



code intelligence

Coverage-Guided Fuzzing for Web  
Applications

Khaled Yakdan  
CTO & Co-Founder

# Why?



Most critical code  
in web services

Enormous security  
impact

Critical OWASP  
vulnerabilities

# Typical Challenges of Web Fuzzing in a Single Piece of Code

The diagram illustrates the flow of structured inputs through a Java code snippet. A central teal box labeled "Structured Inputs" has arrows pointing to several specific parts of the code:

- An arrow points from the "Structured Inputs" box to the `@RestController` annotation at the top of the class.
- An arrow points from the "Structured Inputs" box to the `@PostMapping("/challenge/5")` annotation.
- An arrow points from the "Structured Inputs" box to the `@RequestParam String username_login` parameter in the `login` method.
- An arrow points from the "Structured Inputs" box to the `@RequestParam String password_login` parameter in the `login` method.

```
@RestController
@Slf4j
public class Assignment5 extends AssignmentEndpoint {
    ...
    @PostMapping("/challenge/5")
    @ResponseBody
    public AttackResult login(@RequestParam String username_login, @RequestParam String password_login) {
        ...
        if (!"Larry".equals(username_login)) {
            return failed(this).feedback("user.not.larry").feedbackArgs(username_login).build();
        }
        ...
        PreparedStatement stmt = connection.prepareStatement("select password from users where userid = '" +
username_login + "' and password = '" + password_login + "'");
        ResultSet resultSet = statement.executeQuery();
        ...
    }
}
```

# Typical Challenges of Web Fuzzing in a Single Piece of Code

```
@RestController
@Slf4j
public class Assignment5 extends AssignmentEndpoint {
    //...
    @PostMapping("/challenge/5")
    @ResponseBody
    public AttackResult login(@RequestParam String username_login, @RequestParam String password_login) {
        //...
        if (!"Larry".equals(username_login)) {
            return failed(this).feedback("user.not.larry").feedbackArgs(username_login).build();
        }
        //...
        PreparedStatement stmt = connection.prepareStatement("select password from users where userid = '" +
username_login + "' and password = '" + password_login + "'");
        ResultSet resultSet = statement.executeQuery();
        //...
    }
}
```

Bug Detectors for  
OWASP Vulnerabilities

# Typical Challenges of Web Fuzzing in a Single Piece of Code

```
@RestController  
@Slf4j  
public class Assignment5 extends AssignmentEndpoint {  
    //...  
    @PostMapping("/challenge/5")  
    @ResponseBody  
    public AttackResult login(@RequestParam String username_login, @RequestParam String password_login) {  
        //...  
        if (!"Larry".equals(username_login)) {  
            return failed(this).feedback("user.not.larry").feedbackArgs(username_login).build();  
        }  
        //...  
        PreparedStatement stmt = connection.prepareStatement("select password from users where userid = '" +  
username_login + "' and password = '" + password_login + "'");  
        ResultSet resultSet = statement.executeQuery();  
        //...  
    }  
}
```

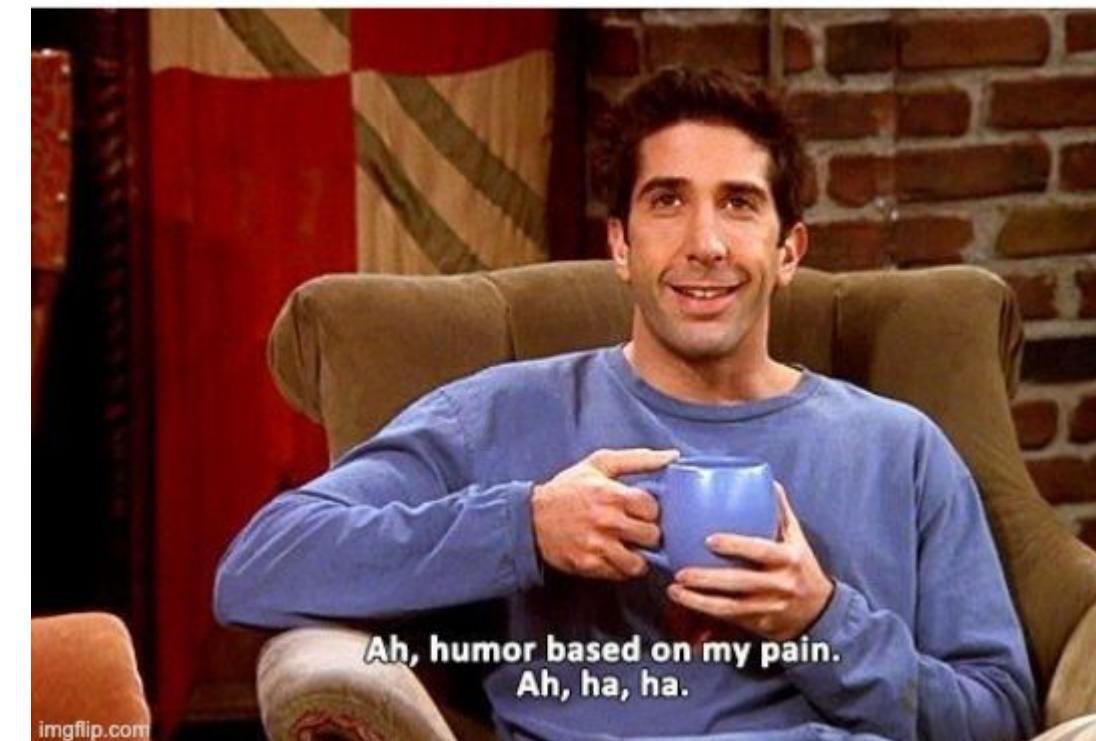
Many Such Checks

# Challenge 4: Containers, Kubernetes and more Complexity for Deployment

```
version: '2.0'

services:
  webgoat:
    image: webgoat/webgoat-8.0
    user: webgoat
    environment:
      - WEBWOLF_HOST=webwolf
      - WEBWOLF_PORT=9090
      - spring.datasource.url=jdbc:postgresql://webgoat_db:5432/webgoat?user=webgoat&password=webgoat
      - spring.datasource.username=webgoat
      - spring.datasource.password=webgoat
      - spring.datasource.driver-class-name=org.postgresql.Driver
      - spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.PostgreSQL10Dialect
      - webgoat.server.directory=/home/webgoat/.webgoat/
      - webgoat.user.directory=/home/webgoat/.webgoat/
    ports:
      - "8080:8080"
  webwolf:
    image: webgoat/webwolf
    environment:
      - spring.datasource.url=jdbc:postgresql://webgoat_db:5432/webgoat?user=webgoat&password=webgoat
      - spring.datasource.username=webgoat
      - spring.datasource.password=webgoat
      - spring.datasource.driver-class-name=org.postgresql.Driver
      - spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.PostgreSQL10Dialect
    ports:
      - "9090:9090"
  webgoat_db:
    image: postgres:10.12
    # Uncomment to store the state of the database on the host.
    # volumes:
    #   - ./database:/var/lib/postgresql
    environment:
      - POSTGRES_PASSWORD=webgoat
      - POSTGRES_USER=webgoat
      - POSTGRES_DB=webgoat
    ports:
      - "5432:5432"
```

**WHEN I SEE A KUBERNETES MEME ON TWITTER**

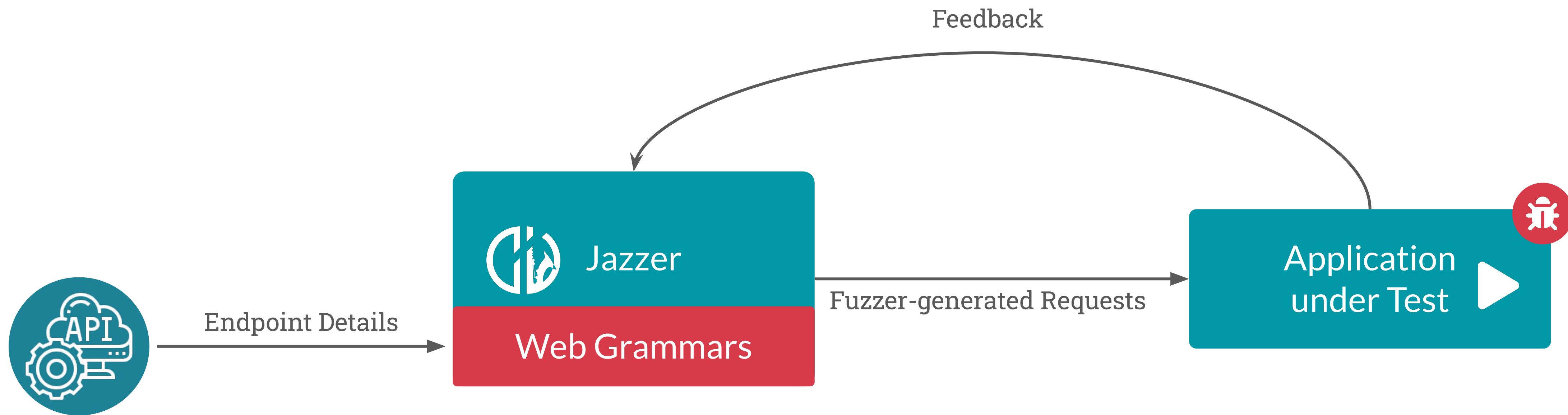


# Challenge 1: We need structured inputs



- Web protocols are pretty standardized
  - REST
  - Web Forms
  - SOAP
  - GraphQL
- Solution
  - Structure-aware fuzzing

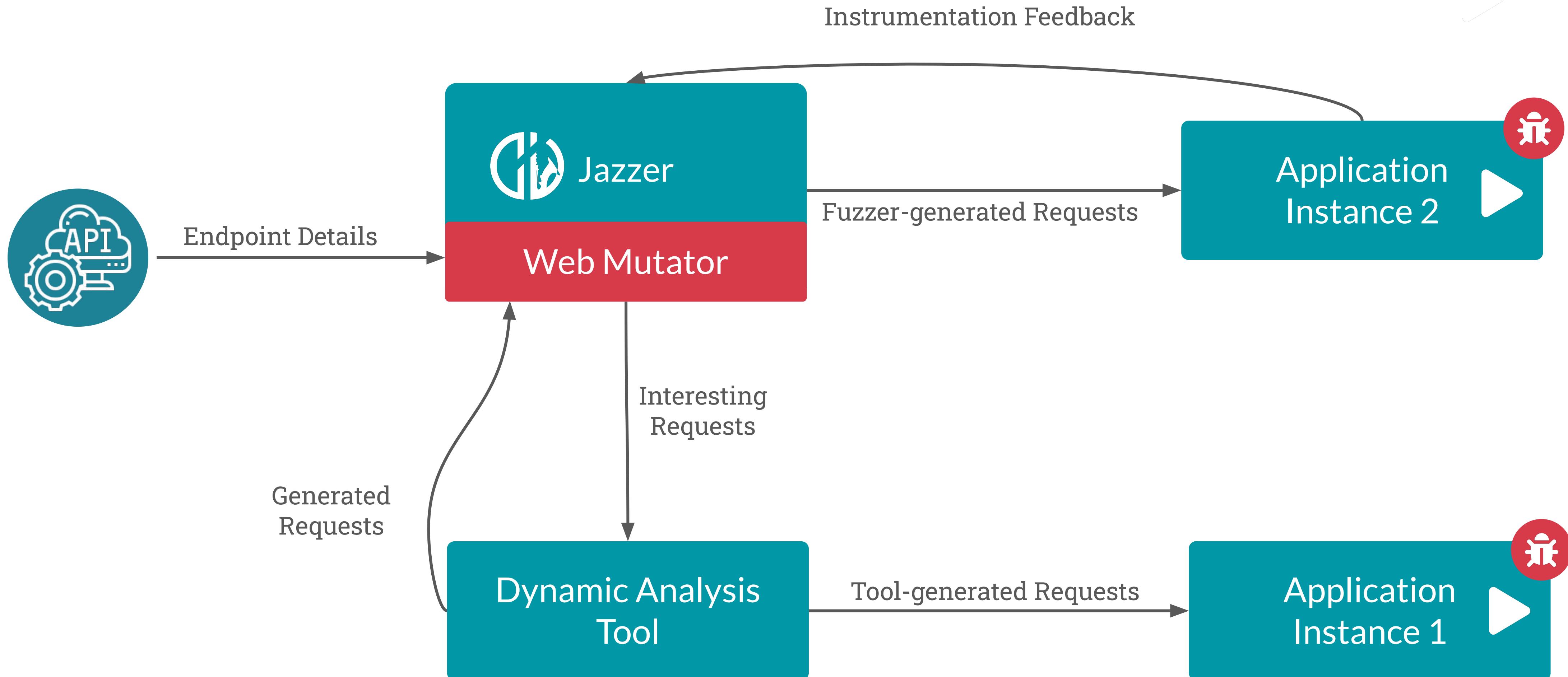
# Solution 1: Let's Bring Structure-aware Fuzzing into Web



## Challenge 2: We need to find OWASP Issues

- We're interested in special classes of security issues
  - OWASP Top 10
  - OWASP API Top 10
- Blackbox testing has false positives
- Solution:
  - Instrumentation-based bug detections
  - Integrating with other DAST tools

## Solution 2: Combine Coverage-based Fuzzing w/ Existing Tooling



## Solution 2: Instrumentation-based Bug Detectors

```
@RestController
@Slf4j
public class Assignment5 extends AssignmentEndpoint {
    //...
    @PostMapping("/challenge/5")
    @ResponseBody
    public AttackResult login(@RequestParam String username_login, @RequestParam String password_login) {
        //...
        if (!"Larry".equals(username_login)) {
            return failed(this).feedback("user.not.larry").feedbackArgs(username_login).build();
        }
        //...
        PreparedStatement stmt = connection.prepareStatement("select password from users where userid = '" +
username_login + "' and password = '" + password_login + "'");
        ResultSet resultSet = statement.executeQuery();  
    }  
}
```

Intercept calls at runtime and perform checks

# Challenge 3: How to handle checks?

```
@RestController  
@Slf4j  
public class Assignment5 extends AssignmentEndpoint {  
    //...  
    @PostMapping("/challenge/5")  
    @ResponseBody  
    public AttackResult login(@RequestParam String username_login, @RequestParam String password_login) {  
        //...  
        if (!"Larry".equals(username_login)) {  
            return failed(this).feedback("user.not.larry").feedbackArgs(username_login).build();  
        }  
        //...  
        PreparedStatement stmt = connection.prepareStatement("select password from users where userid = '" +  
username_login + "' and password = '" + password_login + "'");  
        ResultSet resultSet = statement.executeQuery();  
        //...  
    }  
}
```

Solution 3: Jazzer handles this out of the box

# Result: Effective Testing and Detailed Bug Reporting

The screenshot shows a bug reporting interface for the 'WebGoat' project. The left sidebar includes filters for 'PROJECT', 'Filters', 'Types' (Vulnerability, Bug, Code Smell), and 'Severity' (Critical, High, Medium, Low). The main area displays a table of findings:

Type	Severity ↑	Location	Action
SQL Injection	Critical	webgoat-less.../SqlInjectionLesson4.java:52	DEBUG
SQL Injection	Critical	webgoat-less.../Assignment5.java:67	DEBUG
Cross Site Scripting	High	POST /WebGoat/CrossSiteScripting/phone-home-xss	
Application Error Disclosure	Medium	GET /WebGoat/Challenge.lesson.lesson	
Parameter Tampering	Medium	POST /WebGoat/register.mvc	
Exception Policy Violation	Low	webgoat-contain.../UserValidator.java:31	DEBUG
Information Disclosure	Low	POST /WebGoat/challenge/5	

A red box highlights the second finding (Assignment5.java:67) and its details panel, which contains the following information:

Method: POST Uri: /challenge/5?username\_login=Larry&password\_login=%27z  
Content type: text\_html  
Body: =

## Challenge 4: Complex Setup & Deployment

- Microservices architecture
- Complex deployments
  - Multiple processes
  - Kubernetes
  - OpenShift
  - Special protocol for internal communication

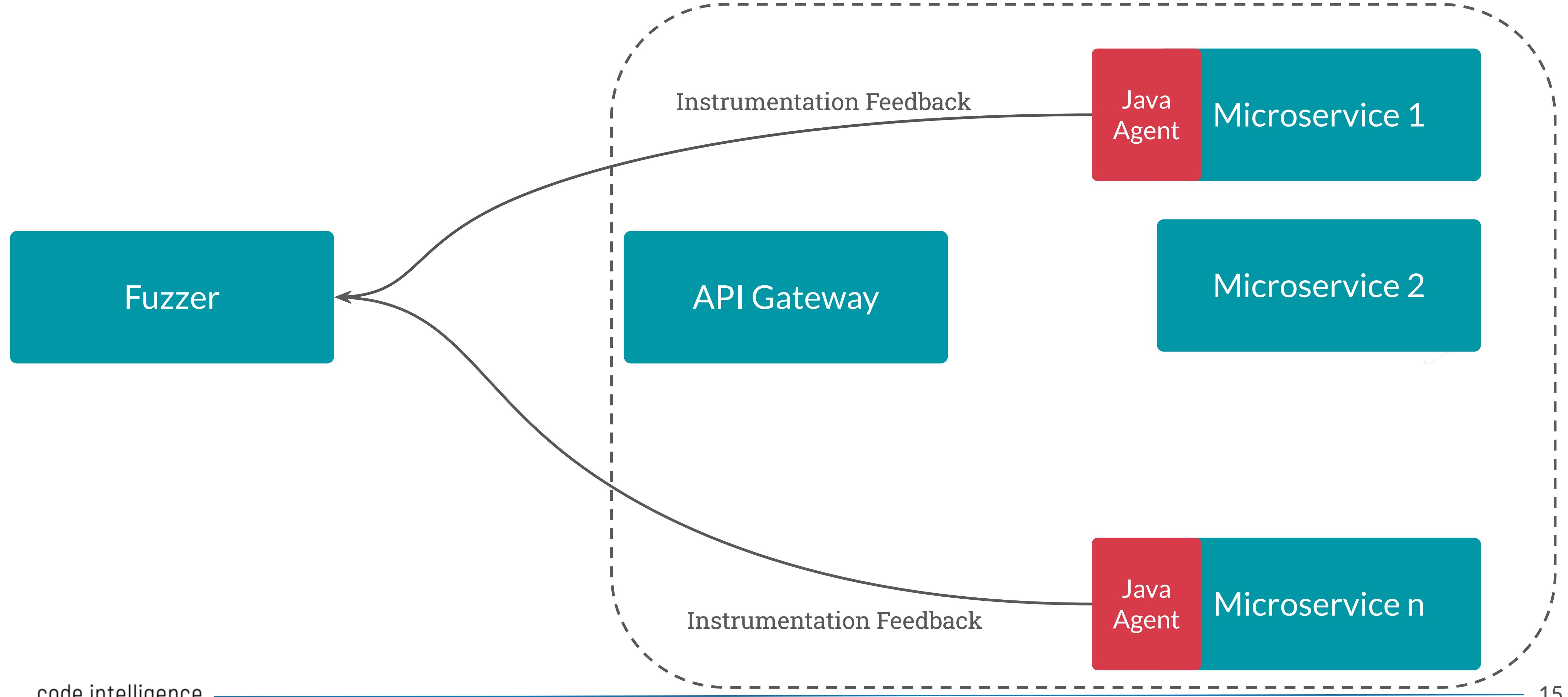


**kubernetes**



**OPENSIFT**

## Solution 4: How to handle complex deployments



# Fuzzing most Effective when Performed Continuously!

**Fuzzing sql injection #4**  
simonresch wants to merge 12 commits into `master` from `fuzzing_sql_injection`

`Open`

- o Merge branch 'master' into action-test
- o Update file
- o Merge branch 'master' into fuzzing\_challenge5
- o Merge branch 'master' into fuzzing\_challenge5
- o Update .gitignore

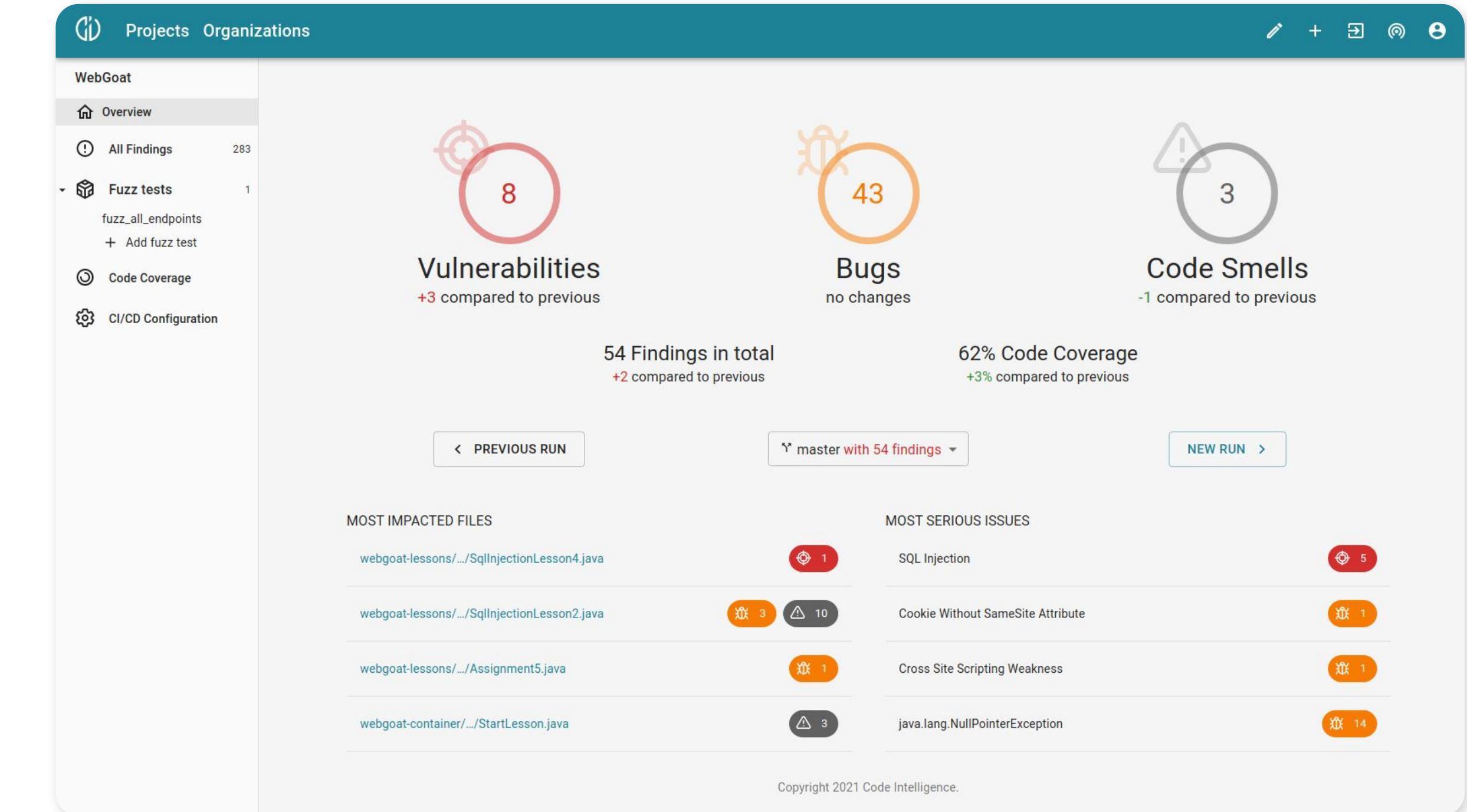
**cifuzz** bot commented on Jan 21

**Found "SQL Injection" while fuzzing:** [View Finding](#)

For more information, check [Code Intelligence Fuzzing Academy](#)

**cifuzz** bot commented on Jan 21

File	Coverage
<code>webgoat-server/src/main/java/org/owasp/webgoat</code>	
<code>HSQLDBDatabaseConfig.java</code>	100%
<code>StartWebGoat.java</code>	100%
<code>webgoat-lessons/secure-passwords/src/main/java/org/owasp/webgoat/secure_password</code>	
<code>SecurePasswordsAssignment.java</code>	100%
<code>SecurePasswords.java</code>	100%
<code>webgoat-lessons/webgoat-introduction/src/main/java/org/owasp/webgoat/introduction</code>	
<code>WebGoatIntroduction.java</code>	100%
<code>webgoat-lessons/challenge/src/main/java/org/owasp/webgoat/challenges/challenge6</code>	
<code>Challenge6.java</code>	100%



# Takeaway: Instrumented Fuzzing Improves Web Testing Immensely

