Testing Injection Flaws through Contract-driven Coevolutionary Algorithms

Gabriele Costa, **Andrea Valenza**, Alessandro Armando

Highlights

Penetration Testing is crucial for web application development

Effective test cases are application-specific, and generating them is complex

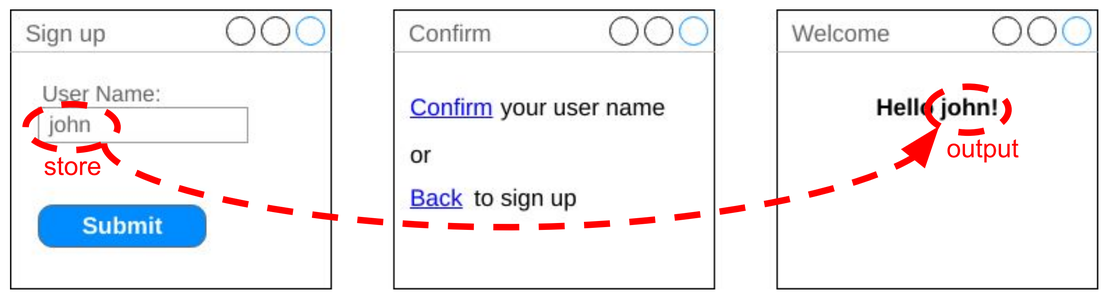
Test cases for **multi-step processes** are especially hard to find

Automatic testing tools cannot completely replace the human analysts

→ **We propose Beagle: a contract-driven, coevolutionary algorithm for automatic, application-specific test case generation**

Motivating example

Multi-step Stored XSS



→ **Traditional scanners (e.g., ZAP) cannot find it autonomously**

Contracts and vulnerabilities

A **vulnerability specification** is a pair **{L,C}** where L is a list of instructions and C is a **contract** (i.e., a predicate) on the variables in L.

A vulnerability is **triggered** when L is executed while C is satisfied.

Example: XSS Vulnerability Specification

{echo x, x ∈ Σ*<script>alert(R)</script>Σ*}

Where R is either a string or a number.

Vulnerability trigger:

```
echo $_GET['usr'];
```

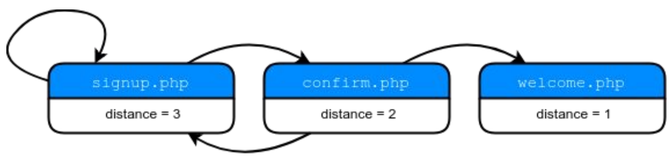
with $_GET['usr'] = "<script>alert(1)</script>"

(or "<script>alert('xss')</script>", ...)

→ A test T is **successful** when it triggers a vulnerability.

Fitness evaluation

Call (graph) distance



CallDistance = 1 ⇔ The page contains L.

Contract distance

The minimum distance between the image of a contract and the actual values generated by the test

Fitness

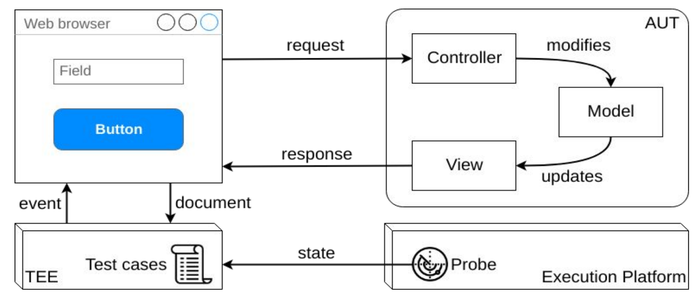
$$Fitness(T) = CallDistance - \frac{1}{1+ContractDistance}$$

Example
T_1 → inject "<script>alert(1a)</script>"
T_2 → inject "<script>alert(xss)</script>"

$$Fitness(T_1) = 1/2 \quad < \quad 2/3 = Fitness(T_2)$$

→ **Fitness(T) = 0 ⇔ T is a successful test.**
→ **Note**: finding *ContractDistance* is hard

Beagle



Genetic Approach

Two coevolving species:

- Test species
  a population of unsuccessful tests
- Contract species
  for *ContractDistance* approximation

Algorithm

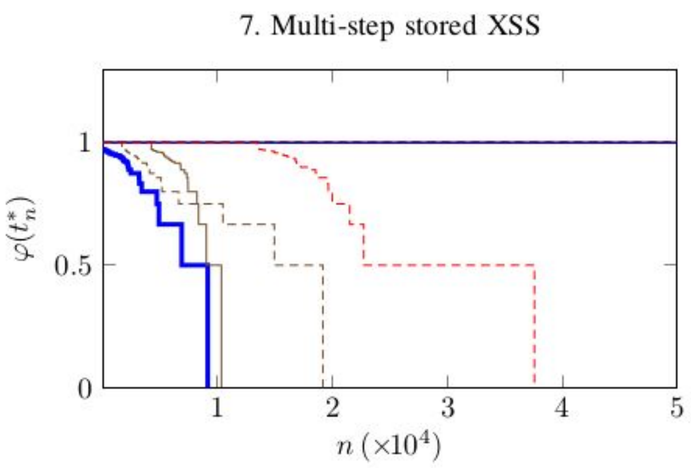
1. Start from random populations
2. Evaluate and select the **fittest tests**
   a. Coevolve Contract species
3. Mate and mutate tests
4. Repeat until a T is successful

→ **T is an executable sequence**

Benchmarks

WackoPicko

Tests applied to Injection Flaw vulnerabilities in WackoPicko (Doupé et al. 2010)



Vulnerability	WSR	Gen.	Time	Injected Payload
Reflected XSS	6/10	7425	787s	<script>alert(44)</script>
Stored XSS	4/10	9380	993s	<script>alert(``s')</script>
Stored SQLi	0/10	50000	5132s	AcuYq6*4M-PaE
Reflected SQLi	6/10	6049	640s	admin'-- IgeBGMBL0`MnGUU99#p
Multi-step XSS	**4/10**	**9137**	**968s**	**n<script>alert(3)</script>FO**
Command-line Injection	1/10	7969	844s	q7d &; ls #X<'SI
XSS behind JS	3/10	9970	1056s	<script>alert(3)</script>^JM
XSS behind Flash	4/10	11683	1238s	<script>alert(13)</sCRipt>

This poster is based on the following paper:

1. Costa G., Valenza A., Armando A.:
   „Why Charles Can Pen-test: an Evolutionary Approach to Vulnerability Testing"
   Submitted at Network and Distributed System Security Symposium 2019 **(NDSS 2019)**

CISPA HELMHOLTZ-ZENTRUM i. G.