Continuous Security Testing mit IAST

Matthias Rohr
Oktober 2019

Über mich

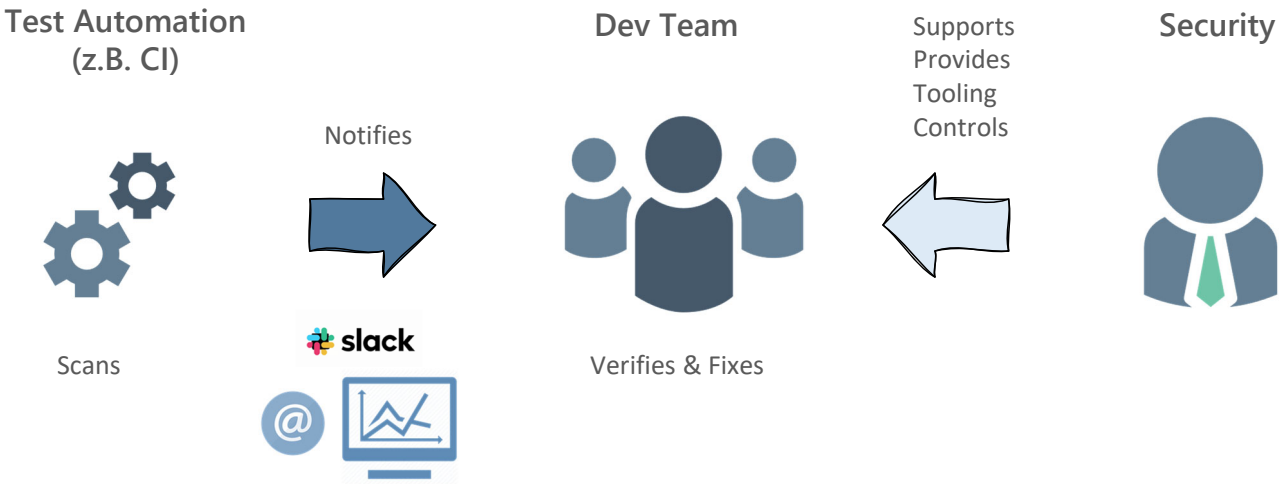
- In AppSec seit 2004 aktiv
- Security Architekt & Geschäftsführer @ Secodis
  - Application & Cloud Security Architektur
  - Integration von Sicherheit in Softwarentwicklung

Security Scanning (AST) Evolution

Variante 1: Zuständigkeit des Security Teams

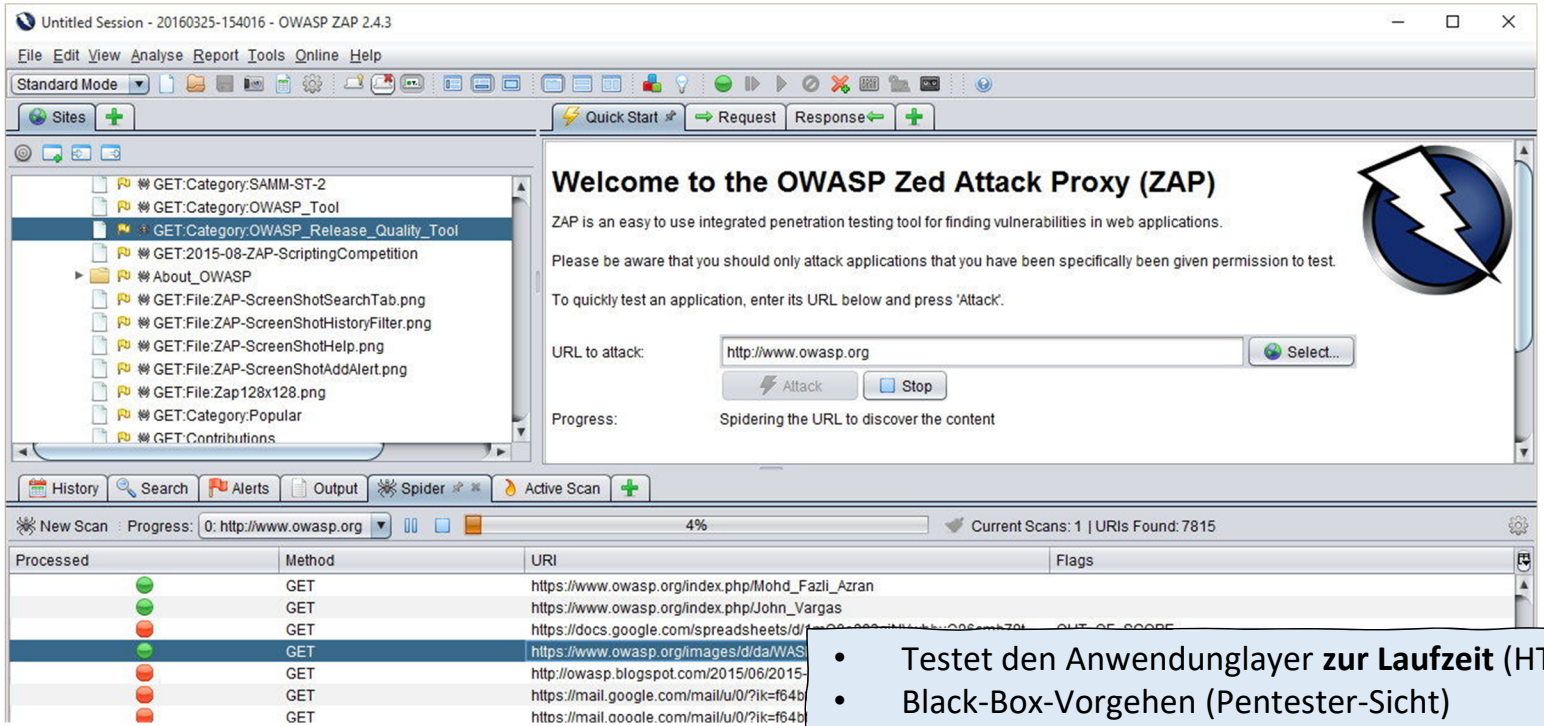
Security Scanning (AST) Evolution

Variante 2: Zuständigkeit der Dev Teams



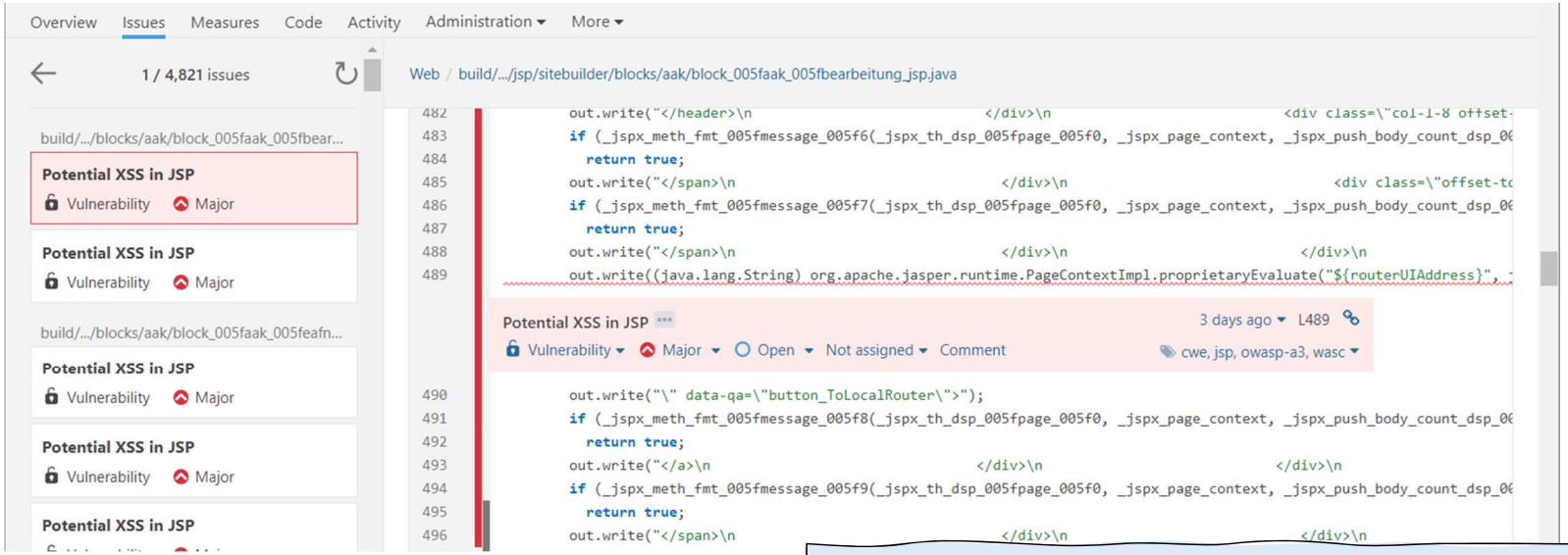
*"**Continuous (security) testing** is the process of executing automated **(security) tests** as part of the software delivery pipeline to obtain **immediate feedback** on the business risks associated with a software release candidate."*
(Wikipedia)

DAST – Web Security Scanning (App Layer)



- Testet den Anwendunglayer **zur Laufzeit** (HTTP)
- Black-Box-Vorgehen (Pentester-Sicht)

SAST – Static Code Security Scanning (Code Layer)



- Testet Source-, Binary- oder Byte-Code **statisch**
- White-Box-Vorgehen (Entwickler-Sicht)

OWASP Benchmark

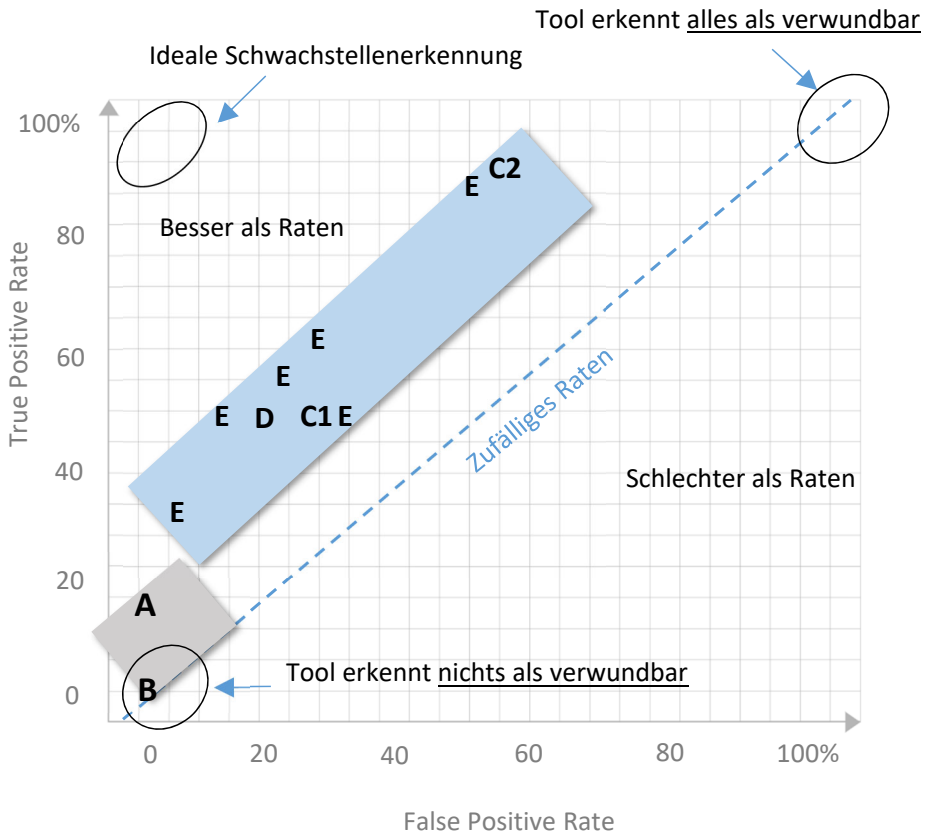
- 21.000 (v1.1) bzw. 2.700 (v1.2) Testcases einiger gängiger Implementierungsfehler in Webapps, die sich gut mit Scannern finden lassen.
- Nur native Java Servlets, keine Frameworks etc.

Vulnerability Area	# of Tests in v1.1	# of Tests in v1.2	CWE Number
Command Injection	2708	251	78
Weak Cryptography	1440	246	327
Weak Hashing	1421	236	328
LDAP Injection	736	59	90
Path Traversal	2630	268	22
Secure Cookie Flag	416	67	614
SQL Injection	3529	504	89
Trust Boundary Violation	725	126	501
Weak Randomness	3640	493	330
XPATH Injection	347	35	643
XSS (Cross-Site Scripting)	3449	455	79
Total Test Cases	21,041	2,740	

https://www.owasp.org/index.php/Benchmark

https://blog.secodis.com/2qp5

Benchmarking SAST und DAST



A: OWASP ZAP 2016-09-05 (DAST, 18%), v1.2

B: PMDv5.2.3, Findbugs v3.01 (reines SCA, kein SAST!, 0%)

+ div. komm. DAST-Tools, v1.2 / v1.1

C1: FindbugsSecPlugin v1.4.0 (SAST, 12%), v1.2

C2: FindbugsSecPlugin v1.4.6 (SAST, 39%), v1.2

D: SonarQube Java Plugin v3.5 (SAST, 33%),v1.2

E: Kommerzielle SAST-Tools (~ 23%), v1.1

DAST vs. SAST

Pro DAST

- Funktioniert gut für (sehr) einfache WebApps
- Funktioniert für Low Hanging Fruits (z.B. Security Header)
- Funktioniert gut gegen Standard-Apps (z.B. Wordpress)
- Funktioniert mit allen server-seitigen Sprachen, Frameworks
- Findings sind häufig besser nachvollziehbar (exploitable)

Pro SAST

- Kann von Entwicklern früh im SDLC eingesetzt warden
- Gute Integrationen in IDEs / Sonar
- Gut Integrierbar in CI / CD (Testen von Artefakten)
- Erfordert keine voll-integrierte Anwendung, kann häufig auch nur Snippets testen

Kontra DAST

- **Versteht keine Geschäftslogik** – muss „antrainiert" werden, z.B.
  - Authentifizierung
  - Mehrstufige Formulare
  - Priviligierte Aktionen (z.B. Admin-Bereiche)
  - Scannen von JS / CLient-Logik ist schwierig
- Erfordert eine voll-integrierte ausgeführte Anwendung (spät im SDLC)

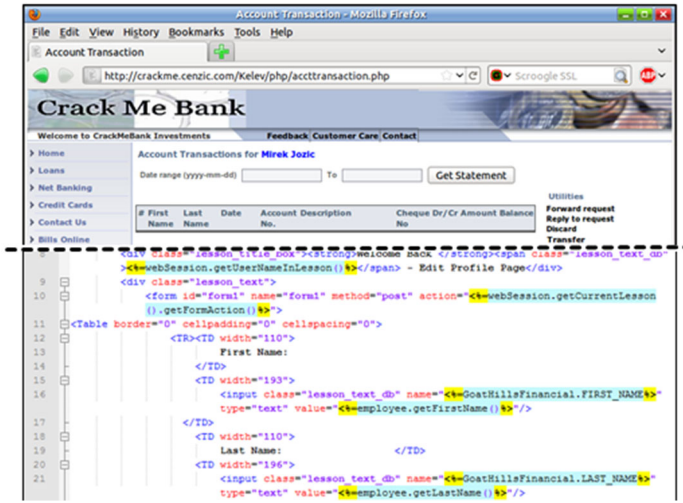
Kontra SAST

- **Versteht keinen Ausführungskontext**. z.B.
  - Ist der Code in Prod deployed oder nur Testcode?
  - Ist die Funktion exponiert?
  - Welche Daten enthält eine Variable?
  - Kann den Daten vertraut werden?
- Scans von 3rd-Party-Komponenten (inkl. Code anderer Teams) problematisch

Tip 1: Differences Between Web Application Scanning Tools when Scanning for XSS and SQLi , AppSecUSA 2017
Tip 2: Practical Tips for Defending Web Applications in the Age of DevOps, BlackHat 2017

AppSec Test Layers

Schwachstellen im Application Layer



Session Mgmt / CSRF,
Insecure client-side,
Business Logic Errors,
Missing Anti Autom.,
…

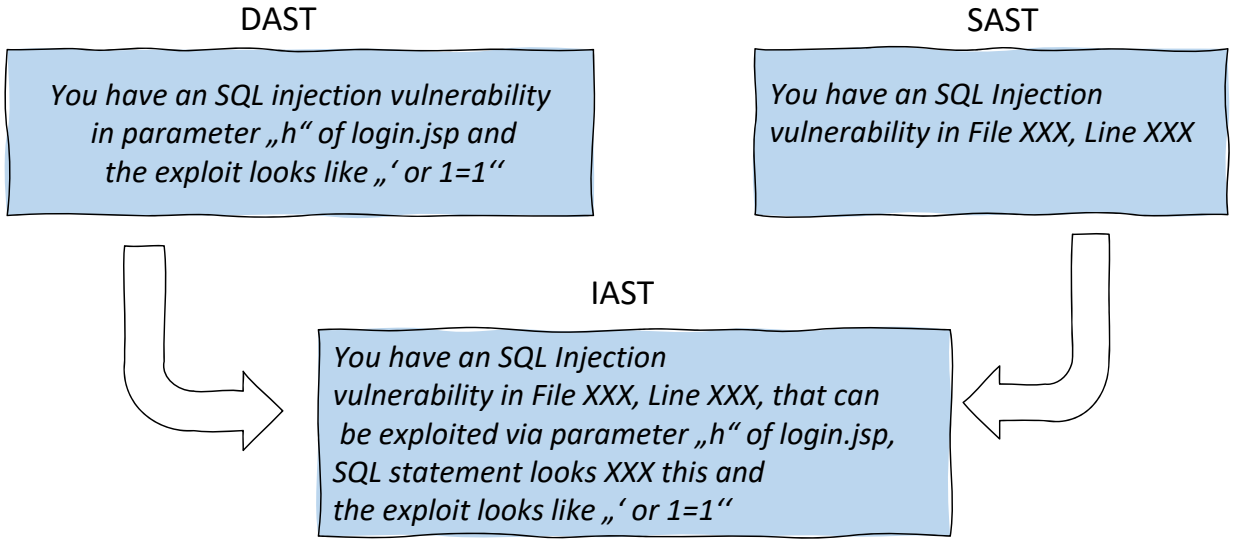
Race Conditions
Buffer Overflows
Backdoors,
Insecure APIs,
Backend,
…

XSS,
SQL Injection,
Datavalidation
Authentication,
Access Controls,
Cryptography,
Inform. Leakage,
Error Handling,
Configuration,
…

Schwachstellen im Code Layer

IAST = Dynamic Security Code Scanning

- Security Code Analyse zur Laufzeit (= dynamisch)
- Kombination von DAST- und SAST-Technologien:



- In der Regel mittels **Instrumentierung**

Wie funktioniert IAST-Instrumentierung?

Was ist Instrumentierung?

“Software instrumentation is a technique that is widely used in software profiling, performance analysis, optimization, testing, error detection, and virtualization.

*Instrumentation, which involves **adding extra code to an application for monitoring some program behavior**, can be performed either statically (i.e., at compile time) or **dynamically (i.e., at runtime)**.”*

Wiley Encyclopedia of Computer Science and Engineering
Torsten Kempf Kingshuk Karuri Lei Gao (2008)

Instrumentierung am Beispiel von JavaAssist

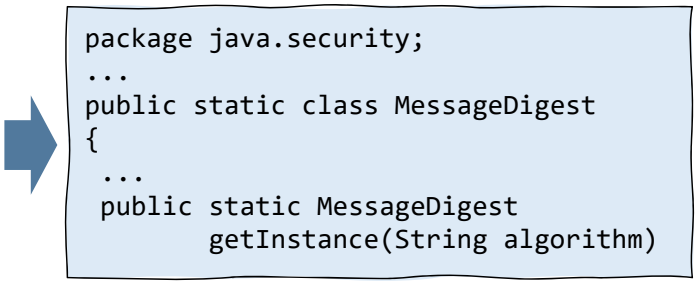
```
ClassPool cPool = ClassPool.getDefault();

// 1. Defining class to instrument
String instrumentedClassName = "java.security.MessageDigest";
CtClass ctClass = cPool.get(instrumentedClassName);

// 2. Defining Method to instrument
String instrumentedMethodName = "getInstance";
String instrumentedMethodDescriptor = "(Ljava/lang/String;)V";
CtMethod ctClassMethod = ctClass.getMethod(instrumentedMethodName, instrumentedMethodDescriptor);

// 3. Injecting code
ctClassMethod.insertBefore("System.out.println(\"[Instrumentation] Entering instrumented method\");");
ctClassMethod.insertAfter("System.out.println(\"[Instrumentation] Exiting instrumented method\");");

// 4. Instrument
ExprEditor instrumentationExpressionEditor = new DemoExpressionEditor();
ctClassMethod.instrument(instrumentationExpressionEditor);
ctClass.toClass();
```



Erkennung von unsicherer API mittels Instrumentierung

java.security.MessageDigest

```
// Before instrumentation
public static MessageDigest getInstance(String algorithm) throws NoSuchAlgorithmException {
        ...
}
```

java.security.MessageDigest (instrumentiert)

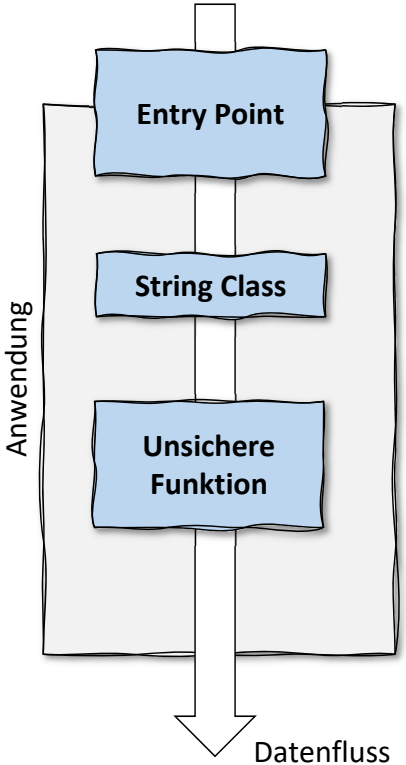
```
// After instrumentation
public static MessageDigest getInstance(String algorithm) throws NoSuchAlgorithmException {
        if (algorithm.equals("MD5") {
                StackTraceElement[] stack = Threat.currentThread().getStackTrace();
                Tracker.report("Insecure hash function (MD5)", stack);
        }
        ...
}
```


Callstack

```
at com.secodis.example.webapp.VulnerableClass.method(VulnerableClass:322)
...
at org.springframework.web.servlet.FrameworkServlet.doGet(FrameworkServlet.java:870)
```

Vereinfachte IAST-Instrumentierung mit JAVA

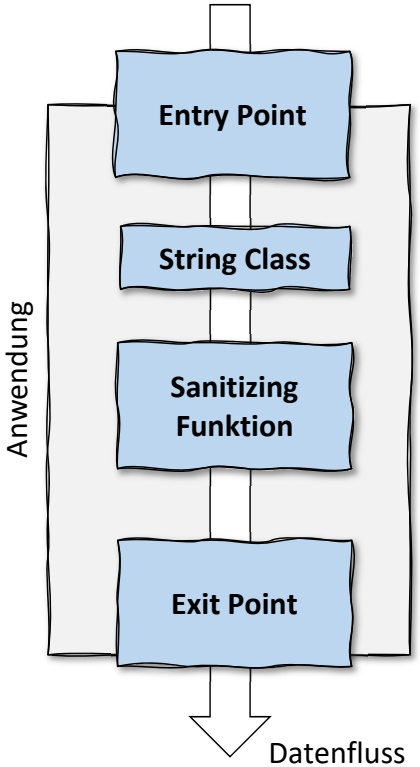
Variante 1: Unsichere Funktion / API



- Instrumentierung von Eintrittspunkten (Entry Points / für Kontext-Information)
  - org.apache.catalina.connector.RequestFacade.getParameterValues(java.lang.String)
  - ...
- Instrumentierung von String-Klasse (für Kontext-Information / Taints)
  - java.lang.String
- Instrumentierung unsicherer Funktionen / APIs
  - java.security.MessageDigest.getInstance(java.lang.String)
  - ...

Vereinfachte IAST-Instrumentierung mit JAVA

Variante 2: Unsicherer Datenfluss

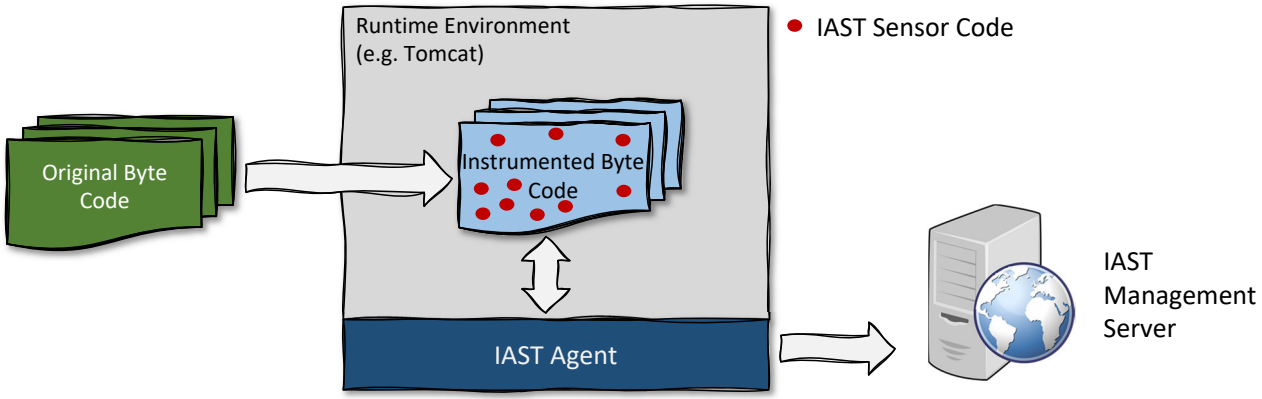


- Instrumentierung von Eintrittspunkten (Entry Points)
  - org.apache.catalina.connector.RequestFacade.getParameterValues(java.lang.String)
  - ...
- Instrumentierung von String-Klasse
  - java.lang.String
- Instrumentierung von Sanitizing-Funktionen
  - org.hibernate.Query, setParameter(int position, Object val)
  - org.apache.commons.text.StringEscapeUtils.escapeHtml4().
  - ...
- Instrumentierung von Austrittspunkten (Exit Points)
  - com.mysql.jdbc.StatementImpl method boolean execute(String sql)
  - ... org.apache.catalina.connector.CoyoteWriter.print(java.lang.String)

IAST in der Praxis

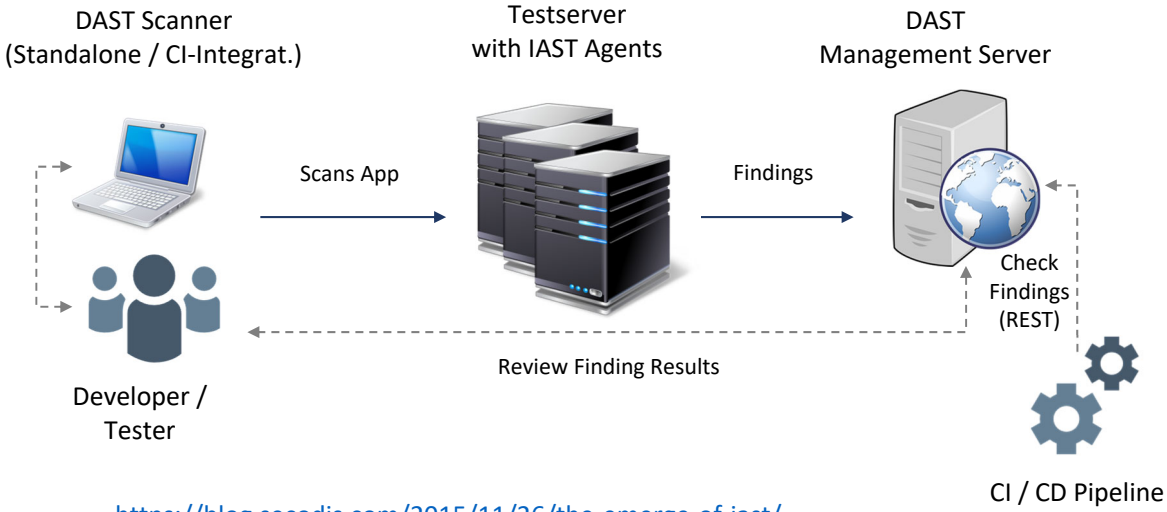
IAST = Dynamic Security Code Scanning

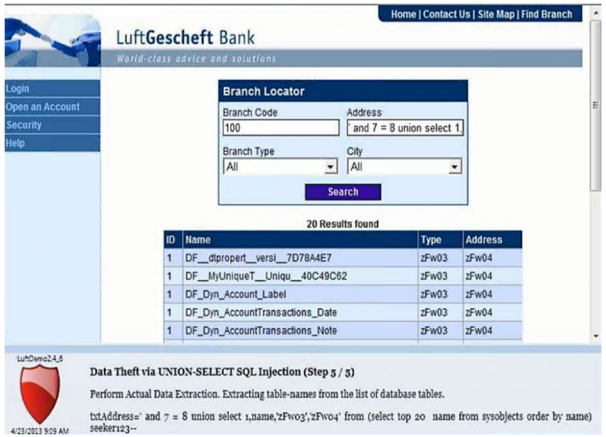
- Kombination von DAST- und SAST-Technologien.
- Funktioniert in der Regel mit **Agenten**, die in die Laufzeitumgebung (JVM oder .NET CLR) den Code **instrumentiert** und zur Laufzeit auf Sicherheitsproblem analyisieren.
- **RASP** = Runtime Protection („Embedded WAF“), oft auf Basis von IAST-Technologien.



Active IAST

- Aka „IAST Light“
- Arbeitet als **Agent für DAST-Scanner**
- Dient primär zur Verbesserung der Qualität von DAST Scannern, löst aber nicht deren zentrale Probleme

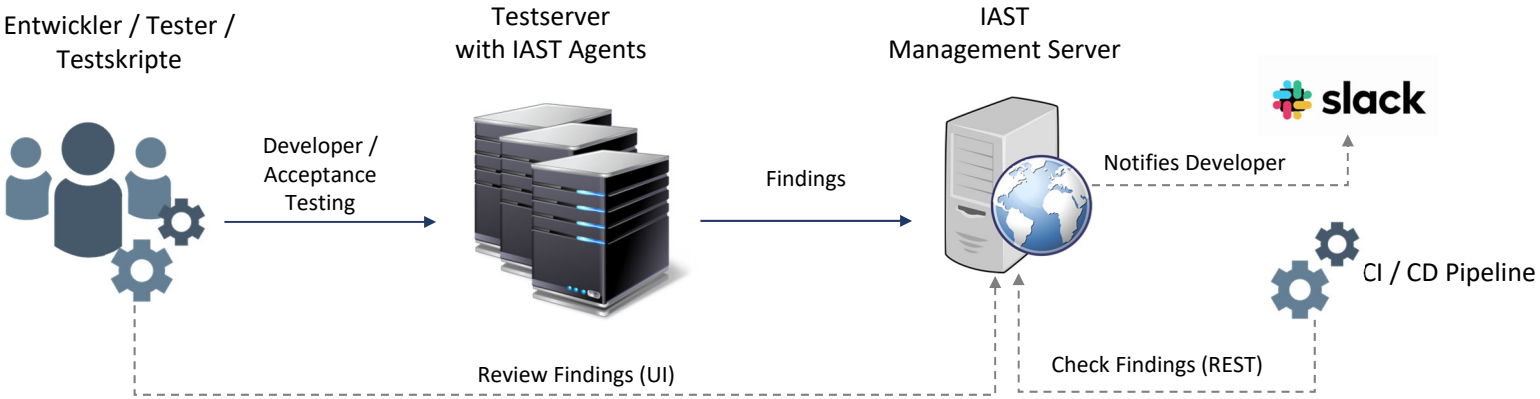




https://blog.secodis.com/2015/11/26/the-emerge-of-iast/

Passive IAST

- Aka „Full IAST“
- Rein **passives Scannen**, ohne DAST
- Keine spezifischen Security-Tests erforderlich
- Wenige Produkte am Markt, in der Regel mit RASP verbunden



https://blog.secodis.com/2015/11/26/the-emerge-of-iast/

IAST - Timeline

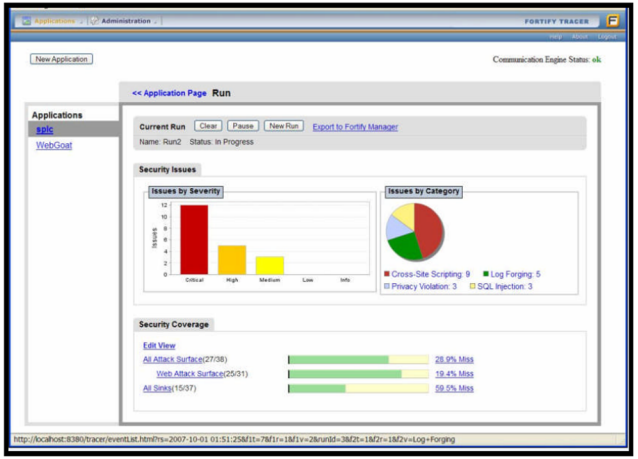
~ **2007** Fortify veröffentlicht Tracer / Program Trace Analyser (PTA)

~ **2011** IAST technology adaptiert bei ersten DAST-Herstellern (IBM und Acunetix)

~ **2011** Gartner führt Begriff IAST als DAST-Variante ein

~ **2013** Erste reine IAST-Hersteller

~ **2018** Die meisten SAST- und DAST-Hersteller bieten IAST-Lösungen an, zwei reine IAST-Hersteller



Unterstützte Technologien (Kommuliert)

- Java
- Weitere JVM-Dialekte (Scala, Groovy, Closure)
- .NET Framework
- .NET Core
- JavaScript (Node.js)
- Teilweise Ruby
- Teilweise Python
- Teilweise PHP (via FPM, FastCGI oder proxy_pass)

OWASP Top Ten Coverage

A1 – Injection 

A2 – Broken Authentication 

A3 – Sensitive Data Exposure (teilweise)

A4 – XML External Entities (XXE) 

A5 – Broken Access Control 

A6 – Security Misconfiguration 

A7 – Cross-Site Scripting (XSS) 

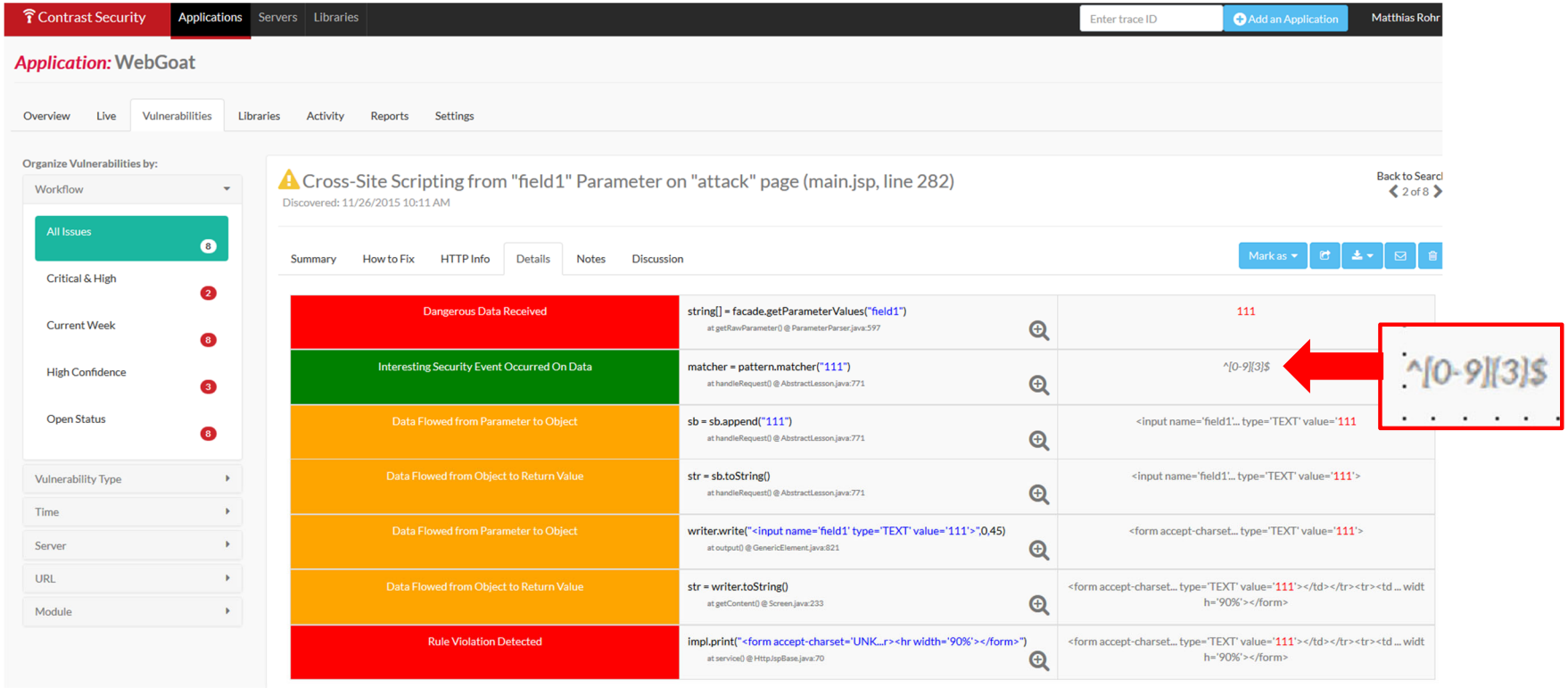
A8 – Insecure Deserilization 

A9 – Using Components with Known Vuln. 

A10 – Insufficient Logging & Monitoring 

DEMO

IAST-Limitationen / Bespiel



IAST vs. SAST

- Erfordert ausfühbare Anwendung
- Erfordert Integration (bei SAST optional)
- Bei Passive IAST müssen Routen / Datenflüsse durch externe Tests getriggert werden (Coverage = Coverage der Testcases)
- Gewöhnlich weniger Regeln
- Beschränkt auf serverseitigen Code
- Vor allem für Web-basierte Technologien gegeignet (kein C/C++ etc. möglich)

Fazit

- IAST is weniger Ersatz als mehr eine **sinvolle Ergänzung** zu SAST und dynamischen Tests, insb. für agil-entwickelte Anwendungen geeignet.
- In Kombination mit IAST lassen andere Tools auf Scan-Regeln mit geringer FPR einschränken.
- AST-Tools generell als **Safety Net** sinnvoll! Können nicht ersetzen:
  - Sichere Architektur
  - Secure Defaults
  - Threat Modeling in Teams
  - Peer Code Reviews in Teams
  - Pentests
  - Secure Coding Know-how
  - Security Culture
  - ....

Danke! Fragen?

Matthias Rohr
m.rohr@secodis.com