



› **AUTOMATED VULNERABILITY RESEARCH**
E.G. BROENINK MSC

› INDEX

AUTOMATED VULNERABILITY RESEARCH (AVR)

01. AUTOMATED VULNERABILITY RESEARCH

02. LOG4J

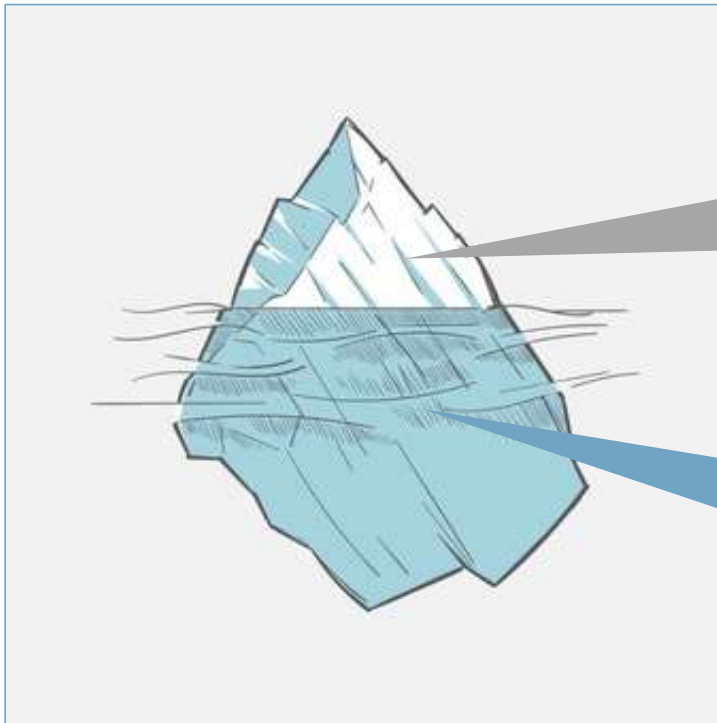
03. KEY APPLICATIONS

04. COLLABORATION

05. TECHNOLOGY FOCUS: FUZZING

06. CURRENT RESEARCH

› AUTOMATED VULNERABILITY RESEARCH (AVR)



current cyber security practice:

protect own IT against
software vulnerabilities that have been
published and patched

AVR:

discovering and patching
unknown/unpublished vulnerabilities
using smart automation to make it scalable

LOG4J

Finding the log4j RCE With Fuzz... x +

code-intelligence.com/blog/java-fuzzing-log4j-rce

code intelligence Product Resources Careers Join the Team

Can Fuzz Testing Help?

Yes – CI Fuzz version 2.28.0 and up can find this vulnerability if log4j is included in the list of libraries to instrument for fuzzing. Due to the high interest in this class of vulnerabilities, we are making the relevant bug detector available to the open-source community. With Jazzer, our open-source fuzzing engine that is part of Google's OSS fuzz, you can now find the log4j vulnerability as well as similar RCE issues in the same way as with our core product CI Fuzz.

Protecting Open-Source Projects With Fuzzing

Since Jazzer is part of OSS-Fuzz, all integrated open-source projects written in Java and other JVM-based languages, are now continuously searched for similar vulnerabilities. A fuzz target for log4j that reproduces the vulnerability can be found [here](#).

```
#16150 NEW cov: 3945 ft: 6197 corp: 405/20Kb lim: 86 exec/s: 2601 rss: 491Mb L: 86/86 MS: 3 ShuffleBytes-ChangeByte-ChangeASCIInt-
#16217 REDUCE cov: 3945 ft: 6197 corp: 405/20Kb lim: 86 exec/s: 2702 rss: 491Mb L: 26/86 MS: 2 Crossover-EraseBytes-
== Java Exception: com.code.intelligence.jazzer.api.FuzzerSecurityIssueCritical: Remote JNDI Lookup
JNDI lookups with attacker-controlled remote URLs can, depending on the JUK
version, lead to remote code execution or the exfiltration of information.
at com.code.intelligence.jazzer.sanitizers.NamingContextLookup.lookupHook(NamingContextLookup.kt:68)
at org.apache.logging.log4j.core.net.JndiManager.lookup(JndiManager.java:172)
at org.apache.logging.log4j.core.lookup.JndiLookup.lookup(JndiLookup.java:56)
at org.apache.logging.log4j.core.lookup.Interpolator.lookup(Interpolator.java:221)
```



14 december 2021 05:30

Laatste update: 14 december 2021 10:59

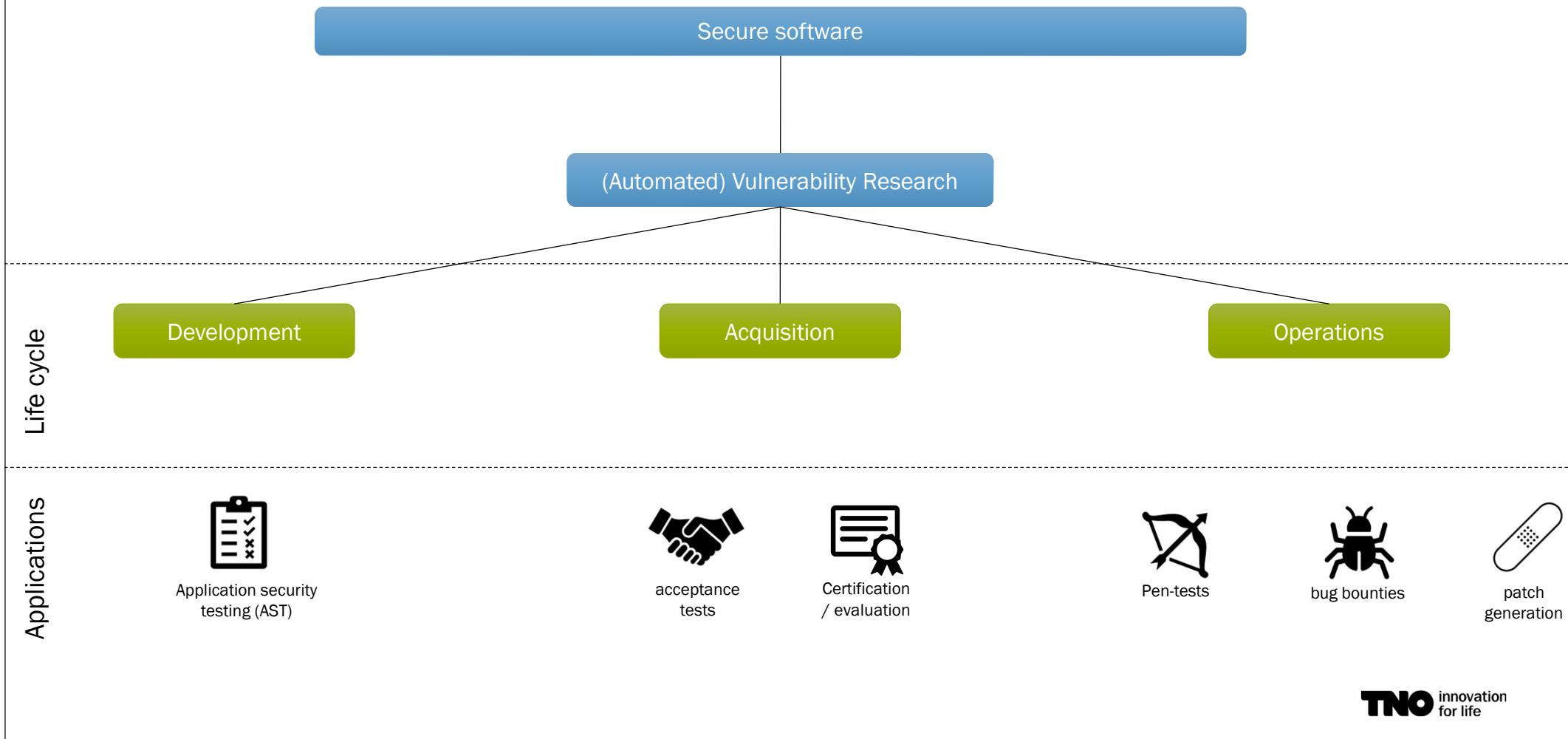
121 NUJij-reacties



Een veelgebruikt stukje software in webservern bevat een ernstige kwetsbaarheid, waardoor veel organisaties kans lopen om gehackt te worden. Er is een oplossing beschikbaar, maar het probleem is dat organisaties lang niet altijd weten dat ze gevaar lopen.

Het gaat om software met de naam Apache Log4j. Dit programma wordt massaal door bedrijven en organisaties gebruikt om bij te houden wat er op hun webservern gebeurt. In dit logboek kan van alles staan, bijvoorbeeld wanneer gebruikers ergens inloggen of waar in de systemen foutmeldingen worden gegeven, zodat beheerders ze kunnen oplossen.

SECURE SOFTWARE AND AVR



› KEY AVR APPLICATIONS

APPLICATION SECURITY TESTING

- › Industry focus on uniform, agile development (DevOps)
- › “shift-left security”: move security to early lifecycle stages
- › AVR to complement current application testing practices

SECURING APPLICATION PROGRAMMING INTERFACES (API)

- › Crucial in modern cloud/microservices architectures
- › External attack surface
- › Securing APIs is an important first line of defence

› COLLABORATIONS

Academic



UNIVERSITY
OF TWENTE.

Application

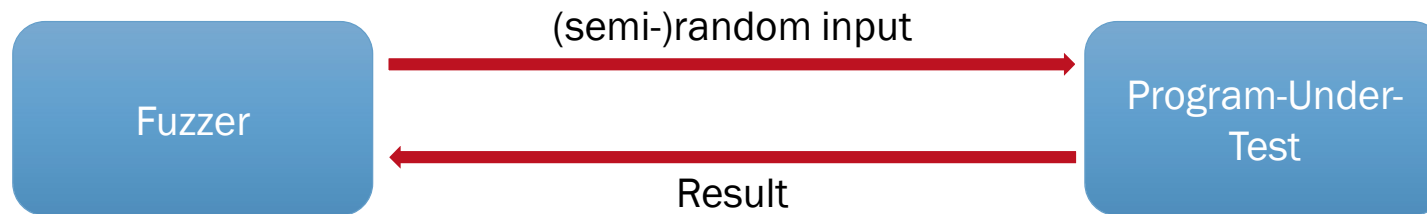


› TECHNOLOGY FOCUS: FUZZING

- › Addition to unit and integration testing
- › SAST tooling already widely used, and can be combined with fuzzing
- › Fuzzing technology is maturing, but not widely applied yet

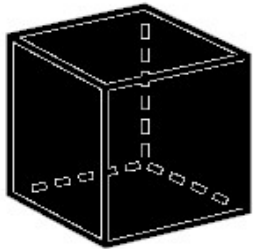
› FUZZING

- Fuzzer sends (semi-) random data to program-under-test.
- Fuzzer tests how the program responds.



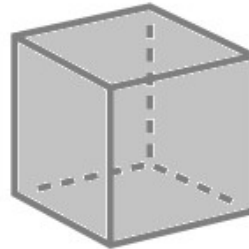
› BLACKBOX – GREYBOX – WHITEBOX

› Different approaches



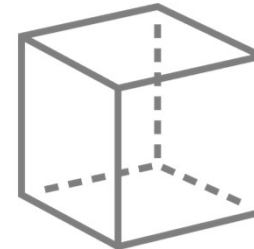
Black box

- No source code
- Only binary file



Grey box:

- No source code
- Instrumented binary

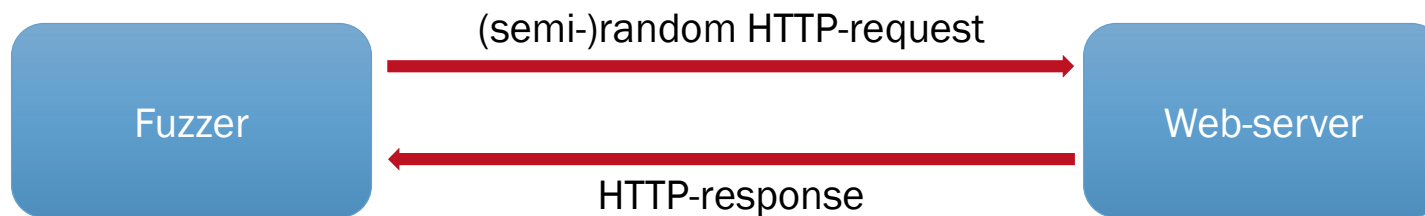
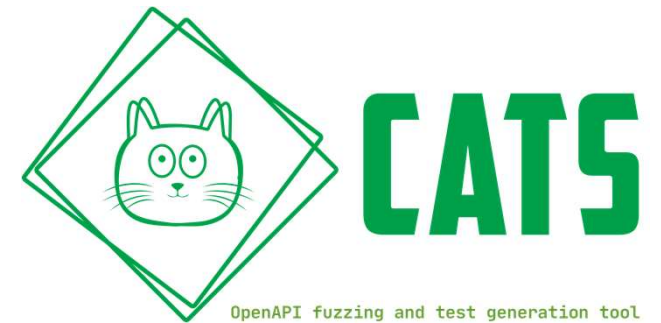


White box:

- Source code

› WEB-API FUZZING – STATE OF THE ART

- Open source tools available
 - CATS
 - RESTler
- Blackbox approach (based on OpenAPI spec)



WEB-API FUZZING – CURRENT RESEARCH

Dashboard Stats

- 1375 Executions per second
- 84 / 110 Total coverage
- 3 Unique crashes

Test Cases:

- + / 4 - 4 - 4 - 4 - 4 - 6 (Statuscode: 200)
- + / 4 - 4 - 4 - 6 (Statuscode: 200)
- + * (Statuscode: 200)
- + / 4 - 4 - 4 + / 4 - 4 - 4 - 6 (Statuscode: 200)
- + / 4 - 4 - 4 - 6 (Statuscode: 200)
- / 4 - 4 - 4 - 6 (Statuscode: 200)
- / 4 - 4 - 4 - (Statuscode: 200)
- 3 * 3 * 3 * (Statuscode: 200)
- 3 * (Statuscode: 200)
- 3 * (Statuscode: 200)
- + / (Statuscode: 200)
- * (Statuscode: 200)
- 4 4 4 (- (Statuscode: 200)
- 4 (- (Statuscode: 200)
- + (Statuscode: 200)
- 4 - 4 - - 4 - (Statuscode: 200)

```
@Override
public double process(String expression) throws CalculatorException {
    String[] tokens = expression.split(" ");
    if (tokens.length > 20) {
        throw new RuntimeException("Runtime Exception!");
    }
    Deque<String> operators = new ArrayDeque<>();
    Deque<Double> numbers = new ArrayDeque<>();
    for (String token : tokens) {
        switch (token) {
            case "+":
            case "-":
            case "*":
            case "/":
                while (shouldEvaluate(tokens, operators.peekFirst())) {
                    String op = operators.pop();
                    double second = numbers.pop();
                    double first = numbers.pop();
                    double result;
                    switch (op) {
                        case "+":
                            result = first + second;
                            break;
                        case "-":
                            result = first - second;
                            break;
                        case "*":
                            result = first * second;
                            break;
                        case "/":
                            result = first / second;
                            break;
                        default:
                            throw new CalculatorException("Unexpected operator");
                    }
                    numbers.push(result);
                }
                operators.push(token);
            case "(":
                operators.push(token);
            case ")":
                for (String op = operators.peekFirst(); !op.equals("("); op = operators.pop()) {
                    double second = numbers.pop();
                    double first = numbers.pop();
                    double result;
                    switch (op) {
                        case "+":
                            result = first + second;
                            break;
                        case "-":
                            result = first - second;
                            break;
                        case "*":
                            result = first * second;
                            break;
                        case "/":
                            result = first / second;
                            break;
                        default:
                            throw new CalculatorException("Unexpected operator");
                    }
                    numbers.push(result);
                }
                operators.pop();
            default:
                double d = Double.parseDouble(token);
                numbers.push(d);
                break;
        }
    }
    return numbers.pop();
}
```

Dashboard Stats

- 1452 Executions per second
- 84 / 110 Total coverage
- 3 Unique crashes

Test Cases:

- + / 4 - 4 - 4 - 4 - 4 - 6 (Statuscode: 200)
- + / 4 - 4 - 4 - 6 (Statuscode: 200)
- + * (Statuscode: 200)
- + / 4 - 4 - 4 + / 4 - 4 - 4 - 6 (Statuscode: 200)
- + / 4 - 4 - 4 - 6 (Statuscode: 200)
- / 4 - 4 - 4 - 6 (Statuscode: 200)
- / 4 - 4 - 4 - (Statuscode: 200)
- 3 * 3 * 3 * (Statuscode: 200)
- 3 * (Statuscode: 200)
- 3 * (Statuscode: 200)
- + / (Statuscode: 200)
- * (Statuscode: 200)
- 4 4 4 (- (Statuscode: 200)
- 4 (- (Statuscode: 200)
- + (Statuscode: 200)
- 4 - 4 - - 4 - (Statuscode: 200)

```
@Override
public double process(String expression) throws CalculatorException {
    String[] tokens = expression.split(" ");
    if (tokens.length > 20) {
        throw new RuntimeException("Runtime Exception!");
    }
    Deque<String> operators = new ArrayDeque<>();
    Deque<Double> numbers = new ArrayDeque<>();
    try {
        for (String token : tokens) {
            switch (token) {
                case "+":
                case "-":
                case "*":
                case "/":
                    while (shouldEvaluate(tokens, operators.peekFirst())) {
                        String op = operators.pop();
                        double second = numbers.pop();
                        double first = numbers.pop();
                        double result;
                        switch (op) {
                            case "+":
                                result = first + second;
                                break;
                            case "-":
                                result = first - second;
                                break;
                            case "*":
                                result = first * second;
                                break;
                            case "/":
                                result = first / second;
                                break;
                            default:
                                throw new CalculatorException("Unexpected operator");
                        }
                        numbers.push(result);
                    }
                    operators.push(token);
                case "(":
                    operators.push(token);
                case ")":
                    for (String op = operators.peekFirst(); !op.equals("("); op = operators.pop()) {
                        double second = numbers.pop();
                        double first = numbers.pop();
                        double result;
                        switch (op) {
                            case "+":
                                result = first + second;
                                break;
                            case "-":
                                result = first - second;
                                break;
                            case "*":
                                result = first * second;
                                break;
                            case "/":
                                result = first / second;
                                break;
                            default:
                                throw new CalculatorException("Unexpected operator");
                        }
                        numbers.push(result);
                    }
                    operators.pop();
                default:
                    double d = Double.parseDouble(token);
                    numbers.push(d);
                    break;
            }
        }
        return numbers.pop();
    } catch (Exception e) {
        throw new RuntimeException("Runtime Exception!");
    }
}
```

› FUZZING IN CI/CD CHAIN

- Early in the development proces, bugs are easy to fix
- CI/CD tooling already have extensive test environment
- Fuzzing takes place at pre-defined moments:
 - Each commit
 - Every day
 - Every weekend

› RESEARCH PROJECT

- › New research project on applying fuzzing technology. (TKI)
- › Aim: make fuzzing technology applicable in:
 - › CI/CD chains
 - › Grey-box web API's
- › Aim: 5 industry partners
- › Timeline: Start Q3/Q4 2022



TNO innovation
for life

Q3 2022

IMPROVING SECURE SOFTWARE DEVELOPMENT

Testing is an important aspect of software development. Only thorough testing ensures that newly developed software functions as intended. Errors in software code may lead to incorrect behavior, or even to system crashes or security vulnerabilities. Today's software testing is largely based on manually defined test scenarios. As a result, the number of scenarios is limited, and mostly focus on parts of the code considered important by testers. Comprehensively testing all parts of an application is not feasible.

Fuzz testing – or fuzzing – is an alternative approach to software testing. Instead of applying predefined test scenarios, fuzzing triggers a large number of executions with varying, (semi-) random inputs. Thus, fuzzing verifies the dynamic behavior of software in unforeseen circumstances or states. It has proven itself as a valuable technique for identifying bugs. Fuzzing does not replace traditional software testing methods. Rather, it adds and enhances the ability to test alternative execution paths at scale.

BENEFITS: FUZZING IN CI/CD FOR BETTER SECURITY

TNO is currently working with various organizations to explore the feasibility of adopting fuzzing in modern CI/CD environments. Based on promising initial results, we aim to intensify the effort and involve more partners. This should take the shape of a joint two-year project for applied research on the following challenges:

- › Integrating fuzzing technology into real life software development chains
- › Preparing applications for effective fuzzing



› **THANK YOU FOR
YOUR TIME**



Paske, B.J. (Bert Jan) te
Senior Consultant Cybersecurity



Rooljakkens, T.A. (Thomas)
Scientist Integrator



Broenink, E.G. (Gerben)
Security specialist