Software security en Architectuur

Steven van der Baan
Sogeti

#EW2012
attacker
security controls & mitigations
assets
influence	no influence
CSRF
S1: Use of Potentially Dangerous Function
S2: Improper Neutralization of Special Elements used in an OS Command
S3: Improper Neutralization of Input During Web Page Generation
A2: Cross-Site Scripting
S4: Unrestricted Upload of File with Dangerous Type
S5: Cross-Site Request Forgeries
S6: URL Redirection to Untrusted Site
A5: Cross-Site Request Forgeries
S7: Buffer Copy without Checking Size of Input
S8: Improper Limitation of a Pathname to a Restricted Directory
S9: Download of Code Without Integrity Check
S10: Inclusion of Functionality from Untrusted Control Sphere
S11: Use of Hard-coded Credentials
S12: Incorrect Calculation of Buffer Size
W1: Information Leakage
W2: Cross-Site Scripting
W3: Content Spoofing
W4: Cross-Site Request Forgeries
W5: Brute Force
W6: Inufficient Authorisation
W7: Predictable Resource Location
W8: SQL Injection
W9: Session fixation
A3: Broken Authentication and Session Management
A4: Insecure Direct Object References
A5: Cross-Site Request Forgeries
A6: Security Misconfiguration
A7: Insecure Cryptographic Storage
A8: Failure to Restrict URL Access
A9: Inufficient Transport Layer Protection
A10: Unvalidated Redirects and Forwards
A11: Use of a Broken or Risky Cryptographic Algorithm
A12: Execution with Unnecessary Privileges
S13: Uncontrolled Format String
S14: Integer Overflow or Wraparound
S15: Missing Authentication for Critical Function
S16: Missing Authorization/Path Traversal
S17: Use of Hard-coded Credentials
S18: Missing Encryption of Sensitive Data
S19: Reliance on Untrusted Inputs in a Security Decision
S20: Execution with Unnecessary Privileges
S21: Incorrect Authorization
S22: Incorrect Permission Assignment for Critical Resource
S23: Use of a Broken or Risky Cryptographic Algorithm
S24: Improper Restriction of Excessive Authentication Attempts
S25: Use of a One-Way Hash without a Salt
S26: Buffer Copy without Checking Size of Input
S27: Error Conditions upon Request for Information
S28: Insecure Deserialization
S29: Injection
S30: Execution with Unnecessary Privileges
S31: Insecure Direct Object References
S32: Use of Hard-coded Credentials
S33: Improper Neutralization of Input During Web Page Generation
S34: Cross-Site Request Forgeries
S35: Insecure Cryptographic Storage
S36: Improper Neutralization of Special Elements used in an OS Command
S37: Insecure Cryptographic Storage
S38: Insecure Cryptographic Storage
**9. Repudiation**
An attacker can deny connection to a server or peer over a link that isn't authenticated (and encrypted).

**7. Spoofing**
An attacker can connect to a server or peer over a link that isn't authenticated (and encrypted).

**K. Information Disclosure**
An attacker can read network information because there's no cryptography used.

**10. Denial of Service**
An attacker can make a server or resource unavailable or unusable ever authenticating an problem persists after attacker goes away.

**J. Tampering**
An attacker can write to some resource because permissions are granted to the world or there are no ACLs.
<table>
<thead>
<tr>
<th>Coding flaw categories</th>
<th>OWASP</th>
<th>WhiteHatSec</th>
<th>Sans</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input Validation</strong></td>
<td>Cross Site Scripting, Injection Flaws, Malicious File Execution</td>
<td>Cross Site Scripting, SQL Injection, Content Spoofing*</td>
<td>Improper Input Validation, Failure to Preserve SQL Query Structure, Failure to Preserve Web Page Structure, Failure to Preserve OS Command Structure, Failure to Constrain Operations within the Bounds of a Memory Buffer, Failure to Control Generation of Code**, Client-Side Enforcement of Server-Side Security**</td>
</tr>
<tr>
<td><strong>Output Encoding</strong></td>
<td>Cross Site Scripting</td>
<td>Cross Site Scripting</td>
<td>Improper Encoding or Escaping of Output, Failure to Preserve Web Page Structure</td>
</tr>
<tr>
<td><strong>Error Handling</strong></td>
<td>Information Leakage and Improper Error Handling</td>
<td>Information Leakage</td>
<td>Error Message Information Leak</td>
</tr>
<tr>
<td><strong>Session Management flaws</strong></td>
<td>Broken Authentication and Session Management, Cross Site Request Forgery</td>
<td>Cross Site Request Forgery</td>
<td>Cross Site Request Forgery, Use of Insufficiently Random Values**</td>
</tr>
<tr>
<td><strong>Insecure Communications</strong></td>
<td>Insecure Communications</td>
<td></td>
<td>Use of a Broken or Risky Cryptographic Algorithm, Cleartext Transmission of Sensitive Information, Use of Insufficiently Random Values**</td>
</tr>
<tr>
<td><strong>Insecure Storage</strong></td>
<td>Insecure Cryptographic Storage,</td>
<td></td>
<td>Use of a Broken or Risky Cryptographic Algorithm, Cleartext Transmission of Sensitive Information, External Control of Critical State Data**</td>
</tr>
<tr>
<td><strong>Insecure Resource Access</strong></td>
<td>Insecure Direct Object Reference, Failure to Restrict URL Access</td>
<td>Predictable Resource Location</td>
<td>External Control of File Name or Path, Untrusted Search Path</td>
</tr>
<tr>
<td><strong>Others</strong></td>
<td>HTTP Response Splitting</td>
<td></td>
<td>Race Condition, Download of Code Without Integrity Check, Improper Resource Shutdown or Release, Improper Initialization, Incorrect Calculation</td>
</tr>
</tbody>
</table>

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

Code Security Flaw Matrix version 1.0
18th January 2009
David Rook
www.securityninja.co.uk
Principles

- Input Validation
- Output Encoding
- Error Handling
- Authentication and Authorisation
- Session Management
- Secure Communications
- Secure Storage
- Secure Resource Management
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(""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
    ```
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.example.com/home?day=<script>alert(document.cookie)</script>
  Becomes:
  day=%3Cscript%3Ealert%28document.cookie%29%3C/script%3E
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 Driver(0x80040E14)
[Microsoft][ODBC SQL Server Driver] [SQL Server]Invalid column name

/example/login.asp, line 10
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)
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)
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
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?
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 got hacked
Links

- owasp.org

- microsoft.com/security/sdl/adopt/eop.aspx
SOGETI
staat voor resultaat