Developing Secure Software: From Mobile Apps to ERP Systems

Achim D. Brucker
achim.brucker@sap.com http://www.brucker.ch/
SAP SE, Vincenz-Priessnitz-Str. 1, 76131 Karlsruhe, Germany

ZertApps Abschlussveranstaltung
“Sichere und datenschutzgerechte Entwicklung von mobilen Apps”
TeleTrusT Bremen, November 17, 2015
Abstract

Developing secure software is, in general, challenging and requires an end-to-end secure software development lifecycle. It is particularly challenging if the secure software development lifecycle needs to fit the whole range of software products from small mobile apps to large scale enterprise systems and needs to be applicable to a wide range of software development methodologies.

In this presentation, we will present SAP’s approach to developing secure software in general and, in particular, highlight the challenges of developing mobile applications securely.
Agenda

1 Background
2 Motivation
3 Risk-based Security Testing as Part of SAP’s S²DL
4 Lesson’s Learned
5 How Does This Apply to Mobile Development?
6 Conclusion
• 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, Germany (close to Heidelberg)
Personal Background

- I wear two hats:
  - **(Global) Security Testing Strategist**
  - Research Expert/Architect
  
  Working for the central software security team

- Background:
  Security, Formal Methods, Software Engineering

- Current work areas:
  - Static code analysis
  - (Dynamic) Security Testing
  - Mobile Security
  - Security Development Lifecycle
  - Secure Software Development Lifecycle

http://www.brucker.ch/
SAP Uses a De-centralised Secure Development Approach

- **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
  - …
Agenda

1 Background
2 Motivation
3 Risk-based Security Testing as Part of SAP’s S²DL
4 Lesson’s Learned
5 How Does This Apply to Mobile Development?
6 Conclusion
Vulnerability Distribution

- Code Execution
- DoS
- Overflow
- Memory Corruption
- Sql Injection
- XSS
- Directory Traversal
- Bypass something
- Gain Privileges
- CSRF
Agenda

1 Background
2 Motivation
3 Risk-based Security Testing as Part of SAP’s S\(^2\)DL
4 Lesson’s Learned
5 How Does This Apply to Mobile Development?
6 Conclusion
Our Start: SAST as a Baseline

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>CVA (SLIN_SEC)</td>
<td>SAP</td>
</tr>
<tr>
<td>JavaScript</td>
<td>Checkmarx CxSAST</td>
<td>Checkmarx</td>
</tr>
<tr>
<td>C/C++</td>
<td>Coverity</td>
<td>Coverity</td>
</tr>
<tr>
<td>Others</td>
<td>Fortify</td>
<td>HP</td>
</tr>
</tbody>
</table>

- Since 2010, mandatory for all SAP products
- Multiple billions lines analyzed
- Constant improvement of tool configuration
- Further details:
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 Risk-based Test Plan

- Combines multiple security testing methods, e.g., code scans, dynamic analysis, manual penetration testing or fuzzing
- Selects the most efficient test tools and test cases based on the risks and the technologies used in the project
- Re-adjusts priorities of test cases based on identified risks for the project
- Monitors false negative findings in the results of risk assessment
SAP’ Secure Software Development Lifecycle (S²DL)

Start of development

Preparation
- Training
  - Security awareness
  - Secure programming
  - Threat modelling
  - Security static analysis
  - Data protection and privacy
  - Security expert curriculum
- Risk Identification
  - SECURIM (Security Risk Identification and Management)
  - Data Privacy Impact Assessment
  - Threat Modeling

Development
- Plan Security Measures
  - Plan product standard compliance
  - Plan security features
  - Plan security tests
  - Code review
- Secure development
  - Secure programming
  - Static code scan
  - Code review

Security testing
- Dynamic testing
- Manual testing
- External security assessment

Transition
- Security Validation
- Independent security assessment

Utilization
- Security Response
- Execute the security response plan

Release decision
Security Validation

- Acts as first customer
- Is not a replacement for security testing during development
- Security Validation
  - Check for “flaws” in the implementation of the $S^2$DL
  - 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)
Security Validation

• Acts as first customer
• Is not a replacement for security testing during development
• Security Validation
  • Check for “flaws” in the implementation of the $S^2$DL
  • 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)
How to Measure Success

- Analyze the vulnerabilities reported by
  - Security Validation
  - External security researchers
- Vulnerability not detected by our security testing tools
  - 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

- Analyze the vulnerabilities reported by
  - Security Validation
  - External security researchers
- Vulnerability not detected by our security testing tools
  - 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

- Analyze the vulnerabilities reported by
  - Security Validation
  - External security researchers
- Vulnerability not detected by our security testing tools
  - 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

• Analyze the vulnerabilities reported by
  • Security Validation
  • External security researchers
• Vulnerability not detected by our security testing tools
  • 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
Agenda

1. Background
2. Motivation
3. Risk-based Security Testing as Part of SAP’s $^{2}$DL
4. Lesson’s Learned
5. How Does This Apply to Mobile Development?
6. Conclusion
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
Key Success Factors

- 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!

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)!
Security testing tools for developers, need to:

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

Security experts need to collaborate with development experts to

- Create easy to use security APIs (ever tried to use an SSL API securely)
- Create languages and frameworks that make it hard to implement insecure systems
- Explain how to program securely
1. Background
2. Motivation
3. Risk-based Security Testing as Part of SAP’s S²DL
4. Lesson’s Learned
5. How Does This Apply to Mobile Development?
6. Conclusion
Key take aways:

- Hybrid applications are becoming the pre-dominant development model (at SAP)
- the challenges of hybrid apps are transferable to
  - web frameworks (EJB, Rails, PHP)
  - enterprise applications (XSJS, SQLScript, ABAP, JS)
  - even mobile apps contain > 500kLOC
- there are a lot of open and interesting security research questions in the area of hybrid development models
Why Are Mobile Apps Special

Organisational Aspects

- Partly not developed by the development organisations (e.g., marketing)
- Fast update cycles (to app store, not necessarily “on device”)
- Mobile apps are not patched (instead: new release)
- Processes partly defined by App Store operators (e.g., Google, Apple, . . .)
Why Are Mobile Apps Special

Technical Aspects

• Limited/different user interface
• High volume of apps released
• Development tools are not fully under own control
• Programming languages might not be used elsewhere
• Lot of frameworks that
  • rather new
  • not as mature
  • might track users (data privacy)
• They are not independent . . .
A final remark:

- usually there is at least one server “in the background”
- many security and data privacy issues are caused by
  - the communication of the app and its “own” server
  - the implementation of its “own” server
  - external servers and/or services
Conclusion

- Secure software development is a
  - Prerequisite for the secure and compliant operation: We need SecDevOps!
  - Risk of operating and maintaining IT systems
- Security requires an end-to-end approach
  - Training of developers, architects, product owners
  - Security testing during development
  - Validation of your security testing efforts
  - Maintenance and security patch management
- Developers are your most important ally
  - Make life easy for them
Thank you!

http://xkcd.com/327/

Dr. Achim D. Brucker
achim.brucker@sap.com
https://www.brucker.ch/
https://logicalhacking.com/
Related Publications

Ruediger Bachmann and Achim D. Brucker.
Developing secure software: A holistic approach to security testing.

Achim D. Brucker, Lukas Brügger, and Burkhart Wolff.
Formal firewall conformance testing: An application of test and proof techniques.

Achim D. Brucker and Uwe Sodan.
Deploying static application security testing on a large scale.
ISBN 978-3-88579-622-0.

Achim D. Brucker and Burkhart Wolff.
On theorem prover-based testing.
ISSN 0934-5043.