**Introducing Security Testing to Developers**

**Experiences and Lessons Learned**

**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/

Checkmarx Security Conference Tokyo /two.lnum/zero.lnum/one.lnum/seven.lnum

実践アプリケーションセキュリティ

December /one.lnumst, /two.lnum/zero.lnum/one.lnum/seven.lnum

Outline

- About Me
- Motivation
- Secure Software Development
- Enabling Developers: From (Mild) Pain to Success
- Lesson’s Learned

---

**About Me**

PhD from ETH Zurich, Switzerland

Eight year experience in secure enterprise software development:
- Member of the central security team, SAP SE (Germany)
- Security Research Expert/Architect
- Work areas at SAP included:
  - Defining the risk-based Security Testing Strategy
  - Evaluation of security testing tools (e.g., SAST, DAST)
  - Roll-out of security testing tools
  - Identification of white spots and improvements of tools
  - Secure Software Development Life Cycle integration
  - Applied security research

Since December 2015:
- Associate Professor, The University of Sheffield, UK
- Head of the Software Assurance & Security Research Team
- Available as consultancy & (research) collaborations

https://www.brucker.ch/

---

**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 88 000 employees worldwide
- Over 25 000 software developers
- Headquarters: Walldorf (Heidelberg), Germany
Example (LinkedIn, May 2016)

- 164 million email addresses and passwords
- Data leaked in 2012, data sold in 2016
- Leaked Data
  - E-mail 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 fell by 4.4 percent in terms of new customers)
- Costs for TalkTalk: ca. £60 million (ca. 90億円)
Example (Ashley Madison, July 2015)
- More than 30 million email addresses & much more
- Leaked data:
  - Date of birth
  - E-mail addresses
  - Ethnicities, Genders
  - Sexual preferences
  - Home addresses, Phone numbers
  - Payment histories
  - Passwords, usernames, security questions and answers
  - Website activity

Outline
- About Me
- Motivation
- Secure Software Development
- Enabling Developers: From (Mild) Pain to Success
- Lesson's Learned

A Path Towards (More) Secure Software
SAP's Secure Software Development Lifecycle (SDLC)

Training
- Security awareness
- Secure programming
- Threat modelling
- Security testing
- Data protection and privacy
- Security expert curriculum ("Masters")
A Path Towards (More) Secure Software
SAP’s Secure Software Development Lifecycle (SDLC)

Risk Identification
- Risk identification (‘high-level threat modelling’)
- Threat modelling
- Data privacy impact assessment

Plan Security Measures
- Plan product standard compliance
- Plan security features
- Plan security tests
- Plan security response

Secure Development
- Secure Programming
- Static code analysis (SAST)
- Code review

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 (SDLC)

Security Validation ("First Customer")
- Check for "flaws" in the implementation of the SDLC
- Ideally, security validation finds:
  - No issues that can be fixed/detected earlier
  - Only issues that cannot be detected 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)

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 (SDLC)

- Training
- Risk Identification
- Plan Security Measures
- Secure Development
- Security Testing
- Security Validation
- Security Response

Secure Software

Secure Software
Finding Security Vulnerabilities

Manual

Penetration Testing
DAST IAST Vulnerability Scanner

Automatic

Static Analysis

SAST

In 2010: Static Analysis Becomes Mandatory

SAST tools used:

<table>
<thead>
<tr>
<th>Language</th>
<th>Tool</th>
<th>Vendor</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABAP</td>
<td>CodePro</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 products
Within two years, multiple billions lines analysed
Constant improvement of tool configuration
Further details:
A De-Centralised Application Security Approach

Improving The Application Development Approach

Governance & approvals

**De-centralized approach**

2009

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

2016

Blending of Security Testing Tools

- Static:
  - SAP Netweaver CVA Add-on, Fortify,
  - Synopsis Coverity, Checkmarx, Breakman
- Dynamic:
  - HP WebInspect, Quotium Seeker
- Others:
  - Burp Suite, OWASP ZAP, Codenomicon
  - Defensics, BDD

Development Teams

- Feel pushed
- Central Security Team
  - Controls development teams
  - Spends a lot of time with granting exemptions
- Danger
  - Only ticking boxes

Central security expert team (SDLC 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

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/2015/10/25/identifying-security-testing-tools/

**Develop a Culture of Security Champions**

- Make security interesting
  - Offer education/talks
  - Gamification
- Encourage (volunteers) security champions
  - Do not force them, they should volunteer
  - Provide incentives
- Build a community
  - Organize knowledge transfer
  - Meet in person
- Empower your security champions
  - Trust their decisions
  - Include them decisions
  - (selection of new tools, process changes, etc.)
- Each developer should know a security champion personally

**How to Start?**

**Start Slow, Grow and Improve Fast**

Start slow:
- Start with a limited scope
  - Only one team
  - Only a subset of vulnerability types
  - Introduce only one tool at a time
- Focus first on newly developed code
  - But develop a plan for fixing old code as well

Grow and improve fast:
- Encourage teams to share their success stories
- To help each other
- Make tools available easily
  - Central budgeting
  - Integration into build/repository infrastructure
**Success criteria by a (bad!) Security Expert:**
Fix all issues so that nothing is reported
(I don't want to understand, why an issue is a false positive ...)

**Listen to your developers:**
forget Security Awareness, a successful application security program needs Developer Awareness

**Thoughts on Success Criteria for Developers**

- Use of frameworks that help to avoid security issues
- Fixing of obvious issues prior to commits
- Taking security fixes seriously
- Use of security testing tools
- How about third party libraries?

**How to Measure Success (and Identify White Spots)**

Non-working performance indicators include:
- Absolute number of reported vulnerabilities
- Absolute number of fixed issues

A new idea:
- Analyze the vulnerabilities reported by
  - Security Validation
  - External security researchers
- Two classes:
  - Vulnerabilities that can be detected by used tools
    - Investigate why issues was missed
  - Vulnerabilities not detected by used tools
    - If risk acceptable: nothing to do
    - If risk not acceptable: improve tooling
Non-working performance indicators include:
- Absolute number of reported vulnerabilities
- Absolute number of fixed issues

A new idea:
- Analyze the vulnerabilities reported by
  - Security Validation
  - External security researchers

Two classes:
- Vulnerabilities that can be detected by used tools
  - Investigate why issues was missed
- Vulnerabilities not detected by used tools
  - if risk acceptable: nothing to do
  - if risk not acceptable: improve tooling

- Vulnerabilities not detected by used tools
  - if risk acceptable: nothing to do
  - if risk not acceptable: improve tooling

- if risk acceptable: nothing to do
  - if risk not acceptable: improve tooling

100% externally reported vuln.
in scope not in scope of current security testing tools

100% externally reported vuln.
in scope not in scope of current security testing tools
not acceptable risk
How to Measure Success (and Identify White Spots)

Non-working performance indicators include:
- Absolute number of reported vulnerabilities
- Absolute number of fixed issues

A new idea:
- Analyze the vulnerabilities reported by
  - Security Validation
  - External security researchers

Two classes:
- Vulnerabilities that can be detected by used tools
  - Investigate why issues were missed
  - If risk acceptable: nothing to do
  - If risk not acceptable: improve tooling
- Vulnerabilities not detected by used tools
  - Risk acceptable: nothing to do
  - Risk not acceptable: improve tooling

Outline

- About Me
- Motivation
- Secure Software Development
- Enabling Developers: From (Mild) Pain to Success
- Lesson’s Learned

Key Success Factors

- A holistic security awareness program for
  - Developers
  - Managers

Success criteria:
Percentage of vulnerabilities not covered by currently used security testing tools increases, i.e., the used tools are used effectively!
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
Recommendations for Selecting Security Testing Tools

Select tools that are
- easy to integrate into your development process and tools
- central scan infrastructure
- source code upload, CLI, Jenkins, github, …
- easy to use by developers
- easy to understand descriptions of findings
- actionable fix recommendations
- integrates teaching
- easy to adapt to your security policies and prioritisation
- report issues that are relevant for you
- focus developers effort on the issues that are critical for you
- allow for tracking your success
- tool internal reporting
- interfaces to your own reporting infrastructure

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)

Bibliography

- Ruediger Bachmann and Achim D. Brucker.
  Developing secure software: A holistic approach to security testing
- Achim D. Brucker and Uwe Sodan.
  Deploying static application security testing on a large scale.
  In Stefan Katzenbeisser, Volkmar Lotz, and Edgar Weippl, editors,
  G. March 2014.

Contact:
Dr. Achim D. Brucker
Department of Computer Science
University of Sheffield
Regent Court
S1 4DP, UK

Blog:
https://logicalhacking.com/blog/
https://twitter.com/adbrucker
https://www.brucker.ch/
https://de.linkedin.com/in/adbrucker/

© 2007 LogicalHacking.com. Public (CC BY-NC-ND /four.lnum./zero.lnum)
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).