Secure Software Development on the Enterprise Level

Achim D. Brucker

a.brucker@sheffield.ac.uk  https://www.brucker.ch/

Software Assurance & Security Research
Department of Computer Science, The University of Sheffield, Sheffield, UK
https://logicalhacking.com/

Shift Left: The Incredible Impact Early Security Testing Makes
January 19, 2017, London, UK

```java
boolean getExtra(JsObject apiObject) {
    if (apiObject != null) {
        callbackContext.sendPluginResult(new PluginResult(PluginResult.Status.OK, new JsValue(apiObject.getProperty("extra")));
        return true;
    } else {
        callbackContext.sendPluginResult(new PluginResult(PluginResult.Status.ERROR));
        return false;
    }
}
```
Outline

1. Background
2. Motivation
3. Secure Software Development
4. From (Mild) Pain to Success: My Experiences at SAP
5. Lesson's Learned
Personal Background

- **Eight year of enterprise secure software development:**
  - Member of the central security team, SAP SE (Germany)
    - (Global) Security Testing Strategist
    - Security Research Expert/Architect
  - Work areas:
    - Defining the risk-based Security Testing Strategy of SAP
    - Introducing security testing tools (e.g., SAST, DAST) at SAP
    - Identify white spots and evaluate and improve tools/methods
    - Secure Software Development Life Cycle integration
    - Applied security research
    - ...

- **Since 12/2015:**
  - Senior Lecturer, The University of Sheffield, UK
  - Head of the Software Assurance & Security Research Team
  - Available as consultant & (research) collaborations

[https://www.brucker.uk/](https://www.brucker.uk/)
SAP SE

- Leader in Business Software
  - Cloud
  - Mobile
  - On premise
- Many different technologies and platforms, e.g.,
  - In-memory database and application server (Hana)
  - Netweaver for ABAP and Java
- More than 25 industries
- 63% of the world’s transaction revenue touches an SAP system
- over 68,000 employees worldwide
  over 25,000 software developers
- Headquarters: Walldorf (Heidelberg), Germany
Outline

1 Background
2 Motivation
3 Secure Software Development
4 From (Mild) Pain to Success: My Experiences at SAP
5 Lesson's Learned
Example (LinkedIn, May 2016)

- 164 million email addresses and passwords from an attack in 2012, offered for sale May 2016
- Compromised data:
  - email addresses
  - passwords
Example (TalkTalk, October 2015)

- nearly 157,000 customer records leaked
- nearly 16,000 records included bank details
- more than 150,000 customers lost (home services market share fall by 4.4 percent in terms of new customers)
- Costs for TalkTalk: around any £60 million
<table>
<thead>
<tr>
<th>Example (Ashley Madison, July 2015)</th>
</tr>
</thead>
<tbody>
<tr>
<td>- more than 30 million email addresses &amp; much more</td>
</tr>
<tr>
<td>- Compromised data:</td>
</tr>
<tr>
<td>- Dates of birth</td>
</tr>
<tr>
<td>- Email addresses</td>
</tr>
<tr>
<td>- Ethnicities, Genders</td>
</tr>
<tr>
<td>- Sexual preferences</td>
</tr>
<tr>
<td>- Home addresses, Phone numbers</td>
</tr>
<tr>
<td>- Payment histories</td>
</tr>
<tr>
<td>- Passwords, Usernames, Security questions and answers</td>
</tr>
<tr>
<td>- Website activity</td>
</tr>
<tr>
<td>- Similar Leak: Mate1 in February 2016: 27 million records with even more personal details (e.g., drinking/drug habits, political views)</td>
</tr>
</tbody>
</table>
What’s the Problem?

Authenticate without a password using "SQL Injection"

Implementation (SQL, simplified):

```
SELECT * FROM 'users' WHERE 'name' = 'Username' AND 'pwd' = 'Password';
```
What’s the Problem?

Authenticate without a password using "SQL Injection"

.implimentation (SQL, simplified):

```sql
SELECT * FROM 'users' WHERE 'name' = 'Username' AND 'pwd' = 'Password';
```
What’s the Problem?
Authenticate without a password using "SQL Injection"

Implementation (SQL, simplified):

```
SELECT * FROM 'users' WHERE 'name' = 'Username' AND 'pwd' = 'Password';
```

Let’s try: user “test” & password “secret”:

```
SELECT * FROM 'users' WHERE 'name' = 'test' AND 'pwd' = 'secret';
```
What’s the Problem?
Authenticate without a password using "SQL Injection"

- Implementation (SQL, simplified):

```
SELECT * FROM 'users' WHERE 'name' = 'Username' AND 'pwd' = 'Password';
```

- Let’s try: user “test” & password “secret”:

```
SELECT * FROM 'users' WHERE 'name' = 'test' AND 'pwd' = 'secret';
```
What’s the Problem?
Authenticate without a password using "SQL Injection"

➤ Implementation (SQL, simplified):

```
SELECT * FROM 'users' WHERE
  'name' = 'Username' AND 'pwd' = 'Password';
```

➤ Let's try: user “test” & password “secret”:

```
SELECT * FROM 'users' WHERE
  'name' = 'test' AND 'pwd' = 'secret';
```
What’s the Problem?
Authenticate without a password using "SQL Injection"

❖ Implementation (SQL, simplified):

```
SELECT * FROM 'users' WHERE 'name' = 'Username' AND 'pwd' = 'Password';
```

❖ Let's try: user “test” & password “secret”:

```
SELECT * FROM 'users' WHERE 'name' = 'test' AND 'pwd' = 'secret';
```

❖ Let's use “' OR '1'='1” as password:
What’s the Problem?
Authenticate without a password using "SQL Injection"

- Implementation (SQL, simplified):
  
  ```sql
  SELECT * FROM 'users' WHERE 'name' = 'Username' AND 'pwd' = 'Password';
  ```

- Let’s try: user “test” & password “secret”:
  
  ```sql
  SELECT * FROM 'users' WHERE 'name' = 'test' AND 'pwd' = 'secret';
  ```

- Let’s use “' OR '1'='1” as password:
  
  ```sql
  SELECT * FROM 'users' WHERE 'name' = 'test' AND 'pwd' = ' OR '1'='1';
  ```
What’s the Problem?
Authenticate without a password using "SQL Injection"

Implementation (SQL, simplified):

```
SELECT * FROM 'users' WHERE 'name' = 'Username' AND 'pwd' = 'Password';
```

Let’s try: user “test” & password “secret”:

```
SELECT * FROM 'users' WHERE 'name' = 'test' AND 'pwd' = 'secret';
```

Let’s use “’ OR ’1’=’1” as password:

```
SELECT * FROM 'users' WHERE 'name' = 'test' AND 'pwd' = '' OR TRUE;
```
What’s the Problem?
Authenticate without a password using "SQL Injection"

- Implementation (simplified):
  
  ```sql
  SELECT * FROM 'users' WHERE 'name' = 'Username' AND 'pwd' = 'Password';
  ```

- Let's try: user "test" & password "secret":

  ```sql
  SELECT * FROM 'users' WHERE 'name' = 'test' AND 'pwd' = 'secret';
  ```

- Let's use "' OR '1'='1" as password:

  ```sql
  SELECT * FROM 'users' WHERE 'name' = 'test' AND 'pwd' = 'secret';
  ```

- No password check!

Root cause: a bug.
Outline

1. Background
2. Motivation
3. Secure Software Development
4. From (Mild) Pain to Success: My Experiences at SAP
5. Lesson's Learned
A Path Towards (More) Secure Software
SAP’s Secure Software Development Lifecycle (S²DL)
A Path Towards (More) Secure Software

SAP’s Secure Software Development Lifecycle ($S^2DL$)

Training

- Security awareness
- Secure programming
- Threat modelling
- Security testing
- Data protection and privacy
- Security expert curriculum (“Masters”)
Risk Identification

- Risk identification ("high-level threat modelling")
- Threat modelling
- Data privacy impact assessment
A Path Towards (More) Secure Software

SAP’s Secure Software Development Lifecycle (S²DL)

Plan Security Measures

- Plan product standard compliance
- Plan security features
- Plan security tests
- Plan security response
A Path Towards (More) Secure Software

SAP’s Secure Software Development Lifecycle (S²DL)

Secure Development

- Secure Programming
- Static code analysis (SAST)
- Code review
A Path Towards (More) Secure Software

SAP’s Secure Software Development Lifecycle (S²DL)

Security Testing
- Dynamic Testing (e.g., IAST, DAST)
- Manual testing
- External security assessment
A Path Towards (More) Secure Software

SAP’s Secure Software Development Lifecycle (S²DL)

Security Validation (“First Customer”)
- Check for “flaws” in the implementation of the S²DL
- Ideally, security validation finds:
  - No issues that can be fixed/detected earlier
  - Only issues that cannot be detect earlier (e.g., insecure default configurations, missing security documentation)

Penetration tests in productive environments are different:
- They test the actual configuration
- They test the productive environment (e.g., cloud/hosting)
A Path Towards (More) Secure Software

SAP’s Secure Software Development Lifecycle (S\(^2\)DL)

**Security Response**
- Execute the security response plan
- Security related external communication
- Incident handling
- Security patches
- Monitoring of third party components
A Path Towards (More) Secure Software

SAP’s Secure Software Development Lifecycle (S^2DL)
A Path Towards (More) Secure Software

SAP’s Secure Software Development Lifecycle (S²DL)
A Path Towards (More) Secure Software

SAP’s Secure Software Development Lifecycle (S²DL)
A Path Towards (More) Secure Software

SAP’s Secure Software Development Lifecycle (S²DL)
A Path Towards (More) Secure Software

SAP’s Secure Software Development Lifecycle (S²DL)
A Path Towards (More) Secure Software

SAP’s Secure Software Development Lifecycle (S²DL)
Secure Software Development Lifecycle for Cloud/Agile

Plan Security Measures
Risk Identification

Define

Build Decision
Build

Secure Development
Security Testing
Security Validation

Security Response

Release Decision
Release

Operate

Build

Define

Release

Build Operate
Define
Release
Decision
Build
Decision
Risk
Identification
Plan Security
Measures
Secure 
Development
Security 
TestingSecurity 
Validation
Security
Response
Outline

1. Background
2. Motivation
3. Secure Software Development
4. From (Mild) Pain to Success: My Experiences at SAP
5. Lesson's Learned
Finding Security Vulnerabilities

- **Manual**
  - Penetration Testing
  - Manual Code Review

- **Automatic**
  - Running Application
    - DAST, IAST Vulnerability Scanner
  - Static Analysis
    - SAST
Finding Security Vulnerabilities

- Manual Penetration Testing
- Automatic DAST, IAST Vulnerability Scanner
- Manual Running Application Static Analysis
- Automatic SAST Manual Code Review

© 2017 LogicalHacking.com. Public (CC BY-NC-ND 4.0)
In 2010: Static Analysis Becomes Mandatory

SAST tools used at SAP:

<table>
<thead>
<tr>
<th>Language</th>
<th>Tool</th>
<th>Vendor</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABAP</td>
<td>CodeProfiler</td>
<td>Virtual Forge</td>
</tr>
<tr>
<td>Others</td>
<td>Fortify</td>
<td>HP</td>
</tr>
</tbody>
</table>

➢ Since 2010: SAST mandatory for all SAP products
➢ Within two years, multiple billions lines analysed
➢ Constant improvement of tool configuration
➢ Further details:
A De-Centralised Application Security Approach
How SAP’s Application Development Approach Developed Over Time

- Governance & approvals
- De-centralized approach

2009 2016

- One Two SAST tools fit all
  - VF CodeProfiler
  - Fortify

- Blending of Security Testing Tools
  - SAST:
    - SAP Netweaver CVA Add-on, Fortify,
    - Synopsis Coverity, Checkmarx,
    - Breakman
  - DAST:
    - HP WebInspect, Quotium Seeker
  - Others:
    - Burp Suite, OWASP ZAP, Codinomicon
    - Fuzzer, BDD
A De-Centralised Application Security Approach

How SAP’s Application Development Approach Developed Over Time

- Governance & approvals
- De-centralized approach

Development Teams
- feel pushed

Central Security Team
- Controls development teams
- Spends a lot time with granting exemptions

Danger
- Only ticking boxes

Blending of Security Testing Tools
- SAST:
  - SAP Netweaver CVA Add-on, Fortify, Synopsis Coverity, Checkmarx, Breakman
- DAST:
  - HP WebInspect, Quotium Seeker
- Others:
  - Burp Suite, OWASP ZAP, Codinomicon Fuzzer, BDD
A De-Centralised Application Security Approach

How SAP’s Application Development Approach Developed Over Time

- Governance & approvals
- De-centralized approach

2009 - 2016

Development Teams
- feel **pushed**

Central Security Team
- Controls development teams
- Spends a lot time with granting exemptions

Danger
- Only ticking boxes

Development Teams
- are **empowered**
- are **responsible**

Central Security Team
- Supports development teams
- Can focuses on improvements
  - filling white spots
  - tooling
  - processes
De-Centralised Approach: Organisational Setup

- **Central security expert team** (S²DL owner)
  - Organizes security trainings
  - Defines product standard “Security”
  - Defines risk and threat assessment methods
  - Defines security testing strategy
  - Selects and provides security testing tools
  - Validates products
  - Defines and executes response process

- **Local security experts**
  - Embedded into development teams
  - Organize local security activities
  - Support developers and architects
  - Support product owners (responsibles)

- **Development teams**
  - Select technologies
  - Select development model
  - Design and execute security testing plan
  - ...
Security Team Focus: Security Testing for Developers

Security testing tools for developers, need to

- Be applicable from the start of development
- Automate the security knowledge
- Be integrated into dev world, e.g.,
  - IDE (instant feedback)
  - Continuous integration
- Provide easy to understand fix recommendations
- Declare their “sweet spots”

https://logicalhacking.com/blog/2016/10/25/classifying-security-testing-tools/
Combining Multiple Security Testing Methods and Tools

- Web Client
- Web Browser
- Server Application
- Runtime Container
- Backend Systems

- Risks of only using only SAST
  - Wasting effort that could be used more wisely elsewhere
  - Shipping insecure software

- Examples of SAST limitations
  - Not all programming languages supported
  - Covers not all layers of the software stack

Combining Multiple Security Testing Methods and Tools

- Risks of only using only SAST
  - Wasting effort that could be used more wisely elsewhere
  - Shipping insecure software
- Examples of SAST limitations
  - Not all programming languages supported
  - Covers not all layers of the software stack

Combining Multiple Security Testing Methods and Tools

- Risks of only using only SAST
  - Wasting effort that could be used more wisely elsewhere
  - Shipping insecure software
- Examples of SAST limitations
  - Not all programming languages supported
  - Covers not all layers of the software stack

Combining Multiple Security Testing Methods and Tools

- Risks of only using only SAST
  - Wasting effort that could be used more wisely elsewhere
  - Shipping insecure software
- Examples of SAST limitations
  - Not all programming languages supported
  - Covers not all layers of the software stack

Combining Multiple Security Testing Methods and Tools

- Risks of only using only SAST
  - Wasting effort that could be used more wisely elsewhere
  - Shipping insecure software
- Examples of SAST limitations
  - Not all programming languages supported
  - Covers not all layers of the software stack
- A comprehensive approach combines
  - Static approaches (i.e., SAST)
  - Dynamic approaches (i.e., IAST or DAST)

How to Measure Success (and Identify White Spots)

- Analyze the vulnerabilities reported by
  - Security Validation
  - External security researchers
- Vulnerability not detected by currently used methods
  - Improve tool configuration
  - Introduce new tools
- Vulnerability detected by our security testing tools
  - Vulnerability in older software release
  - Analyze reason for missing vulnerability
How to Measure Success (and Identify White Spots)

- Analyze the vulnerabilities reported by
  - Security Validation
  - External security researchers
- Vulnerability not detected by currently used methods
  - Improve tool configuration
  - Introduce new tools
- Vulnerability detected by our security testing tools
  - Vulnerability in older software release
  - Analyze reason for missing vulnerability

Success criteria:
Percentage of vulnerabilities not covered by our security testing tools increases
How to Measure Success (and Identify White Spots)

- Analyze the vulnerabilities reported by
  - Security Validation
  - External security researchers
- Vulnerability not detected by currently used methods
  - Improve tool configuration
  - Introduce new tools
- Vulnerability detected by our security testing tools
  - Vulnerability in older software release
  - Analyze reason for missing vulnerability

Success criteria:
Percentage of vulnerabilities not covered by our security testing tools increases
How to Measure Success (and Identify White Spots)

- Analyze the vulnerabilities reported by
  - Security Validation
  - External security researchers
- Vulnerability not detected by currently used methods
  - Improve tool configuration
  - Introduce new tools
- Vulnerability detected by our security testing tools
  - Vulnerability in older software release
  - Analyze reason for missing vulnerability

Success criteria:
Percentage of vulnerabilities not covered by our security testing tools increases
Outline

1. Background
2. Motivation
3. Secure Software Development
4. From (Mild) Pain to Success: My Experiences at SAP
5. Lesson's Learned
Key Success Factors

- A holistic security awareness program for
  - Developers
  - Managers
Key Success Factors

- A holistic security awareness program for
  - Developers
  - Managers
- Yes, security awareness is important
A holistic security awareness program for
  Developers
  Managers
Yes, security awareness is important but
Key Success Factors

- A holistic security awareness program for
  - Developers
  - Managers
- Yes, security awareness is important but
  
  **Developer awareness** is even more important!
Listen to Your Developers And Make Their Life Easy!

We are often talking about a lack of security awareness and, by that, forget the problem of lacking development awareness.

- Building a secure system more difficult than finding a successful attack.
- Do not expect your developers to become penetration testers (or security experts)!

Organisations can make it hard for developers to apply security testing skills!

- Don't ask developers to do security testing, if their contract doesn't allows it
- Budget application security activities centrally
- Educate your developers and make them recognised experts
Final remarks

What works well:

- Delegate power and accountability to development teams
- Multi-tiered model of security experts:
  - local experts for the local implementation of secure development
  - global experts that support the local security experts (champions):
    - act as consultant in difficult/non-standard situations
    - evaluate, purchase, and operate widely used security testing tools
    - can mediate between development teams and response teams
- Strict separation of
  - security testing supporting developers and
  - security validation

What does not work well:

- Forcing tools, processes, etc. on developers
- Penetration testing as “secure development” approach
  - Penetration has its value, e.g.,
    - as security integration test
    - as “meta-test” for your secure development process (validation)
Thank you for your attention!
Any questions or remarks?

Contact:
Dr. Achim D. Brucker
Department of Computer Science
University of Sheffield
Regent Court
211 Portobello St.
Sheffield S1 4DP, UK
a.brucker@sheffield.ac.uk
@adbrucker
https://de.linkedin.com/in/adbrucker/
https://www.brucker.ch/
https://logicalhacking.com/blog/
Bibliography


© 2017 LogicalHacking.com, A.D. Brucker.

This presentation is classified as Public (CC BY-NC-ND 4.0): Except where otherwise noted, this presentation is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International Public License (CC BY-NC-ND 4.0).