The role of program analysis in web security

Behnaz Hassanshahi
Principal Researcher at Oracle Labs Australia

BBC

Home News Sport Business Innovation Culture Travel Earth Video Live

Optus: How a massive data breach has exposed Australia

29 September 2022

Share

By **Tiffanie Turnbull,** BBC News, Sydney

Optus is the country's second-largest telecommunications company

Last week, Australian telecommunications giant Optus revealed about 10 million customers - about 40% of the population - had personal data stolen in what it calls a cyber-attack.

https://www.bbc.com/news/world-australia-63056838

Copyright © 2024, Oracle and/or its affiliates

What happened in the Optus data breach incident?

Optus is Australia's second-largest telecoms provider

In September 2022 Optus announced it had been the subject of a massive data breach with as many as 10 million customer accounts exposed, current and former customers' data was stolen - including **names**, **birthdates**, **home addresses**, **phone** and e**mail contacts**, and **passport** and **driving licence numbers**

Problem: a public API endpoint did not require authentication, exposed to anyone on the internet with knowledge of that endpoint [URL]

Optus still hasn't confirmed how the data was accessed and says the attack was sophisticated

But the home affairs minister, Claire O'Neil, has said the vulnerability was akin to **Optus leaving a window open**

Copyright © 2024, Oracle and/or its affiliates

Background: Web Application N-tier Architecture

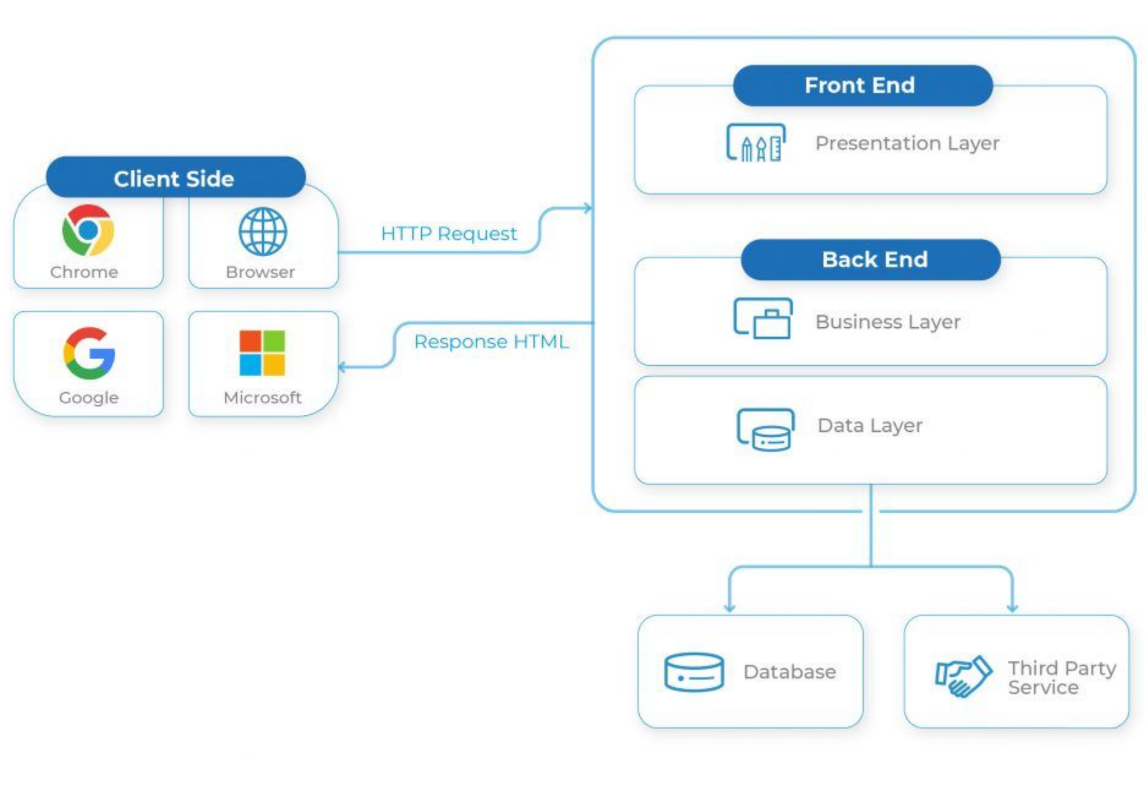
Divides an application into logical layers and physical tiers:

Presentation Layer

Business layer

- Client-side Component (Front-end)
- Application Programming Interface (API)
- Server Instance / Cloud Instance

Data Layer: Database



https://images.clickittech.com/2020/wp-content/uploads/2022/03/10202347/Diagram-54-1024x815.jpg

Copyright © 2024, Oracle and/or its affiliates

Presentation Layer: Client-side Component (Front-end)

The client-side of a web application enables users to interact with the backend

There are two main types of web applications:

- Single-page applications: takes all data from a single page of HTML and updates the content using AJAX calls
- Traditional or legacy HTML applications: the server hosts multiple pages and if there is any update, the user reloads that particular page

Modern technologies: React, Vue.js, Angular.js, etc.

Copyright © 2024, Oracle and/or its affiliates

Business Layer: Application Programming Interface (API)

A mediator that allows apps to communicate with each other.

Different types of APIs:

- RESTful API: Representational State Transfer API in lightweight JSON format. It is meant to be stateless and fast
- SOAP: Simple Object Access Protocol uses XML for data transmission. It usually requires more bandwidth and advanced security

* XML-RPC: Extensible Markup Language – Remote Procedure Calls use specific XML format for data transmission
* JSON-RPC: It uses JSON format for data transmission

 Copyright © 2024, Oracle and/or its affiliates

Server-side vs client-side rendering

Server-side rendering

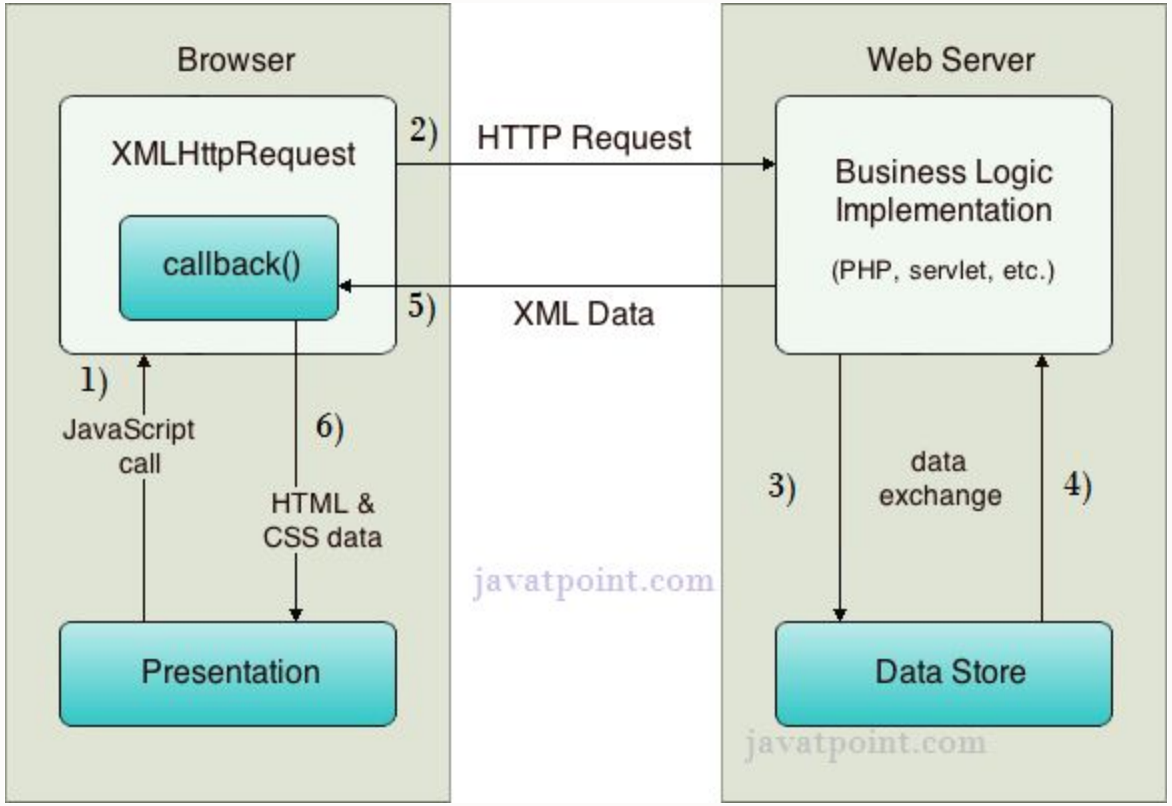
- Visiting a website using a URL, a request is sent to a server and response received by browser
- Response can contain CSS, JavaScript & HTML

Client-side rendering

- A request is sent to the server and the main app Skelton is received.
- After that, if you go to different pages, the content is made dynamically using JavaScript

Copyright © 2024, Oracle and/or its affiliates

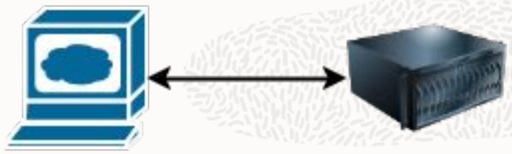
Asynchronous Javascript And XML (AJAX)



https://www.javatpoint.com/how-ajax-works

Copyright © 2024, Oracle and/or its affiliates

Web application security model



Client side:

- Same origin policy: browsers isolate sites from each other, while running in the same process

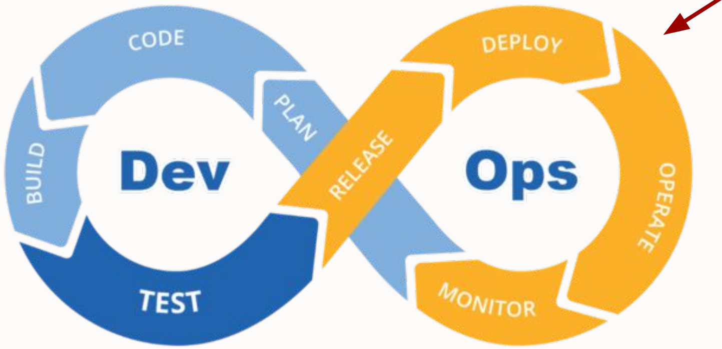
Server side:

- Protecting HTTP endpoints on the server from malicious input
- Sanitizing outputs

Copyright © 2024, Oracle and/or its affiliates

Software Development Life Cycle and DevOps

Security is often neglected until the end and penetration testers test the application in production, semi-manually.



Copyright © 2024, Oracle and/or its affiliates

Security testing post-development

Penetration testers or pen-testers are often tasked with finding security issues once the development is finished and application is ready for deployment or is in production

Pen-testers adopt the mindset of an attacker and search for ways to breach the application

Pen-testers usually use tools like Burp Suite and ZAP

- A collection of tools and scripts like crawlers and payload generators
- Intercept HTTP requests and perform API testing
- Require manual guidance and more suitable for humans
- Limited support for client-side vulnerabilities, such as DOM-based XSS

Issues can be much more difficult and costly to address when security testing is left to the end

 Copyright © 2024, Oracle and/or its affiliates

DevSecOps

DevSecOps is an enhancement to DevOps that builds security into all aspects of the software development process

Allows addressing security issues from the very start of the project

Through threat modeling, team discussion, and **Continuous Integration (CI) testing**, security implications are identified in development environments, rather than waiting until the end

Another name for this approach is shift-left security.

 Copyright © 2024, Oracle and/or its affiliates

Security testing as part of Continuous Integration (CI)

Security testing in CI requires scanning the code as developers commit code to the shared software repository

Requirements to be practical:

- **Automation**: the scanner should be fully automated. Manual reviews and audits are hard (similar to needle in a haystack problem) and error-prone
- **Performance**: the scanner should take a reasonable amount of time to allow fast development and save costs especially in cloud environments that charge per minute
- **Reasonable precision**: if tools produce too many false positives, developers might turn it off

Different flavours: static and dynamic analysis

 Copyright © 2024, Oracle and/or its affiliates

Back to security analysis of web applications

We need program analysis to create automated security testing tools

Program analysis is the systematic examination of a program to determine properties, such as correctness. It requires representations of the code that is feasible to process and abstractions that model the behavior of the program and its states

Should be suitable for the nature of web applications

- Consist of a client and server components
- Can be multi-layered
- Can be a single-page application, etc.

Certain aspects of web applications should be tested statically or dynamically based on their nature

 Copyright © 2024, Oracle and/or its affiliates

Static and dynamic web application analysis

Static analysis

- Can be applied to the server-side code written in languages like Java and Go
- Suitable for intra-procedural or simple inter-procedural bugs
- Can miss complex vulnerabilities, especially in dynamically typed languages like JavaScript and Python

Dynamic analysis and fuzzing

- More suitable for dynamically typed languages like JavaScript and Python
- Usually has low false positives
- Should achieve high coverage but usually the ground-truth is hard to construct

Copyright © 2024, Oracle and/or its affiliates

Access to source code

White-box analysis

- Full access and analysis of the source code
- Advantage: detects more complex bugs
- Disadvantages: expensive and does not scale well and can be hard to set up

Grey-box analysis

- Access to certain parts of the source code with lightweight analysis
- Advantage: Scales to large applications and can leverage feedback for improving performance
- Disadvantages: Can miss complex bugs and hard to set up

Black-box analysis

- No access to the source code
- Advantage: Easy to set up
- Disadvantages: Can miss bugs and have lower coverage than white-box and grey-box

Copyright © 2024, Oracle and/or its affiliates

Gelato: our Dynamic Application Security Testing (DAST) project [1]

Based on requests from product teams, our first goal was to detect client-side security issues such as DOM-based XSS and test REST APIs on the backend

- Static analysis tools like TAJS [2] could not scale to real-world applications due to challenges in analyzing JavaScript
- Dynamic analysis tools like Jalangi2 [3] that performed dynamic symbolic execution of JavaScript had scalability issues
- Basically, none of the tools worked on complex applications that were developed at Oracle

[1] "Gelato: Feedback-driven and guided security analysis of client-side web applications", Behnaz Hassanshahi, Hyunjun Lee, Paddy Krishnan, SANER 2022
[2] https://github.com/cs-au-dk/TAJS
[3] https://github.com/Samsung/jalangi2

 Copyright © 2024, Oracle and/or its affiliates

Design

Automated crawling

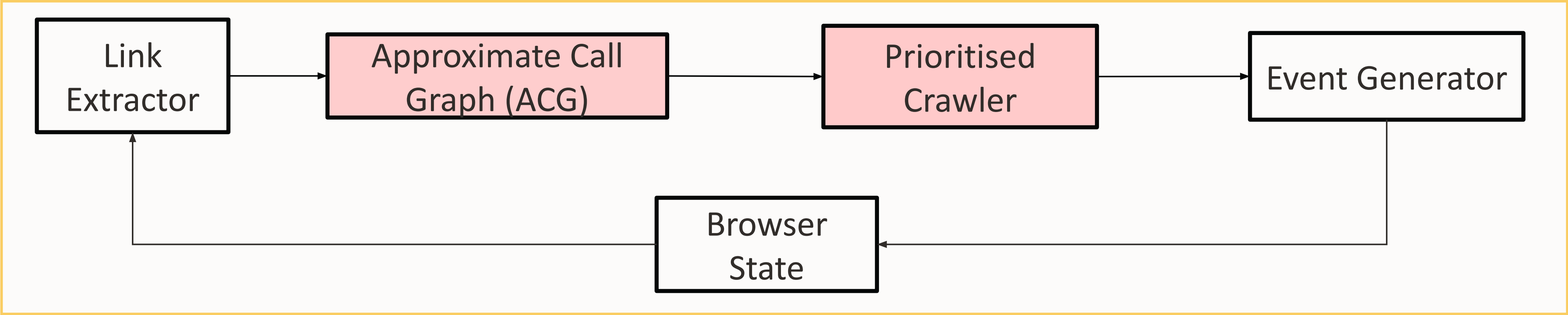
- Guided to reach security sensitive methods: use static analysis to create call graph and compute distance to security-sensitive methods
- Feedback-driven: refines the call graph model in a feedback-driven way using execution traces generated by the crawler at runtime

Security analysis

- Practical taint analysis: use a novel light-weight staged taint inference
- REST API detection: detect AJAX calls in JavaScript as well as other REST calls
- Reports wrong sanitizers based on the context in the HTTP response

Copyright © 2024, Oracle and/or its affiliates

Automated crawling



ACG

- Seeded using simple static analysis
- Refined as different pages are visited
- Compute distance to security-sensitive points

Prioritisation

- Unseen events
- Search boxes
- Avoid revisiting states

Features Supported

- Authentication
- Session specific information

 Copyright © 2024, Oracle and/or its affiliates

Automatic Root Cause Quantification for Missing Edges in JavaScript Call Graphs [1]

We tried to quantify the relative importance of different root causes of missed edges in JavaScript static call graphs on modern frameworks

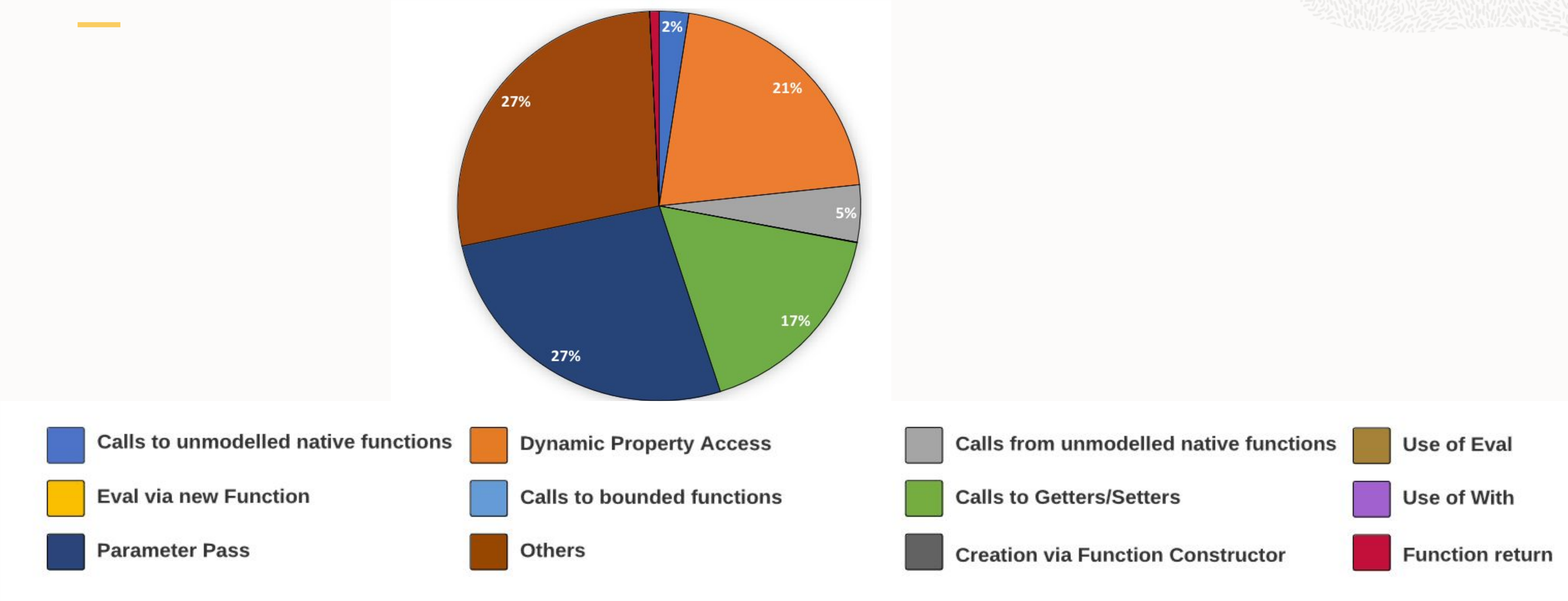
We measured call graph recall for our benchmarks by comparing the ACG static call graphs with our collected dynamic call graphs.

Experiments were conducted on TodoMVC [2] benchmarks

[1] "Automatic Root Cause Quantification for Missing Edges in JavaScript Call Graphs", Madhurima Chakraborty, Renzo Olivares, Manu Sridharan, Behnaz Hassanshahi, ECOOP 2022
[2] https://github.com/tastejs/todomvc

 Copyright © 2024, Oracle and/or its affiliates

Root causes for pessimistic ACG for Juice Shop



Copyright © 2024, Oracle and/or its affiliates

Call Graph Refinement

Example:

- Static call graph construction cannot resolve fn
  - No calls to target exist
- Event **add1** trigger call at runtime: Updates call graph with edge
  - New edge used when **add2** processed

```
function target() { fetch('');}

function handle_click() {fn.call(this);}

function btn_cb_1() {handle_click();}
function btn_cb_2() {handle_click();}

<button id='add1' onclick='btn_cb_1()'>
 button_1
</button>
<button id='add2' onclick='btn_cb_2()'>
 button_2
</button>
```

Added dynamically

Copyright © 2024, Oracle and/or its affiliates

DOM-based XSS Detection

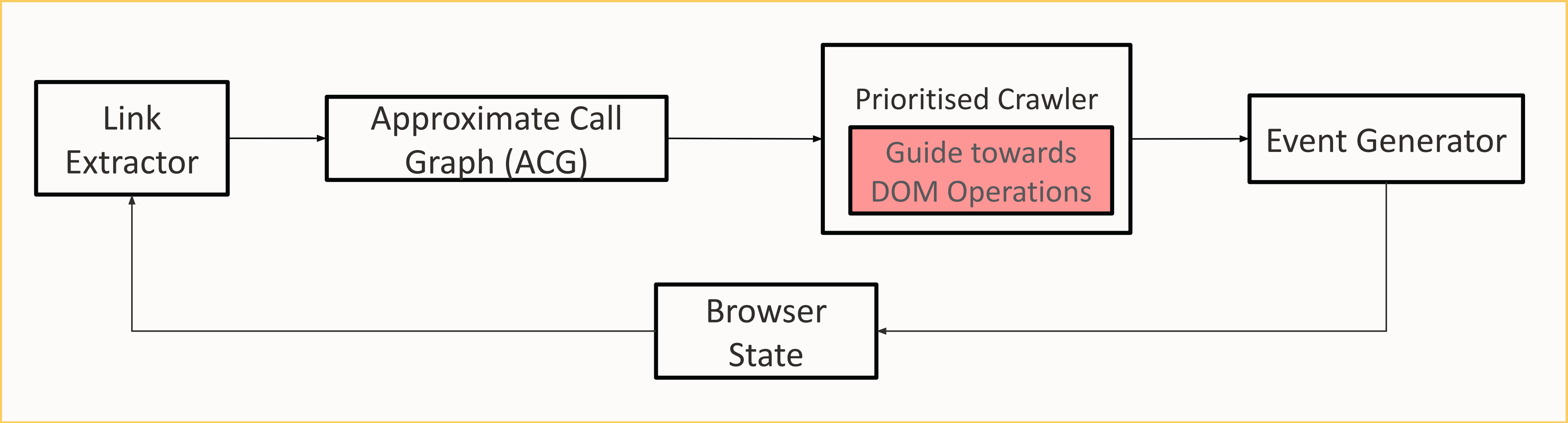
- Client-side script writes malicious data into the HTML document
- Staged taint inference:
  - String matching algorithms
  - Differential testing
  - Trace analysis

```
// Taint source
1. var tmp = location.hash;
// Value "similar" to value in taint source
2. tmp1 = tmp.substring(i,j);
3. tmp1 = tmp1 + "123";
//Write operation: DOM-XSS
4. document.write(tmp1);
// "…123" is written
5. document.write(safeValue + "123");
```

- At line 4 taint tracking reports correlation with value from `location.hash`
- Value written at line 5 is not reported: Benign value in `safeValue`

 Copyright © 2024, Oracle and/or its affiliates

Guiding the crawler towards DOM-based XSS sinks



ACG

- Seeded using simple static analysis
- Refined as different pages are visited
- Compute distance to security-sensitive points

Prioritisation

- Unseen events
- Search boxes
- Avoid revisiting states
- JavaScript DOM Operations

Features Supported

- Authentication
- Session specific information

 Copyright © 2024, Oracle and/or its affiliates

Gelato intercepts requests made by the browser

Crawl JavaScript client code

- Gelato's state aware crawler
- Generate browser events

Intercept requests with MITM proxy

- Record requests
- Pass requests to server-side web application

Replay recorded requests

- Send payload(s)
- Integrate with a server-side fuzzer

Investigate reports

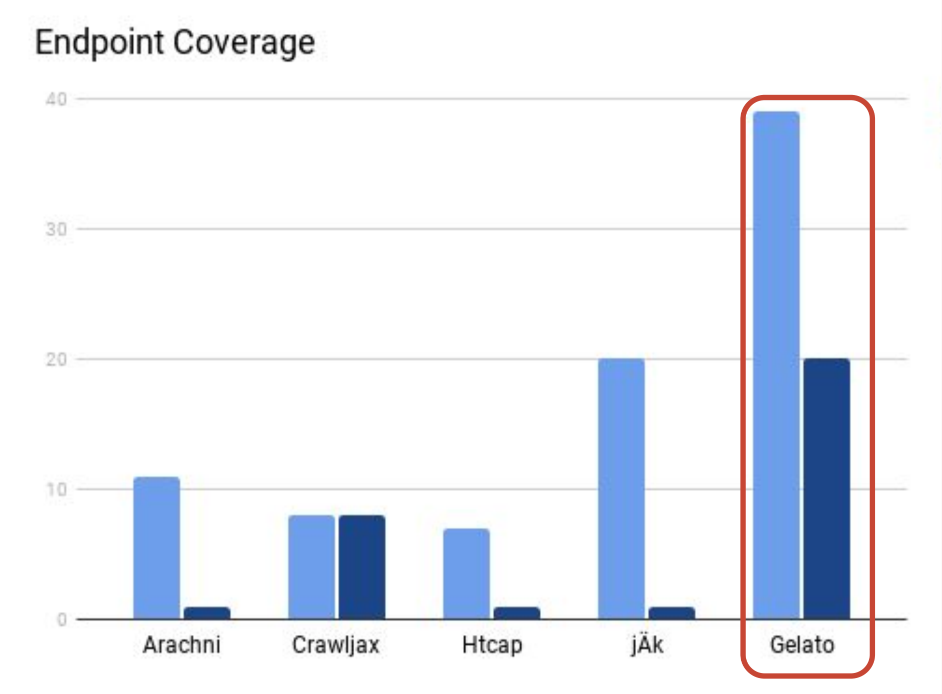
- Review and confirmation of security issues (may require manual investigation)

Copyright © 2024, Oracle and/or its affiliates

Gelato outperforms existing tools

Code coverage

Comparing crawlers on two modern Oracle apps



Open-source

Found a **zero-day** in **MongoDB npm package**

- Found through analyzing **mongo-express v 0.51.0 application**
- Note that the fuzzer would be clueless w/o Gelato
- Details of the vulnerability:
  - Denial of Service (DoS) in mongodb v3.1.12 npm and its prior versions
  - Severity: High
  - ~ 2 Million downloads per week
  - https://www.npmjs.com/advisories/1203

 Copyright © 2024, Oracle and/or its affiliates

Example: HTML Report

Gelato Report			
Gelato Version	0.3.1 (a324bfad36cb67cf568f3baf7c50e2ed7e7e7f90)	**GET Requests**	353
Target Host	localhost	**POST Requests**	41
Target Port	3000	**Other Requests**	2
Crawler Start/End	2020-07-26 20:56:52+00:00 / 2020-07-28 14:58:35+00:00	**Endpoints**	25
Replay Start/End		**DOM-XSS Flows**	1
Crawler Configuration	urls.json crawler.json	**Reflected Requests**	28
Replay Configuration	replay.json	**Insecure Headers**	4
Swagger Specification	swagger-spec.yaml	**Wrong Sanitizers**	4

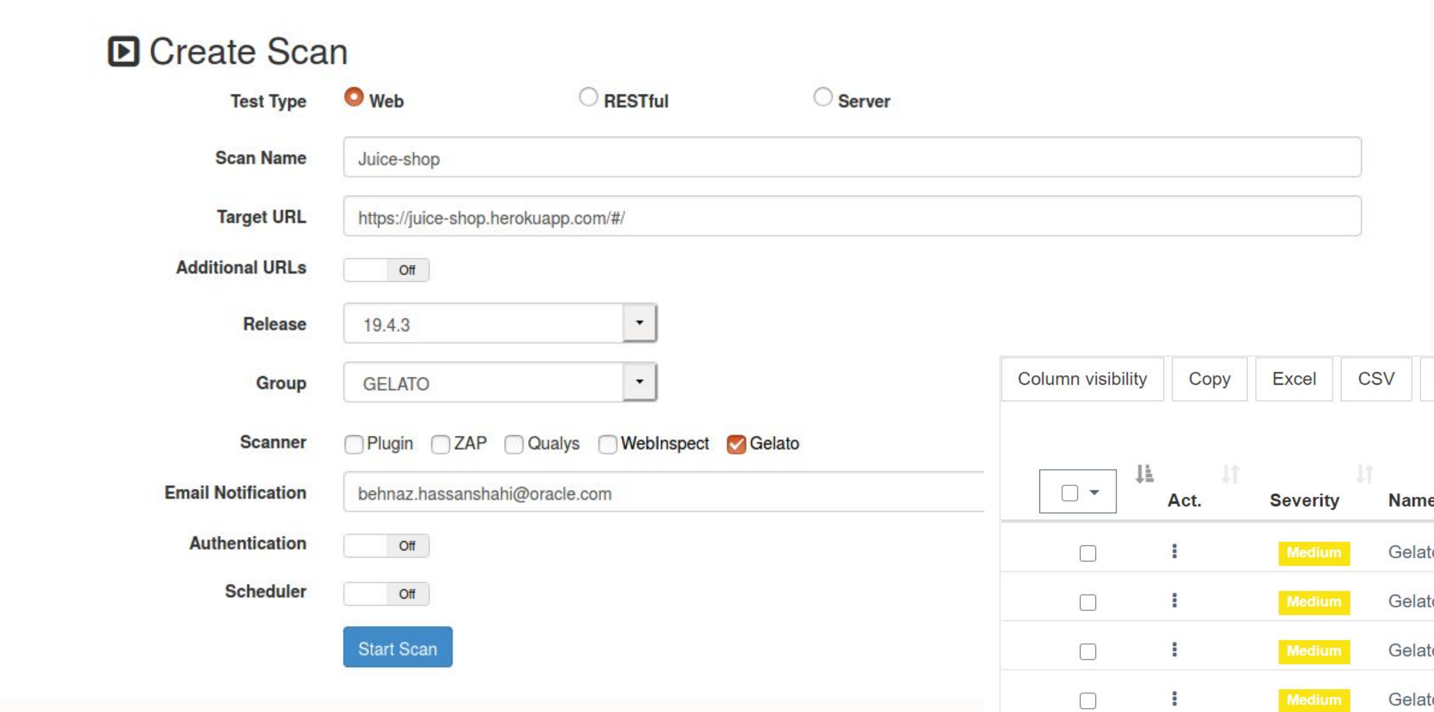
Coverage Statistics					Security Reports 37			
GET Requests	POST Requests	Other Requests	Endpoints	Swagger Specification	DOM-XSS Flows	Reflected Requests	Insecure Headers	Wrong Sanitizers

GET Requests	get_list.txt
Total Number	353

1	http://localhost:3000/
2	http://localhost:3000/about

Copyright © 2024, Oracle and/or its affiliates

Gelato is integrated into the CI/CD of many applications at Oracle



	Act.	Severity	Name	CWE	Date	Age	SLA	Reporter	Status
	⋮	Medium	Gelato_Insecure_Headers_1		Jan. 27, 2021	1	89	famonolith_api	Active
	⋮	Medium	Gelato_Insecure_Headers_2		Jan. 27, 2021	1	89	famonolith_api	Active
	⋮	Medium	Gelato_Insecure_Headers_3		Jan. 27, 2021	1	89	famonolith_api	Active
	⋮	Medium	Gelato_DOM_XSS_1		Jan. 27, 2021	1	89	famonolith_api	Active
	⋮	Medium	Gelato_DOM_XSS_2		Jan. 27, 2021	1	89	famonolith_api	Active

Copyright © 2024, Oracle and/or its affiliates

Grey-box fuzzing of server-side

What if we could collect feedback from the server-side code?

The **server-side** of web applications is treated as a black-box in Gelato

Research question: can we benefit from the tremendous advancements in fuzzing for binary programs that have followed the advent of the American Fuzzy Lop (AFL)?

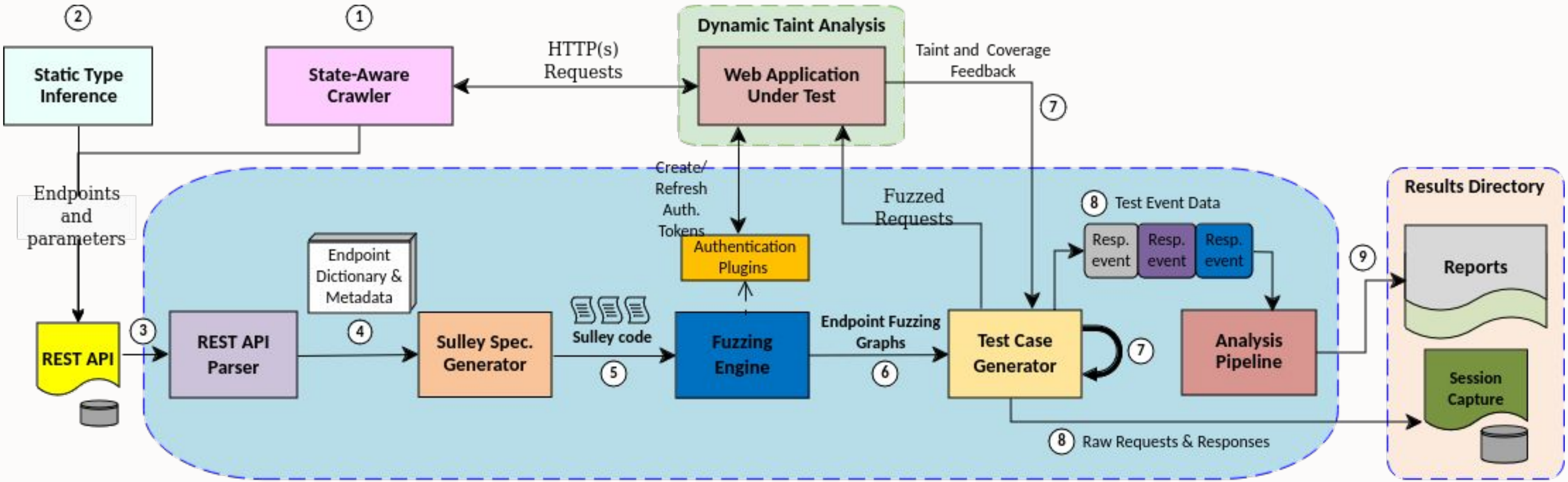
Grey-box fuzzing of server-side: requires lightweight instrumentation of the server-side code to collect feedback

Feedback:

- Coverage: use coverage information to skip inputs in the test plan that would likely exercise well-covered code.
- Taint: use taint inference to locate sections of input that influence values at key progra locations (e.g. buffer index, or magic byte checks).

 Copyright © 2024, Oracle and/or its affiliates

Model-Based, Feedback-Driven, Greybox Web Fuzzing with BackREST



“Experience: Model-Based, Feedback-Driven, Greybox Web Fuzzing with BackREST”, François Gauthier, Behnaz Hassanshahi, Benjamin Selwyn-Smith, Trong Nhan Mai, Max Schlüter, Micah Williams, ECOOP 2022

Copyright © 2024, Oracle and/or its affiliates

Results on open-source Node.js packages

Table 3 Impact of the coverage (C) and taint (T) feedback loops on runtime and total coverage, compared to baseline (B)

Benchmark	Coverage (%)			Time (hh:mm:ss)				
	B	C	CT	B	C		CT	
Nodegoat	80.31	78.54	75.59	0:42:39	0:06:07	(7.0×)	00:05:44	(7.4×)
Keystone	48.31	48.05	45.43	5:46:29	0:49:25	(7.0×)	0:13:23	(25.9×)
Apostrophe	—	48.40	45.52	—	11:11:42	–	6:17:34	–
Juice-Shop	74.73	76.34	75.85	12:48:15	1:10:31	(10.9×)	1:08:26	(11.2×)
Mongo-express	69.62	69.57	66.59	2:21:49	0:16:07	(8.8×)	0:11:07	(12.8×)

Copyright © 2024, Oracle and/or its affiliates

Results on open-source Node.js packages

Table 4 Impact of the coverage (C) and taint (T) feedback loops on bug reports, compared to baseline (B)

Benchmark	(No)SQLi			Cmd injection			XSS			DoS		
	B	C	CT	B	C	CT	B	C	CT	B	C	CT
Nodegoat	0	0	3	0	0	3	5	5	5	0	0	0
Keystone	0	0	0	0	0	0	1	1	0	0	0	0
Apostrophe	0	0	0	0	0	0	1	1	1	2	1	1
Juice-Shop	1	1	2	0	0	1	4	1	1	1	0	0
Mongo-express	0	0	5	0	0	2	0	0	0	3	3	3
Total	1	1	10	0	0	6	11	8	7	6	4	4

Copyright © 2024, Oracle and/or its affiliates

Takeaways from the BackREST exploration

We filed eight 0-days, out of which six were not reported by any other fuzzer.

We showed that extending a black-box web application fuzzer with simple grey-box analyses that use coverage and taint feedback to skip and select rather than derive new inputs can reduce runtime while increasing the number of reported bugs

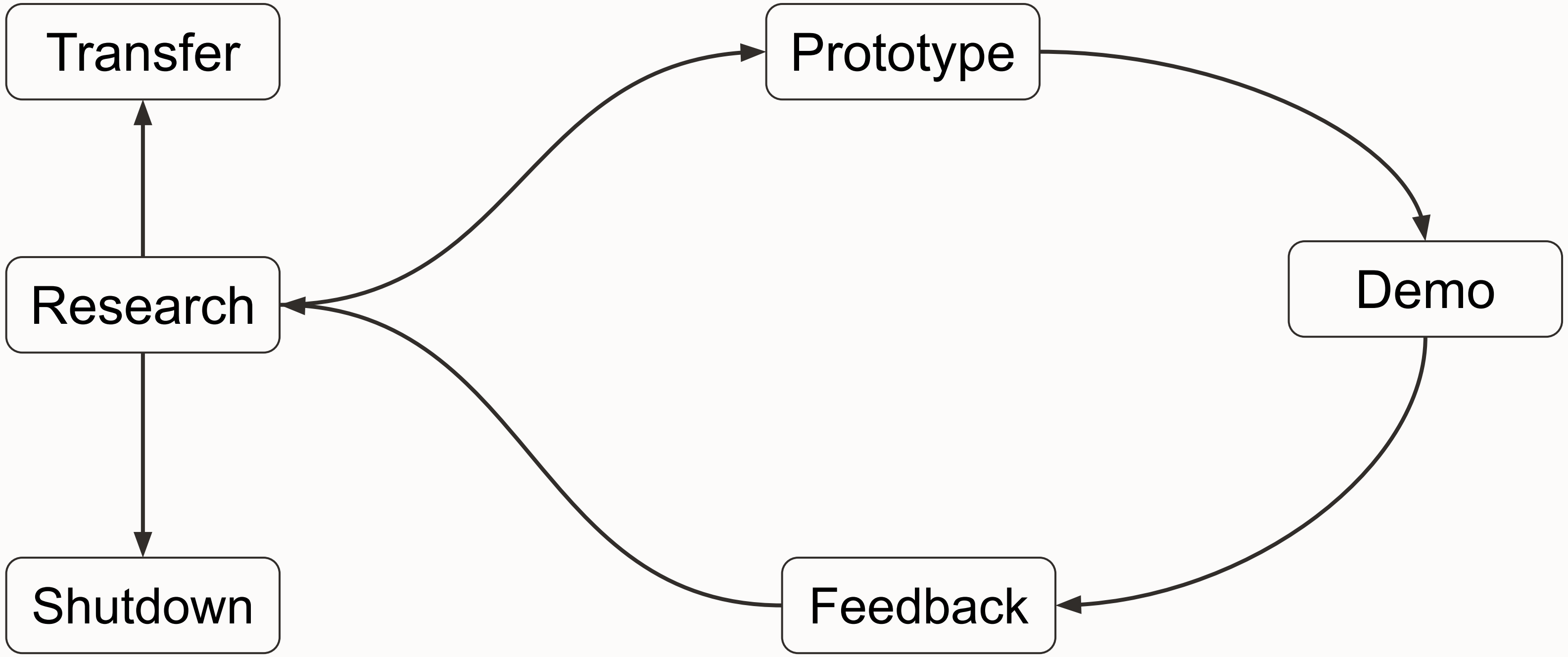
We faced challenges in instrumenting complex Node.js products

Other findings: we tried to infer endpoint dependencies from crawling logs and then constrained the fuzzing schedule of BackREST to honour them. This did not improve coverage for all but the Mongo-express application

 Copyright © 2024, Oracle and/or its affiliates

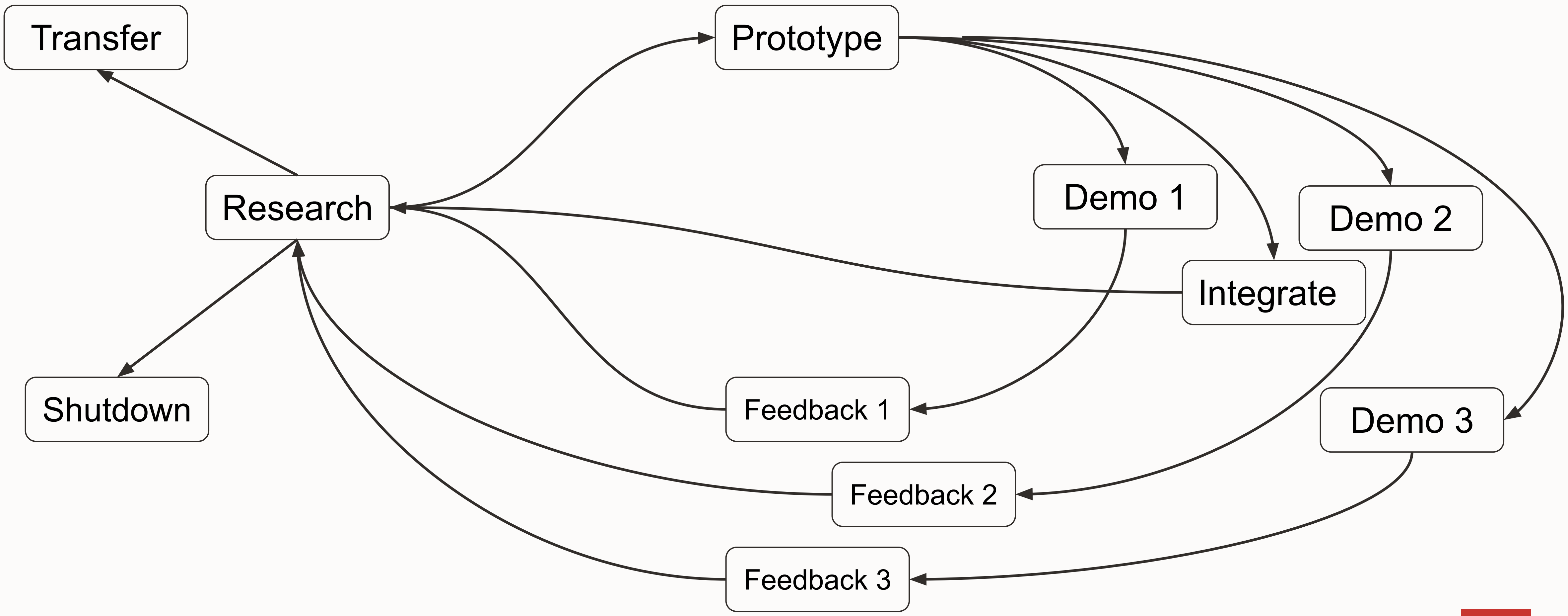
From a research prototype to a product

The project life cycle we expect



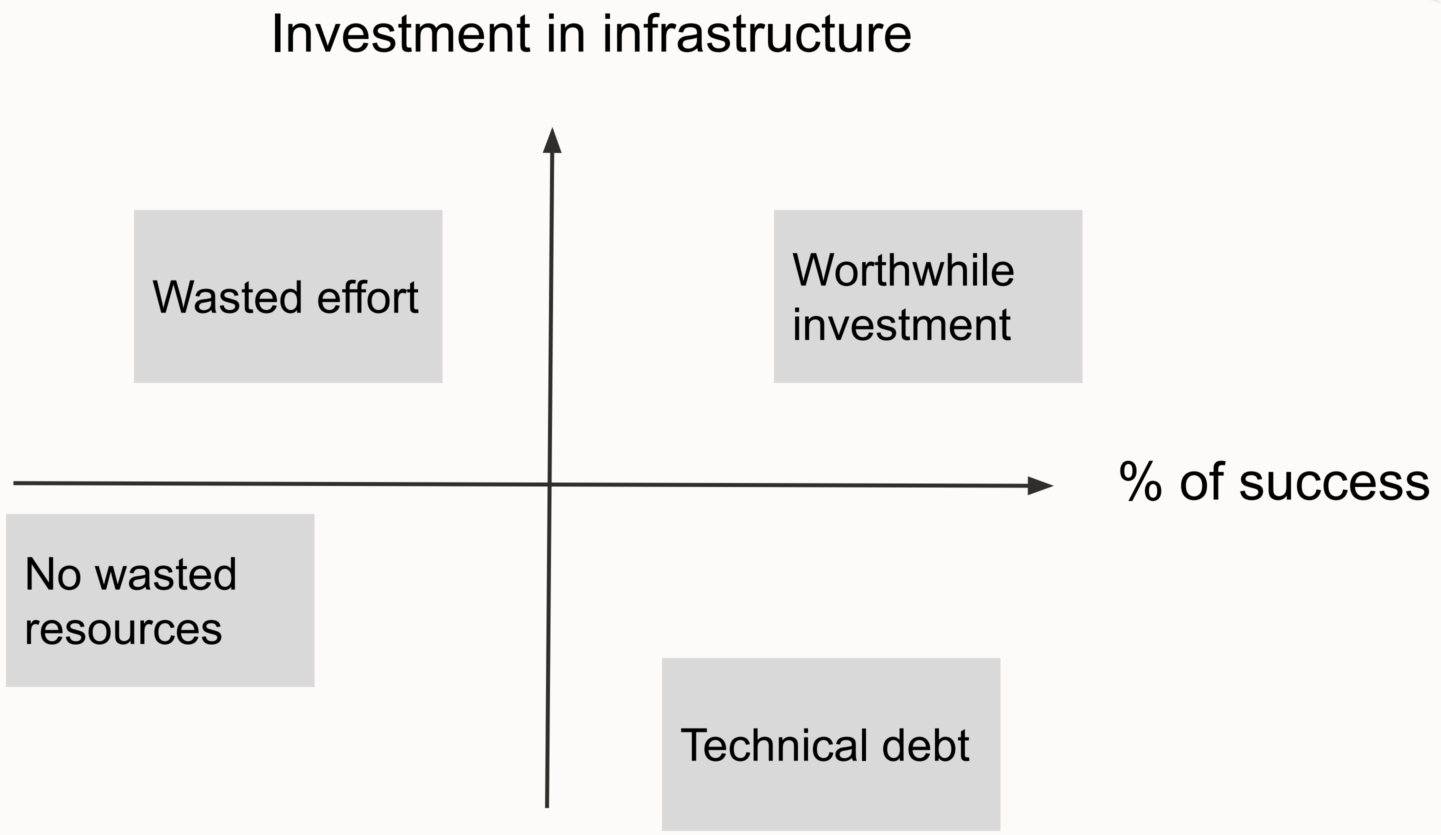
 Copyright © 2024, Oracle and/or its affiliates

What actually happens



Copyright © 2024, Oracle and/or its affiliates

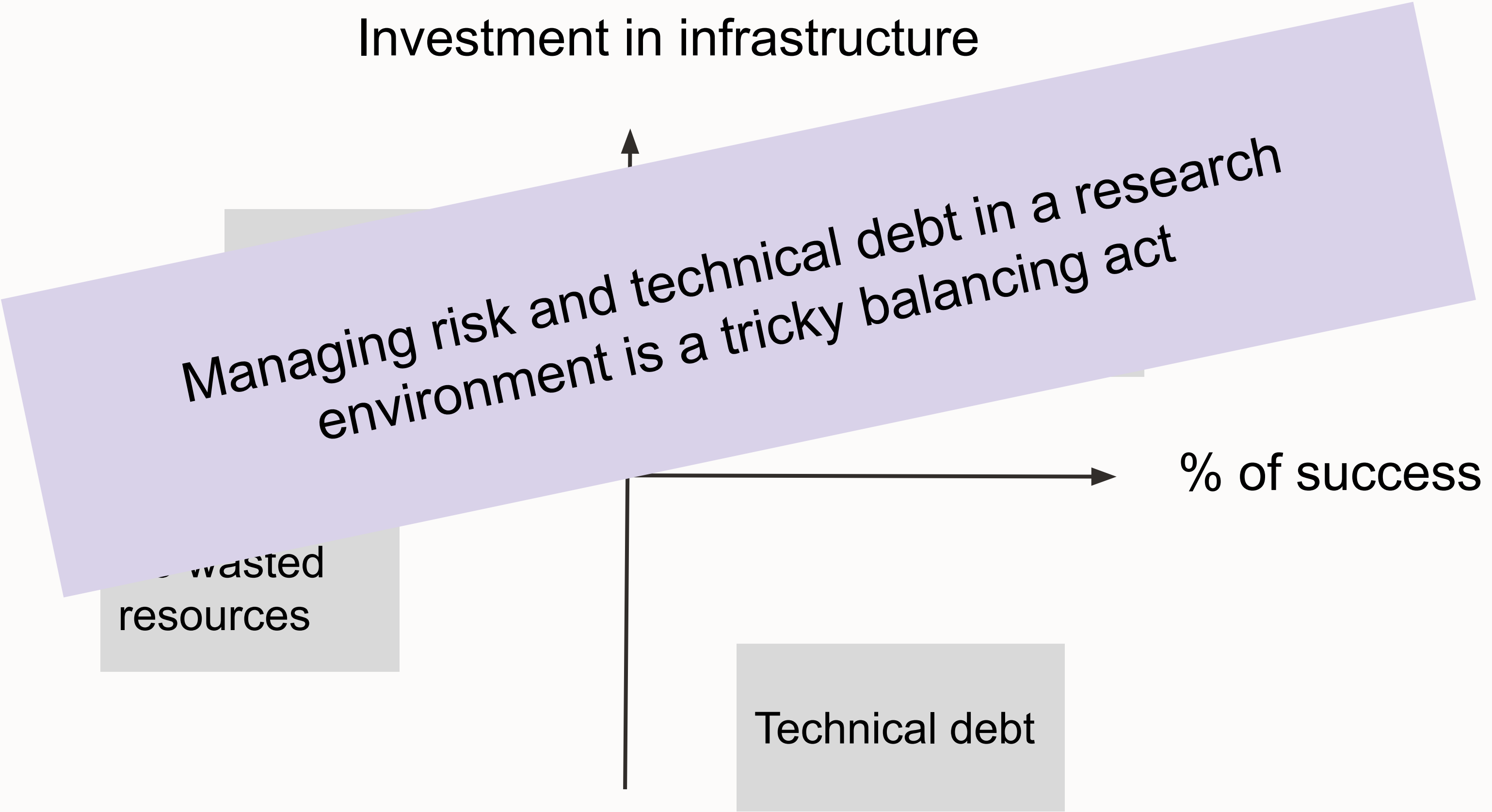
Problem space



“Trade-offs in managing risk and technical debt in industrial research labs: an experience report”, François Gauthier, Alexander Jordan, Padmanabhan Krishnan, Behnaz Hassanshahi, Jörn Guy Süß, Sora Bae, Hyunjun Lee, Proceedings of the 3rd International Conference on Technical Debt, 2020

Copyright © 2024, Oracle and/or its affiliates

Problem space



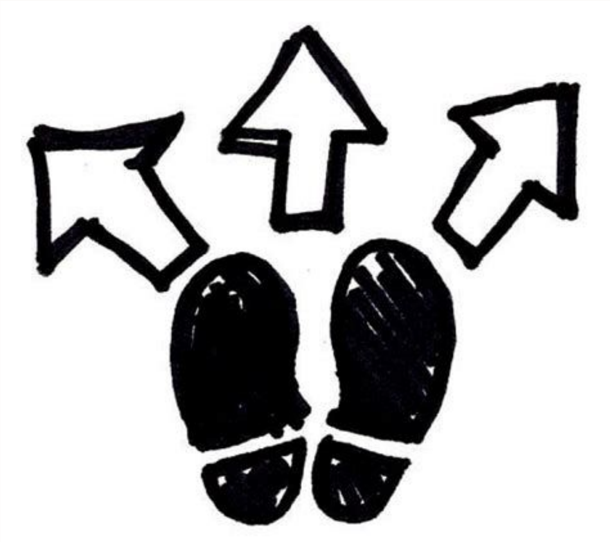
 Copyright © 2024, Oracle and/or its affiliates

Deployment at scale

Gelato as a service

Gelato as a plugin in ZAP and Burp Suite

How will support be provided?



Copyright © 2024, Oracle and/or its affiliates

What did help?

Having a variety of apps to test Gelato

Automated packaging system via GitLab CI

Gelato addressed a niche problem

 Copyright © 2024, Oracle and/or its affiliates

Summary

Even though many tools have been developed for web vulnerability detection over the past decade, existing tools still are not effective in practice

- Unacceptable coverage due to the complexity of JavaScript frameworks
- Highly event-driven UIs (manual crawling is out of question)
- Security analysis (e.g., DOM-XSS detection tools) ignore coverage

Gelato has been proven to outperform the existing crawlers and DOM-XSS detection tools

- We have found zero-days that the server-side fuzzer could not find without the crawler in Gelato

Copyright © 2024, Oracle and/or its affiliates

Remaining challenges and future work

The client-side crawler

- Needs to be even more efficient for nightly builds
- An incremental analysis would be nice
- Improving coverage of the client side
- Better support for frameworks

DOM-based XSS

- Needs to be more efficient, especially on complex frameworks
- Existing open-source instrumentation frameworks like Jalangi2 need maintenance
- Need to educate developers what they are and how to prevent them

Server-side fuzzing

- Instrumenting complex server-side code is very challenging in practice
- Use server-side feedback to improve the client-side crawler

 Copyright © 2024, Oracle and/or its affiliates

Back to the Optus incident

Could our tools detect the unprotected endpoint?

We cannot answer for sure of course, but that's the typical issue that Gelato could have found early if it was used in CI tests

BBC

Home News Sport Business Innovation Culture Travel Earth Video Live

Optus: How a massive data breach has exposed Australia

29 September 2022

Share

By Tiffanie Turnbull, BBC News, Sydney

Optus is the country's second-largest telecommunications company

Last week, Australian telecommunications giant Optus revealed about 10 million customers - about 40% of the population - had personal data stolen in what it calls a cyber-attack.

 Copyright © 2024, Oracle and/or its affiliates

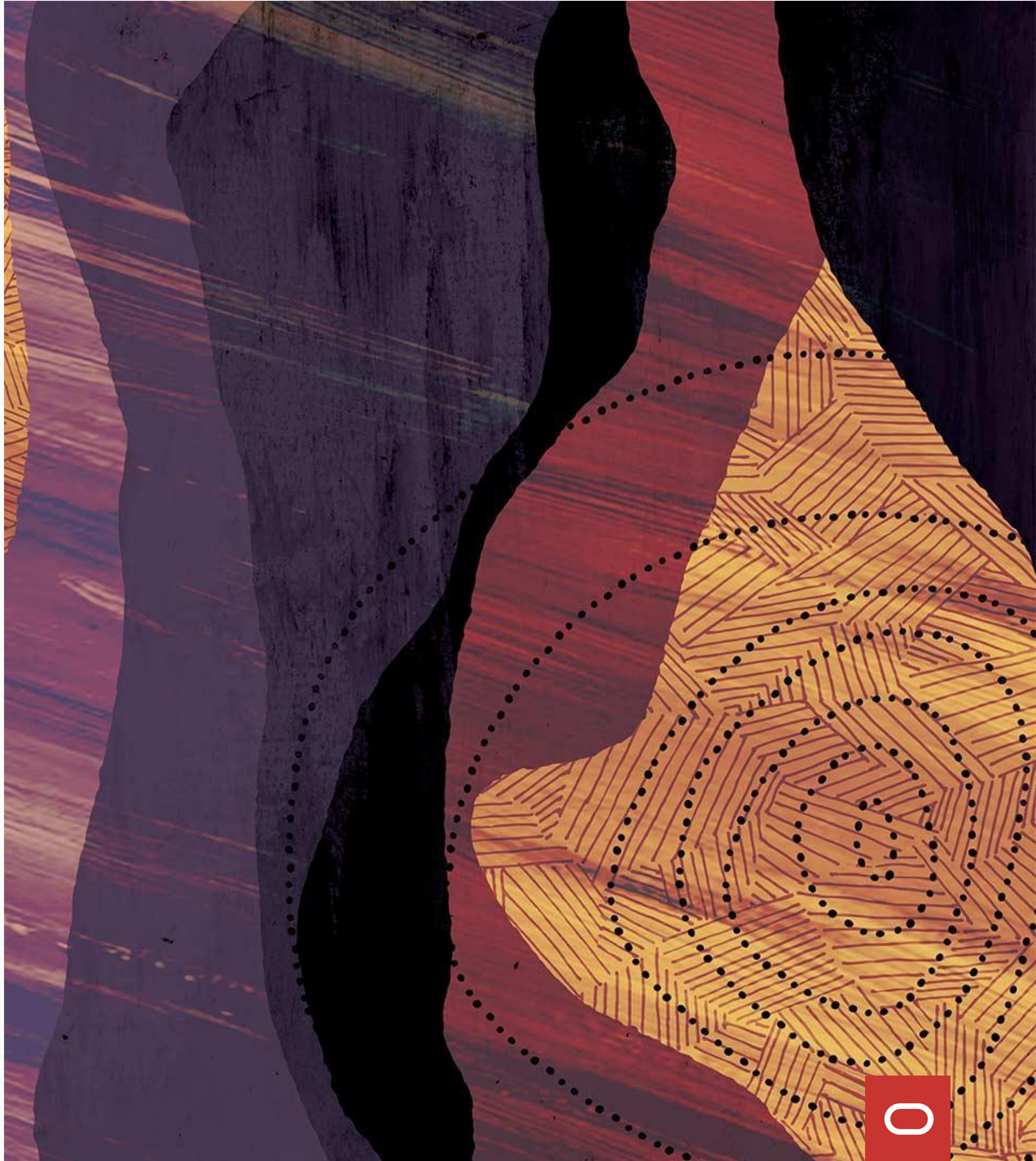
Thank you

Papers:

1. Gelato: Feedback-driven and Guided Security Analysis of Client-side Web Applications
2. BackREST: A Model-Based Feedback-Driven Greybox Fuzzer for Web Applications
3. Automatic Root Cause Quantification for Missing Edges in JavaScript Call Graphs
4. Trade-offs in managing risk and technical debt in industrial research labs: an experience report

Questions?

Email: behnaz.hassanshahi@oracle.com
Linkedin: https://www.linkedin.com/in/behnaz-hassanshahi-87962334/



 Copyright © 2021, Oracle and/or its affiliates