Automatic Test Case Generation: Toward Its Application in Exploit Generation for Known Vulnerabilities

Emanuele Iannone
University of Salerno, Italy

Automatic Test Case Generation: Toward Its Application in Exploit Generation for Known Vulnerabilities

Emanuele Iannone
University of Salerno, Italy

Automatic Test Case Generation: Toward Its Application in Exploit Generation for Known Vulnerabilities

Emanuele Iannone
University of Salerno, Italy

sesa lab
SOFTWARE ENGINEERING SALERNO

Emanuele Iannone
I Year PhD Student, 24 y.o.

emaiannone.github.io

EmanueleIannone3

eiannone@unisa.it

Emanuele Iannone
I Year PhD Student, 24 y.o.

emaiannone.github.io

EmanueleIannone3

eiannone@unisa.it

Empirical Software Engineering

SOFTWARE ENGINEERING SALERNO

Emanuele Iannone
I Year PhD Student, 24 y.o.

emaiannone.github.io

EmanueleIannone3

eiannone@unisa.it

sesa lab
SOFTWARE ENGINEERING SALERNO

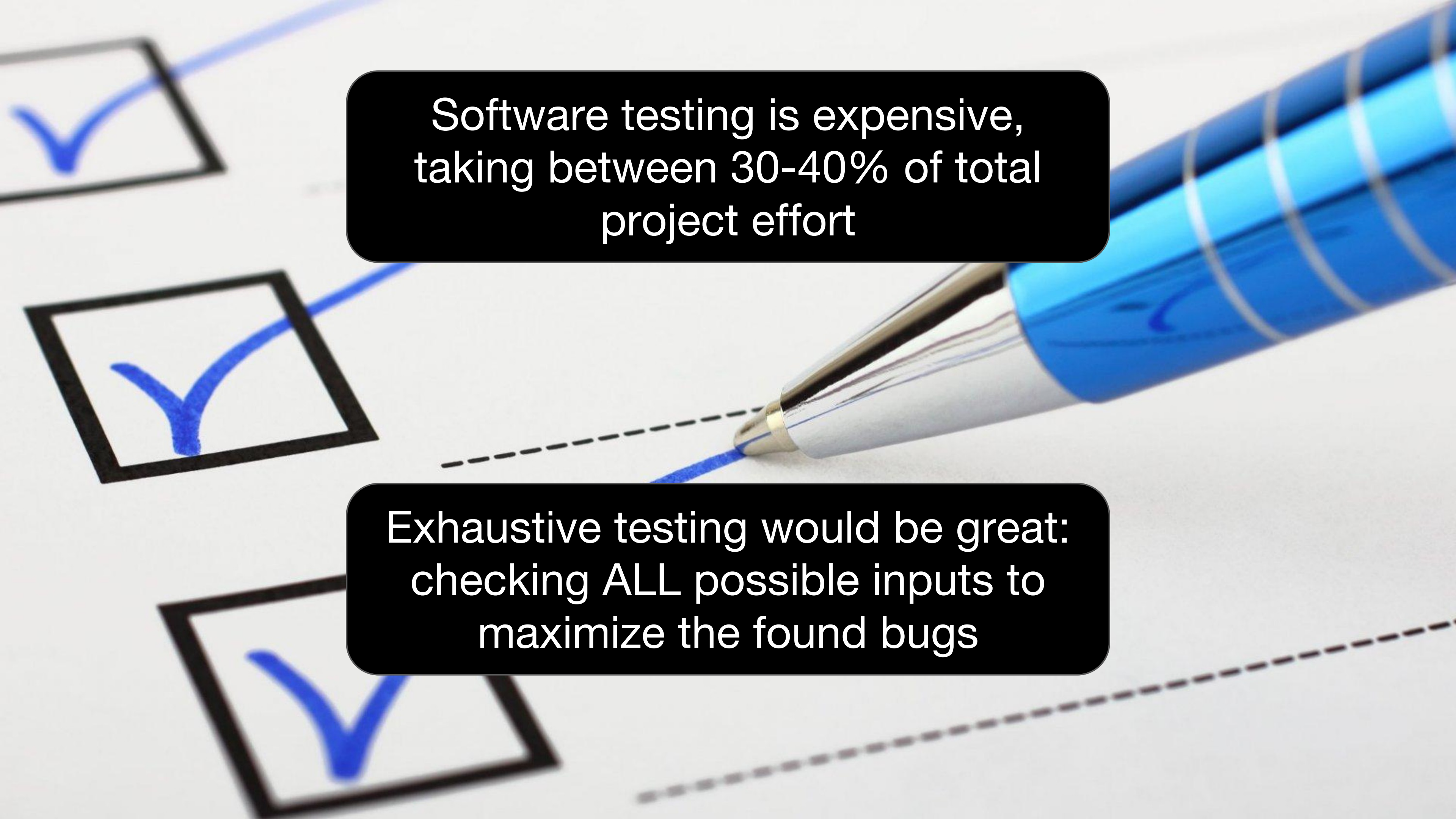
Software Quality

Empirical Software Engineering

Software Security

Software Maintenance and Evolution

Software testing is expensive, taking between 30-40% of total project effort


Software testing is expensive, taking between 30-40% of total project effort
Exhaustive testing would be great: checking ALL possible inputs to maximize the found bugs

Software testing is expensive, taking between 30-40% of total project effort

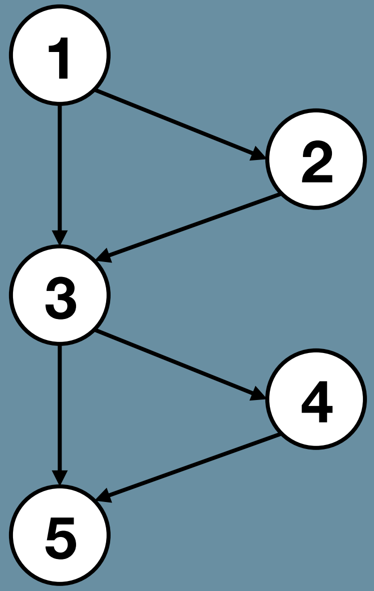
UNFEASIBLE

great:
ssible inputs to
ximize the found bugs

There exists approximate but **systematic** approaches

There exists approximate but **systematic** approaches

```
void foo (int a, int b) {
1 if (a < 0)
2     System.out.println("a is negative");
3 if (b < 0)
4     System.out.println("b is negative");
5 return;
}
```

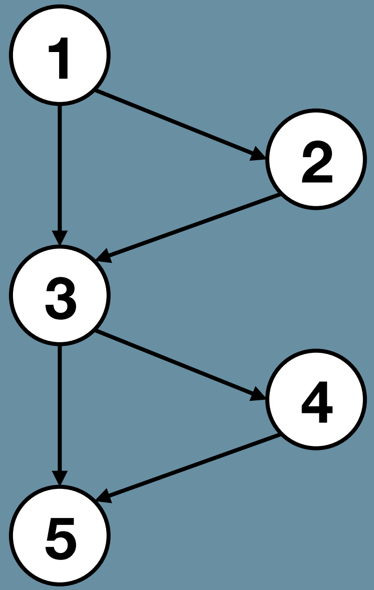


There exists approximate but **systematic** approaches

```
void foo (int a, int b) {
1 if (a < 0)
2     System.out.println("a is negative");
3 if (b < 0)
4     System.out.println("b is negative");
5 return;
}
```

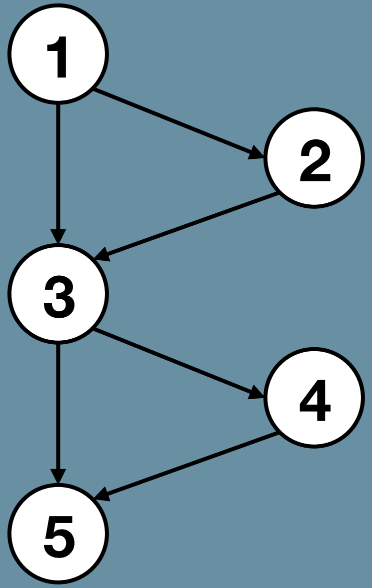
Criterion

Statement Coverage



There exists approximate but **systematic** approaches

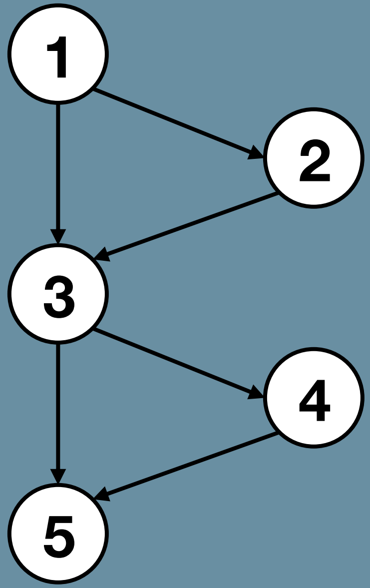
```
void foo (int a, int b) {
1 if (a < 0)
2     System.out.println("a is negative");
3 if (b < 0)
4     System.out.println("b is negative");
5 return;
}
```



Criterion	Goals
Statement Coverage	{1, 2, 3, 4, 5}

There exists approximate but **systematic** approaches

```
void foo (int a, int b) {
1 if (a < 0)
2     System.out.println("a is negative");
3 if (b < 0)
4     System.out.println("b is negative");
5 return;
}
```



Criterion

Statement Coverage

Goals

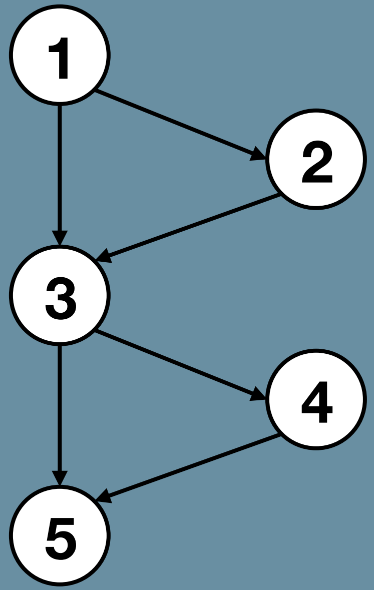
{1, 2, 3, 4, 5}

TC

foo(-1,-1)

There exists approximate but **systematic** approaches

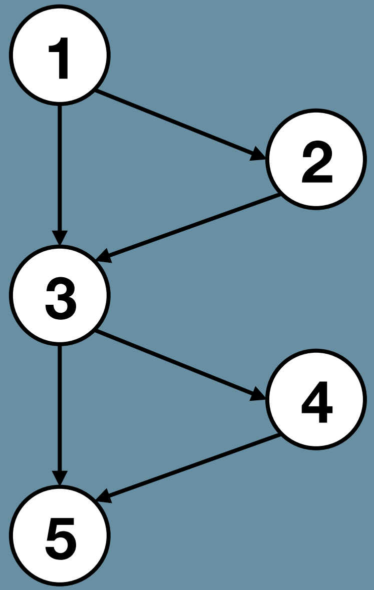
```
void foo (int a, int b) {
1 if (a < 0)
2     System.out.println("a is negative");
3 if (b < 0)
4     System.out.println("b is negative");
5 return;
}
```



Criterion	Goals	TC
Path Coverage	{<1,3,5>, <1,2,3,5>, <1,3,4,5>, <1,2,3,4,5>}	foo(1,1) foo(-1,1) foo(1,-1) foo(-1,-1)

There exists approximate but **systematic** approaches

```
void foo (int a, int b) {
1 if (a < 0)
2     System.out.println("a is negative");
3 if (b < 0)
4     System.out.println("b is negative");
5 return;
}
```



Criterion

Branch Coverage

Goals

{<1,2>, <1,3>, <3,4>, <3,5>}

TC

foo(1,1)
foo(-1,-1)

There exists approximate but **systematic** approaches

Unfortunately, this is tedious if done manually

There exists approximate but **systematic** approaches

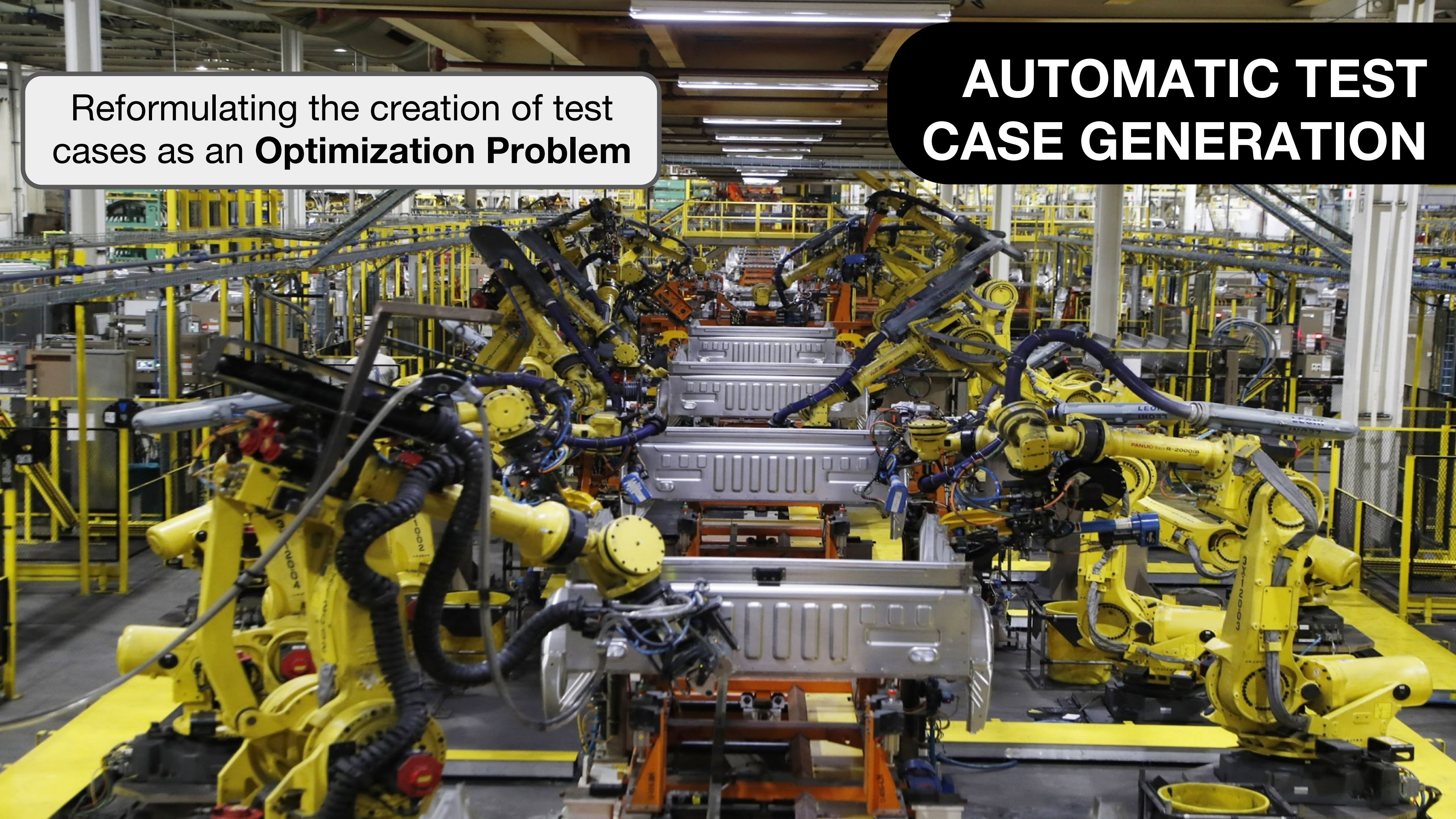
Unfortunately, this is tedious if done manually

Fortunately, we have automated solutions

AUTOMATIC TEST CASE GENERATION

AUTOMATIC TEST CASE GENERATION

Reformulating the creation of test cases as an **Optimization Problem**



AUTOMATIC TEST CASE GENERATION

Reformulating the creation of test cases as an **Optimization Problem**

METAHEURISTICS

Generic procedures to define an optimization algorithm able to quickly explore the search space and provide near-optimal solutions

AUTOMATIC TEST CASE GENERATION

Reformulating the creation of test cases as an **Optimization Problem**

METAHEURISTICS

Generic procedures to define an optimization algorithm able to quickly explore the search space and provide near-optimal solutions

Tabu Search

GENETIC ALGORITHMS

Ant Colony Optimization

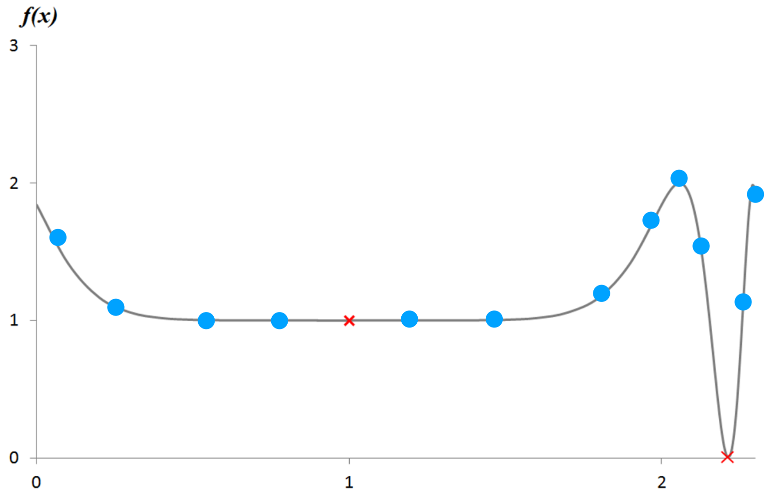
Simulated Annealing

GENETIC ALGORITHMS

Inspired by the natural selection mechanisms, **evolves** a set of candidate solutions to **optimize** a given fitness function

GENETIC ALGORITHMS

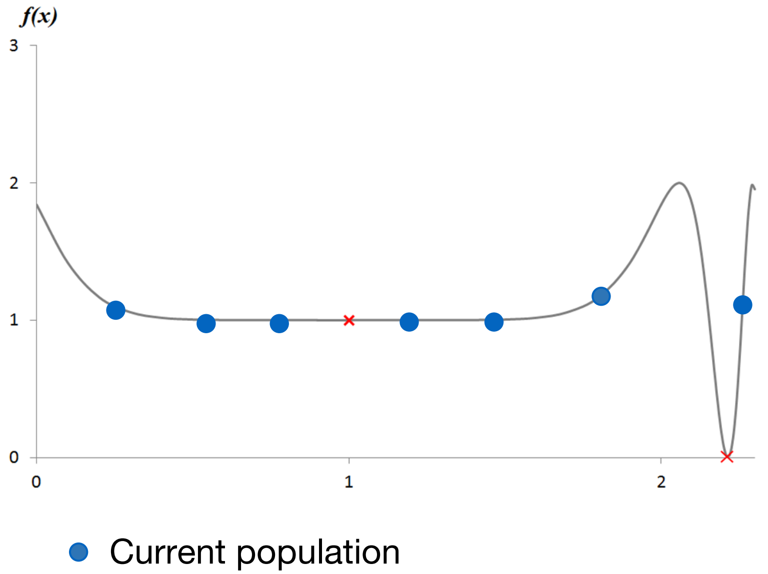
Inspired by the natural selection mechanisms, **evolves** a set of candidate solutions to **optimize** a given fitness function

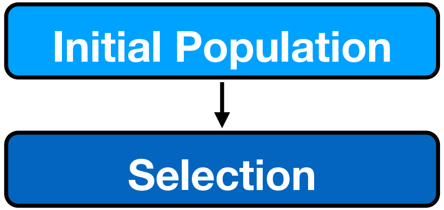


Initial Population

GENETIC ALGORITHMS

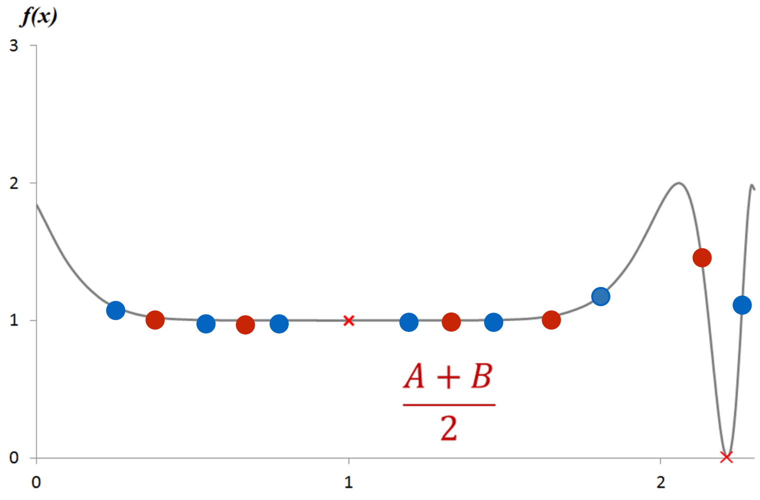
Inspired by the natural selection mechanisms, **evolves** a set of candidate solutions to **optimize** a given fitness function





GENETIC ALGORITHMS

Inspired by the natural selection mechanisms, **evolves** a set of candidate solutions to **optimize** a given fitness function



- Current population
- New solutions (offsprings)

Initial Population

↓

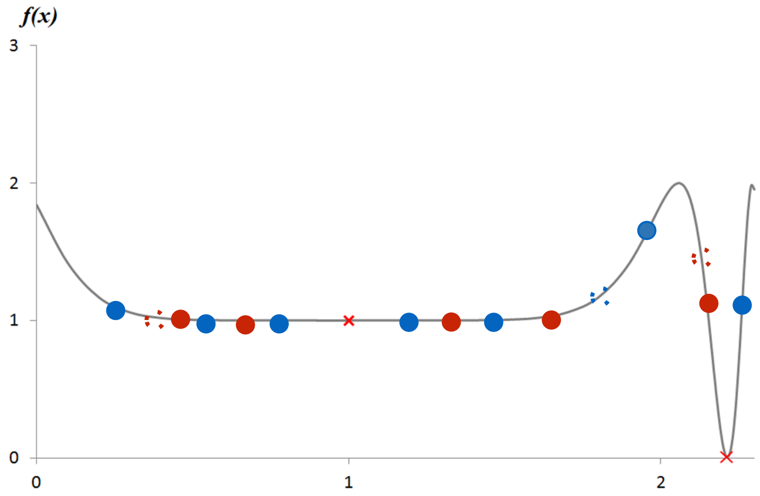
Selection

↓

Crossover

GENETIC ALGORITHMS

Inspired by the natural selection mechanisms, **evolves** a set of candidate solutions to **optimize** a given fitness function

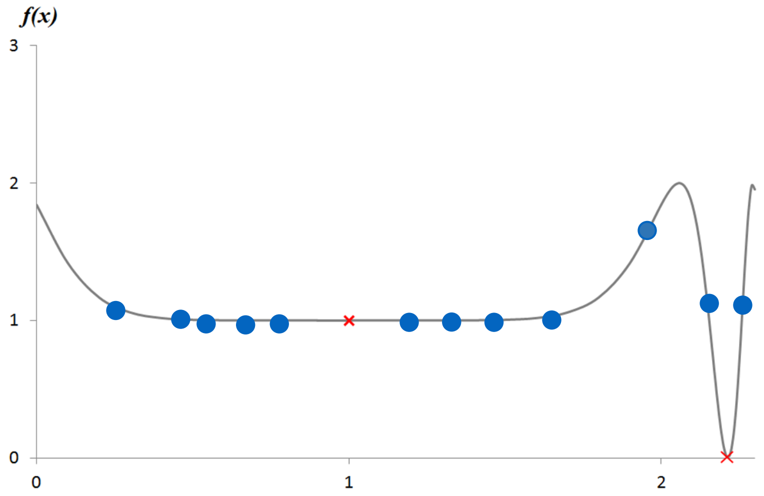


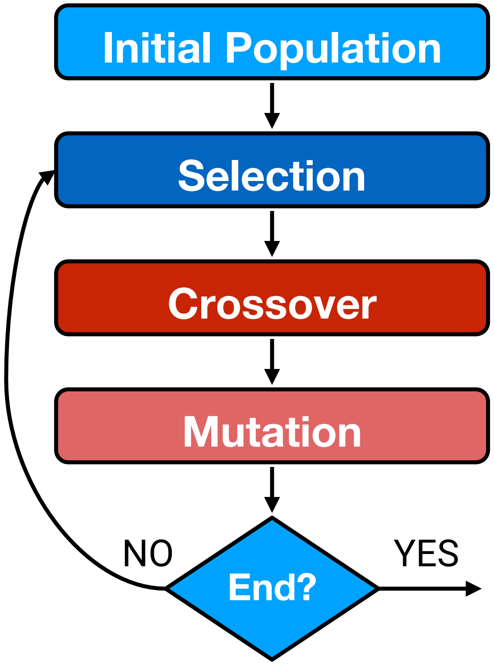
- Current population
- New solutions (offsprings)

Initial Population → Selection → Crossover → Mutation

GENETIC ALGORITHMS

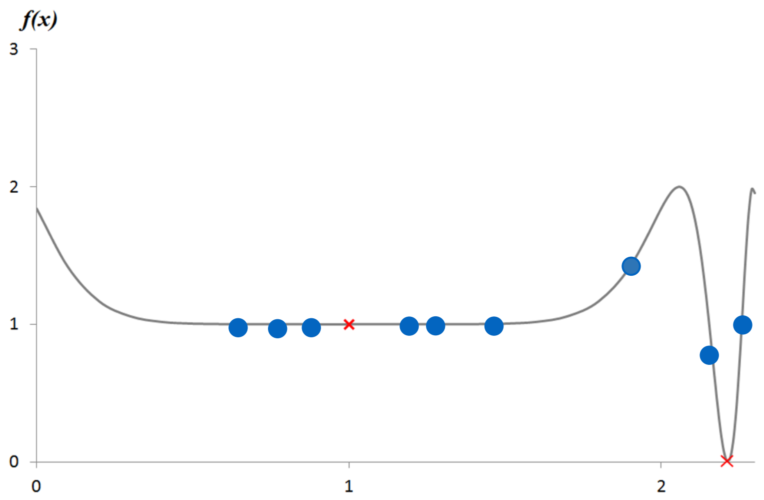
Inspired by the natural selection mechanisms, **evolves** a set of candidate solutions to **optimize** a given fitness function

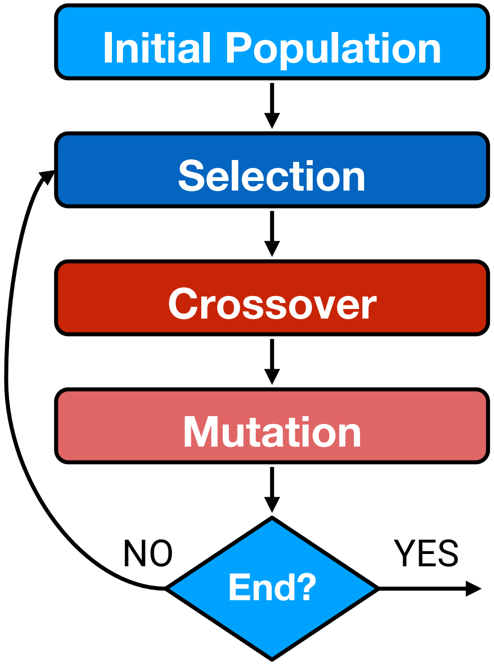




GENETIC ALGORITHMS

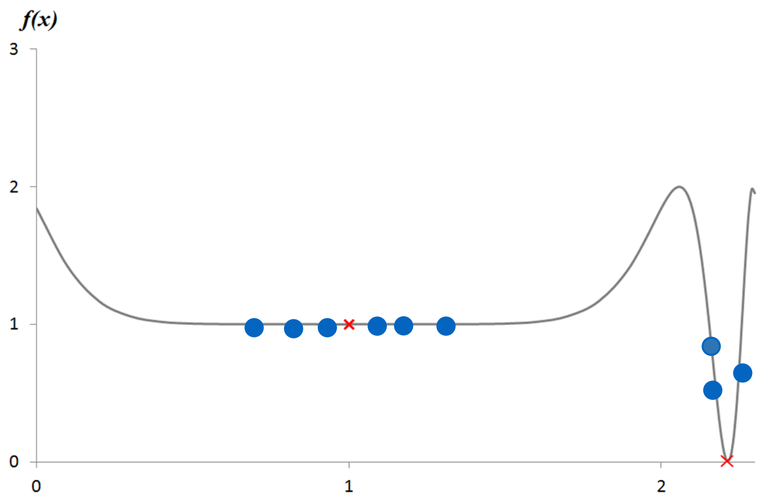
Inspired by the natural selection mechanisms, **evolves** a set of candidate solutions to **optimize** a given fitness function

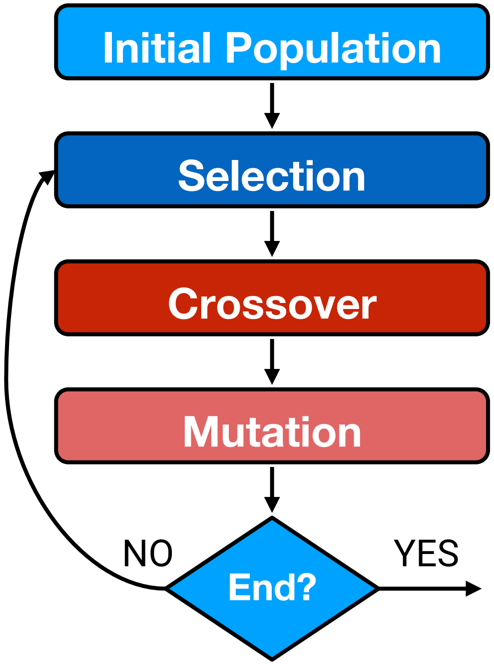




GENETIC ALGORITHMS

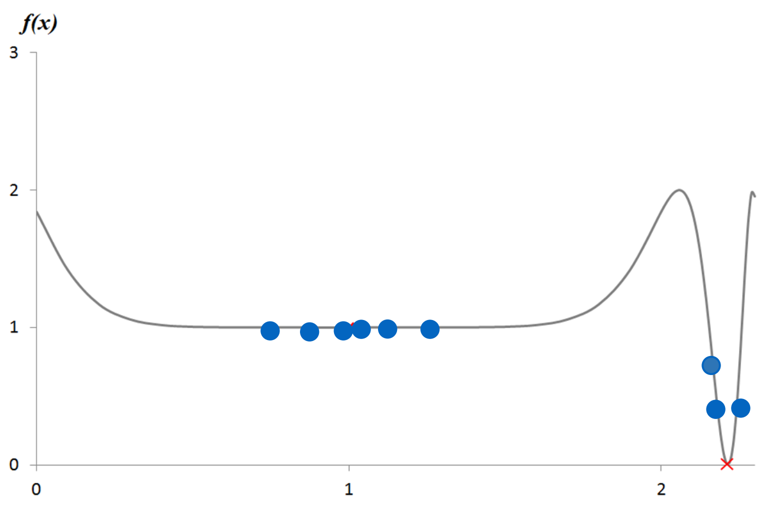
Inspired by the natural selection mechanisms, **evolves** a set of candidate solutions to **optimize** a given fitness function

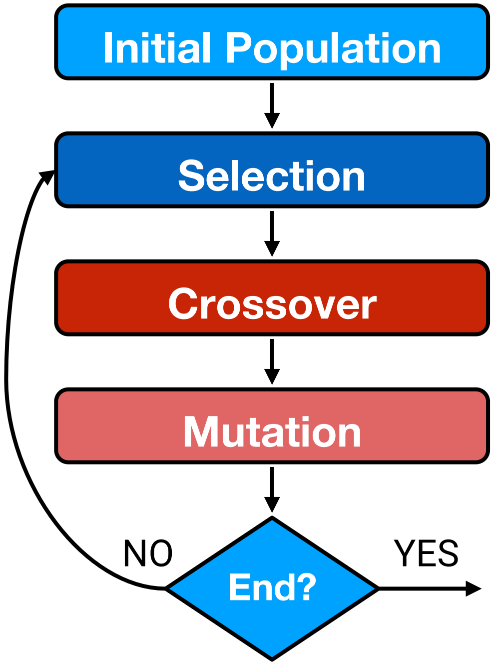




GENETIC ALGORITHMS

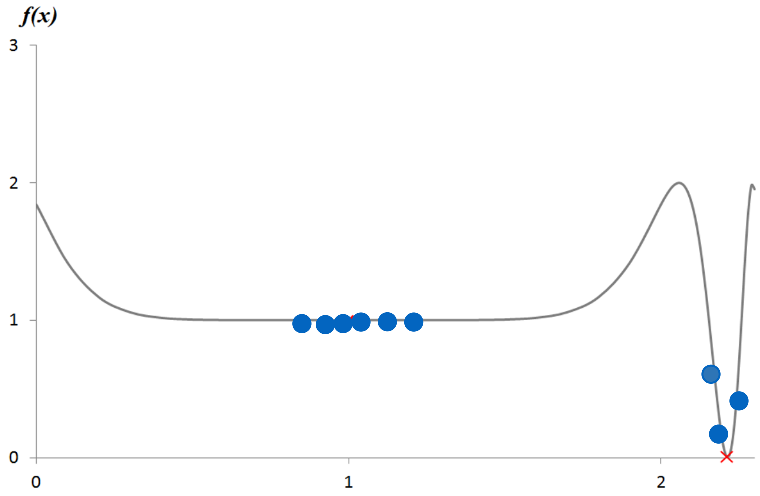
Inspired by the natural selection mechanisms, **evolves** a set of candidate solutions to **optimize** a given fitness function

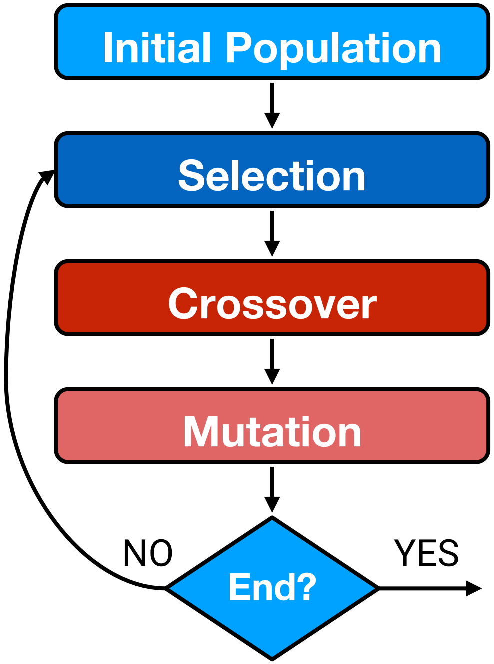




GENETIC ALGORITHMS

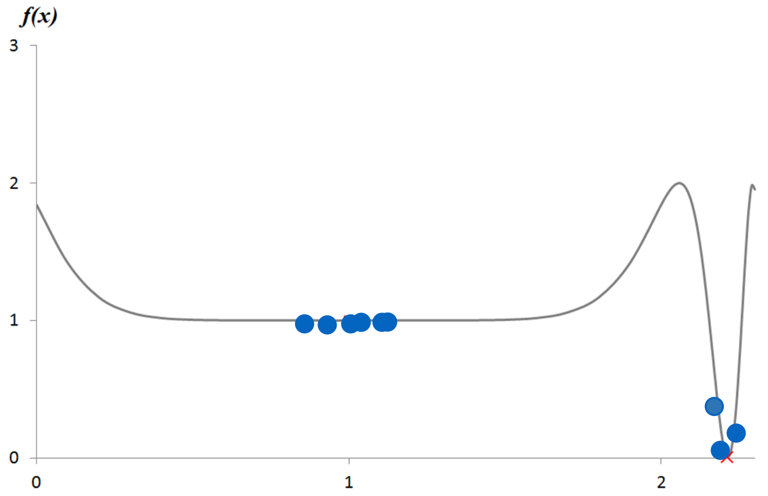
Inspired by the natural selection mechanisms, **evolves** a set of candidate solutions to **optimize** a given fitness function

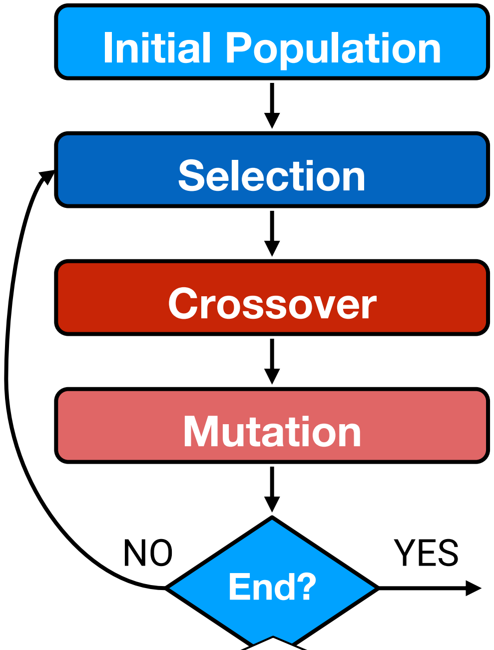




GENETIC ALGORITHMS

Inspired by the natural selection mechanisms, **evolves** a set of candidate solutions to **optimize** a given fitness function

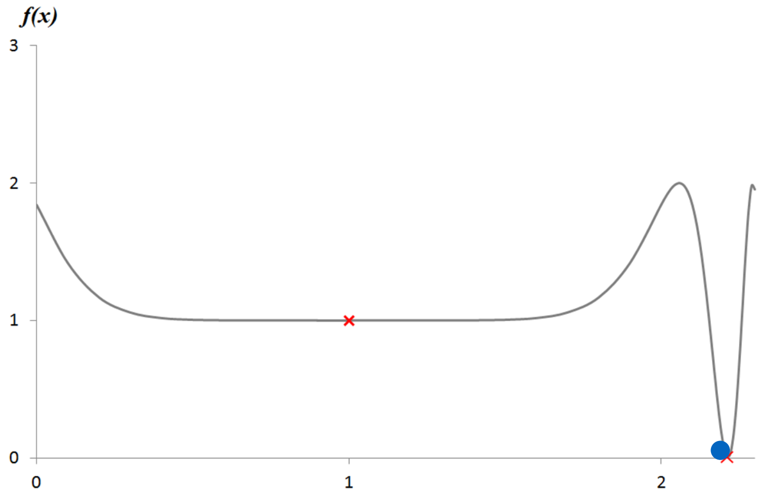


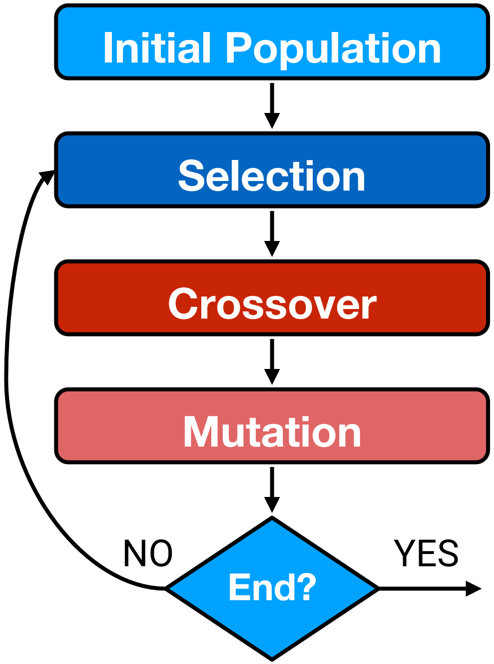


Stopping condition based on **search budget** or when **convergence** is reached

GENETIC ALGORITHMS

Inspired by the natural selection mechanisms, **evolves** a set of candidate solutions to **optimize** a given fitness function





Let's use a GA to generate tests for this method

```
void computeTriangleType() {
1  if (a == b) {
2    if (b == c)
3      type = "EQUILATERAL";
     else
4      type = "ISOSCELES";
     }
5    else if (a == c) {
6      type = "ISOSCELES";
     } else {
7      if (b == c)
8        type = "ISOSCELES";
       else
9        checkRightAngle();
       }
10   System.out.println(type);
}
```

Let's use a GA to generate tests for this method

```java
void computeTriangleType() {
1 if (a == b) {
2   if (b == c)
3     type = "EQUILATERAL";
    else
4     type = "ISOSCELES";
    }
5   else if (a == c) {
6     type = "ISOSCELES";
    } else {
7     if (b == c)
8       type = "ISOSCELES";
      else
9       checkRightAngle();
      }
10   System.out.println(type);
}
```

Individual Encoding

$t=Triangle(int,int,int):$t.computeTriangleType() @
10, 12, 5

```java
@Test
public void test(){
 Triangle t = new Triangle(10,12,5);
 t.computeTriangleType();
}
```

Let's use a GA to generate tests for this method

```
void computeTriangleType() {
1 if (a == b) {
2   if (b == c)
3     type = "EQUILATERAL";
    else
4     type = "ISOSCELES";
    }
5   else if (a == c) {
6     type = "ISOSCELES";
    } else {
7     if (b == c)
8       type = "ISOSCELES";
      else
9       checkRightAngle();
      }
10   System.out.println(type);
}
```

Individual Encoding

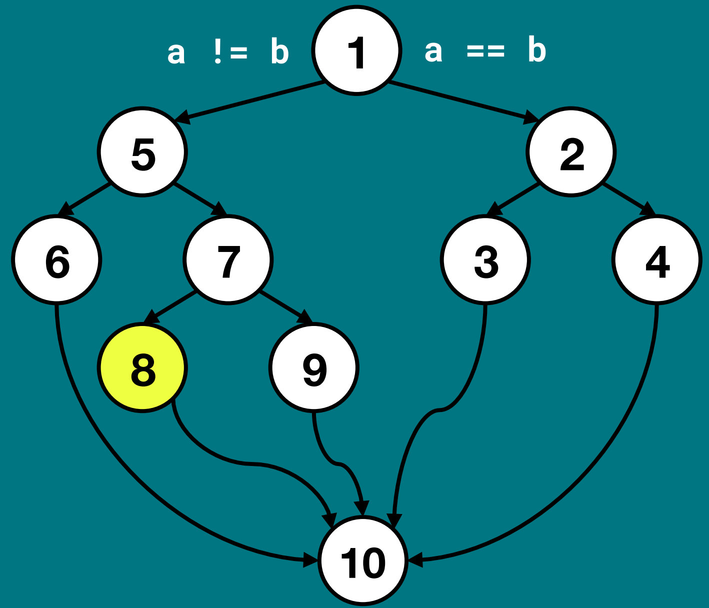
$t=Triangle(int,int,int):$t.computeTriangleType() @ 10, 12, 5

Statement coverage

f(x) = AL(P(x),t) + BD(P(x),t)

Let's use a GA to generate tests for this method

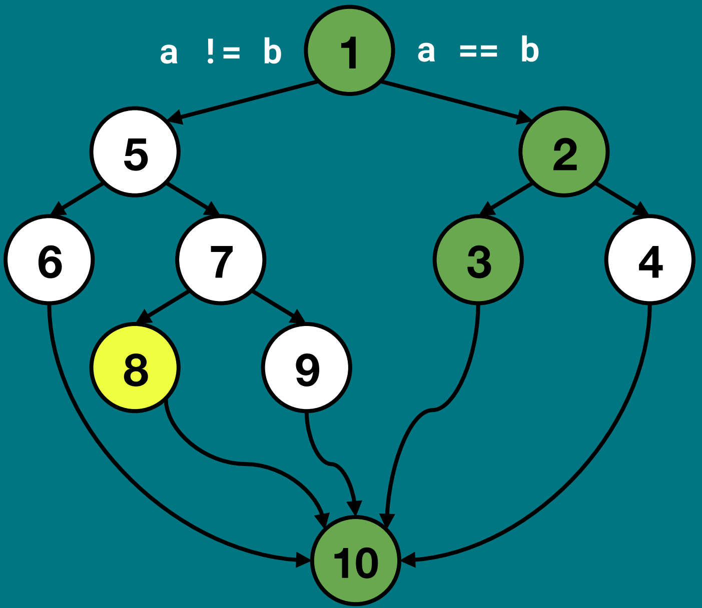
```
void computeTriangleType() {
1 if (a == b) {
2   if (b == c)
3     type = "EQUILATERAL";
    else
4     type = "ISOSCELES";
    }
5   else if (a == c) {
6     type = "ISOSCELES";
    } else {
7     if (b == c)
8       type = "ISOSCELES";
      else
9       checkRightAngle();
      }
10  System.out.println(type);
}
```



~~$t=Triangle(int,int,int):$t.computeTriangleType()~~
@
~~2, 2, 2~~

Let's use a GA to generate tests for this method

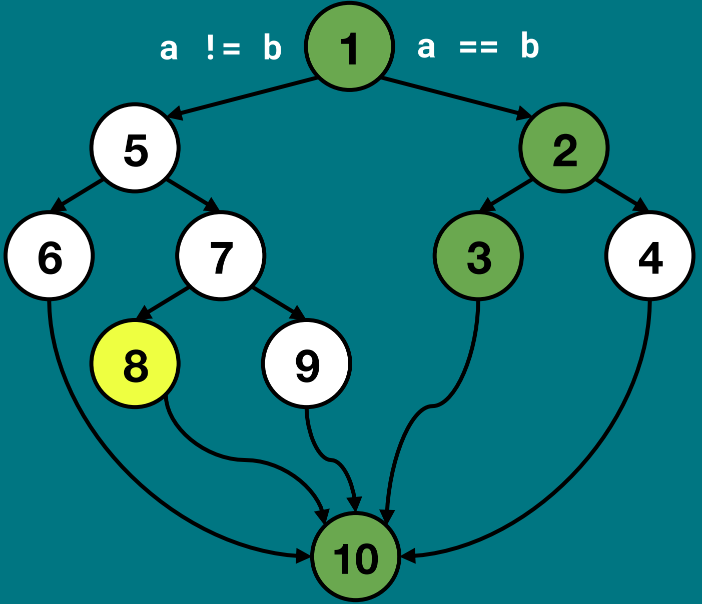
```
void computeTriangleType() {
1 if (a == b) {
2   if (b == c)
3     type = "EQUILATERAL";
    else
4     type = "ISOSCELES";
    }
5   else if (a == c) {
6     type = "ISOSCELES";
    } else {
7     if (b == c)
8       type = "ISOSCELES";
      else
9       checkRightAngle();
      }
10   System.out.println(type);
}
```



~~$t=Triangle(int,int,int):$t.computeTriangleType()~~
@
~~2, 2, 2~~

Let's use a GA to generate tests for this method

```
void computeTriangleType() {
1 if (a == b) {
2   if (b == c)
3     type = "EQUILATERAL";
    else
4     type = "ISOSCELES";
    }
5   else if (a == c) {
6     type = "ISOSCELES";
    } else {
7     if (b == c)
8       type = "ISOSCELES";
      else
9       checkRightAngle();
      }
10  System.out.println(type);
}
```

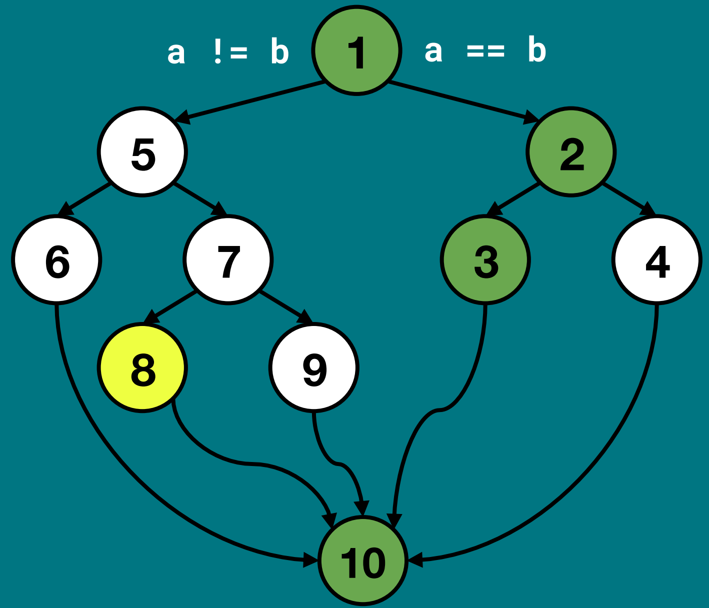


$t=Triangle(int,int,int):$t.computeTriangleType()
@
2, 2, 2

AL = 2

Let's use a GA to generate tests for this method

```
void computeTriangleType() {
1 if (a == b) {
2   if (b == c)
3     type = "EQUILATERAL";
    else
4     type = "ISOSCELES";
    }
5   else if (a == c) {
6     type = "ISOSCELES";
    } else {
7     if (b == c)
8       type = "ISOSCELES";
      else
9       checkRightAngle();
      }
10  System.out.println(type);
}
```



$t=Triangle(int,int,int):$t.computeTriangleType()
@
2, 2, 2

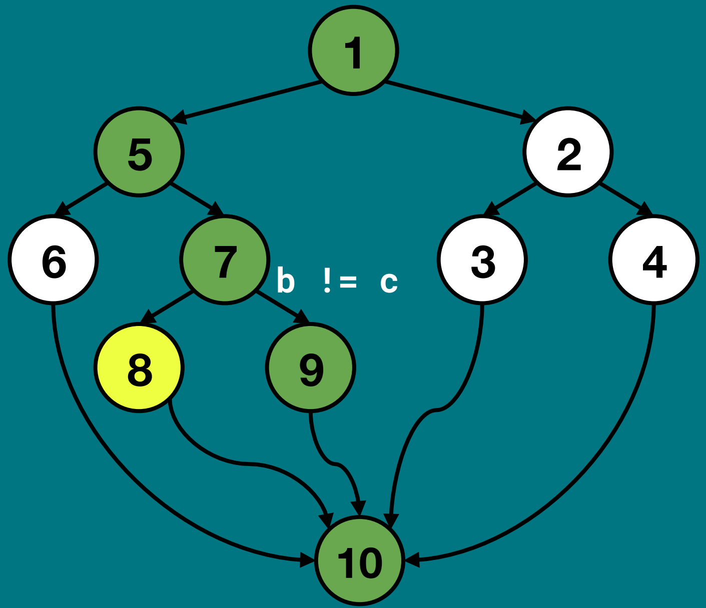
AL = 2

BD = 0.5

f(x) = 2.5

Let's use a GA to generate tests for this method

```
void computeTriangleType() {
1 if (a == b) {
2   if (b == c)
3     type = "EQUILATERAL";
    else
4     type = "ISOSCELES";
    }
5   else if (a == c) {
6     type = "ISOSCELES";
    } else {
7     if (b == c)
8       type = "ISOSCELES";
      else
9       checkRightAngle();
      }
10  System.out.println(type);
}
```



$t=Triangle(int,int,int):$t.computeTriangleType()
@
2, 3, 4

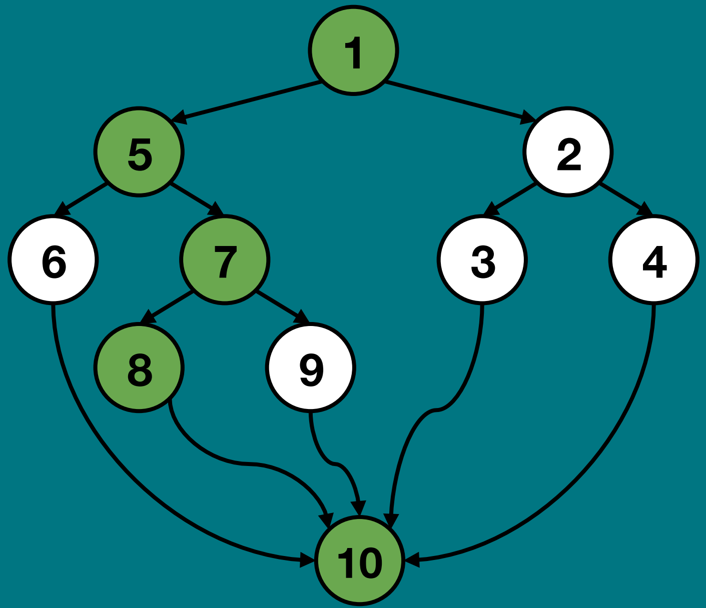
AL = 0

BD = 0.5

f(x) = 0.5

Let's use a GA to generate tests for this method

```
void computeTriangleType() {
1 if (a == b) {
2   if (b == c)
3     type = "EQUILATERAL";
    else
4     type = "ISOSCELES";
    }
5   else if (a == c) {
6     type = "ISOSCELES";
    } else {
7     if (b == c)
8       type = "ISOSCELES";
      else
9       checkRightAngle();
      }
10   System.out.println(type);
}
```



$t=Triangle(int,int,int):$t.computeTriangleType()
@
2, 3, 3

AL = 0

BD = 0

f(x) = 0

Let's use a GA to generate tests for this method

```
void computeTriangleType() {
1 if (a == b) {
2   if (b == c)
3     type = "EQUILATERAL";
    else
4     type = "ISOSCELES";
    }
5   else if (a == c) {
6     type = "ISOSCELES";
    } else {
7     if (b == c)
8       type = "ISOSCELES";
      else
9       checkRightAngle();
      }
10  System.out.println(type);
}
```

x_1 = **2, 2, 2**

x_3 = **-2, 3, 6**

x_2 = **2, 3, 4**

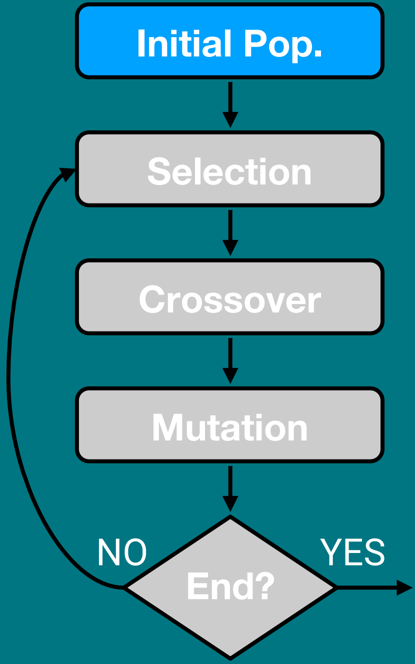
x_4 = **2, 3, 7**

x_5 = **2, 2, 5**

x_6 = **3, 4, 5**

x_7 = **3, 5, 7**

x_8 = **6, 8, 4**



Let's use a GA to generate tests for this method

```
void computeTriangleType() {
1 if (a == b) {
2   if (b == c)
3     type = "EQUILATERAL";
    else
4     type = "ISOSCELES";
    }
5   else if (a == c) {
6     type = "ISOSCELES";
    } else {
7     if (b == c)
8       type = "ISOSCELES";
      else
9       checkRightAngle();
      }
10  System.out.println(type);
}
```

Rank Selection

x_1 = **2, 2, 2**

x_2 = **2, 3, 4**

x_3 = **-2, 3, 6**

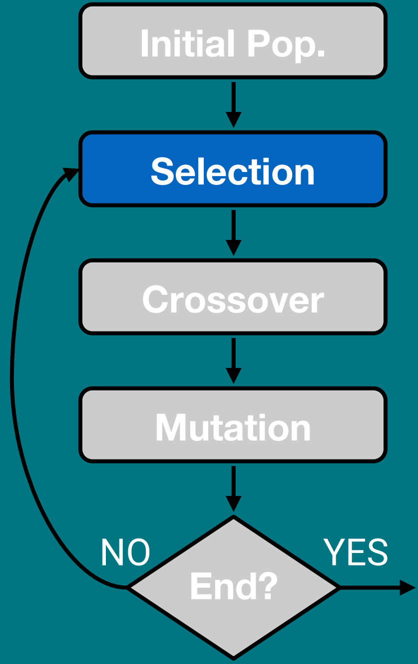
x_4 = **2, 3, 7**

x_5 = **2, 2, 5**

x_6 = **3, 4, 5**

x_7 = **3, 5, 7**

x_8 = **6, 8, 4**



Let's use a GA to generate tests for this method

```java
void computeTriangleType() {
  if (a == b) {
    if (b == c)
      type = "EQUILATERAL";
    else
      type = "ISOSCELES";
    }
    else if (a == c) {
      type = "ISOSCELES";
    } else {
      if (b == c)
        type = "ISOSCELES";
      else
        checkRightAngle();
      }
   System.out.println(type);
}
```

Single Point Crossover

$\alpha = 0.8$

x_1 = **2, 2, 2**

x_3 = **-2, 3, 6**

x_2 = **2, 4, 5**

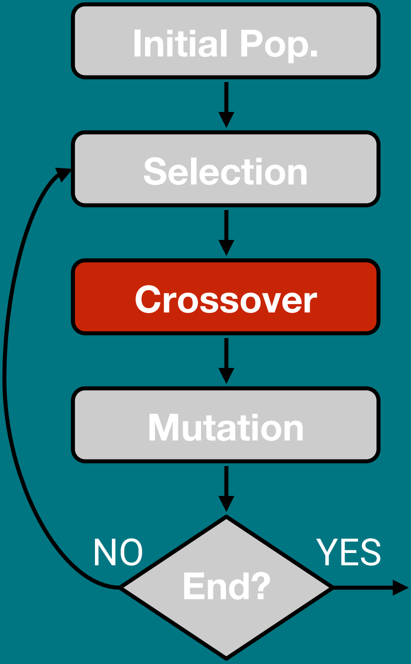
x_4 = **2, 3, 7**

x_5 = **2, 2, 5**

x_6 = **3, 3, 4**

x_7 = **3, 5, 4**

x_8 = **6, 8, 7**



Let's use a GA to generate tests for this method

```
void computeTriangleType() {
1 if (a == b) {
2   if (b == c)
3     type = "EQUILATERAL";
    else
4     type = "ISOSCELES";
    }
5   else if (a == c) {
6     type = "ISOSCELES";
    } else {
7     if (b == c)
8       type = "ISOSCELES";
      else
9       checkRightAngle();
      }
10  System.out.println(type);
}
```

Uniform Mutation

$\alpha = 0.4$

x_1 = **2, 2, 2**

x_3 = **-2, 3, 6**

x_2 = **2, 5, 5**

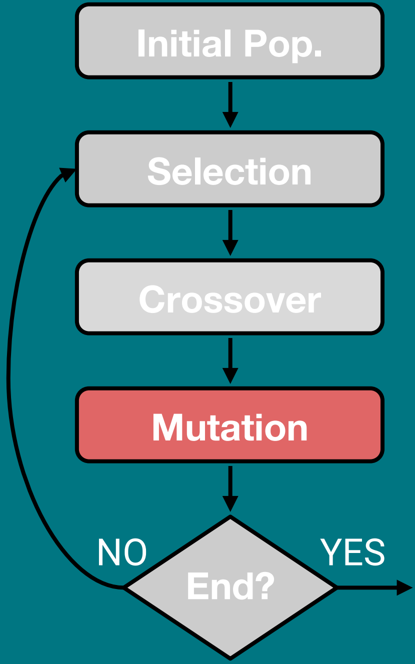
x_4 = **2, 8, 7**

x_5 = **2, 2, 5**

x_6 = **3, 3, 4**

x_7 = **3, 5, 10**

x_8 = **6, 8, 7**



Let's use a GA to generate tests for this method

```
void computeTriangleType() {
1 if (a == b) {
2   if (b == c)
3     type = "EQUILATERAL";
    else
4     type = "ISOSCELES";
    }
5   else if (a == c) {
6     type = "ISOSCELES";
    } else {
7     if (b == c)
8       type = "ISOSCELES";
      else
9       checkRightAngle();
      }
10  System.out.println(type);
}
```

x_1 = **2, 2, 2**

x_3 = **-2, 3, 6**

x_2 = **2, 5, 5**

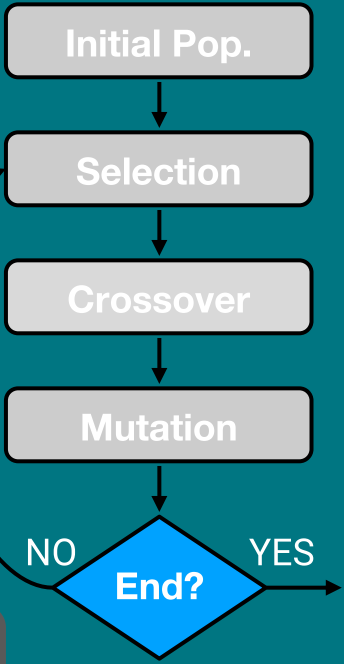
x_4 = **2, 8, 7**

x_5 = **2, 2, 5**

x_6 = **3, 3, 4**

x_7 = **3, 5, 10**

x_8 = **6, 8, 7**



Convergence reached! The evolution stops and returns the best individual

Let's use a GA to generate tests for this method

```
void computeTriangleType() {
1 if (a == b) {
2   if (b == c)
3     type = "EQUILATERAL";
    else
4     type = "ISOSCELES";
    }
5   else if (a == c) {
6     type = "ISOSCELES";
    } else {
7     if (b == c)
8       type = "ISOSCELES";
      else
9       checkRightAngle();
      }
10  System.out.println(type);
}
```

x_1 = **2, 2, 2**

x_3 = **-2, 3, 6**

x_2 = **2, 5, 5**

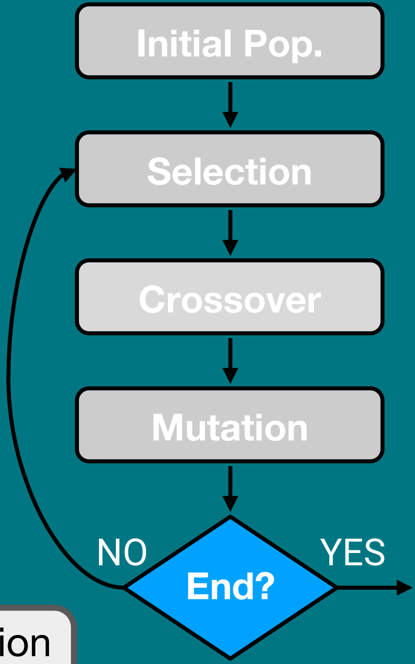
x_4 = **2, 8, 7**

x_5 = **2, 2, 5**

x_6 = **3, 3, 4**

x_7 = **3, 5, 10**

x_8 = **6, 8, 7**



Convergence reached! The evolution stops and returns the best individual

Now we can repeat the entire process selecting a different coverage target.

Use Cases of ATCG

Making the System Crash

Facilitate the Tester's Job

Supporting Debugging

Use Cases of ATCG

Making the System Crash

Facilitate the Tester's Job

Supporting Debugging

Drawbacks of ATCG

The Oracle Problem

Test Code Quality

Setting the Metaheuristic

Automatic Test Case Generation: Toward Its Application in Exploit Generation for Known Vulnerabilities

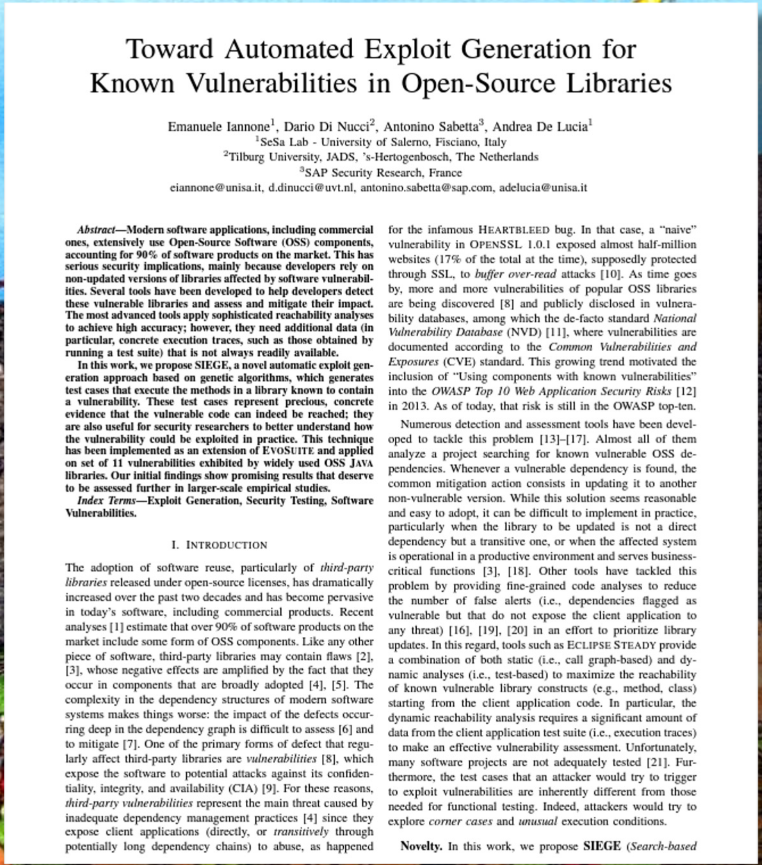
Automatic Test Case Generation: Toward Its Application in Exploit Generation for Known Vulnerabilities

~~Discovering Vulnerabilities?~~

Known Vulnerabilities Assessment

Generate Tests!

Automatic Test Case Generation: Toward Its Application in Exploit Generation for Known Vulnerabilities



Toward Automated Exploit Generation for Known Vulnerabilities in Open-Source Libraries

Emanuele Iannone[1], Dario Di Nucci[2], Antonino Sabetta[3], Andrea De Lucia[1]
[1]SeSa Lab - University of Salerno, Fisciano, Italy
[2]Tilburg University, JADS, 's-Hertogenbosch, The Netherlands
[3]SAP Security Research, France
eiannone@unisa.it, d.dinucci@uvt.nl, antonino.sabetta@sap.com, adelucia@unisa.it

***Abstract*—Modern software applications, including commercial ones, extensively use Open-Source Software (OSS) components, accounting for 90% of software products on the market. This has serious security implications, mainly because developers rely on non-updated versions of libraries affected by software vulnerabilities. Several tools have been developed to help developers detect these vulnerable libraries and assess and mitigate their impact. The most advanced tools apply sophisticated reachability analyses to achieve high accuracy; however, they need additional data (in particular, concrete execution traces, such as those obtained by running a test suite) that is not always readily available.**

In this work, we propose SIEGE, a novel automatic exploit generation approach based on genetic algorithms, which generates test cases that execute the methods in a library known to contain a vulnerability. These test cases represent precious, concrete evidence that the vulnerable code can indeed be reached; they are also useful for security researchers to better understand how the vulnerability could be exploited in practice. This technique has been implemented as an extension of EvoSuite and applied on set of 11 vulnerabilities exhibited by widely used OSS Java libraries. Our initial findings show promising results that deserve to be assessed further in larger-scale empirical studies.

***Index Terms*—Exploit Generation, Security Testing, Software Vulnerabilities.**

I. Introduction

The adoption of software reuse, particularly of *third-party libraries* released under open-source licenses, has dramatically increased over the past two decades and has become pervasive in today's software, including commercial products. Recent analyses [1] estimate that over 90% of software products on the market include some form of OSS components. Like any other piece of software, third-party libraries may contain flaws [2], [3], whose negative effects are amplified by the fact that they occur in components that are broadly adopted [4], [5]. The complexity in the dependency structures of modern software systems makes things worse: the impact of the defects occurring deep in the dependency graph is difficult to assess [6] and to mitigate [7]. One of the primary forms of defect that regularly affect third-party libraries are *vulnerabilities* [8], which expose the software to potential attacks against its confidentiality, integrity, and availability (CIA) [9]. For these reasons, *third-party vulnerabilities* represent the main threat caused by inadequate dependency management practices [4] since they expose client applications (directly, or *transitively* through potentially long dependency chains) to abuse, as happened for the infamous Heartbleed bug. In that case, a "naive" vulnerability in OpenSSL 1.0.1 exposed almost half-million websites (17% of the total at the time), supposedly protected through SSL, to *buffer over-read* attacks [10]. As time goes by, more and more vulnerabilities of popular OSS libraries are being discovered [8] and publicly disclosed in vulnerability databases, among which the de-facto standard *National Vulnerability Database* (NVD) [11], where vulnerabilities are documented according to the *Common Vulnerabilities and Exposures* (CVE) standard. This growing trend motivated the inclusion of "Using components with known vulnerabilities" into the *OWASP Top 10 Web Application Security Risks* [12] in 2013. As of today, that risk is still in the OWASP top-ten.

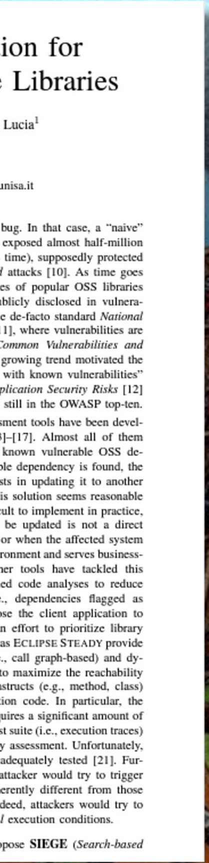
Numerous detection and assessment tools have been developed to tackle this problem [13]–[17]. Almost all of them analyze a project searching for known vulnerable OSS dependencies. Whenever a vulnerable dependency is found, the common mitigation action consists in updating it to another non-vulnerable version. While this solution seems reasonable and easy to adopt, it can be difficult to implement in practice, particularly when the library to be updated is not a direct dependency but a transitive one, or when the affected system is operational in a productive environment and serves business-critical functions [3], [18]. Other tools have tackled this problem by providing fine-grained code analyses to reduce the number of false alerts (i.e., dependencies flagged as vulnerable but that do not expose the client application to any threat) [16], [19], [20] in an effort to prioritize library updates. In this regard, tools such as Eclipse Steady provide a combination of both static (i.e., call graph-based) and dynamic analyses (i.e., test-based) to maximize the reachability of known vulnerable library constructs (e.g., method, class) starting from the client application code. In particular, the dynamic reachability analysis requires a significant amount of data from the client application test suite (i.e., execution traces) to make an effective vulnerability assessment. Unfortunately, many software projects are not adequately tested [21]. Furthermore, the test cases that an attacker would try to trigger to exploit vulnerabilities are inherently different from those needed for functional testing. Indeed, attackers would try to explore *corner cases* and *unusual* execution conditions.

Novelty. In this work, we propose **SIEGE** (*Search-based*

Toward Automated Exploit Generation for Known Vulnerabilities in Open-Source Libraries

E. Iannone, D. Di Nucci, A. Sabetta, A. De Lucia.

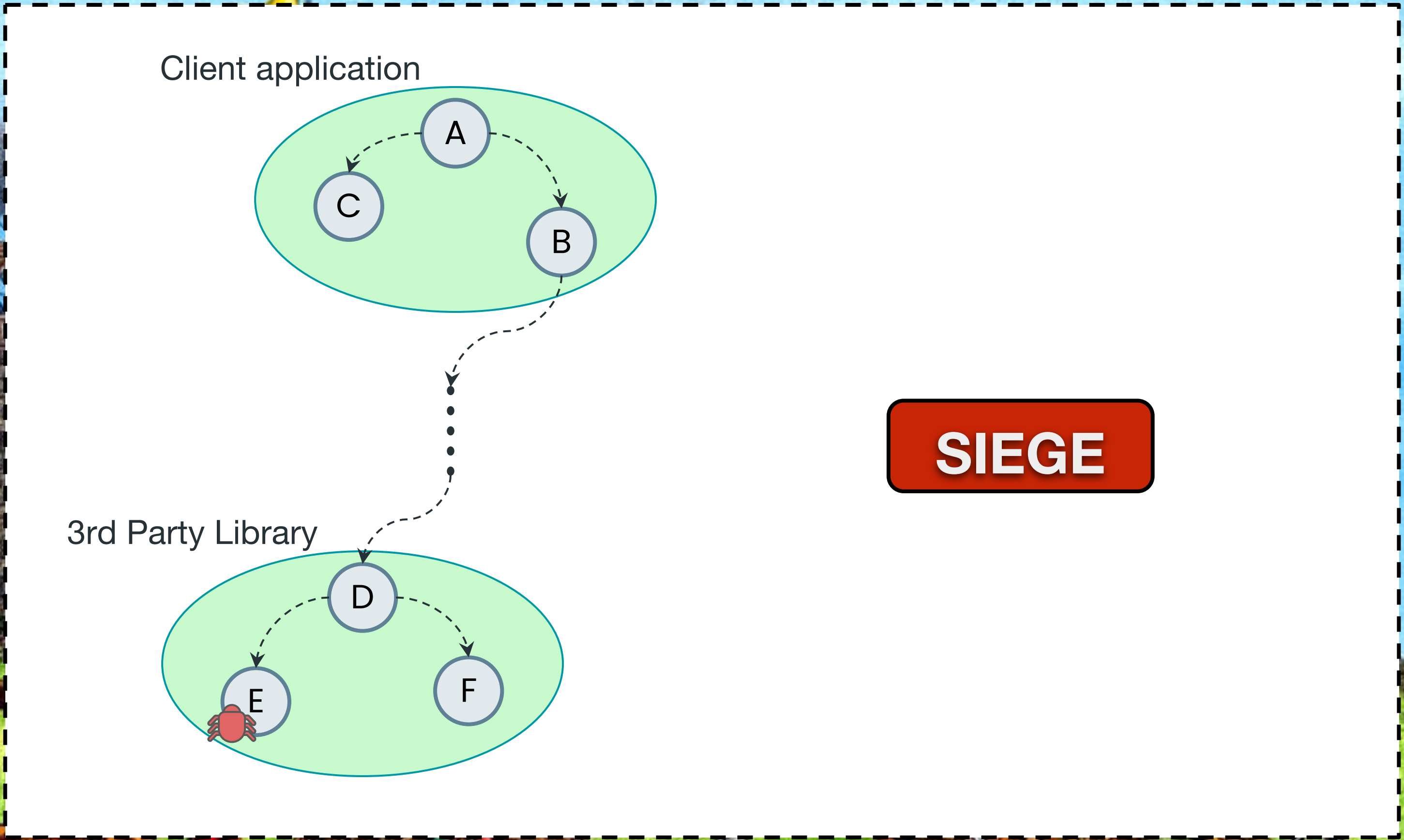
In: Proceedings of the 29th IEEE/ACM International Conference on Program Comprehension (ICPC), 2021.


ion for
Libraries
Lucia[1]
unisa.it

bug. In that case, a "naive" exposed almost half-million time), supposedly protected attacks [10]. As time goes es of popular OSS libraries blicly disclosed in vulnera- e de-facto standard *National* 1], where vulnerabilities are *ommon Vulnerabilities and* growing trend motivated the with known vulnerabilities" *plication Security Risks* [12] still in the OWASP top-ten.

sment tools have been devel-]–[17]. Almost all of them known vulnerable OSS de- ble dependency is found, the sts in updating it to another is solution seems reasonable ult to implement in practice, be updated is not a direct or when the affected system ronment and serves business- her tools have tackled this ed code analyses to reduce ., dependencies flagged as se the client application to n effort to prioritize library as ECLIPSE STEADY provide ., call graph-based) and dy- to maximize the reachability structs (e.g., method, class) ion code. In particular, the uires a significant amount of st suite (i.e., execution traces) y assessment. Unfortunately, adequately tested [21]. Fur- attacker would try to trigger erently different from those deed, attackers would try to *l* execution conditions.

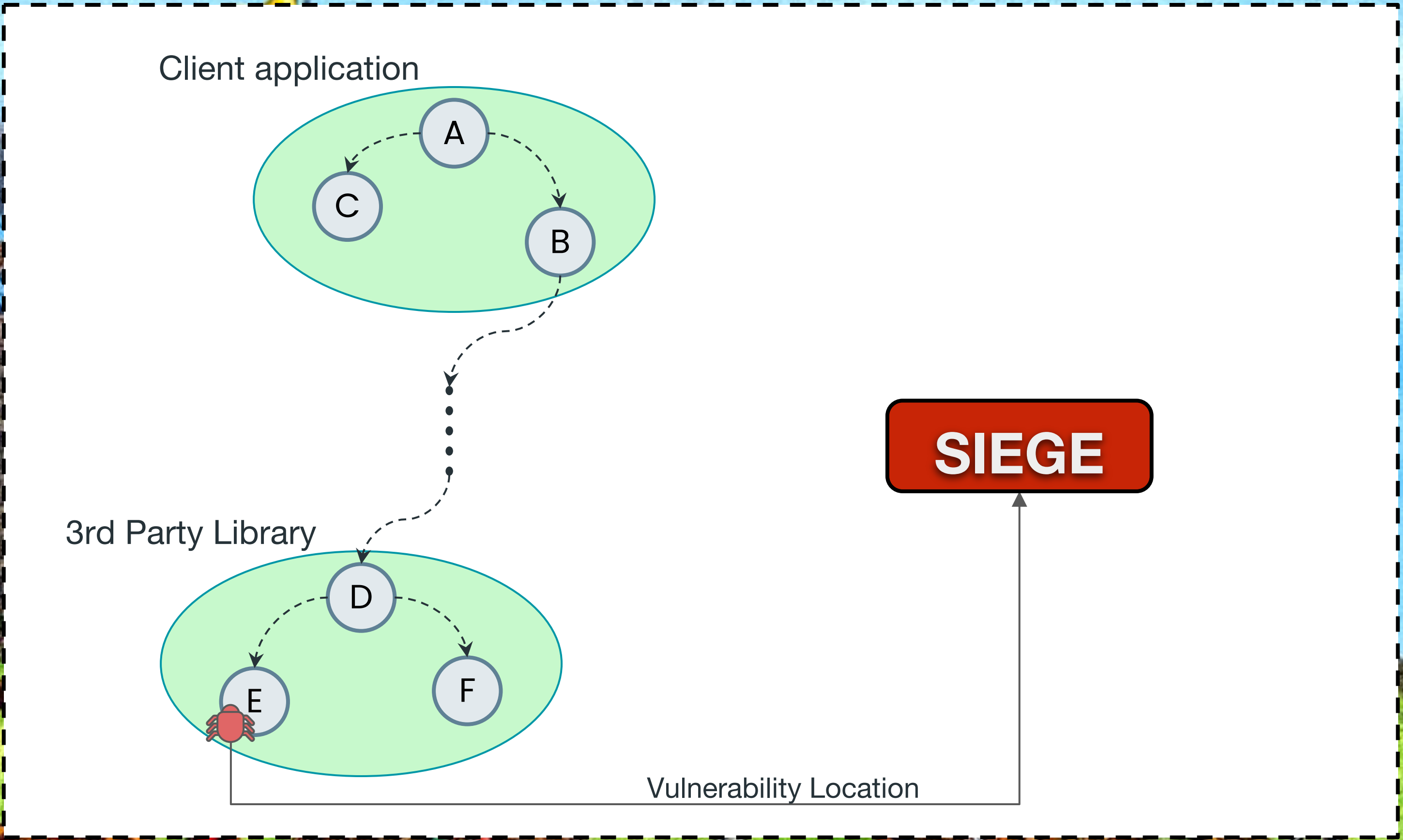
pose **SIEGE** (*Search-based*



Toward Automated Exploit Generation for Known Vulnerabilities in Open-Source Libraries

E. Iannone, D. Di Nucci, A. Sabetta, A. De Lucia.

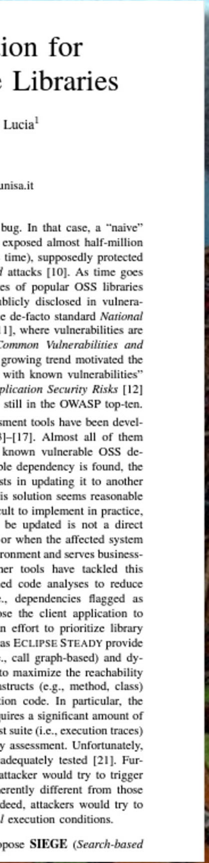
In: Proceedings of the 29th IEEE/ACM International Conference on Program Comprehension (ICPC), 2021.

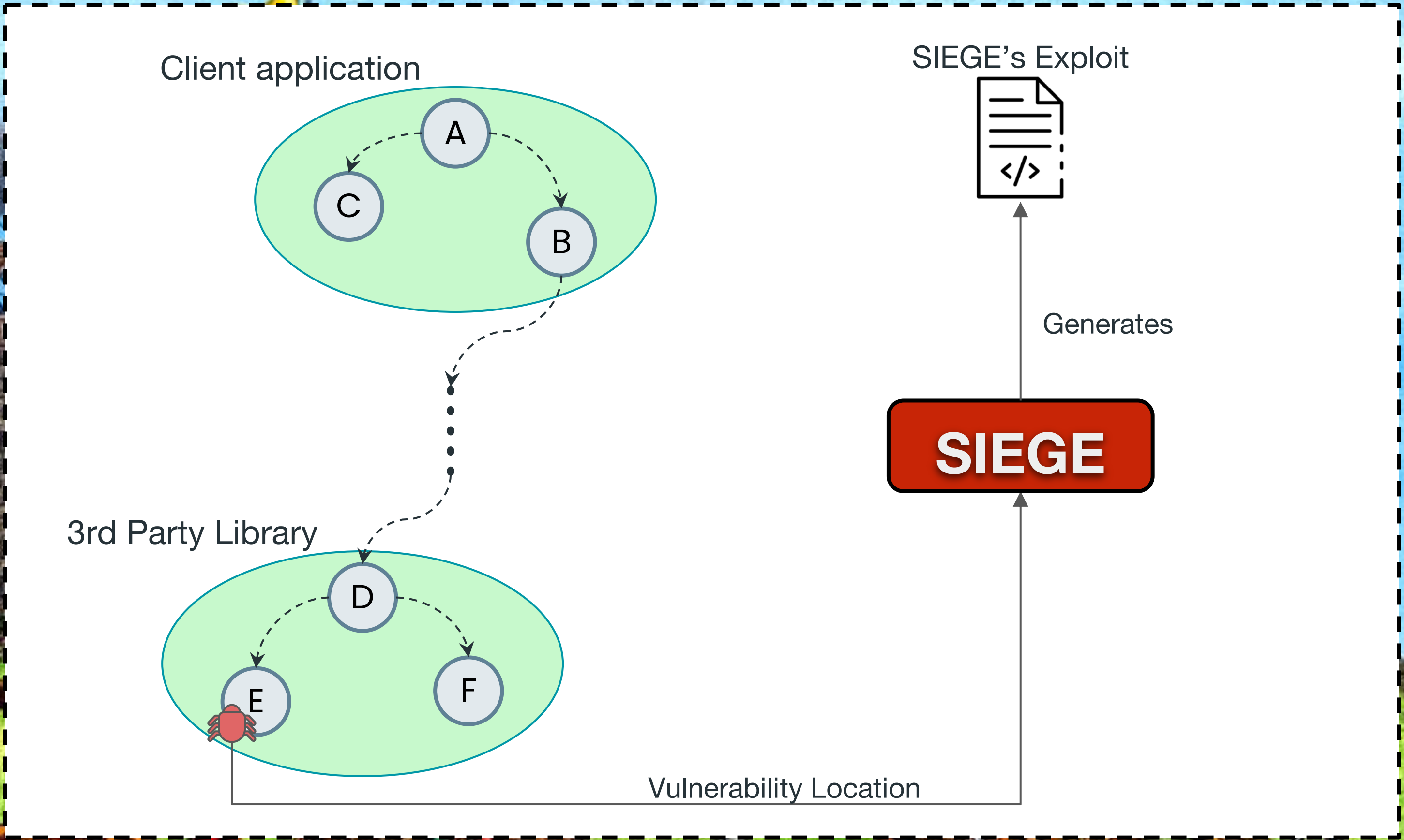


Toward Automated Exploit Generation for Known Vulnerabilities in Open-Source Libraries

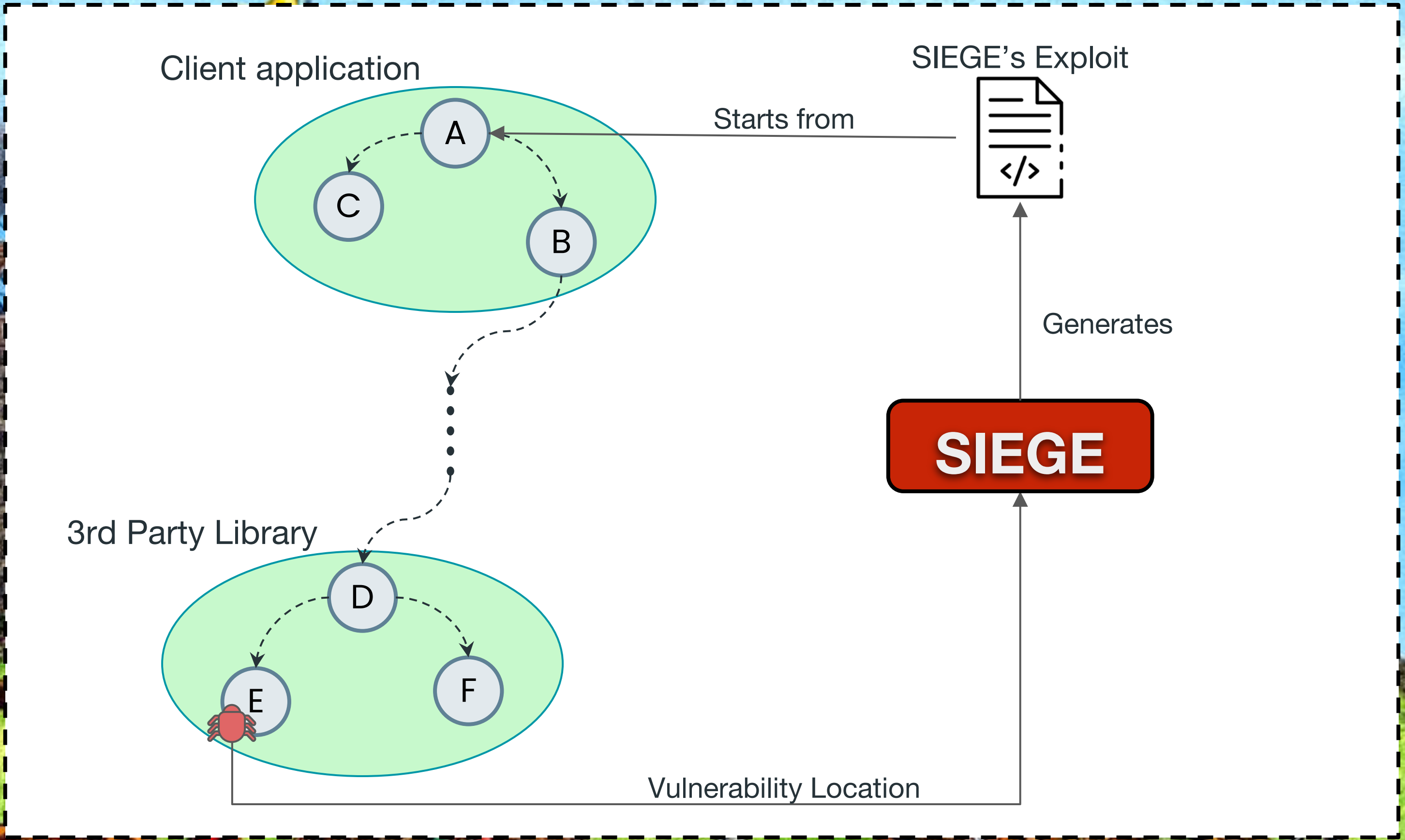
E. Iannone, D. Di Nucci, A. Sabetta, A. De Lucia.

In: Proceedings of the 29th IEEE/ACM International Conference on Program Comprehension (ICPC), 2021.


ion for
Libraries
Lucia[1]
unisa.it
bug. In that case, a "naive"
exposed almost half-million
time), supposedly protected
attacks [10]. As time goes
es of popular OSS libraries
blicly disclosed in vulnera-
e de-facto standard *National*
1], where vulnerabilities are
Common Vulnerabilities and
growing trend motivated the
with known vulnerabilities"
plication Security Risks [12]
still in the OWASP top-ten.
sment tools have been devel-
]–[17]. Almost all of them
known vulnerable OSS de-
ble dependency is found, the
sts in updating it to another
is solution seems reasonable
ult to implement in practice,
be updated is not a direct
or when the affected system
ronment and serves business-
her tools have tackled this
ed code analyses to reduce
., dependencies flagged as
se the client application to
n effort to prioritize library
as ECLIPSE STEADY provide
., call graph-based) and dy-
to maximize the reachability
structs (e.g., method, class)
ion code. In particular, the
uires a significant amount of
st suite (i.e., execution traces)
y assessment. Unfortunately,
adequately tested [21]. Fur-
attacker would try to trigger
erently different from those
deed, attackers would try to
l execution conditions.
pose **SIEGE** (*Search-based*



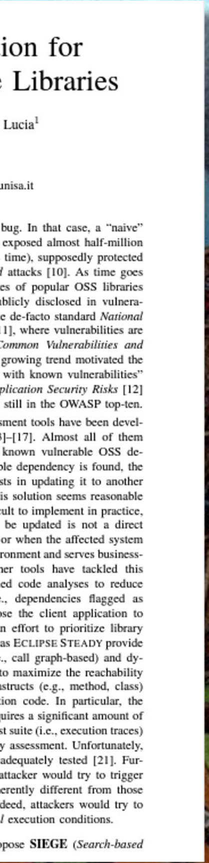
Toward Automated Exploit Generation for Known Vulnerabilities in Open-Source Libraries
E. Iannone, D. Di Nucci, A. Sabetta, A. De Lucia.
In: Proceedings of the 29th IEEE/ACM International Conference on Program Comprehension (ICPC), 2021.

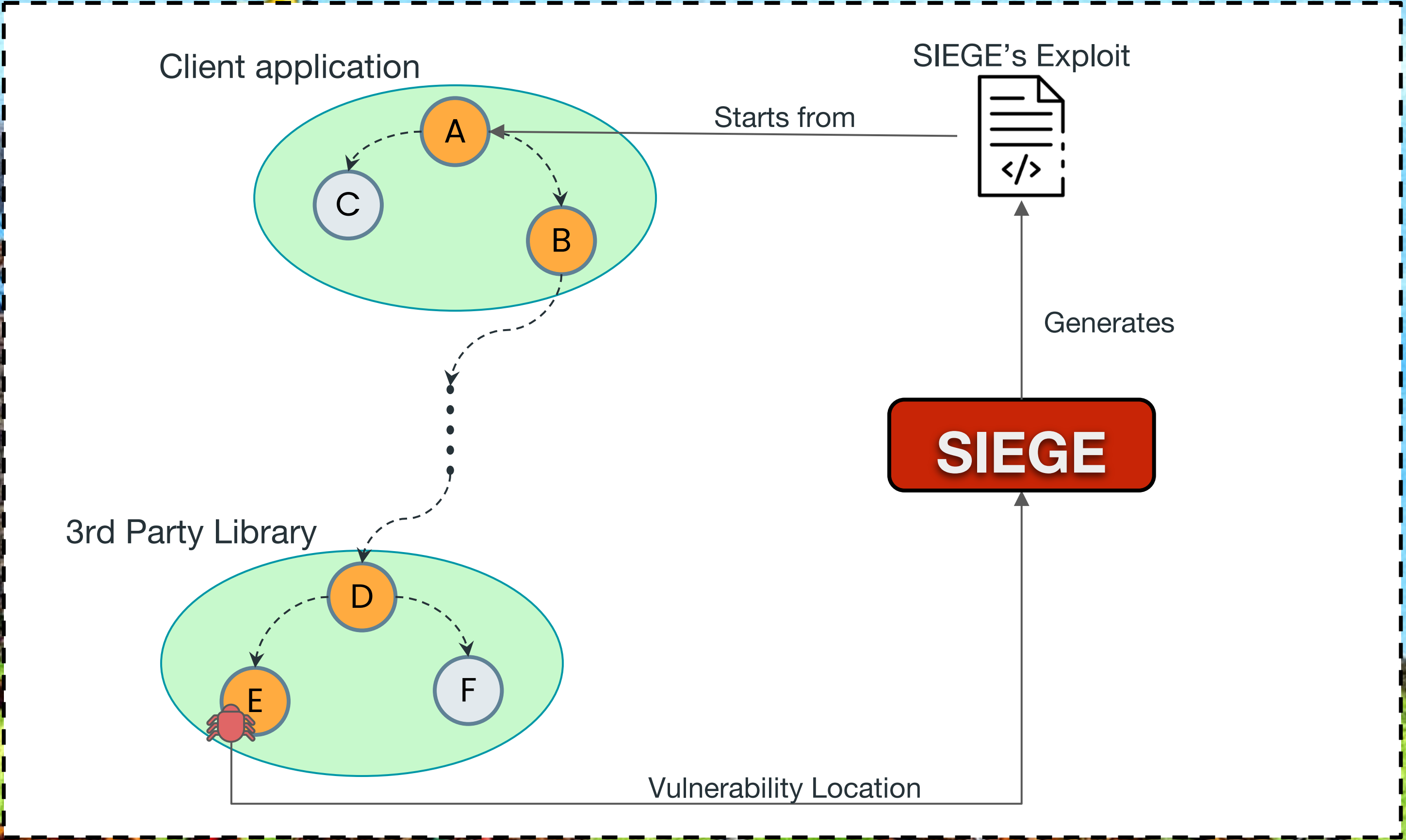


Toward Automated Exploit Generation for Known Vulnerabilities in Open-Source Libraries

E. Iannone, D. Di Nucci, A. Sabetta, A. De Lucia.

In: Proceedings of the 29th IEEE/ACM International Conference on Program Comprehension (ICPC), 2021.

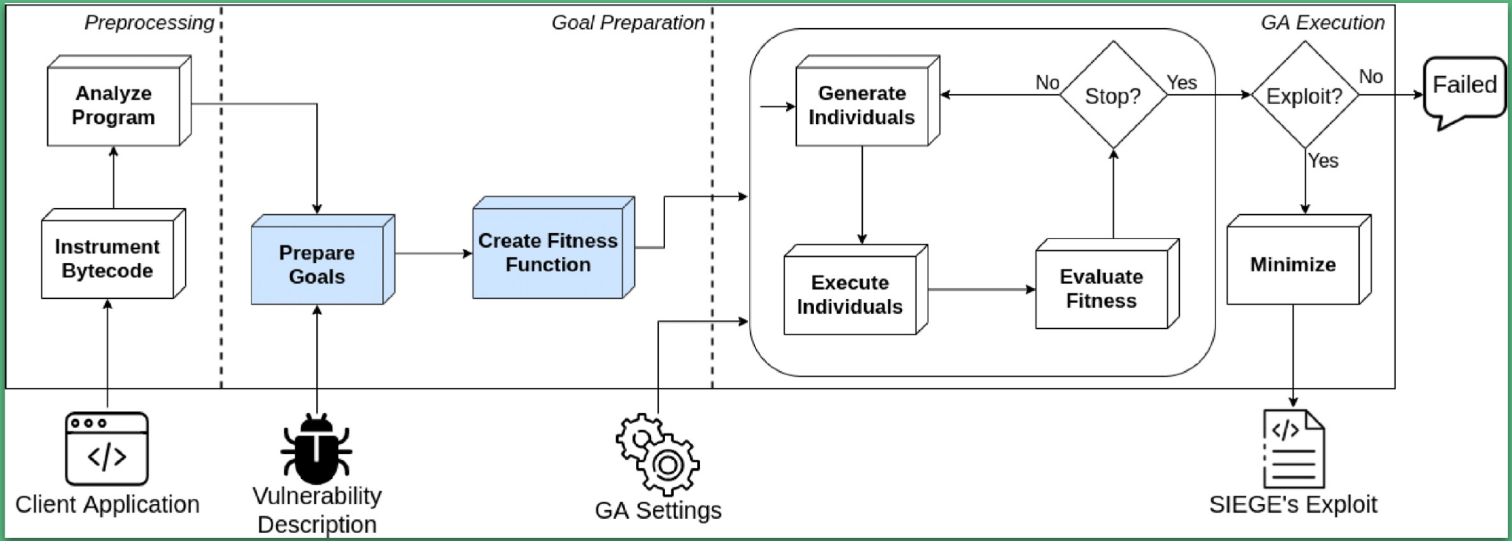

ion for
Libraries
Lucia[1]
unisa.it
bug. In that case, a "naive"
exposed almost half-million
time), supposedly protected
attacks [10]. As time goes
es of popular OSS libraries
blicly disclosed in vulnera-
e de-facto standard *National*
1], where vulnerabilities are
Common Vulnerabilities and
growing trend motivated the
with known vulnerabilities"
plication Security Risks [12]
still in the OWASP top-ten.
sment tools have been devel-
]–[17]. Almost all of them
known vulnerable OSS de-
ble dependency is found, the
sts in updating it to another
is solution seems reasonable
ult to implement in practice,
be updated is not a direct
or when the affected system
ronment and serves business-
ner tools have tackled this
ed code analyses to reduce
., dependencies flagged as
se the client application to
n effort to prioritize library
as ECLIPSE STEADY provide
., call graph-based) and dy-
to maximize the reachability
structs (e.g., method, class)
tion code. In particular, the
uires a significant amount of
st suite (i.e., execution traces)
y assessment. Unfortunately,
adequately tested [21]. Fur-
attacker would try to trigger
erently different from those
deed, attackers would try to
l execution conditions.
pose **SIEGE** (*Search-based*

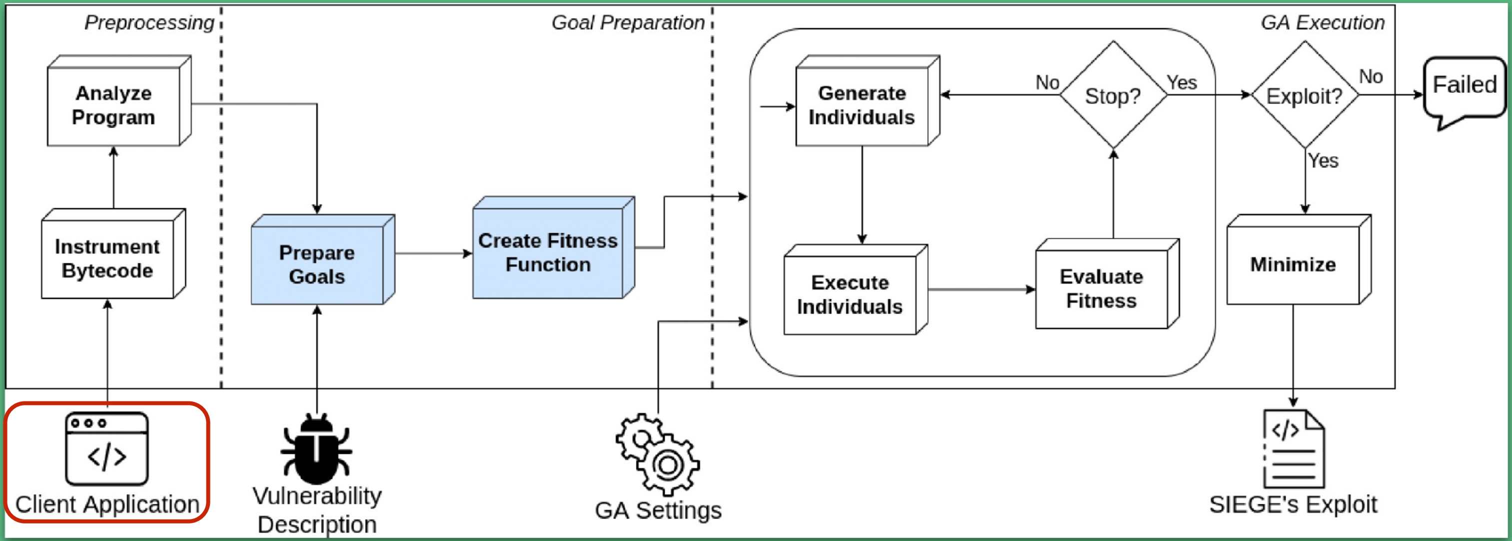


Toward Automated Exploit Generation for Known Vulnerabilities in Open-Source Libraries

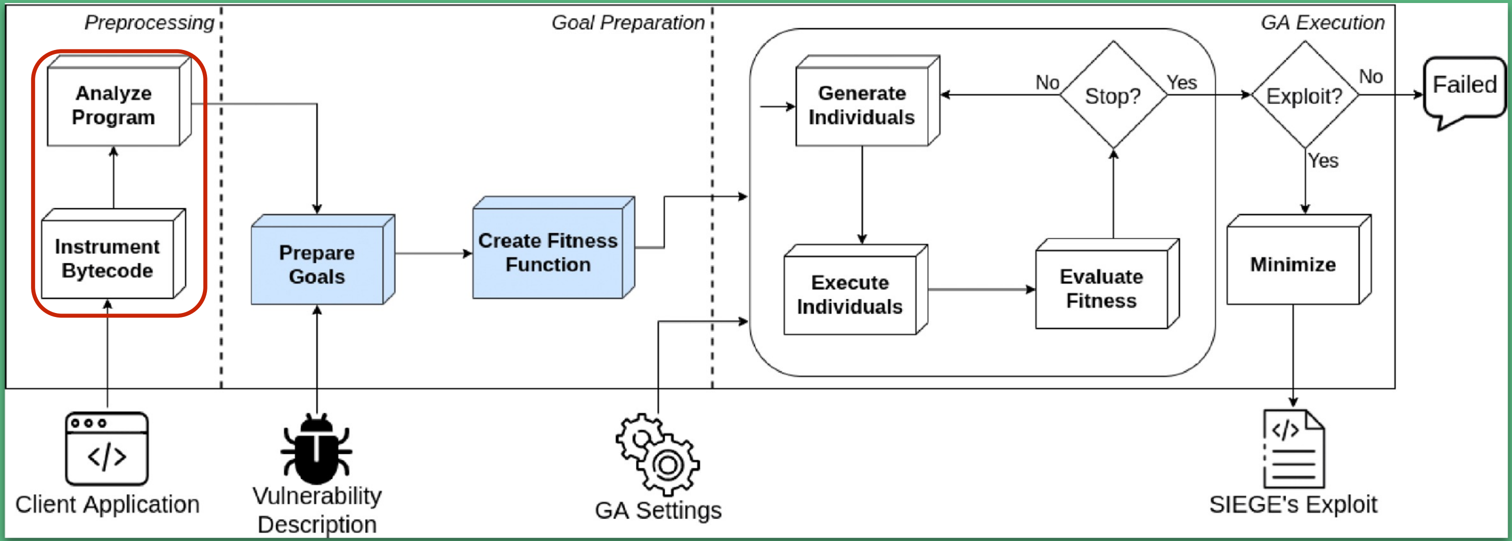
E. Iannone, D. Di Nucci, A. Sabetta, A. De Lucia.

In: Proceedings of the 29th IEEE/ACM International Conference on Program Comprehension (ICPC), 2021.


Preprocessing
Goal Preparation
GA Execution
Analyze Program
Instrument Bytecode
Prepare Goals
Create Fitness Function
Generate Individuals
Execute Individuals
Evaluate Fitness
Stop?
No
Yes
Exploit?
No
Yes
Failed
Minimize
Client Application
Vulnerability Description
GA Settings
SIEGE's Exploit

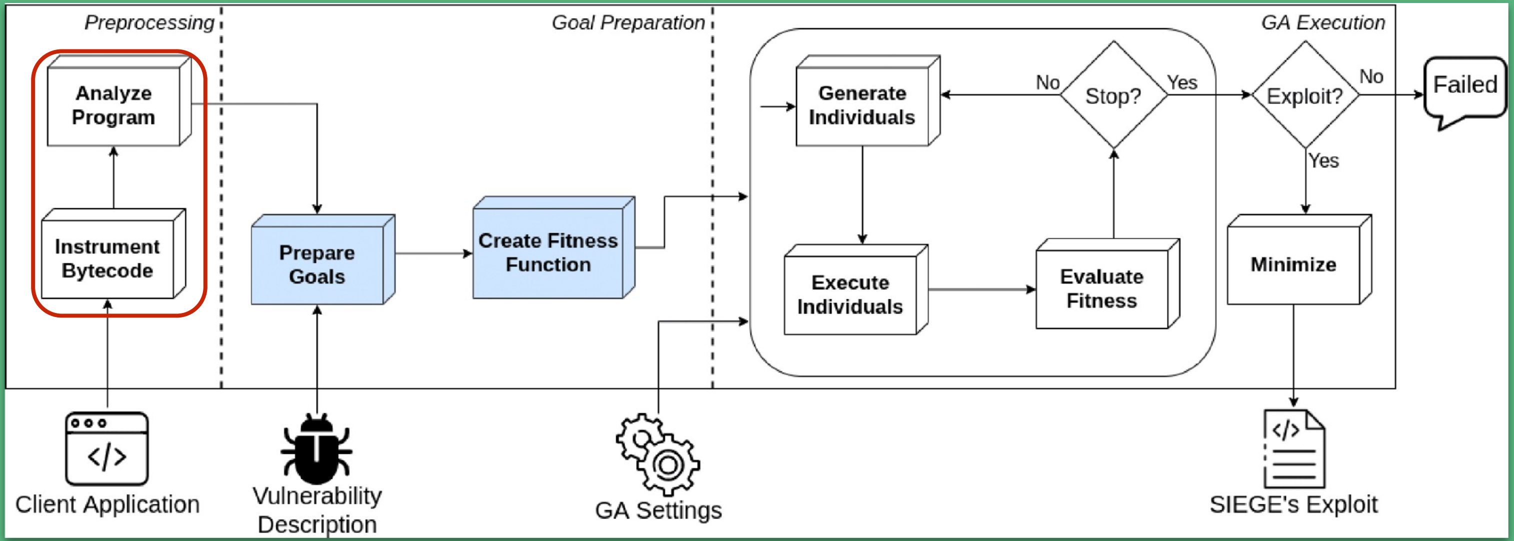


SIEGE runs on an arbitrary Java application that includes vulnerable dependencies



SIEGE runs on an arbitrary Java application that includes vulnerable dependencies

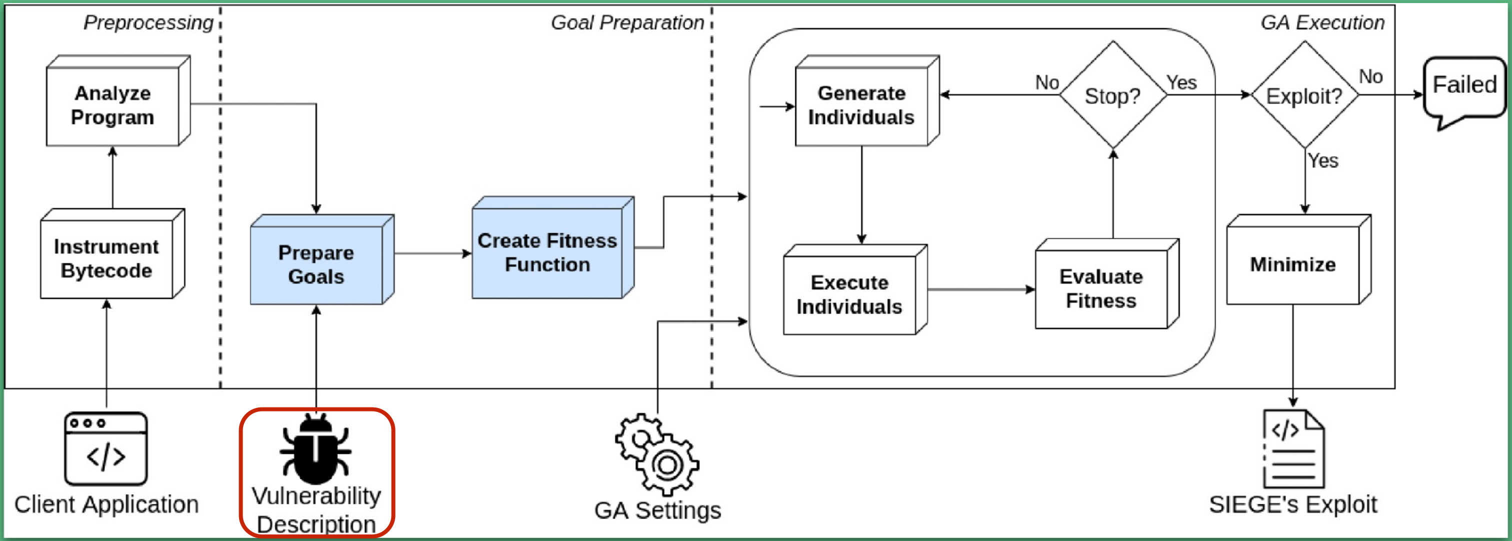
SIEGE extracts the entire **classpath call graph** and the **control flow graphs**



SIEGE runs on an arbitrary Java application that includes vulnerable dependencies

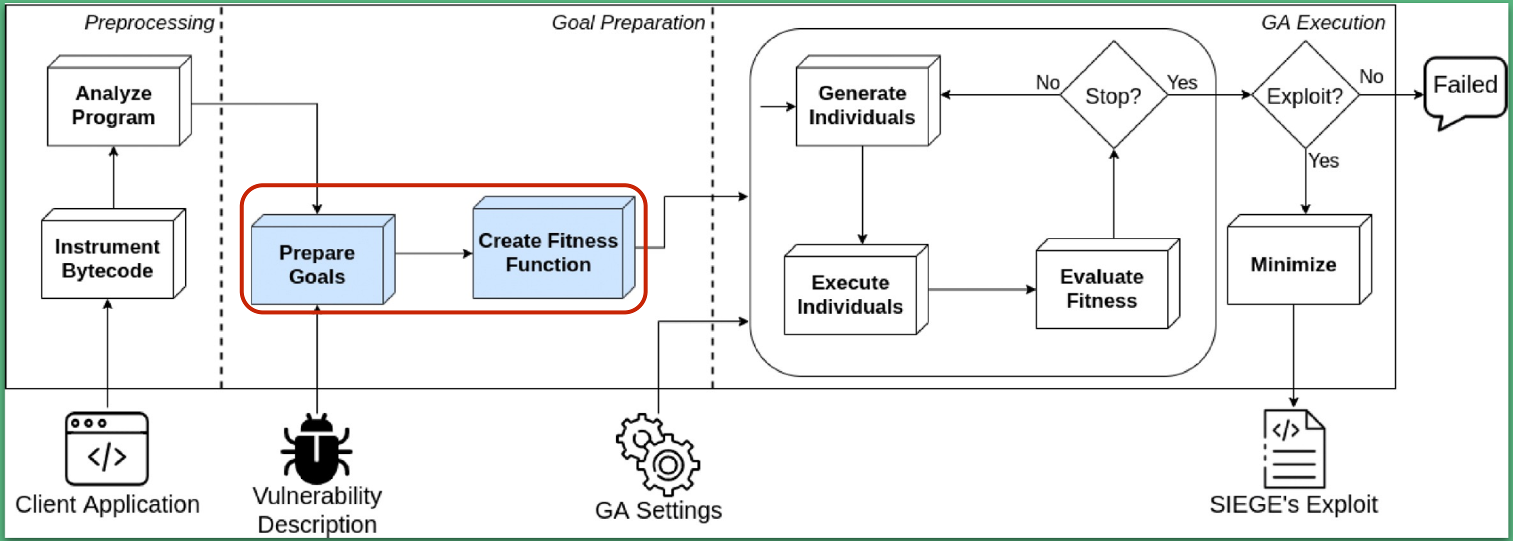
SIEGE extracts the entire **classpath call graph** and the **control flow graphs**

SIEGE largely reuses of **EvoSuite** features: program analysis, bytecode instrumentation, ATCG infrastructure, test execution engine.



SIEGE needs to locate the target vulnerable construct:

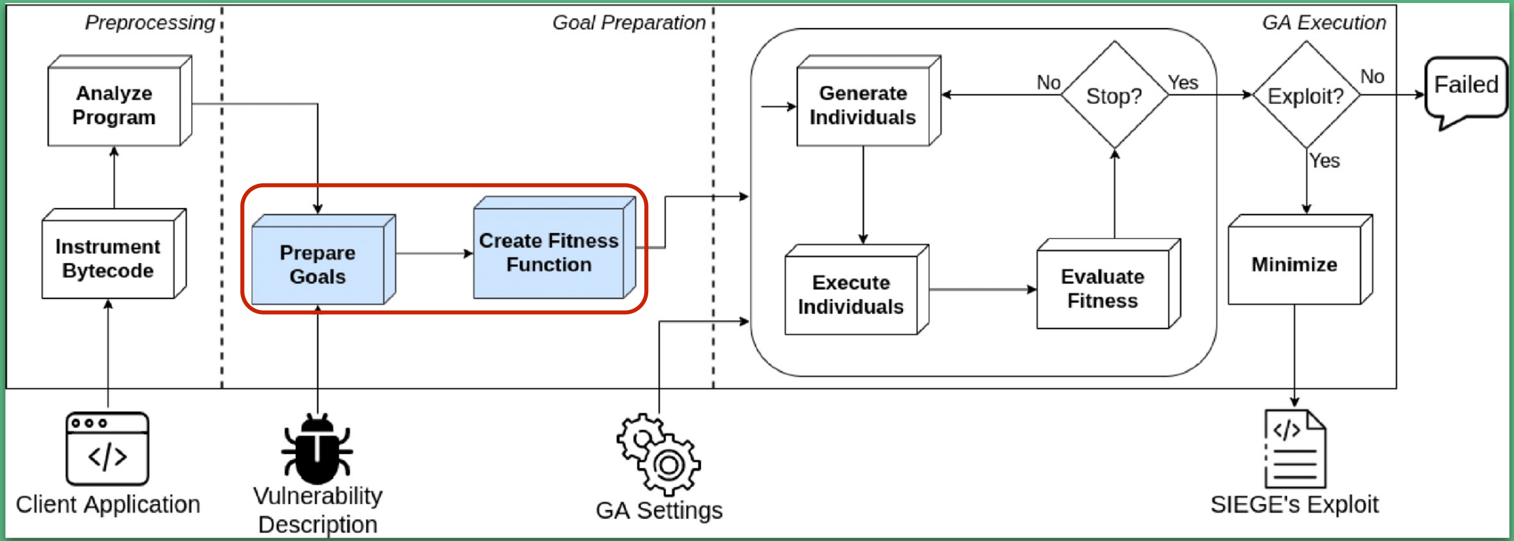
1. Class name
2. Method name
3. Line number



SIEGE needs to locate the target vulnerable construct:

(1) Class name
(2) Method name
(3) Line number

Prepare the fitness function that rewards the test cases that are closer to the target line



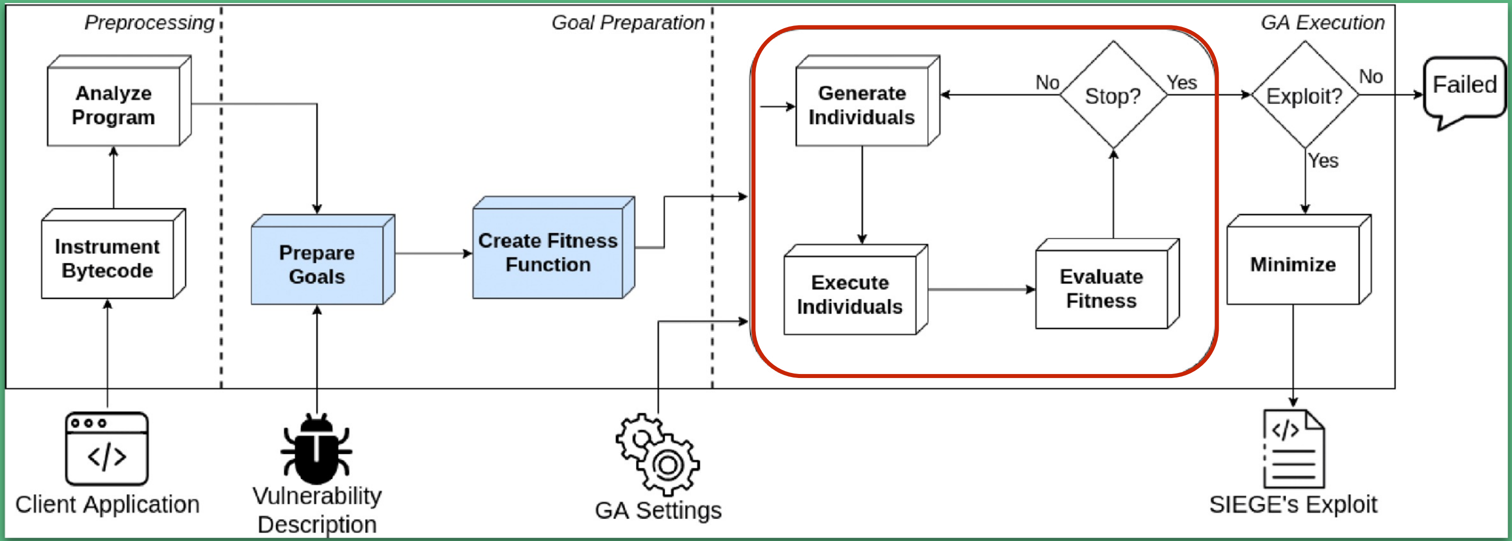
SIEGE needs to locate the target vulnerable construct:

(1) Class name
(2) Method name
(3) Line number

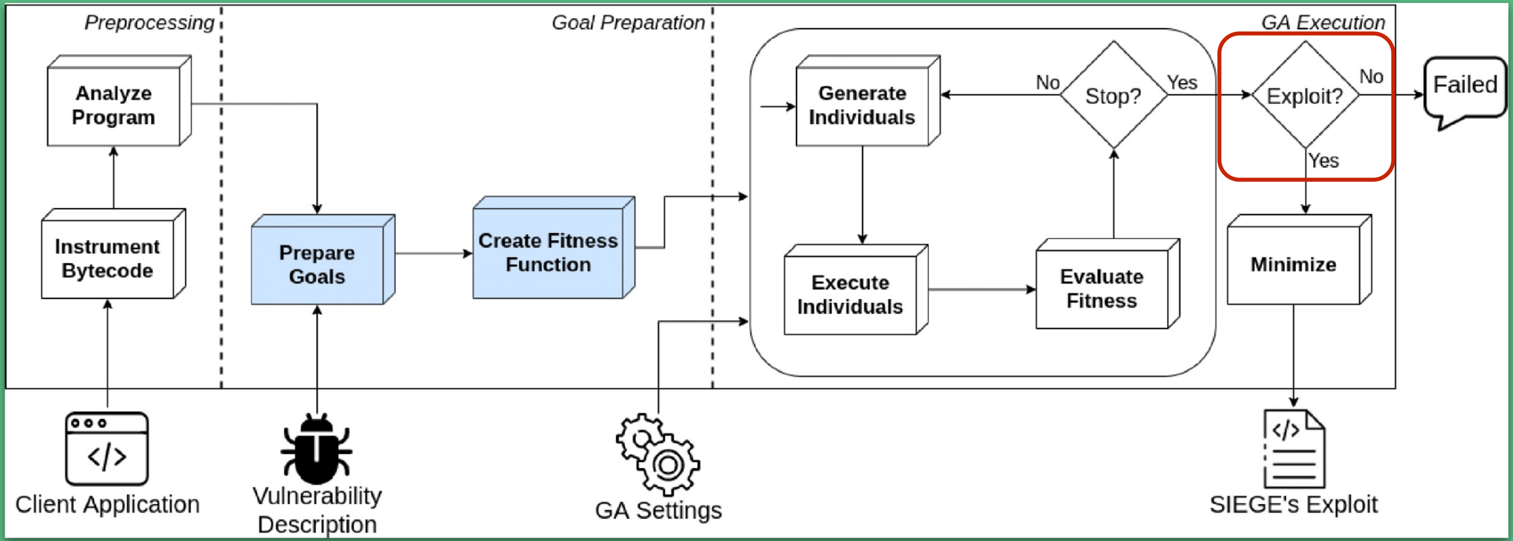
Prepare the fitness function that rewards the test cases that are closer to the target line

```java
public void process(final HttpRequest request, final HttpContext context) {
  if (request == null) {
    throw new IllegalArgumentException("HTTP request may not be null");
  }
  if (context == null) {
    throw new IllegalArgumentException("HTTP context may not be null");
  }

  if (request.containsHeader(AUTH.PROXY_AUTH_RESP)) {
    return;
  }

  // Obtain authentication state
  AuthState authState = (AuthState) context.getAttribute(
    ClientContext.PROXY_AUTH_STATE);
...
}
```

CVE-2011-1498

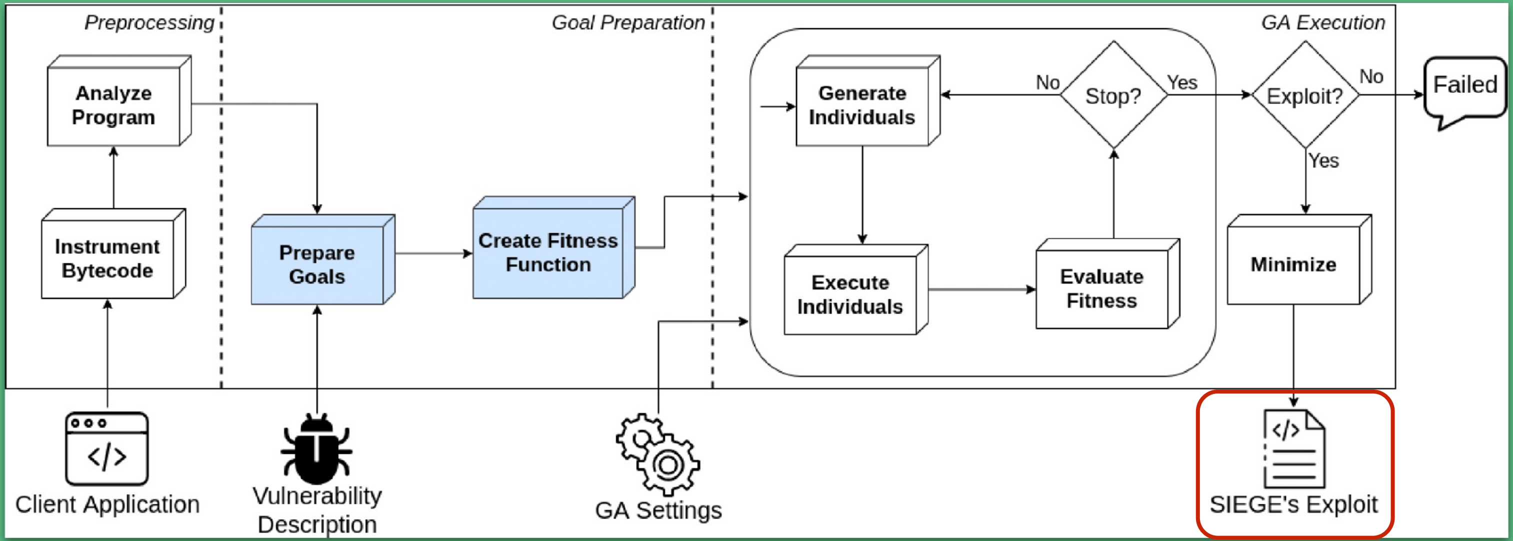


A population of JUnit test cases is evolved with a GA...



A population of JUnit test cases is evolved with a GA...

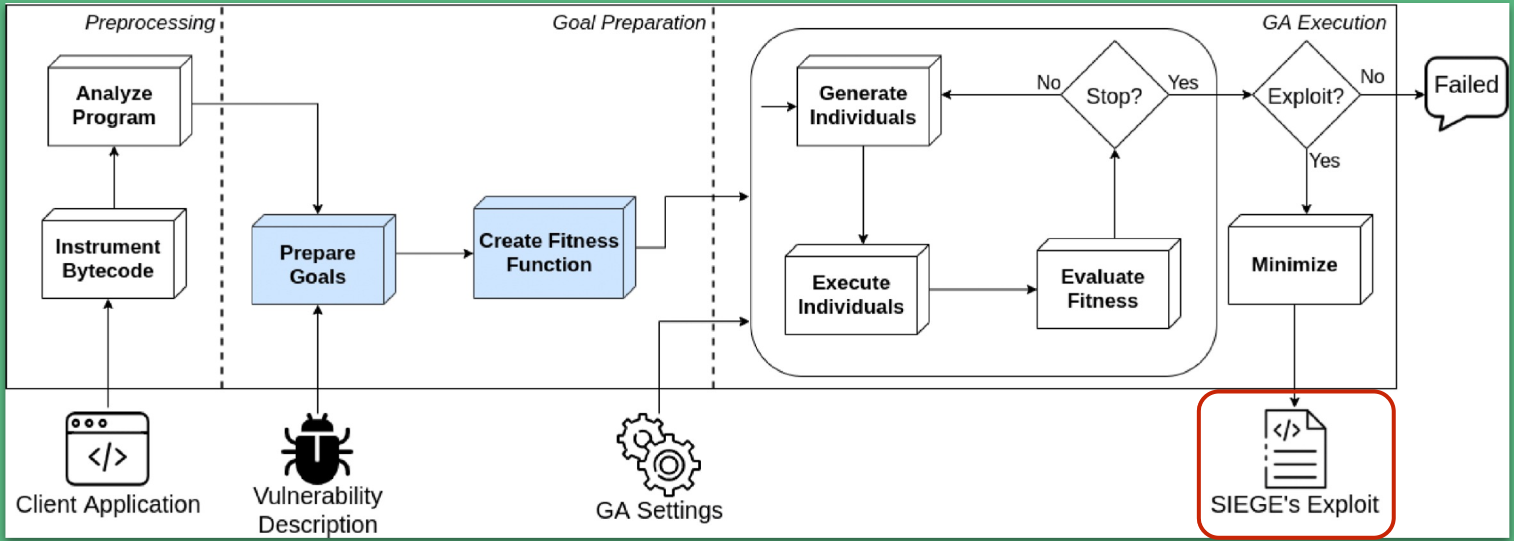
...if a test case covers the target vulnerable line...



A population of JUnit test cases is evolved with a GA...

...if a test case covers the target vulnerable line...

...it is considered an **exploit**!



Exploit for CVE-2011-1498

```
public void test0() throws Throwable {
  CallingClient1 callingClient1_0 = new CallingClient1();
  BasicHttpRequest basicHttpRequest0 =
    new BasicHttpRequest("", "");
  BasicHttpContext basicHttpContext0 =
    new BasicHttpContext((HttpContext) null);
  callingClient1_0.call(basicHttpRequest0,
basicHttpContext0);
}
```

Exploratory Evaluation

Does SIEGE succeed in generating exploits of third-party vulnerabilities included within client applications?

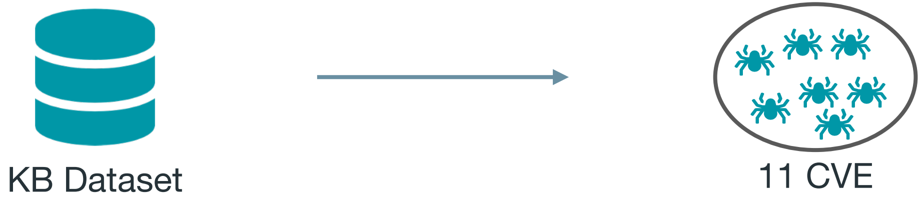
Exploratory Evaluation

Does SIEGE succeed in generating exploits of third-party vulnerabilities included within client applications?

KB Dataset

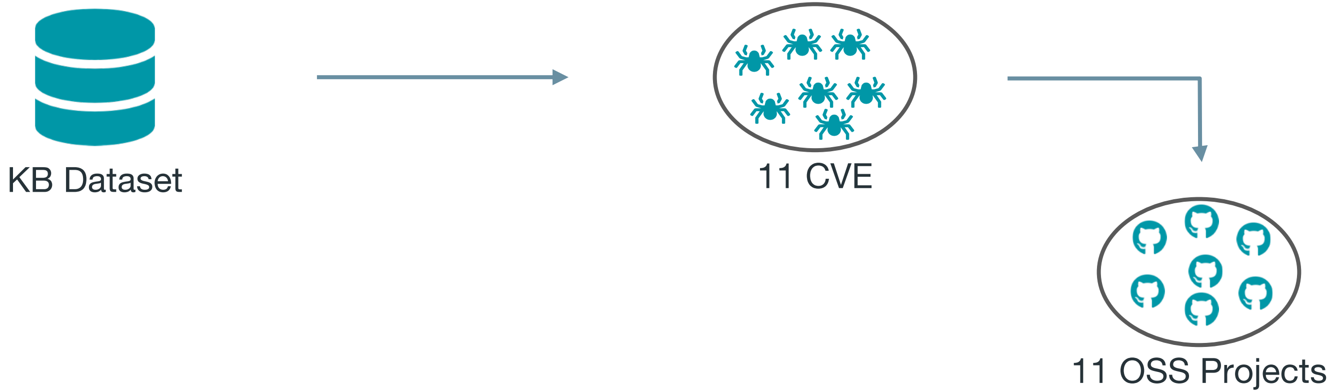
Exploratory Evaluation

Does SIEGE succeed in generating exploits of third-party vulnerabilities included within client applications?



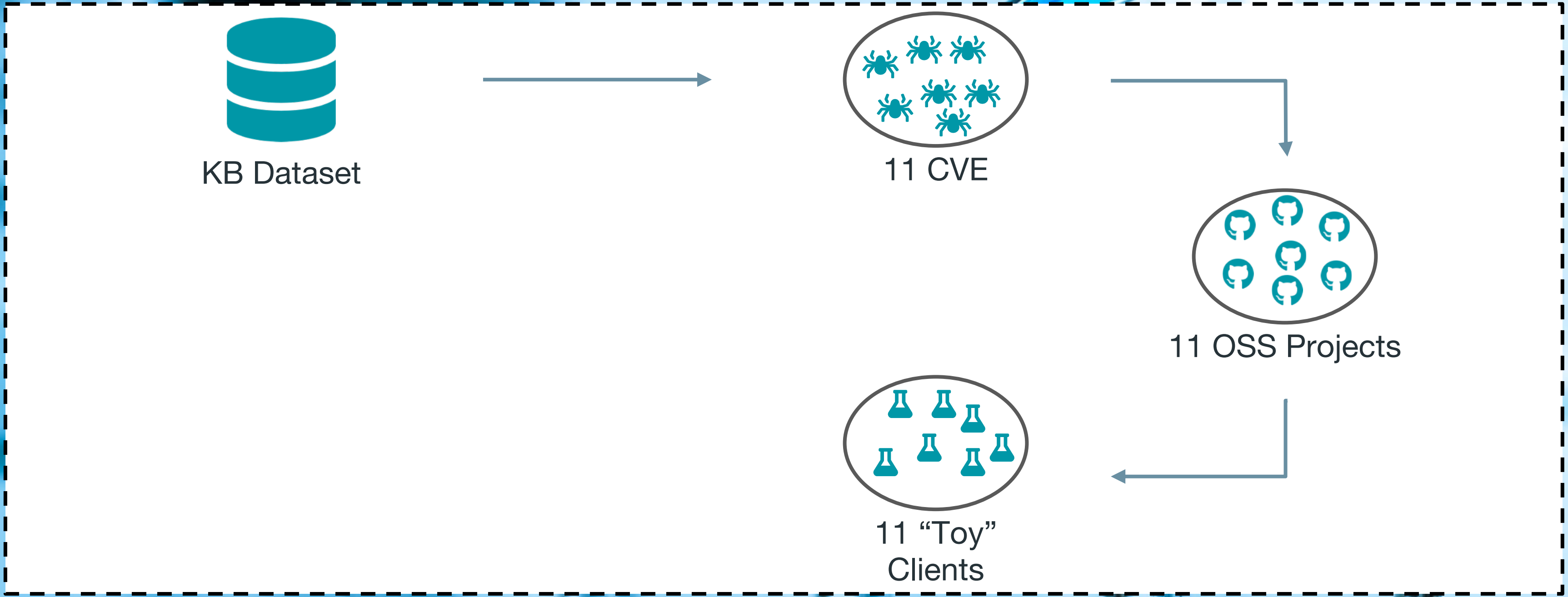
Exploratory Evaluation

Does SIEGE succeed in generating exploits of third-party vulnerabilities included within client applications?



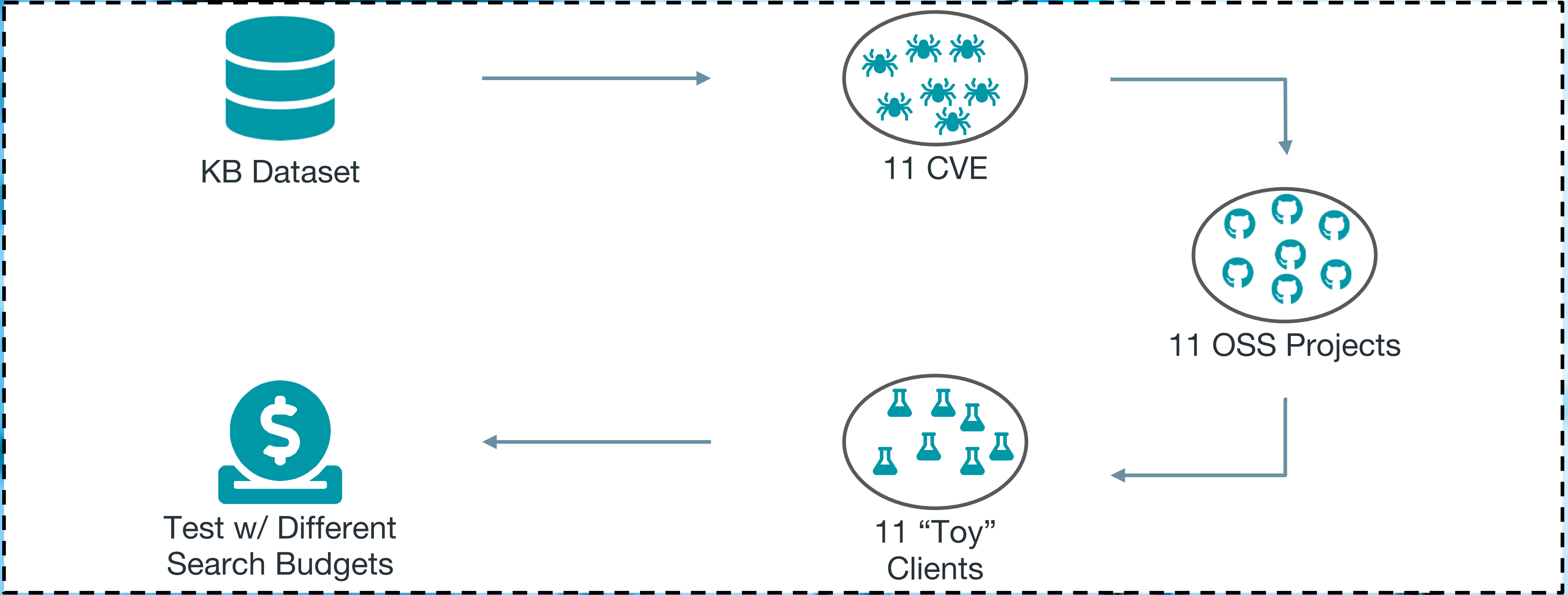
Exploratory Evaluation

Does SIEGE succeed in generating exploits of third-party vulnerabilities included within client applications?



Exploratory Evaluation

Does SIEGE succeed in generating exploits of third-party vulnerabilities included within client applications?



Exploratory Evaluation

Does SIEGE succeed in generating exploits of third-party vulnerabilities included within client applications?

- Commons Compress
- Tomcat
- Jasypt
- Jenkins
- Multijob
- Commons FileUpload
- HttpCommons Client
- Zeppelin
- Nifi
- Mailer
- Primefaces

Exploratory Evaluation

Does SIEGE succeed in generating exploits of third-party vulnerabilities included within client applications?

- Commons Compress
- Tomcat
- Jasypt
- Jenkins
- Multijob
- Commons FileUpload
- HttpCommons Client
- Zeppelin
- Nifi
- Mailer
- Primefaces

Exploratory Evaluation

Does SIEGE succeed in generating exploits of third-party vulnerabilities included within client applications?

- Commons Compress
- Tomcat
- Jasypt
- Jenkins
- Multijob
- Commons FileUpload
- HttpCommons Client
- Zeppelin
- Nifi
- Mailer
- Primefaces

Exploratory Evaluation

Does SIEGE succeed in generating exploits of third-party vulnerabilities included within client applications?

- Commons Compress
- Tomcat
- Jasypt
- Jenkins
- Multijob
- Commons FileUpload
- HttpCommons Client
- Zeppelin
- Nifi
- Mailer
- Primefaces

Exploratory Evaluation

Does SIEGE succeed in generating exploits of third-party vulnerabilities included within client applications?

Findings

The **intrinsic complexity** of a vulnerability makes the exploit generation harder

Exploratory Evaluation

Does SIEGE succeed in generating exploits of third-party vulnerabilities included within client applications?

Findings

The **intrinsic complexity** of a vulnerability makes the exploit generation harder

The **way** the client application "guards" the vulnerable constructs makes the exploit generation harder

Exploratory Evaluation

Does SIEGE succeed in generating exploits of third-party vulnerabilities included within client applications?

Findings

The **intrinsic complexity** of a vulnerability makes the exploit generation harder

The **way** the client application “guards” the vulnerable constructs makes the exploit generation harder

The **GA settings** influences the exploit generation performance

Future Directions

Future Directions

Risk Reporting

SIEGE could produce a report in which it **explains** why it succeeded/failed.


Future Directions
Risk Reporting
SIEGE could produce a report in which it **explains** why it succeeded/failed.
Vulnerability Generalized Description
Automatically build the fitness function using Steady's Patch Analyzer

Future Directions

Risk Reporting

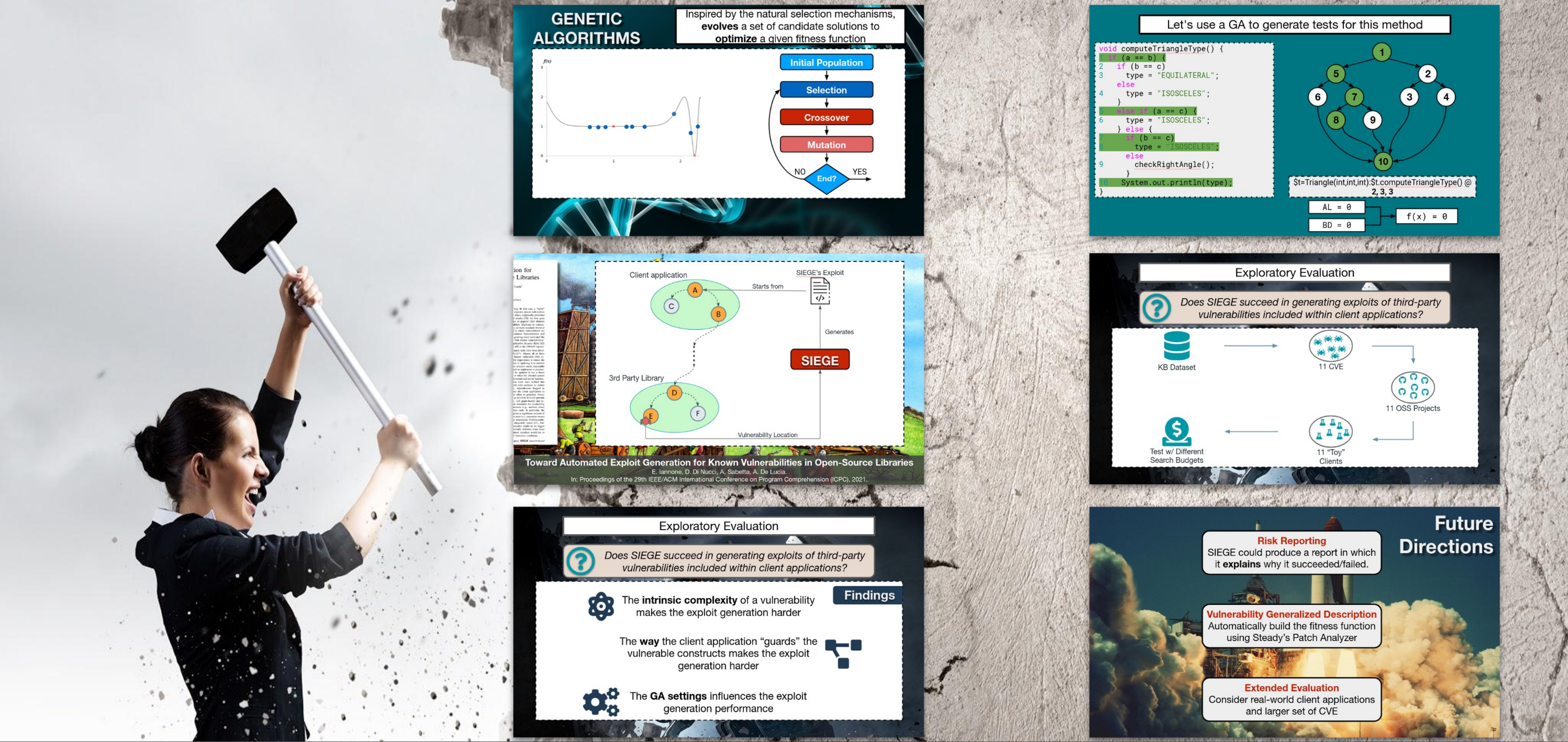
SIEGE could produce a report in which it **explains** why it succeeded/failed.

Vulnerability Generalized Description

Automatically build the fitness function using Steady’s Patch Analyzer

Extended Evaluation

Consider real-world client applications and larger set of CVEs



Automatic Test Case Generation: Toward Its Application in Exploit Generation for Known Vulnerabilities