



Tool-basierte Web Security

Matthias Rohr
20. März 2014



OWASP

The Open Web Application Security Project

About Me



OWASP

The Open Web Application Security Project

- Matthias Rohr
- CISSP, CSSLP, CISM, CCSK
- Application Security seit knapp 10 Jahren
- Autor sowie Berater und Geschäftsführer bei der Secodis GmbH in Hamburg
- Schwerpunkte:
 - Secure Development Lifecycle (SDL)
 - Tool-basierte Software-Sicherheitsanalysen

secodis
APPLICATION SECURITY SERVICES



OWASP

The Open Web Application Security Project

Zunächst ein wenig FUD* ...

* = Fear, Uncertainty & Doubt

„Cyber-Angriffe“



OWASP

The Open Web Application Security Project

“The cyber threat is one of the most serious economic and national security challenges we face as a nation”

Obama, 29. Mai 2009



In einer Studie gaben 93% der größeren und 76% der kleineren Unternehmen in UK an, in 2011 von einem Cyber-Angriff betroffen gewesen zu sein

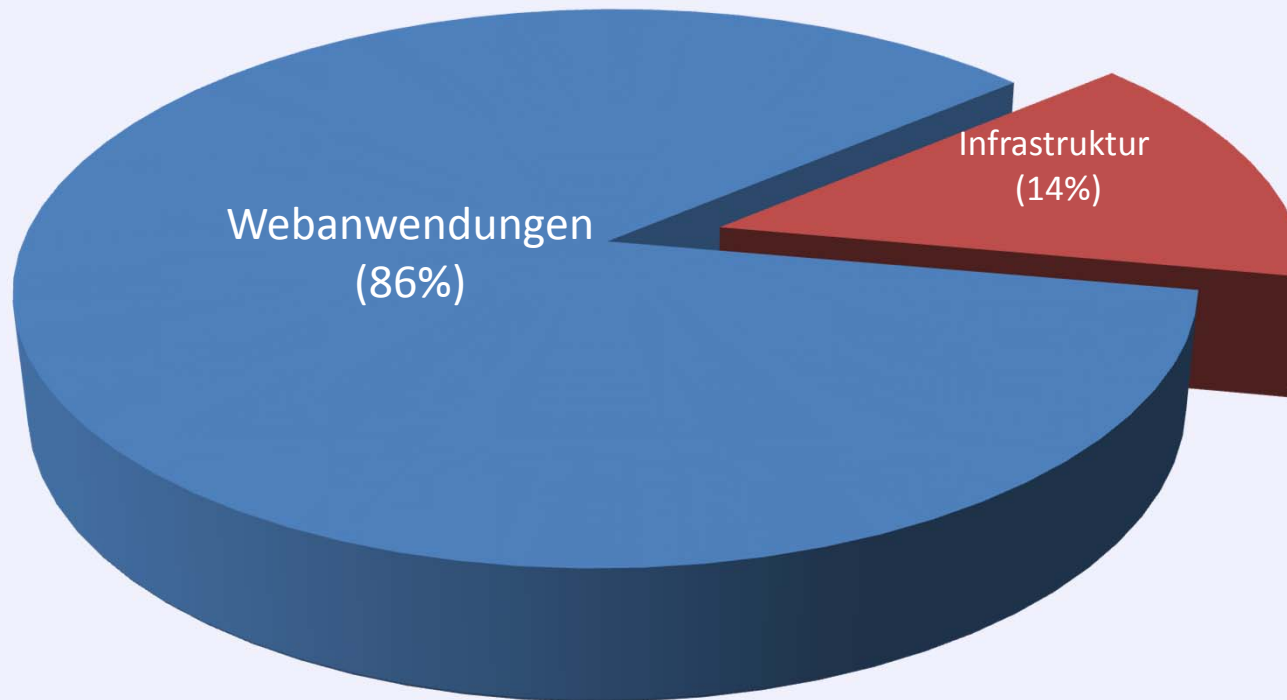
Information security breaches survey –
Technical report, April 2012, PwC

„Cyber-Angriffe“ = Angriffe auf Webanwendungen



OWASP

The Open Web Application Security Project



Quelle: UK Security Breach Investigations Report 2010



OWASP

The Open Web Application Security Project

OWASP Top 10 – 2013

A1 – Injection

A2 – Broken Authentication and Session Management

A3 – Cross-Site Scripting (XSS)

A4 – Insecure Direct Object References

A5 – Security Misconfiguration

A6 – Sensitive Data Exposure

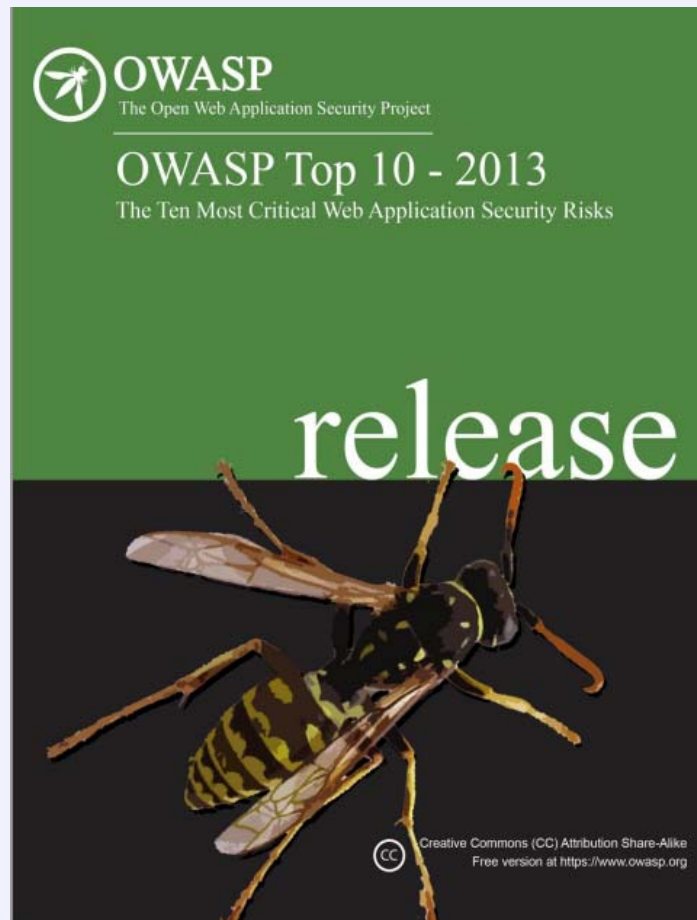
A7 – Missing Function Level Access Control

A8 – Cross-Site Request Forgery (CSRF)

A9 – Using Known Vulnerable Components

A10 – Unvalidated Redirects and Forwards

Merged with 2010-A7 into new 2013-A6



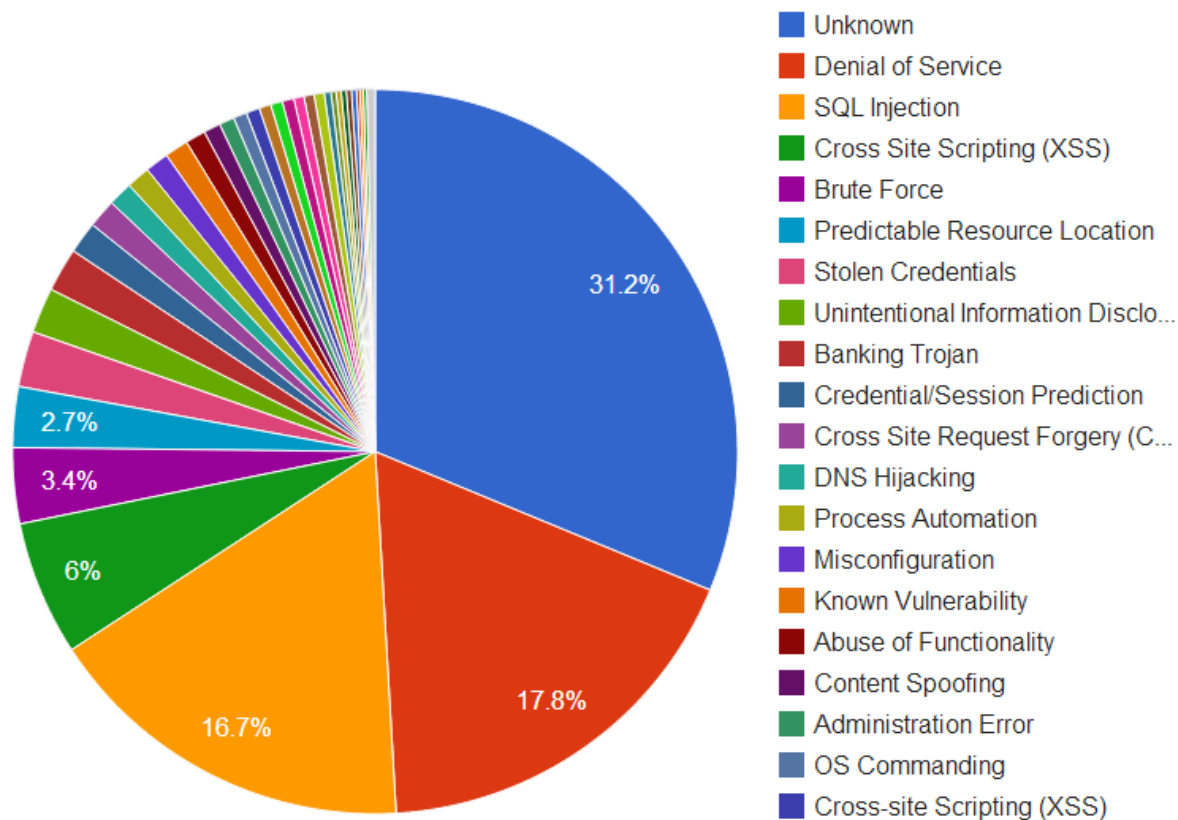
https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project

Angriffe



OWASP

The Open Web Application Security Project



1999-2011

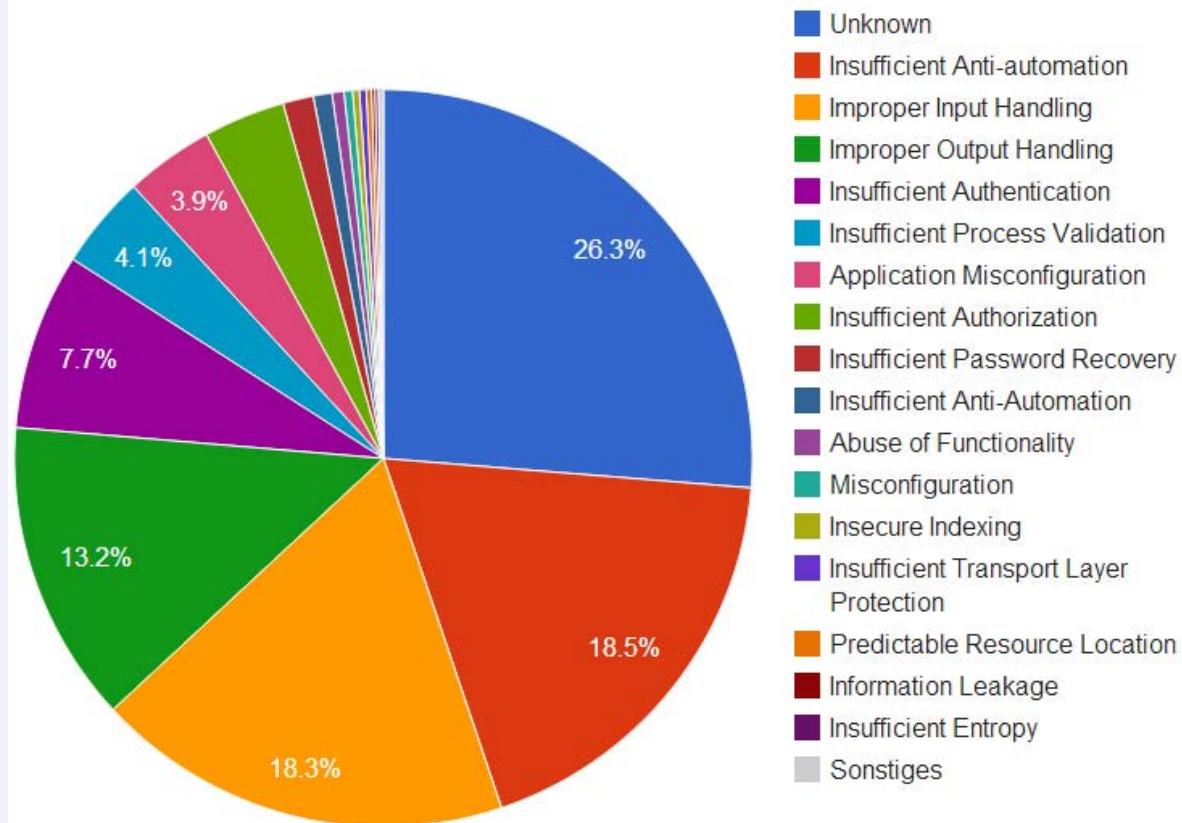
<http://projects.webappsec.org/w/page/13246995/Web-Hacking-Incident-Database>

Schwachstellen



OWASP

The Open Web Application Security Project



1999-2011

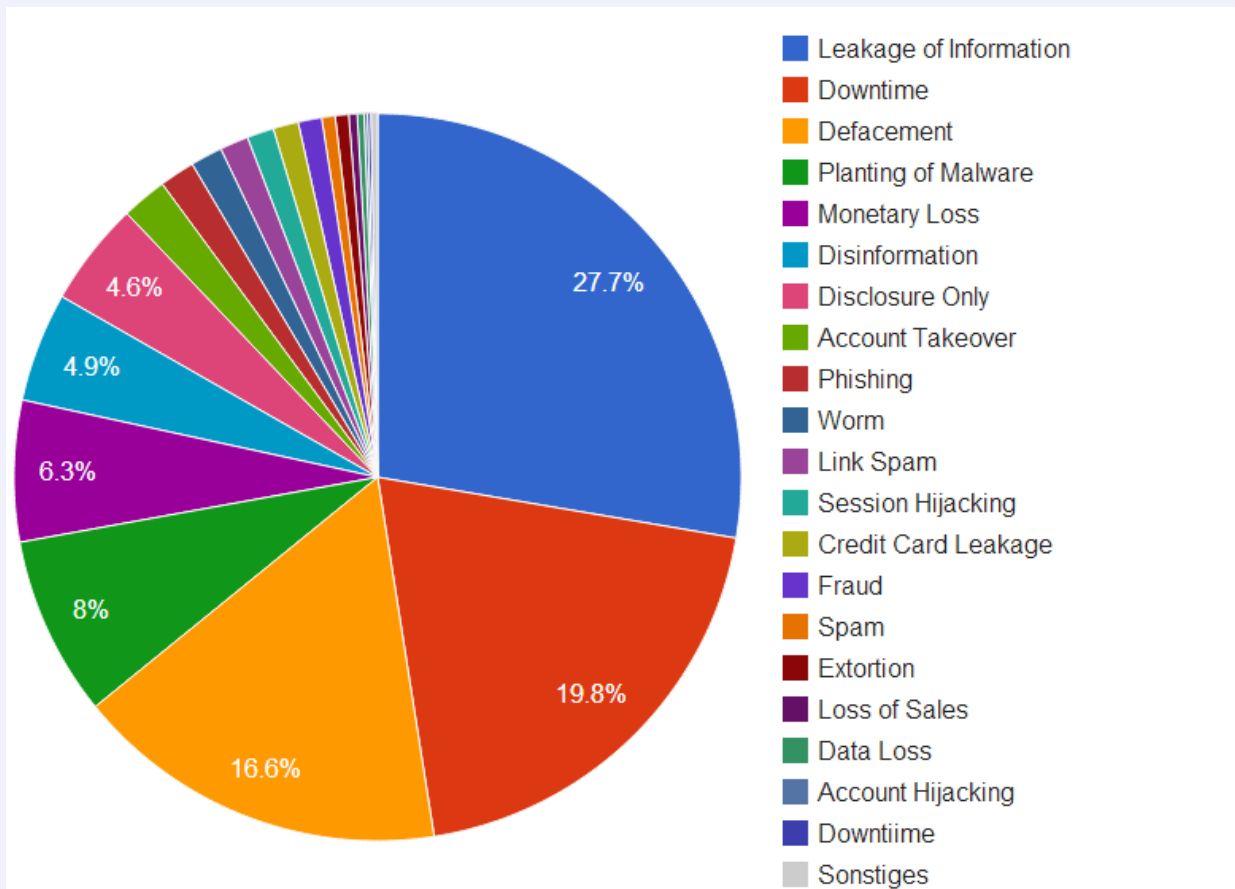
<http://projects.webappsec.org/w/page/13246995/Web-Hacking-Incident-Database>

Schäden durch Angriffe auf Webanwendungen



OWASP

The Open Web Application Security Project



1999-2011

<http://projects.webappsec.org/w/page/13246995/Web-Hacking-Incident-Database>

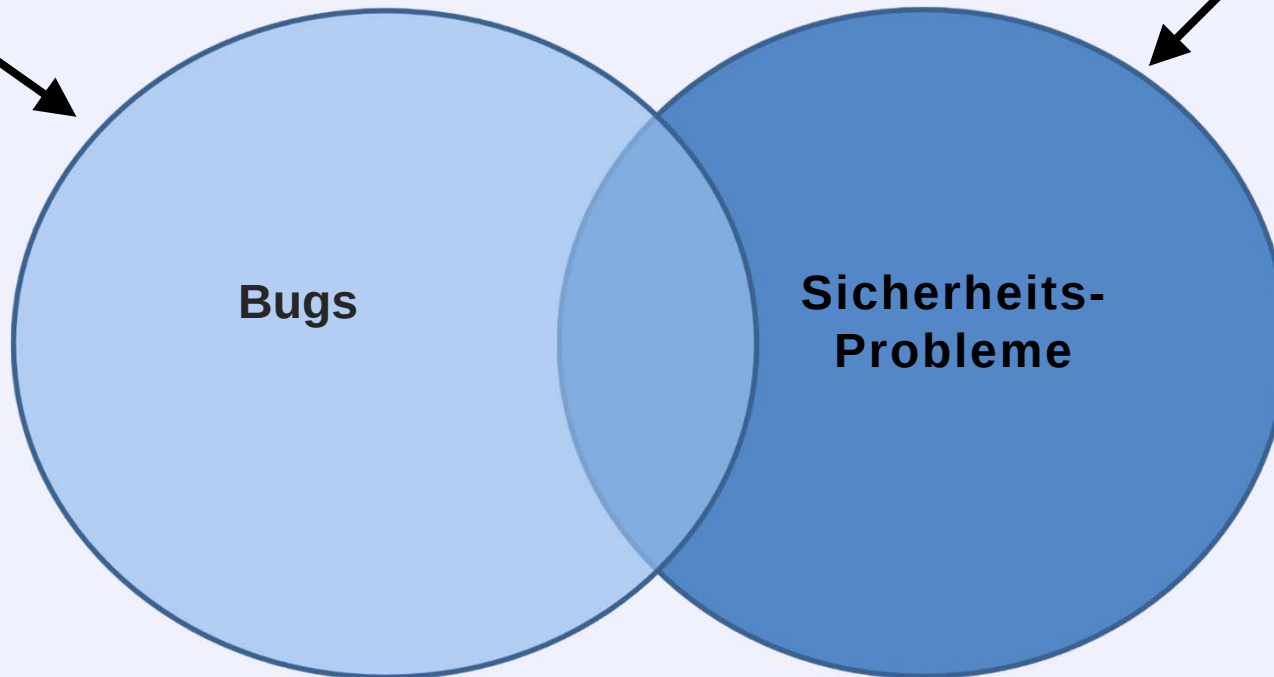
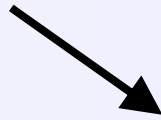
Bugs vs. Sicherheitsprobleme



OWASP

The Open Web Application Security Project

Anforderungen



Implementierung
(handwerkliche Fehler)



Bugs

**Sicherheits-
Probleme**

Quelle: Joe Jarzombek, Software-Assurance: Enabling Enterprise Resilience through Security Automation and Software Supply Chaining Risk Management



OWASP

The Open Web Application Security Project

Tools ...

Code vs. Application Layer



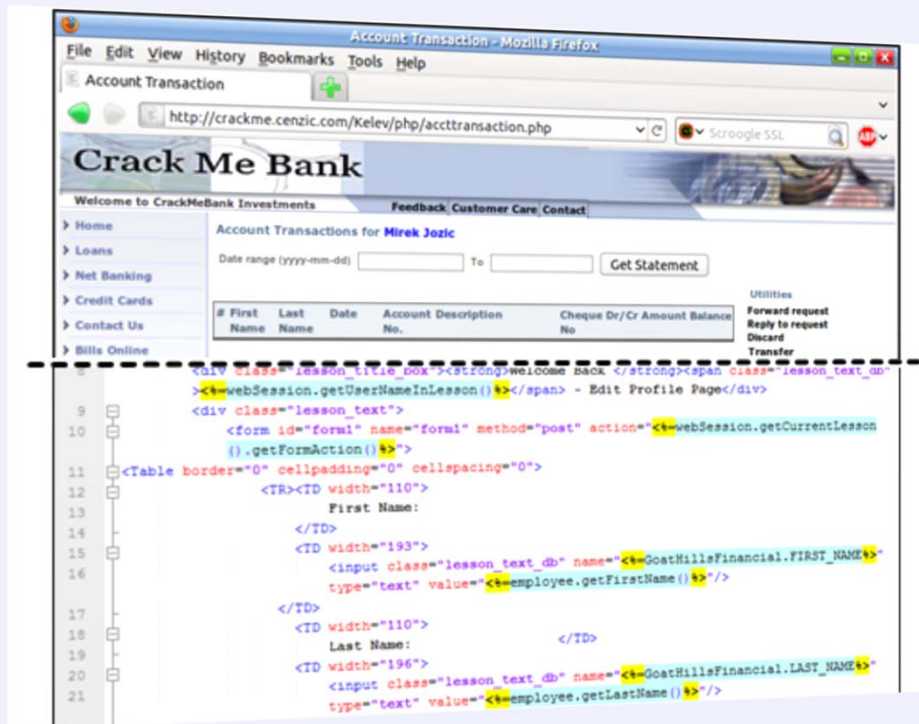
OWASP

The Open Web Application Security Project

Schwachstellen in der laufenden Anwendung

Manuell: Pentest

Autom: Dynamic App. Security Testing (DAST)



Session Mgmt /
CSRF,
Unsichere
Einbindung,
Logikfehler,
Anti-
Automatisierung,
...

Race Conditions
Buffer Overflows
Backdoors,
Unsichere APIs,
Anbindung
Backend,
...

XSS,
SQL Injection,
Datenval. allg.
Authentifizierung,
Zugriffsschutz,
Kryptographie,
Information
Leakage,
Fehlerbehandlung,
Konfiguration,
...

Schwachstellen auf Codeebene

Manuell: Code Review

Autom: Static App. Security Testing (SAST)

Application Security Testing Tools



OWASP

The Open Web Application Security Project

- Dynamic App. Security Testing (DAST)
- Static App. Security Testing (SAST)
- Hybride Ansätze



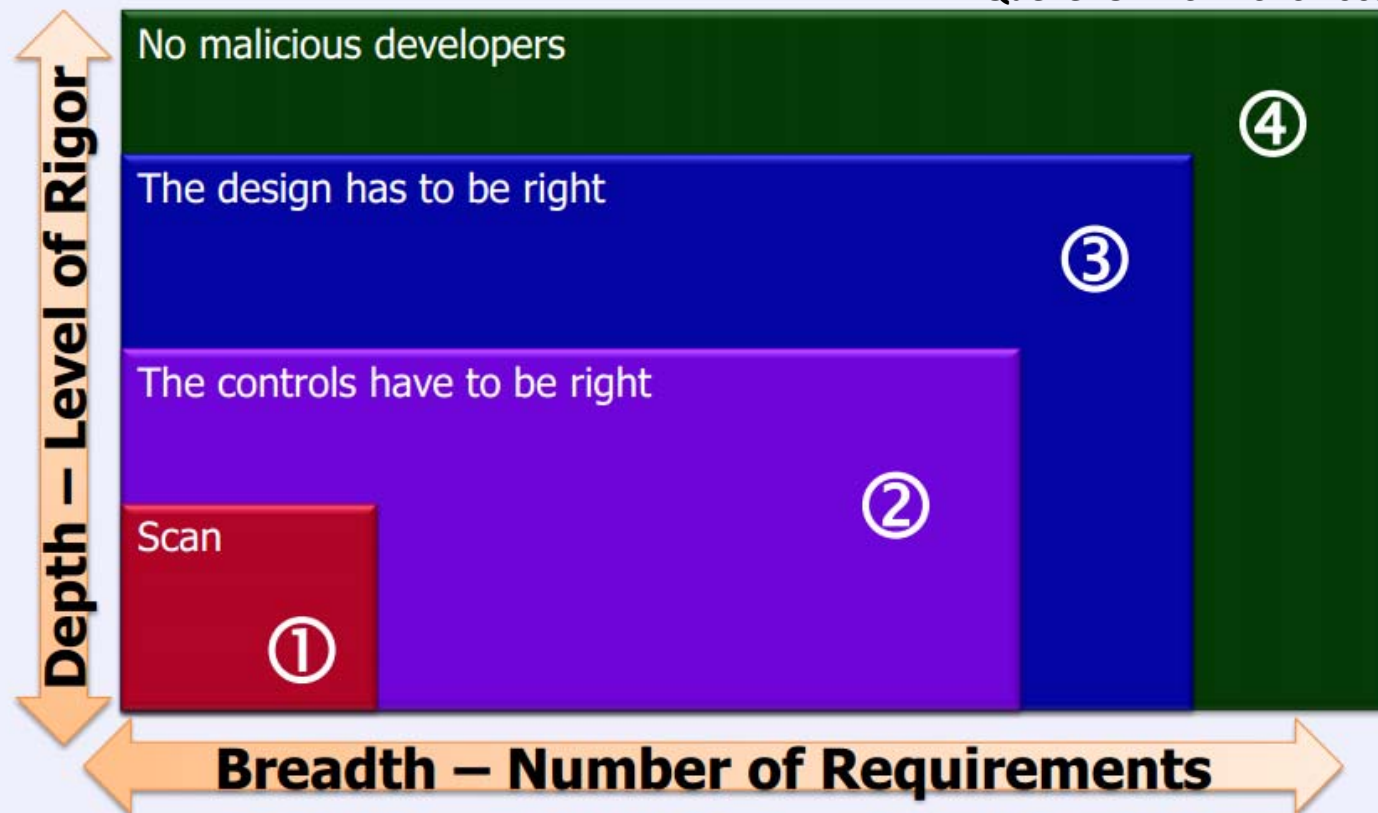
Tools vs. Manuelle Verifikation



OWASP

The Open Web Application Security Project

Quelle: OWASP ASVS 2009



Tools liefern keine Schwachstellen, nur Findings. Eine manuelle Verifikation und Bewertung ist immer erforderlich!

Häufige Vorbehalte



OWASP

The Open Web Application Security Project

- Zu teuer
- Viele False Positives (Schwachstellen die keine sind)
- Viele False Negatives (Schwachstellen die nicht gefunden werden)*
- Unzureichende Abdeckung
- Niemand der sie bedienen kann
- Gefährdet die Stabilität meiner Anwendung

* Vadim Okun. Aurelien Delaitre, Paul E. Black , The Second Static Analysis Tool Exposition (SATE), Special Publication 500-287, Juni 2010, http://samate.nist.gov/docs/NIST_Special_Publication_500-287.pdf



OWASP

The Open Web Application Security Project

10 Gründe,
warum Tools
trotzdem **wichtig** sind ...

Warum Tools? (1)



OWASP

The Open Web Application Security Project

1. Anwendungen werden immer größer
 - Große Webanwendungen > 100 KLOC
 - Ein Security Reviewer schafft im Schnitt 1 KLOC pro Tag
2. Anwendungen ändern sich laufend
 - Agile Entwicklung
 - Schnelle Änderungen durch Änderungen im Deployment
 - Laut Whitehat, wurden in 2012 im Schnitt 56 kritische Schwachstellen in Webanwendung eingebracht.*
 - Änderungen von Anwendungen und deren Einbindung im Betrieb (Web 2.0)
3. Security-Experten sind teuer und rar



* <http://de.slideshare.net/duncant75/whitehat-security-website-security-statistics-report-may-2013>

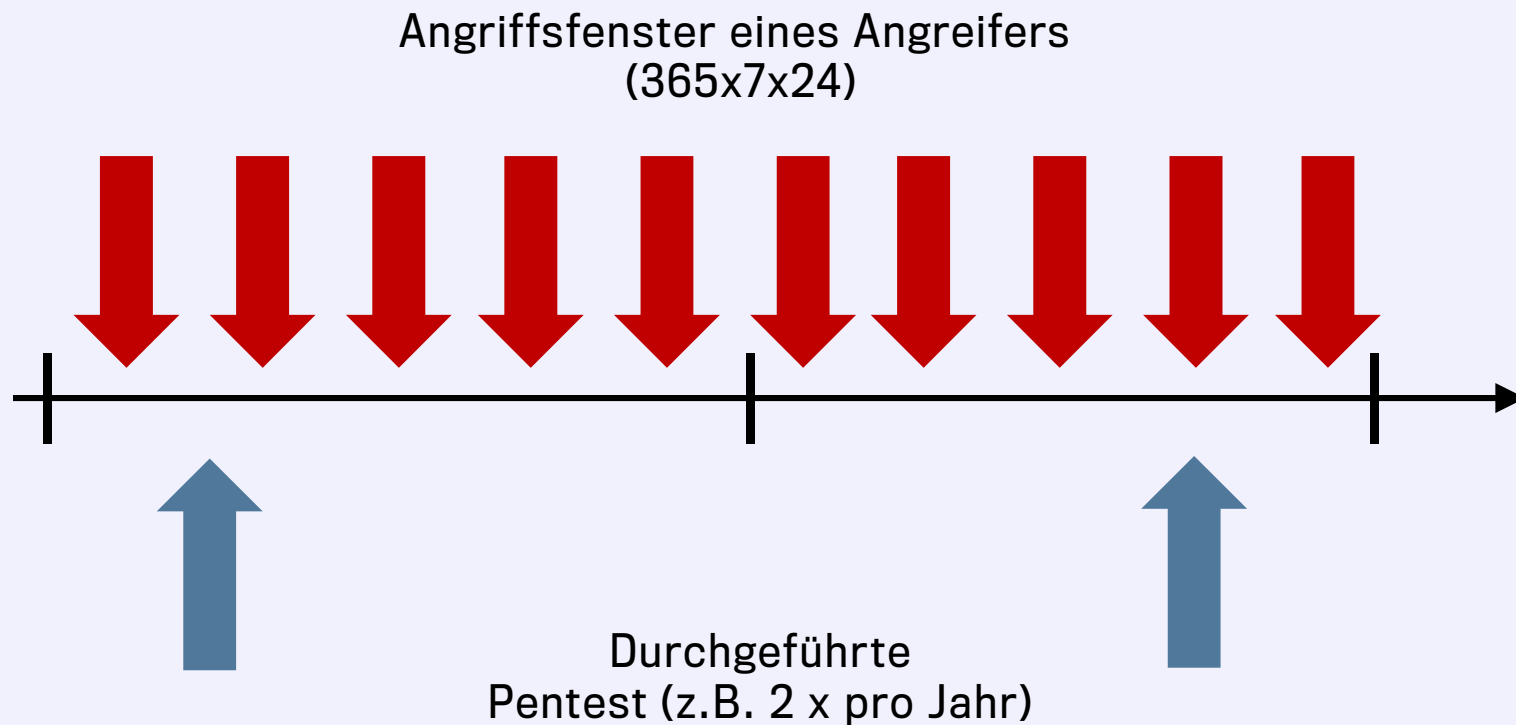
Warum Tools? (2)



OWASP

The Open Web Application Security Project

4. Tools arbeiten schnell und können kontinuierlich ausgeführt werden



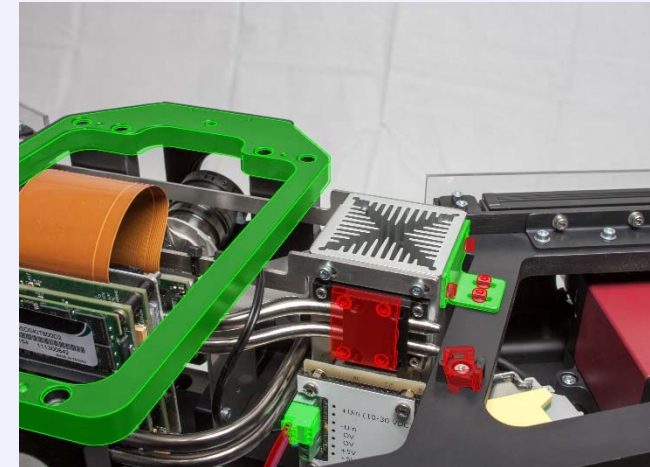
Warum Tools? (3)



OWASP

The Open Web Application Security Project

5. Tools können manuelle Analysen deutlich wirkungsvoller (bzw. überhaupt erst durchführbar) machen.
6. Mittels Tools lässt sich die Einhaltung spezifischer Policys / Vorgaben laufend prüfen.
7. Tools ermöglichen nachvollziehbare, objektive und standardisierte Analysen.



www.iff.fraunhofer.de

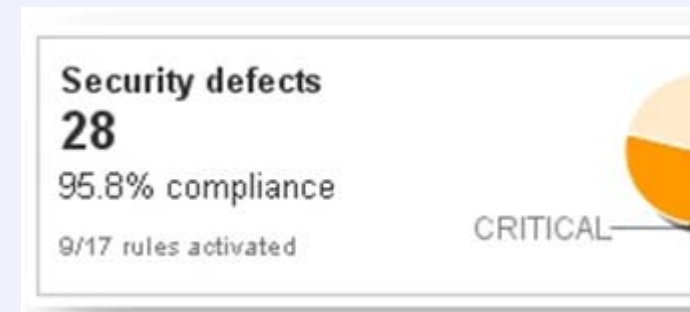
Warum Tools? (3)



OWASP

The Open Web Application Security Project

- 8. Tools ermöglichen Nachhaltigkeit und kontinuierliche Verbesserung.
- 9. Tools ermöglichen die Ermittlung von Sicherheitskennzahlen (Metriken, KPIs).
- 10. Tools lassen sich in andere Tools integrieren (=> QA)



**Limitationen von Tools müssen verstanden,
aber auch Chancen durch deren Einsatz
genutzt werden!**



OWASP

The Open Web Application Security Project

Dynamic Application Security Testing (DAST)

= Tools, die eine laufende Anwendung auf potentielle Sicherheitsprobleme hin untersuchen

Sicherheitsprobleme, die DAST-Tools finden können...



OWASP

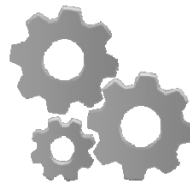
The Open Web Application Security Project

Implementierungs-Fehler



SQL Injection*
XSS*
Known Vuln.
etc.

Konfigurationsfehler



Error Handling*
TLS-Config
etc.

Ungültige TLS-Zerts



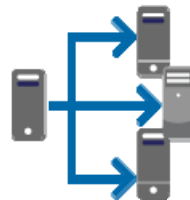
Abgelaufene
oder invalide
TLS/SSL-
Zertifikate

Sicherheitsanforderungen



Loginversuche
Passwortlänge
Access Controls
etc.

Deploymentfehler



Testzugänge
SVN-
Dateien
etc.

Malware



Schadcode auf
den eigenen
oder in
dynamisch
eingebundenen
Seiten
(Werbebanner)

* Low Hanging Fruits

Pentesting Tools - Burp



OWASP

The Open Web Application Security Project

Burp Suite Free Edition v1.5

Target | **Proxy** | Spider | Scanner

Intruder | Repeater | Sequencer | Decoder | Comparer | Options | Alerts

1 x 2 x 3 x 4 x ...

Target | Positions | **Payloads** | Options

Payload Positions

Configure the positions where payloads will be inserted into the base request. The attack type determines the way in which payloads are assigned to payload positions – see help for full details.

Attack type: **Sniper**

```
GET /bookDetails.html?id=$735$&userId=$341$
HTTP/1.1
Host: www.example.com
User-Agent: Mozilla/5.0 (Macintosh; Intel
Mac OS X 10.8; rv:25.0) Gecko/20100101
Firefox/25.0
Cookie:
JSESSIONID=55E8F00744281581FEDE6C8DCDB81BE2;
env=$prod2$
```

3 payload positions | Length: 409

Burp Suite Free Edition v1.5

Repeater | Sequencer | Decoder | Comparer | Options | Alerts

Target | **Proxy** | Spider | Scanner | Intruder

1 x 2 x 3 x 4 x ...

Target | Positions | **Payloads** | Options

Payload set: **1** | Payload count: 512

Payload type: **Simple list** | Request count: 1536

Payload Options [Simple list]

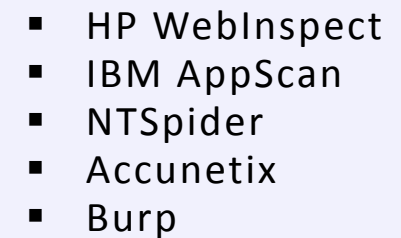
This payload type lets you configure a simple list of strings that are used as payloads.

Paste | Load ... | Remove | Clear

```
!@#%&'()*~+-=,.;:[]{}|_`" 'or "a"="a
"or "x"="x
"or 0=0 #
"or 0=0 --
"or 1=1 or ""="
```

Außerdem: OWASP ZAP, Fiddler, (OWASP WebScarab), diverse Browser-Plugins

OWASP
The Open Web Application Security Project

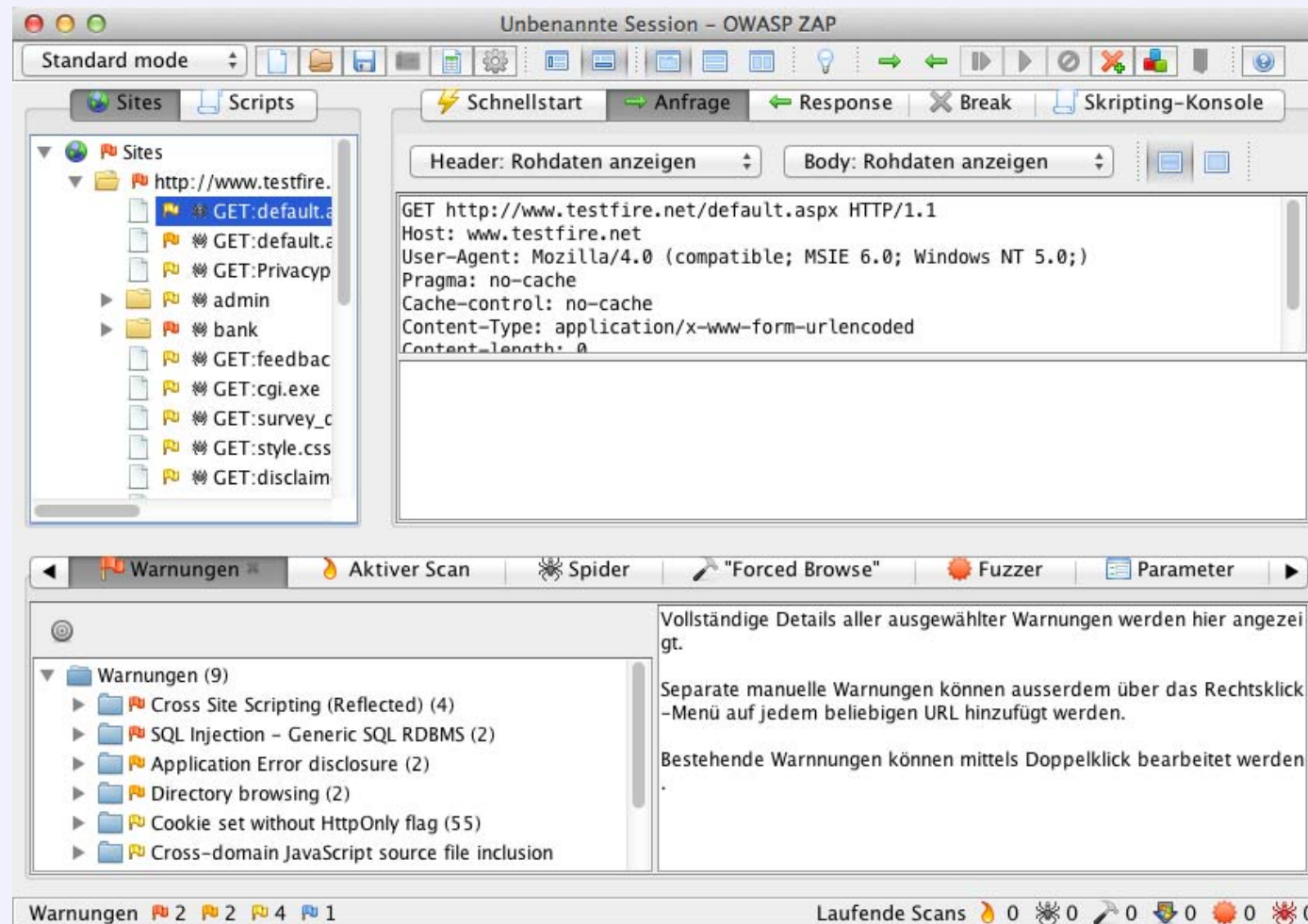
[illegible]

Web Security Scanner (kostenfrei)



OWASP

The Open Web Application Security Project



- OWASP ZAP
- w3af
- skipfish
- Nikto / Wikto
- SSLScan / SSL Labs

Mittels ThreadFix lassen sich aus Scanner-Ergebnissen Virtual-Patching-Regeln für ModSecurity generieren.

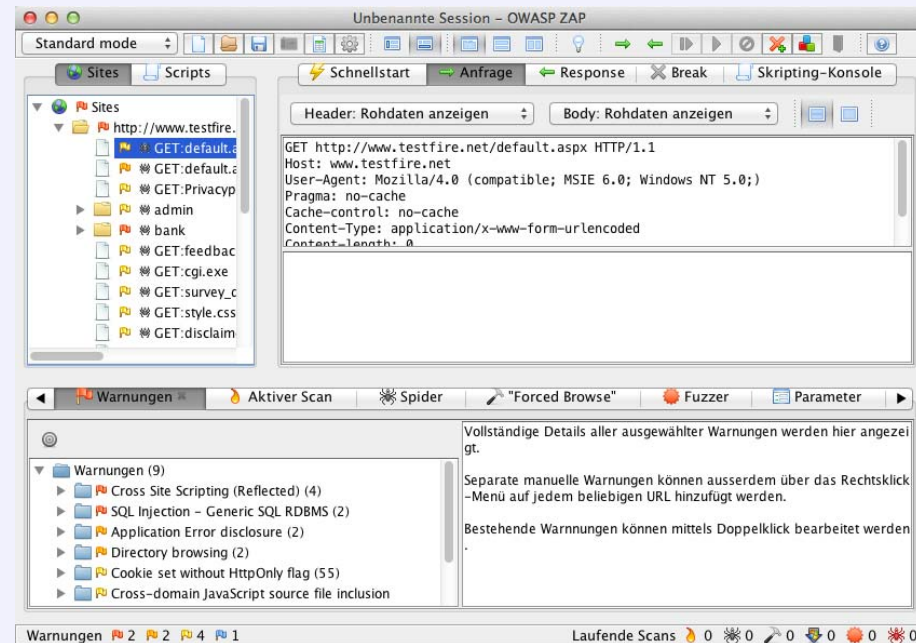


OWASP

The Open Web Application Security Project

OWASP ZAP

- Aktive und Passive Scans
- Sehr aktive Community (laufende Weiterentwicklung)
- UI und Daemon Mode
- Offenes Design:
 - API
 - Eigene Add-Ons
 - Eigene Skripte
 - Programmaufrufe



Vorsicht vor dem Einsatz auf produktiven Systemen!

OWASP
The Open Web Application Security Project

Weiterhin: Whitehat Sentinal, NTOSpider On-Demand

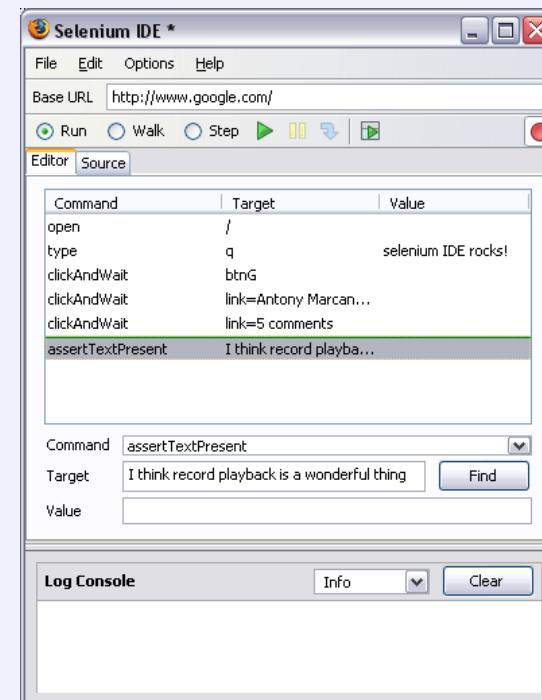
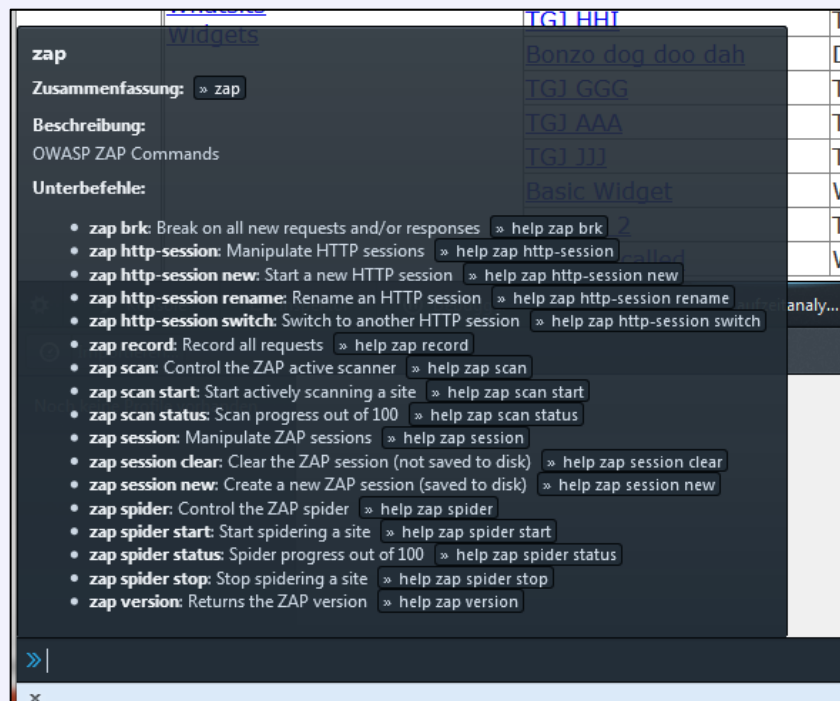
Comming Next ... Bessere Tool Integration



OWASP

The Open Web Application Security Project

- SAST – DAST - Integration
- QA-Tool-Integration (z.B. Selenium, Unit Tests, QuickTest Pro)
- Browser-Integration (Plug-n-Hack, FF Firebug, Vendor Plugins)





OWASP

The Open Web Application Security Project

Static Application Security Testing (SAST)

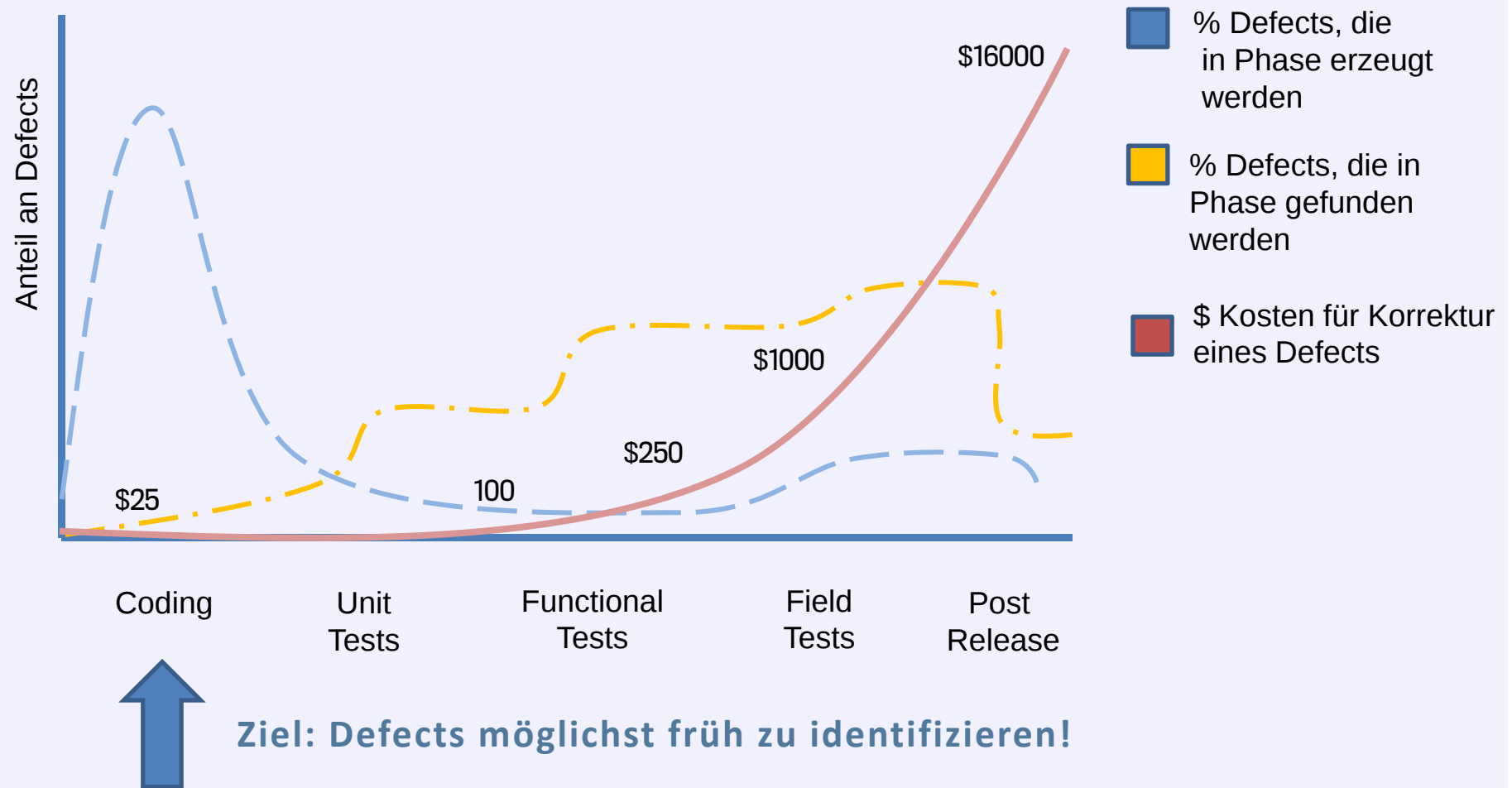
= Tools, die entweder den Sourcecode, Binärcode oder Bytecode auf potentielle Sicherheitsmängel hin untersuchen

Sicherheit im SDLC



OWASP

The Open Web Application Security Project



Caspers Jones, Applied Software Measurement: Global Analysis of Productivity and Quality, 1996

Sicherheitsprobleme, die SAST-Tools finden können...



OWASP

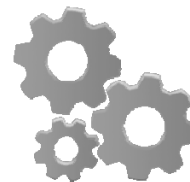
The Open Web Application Security Project

Implementierungs-Fehler



- SQL Injection / XSS
- Buffer Overflows / Race Conditions
- Unsichere APIs / Aufrufe
- etc.

Konfigurationsfehler



- Fehlerbehandlung
- Validierung
- Hartkodierte Passw.
- etc.

Security Anforderungen



- Secure Coding Guidelines / Policies
- Architektur-Constraints
- Etc.

Sensible Änderungen



Unautorisierte Änderung an sensiblen Dateien (z.B. Bibliotheken)

SAST-Analyseverfahren



OWASP

The Open Web Application Security Project

Lexikalische Analyse

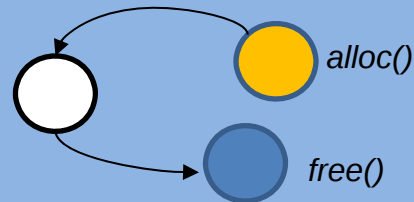
grep -i -r "exec" *

Identifizierung verdächtiger Zeichenketten im Code (Reguläre Ausdrücke)

Beispiele

- Hartkod. Passwörter
- Entwicklerkommentare
- Kommentierter Code

Kontrollflussanalyse

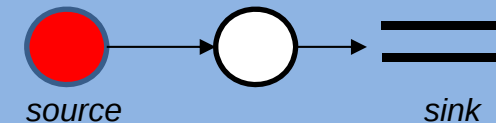


Identifiziert fehlerhaft freigegebenen Speicher, unsichere Sequenz von Funktionsaufrufen, etc.

Beispiele

- Buffer Overflows
- Race Conditions

Datenflussanalyse



Dient dem Auffinden von Fehlern in der Datenvalidierung

Beispiele

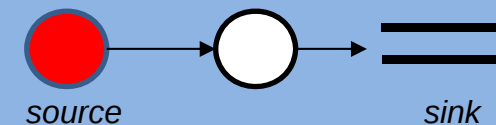
- Cross-site Scripting
- SQL Injection

Weiterhin: Semantische Analyse, Strukturanalyse, Konfigurationsanalysen



- Die Datenflussanalyse ist das wichtigste Analyse-Verfahren eines SAST-Tools!
- Die meisten kostenfreien Tools (PMD, Findbugs, etc.) bieten keine (bzw. nur eine eingeschränkte) Datenflussanalyse.

Datenflussanalyse



Dient dem Auffinden von Fehlern in der Datenvalidierung

Beispiele

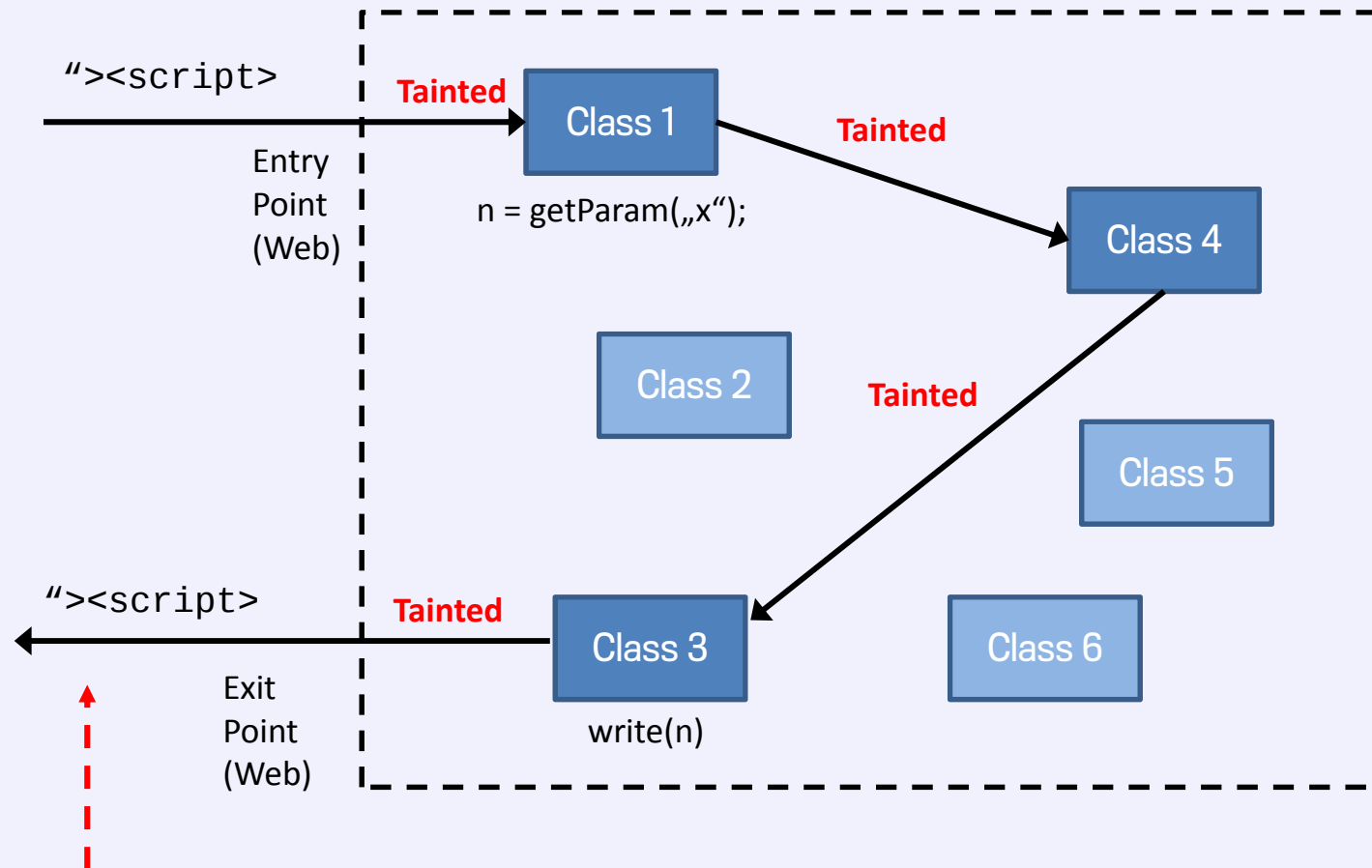
- Cross-site Scripting
- SQL Injection

Datenflussanalysen (aka Taint Analyse)



OWASP

The Open Web Application Security Project



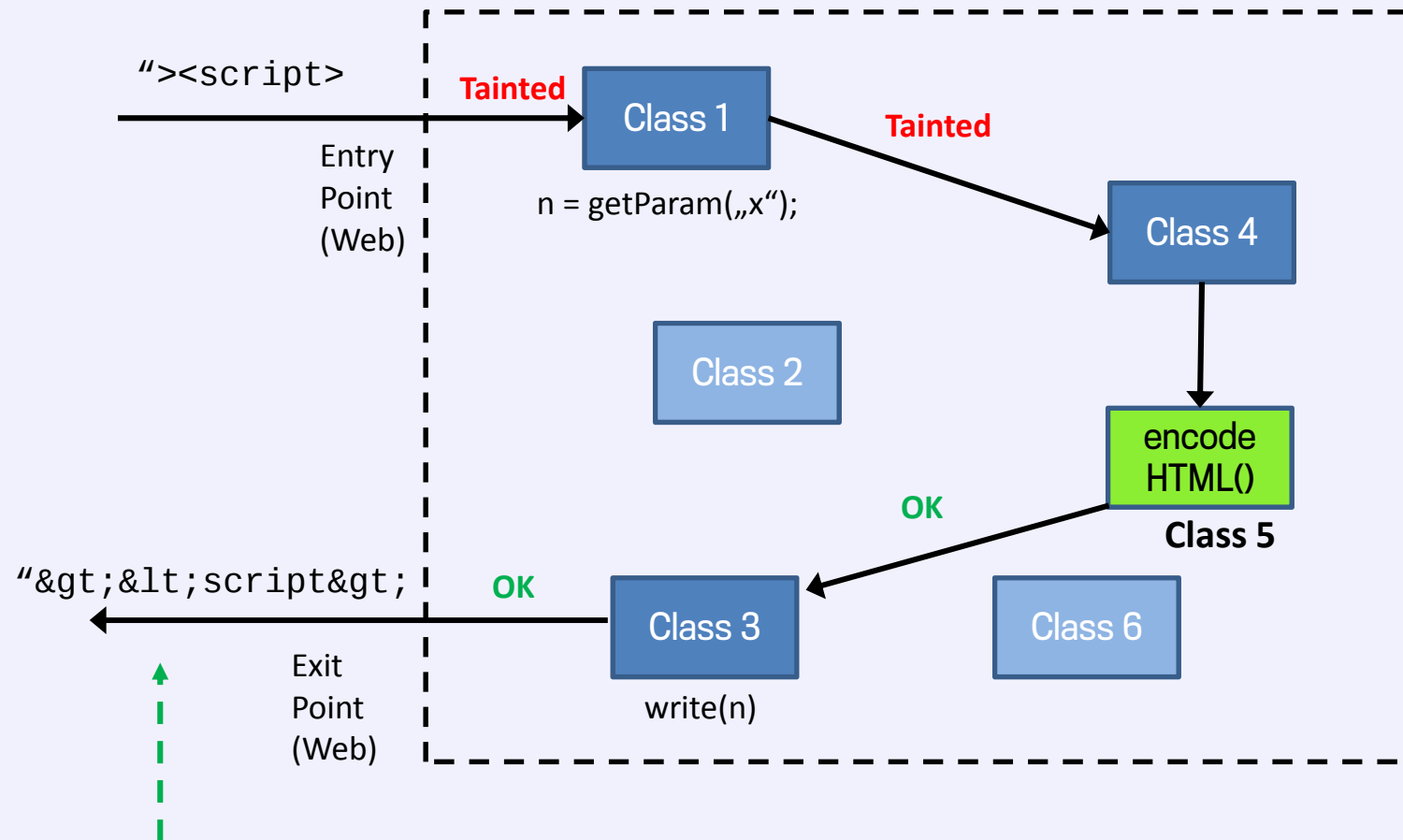
Cross-site Scripting!

Datenflussanalysen (aka Taint Analyse)



OWASP

The Open Web Application Security Project



kein Cross-site Scripting!

HP Fortify - Auditworkbench



OWASP

The Open Web Application Security Project

WebGoat5.0 - JavaSource/org/owasp/webgoat/lessons/WeakSessionID.java - Audit Workbench

Summary | Audit Guide | Scan | Reports

AUDIT WORKBENCH FORTIFY

Filter Set: Security Auditor View | My Issues

463 Critical (463)

Group By: Category

- Command Injection - [0 / 3]
- Cross-Site Scripting: Persistent - [0 / 38]
- Cross-Site Scripting: Reflected - [0 / 260]
 - AbstractLesson.java:688 (Shared Sink) - [0 / 2]
 - AbstractLesson.java:872 (Shared Sink) - [0 / 2]
 - AbstractLesson.java:920 (Cross-Site Scripting: Reflected) - [0 / 2]
 - AbstractLesson.java:1086 (Shared Sink) - [0 / 2]
 - BackDoors.java:235 (Cross-Site Scripting: Reflected) - [0 / 2]
 - BasicAuthentication.java:143 (Cross-Site Scripting: Reflected) - [0 / 2]
 - BasicAuthentication.java:145 (Cross-Site Scripting: Reflected) - [0 / 2]
 - BlindSqlInjection.java:83 (Cross-Site Scripting: Reflected) - [0 / 2]
 - CrossSiteScripting.jsp:20 (Shared Sink) - [0 / 4]
 - EditProfile.jsp:16 (Shared Sink) - [0 / 2]
 - EditProfile.jsp:16 (Shared Sink) - [0 / 2]
 - EditProfile.iso:16 (Shared Sink) - [0 / 2]

Advanced...

Analysis Evidence

- ParameterParser.java:704 - clean(0 : return)
- ParameterParser.java:704 - Assignment to value
- ParameterParser.java:712 - Return value
- ParameterParser.java:729 - getStringParameter(return)
- ParameterParser.java:729 - Return
- ReflectedXSS.java:123 - getStringParameter(return)
- ReflectedXSS.java:123 - Input(2)

Rule ID: DBD8BFC6-DE26-4FC5-8347-D48032B2BF5D
Taint Flags: WEB, XSS
Direct Function Call: org.apache.ecs.html.Input.Input()

```
119 .addElement("Studio RTA - Laptop/Reading Cart with  
120 tr.addElement(new TD().addElement("69.99").setAlign("right"  
121 tr.addElement(new TD().addElement(  
122     new Input(Input.TEXT, "QTY1", s.getParser()  
123     .getStringParameter("QTY1", "1"))  
124     .setAlign("right"));  
125 quantity = s.getParser().getFloatParameter("QTY1", 1.0f);  
126 total = quantity * 69.99f;  
127 runningTotal += total;  
128 tr.addElement(new TD().addElement("$" + total));  
129 t.addElement(tr);  
130 tr = new TR();  
131 tr.addElement(new TD()  
132     .addElement("Dyrex - Traditional Notebook Case");  
133 tr.addElement(new TD().addElement("27.99").setAlign("right"
```

Summary | Details | Recommendations | History | Diagram | Correlated Issues | Screenshots | Security

ReflectedXSS.createContent

ParameterParser.getStringParameter

getStringParameter(return) 123

getStringParameter(return)

Return

Input(2) 123 sink

Functions

Show: All

Group by: package

Include external API Legend...

- Top-level functions
- /
- /lessons/ConfManagement
- /lessons/CrossSiteScripting
- /lessons/General
- /lessons/RoleBasedAccessControl
- /lessons/SQLInjection
- http://java.sun.com/JSP/Page
- java.io
- java.lang
- java.net
- java.nio
- java.nio.charset
- java.security
- java.sql
- java.text
- java.util
- java.util.regex
- javax.crypto
- javax.crypto.spec
- javax.servlet
- javax.servlet.http
- javax.servlet.jsp
- javax.xml.namespace
- org.apache.axis
- org.apache.axis.client
- org.apache.catalina
- org.apache.catalina.users
- org.apache.ecs
- org.apache.ecs.html

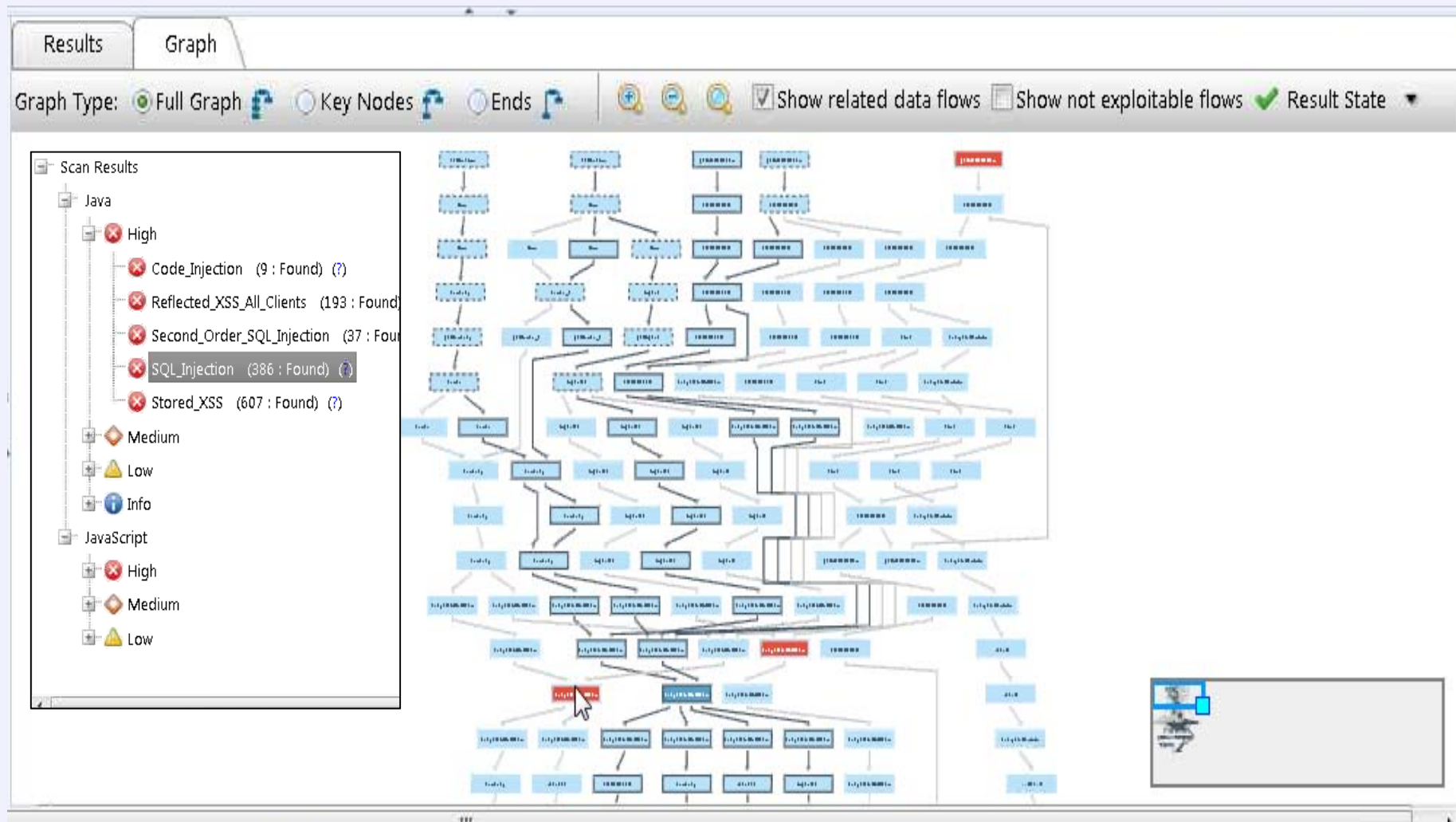
Search:

Checkmarx - Datenflussanalysen



OWASP

The Open Web Application Security Project

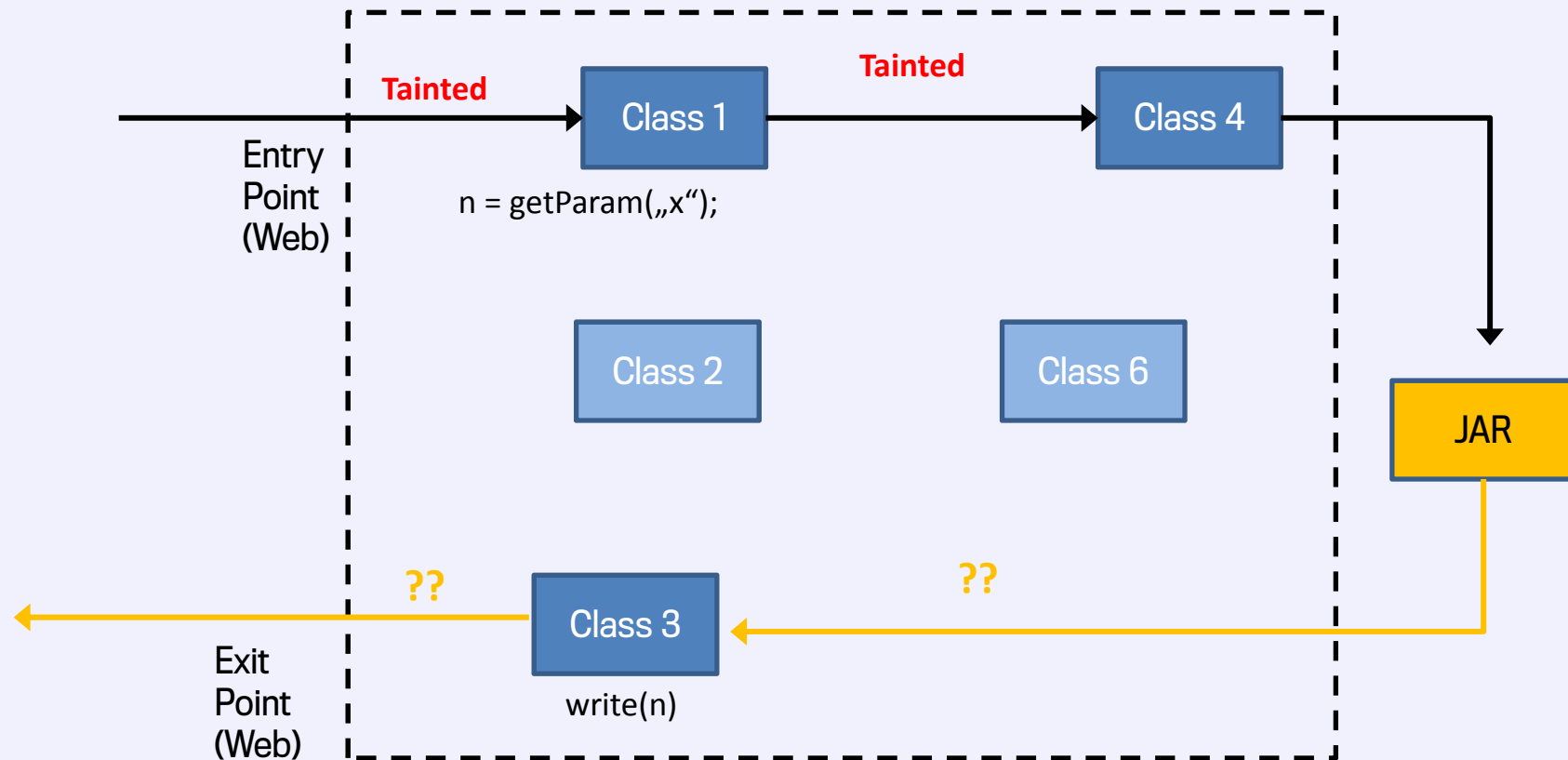


Problem von Source Code Analysen



OWASP

The Open Web Application Security Project



Weitere Probleme: Dependency Injection, komplexe Validierungsfunktionen (z.B. via Regulären Ausdrücken), ... Ergebnis: viele False Positives

Bytecodeanalysen mit Veracode



OWASP

The Open Web Application Security Project

Application: A Goat on the Web

VERACODE

Scan: Static: 11 Nov 2013 Static | Dynamic: 2 Oct 2013 Dynamic

Request Readout

Show: ☐ Fix First Analyzer ☒ Source Code Viewer ☐ None

Source Code View (HTML5 Version): Exec.java

Lines 1-544

Load Different File

Goto Line

Source History...

```
282   ByteArrayOutputStream errors = new ByteArrayOutputStream();
283   ExecResults results = new ExecResults(command, input, successCode,
284       timeout);
285   BitSet interrupted = new BitSet(1);
286   boolean lazyQuit = false;
287   ThreadWatcher watcher;
288
289   try
290   {
291       // start the command
292       child = Runtime.getRuntime().exec(command);
293
294       // get the streams in and out of the command
295       InputStream processIn = child.getInputStream();
296       InputStream processError = child.getErrorStream();
```

Flaws: 268 of 268

Rows/Page: 50

<< 1 2 3 4 5 6 >>

All 268 Flaws 0 Flaws Search: Id = GO CLEAR Take Selected Action GO

Id	Sev	Exp	CWE ID & Name	Module	Source	Count	Status	Mitigation
41	5	V.Likely	78 Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')	WebGoat-5.0-with-jsp.war	Exec.java: 103	1	Open	none
76	5	V.Likely	78 Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')	WebGoat-5.0-with-jsp.war	Exec.java: 292	1	Open	none
24	4	V.Likely	89 Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	WebGoat-5.0-with-jsp.war	Login.java: 149	1	Open	none
25	4	V.Likely	89 Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	WebGoat-5.0-with-jsp.war	UpdateProfile.java: 176	1	Open	none
37	4	V.Likely	89 Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	WebGoat-5.0-with-jsp.war	ViewProfile.java: 178	1	Open	none
73	4	V.Likely	89 Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	WebGoat-5.0-with-jsp.war	SqlNumericInjection.java: 130	1	Open	none

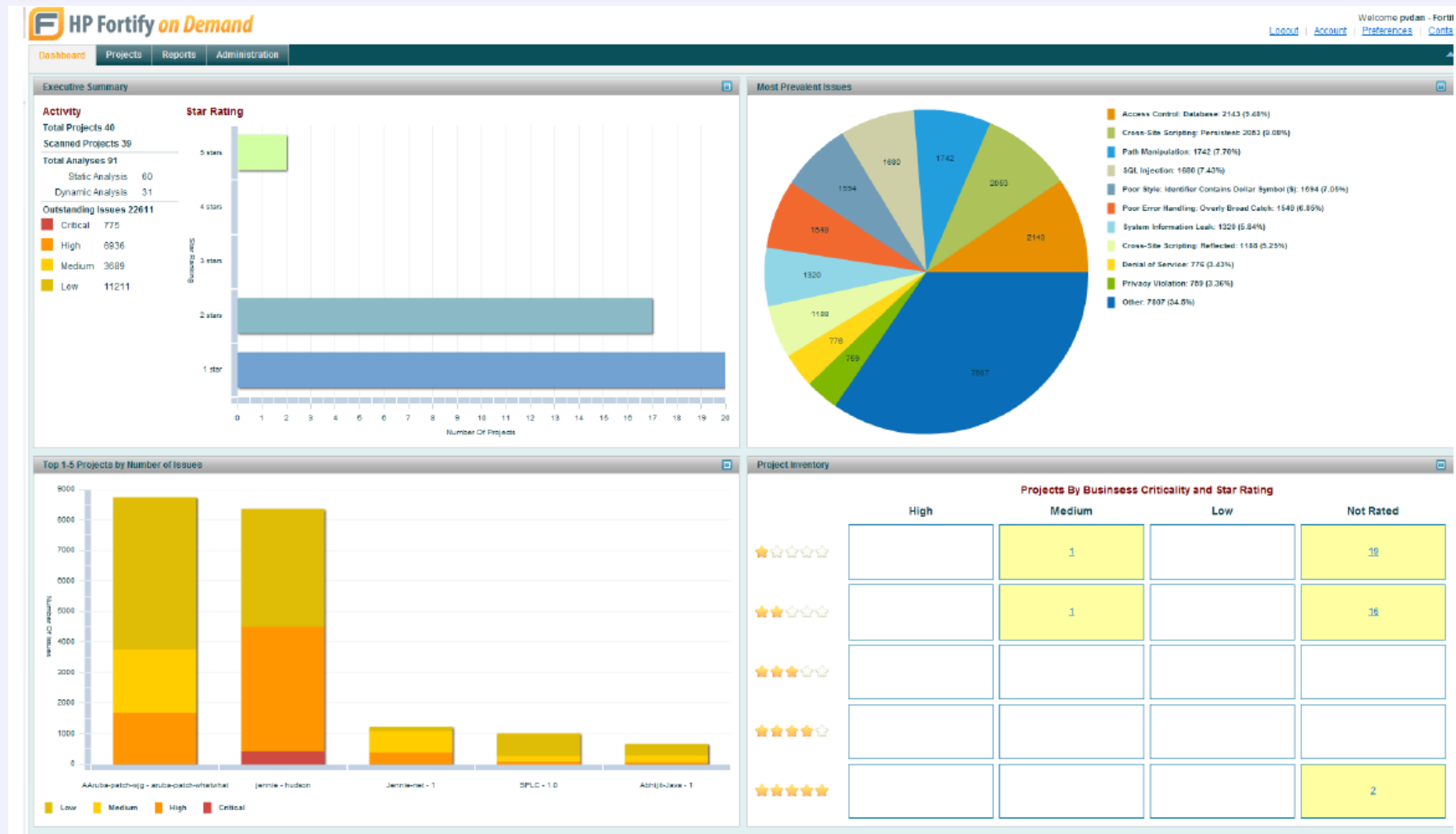
Flaws Call Stack

Fortify on Demand (FOD)



OWASP

The Open Web Application Security Project



SAST-Deployment: Tools der 1. Generation



OWASP

The Open Web Application Security Project



Security Analyst



Code



Entwickler



Ergebnis-
Bericht



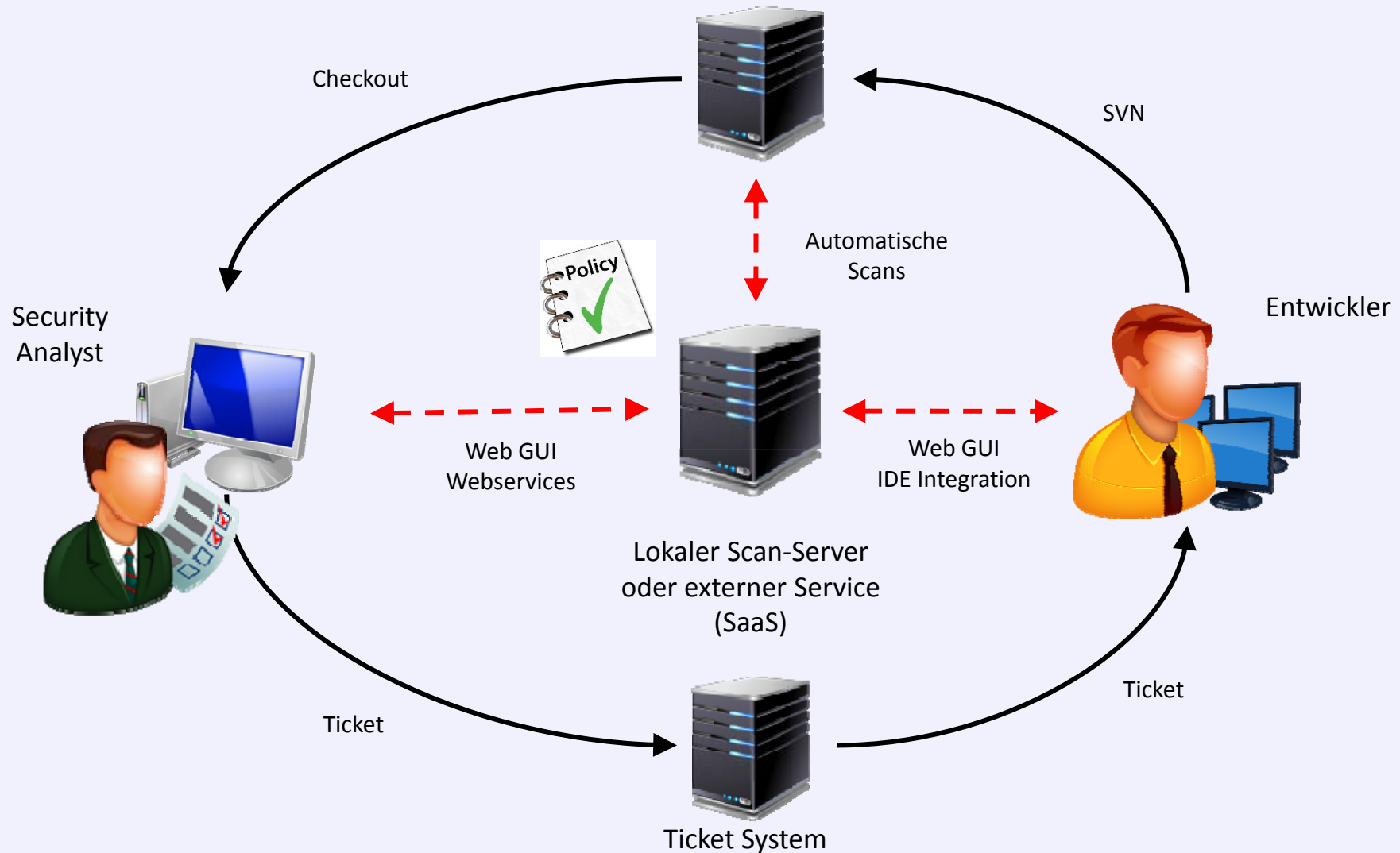
SAST Deployment: Heutige Integration



OWASP

The Open Web Application Security Project

Code Repository
(GIT, SVN, TFS, ...)



On-Premise vs. Off-Premise



OWASP

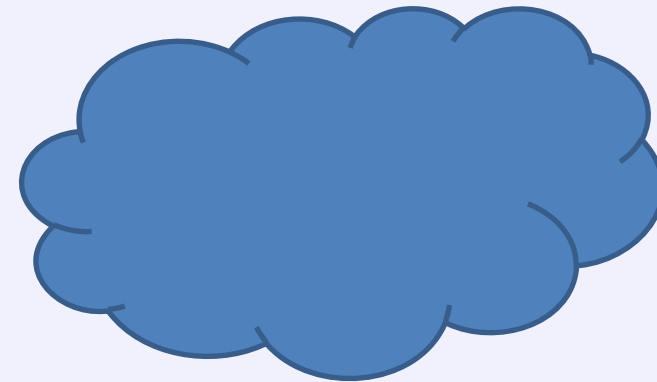
The Open Web Application Security Project

On-Premise
(Lokal intalliert)



Vs.

Off-Premise
(Cloud-Based)



SAST Deployment: Heutige Integration (2)



OWASP

The Open Web Application Security Project

Build Server
(Jenkins, Hudson,
Bamboo, ...)

Code Repository



Automatischer
Scan



Scan Server /
SaaS



Review

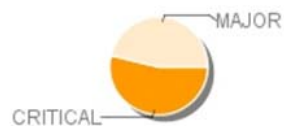


Entwickler



QA

Security defects
28
95.8% compliance
9/17 rules activated

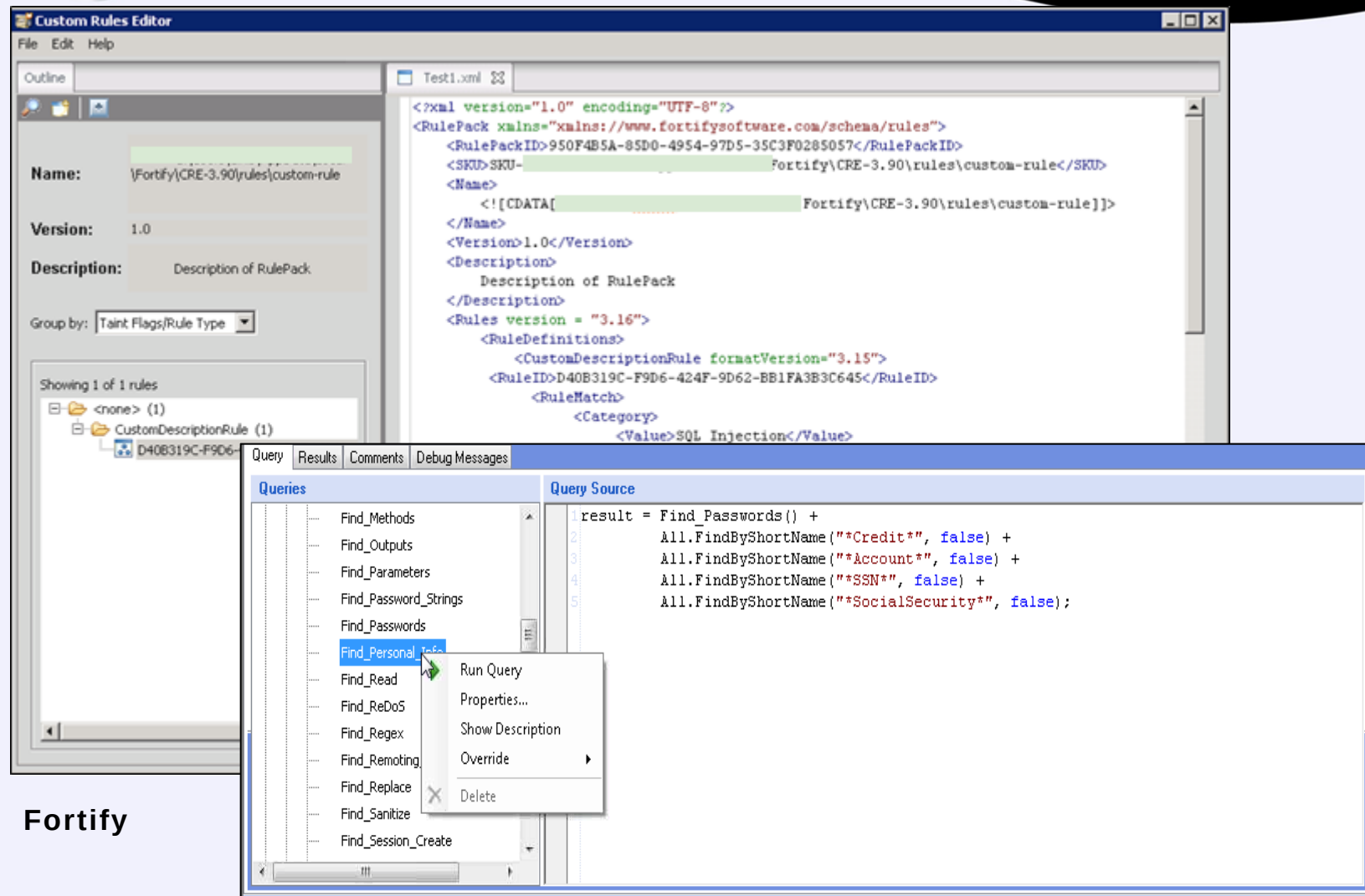


Custom Rules



OWASP

The Open Web Application Security Project



Fortify

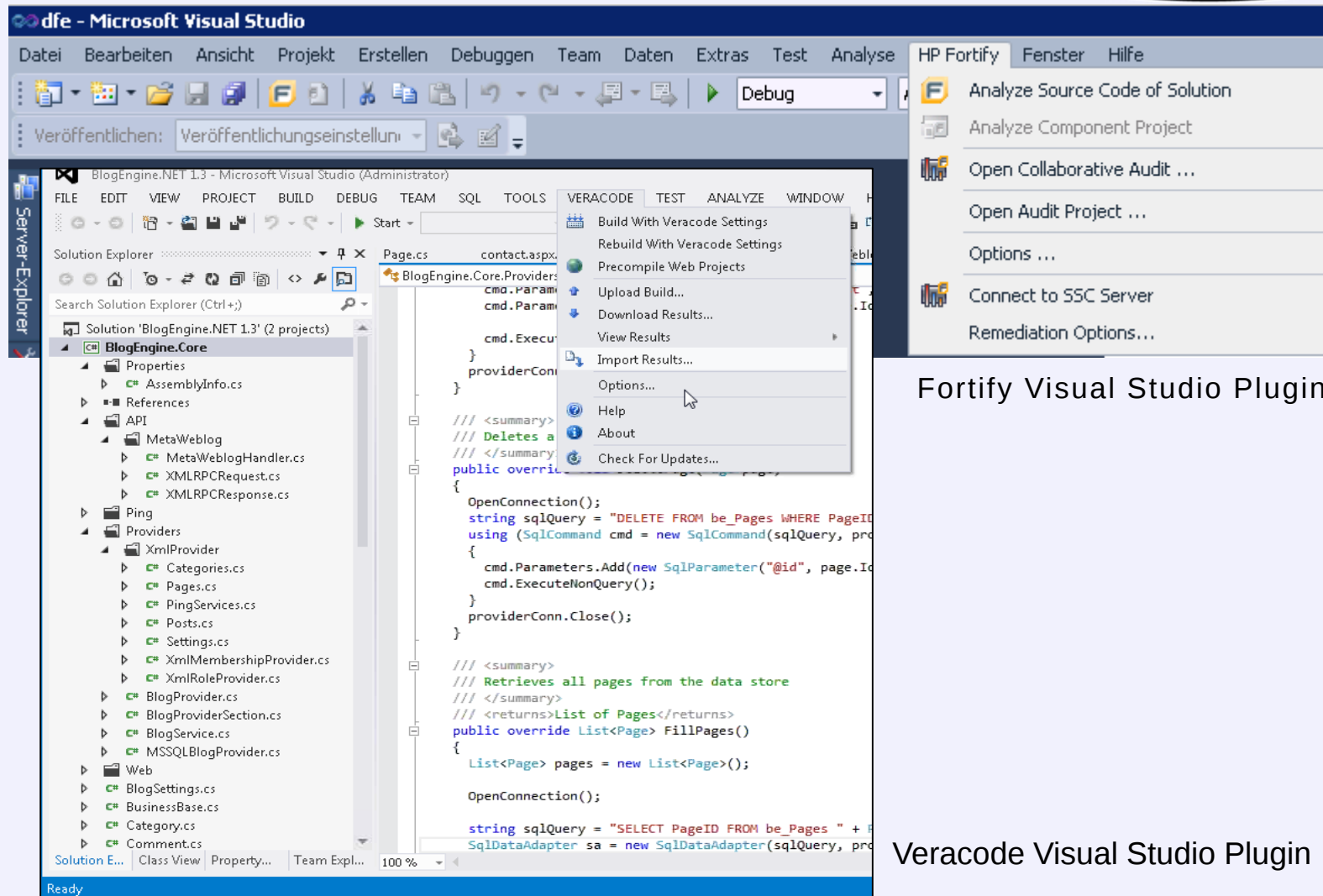
Checkmarx

Entwickler IDE Integration



OWASP

The Open Web Application Security Project



Fortify Visual Studio Plugin

Veracode Visual Studio Plugin

Wenige Hersteller, große Unterschiede



OWASP

The Open Web Application Security Project

- Qualität der Findings / Scan-Engine (False-Negatives / -Positives)?
- On-Site oder Off-Site („Cloud-Based“)?
- Sourcecode oder Bytecode?
- Integration: Kann das Tool in meine QA / Entwicklung integrieren?
- Unterstützte Sprachen: Werden alle relevanten Technologien (Sprachen, Frameworks) unterstützt, wenn ja wie vollständig?
- Möglichkeit eigener Scan Policys / Custom Regeln?
- Bedienbarkeit: Wer bedient das Tool (QA, Security Experten, Entwickler), wie gut lassen damit Analysen durchführen?
- Customizing: Wie viel ist wünschenswert, wieviel handelbar?
- Kosten!
-

Tools sind unterschiedlich für bestimmte Organisationen / Anforderungen geeignet und sollten daher immer vor einem Kauf evaluieren!



OWASP

The Open Web Application Security Project

Fazit



1. Der Einsatz von Tools ist häufig wichtig (bzw. erforderlich) um die Sicherheit einer (Web-)Anwendung zu gewährleisten.
2. Jedes Tool hat eine individuelle Eignung und Limitationen, die Verstanden werden müssen.
3. Ohne entsprechende Prozesse, Verantwortlichkeiten und Mitarbeiter die es bedienen können, ist ein (intern eingesetztes) Tool wertlos.





4. Jedes Tool ist nur so gut, wie der, der es bedient.
5. Jedes Tool kann stets nur eine Ergänzung zu manueller Verifikation darstellen, keinen Ersatz!
6. Insbesondere Enterprise SAST Tools erfordern ein Auswahl- und Einführungsverfahren (Pilot, On Boarding, etc.).





OWASP

The Open Web Application Security Project

Danke!

Fragen?