• Gabriele Guasco

Senior Consultant - Security Governance and Ethical Hacking
What’s in this Security Sandwich?

- define Security requirements at the start
- write your code ... faster than light
- do some security tests at the end

Application Security
Ensuring Application guarantees
- Confidentiality
- Integrity
- Availability
- Accountability of the Information it processes

https://www.thoughtworks.com/radar/techniques/security-sandwich
why this sandwich is not so good?
Usually it’s just half-a-sandwich

no security design

no security tests
Does not keep up with Agile processes

where new **Design choices**, and even **Requirements**, emerge through the project lifespan

and this includes their **impact on Security**
When problems are discovered...

there is no more time ...or no more $$$

# of vulnerabilities

theory

bug fixing starts after final PenTest near planned release date

sad reality
The more time passes...

...the more cost-to-fix increases

- because the change implies revising components written months before
- possibly by developers no more with the team
- because the complexity of the project has increased
- steep cost / delay increase after release to production
With the *security sandwich* it’s often too late
Continuous Security

Embed Security validation (bread) across the entire Software Development Lifecycle
Continuous Security - Analysis & Design

At project start
- evaluate Risk Level and potential threats (Threat Model)
- define high level Security Requirements and guidelines
- evaluate Team need for Security Training

High level Architecture Review
- ask for security expert as an advisor
- good ROI because you can solve root cause on many security problems in design phase

Design Reviews
- when changing security sensitive parts
Continuous Security - Implementation

Detail Design and Implementation
- follow **Secure Coding Principles**

At first “vertical slice” prototype
- Vulnerability Assessment & **Pen Test**
- check for **implementation errors**
on design decisions,
- **avoid errors in future** implementation

add feature / component 1
- **non-regression** security tests

... add feature / component N
- **non-regression** security tests

**Secure Coding Principles**
Do not trust inputs
Minimize attack surface area (and window of opportunity)
Establish secure defaults
Principle of Least privilege
Principle of Defense in depth
Fail securely
Don’t trust services
Separation of duties
Avoid security by obscurity
Keep security simple
Fix security issues correctly
If you can't protect, detect
Get your users involved
Continuous Security - Test & Production

At pre-production stage
- **full Vulnerability Assessment** & Pen Test
- pre-allocate time for final fixing

In production
- log **application - level Security Events**
  - failed logins, unauthorized requests
  - accesses from unusual clients
- **Keep Users Involved**
  - give them information needed to detect potential threats

Updates and new releases
- **non-regression** security tests
Why is this better?

Identify Design **flaws** and vulnerabilities **as they occur**

Easier (and cheaper) to fix within their **context**

<table>
<thead>
<tr>
<th>There are 15 vulnerabilities in your 50kloc codebase</th>
<th>VS</th>
<th>There is 1 vulnerability in feature X that you committed yesterday</th>
</tr>
</thead>
</table>

The team constantly **learns** about security issues and fixes and can apply this experience in the rest of the project

| Nice VA report, but starting tomorrow I’ll move to another project | VS | That’s interesting… we’ll avoid this in the next features |
Security sandwich approach

lines of code

security bugs

security bugs to be fixed ... in no time :-)

end-of-project vulnerability assessment
Continuous Security: tests & fix during SDLC

- Lines of code
- Security bugs
- Vulnerability assessment
Security tests during SDLC

types of bugs

- lines of code
- non regression security tests
- security bugs found with “simple” test and/or with known solution
- d-day for pre-production release (pentest)
- security bugs discovered during vuln. assessment
Residual Vulnerabilities will never be 0

Residual vulnerabilities
- detected but too complex to fix
- NOT detected by VA
  - there is no magic see-it-all tool!

Need a **Risk Management** approach
**minimize probability** - **minimize impact**
Continuous Security recipe in short

- Security throughout the **whole SDLC**

- **Complementary** techniques and **tools**
  (again, sorry there’s no single-magic-silver-bullet)

- **Synergy** of the whole **Team** + external security **Experts**
Security through the whole SDLC
- I don’t have time to do security tests too
- Security is an overhead

Different techniques and instruments
- I don’t have the tools
- Tools costs $$$

Synergy of the whole team + external security experts
- Team doesn’t have the skills
- Cannot hire Security Experts for a long time
To stay healthy....

sometimes you need super-experts

periodically you need an advice/review

in many cases you can check on your own
To stay secure...

- Sometimes you need super-experts
- Periodically you need an advice/review
- In many cases you can check on your own

Besides, many checks can also be automatic or semi-automatic.

Professional Pen Tester

Security-trained Architect

Developer, IDE, C.I. server
Better Security **Requirements** with OWASP Application Security Verification Standard (**ASVS**)  
- standardized criteria for common security Use Cases: authentication, authorization, ...
- increasing levels of protection  
  - web portals vs medical records

- Review by Security Architect

Promising approach to partially automate Security Requirement definition and tracking  
[http://securitycompass.com/sdelements](http://securitycompass.com/sdelements)
Manual validations & test

- Design review
- Full Vulnerability Assessment & Penetration Test
  - at 1st prototype
  - at pre-production stage
- Focused Vulnerability Assessment everytime a new “integration point” is introduced
Static Analysis with FindSecurityBugs

Integrated with IDE
Run by the developer while writing code
Very reliable for SQL Injections and dangerous API calls
Can annotate and disable false positives
Intercepts and analyzes all requests to the application, then:

- spidering (with context)
- passive scan
- active attack
- ....and more
Use of ZAP to check **http security headers** (X-XSS protection, X-Frame-options, Content-Security-Policy, etc.)
Use of ZAP to **check http cookie parameters** (flag http-only, flag secure, scope, lifetime)

Automatic test with ZAP
Automatic test

SSL settings active test

Open Source: https://github.com/rbsec/sslscan & ZAP plugins
Semi-automatic test: components

(execution is “simple”, results needs review)

<table>
<thead>
<tr>
<th>Dependency</th>
<th>CPE</th>
<th>GAV</th>
<th>Highest Severity</th>
<th>CVE Count</th>
<th>CPE Confidence</th>
<th>Evidence Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>MySql Data.dll</td>
<td>cpe:a:mysofl.mysq5:1.4</td>
<td></td>
<td>High</td>
<td>110</td>
<td>HIGHEST</td>
<td>4</td>
</tr>
</tbody>
</table>

**Dependencies**


File Path: C:Users\guasco\Desktop\dependency-check-1.3.1-release\Extensl.Rest\Bin\Microsoft.Practices.EnterpriseLibrary.Data.dll
MD5: 79403fc49ae0c6b3e3ca22d93229b26
SHA1: bc0ade2532271d00149f1669757ee7dc46c508d7

**Evidence**

**Related Dependencies**

**Identifiers**

- cpe: cpe:a:micr0soft.enterprise_library:4.1  Confidence:HIGHEST  suppress

**Published Vulnerabilities**

**CVE-2009-3275  suppress**

Severity: Medium
CVSS Score: 5.0 (AV:N/AC:L/ Au:N/C:N/I:N/A/P)
CWE: CWE-134 Uncontrolled Format String

Blocks/Common/Src/Configuration/Manageability/AdminContentBuilder.cs in Microsoft patterns & practices Enterprise Library (aka EntLib) allows context-dependent attacks composed of many \ (backslash) characters followed by a ” (double quote), related to a certain regular expression, aka a “ReDoS” vulnerability.

- BUGTRAQ - 20090910 Regular Expression Denial of Service
Semi-automatic test Configuration

...and deployment check

port scanning (nmap)
vulnerability scanning

OpenVAS,
Nessus(Commercial)
Security Tests vs Continuous Integration and Continuous Delivery
Continuous Integration - Static Analysis

Jenkins running FindSecurityBugs at each build

Effective for
- avoiding dangerous APIs
- detecting SQL injection

Aim at constantly bring down count to zero
Continuous Integration - Comp. Security

Jenkins running Dependency Check at each build

Even more sophisticated Dependency filtering and analysis in tools like Nexus Lifecycle (Commercial)
Continuous Integration - ZAP

Jenkins running ZAP daily

1. Jenkins Job
   - Build WAR
   - Deploy
   - Start ZAP, Webapp
   - Functional Tests
   - Trigger ZAP attack
   - Stop Server ZAP
   - Publish Report

2. Proxy
   - ZAP
   - REST API
   - Spider / Attack

3. WebApp
   - Server (e.g. Tomcat)

4. Output
   - xml / html
<table>
<thead>
<tr>
<th>Phase</th>
<th>Activity</th>
<th>Who?</th>
<th>Tools / Methods</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Analysis</td>
<td>Define Requirements</td>
<td>Dev, Analyst</td>
<td>Owasp ASVS</td>
<td>Manual</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Security Architect</td>
<td>Thread Model, Risk review</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Add other features</td>
<td>Developers</td>
<td>Static Code Analysis in IDE</td>
<td>Mostly Automatic</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>FindSecBugs, ZAP, DepCheck</td>
<td></td>
</tr>
<tr>
<td>Test</td>
<td>Vulnerability Assessment &amp; Pen Test</td>
<td>Security Expert</td>
<td>ZAP, Nmap, Nessus (commercial), many others</td>
<td>Manual</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Semi-auto Automatic</td>
<td></td>
</tr>
<tr>
<td>Production</td>
<td>Monitoring</td>
<td>Sysadmin/DevOps</td>
<td>Log Application-Level Security Events</td>
<td>Semi-auto</td>
</tr>
</tbody>
</table>
Remember three things

1. **Fail Fast** .... validate security & perform tests as early as possible

2. **Automate** where you can ... you’ll earn more time to focus on tougher security issues

3. Don’t skip **periodic expert Design Review**
References

Owasp Secure Coding Principles
  ● https://www.owasp.org/index.php/Secure_Coding_Principles

OWASP Testing Guide

OWASP Application Security Verification Standard
The Tools - Open Source

ZAP

Jenkins
  ● [https://jenkins-ci.org/](https://jenkins-ci.org/)

Plugins
  ● [https://wiki.jenkins-ci.org/display/JENKINS/OWASP+Dependency-Check+Plugin](https://wiki.jenkins-ci.org/display/JENKINS/OWASP+Dependency-Check+Plugin)
  ● [https://wiki.jenkins-ci.org/display/JENKINS/ZAProxy+Plugin](https://wiki.jenkins-ci.org/display/JENKINS/ZAProxy+Plugin)

Find Security Bugs

Dependency Check
  ● [https://www.owasp.org/index.php/OWASP_Dependency_Check](https://www.owasp.org/index.php/OWASP_Dependency_Check)
The Tools - Commercial

Coverity
  ● http://www.coverity.com/

Nessus

Sonatype Nexus Lifecycle
Interested?

Questions? gabriele.guasco@nispro.it

Credits: presentation concepts developed together with Carlo Bonamico (twitter @carlobonamico)
Projector test