
ORACLE®

A Study on Dynamic Analysis and Penetration Testing Tools for Web Applications

Behnaz Hassanshahi

Postdoctoral Researcher

Oracle Labs Australia

November 2016

Safe Harbor Statement

The following is intended to provide some insight into a line of research in Oracle Labs. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. Oracle reserves the right to alter its development plans and practices at any time, and the development, release, and timing of any features or functionality described in connection with any Oracle product or service remains at the sole discretion of Oracle. Any views expressed in this presentation are my own and do not necessarily reflect the views of Oracle.

Program Agenda

1. Motivation
2. Overview
3. Web Application Analyzers
4. Research Opportunities & Conclusion

The ease of use and wide availability of web attack toolkits is feeding the number of web attacks, which is doubled in 2015.

Symantec Internet Security
Threat Report, 2016

Finding Security-Critical Vulnerabilities in Web Apps

Motivation

- Injection vulnerabilities are serious
  - 78% of websites are vulnerable to injection attacks [Symantec'16]
  - E.g., millions of Wix.com websites vulnerable in Nov 2016
  - E.g., 13 injection vulnerabilities in Joomla in the past 2 years
- Example injection attacks: XSS, SQLi
- Why client-side code?
- Why JavaScript?

Key Challenges in Analyzing JavaScript

- Event driven
- User interactive
- String intensive
- An untyped language
- That's why **dynamic analysis** makes sense!

Dynamic Analysis Needs Inputs

Focus of Our Study

- Test case generation for programming errors
  - Improve coverage
  - More suitable for developers

- Exploit generation
  - Exploits need more domain knowledge
  - Zero-day exploits can have complex patterns
  - Additional validation for confirmation required
  - **Less** dependent on the developers

Input Generation for JavaScript

- Value space
  - Less user interaction
  - Focus of web security toolkits
  - Easier to find => more security critical
- Event space
  - Finding programming bugs
  - More automation
  - Improves coverage
  - Mostly studied in research papers

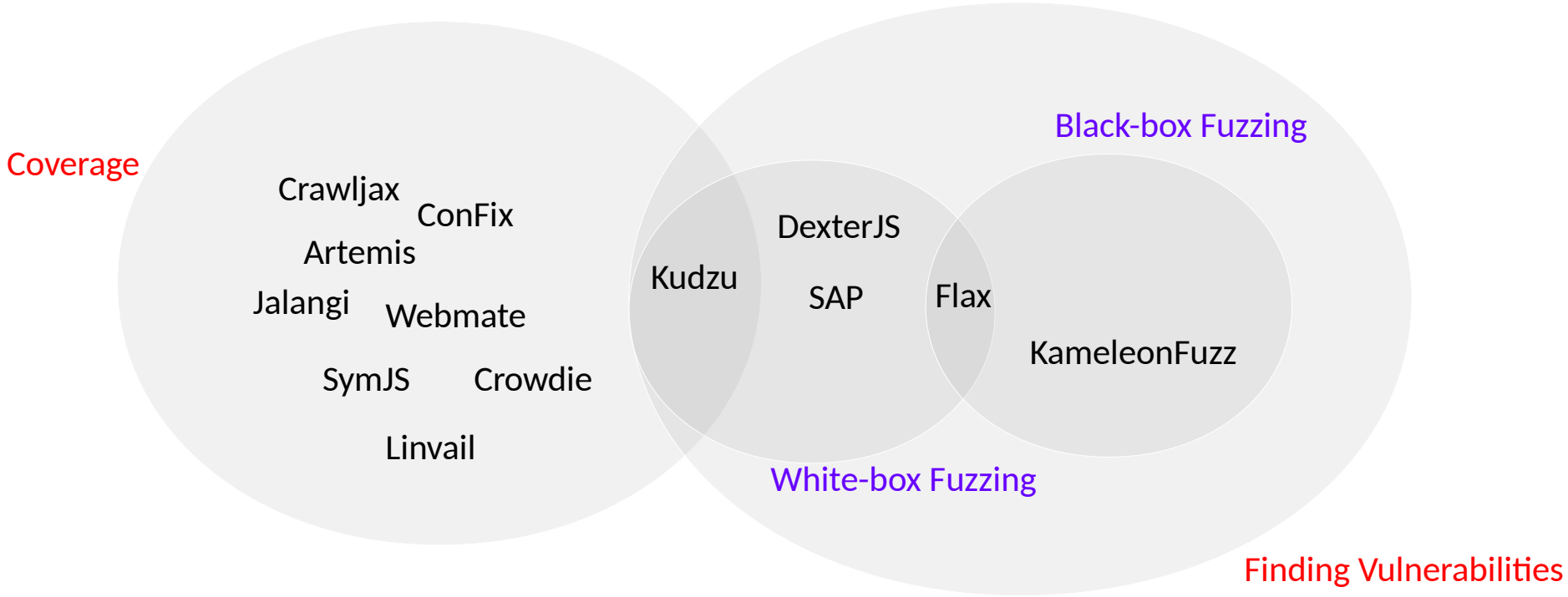
White-box vs Black-box Fuzzing

- Some people call test generators, fuzzers!
- **White-box**
  - Analyzes the source code e.g., dynamic symbolic execution
  - Better coverage, more automated
- **Black-box**
  - No source-code analysis
  - Light-weight, suitable for low hanging fruits

White-box vs Black-box Fuzzing

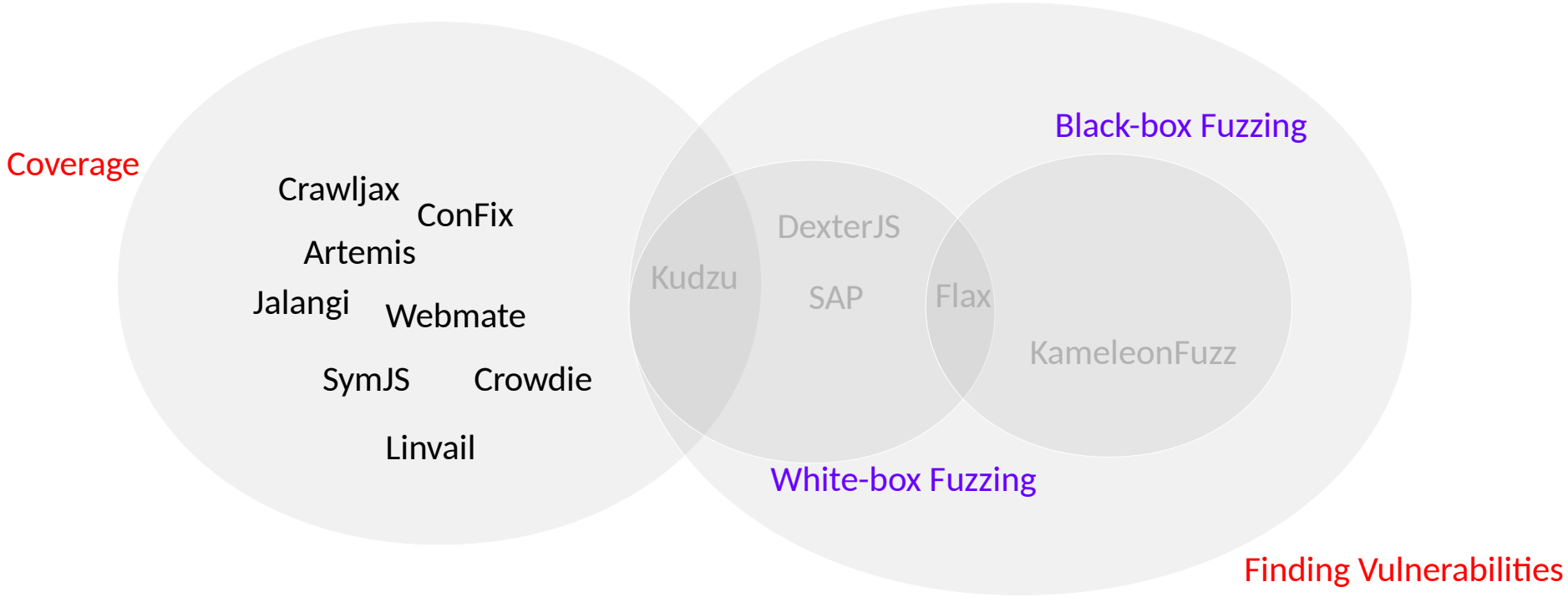
- Some people call test generators, fuzzers!
- **White-box**
  - Analyzes the source code e.g., dynamic symbolic execution
  - Better coverage, more automated
- **Black-box**
  - No source-code analysis
  - Light-weight, suitable for low hanging fruits
- **Research Problem:** finding the sweet spot
  - High coverage, light-weight, automatic

Research Prototypes



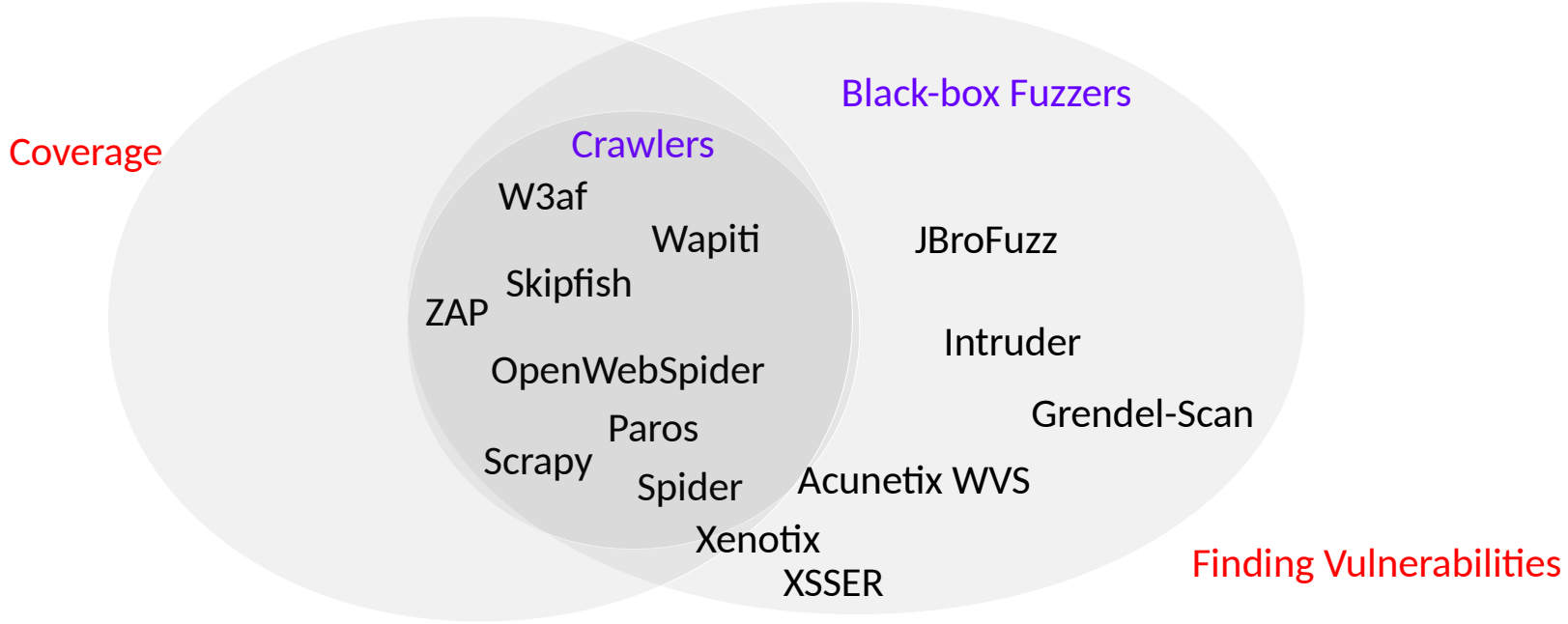
Research Prototypes

Available



Penetration Testing Tools

Available



Literature Review

Exploit Generation

Literature Review

- Goal
  - Confirming security vulnerabilities
- Scalability
  - Can test thousands of websites, more suitable for shallow vulnerabilities [Parameshwaran et al., FSE'15], [Lekies et al., CCS'13]

Exploit Generation

Based on Dynamic Taint Tracking

- Flax [Saxena et al., NDSS'10], SAP [Lekies et al., CCS'13], DexterJS [Parameshwaran et al., FSE'15]
- How do they work?
  1) Test harness, crawling to run
  2) Propagating taints from sources to sinks
  3) Logging useful information e.g., sink type, context, built-in filters
  4) Exploit generation: existing attack vectors/payloads + taint flows + meta data
  5) Exploit validation
- Need initial inputs

Exploit Generation

Based on Symbolic Execution

- Kudzu [Saxena et al., SP'10]
  - Main focus on complex string operations
  - Finds client-side injection vulnerabilities
  - No need for initial test harness
- How does it work?
  1) Random GUI exploration to generate event sequences
  2) Recording an execution trace of the program with concrete inputs
  3) Symbolic execution on the trace
  4) Generating new input values and executing them with same event sequences
  5) Goes to 2
- Scalability: only tested on few apps, not clear

Test Case Generation

Literature Review

- Goal
  - Finding programming errors with good coverage
- Scalability
  - Test few apps with high coverage [Artzi et al., ICSE'11], [Sen et al., ESEC/FSE'13], [Li et al., FSE'14], [Christophe et al., SANER'16]

Test Case Generation

Random Testing

- Artemis [Artzi et al., ICSE'11]
  - Focuses on event-driven aspect of JavaScript
  - Improving coverage
- How does it work?
  1) Starts with random events
  2) Observes the effect and generates new inputs
     - Explores new paths
     - Several prioritization rules
  3) Runs the input and goes to 2

Test Case Generation

Model-based Testing

- Crawljax [Mesbah et al., ICWE'08]
  - Focuses on AJAX-based apps
  - Improves event coverage
- How does it work?
  1) The robot simulates user actions, e.g., clicks and text input
  2) Updates state-flow graph of the application
  3) Generates a static page
  4) Explores all clickable elements
  5) Runs and goes to 2
- Complicated, might not be stable in practice

Test Case Generation

Symbolic Execution

- Jalangi [Sen et al., ESEC/FSE'13], SymJS [Li et al., FSE'14]
- How do they work?
  1) Start with random event and data inputs
  2) Collect paths constraints, flip a condition to generate new input
  3) Run the input and go to 2
- Jalangi based on record-replay is not supported anymore
- SymJS tested on few apps and not available

Technical Challenges

Instrumentation

- Browser-based
  - Advantages: fast
  - Disadvantages: compatibility
  - Examples: Artemis [Artzi et al., ICSE'11] , SAP [Lekies et al., CCS'13]
- Source-to-source rewriting
  - Advantages: compatibility
  - Disadvantages: slow, possible to change the semantics
  - Examples: Jalangi [Sen et al., ESEC/FSE'13], DexterJS [Parameshwaran et al., FSE'15], Linvail [Christophe et al., SANER'16]

Tool	Source code	bug	Security Vulnerability	Focus	Technique	Available	Organisation
Kudzu	JS	-	DOM-based XSS	Structured inputs	Dynamic symbolic execution	no	UC Berkeley Oakland 2010
Artemis	JS	Runtime errors	-	Coverage	Feedback-directed event sequence generation	yes	Aarhus University, originally by IBM ICSE 2011
Jalangi	JS	Undefined origin, etc.	-	Instrumentation	Source-to-source rewriting to provide callbacks for analysis	yes	UC Berkeley, Samsung ESEC/FSE 2013
SymJS	JS	-	-	Coverage	Symbolic execution	no	Fujitsu FSE 2014
DexterJS	JS	-	DOM-based XSS	Exploit generation	Crawling+ instrumentation+ taint tracking	no	NUS, Acquired by Intel 2015
Linvail	JS	NaN, etc.	-	Instrumentation	Shadow execution	Yes	Vrije Universiteit Brussel SANER 2016

Some Free Penetration Testing Tools

Burp Suite

w3af

ZAP

Free Black-box Fuzzers

How do they work?

- Leverage a database of known exploit payloads
- Start by crawling the target web application
- Identify reachable entry points
  - Enumerating all e.g., URL parameters, input fields, cookies
  - Manual selection
- Generate and execute (mutations of) input strings based on the payloads
- Analyze the HTTP responses for keywords and patterns

Free Black-box Fuzzers

Observations

- XSS Peeker (Bazzoli et al., IFIP'16) evaluates fuzzers
  - Redundant payloads
  - Problems in validation
  - Lack of feedback
- Crawlers not effective at following links through active content technologies [Bau et al., S&P'10]
- Not there yet for automatic vulnerability detection!

Tool	Security Vulnerability	Focus	Technique	Available	Organisation
Burp Suite	SQL injection, XSS, etc.	Scanning, exploit generation etc.	Crawling+Mutating over a set of known inputs	Yes, trial	PortSwigger Web Security
JBroFuzz	SQL injection, XSS, buffer overflow, etc.	Fuzzing existing inputs	Mutating over a set of known inputs	Yes	OWASP
ZAP	SQL injection, XSS, etc.	Scanning, exploit generation etc.	Mutating over a set of known inputs	Yes	OWASP
Paros Proxy	SQL injection, XSS, etc.	Editing/viewing HTTP(S) messages, fuzzing	Crawling+Mutating over a set of known inputs	Yes	-
w3af	SQL injection, XSS, etc.	Editing/viewing HTTP(S) messages, fuzzing	Crawling+Mutating over a set of known inputs	Yes	-

Observations

- It is important to design an analysis with **security problems** in mind
  - Finding opportunities to improve scalability e.g., several optimizations through staged analysis
  - Avoid collecting unnecessary information e.g., slicing
  - Avoid missing useful information e.g., precise string analysis
  - That's why web toolkits are widely used!
- Scaling coverage-based techniques for security to achieve automation
- Analyzing client + server side code for deeper vulnerabilities
- More intelligent fuzzers
  - Feedback-directed

Conclusion

- There is a gap between research prototypes and web scanning tools
- Fuzzers are highly dependent on known payloads and manual effort
- Research tools are mostly focused on programming errors and coverage

- What to do next?
  - Adapting coverage-based techniques for security problems
    - High coverage
    - More automation
    - Deeper vulnerabilities

Thanks

E-mail: behnaz.hassanshahi@oracle.com