

# 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 15, 2017

...

```
callbackContext.sendPluginResult(new PluginResult
```

```
} else if ("delete".equals(action)) {
```

```
// do something
```

# Outline

---

1 Motivation

2 Secure Software Development

3 Enabling Developers: From (Mild) Pain to Success

4 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)

TalkTalk

- ❑ 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 (S<sup>2</sup>DL)

---





# A Path Towards (More) Secure Software

SAP's Secure Software Development Lifecycle (S<sup>2</sup>DL)



## 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 (S<sup>2</sup>DL)



## 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 (S<sup>2</sup>DL)



## 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 (S<sup>2</sup>DL)



## Secure Development

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

# A Path Towards (More) Secure Software

SAP's Secure Software Development Lifecycle (S<sup>2</sup>DL)



## 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 (S<sup>2</sup>DL)



## Security Validation ("First Customer")

- ❖ Check for "flaws" in the implementation of the S<sup>2</sup>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)

# A Path Towards (More) Secure Software

SAP's Secure Software Development Lifecycle (S<sup>2</sup>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

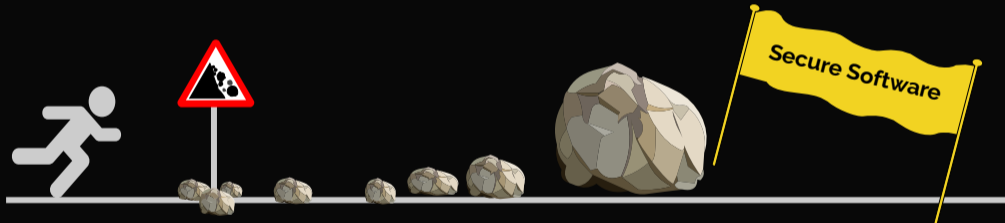
SAP's Secure Software Development Lifecycle (S<sup>2</sup>DL)





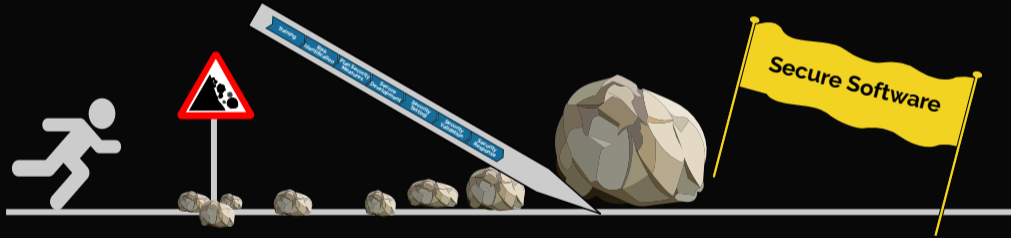
# A Path Towards (More) Secure Software

SAP's Secure Software Development Lifecycle (S<sup>2</sup>DL)



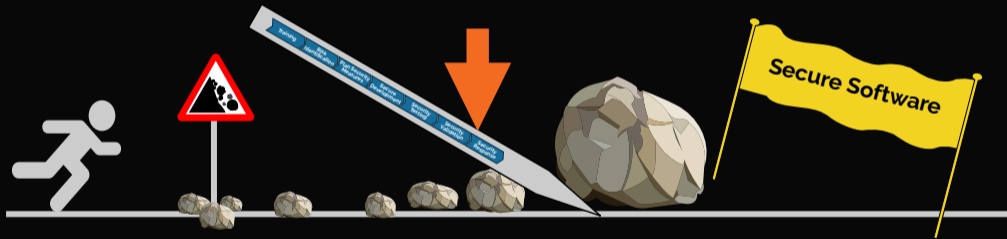
# A Path Towards (More) Secure Software

SAP's Secure Software Development Lifecycle (S<sup>2</sup>DL)



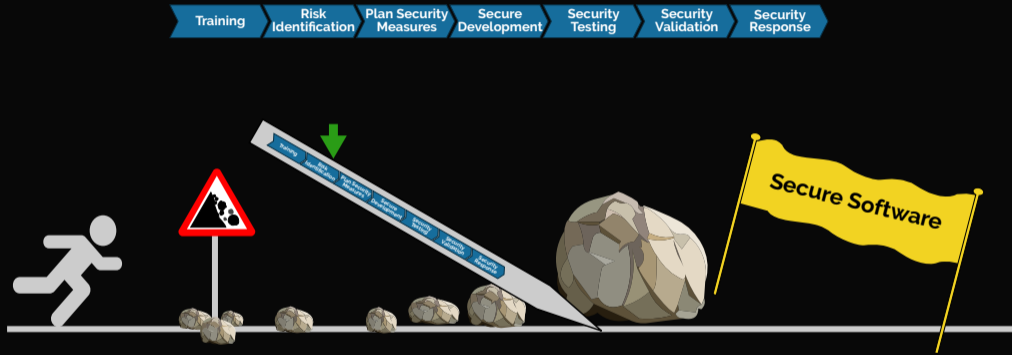
# A Path Towards (More) Secure Software

SAP's Secure Software Development Lifecycle (S<sup>2</sup>DL)



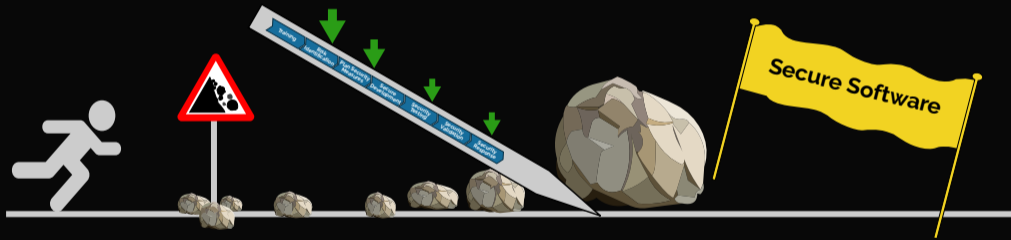
# A Path Towards (More) Secure Software

SAP's Secure Software Development Lifecycle (S<sup>2</sup>DL)

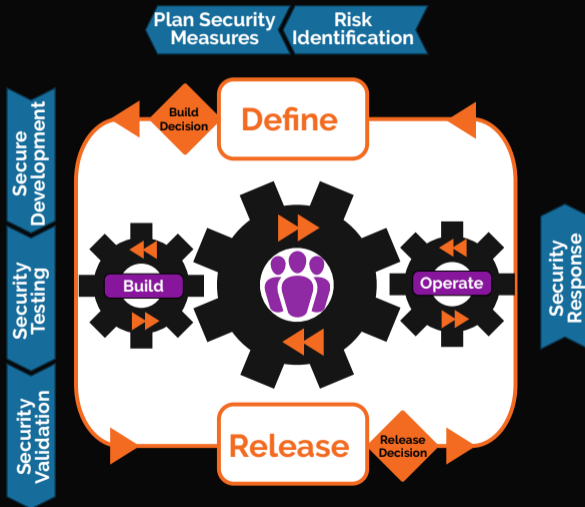


# A Path Towards (More) Secure Software

SAP's Secure Software Development Lifecycle (S<sup>2</sup>DL)



# Secure Software Development Lifecycle for Cloud/Agile

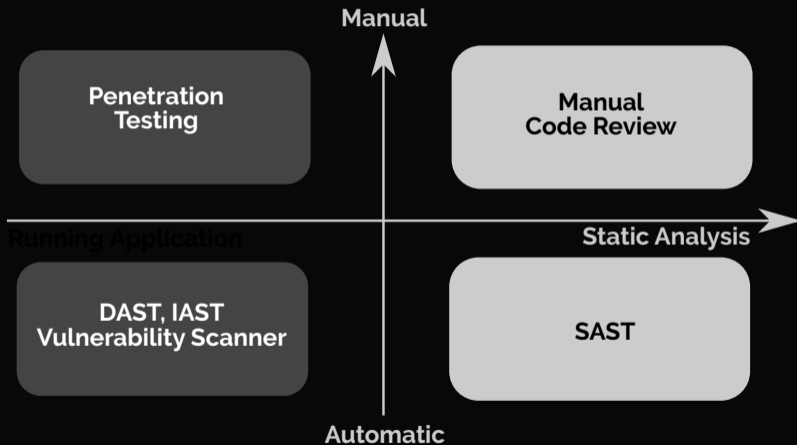


# Outline

---

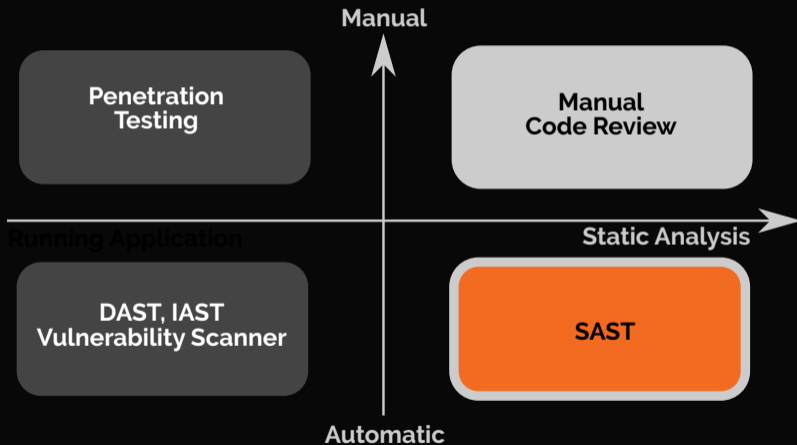
- 1 Motivation
- 2 Secure Software Development
- 3 Enabling Developers: From (Mild) Pain to Success
- 4 Lesson's Learned

# Finding Security Vulnerabilities



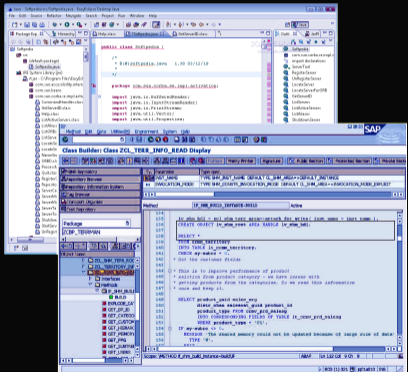


# Finding Security Vulnerabilities



# In 2010: Static Analysis Becomes Mandatory

SAST tools used:



Language	Tool	Vendor
ABAP	CodeProfiler	Virtual Forge
Others	Fortify	HP

- ❖ Since 2010: SAST mandatory for all products
- ❖ Within two years, multiple billions lines analysed
- ❖ Constant improvement of tool configuration
- ❖ Further details:  
Deploying Static Application Security Testing on a Large Scale. In GI Sicherheit 2014. Lecture Notes in Informatics, 228, pages 91-101, GI, 2014.

# A De-Centralised Application Security Approach

Improving The Application Development Approache

❖ Governance & approvals

❖ De-centralized approach

2009



2016

❖ ~~One~~ Two SAST tools fit all

- ❖ VF CodeProfiler
- ❖ 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

# A De-Centralised Application Security Approach

Improving The Application Development Approache

❖ 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

❖ Blending of Security Testing Tools

❖ Static:

SAP Netweaver CVA Add-on, Fortify, Synopsys Coverity, Checkmarx, Breakman

❖ Dynamic:

HP WebInspect, Quotium Seeker

❖ Others:

Burp Suite, OWASP ZAP, Codenomicon Defensics, BDD

# A De-Centralised Application Security Approach

Improving The Application Development Approache

❖ 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 (S<sup>2</sup>DL 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/>

# How to Measure Success (and Identify White Spots)

---

Listen to your developers



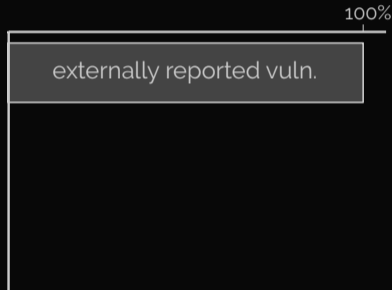
# How to Measure Success (and Identify White Spots)

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



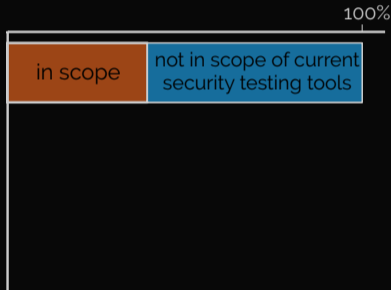
# How to Measure Success (and Identify White Spots)

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



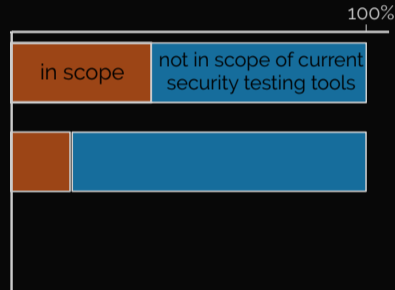
# How to Measure Success (and Identify White Spots)

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



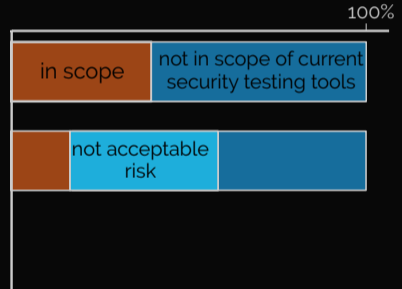
# How to Measure Success (and Identify White Spots)

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



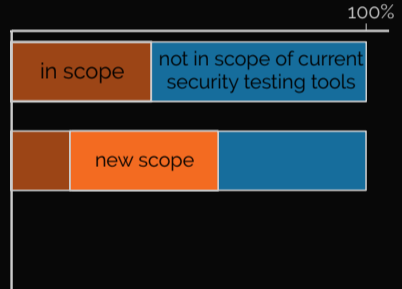
# How to Measure Success (and Identify White Spots)

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



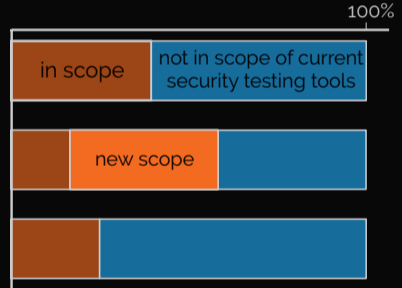
# How to Measure Success (and Identify White Spots)

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



# How to Measure Success (and Identify White Spots)

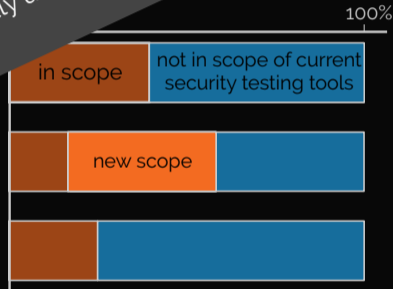
Non-working performance indicators include:

- ❑ Absolute number of reported vulnerabilities
- ❑ Absolute number of fixed issues

A new idea:

- ❑ Analyze the vulnerabilities reported by used tools
  - ❑ Security Validation
  - ❑ External security audits
- ❑ Two classes of vulnerabilities:
  - ❑ vulnerabilities detected by used tools
  - ❑ vulnerabilities not detected by used tools
- ❑ if risk acceptable: nothing to do
- ❑ if risk not acceptable: improve tooling

**"Success criteria:**  
Percentage of vulnerabilities not covered by currently used security testing tools increases, i.e., the used tools are used effectively!



# Outline

---

- 1 Motivation
- 2 Secure Software Development
- 3 Enabling Developers: From (Mild) Pain to Success
- 4 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

# 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 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 allow 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](mailto:a.brucker@sheffield.ac.uk)  
 [@adbrucker](https://twitter.com/adbrucker)  
 <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.  
*Datenschutz und Datensicherheit (DuD)*, 38(4):257–261, April 2014.



Achim D. Brucker and Uwe Sodan.

Deploying static application security testing on a large scale.  
In Stefan Katzenbeisser, Volkmar Lotz, and Edgar Weippl, editors, *GI Sicherheit 2014*, volume 228 of *Lecture Notes in Informatics*, pages 91–101. GI, March 2014.



Michael Felderer, Matthias Büchler, Martin Johns, Achim D. Brucker, Ruth Breu, and Alexander Pretschner.

Security testing: A survey.  
*Advances in Computers*, 101:1–51, March 2016.



# Document Classification and License Information

---

© 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).

# Combining Multiple Security Testing Methods and Tools

Web Client

Web Browser

Server Application

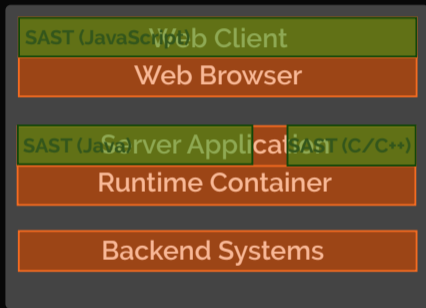
Runtime Container

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

<https://logicalhacking.com/blog/2017/01/11/sast-vs-dast-vs-iaast/>

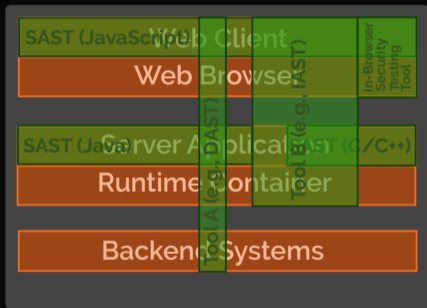
# Combining Multiple Security Testing Methods and Tools



- ❖ 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

<https://logicalhacking.com/blog/2017/01/11/sast-vs-dast-vs-iastr/>

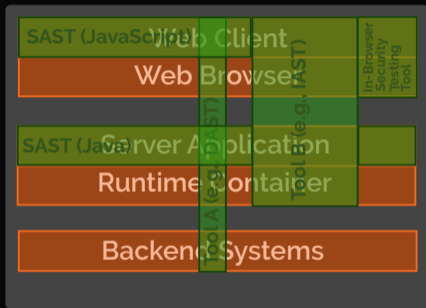
# Combining Multiple Security Testing Methods and Tools



- ❖ 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

<https://logicalhacking.com/blog/2017/01/11/sast-vs-dast-vs-iastr/>

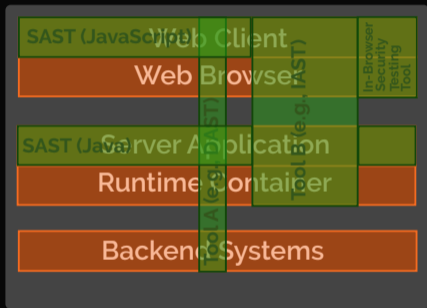
# Combining Multiple Security Testing Methods and Tools



- ❖ 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

<https://logicalhacking.com/blog/2017/01/11/sast-vs-dast-vs-iastr/>

# Combining Multiple Security Testing Methods and Tools



<https://logicalhacking.com/blog/2017/01/11/sast-vs-dast-vs-iaast/>

- ❖ 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)