Security in Agile Development

So, you’re telling me it can be agile and secure?
Who am I?

Daniel Schwarz
Senior Security Architect @ condignum GmbH in Vienna
Lecturer @ St. Pölten University of Applied Sciences

- appropriate steps to ensure information security -
Good ol' Waterfall times

Initial Planning
- Security Training
- Threat Model
- Security Requirements
- Quality Gates / Bug Bars
- Security/Privacy Risk Assessment
- Attack Surface Analysis

Req. & Design
- Code Style Guides
- Tool Approvals
- Deprecate Unsafe Functions
- Static Analysis

Code
- Unit Tests
- Manual Tests
- Quality Gates
- Performance Monitoring

Test
- Pentest
- Fuzz Testing
- Attack Surface Review
- Final Security Review

Deploy
- Incident Response Plan
- Execute Incident Response Plan

Production
Good ol' Waterfall times
Let’s get agile


© condignum GmbH 2024. All rights reserved.
Let’s get agile

Pentest in every sprint?

© condignum GmbH 2024. All rights reserved.
Within a year of Amazon’s move to AWS, engineers were deploying code every 11.7 seconds, on average.

https://techbeacon.com/app-dev-testing/10-companieskilling-it-devops

Multiple pentests in every sprint?

© condignum GmbH 2024. All rights reserved.
Automation
of most security verifications

Know-How
with great power comes great responsibility

Standardization
for automatic quality gate enforcement
Threat Modeling

Identify things that can go wrong in your system and prevent them

Initial Planning
- Req. & Design
- Code
- Test
- Deploy

Sprint x

Sprint y
- Production
- Req. & Design

Initial Threat Model
- Derive Threats for Current Task

Extend Threat Model
- Architecture Changes
- Reoccurring Problems / Vulnerabilities

© condignum GmbH 2024. All rights reserved.
Threat Modeling

1. What are we building?
   - Identify your sensitive data
   - Create a diagram together

   little example:
   - We’re creating a pizza order system
   - Customers can save
     - their address
     - their personal pizza configurations
     - their credit card
   - Payment is done via an external payment-provider

© condignum GmbH 2024. All rights reserved.
## Threat Modeling

### 1. What are we building?
- Identify your sensitive data
- Create a diagram together

### 2. What can go wrong?
- Brainstorm Worst-Case scenarios
- Use STRIDE as a checklist for major categories

<table>
<thead>
<tr>
<th>Worst Case Scenarios</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Spoofing</strong></td>
</tr>
<tr>
<td>Login as another customer and order a free pizza</td>
</tr>
<tr>
<td><strong>Tampering</strong></td>
</tr>
<tr>
<td>Competitor increases pizza prices; Customer modifies pizza prices to 0 €</td>
</tr>
<tr>
<td><strong>Repudiation</strong></td>
</tr>
<tr>
<td>Somebody orders a pizza and afterwards says he didn’t</td>
</tr>
<tr>
<td><strong>Information Disclosure</strong></td>
</tr>
<tr>
<td>Credit Card data gets leaked</td>
</tr>
<tr>
<td><strong>Denial of Service</strong></td>
</tr>
<tr>
<td>Nobody can order pizza</td>
</tr>
<tr>
<td><strong>Elevation of Privileges</strong></td>
</tr>
<tr>
<td>Somebody can get admin privileges</td>
</tr>
</tbody>
</table>
Threat Modeling

1. **What are we building?**
   - Identify your sensitive data
   - Create a diagram together

2. **What can go wrong?**
   - Brainstorm Worst-Case scenarios
   - Use STRIDE as a checklist for major categories

3. **What can we do about it?**
   - Use OWASP ASVS to derive requirements

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1.1</td>
<td>Verify that user set <strong>passwords</strong> are at least <strong>12 characters</strong> in length.</td>
</tr>
<tr>
<td>2.2.1</td>
<td>Verify that <strong>anti-automation controls</strong> are effective [...] controls include <strong>blocking the most common breached passwords</strong>, <strong>soft lockouts</strong>, rate limiting, CAPTCHA, ever increasing delays between attempts [...]</td>
</tr>
<tr>
<td>2.2.4</td>
<td>Verify impersonation resistance against phishing, such as the use of <strong>multi-factor authentication</strong> [...]</td>
</tr>
<tr>
<td>9.1.1</td>
<td>Verify that <strong>secured TLS</strong> is used for all client connectivity, and does not fall back to insecure or unencrypted protocols.</td>
</tr>
</tbody>
</table>

© condignum GmbH 2024. All rights reserved.
Using ChatGPT as a Threat Model „Expert“

• Give it a lot of details for your initial threat model

• Shape the prompt according to best practices
  • e.g.: https://danielmiessler.com/p/response-shaping-how-to-move-from-ai-prompts-to-ai-whispering/
    • Tell the system who to behave as
    • Tell the system what format it produces
    • Give it the main task you want done
    • ...

• Explain your current task for the ongoing threat model
The initial Threat Model

You

Hi,

Please act as a cybersecurity professional who is a master in threat modeling. Your job is to create a threat model for the following system:

We plan to build an online pizza order system. Customers can save their address, their personal preferred pizza configurations and their credit card data.

Of course the system also needs to save all the different available pizzas and their current prices. And for user logins it will save usernames and passwords.

There are three types of users:
1) anonymous users, who can only view public content like our pizza menu
2) authenticated customers, who can update their profile, configure pizzas and order these pizzas.
3) authenticated admins, who can update the pizza prizes

The whole system runs in containers in AWS. Data is stored in a PostgreSQL database. The frontend is a React SPA which communicates with a JSON REST API written in Python.

The authentication and session management is done via JWTs.

The payment of the pizza orders is done via an external payment provider. The communication with the external payment provider will be done via a REST API.

Please create a threat model for this system. It should contain 4 chapters:

1) Management Summary
   This should be a short management summary to give an overview of the system.

2) Worst-Case Scenarios
   This chapter should list a few worst-case scenarios the system faces and their potential business impact. There should be at least one realistic worst-case scenario for each STRIDE category.

3) Identified Threats
   This is the main chapter which contains all possible identified threats and potential vulnerabilities (at least 10) in an extensive markdown table. The table should have the following columns: ID, Title, Threat Scenario, Potential Vulnerability, Impact, Risk, Countermeasures, ASVS-Requirements. The column ASVS-Requirements should contain the IDs of the OWASP ASVS-Requirements relevant to the respective finding.

4) Countermeasure details
   This chapter contains a detailed description of the countermeasures. It explains the benefits the countermeasure brings. Wherever possible please include code or configuration examples, but also describe what exactly these examples are doing.
The ongoing Threat Model

You

We would like to extend our system for a function to share pizza configurations with other users. So it should be possible for a user to create a link to a specific pizza configuration he defined, share this link with a friend and this friend should be able to add this configuration to his private profile.

Do you see any threats we should consider here?
Secure Coding Guidelines

Checklist to ensure secure coding best practices

Initial Planning

Sprint x

- Req. & Design
- Code
- Test
- Deploy

Sprint y

- Req. & Design
- ...

Initial Sec Coding Guide

- Derive from Company- or Generic Standard
- Get Expert Help
- Use Threat Model as Input

© condignum GmbH 2024. All rights reserved.
Secure Coding Guidelines

Checklist to ensure secure coding best practices

1. Initial Planning
2. Initial Sec Coding Guide
3. Select relevant Chapters as Checklist
4. Extend Sec Coding Guide
5. Architecture Changes
6. Reoccurring Problems / Vulnerabilities

- Sprint x:
  - Req. & Design
  - Code
  - Test
  - Deploy

- Sprint y:
  - Req. & Design
Security Verifications

- **Sec. Code Reviews**
  - Manual security check at peer reviews
  - Make mandatory with pull requests

- **Security Scans**
  - Automatic vulnerability check
  - Prone to false positives
  - SAST (Static)
  - DAST (Dynamic)
  - IAST (Interactive)

- **Dependency Checks**
  - Publicly known vulns in libraries and components

- **Anomaly Detection**
  - Attack detection in container environments

- **Pentest**
  - Manual security check by security expert
  - Very effective
  - Heavy in time and resources

- After establishing basic architecture
  - Before 1st go live
  - After main architectural changes
    - new datastores
    - new technologies
    - new interfaces/services
    - changes in authentication/authorization
Key Messages

1. **Security is more important than ever**
   We also need to consider it in short release cycles

2. **Automate (most) security tests and make them mandatory by pipelines**
   Smart combination of tools and manual pentests is most efficient

3. **Start threat modeling and utilize AI**
   A little brainstorming for worst-case scenarios already gives you a good starting point

4. **Use secure coding guidelines as checklists**
   It's hard to remember everything – checklists are great!
Resources

- Diagrams
  - C4 Model: https://c4model.com/

- Secure Design
  - IEEE CSD: https://cybersecurity.ieee.org/center-for-secure-design/

- Threat Modeling Methods
  - OCTAVE: https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=13473
  - TRIKE: http://www.octotrike.org/
  - LINDDUN: https://www.linddun.org

- Starting Points for Secure Coding Guides
  - OWASP ASVS: https://owasp.org/www-project-application-security-verification-standard/
  - OWASP Top 10: https://owasp.org/www-project-top10/

- AI Prompt Shaping
Daniel Schwarz
Senior Security Architect
daniel.schwarz@condignum.com