

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 Softwareentwicklung



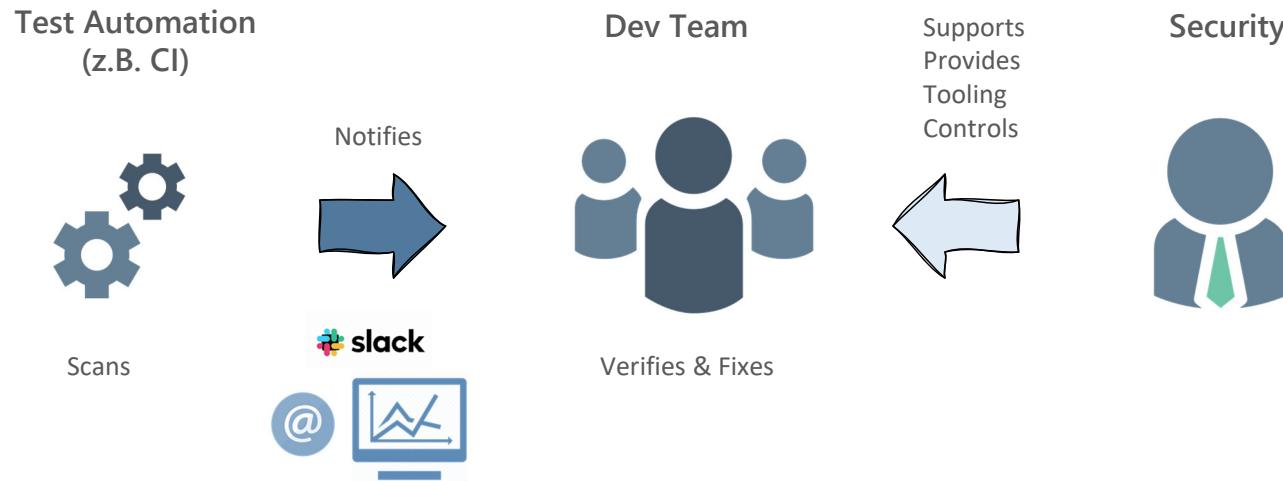
Security Scanning (AST) Evolution

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)

The screenshot shows the OWASP Zed Attack Proxy (ZAP) version 2.4.3 interface. The main window displays the "Welcome to the OWASP Zed Attack Proxy (ZAP)" page, which includes a lightning bolt logo and instructions for attacking applications. On the left, a tree view shows the URLs being crawled, including "GET:Category:OWASP_Release_Quality_Tool". The bottom half of the interface shows the "Spider" tab with a table of results. The table has columns for "Processed", "Method", "URI", and "Flags". Several rows are listed, with the last row highlighted in blue. The "URI" column for the highlighted row contains several URLs. A callout box on the right side of the interface lists two bullet points:

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

Processed	Method	URI	Flags
●	GET	https://www.owasp.org/index.php/Mohd_Fazli_Azran	
●	GET	https://www.owasp.org/index.php/John_Vargas	
●	GET	https://docs.google.com/spreadsheets/d/1Cw...	OUT OF SCOPE
●	GET	https://www.owasp.org/images/d/da/WAS...	
●	GET	http://owasp.blogspot.com/2015/06/2015...	
●	GET	https://mail.google.com/mail/u/0/?ik=f64b	
●	GET	https://mail.google.com/mail/u/0/?ik=f64b	

SAST - Static Code Security Scanning (Code Layer)

The screenshot shows a static code security scanning interface with the following details:

- Header:** Overview, Issues (selected), Measures, Code, Activity, Administration ▾, More ▾
- Breadcrumbs:** Web / build/.../jsp/sitebuilder(blocks/aak/block_005faak_005fbearbeitung.jsp.java)
- Issues List:** 1 / 4,821 issues
- Issue 1:** Potential XSS in JSP (Vulnerability, Major)
Details: 3 days ago, L489, cwe, jsp, owasp-a3, wasc
Code snippet:

```
out.write("</header>\n");  
if (_jspx_meth_fmt_005fmessage_005f6(_jspx_th_DSP_005fpage_005f0, _jspx_page_context, _jspx_push_body_count_DSP_005f0))  
    return true;  
out.write("</span>\n");  
if (_jspx_meth_fmt_005fmessage_005f7(_jspx_th_DSP_005fpage_005f0, _jspx_page_context, _jspx_push_body_count_DSP_005f0))  
    return true;  
out.write("</span>\n");  
out.write((java.lang.String) org.apache.jasper.runtime.PageContextImpl.proprietaryEvaluate("${routerUIAddress}"));  
out.write("<\" data-qa=\"button_ToLocalRouter\">");
```
- Issue 2:** Potential XSS in JSP (Vulnerability, Major, Open, Not assigned, Comment)
Details: 3 days ago, L489, cwe, jsp, owasp-a3, wasc
Code snippet:

```
if (_jspx_meth_fmt_005fmessage_005f8(_jspx_th_DSP_005fpage_005f0, _jspx_page_context, _jspx_push_body_count_DSP_005f0))  
    return true;  
out.write("</a>\n");  
if (_jspx_meth_fmt_005fmessage_005f9(_jspx_th_DSP_005fpage_005f0, _jspx_page_context, _jspx_push_body_count_DSP_005f0))  
    return true;  
out.write("</span>\n");
```
- Bottom Summary:** Testet Source-, Binary- oder Byte-Code **statisch**, White-Box-Vorgehen (Entwickler-Sicht)

OWASP Benchmark

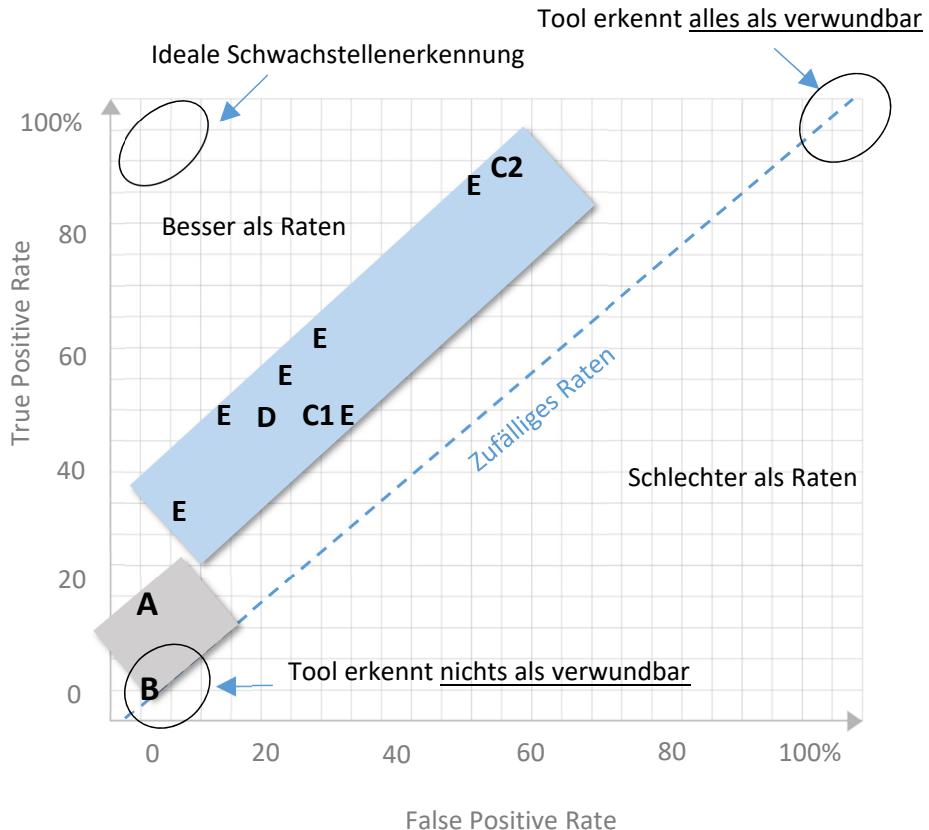
- 21.000 (v1.1) bzw. 2.700 (v1.2) Testcases einiger gängiger Implementierungsfehler in Webapps, die sich gut mit Scannern finden lassen.
- Nur native Java Servlets, keine Frameworks etc.

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.secidis.com/2qp5>

Benchmarking SAST und DAST



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

B: PMDv5.2.3, Findbugs v3.0.1 (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 werden
- 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
 - Privilegierte Aktionen (z.B. Admin-Bereiche)
 - Scannen von JS / Client-Logik ist schwierig
- Erfordert eine voll-integrierte ausgeführte Anwendung (spät im SDLC)

Kontra SAST

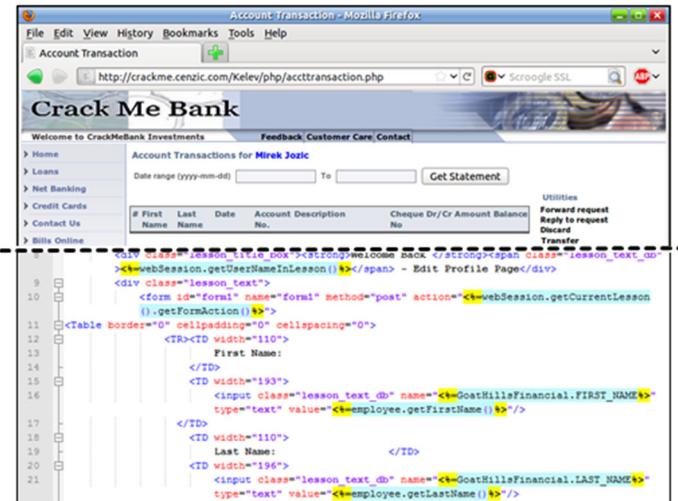
- **Versteht keinen Ausführungskontext.** z.B.
 - Ist der Code in Prod deployed oder nur Testcode?
 - Ist die Funktion exponiert?
 - Welche Daten enthält eine Variable?
 - Kann den Daten vertraut werden?
- Scans von 3rd-Party-Komponenten (inkl. Code anderer Teams) problematisch

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



The screenshot shows a Firefox browser window titled "Account Transaction - Mozilla Firefox". The URL is <http://crackme.cenzic.com/Kelev/php/acctransaction.php>. The page displays account transactions for a user named "Mirek Jozic". Below the UI, a code editor shows the corresponding PHP source code for the transaction page. The code includes session management logic, form handling, and database access via a class named "GoatHillsFinancial". Several lines of code are highlighted in yellow, indicating specific points of interest or vulnerability.

```
<strong><span> session_text_db</span> - Edit Profile Page</strong>
<div class="lesson_text">
<form id="form1" name="form1" method="post" action="<?php $webSession = <?php $webSession->getCurrentLesson();
$lesson = $webSession->getLesson(); $lesson->getFormAction(); ?>">
<input type="text" value="<?php $employee = $lesson->getEmployee();
echo $employee->getFirstName(); ?>"/>
<input type="text" value="<?php $employee = $lesson->getEmployee();
echo $employee->getLastName(); ?>"/>

```

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

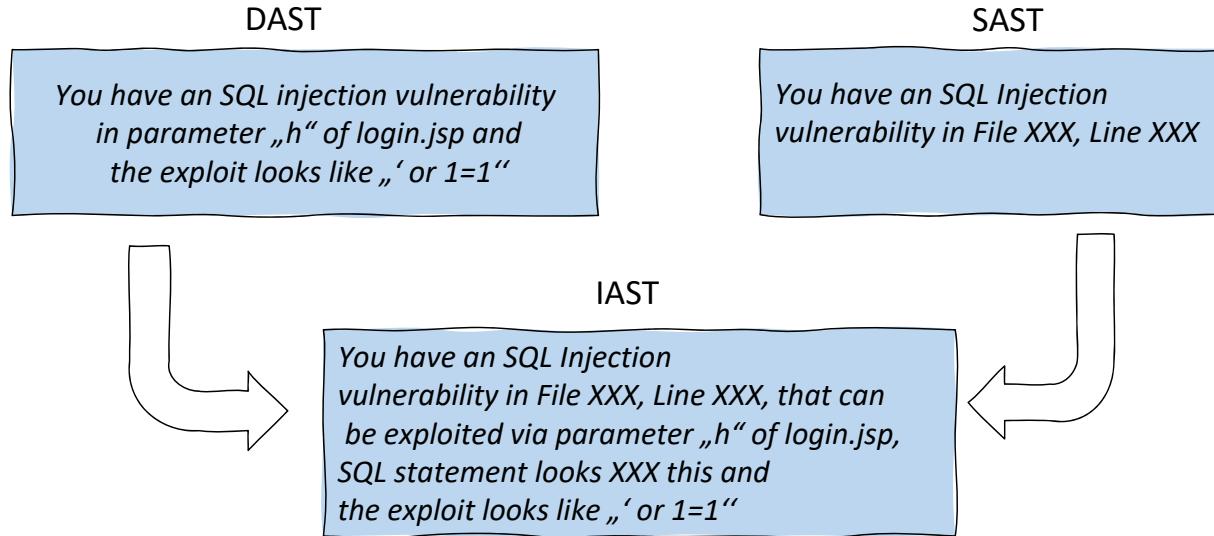
Race Conditions
Buffer Overflows
Backdoors,
Insecure APIs,
Backend,
...

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();
```



```
package java.security;
...
public static class MessageDigest
{
    ...
    public static MessageDigest
        getInstance(String algorithm)
```

Erkennung von unsicherer API mittels Instrumentierung

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

java.security.MessageDigest

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

java.security.MessageDigest (instrumentiert)

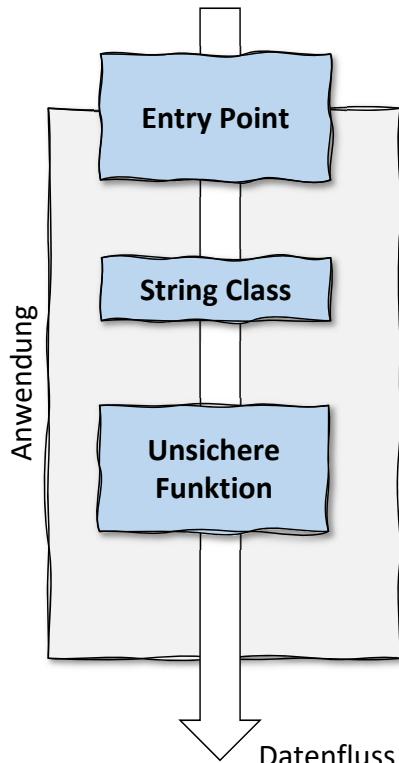
Callstack

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



Vereinfachte IAST-Instrumentierung mit JAVA

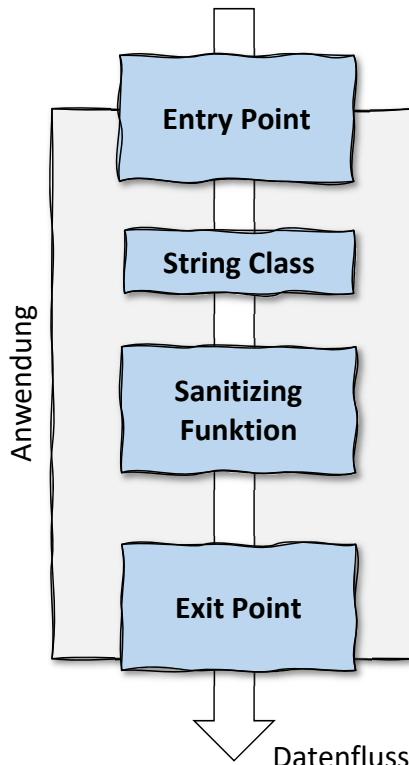
Variante 1: Unsichere Funktion / API



- 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

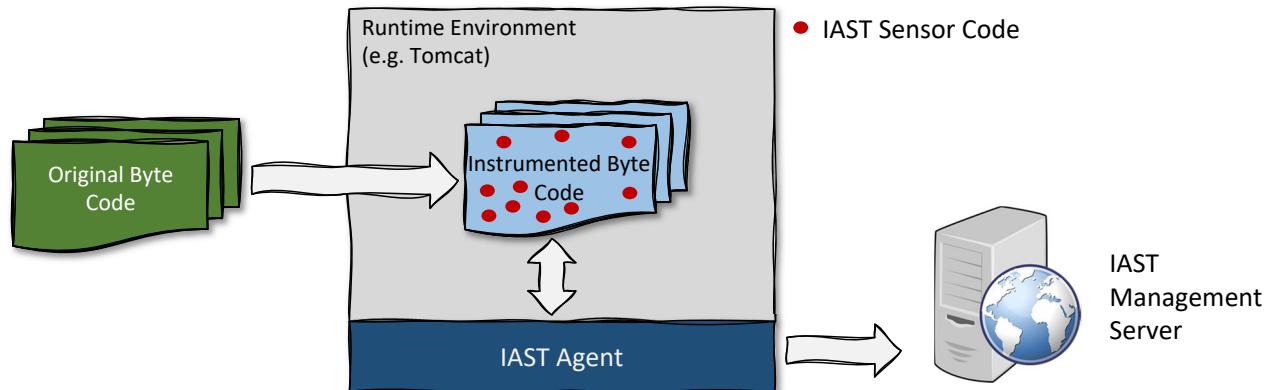


- Instrumentierung von Eintrittspunkten (Entry Points)
 - org.apache.catalina.connector.RequestFacade.getParameterValues(java.lang.String)
 - ...
- Instrumentierung von String-Klasse
 - java.lang.String
- Instrumentierung von Sanitizing-Funktionen
 - org.hibernate.Query.setParameter(int position, Object 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

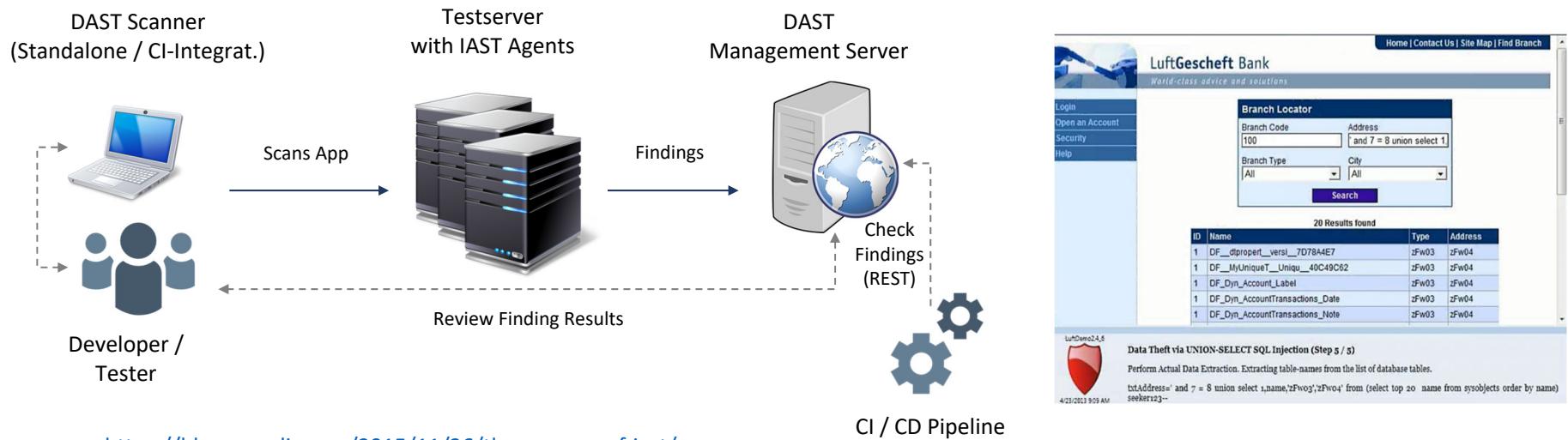
IAST = Dynamic Security Code Scanning

- 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 analysieren.
- RASP = Runtime Protection („Embedded WAF“), oft auf Basis von IAST-Technologien.



Active IAST

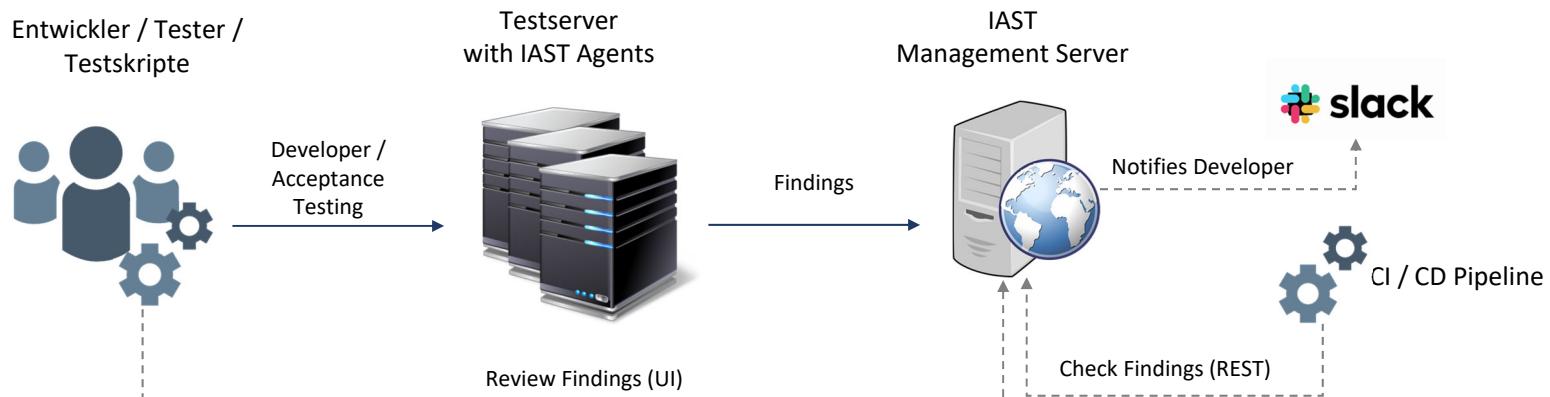
- Aka „IAST Light“
- Arbeitet als **Agent für DAST-Scanner**
- Dient primär zur Verbesserung der Qualität von DAST Scannern, löst aber nicht deren zentrale Probleme



<https://blog.secidis.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



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

Gartner.
WHY GARTNER | ANALYSTS | RESEARCH | EVENTS | CONSULTING | ABOUT |

Gartner Blog Network



Neil MacDonald
A member of the Gartner Blog Network

[« Back to GBN Home](#)

Like 0 Tweet Share

Interactive Application Security Testing

by Neil MacDonald | January 30, 2012 | 8 Comments

Dynamic Application Security Testing (DAST) solutions test applications from the "outside in" to detect security vulnerabilities. In contrast, Static Application Security Testing (SAST) solutions test applications from the "inside out" by looking at source code, byte code or binaries.

Both approaches have their pros and cons and, until recently, the market for these tools has evolved separately with different vendors and solutions. Even when a single vendor offers both DAST and SAST solutions, they have not historically been integrated.



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

A4 – XML External Entities (XXE)

A5 – Broken Access Control

A6 – Security Misconfiguration

A7 – Cross-Site Scripting (XSS)

A8 – Insecure Deserialization

A9 – Using Components with Known Vuln.

A10 – Insufficient Logging & Monitoring



DEMO

IAST-Limitationen / Beispiel

Screenshot of the Contrast Security application security tool interface showing a Cross-Site Scripting (XSS) vulnerability in WebGoat.

Application: WebGoat

Vulnerabilities tab selected.

Organize Vulnerabilities by:

- Workflow
- All Issues (8)
- Critical & High (2)
- Current Week (8)
- High Confidence (3)
- Open Status (8)
- Vulnerability Type
- Time
- Server
- URL
- Module

Details for XSS Vulnerability:

Summary: Cross-Site Scripting from "field1" Parameter on "attack" page (main.jsp, line 282)
Discovered: 11/26/2015 10:11 AM

How to Fix: Add input validation to prevent script injection.

HTTP Info: Shows the raw HTTP request and response.

Details: Shows the code flow and specific lines of code where the vulnerability was found.

Notes: Shows notes related to the vulnerability.

Discussion: Shows comments from other users.

Actions: Buttons for marking the issue as resolved or unverified, and for sharing via email or social media.

Code Snippets (highlighted with red boxes):

- Dangerous Data Received:
`string[] = facade.getParameterValues("field1")
at getRawParameter() @ ParameterParser.java:597`
- Interesting Security Event Occurred On Data:
`matcher = pattern.matcher("111")
at handleRequest() @ AbstractLesson.java:771`
- Data Flowed from Parameter to Object:
`sb = sb.append("111")
at handleRequest() @ AbstractLesson.java:771`
- Data Flowed from Object to Return Value:
`str = sb.toString()
at handleRequest() @ AbstractLesson.java:771`
- Data Flowed from Parameter to Object:
`writer.write("<input name='field1' type='TEXT' value='111'>".45)
at output() @ GenericElement.java:821`
- Data Flowed from Object to Return Value:
`str = writer.toString()
at getContent() @ Screen.java:233`
- Rule Violation Detected:
`impl.print("<form accept-charset='UNK...r><hr width='90%'></form>")
at service() @ HttpJspBase.java:70`

A red arrow points to the regular expression `^[0-9]{3}$` in the "Interesting Security Event Occurred On Data" section, highlighting a limitation in the tool's detection logic.

IAST vs. SAST

- Erfordert ausführbare 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 geeignet (kein C/C++ etc. möglich)

Fazit

- IAST ist 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