Fuzzing for Security and Resilience

Gilles Coremans

gilles.coremans@vub.be

Test cases are great!

Quick to find bugs or regressions

Writing tests improves understanding of code

… but limited

Production deployments always have more cases!

Writing good tests takes a lot of time

A different approach: Fuzzing

What if we just tried every input?

Fuzzing tests programs by sending random inputs

Unlike other automatic testing techniques: no false positives!

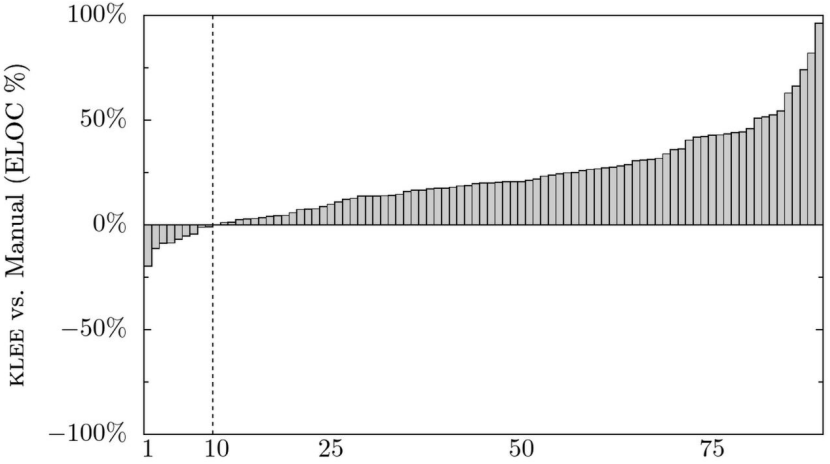
Fuzzing can outperform test suites

Fuzzing outperforms manual test suites on GNU Coreutils

Fuzzing found new crashes in GNU Coreutils

However: fuzzing campaigns can be slow

Fuzzing requires test oracles to distinguish bugs




```
paste -d\\ abcdefghijklmnopqrstuvwxyz
pr -e t2.txt
tac -r t3.txt t3.txt
mkdir -Z a b
mkfifo -Z a b
mknod -Z a b p
md5sum -c t1.txt
ptx -F\\ abcdefghijklmnopqrstuvwxyz
ptx x t4.txt
seq -f %0 1
```

t1.txt: `"\t \tMD5("`
t2.txt: `"\b\b\b\b\b\b\t"`
t3.txt: `"\n"`
t4.txt: `"a"`

Testing Oracles

Fuzzing for security has “easy” oracles:

- Segmentation faults
- AdressSanitizer
- etc

For other bug types we have to get creative

- Check violations of expected properties
- Check for unexpected errors
- Check execution time/latency

Speeding up fuzzing

Grammar-based Fuzzing
Generate structured inputs that conform to what the program expects

Feedback-driven Fuzzing
Determine which inputs are interesting, then mutate those

Feedback-driven fuzzing

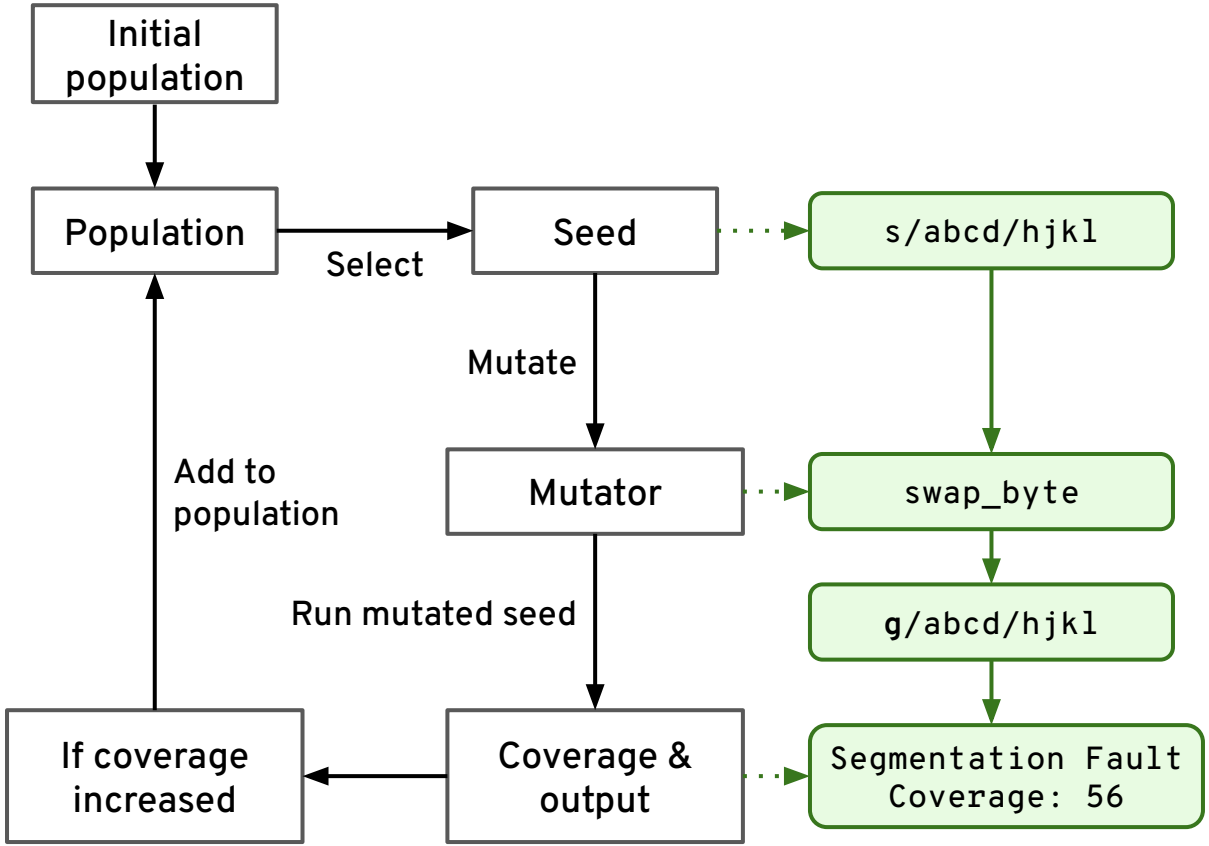
Coverage tells us whether an input is interesting

Mutating interesting inputs is yields more interesting inputs

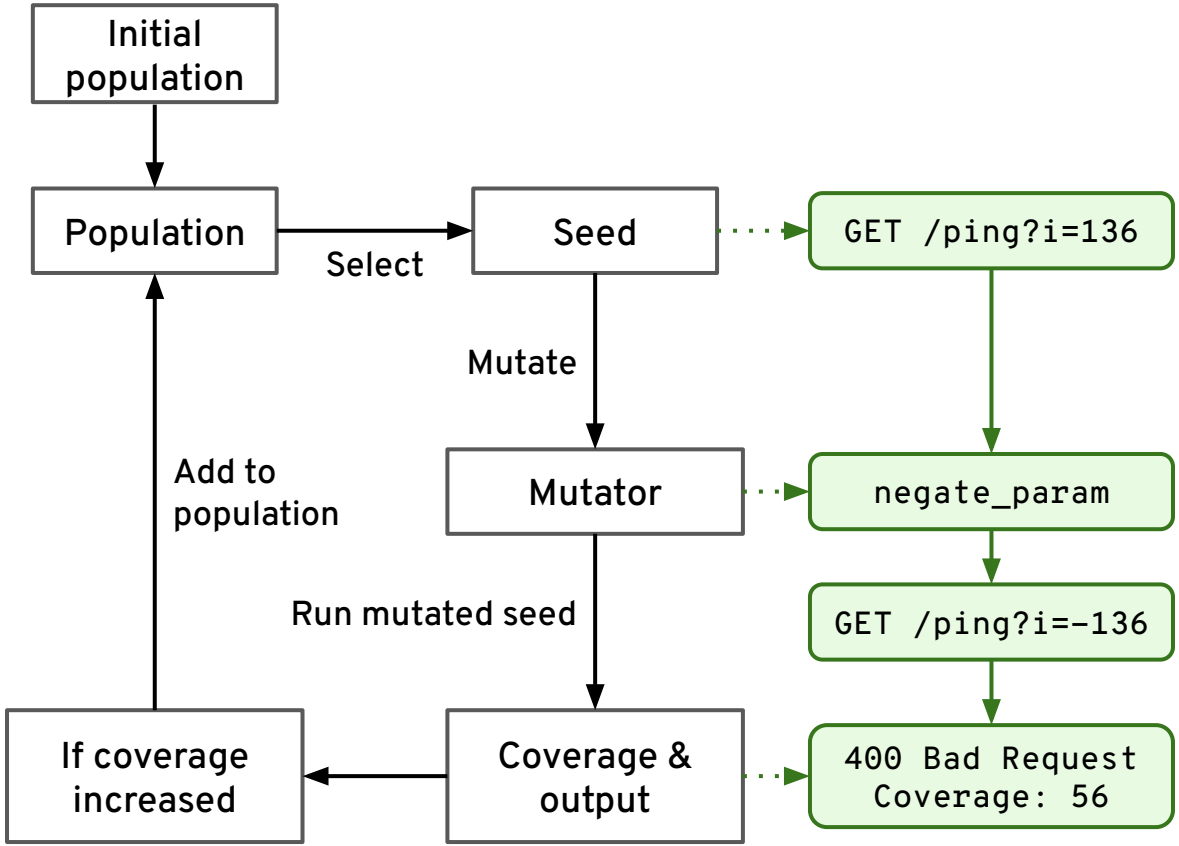
Many vulnerabilities found this way

IJG jpeg [1]	libjpeg-turbo [1 2]	libpng [1]
libtiff [1 2 3 4 5]	mozjpeg [1]	PHP [1 2 3 4 5 6 7 8]
Mozilla Firefox [1 2 3 4]	Internet Explorer [1 2 3 4]	Apple Safari [1]
Adobe Flash / PCRE [1 2 3 4 5 6 7]	sqlite [1 2 3 4...]	OpenSSL [1 2 3 4 5 6 7]
LibreOffice [1 2 3 4]	poppler [1 2...]	freetype [1 2]
GnuTLS [1]	GnuPG [1 2 3 4]	OpenSSH [1 2 3 4 5]
PuTTY [1 2]	ntpd [1 2]	nginx [1 2 3]
bash (post-Shellshock) [1 2]	tcpdump [1 2 3 4 5 6 7 8 9]	JavaScriptCore [1 2 3 4]
pdfium [1 2]	ffmpeg [1 2 3 4 5]	libmatroska [1]
libarchive [1 2 3 4 5 6 ...]	wireshark [1 2 3]	ImageMagick [1 2 3 4 5 6 7 8 9 ...]
BIND [1 2 3 ...]	QEMU [1 2]	lcms [1]
Oracle BerkeleyDB [1 2]	Android / libstagefright [1 2]	iOS / ImageIO [1]
FLAC audio library [1 2]	libsndfile [1 2 3 4]	less / lesspipe [1 2 3]
strings (+ related tools) [1 2 3 4 5 6 7]	file [1 2 3 4]	dpkg [1 2]
rcs [1]	systemd-resolved [1 2]	libyaml [1]
Info-Zip unzip [1 2]	libtasn1 [1 2 ...]	OpenBSD pfctl [1]
NetBSD bpf [1]	man & mandoc [1 2 3 4 5 ...]	IDA Pro [reported by authors]
clamav [1 2 3 4 5 6]	libxml2 [1 2 4 5 6 7 8 9 ...]	glibc [1]
clang / llvm [1 2 3 4 5 6 7 8 ...]	nasm [1 2]	ctags [1]
mutt [1]	procmail [1]	fontconfig [1]
pdksh [1 2]	Qt [1 2...]	wavpack [1 2 3 4]
redis / lua-cmsgpack [1]	taglib [1 2 3]	privoxy [1 2 3]
perl [1 2 3 4 5 6 7...]	libxmp	radare2 [1 2]
SleuthKit [1]	fwknop [reported by author]	X.Org [1 2]
exifprobe [1]	jhead [?]	capnproto [1]
Xerces-C [1 2 3]	metacam [1]	djvulibre [1]
exiv [1 2]	Linux btrfs [1 2 3 4 6 7 8]	Knot DNS [1]
curl [1 2 3]	wpa_supplicant [1]	libde265 [reported by author]
dnsmasq [1]	libbpg [1]	lame [1 2 3 4 5 6]
libwmf [1]	uudecode [1]	MuPDF [1 2 3 4]
imlib2 [1 2 3 4]	libraw [1]	libbson [1]
libsass [1]	yara [1 2 3 4]	W3C tidy-html5 [1]
VLC [1 2]	FreeBSD syscons [1 2 3]	John the Ripper [1 2]
screen [1 2 3]	tmux [1 2]	mosh [1]
UPX [1]	indent [1]	openjpeg [1 2]
MMIX [1]	OpenMPT [1 2]	rxvt [1 2]
dhcpcd [1]	Mozilla NSS [1]	Nettle [1]
mbed TLS [1]	Linux netlink [1]	Linux ext4 [1]
Linux xfs [1]	botan [1]	expat [1 2]
Adobe Reader [1]	libav [1]	libical [1]

The fuzzing loop for a command line tool



The fuzzing loop for a REST API

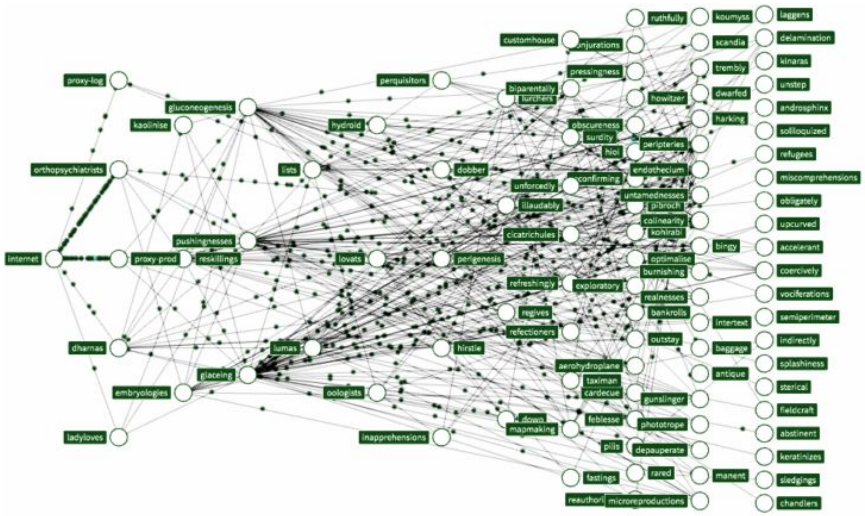


Our research: fuzzing for micro-service backends

Backend split into many small services which communicate

Used for large, scalable, constantly evolving backends

Developers need to consider fault tolerance



Fuzzing for resilience defects

Resilience defects aren’t triggered on developer machines or in testing

With fuzzing, we can find these bugs automatically in testing environments

Resilience defects

```java
@PostMapping("/pong")
public ResponseEntity<Pong> pongPost(RestTemplate rest, @RequestBody Increment i) {
    long inc = i.increment();
    if(inc >= 0 && inc < 10000) {
        long cur = counter.addAndGet(inc);
        return ResponseEntity.ok().body(new Pong(cur));
    } else {
        return ResponseEntity.badRequest().build();
    }
}
```

200 OK if in range

400 Bad Request if out of range

```java
@PostMapping("/ping")
public ResponseEntity<Ping> pingPost(RestTemplate rest, @RequestBody Increment i) {
    long cur;
    ResponseEntity<Pong> ponge;
    try {
        ponge = rest.postForEntity("http://pong/pong", i, Pong.class);
        cur = counter.addAndGet(i.increment());
    } catch(HttpClientErrorException e) {
        logger.warn(e);
        return ResponseEntity.badRequest().build();
    } catch(RestClientException e) {
        // Retry
        logger.error(e);
        ponge = rest.postForEntity("http://pong/pong", i, Pong.class);
        cur = counter.addAndGet(i.increment());
    }
    Pong pong = ponge.getBody();
    return ResponseEntity.ok(new Ping(cur, pong.id()));
}
```

Check and handle 400 response

Check for connection error and retry

Fails on two errors
Does not handle 400 response

Can't be found without fault injection!

Resilience defects

```java
@PostMapping("/pong")
public ResponseEntity<Pong> pongPost(RestTemplate rest, @RequestBody Increment i) {
    long inc = i.increment();
    if(inc >= 0 && inc < 10000) {
        long cur = counter.addAndGet(inc);
        return ResponseEntity.ok().body(new Pong(cur));
    } else {
        return ResponseEntity.badRequest().build();
    }
}
```

200 OK if in range

400 Bad Request if out of range

```java
@PostMapping
public Respo
    long cur
    Response
    try {
        pong
        cur
    } catch(
        logg
        retu
    } catch(RestClientException e) {
        // Retry
        logger.error(e);
        ponge = rest.postForEntity("http://pong/pong", i, Pong.class);
        cur = counter.addAndGet(i.increment());
    }
    Pong pong = ponge.getBody();
    return ResponseEntity.ok(new Ping(cur, pong.id()));
}
```

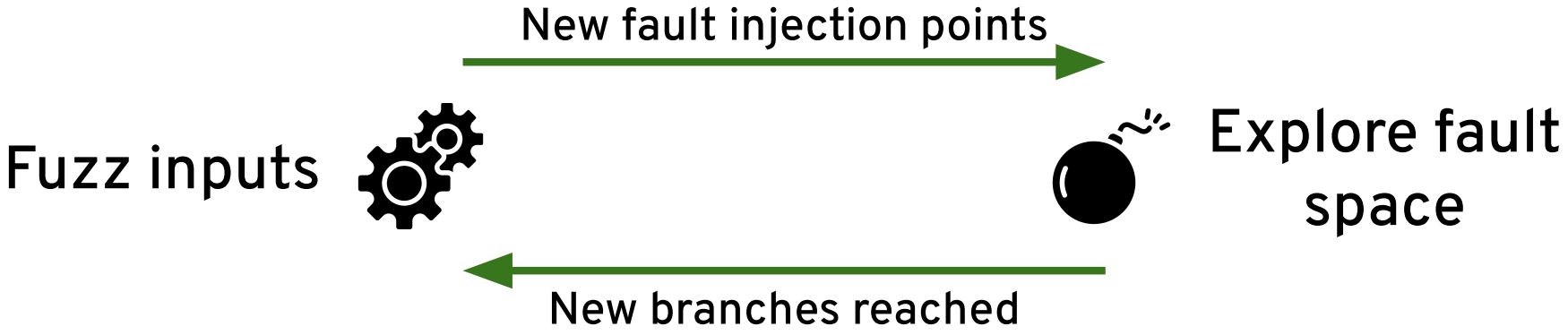
1. These bugs won't be found without fault injection
2. Some bugs require multiple faults
3. Some bugs require specific inputs **and** faults

Check for connection error and retry

Fails on two errors
Does not handle 400 response

Can't be found without fault injection!

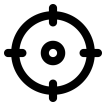
Faults and inputs



We can use fuzzing to explore faults as well!

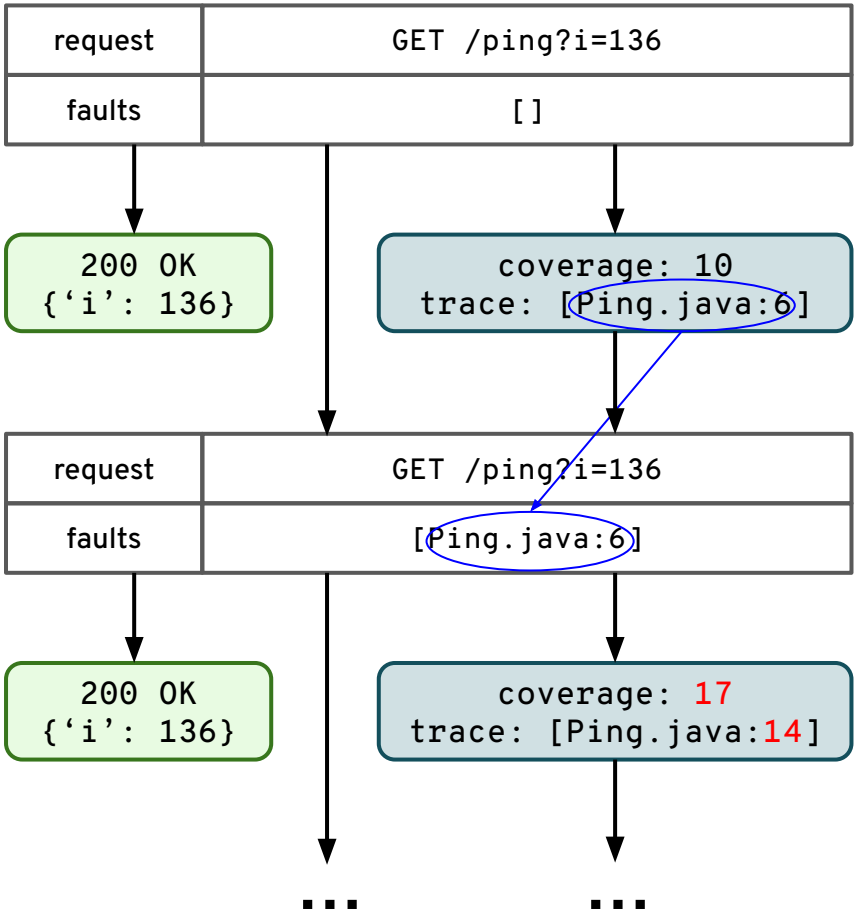
Fault injection

Faults are part of the seed along with request



Fault injection points are found during execution

Faults added to seed using mutators

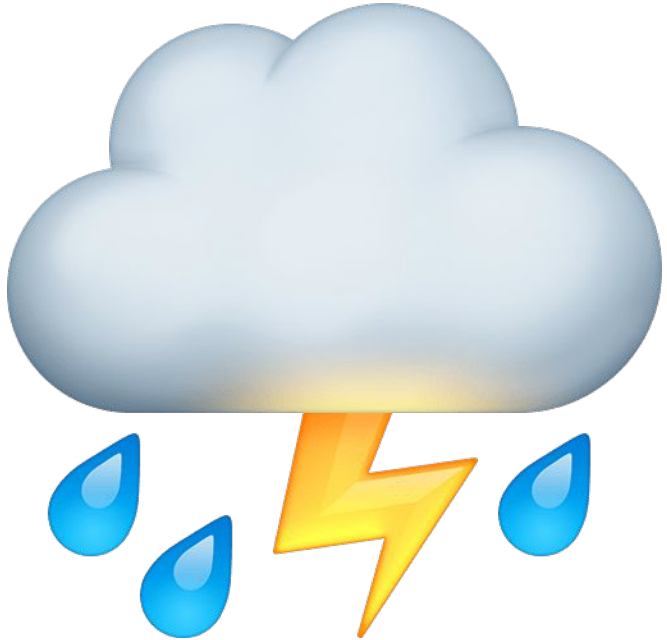


Onweer

Automated resilience tester based on these concepts

Faults are collected and injected by an instrumentation agent

Faults are connection error exceptions thrown during REST requests



Evaluation

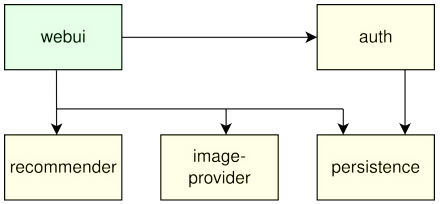
? RQ1: Does fault injection increase coverage?

? RQ2: Does fault injection cover code fuzzing alone can't reach?

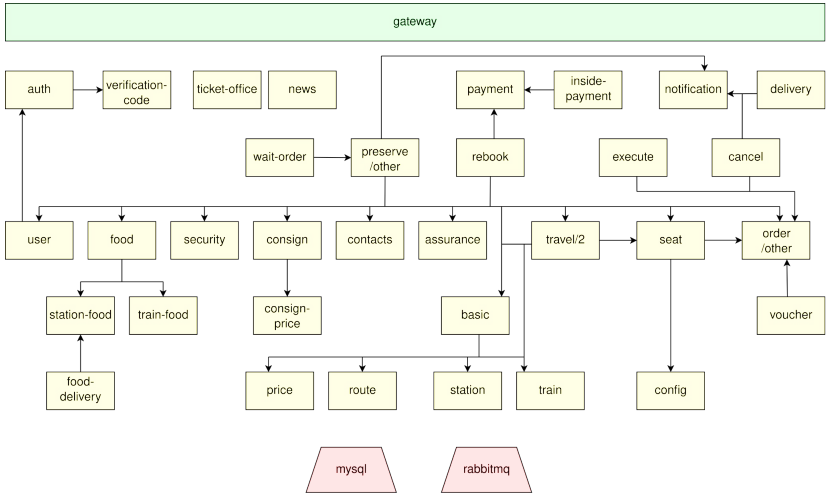
? RQ3: Does adding fault injection find resilience defects?

! Method: Compare Onweer with and without fault injection and RESTler

TeaStore [von Kistowski et al. 2018]



TrainTicket [Zhou et al. 2018]



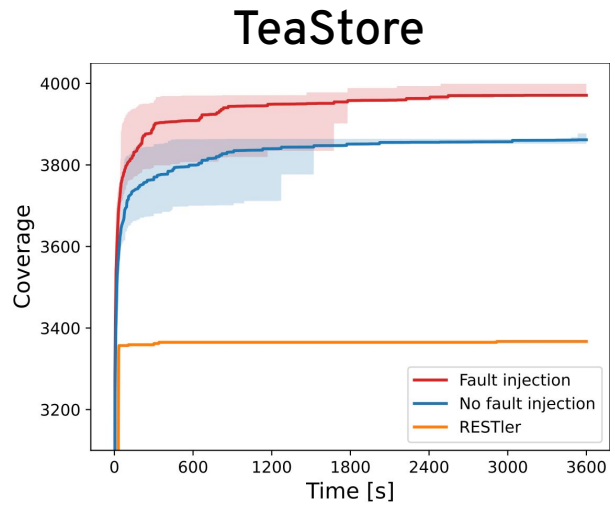
RQ1: Code coverage

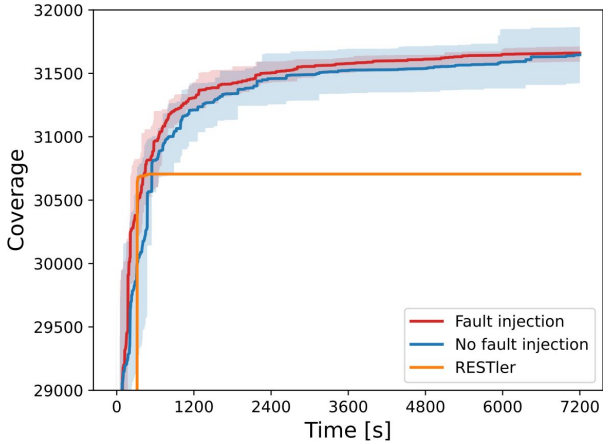
Coverage improves for TeaStore!

But not for TrainTicket?

RQ2: Because all coverage increase is in fault handlers

TrainTicket has no fault handling

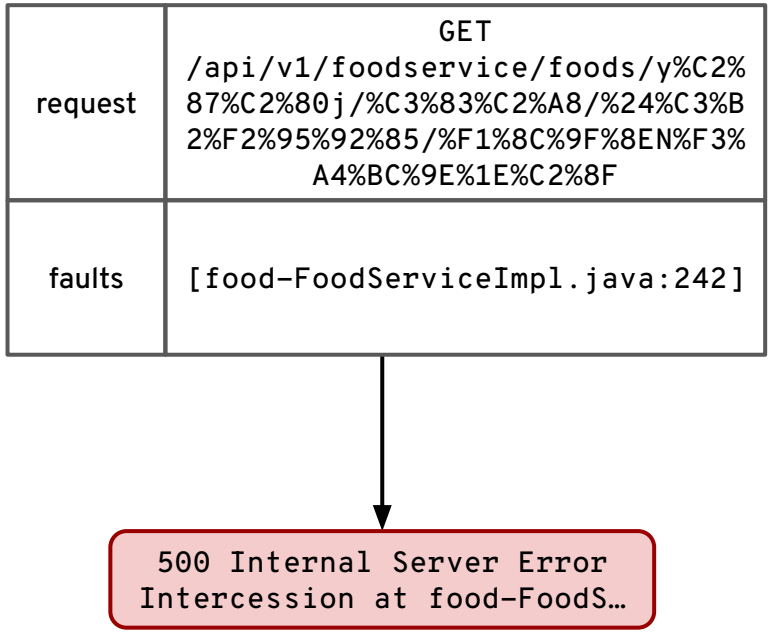




RQ3: Finding resilience defects

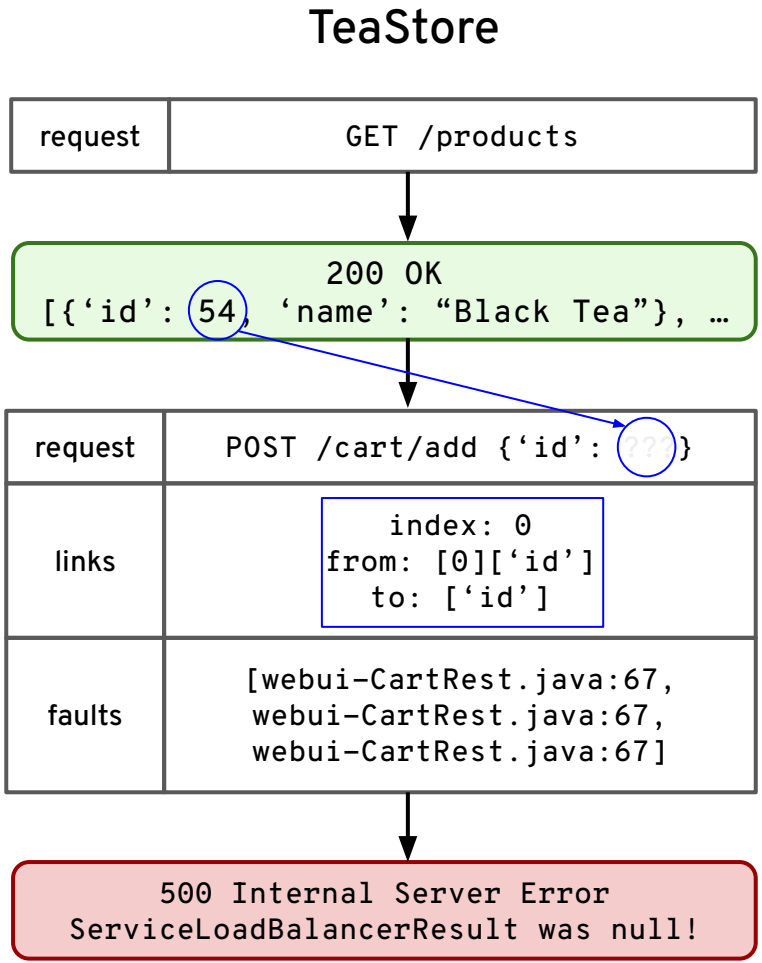
TrainTicket

We find several resilience defects in both projects!



RQ3: Finding resilience defects

We find several resilience defects in both projects!



Conclusion

- Are you applying fuzzing, chaos engineering, resilience testing?
  - We would love to hear about your experiences and challenges
  - We’re eager to put our tools to the test on your code
- We can help:
  - Test and secure your devices using fuzzing
  - Test and secure your backends using fuzzing
  - Tailor fault injection to the specifics of your application
  - Develop bespoke fuzzers
- Within individual and collaborative research and development projects (O&O, ICON, …)