How to Enable Developers to Deliver Secure Code

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/

March /one.lnum/five.lnum, /two.lnum/zero.lnum/one.lnum/seven.lnum

Outline

1 Motivation

Secure Software Development

Enabling Developers: From (Mild) Pain to Success

Lesson’s Learned

Example (LinkedIn, May /two.lnum/zero.lnum/one.lnum/six.lnum)

/one.lnum/six.lnum/four.lnum million email addresses and passwords
from an attack in /two.lnum/zero.lnum/one.lnum/two.lnum, offered for sale May /two.lnum/zero.lnum/one.lnum/six.lnum

Compromised data:

email addresses
passwords

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

Example (Ashley Madison, July 2015)

- more than 30 million email addresses & much more
- Compromised data:
  - Dates of birth
  - Email addresses
  - Ethnicities, Genders
  - Sexual preferences
  - Home addresses, Phone numbers
  - Payment histories
  - Passwords, Usernames, Security questions and answers
  - Website activity
- Similar Leak: Mate1 in February 2016:
  27 million records with even more personal details
  (e.g., drinking/drug habits, political views)

Outline

1. Motivation
2. Secure Software Development
3. Enabling Developers: From (Mild) Pain to Success
4. Lesson’s Learned

A Path Towards (More) Secure Software

SAP’s Secure Software Development Lifecycle (S2DL)

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

© 2007 LogicalHacking.com. Public (CC BY-NC-ND) Page 6 of 18
# A Path Towards (More) Secure Software

## SAP’s Secure Software Development Lifecycle (S²DL)

### 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

### 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
A Path Towards (More) Secure Software
SAP’s Secure Software Development Lifecycle (S\textsuperscript{2}DL)

Security Testing
- Dynamic Testing (e.g., IAST, DAST)
- Manual testing
- External security assessment

Security Validation (“First Customer”)
- Check for “flaws” in the implementation of the S\textsuperscript{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)

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 (S2DL)

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

Secure Software
A Path Towards (More) Secure Software

SAP’s Secure Software Development Lifecycle (S2DL)

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

Secure Software Development Lifecycle for Cloud/Agile

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

Outline

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

Finding Security Vulnerabilities

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

- Manual
  - Penetration Testing
  - DAST IAST Vulnerability Scanner
  - Manual Code Review

- Automatic
  - Static Analysis
  - SAST
  - SAST Vulnerability Scanner

In 2010: Static Analysis Becomes Mandatory

- SAST tools used:
  - Language
  - Tool
  - Vendor
  - ABAP
  - CodePro/filer
  - Virtual Forge HP

- 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
  - 2009
  - De-centralized approach
  - 2016
- One Two SAST tools fit all
  - VF CodePro/filer
  - Fortify

- 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 time with granting exemptions
  - Danger
  - Only ticking boxes
A De-Centralised Application Security Approach

Improving The Application Development Approaches

Governance & approvals

- De-centralized approach

2009

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

2016

Development Teams
- are empowered
- are responsible
                  Central Security Team
                  ▪ Supports development teams
                  ▪ Can focuses on improvements
                  ▪ filing white spots
                  ▪ tooling
                  ▪ processes

De-Centralised Application Security Approach

Improving The Application Development Approaches

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

Development teams
- Select technologies
- Select development model
- Design and execute security testing plan
- ...

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

De-Centralised Application Security Approach: Organisational Setup

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”

How to Measure Success (and Identify White Spots)

Listen to your developers
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
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

Outline

- 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

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)

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

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, Volker Lotz, and Edgar Weippl, editors, GI Sicherheit 2014, volume 228 of

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

### Tools
- **Tool A** (e.g., DAST)
- **Tool B** (e.g., IAST)

---

https://logicalhacking.com/blog/2023/03/22/sast-vs-dast-vs-iast/
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)

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