Hybrid Taint Analysis for Java EE

Florian D. Loch, Martin Johns, Martin Hecker, Martin Mohr, Gregor Snelting

ACM SAC 2020, Brno, Czech Republic
The 35th Annual ACM Symposium on Applied Computing

SAP SE & Karlsruhe Institute of Technology (KIT)

About Me

- Florian Loch
  - Based in Karlsruhe, Germany
  - M. Sc. in computer science
  - Avid software engineer
  - Focus on applied security (research)
  - Currently taking a year off, looking for new opportunities in 2021
  - https://fdlo.ch

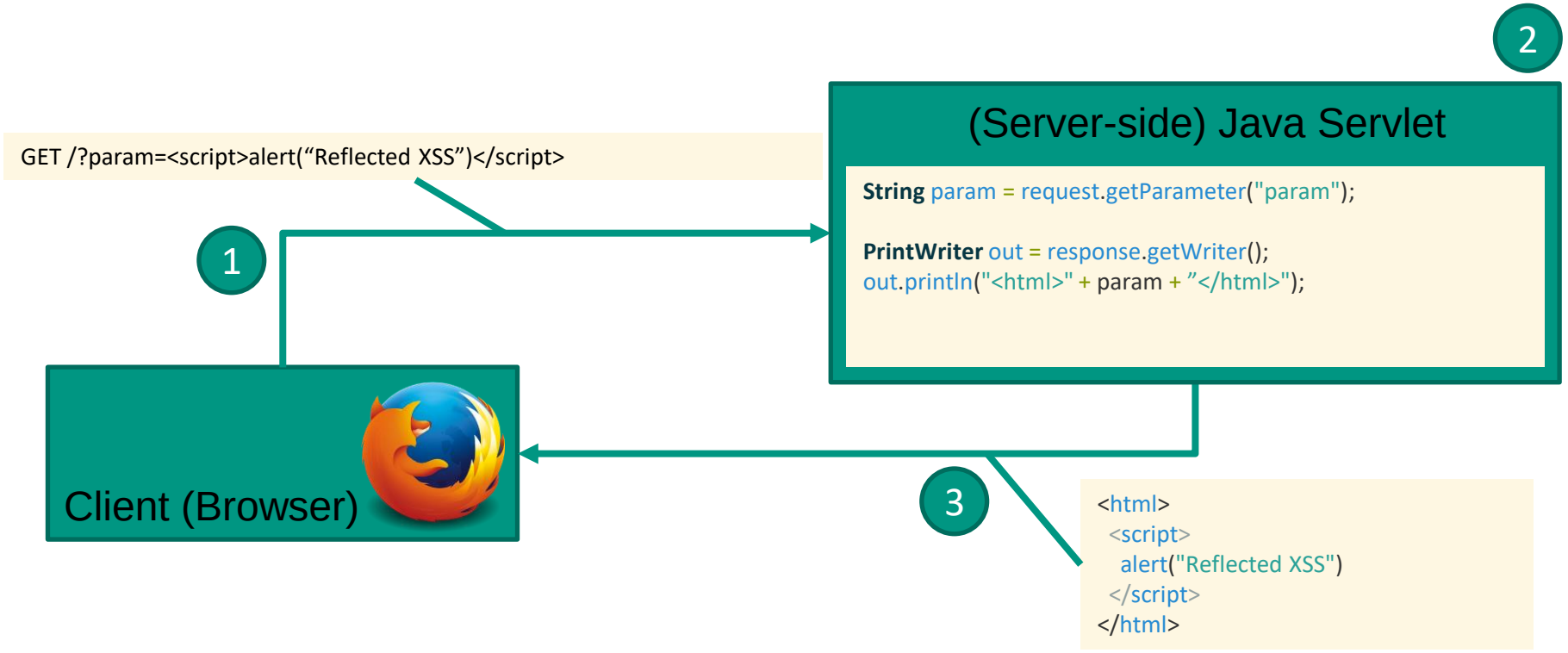
- This work is based on my master thesis *"Juturna: Lightweight, Pluggable and Selective Taint Tracking for Java"*

Motivation

- **Injection attacks** are emerging together with modern web applications
- → Attacker is able to control applications behaviour in malicious ways
- Underlying problem: **Improper Input Validation** (CWE-20)
- Many manifestations
  - Cross-Site-Scripting (XSS) (CWE-79)
  - OS Command Injection (CWE-78)
  - SQL Injection (CWE-89)
  - …

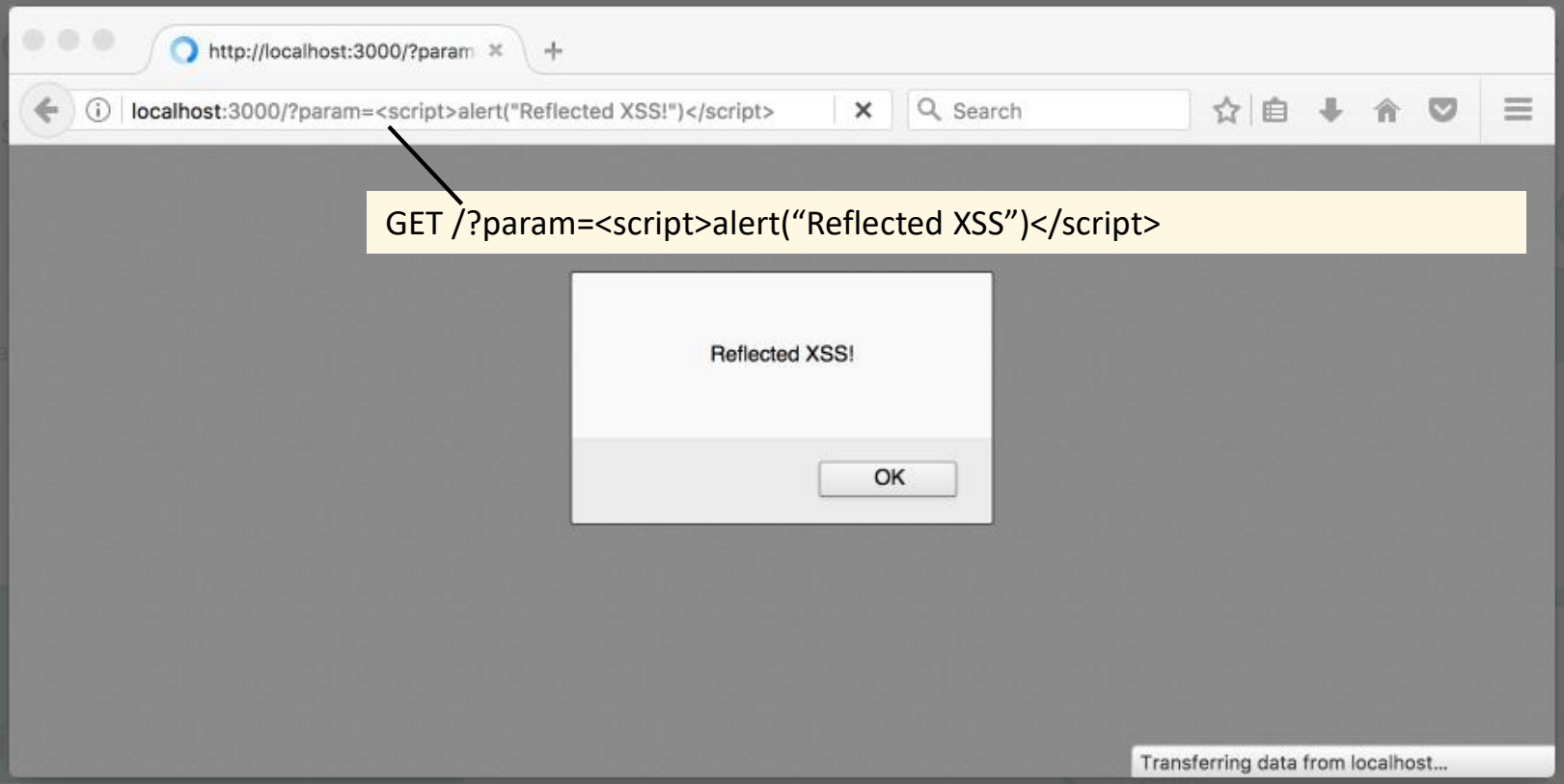
XSS: Injection Attacks in the Web

- XSS: *Cross-site Scripting*
- Goal: Attacker tries to execute JavaScript code in a users browser (in a specific security-context)



XSS: injection attacks in the Web

- XSS: *Cross-Site-Scripting*

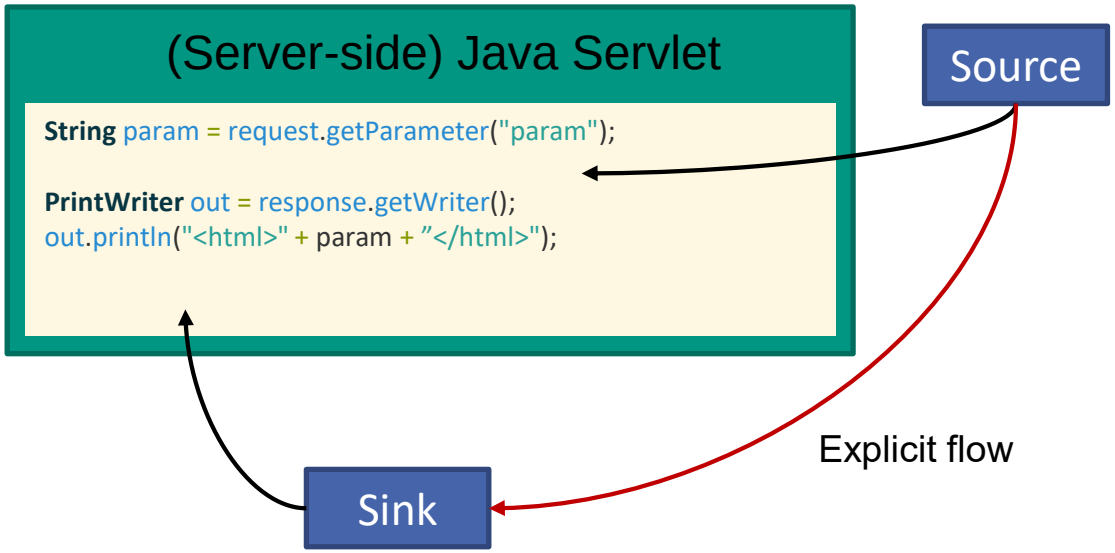


Analyzing the problem

- Untrusted input influences behavior of application inadvertently
- → Typical **Information Flow Control (IFC)** problem
  - Need to check application for **integrity**

- **Static** and **dynamic** techniques available
- Dynamic approach has been chosen as foundation, static methods as extension
- **Taint tracking** has been proven to be an effective measure to detect/block such attacks [2, 3, 4]
  - Adds metadata to a variable's content and tracks it during program execution

Analyzing the problem (II)



Current state

- Taint tracking is not a new concept, also there are implementations in Java
- But: present taint tracking systems got some serious drawbacks
  - Adds (massive) **memory and computation overhead**
  - Often needs **modifications to the runtime** on a low level; most nowadays implementations massively **touch the JRE or require special JREs/JVMs**
- Objective of this work was to tackle these issues and integrate new concepts

Implementation of the Taint Tracking System

- **Augmentation** of Java's standard class library
  - Adding code for storing and propagating taint information (**taint ranges**)
  - **String-only tracking**, considering the scenario this trade-off between **correctness**, **recall** and **performance** seems acceptable.
  - **Character-granularity** allows very high precision
  - Using "**bootclasspath override**", no modifications to JRE
  - Support for **Java Reflection**
- **Bytecode instrumentation**
  - Instrumenting sources, sinks and sanitization functions
  - Happens on-the-fly (**Java Agent**)**,** no pre-processing step needed

Selective Taint Tracking

- **Overhead intrinsic** to taint tracking, so how to reduce it significantly?
- Idea: reduce overhead by **reduced taint tracking/reduced augmentation**!
- Some parts of the application are not relevant for security and do not need to be taint-aware
  - Parts that provably never get in touch with tainted data
  - Parts operating on tainted-data that provably never reaches a sink
  - → Deactivate taint tracking for these parts
  - → But dynamic analysis cannot deliver these insights a priori

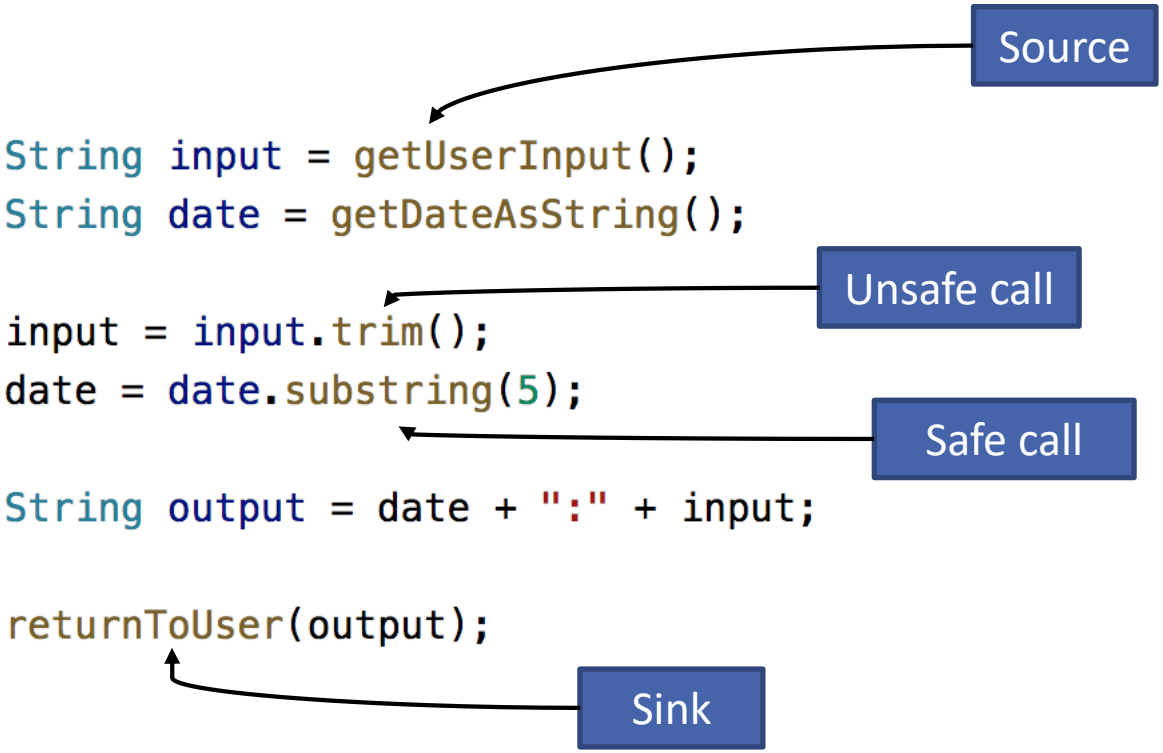
- → Use **static IFC** to determine such parts!

Selective Taint Tracking (II)

- Due to modified standard classes all operations on strings are taint-aware by default (“secure by default”)
- Invocations of an augmented method on a variable (not an instance!) v can be replaced with calls to an unaugmented method in case:
  - o $\notin chop(S_{source}, S_{sink})$; $\forall o \in pointsTo(v)$
  - S_{source} and S_{sink} are sets comprising all sources and sinks
- Such invocations can be determined by the JOANA-Adapter using JOANA’s PDG-based analysis

- An additional bytecode instrumenter modifies call instructions
- Prerequisite: Unaugmented methods need to be available

Selective Taint Tracking (III)

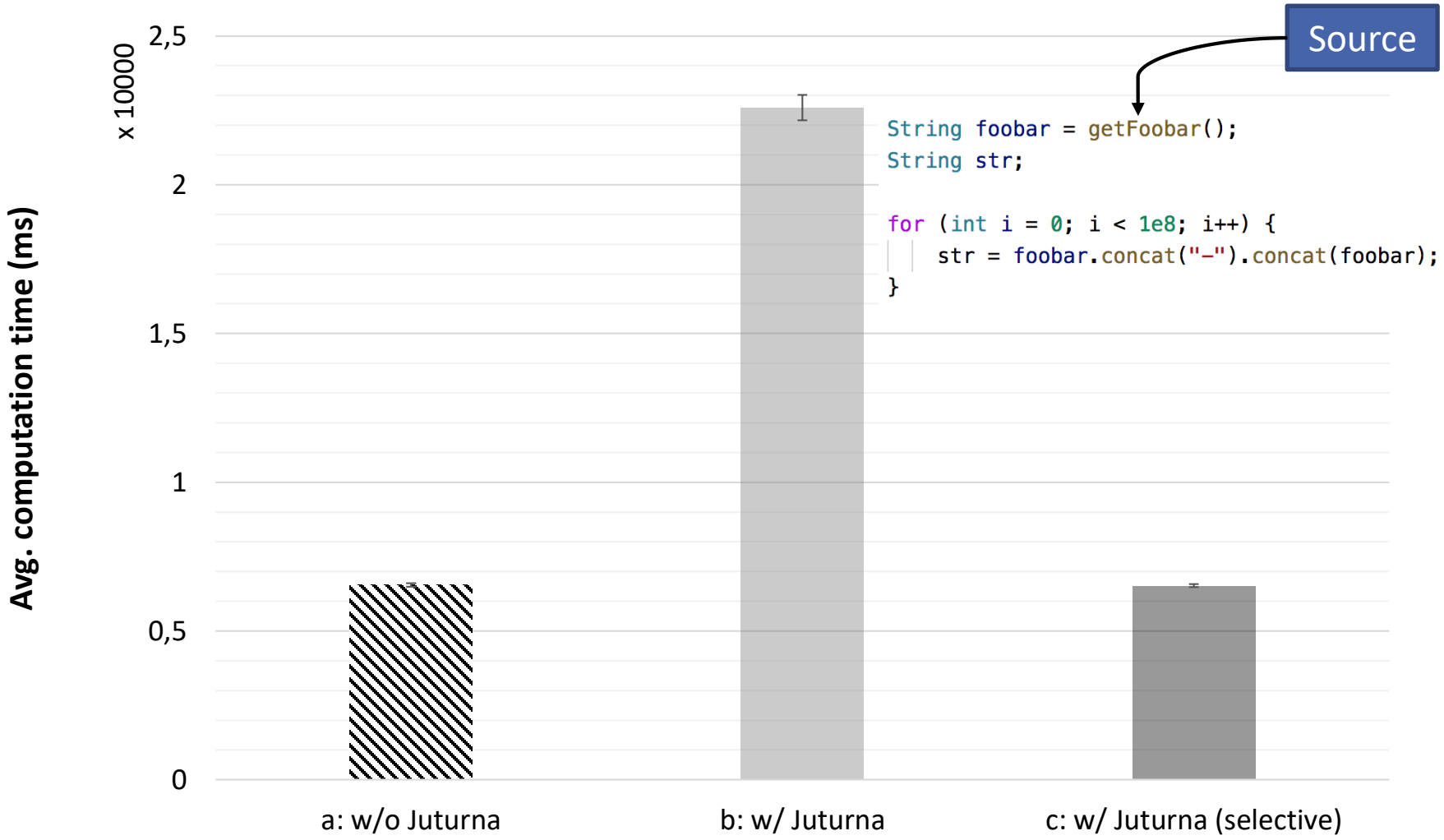


Selective Taint Tracking (IV)

```
 0 invokestatic #2 <Mixed.getUserInput>
 3 astore_1
 4 invokestatic #3 <Mixed.getDateAsString>
 7 astore_2
 8 aload_1
 9 invokevirtual #4 <java/lang/String.trim>
12 astore_1
13 aload_2
14 iconst_5
15 invokevirtual #5 <java/lang/String.substring>
18 astore_2
19 new #6 <java/lang/StringBuilder>
22 dup
23 invokespecial #7 <java/lang/StringBuilder.<init>>
26 aload_2
27 invokevirtual #8 <java/lang/StringBuilder.append>
30 ldc #9 <:>
32 invokevirtual #8 <java/lang/StringBuilder.append>
35 aload_1
36 invokevirtual #8 <java/lang/StringBuilder.append>
39 invokevirtual #10 <java/lang/StringBuilder.toString>
42 astore_3
43 aload_3
44 invokestatic #11 <Mixed.returnToUser>
47 return
```

```
 0 invokestatic #15 <Mixed.getUserInput>
 3 astore_1
 4 invokestatic #18 <Mixed.getDateAsString>
 7 astore_2
 8 aload_1
 9 invokevirtual #23 <java/lang/String.trim>
12 astore_1
13 aload_2
14 iconst_5
15 istore 5
17 astore 4
19 aconst_null
20 astore 6
22 aload 4
24 iload 5
26 invokevirtual #27 <java/lang/String.___substring>
29 astore 6
31 aload 6
33 astore_2
34 new #29 <java/lang/StringBuilder>
37 dup
38 invokespecial #30 <java/lang/StringBuilder.<init>>
41 aload_2
42 invokevirtual #34 <java/lang/StringBuilder.append>
45 ldc #36 <:>
47 invokevirtual #34 <java/lang/StringBuilder.append>
50 aload_1
51 invokevirtual #34 <java/lang/StringBuilder.append>
54 invokevirtual #39 <java/lang/StringBuilder.toString>
57 astore_3
58 aload_3
59 invokestatic #43 <Mixed.returnToUser>
62 return
```

Selective Taint Tracking (V)



Summary

- **Reduced memory footprint** by using taint ranges
- **Portable** and **non-invasive**, JRE never gets touched
- Flexible, configurable and extensible
- Evaluation of the taint tracking system showed **reasonable performance**
- Vulnerabilities in test cases have been detected, interoperation with Java EE Servlet Container is working
- **Hybrid approach** is working, depending on scenario it might reduce overhead massively
  - Joining Juturna's dynamic capabilities with JOANA's static analysis

Thank you for your kind attention!

Florian Loch
Feel free to contact me: me@fdlo.ch | https://fdlo.ch

References

[1] OWASP. *OWASP Ton 10 – 2017 rc1*. https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project, retrieved on 20th of May

[2] V. Haldar, D. Chandra and M. Franz. *Dynamic taint propagation for Java*. In: *Proceedings . Annual Computer Security Applications Conference, ACSAC*. 2005. pp 303-311.

[3] W. G. J. Halfond, A. Orso and P. Manolios. *Using Positive Tainting and Syntax-Aware Evaluation to Counter SQL Injection Attacks. 2006.*

[4] S. Lekies, B. Stock and M. Johns, *25 million flows later: large-scale detection of DOM-based XSS*. In: *Proceedings of the 2013 ACM SIGSAC conference on computer & communications security. 2013. pp. 1193-1204.*

[5] N. Patwardhan, E. Siever and S. Spainhour. *Perl in a Nutshell: A Desktop Quick Reference*. O'Reilly Media. 2002. ISBN: 0-596-00241-6.

[6] E. Chin and D. Wagner. *Efficient character-level taint tracking for Java*. In: *Proceedings of the 2009 ACM workshop on secure web services*. 2009. pp. 3-12.

[7] J. Bell and G. Kaiser. *Dynamic Taint Tracking for Java with Phosphor (Demo)*. 2015.

References

[8] M. Mongiovì et al.. *Combining static and dynamic data ow analysis : a hybrid approach for detecting data leaks in Java applications*. In: *Proceedings of the 30th Annual ACM Symposium on Applied Computing*. 2015, pp. 1573–1579.

[9] Jingling Zhao et al. *Dynamic taint tracking of web application based on static code analysis*. In: *Proceedings - 2016 10th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*, IMIS 2016. 2016, pp. 96–101.

[10] Christian Hammer and Gregor Snelting. *Flow-sensitive, Context-sensitive, and Object-sensitive Information Flow Control Based on Program Dependence Graphs*. In: *Int. J. Inf. Secur.* 8.6 (October 2009), pp. 399–422.

Pictorial Sources

Picture on title slide: https://www.pexels.com/de/foto/bohnen-koffein-kaffee-tasse-34079/

Icon of Firefox: The Mozilla Foundation, CC BY 3.0, https://commons.wikimedia.org/w/index.php?curid=29365482

Logo of JOANA: IPD Snelting, https://pp.ipd.kit.edu/projects/joana/joana-logo-250.png