed him for a day, teach him to fish and you feed him for a lifetime.

I want to apply this to secure application development:

Teach a developer about a vulnerability and he will prevent it, teach him how to develop securely and he will prevent many vulnerabilities.
The current approach
(And why I think it fails to deliver secure applications)

• The cart before the horse

  • Security guys tell developers about specific vulnerabilities
  • We hope they figure out how to prevent them
  • Inevitably security flaws end up in live code
  • Security guys complain when data gets stolen
The current approach
(And why I think it fails to deliver secure applications)

• What if we taught drivers in the same way?
  • Instructor tells driver about the different ways to crash
  • We hope the driver figures out how not to crash
  • Inevitably the driver will crash
  • People complain when they get crashed into
The current approach
(And why I think it fails to deliver secure applications)

- Many lists of vulnerabilities
  - OWASP Top 10
  - White Hat Sec Top 10
  - SANS Top 25
  - Others??

- != Secure development guidance
The current approach
(And why I think it fails to deliver secure applications)

- Many lists of vulnerabilities
  - OWASP Top 10
  - White Hat Sec Top 10
  - SANS Top 25
  - Others??

- != Secure development guidance
- 45 vulnerabilities, 42 unique names
- 8 secure coding principles to prevent them
What we need to do

- Put the application security horse before the cart
  - Security guys tell developers how to write secure code
  - Developer doesn’t need to guess anymore
  - Common vulnerabilities prevented in applications
  - Realistic or just a caffeine fueled dream?
Let's make secure development easier

- Keep It Short and Simple (KISS)
  - The principles must be clearly defined
  - Language/Platform/Framework independent
  - Should cover more than just the common vulnerabilities
  - More secure software and greater ROI on security training?
The Principles of Secure Development

SECURITY

Input Validation
Output Validation
Error Handling
Authentication and Authorisation
Session Management
Secure Communications
Secure Storage
Secure Resource Access
The Principles of Secure Development

SECURITY

• Input Validation

  • Identify and define the data your application must accept
The Principles of Secure Development

SECURITY

• Input Validation
  • Identify and define the data your application must accept
  • Create regex’s to validate each data type (content and size)
  • For example, a credit card number data type: \d{12,16}$_
The Principles of Secure Development

SECURITY

• Input Validation
  
  • Identify and define the data your application must accept
  • Create regex’s to validate each data type (content and size)
  • For example, a credit card number data type: \d{12,16}$
  • Use whitelisting for validation where possible
The Principles of Secure Development

SECURITY

• Input Validation

• Identify and define the data your application must accept
• Create regex’s to validate each data type (content and size)
• For example, a credit card number data type: \d{12,16}$
• Use whitelisting for validation where possible
• Blacklisting approach harder and potentially less secure
• Blacklist example, replacing single quotes:

```java
s.replaceAll(Pattern.quote("\""),
Matcher.quoteReplacement("\"\""))
```
The Principles of Secure Development

SECURITY

• Output Validation

  • Identify and define the data your application must output
The Principles of Secure Development

SECURITY

• Output Validation

  • Identify and define the data your application must output
  • Understand where (i.e. in a URL) your data should end up
  • Choose the correct output encoding for the data's destination
The Principles of Secure Development

SECURITY

• Output Validation
  • Identify and define the data your application must output
  • Understand where (i.e. in a URL) your data should end up
  • Choose the correct output encoding for the data's destination
  • Proper encoding means this attack:

  
  www.exampleSite.com/home.html?day=<script>alert(document.cookie)</script>

  Becomes:

  
  day=%3Cscript%3Ealert%28document.cookie%29%3C/script%3E
The Principles of Secure Development

SECURITY

• Error Handling

  • Even the best apps will crash at some point, be prepared!
The Principles of Secure Development

SECURITY

• Error Handling

• Even the best apps will crash at some point, be prepared!
• Crashes/errors can help an attacker if you don’t handle them
The Principles of Secure Development

SECURITY

• Error Handling

• Even the best apps will crash at some point, be prepared!
• Crashes/errors can help an attacker if you don’t handle them
• Handle error conditions securely, sanitise the message sent
The Principles of Secure Development

SECURITY

• Error Handling

• Even the best apps will crash at some point, be prepared!
• Crashes/errors can help an attacker if you don’t handle them
• Handle error conditions securely, sanitise the message sent
• No error handling = information leakage

Microsoft OLE DB Provider for ODBC Drivers (0x80040E14)
[Microsoft][ODBC SQL Server Driver][SQL Server]Invalid column name

/example site/login.asp, line 10
The Principles of Secure Development

SECURITY

• Authentication and Authorisation
  
  • Even simple apps often have a need to authenticate users
The Principles of Secure Development

SECURITY

• Authentication and Authorisation
  • Even simple apps often have a need to authenticate users
  • Often at least two levels of authorisation
The Principles of Secure Development

SECURITY

• Authentication and Authorisation
  • Even simple apps often have a need to authenticate users
  • Often at least two levels of authorisation
  • Need to prevent horizontal and vertical privilege escalation
The Principles of Secure Development

SECURITY

- Authentication and Authorisation
  - Even simple apps often have a need to authenticate users
  - Often at least two levels of authorisation
  - Need to prevent horizontal and vertical privilege escalation
  - Implement strong passwords and management systems

Friday, 18 September 2009
The Principles of Secure Development

SECURITY

• Authentication and Authorisation
  • Even simple apps often have a need to authenticate users
  • Often at least two levels of authorisation
  • Need to prevent horizontal and vertical privilege escalation
  • Implement strong passwords and management systems
  • Ensure A+A is secure, not a false sense of security (CAPTCHA?)
The Principles of Secure Development

SECURITY

• Authentication and Authorisation

  • Even simple apps often have a need to authenticate users
  • Often at least two levels of authorisation
  • Need to prevent horizontal and vertical privilege escalation
  • Implement strong passwords and management systems
  • Ensure A+A is secure, not a false sense of security (CAPTCHA?)
  • Don’t rely on fields that are easily spoofed (referrer field)
The Principles of Secure Development

Session Management

- Used to manage authenticated users, no need to re-auth
The Principles of Secure Development

SECURITY

• Session Management

  • Used to manage authenticated users, no need to re-auth
  • You need to ensure that your sessionID’s have sufficient entropy
The Principles of Secure Development

SECURITY

• Session Management

• Used to manage authenticated users, no need to re-auth
• You need to ensure that your sessionID’s have sufficient entropy
• SessionID’s must not be predictable or reusable
• Session Management

• Used to manage authenticated users, no need to re-auth
• You need to ensure that your sessionID’s have sufficient entropy
• SessionID’s must not be predictable or reusable
• Never build your own session management, it will fail
The Principles of Secure Development

Session Management

- Used to manage authenticated users, no need to re-auth
- You need to ensure that your sessionID’s have sufficient entropy
- SessionID’s must not be predictable or reusable
- Never build your own session management, it will fail
- Protect sessionID’s when in transit (i.e. SSL!)
The Principles of Secure Development

**Session Management**

- Used to manage authenticated users, no need to re-auth
- You need to ensure that your sessionID’s have sufficient entropy
- SessionID’s must not be predictable or reusable
- Never build your own session management, it will fail
- Protect sessionID’s when in transit (i.e. SSL!)
- Issue a new value for sensitive actions (i.e. funds transfer)
The Principles of Secure Development

SECURITY

• Secure Communications

  • Protect data (i.e. CC no, passwords, sessionID’s) in transit
The Principles of Secure Development

SECURITY

• Secure Communications

  • Protect data (i.e. CC no, passwords, sessionID’s) in transit
  • As with all crypto, don’t create your own
Secure Communications

- Protect data (i.e. CC no, passwords, sessionID’s) in transit
- As with all crypto, don’t create your own
- Don’t use broken protection mechanisms (i.e. SSLv2)
The Principles of Secure Development

SECURITY

• Secure Communications

  • Protect data (i.e. CC no, passwords, sessionID’s) in transit
  • As with all crypto, don’t create your own
  • Don’t use broken protection mechanisms (i.e. SSLv2)
  • Don’t just use SSL/TLS for logon pages, protect the session!
Secure Communications

- Protect data (i.e. CC no, passwords, sessionID’s) in transit
- As with all crypto, don’t create your own
- Don’t use broken protection mechanisms (i.e. SSLv2)
- Don’t just use SSL/TLS for logon pages, protect the session!
- Try to avoid mixing secure and insecure traffic on a page
The Principles of Secure Development

SECURITY

• Secure Storage

  • Protect data (i.e. CC no, passwords, sessionID’s) when stored
The Principles of Secure Development

SECURITY

• Secure Storage

  • Protect data (i.e. CC no, passwords, sessionID’s) when stored
  • As with all crypto, don’t create your own
The Principles of Secure Development

SECURITY

• Secure Storage

  • Protect data (i.e. CC no, passwords, sessionID’s) when stored
  • As with all crypto, don’t create your own
  • Don’t use broken protection mechanisms (i.e. DES)
The Principles of Secure Development

SECURITY

- Secure Storage
  - Protect data (i.e. CC no, passwords, sessionID’s) when stored
  - As with all crypto, don’t create your own
  - Don’t use broken protection mechanisms (i.e. DES)
  - Don’t store data in places where you can’t confidently secure it
The Principles of Secure Development

SECURITY

• Secure Storage
  
  • Protect data (i.e. CC no, passwords, sessionID’s) when stored
  • As with all crypto, don’t create your own
  • Don’t use broken protection mechanisms (i.e. DES)
  • Don’t store data in places where you can’t confidently secure it
  • Strong protection mechanisms, how strong should it be?
The Principles of Secure Development

Friday, 18 September 2009

```javascript
var valid_codes = new Array();
valid_codes[0] = 'b50339a10e1de285ac99d4c3990b8693:357';
valid_codes[1] = '3164d90f7e8107290b44c423e735f264:360';
valid_codes[2] = '3907192d4e4c7dc5f2a858ea07097c62:361';
valid_codes[3] = '689f1db9349ec76ef0c295b5e23cd1a:362';
valid_codes[4] = '17e7245eced7cb9b541511c4baa5bb14:363';
valid_codes[5] = '85c0039ec9dd90329aa27167fcda4c88:364';
valid_codes[6] = 'f65d7bcf3a814ed5cc3b48127a72cf:365';
valid_codes[7] = '7d4b18a3fcdde1c4edcdd09668ff0e8:366';
valid_codes[8] = 'a1e768492d70531e2205e44f64d4ffb:367';
valid_codes[9] = 'db6f9c051d7f8c4641ce166208239051:368';
valid_codes[10] = 'f4a4b34cf660ac92128868854c879f1dc:369';
valid_codes[12] = 'dbd3fd41b442624ebcfee51da44ed6f:371';
valid_codes[13] = '1afea6b23b96e2da9e9edc937cfa1ba8:372';
valid_codes[14] = '22c83facdbc2819d7cf7109ea220e0a:373';
valid_codes[15] = 'ce4b27a32419aaf3f1cd2d235c804777:374';
valid_codes[16] = '4aa592f7db9e5ce0d21251839f28d647:375';
valid_codes[17] = '24e47da5ddc94d38441a3ac8fa16f95d:376';
valid_codes[18] = '63df7661fba67b75f9fd052c8a2b6d08:377';
valid_codes[19] = '0a927cc69f8273be0cc0acdb1b9acb7:378';
valid_codes[20] = '8e9866383fe99765c23a6952bf580548:379';
valid_codes[21] = '2ab87df7a6de657a8b1211a2545f8fc:380';
valid_codes[22] = 'ba9af4260c964d9cfdd48ac3366119e:381';
valid_codes[23] = '858e8999193647650191c9cfaa36ae:382';
valid_codes[24] = '32d0b92d11ac680fb3a30356e27161fc:383';
valid_codes[25] = '447842e7b999367b64d31c6b927cb587:384';
valid_codes[26] = 'e7e0092245f990a1c44621027146d0c8:385';
valid_codes[27] = '1785a5f480defa0075c21965ab472b95:386';
```
The Principles of Secure Development

- Secure Resource Access
  - Obscurity ≠ security, don’t try to hide sensitive resources
The Principles of Secure Development

SECURITY

• Secure Resource Access
  • Obscurity ≠ security, don’t try to hide sensitive resources
  • Understand the users flow through an app, cover weak spots
The Principles of Secure Development

SECURITY

• Secure Resource Access
  • Obscurity != security, don’t try to hide sensitive resources
  • Understand the users flow through an app, cover weak spots
  • T-Mobile didn’t do the above, Paris Hiltons account hacked
Lets redefine what secure development means

SECURITY

• Follow a small, repeatable set of principles

• Try not to focus on specific vulnerabilities

• Develop securely, not to prevent "hot vuln of the day"

• Build security into the code, don’t try to bolt it on at the end
Evolution, not revolution

- Don’t make things more difficult than they need to be
  - This isn’t a new wheel, it’s just a smoother, easier to use wheel
  - Don’t treat security as something separate, integrate it
  - By integrating security fully a security bug is just another bug
  - Secure development doesn’t have to be hard, KISS it!
The new approach is working

SECURITY

- Private banking development company, Switzerland
  - Application Security lead saw the secure development principles
The new approach is working

• Private banking development company, Switzerland
  • Application Security lead saw the secure development principles
  • Re-designed secure development training for his company
The new approach is working

SECURITY

• Private banking development company, Switzerland
  • Application Security lead saw the secure development principles
  • Re-designed secure development training for his company
  • Security training costs down, quicker "spin up" of developers
The new approach is working

SECURITY

- Private banking development company, Switzerland
  - Application Security lead saw the secure development principles
  - Re-designed secure development training for his company
  - Security training costs down, quicker "spin up" of developers
  - Security within their SDLC now based on the principles
The new approach is working

• Private banking development company, Switzerland
  • Application Security lead saw the secure development principles
  • Re-designed secure development training for his company
  • Security training costs down, quicker "spin up" of developers
  • Security within their SDLC now based on the principles
  • In his own words:

You released the "secure development principles" at a time I had issues with my dev teams in how to teach them secure development. **Your approach convinced me to look in another direction, not trying to teach every vulnerability but finding the basic principles that help prevent their existence.** At that time, this was genius for me: most of my training since has been inspired by your secure development principles.
The new approach is working

They modified the principles matrix to match their own terminology

<table>
<thead>
<tr>
<th>Principles</th>
<th>Specific vulnerabilities for each principle</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>OWASP</td>
</tr>
<tr>
<td>Input Validation</td>
<td>Cross Site Scripting, Injection Flaws, Malicious File Execution</td>
</tr>
<tr>
<td>Output Validation</td>
<td>Cross Site Scripting</td>
</tr>
<tr>
<td>Error Handling</td>
<td>Information Leakage and Improper Error Handling</td>
</tr>
<tr>
<td>Session Management</td>
<td>Broken Authentication and Session Management, Cross Site Request Forgery</td>
</tr>
<tr>
<td>Secure Communications</td>
<td>Insecure Communications</td>
</tr>
<tr>
<td>Secure Resource Access</td>
<td>Insecure Direct Object Reference, Failure to Restrict URL Access</td>
</tr>
<tr>
<td>Secure Storage</td>
<td>Insecure Cryptographic Storage,</td>
</tr>
</tbody>
</table>

* - based on description from WhiteHatSec
** - based on description from Sans/CWE

Code Security Flaw Matrix version 2.0
April 2009
David Rook
www.securityninjaco.uk

Friday, 18 September 2009
The new approach is working

They modified the principles matrix to match their own terminology

<table>
<thead>
<tr>
<th>Development principle</th>
<th>Clues</th>
<th>OWASP</th>
<th>WhiteHatSec</th>
<th>SANS Top 25</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Input validation</td>
<td>Know your entry points&lt;br&gt;- Validate all input&lt;br&gt;- Validate at the server-side&lt;br&gt;- White list is EXCELLENT&lt;br&gt;- Regex is GOOD&lt;br&gt;- Blacklist is WEAK</td>
<td>Injection flaws, Malicious file execution</td>
<td>Content spoofing, SQL injection, HTTP response splitting</td>
<td>Improper input validation, failure to preserve SQL structure, failure to preserve OS command structure, failure to constrain operations within the bounds of a memory buffer, external control of critical state data, untrusted search path, external control of file name and path, failure to control generation of code, download of code without integrity check, incorrect calculation, client-side enforcement of server-side security</td>
</tr>
<tr>
<td>2. Output encoding</td>
<td>- Webapps: encode for HTML, Javascript, XML&lt;br&gt; - Encode all exit points (system, OS, email, T2M, third-party, PDF, office, etc.)</td>
<td>Cross-site scripting</td>
<td>Cross-site scripting</td>
<td>Improper escaping or encoding of output, failure to preserve web page structure</td>
</tr>
<tr>
<td>3. Secure failure</td>
<td>Never display error messages, generate ticket instead and log error.&lt;br&gt;- Use fail-safe logic (if/else -&gt; default is secure)&lt;br&gt; - Open design: a hacker should read our specs without danger</td>
<td>Information leakage and improper error handling</td>
<td>Information leakage</td>
<td>Error message information leak</td>
</tr>
<tr>
<td>4. Authentication and authorization hardening</td>
<td>Require authorization even if the ‘URL’ is known&lt;br&gt;- Authorize at UI layer, then authorize discretely at business layer&lt;br&gt; - Prevent horizontal escalation: what if another ‘ID’ is used?&lt;br&gt; - Password recovery: authenticate before starting procedure&lt;br&gt; - NO CUSTOM authentication/authorization managers!!!&lt;br&gt; - Authenticate users AND data (ACLs and configuration file integrity)</td>
<td>Insecure direct object reference, broken auth. management, failure to restrict URL access</td>
<td>Predictable resource location, insufficient authentication, insufficient authorization</td>
<td>Improper access control, execution with unnecessary privileges, insecure permission assignment for critical resources</td>
</tr>
<tr>
<td>5. Session hardening</td>
<td>Don’t confuse identification (“saying who she is”) and authentication (“proving who she is”)&lt;br&gt;- NO CUSTOM session managers!!!&lt;br&gt; - Session lifetime&lt;br&gt; - Issue new IDs when appropriate (sensitive ops)&lt;br&gt; - Protect session store&lt;br&gt; - Cookies: Secure + htaccess&lt;br&gt; - Use anti-autamation mechanisms: userkey viewstate is OKAY for non-sensitive&lt;br&gt; - captcha for sensitive&lt;br&gt; - token for critical</td>
<td>Cross-site request forgery, broken session management</td>
<td>Session fixation, Cross-site request forgery</td>
<td>Cross-site request forgery</td>
</tr>
<tr>
<td>6. Secrecy of sleeping and traveling data</td>
<td>- Use the standard API (no calls to system.security.cryptography) for hashing and encryption&lt;br&gt; - don’t send credentials, prove you know them&lt;br&gt; - don’t send keys (use key exch.)&lt;br&gt; - protect keys by master key and don’t store MKey&lt;br&gt; - protect in-memory access (securestrings + DAP!)&lt;br&gt; - if https, don’t allow http -&gt; kill session if detected.&lt;br&gt; - check with SO when encryption is used</td>
<td>Insecure cryptographic storage, insecure communications</td>
<td>Cleartext transmission of sensitive information, use of broken or risky cryptographic algorithm, hard-coded password, use of insufficiently random values</td>
<td></td>
</tr>
<tr>
<td>7. Traceability</td>
<td>Trace all business cases (WHO did WHAT from WHERE and WHEN)</td>
<td></td>
<td>Abuse of functionality</td>
<td></td>
</tr>
<tr>
<td>8. Economy of mechanisms and resources</td>
<td>Only allocate when needed&lt;br&gt; - beware of session state size&lt;br&gt; - beware of serialization cascades&lt;br&gt; - deallocate resources ASAP&lt;br&gt; - beware of DB pooling</td>
<td></td>
<td>Improper resource shutdown or release, improper initialization</td>
<td></td>
</tr>
</tbody>
</table>

Coverage: 100% 100% 96%

Uncovered vulnerabilities: Directory indexing (config.) Race conditions
Security Ninja new site launch!

- Security Ninja, brought you by Realex Payments
  - Free application security and compliance resource site
Security Ninja new site launch!

• Security Ninja, brought you by Realex Payments
  • Free application security and compliance resource site
  • Blog and site managed and updated by myself
Security Ninja new site launch!

- Security Ninja, brought you by Realex Payments
  - Free application security and compliance resource site
  - Blog and site managed and updated by myself
  - Security presentations, whitepapers, videos and audio online

Friday, 18 September 2009
Security Ninja new site launch!

• Security Ninja, brought you by Realex Payments

• Free application security and compliance resource site
• Blog and site managed and updated by myself
• Security presentations, whitepapers, videos and audio online
• Secure Development Principles whitepaper available here today
QUESTIONS?
www.securityninja.co.uk
Twitter: @securityninja