Server-side Web Application Fuzzing

I Putu Arya Dharmaadi
PhD Student
Information System Research Group
University of Groningen, Netherlands

Introduction: Why Web App Fuzzing?

- much less popular than research on binary fuzzing
  - Many spaces/opportunities can be explored!
- has its own challenges and many **techniques from binary fuzzing are not directly applicable**
  - web APIs only accept valid HTTP requests to be executed

Conduct a literature review to summarise existing works on web application fuzzing ▪

Fuzzing (Fuzz Testing)

- an automatic software testing intended to find vulnerabilities.
  - creating malformed/semi-malformed inputs,
  - injecting them into the software under test (SUT),
  - watching the software’s behaviour ⇒ a crash, fault, or hang
- Two recent approaches ⇒ **mutation-based and grammar-based input generation**.

Mutation-based Input Generation

- making small, random changes to the existing input
  - flipping some bits, inserting or deleting characters, etc
- Quality of the initial inputs **significantly influences** the fuzzing performance

In the web application context, a fuzzer can change:

- the HTTP request order in a sequence,
- HTTP request structure, or
- HTTP parameter values■

Grammar-based Input Generation

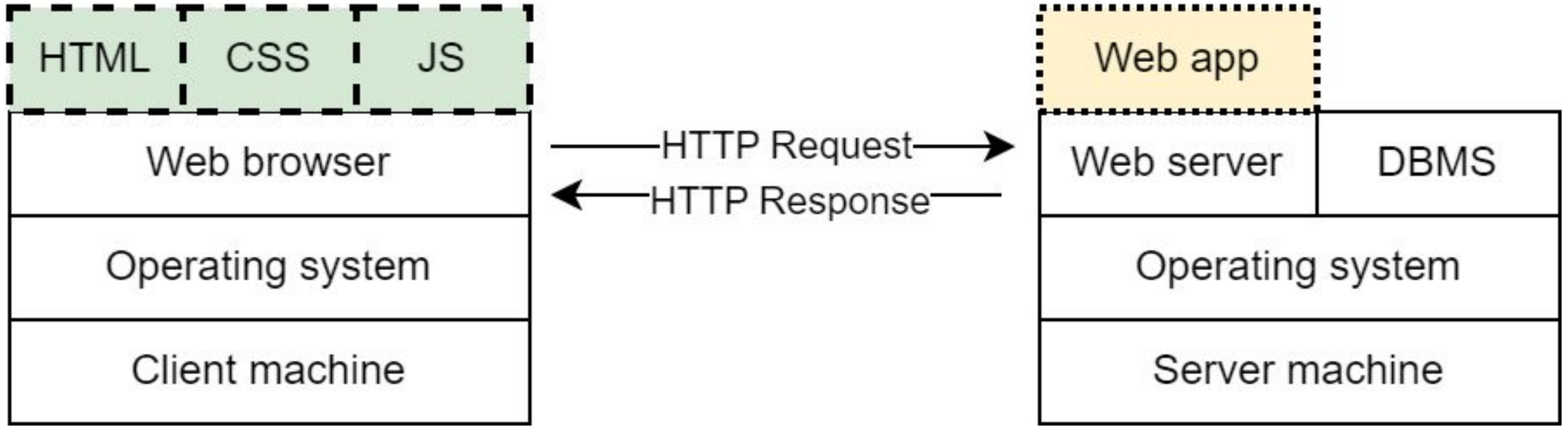
› employ a grammar ⇒ specifies the structure and input constraints

› can be called specification-based testing because it generates test cases based on the test specification (a.k.a grammar)

In the web application context:

a fuzzer ⇒ OpenAPI Specification ⇒ **grammar ⇒ valid HTTP requests**

Web App as the Web Under Test (WUT)



› Fuzzer **<u>targets server-side web applications</u>** (or we only use the term ”***web application***” for ease of reference)

› Fuzzer sends requests to the web application through the web API■

Web Fuzzing vs Other Fuzzing Types

Compared to binary fuzzing, Web Fuzzing **is not just an engineering issue!**

- Generating test cases for web APIs is a complex tasks.
  - Web server may reject malformed inputs from standard fuzzers
- Usually connected to database systems (it is becoming **a stateful system**!)
- One test case contains many HTTP calls
- Need to setup all the appropriate HTTP headers
- One HTTP call contains:
  - HTTP verb (e.g., POST or GET),
  - URL path parameters,
  - URL query parameters
  - HTTP body payloads.

Clarification (1): Web Fuzzer vs Vulnerability Scanner

- Fuzzer ⇒ produces plenty of malformed/semi-malformed inputs to **make a software crash** and let the software developers analyse any vulnerability (mostly 0-day vulnerability) behind the crash
- Scanner ⇒ scans and injects available APIs with malicious payload for **finding pre-defined vulnerabilities**, such as SQL injection and XSS

It is blurred because, recently, there have also been vulnerability-driven fuzzers■

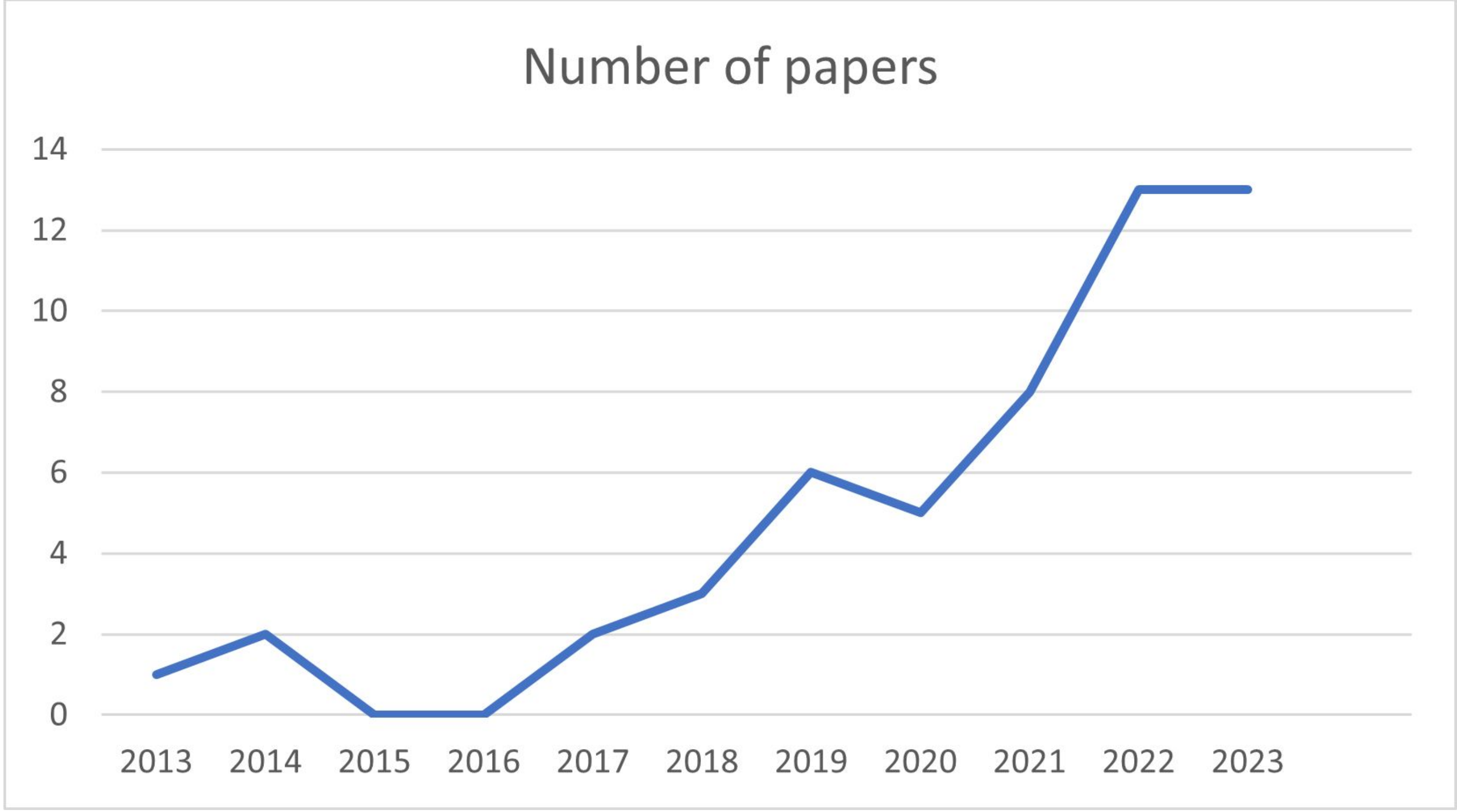
Clarification (2): Web Fuzzer vs Vulnerability Scanner

- Fuzzer ⇒ performs dynamic testing using either a **black-box, grey-box, or white-box** approach
- Dynamic Scanner ⇒ works from outside of the target (a.k.a **only uses the black-box** approach[1])■

[1]Reference: https://owasp.org/www-community/Vulnerability_Scanning_Tools

Paper Review

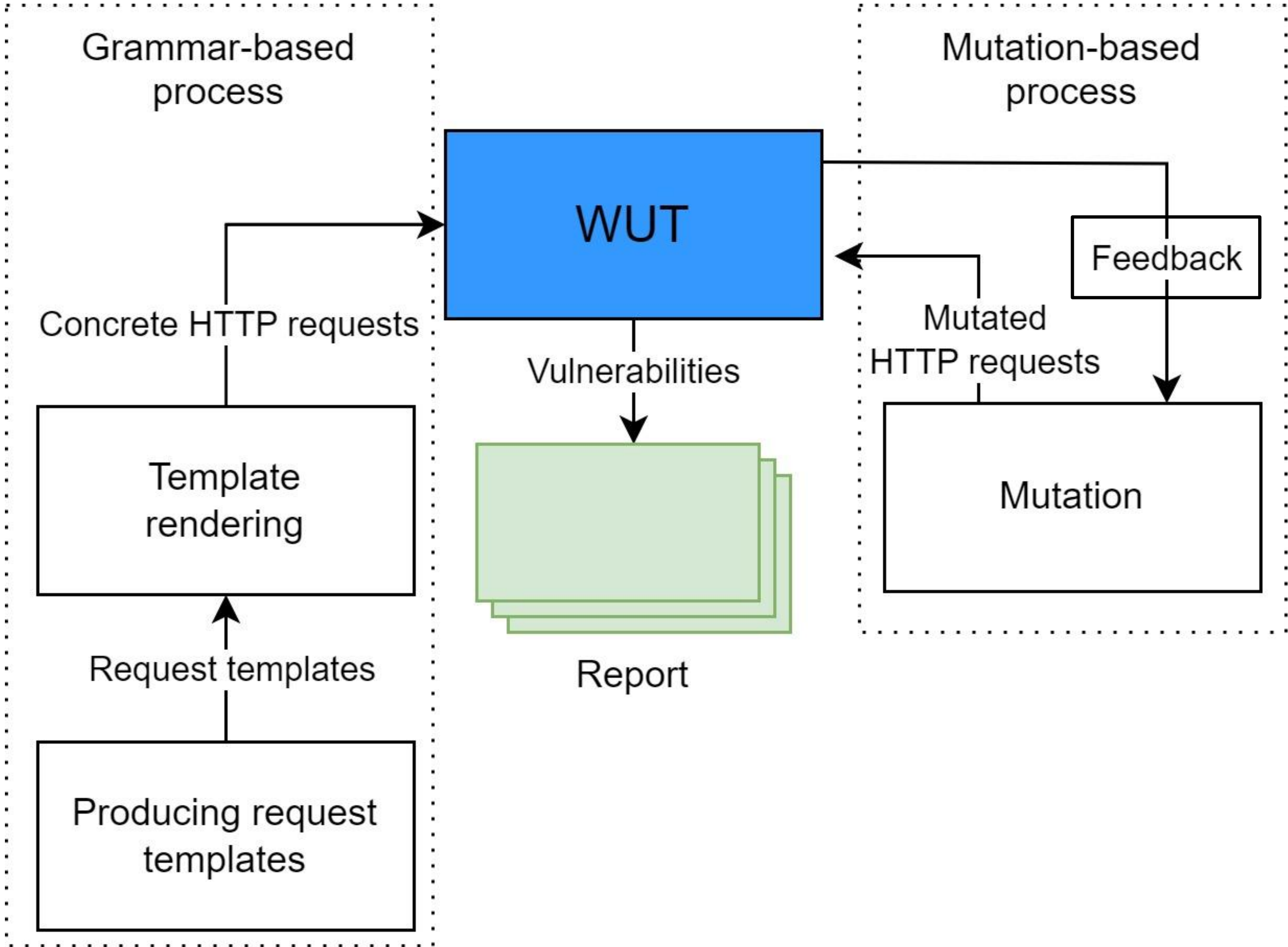
- **<u>review 53 articles</u>**, including 38 initially found by search and 15 new ones by performing the collection expansion step■



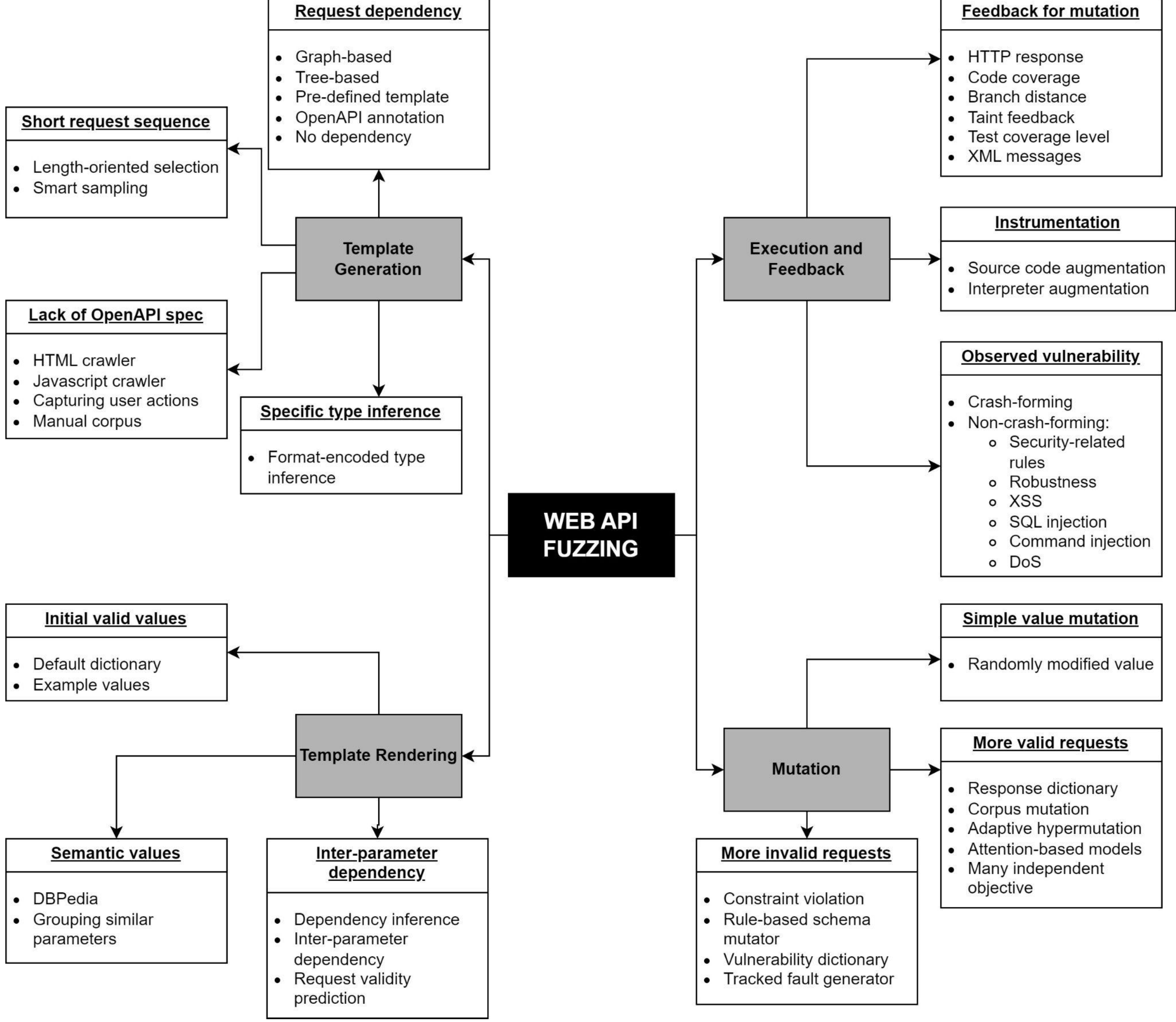
Review Focus

- Existing fuzzers that are specially designed for web application testing through web API.
  - generating valid HTTP requests,
  - utilising feedback from the WUTs, and
  - expanding relevant input space■

General Overview of Existing Web Application Fuzzing



Taxonomy of the problem-solution in prior studies



1. Template Generation

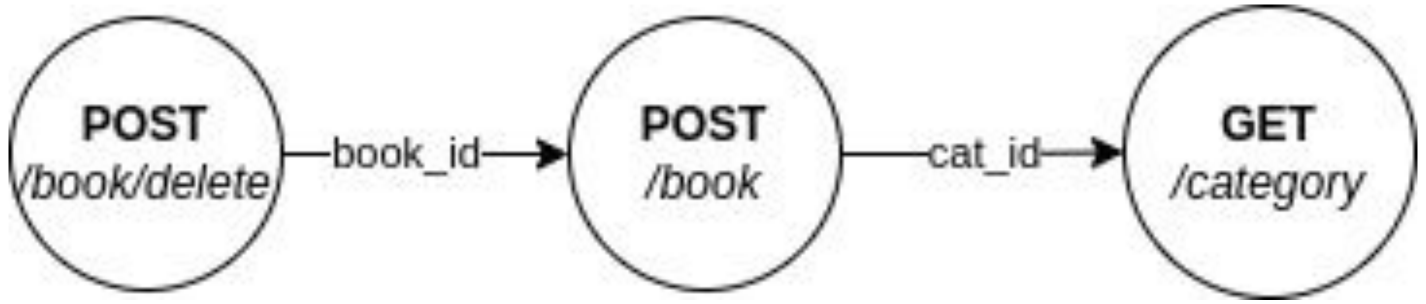
Problem: HTTP Request dependency ⇒ how to <u>**sort requests**</u> to create a valid order?

- If relationship among requests exist, a random technique will create wrong request order, leading to bad coverage in WUT.
- Incorrect order makes the WUT reject the request because the state is not satisfied

Existing Solution: Create Graph-based Dependency from OpenAPI doc

- If an output parameter of a HTTP request appears in an input parameter of another HTTP request, the former request must be executed first▪

Example of the Dependency



DELETE request requires ‘**book_id**’ values

BOOK request requires ‘**cat_id**’ values and produces ‘**book_id**’

2. Template Rendering

› **Problem: Semantic value** ⇒ How to fill in **<u>semantically valid values</u>** in the HTTP request templates?

- Only syntactically valid values (not semantically valid) may force certain WUT functions to reject the values, leading to execute always ‘else’ statements

  - **Existing Solution: Grouping similar parameters**

    The meaningful outputs from sent requests are stored and look-alike input parameter will use the output values ■

3. Execution and Feedback

- **Problem: Feedback for mutation** ⇒ What information is used **<u>to decide</u>** which existing request to be expanded/mutated?
  - Random picking or use-it-all makes the fuzzers to work inefficiently
    - **Existing Solution: Code coverage**

      The WUT is instrumented to show the execution path of the injected request. Requests leading to new execution paths are chosen▪

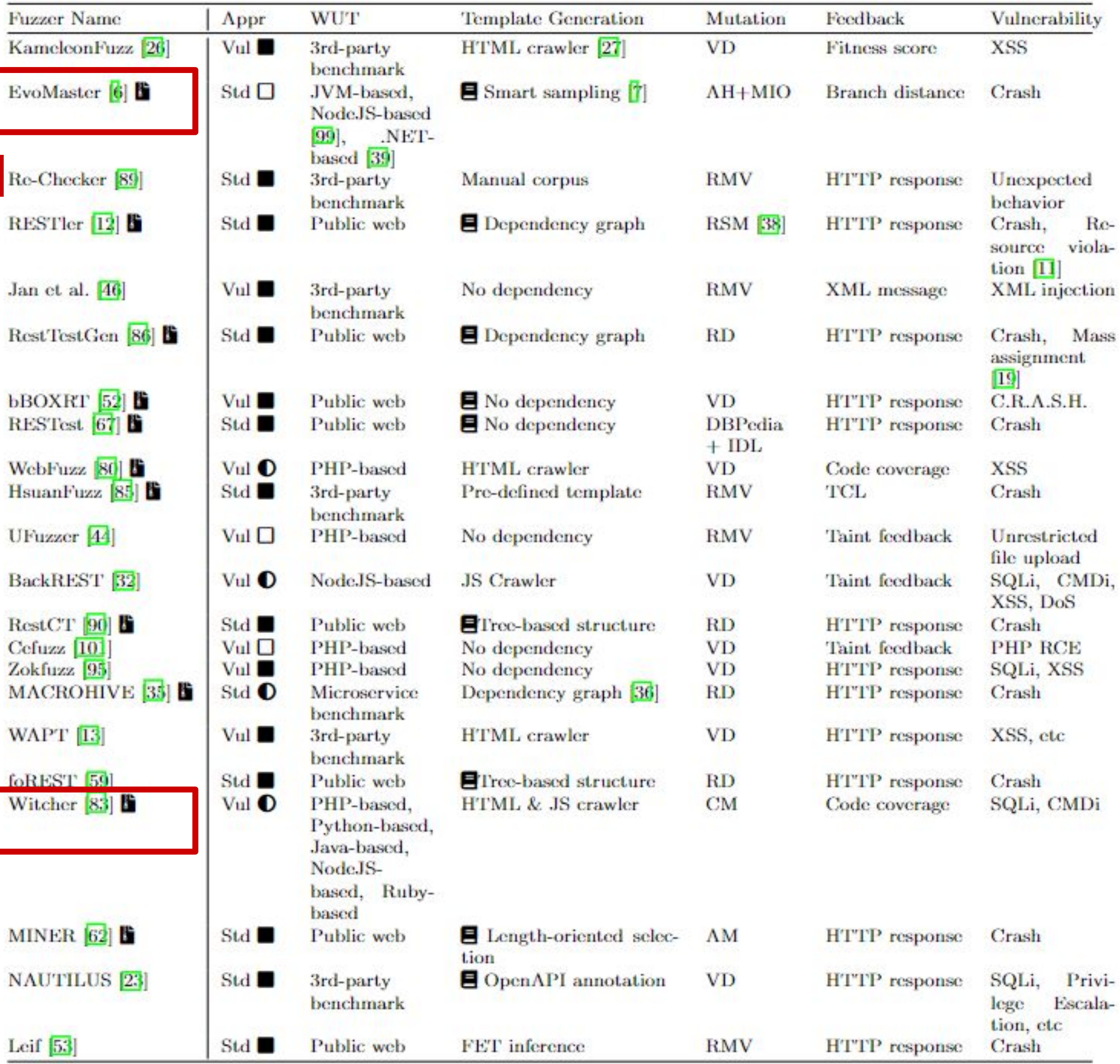
4. Mutation

- **Problem: More invalid requests** ⇒ After producing valid request sequences, how to create and insert invalid request **<u>to trigger a crash</u>**?
  - WUT may have already executed the vulnerable codes, but the crash/vulnerability does not show up because the crash requirement is not satisfied
    - **Existing Solution: Constraint violation**

      Make requests that violate the rule■

university of groningen

Existing Web API fuzzer



Fuzzer Name	Appr	WUT	Template Generation	Mutation	Feedback	Vulnerability
KameleonFuzz [26]	Vul ■	3rd-party benchmark	HTML crawler [27]	VD	Fitness score	XSS
EvoMaster [6] 🗎	Std □	JVM-based, NodeJS-based [99], .NET-based [39]	▤ Smart sampling [7]	AH+MIO	Branch distance	Crash
Re-Checker [89]	Std ■	3rd-party benchmark	Manual corpus	RMV	HTTP response	Unexpected behavior
RESTler [12] 🗎	Std ■	Public web	▤ Dependency graph	RSM [38]	HTTP response	Crash, Resource violation [11]
Jan et al. [46]	Vul ■	3rd-party benchmark	No dependency	RMV	XML message	XML injection
RestTestGen [86] 🗎	Std ■	Public web	▤ Dependency graph	RD	HTTP response	Crash, Mass assignment [19]
bBOXRT [52] 🗎	Vul ■	Public web	▤ No dependency	VD	HTTP response	C.R.A.S.H.
RESTest [67] 🗎	Std ■	Public web	▤ No dependency	DBPedia + IDL	HTTP response	Crash
WebFuzz [80] 🗎	Vul ◐	PHP-based	HTML crawler	VD	Code coverage	XSS
HsuanFuzz [85] 🗎	Std ■	3rd-party benchmark	Pre-defined template	RMV	TCL	Crash
UFuzzer [44]	Vul □	PHP-based	No dependency	RMV	Taint feedback	Unrestricted file upload
BackREST [32]	Vul ◐	NodeJS-based	JS Crawler	VD	Taint feedback	SQLi, CMDi, XSS, DoS
RestCT [90] 🗎	Std ■	Public web	▤Tree-based structure	RD	HTTP response	Crash
Cefuzz [101]	Vul □	PHP-based	No dependency	VD	Taint feedback	PHP RCE
Zokfuzz [95]	Vul ■	PHP-based	No dependency	VD	HTTP response	SQLi, XSS
MACROHIVE [35] 🗎	Std ◐	Microservice benchmark	Dependency graph [36]	RD	HTTP response	Crash
WAPT [13]	Vul ■	3rd-party benchmark	HTML crawler	VD	HTTP response	XSS, etc
foREST [59]	Std ■	Public web	▤Tree-based structure	RD	HTTP response	Crash
Witcher [83] 🗎	Vul ◐	PHP-based, Python-based, Java-based, NodeJS-based, Ruby-based	HTML & JS crawler	CM	Code coverage	SQLi, CMDi
MINER [62] 🗎	Std ■	Public web	▤ Length-oriented selection	AM	HTTP response	Crash
NAUTILUS [23]	Std ■	3rd-party benchmark	▤ OpenAPI annotation	VD	HTTP response	SQLi, Privilege Escalation, etc
Leif [53]	Std ■	Public web	FET inference	RMV	HTTP response	Crash

Open Challenges

- Ineffectiveness of Instrumentation
  - The instrumentation makes the source code bigger and slows down the server execution
  - When the available testing time is very short, instrumentation processes (compile and build processes) may **take longer** than the fuzzing campaign itself
- Complexity of Handling Microservice Architecture
  - A fuzzer may only test the main service without considering such **complex connections** behind it■

Open Challenges (2)

- The Difficulty of Testing Public WUT
  - Majority of those popular web applications are closed-source, users can only test them using a black-box web API fuzzer, leading to **code coverage ignorance**.

Open Challenges (3)

- Low-quality Corpus
  - Most studies rely on OpenAPI Documents or HTML pages to create the initial corpus; **it is insufficient** to produce highly diverse yet correct corpus
- Lack of Web API Fuzzing Benchmarks
  - no widely accepted benchmark for web API fuzzers and no established **security-related metrics** in comparing the performances of the fuzzers■

Potential Research Directions

› Fuzzing for **Web Client Programming**

  - Web client programming as one of the solutions to distribute the server load due to the increasingly sophisticated services provided

› Fuzzing for **Mobile Web**

  - Mobile web does not show complex user interfaces to make users enjoy the web on a smaller screen

› **Generative AI** for Web Testing

  - AI learn the patterns and structures present in the training data to produce highly diverse test cases■

Potential Research Directions (2)

- Support for **Diverse Web Security Vulnerabilities**
  - Extend the OWASP vulnerability list
  - Potential faults that may not happen yet massively ▪

Conclusion

We reviewed 53 articles working on web API fuzzing frameworks and classified the prior works based on their testing objectives and techniques used to generate HTTP requests, utilise feedback from the WUT, and mutate existing requests.

Let’s explore this topic together!

Reference

Dharmaadi, I. P., Athanasopoulos, E., & Turkmen, F. (2024). Fuzzing Frameworks for Server-side Web Applications: A Survey. *ArXiv*. /abs/2406.03208