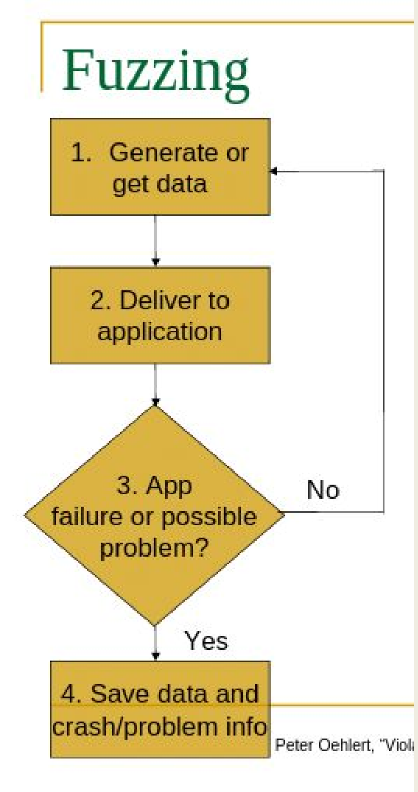
# XSS Vulnerability Detection Using Model Inference Assisted Evolutionary Fuzzing

Fabien Duchene, Roland Groz, Sanjay Rawat, Jean-Luc Richier

- Laboratoire d’Informatique de Grenoble, France
- 2012

*Presented as a part of Research Practice by,*
*Bofin Babu | 2013H313085H*

# Fuzzing


Fuzzing
1. Generate or get data
2. Deliver to application
3. App failure or possible problem?
No
Yes
4. Save data and crash/problem info
Peter Oehlert, "Viol

# Genetic Algorithm (GA)

Uses techniques inspired by natural evolution such as,

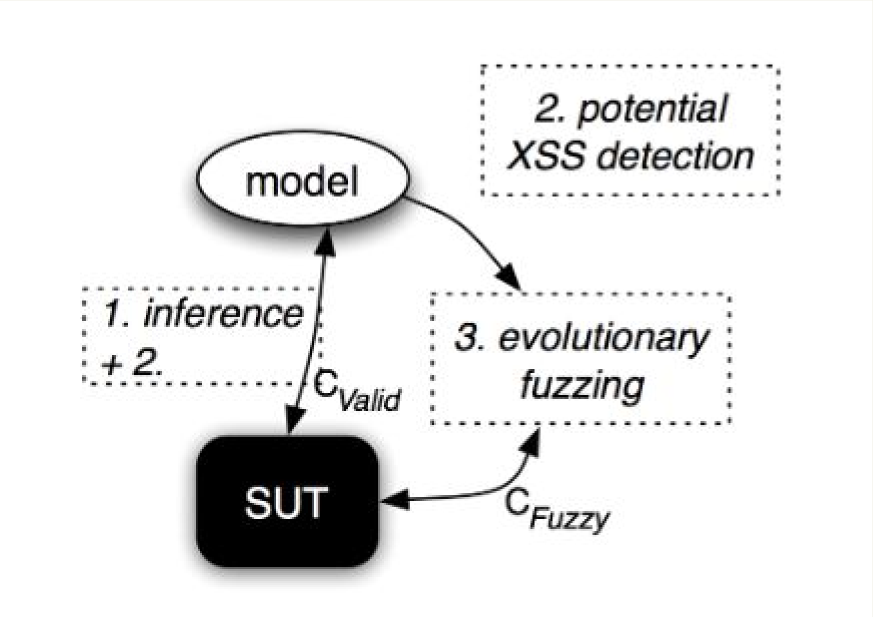
- **Inheritance**
  - The ability of modeled objects to mate, mutate and propagate their problem solving genes to the next generation, in order to produce an evolved solution to a particular problem.
  - The selection of objects that will be inherited from in each successive generation is determined by a fitness function.
- **Mutation**
  - Mutation alters one or more gene values in a chromosome from its initial state
  - Used to maintain genetic diversity from one generation of a population of genetic algorithm chromosomes to the next.
- **Selection**
  - The stage of a genetic algorithm in which individual genomes are chosen from a population for later breeding (using crossover operator).
- **Crossover**
  - A genetic operator used to vary the programming of a chromosome or chromosomes from one generation to the next.
  - Analogous to reproduction - taking more than one parent solutions and producing a child solution from them.

# Reflected XSS

- Also known as non-persistent XSS attacks and, since the attack payload is delivered and executed via a single request and response
- A a web application is vulnerable to this type of attack if it passes unvalidated input.

**http://example.com/index.php?user=<script>alert(123)</script>**

# Approach Overview



# Deeper technical insight

## Web Application

Transition(u) : A mapping from **n** user inputs $i_l^u \in \Sigma^*$: $I_u = i_1^u, ..., i_n^u$ to output $q = q_1 \cdot q_2 \cdot ... \cdot q_k,\ q \in \Sigma^*$.

Each **qj** is either a web-server filtered input parameter $i_l^u$ - i.e. $\exists f_r \in Filters, \underset{\sim}{q_j} = f_r(i_l^u)$ or, a string $q_h$ surrounding one or more $\underset{\sim}{q_j}$.

An **individual** is a sequence $I = (I_1, ..., I_m)$ where each **I**u satisfies the above definition.

# Deeper technical insight

## Potential XSS Detection.

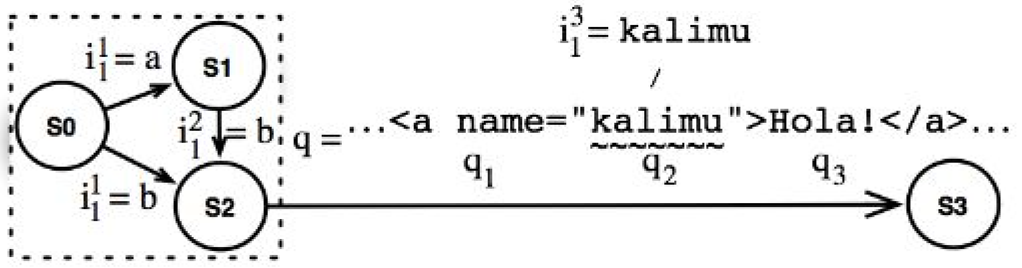



Fig. 2. When the value of an input parameter $i_l$ is observed in the output $q$, the fuzzing starts from that initiating state on that very same $i_l$

The value if **I**3 (kalimu) is observed in **q**, the output of the transition **S**2 - > **S**3.

We assume a possibility of XSS there and start fuzzing on **I**3 from **S**2.

# Deeper technical insight

## XSS Fuzzing attack grammer

- An an input grammar is needed to impose some restrictions on EA to generate inputs by constraining mutations and crossovers.
- This helps to be closer to the behaviour of an attacker who would mainly modify interesting input parameters.

```
<a name="[USER_INFLUENCED_INPUT]">Hola!</a>
```

Here q = q1 + *q2* +q3 meaning that *q2* is a result of a filtering function applied to an input parameter $i_1^3$

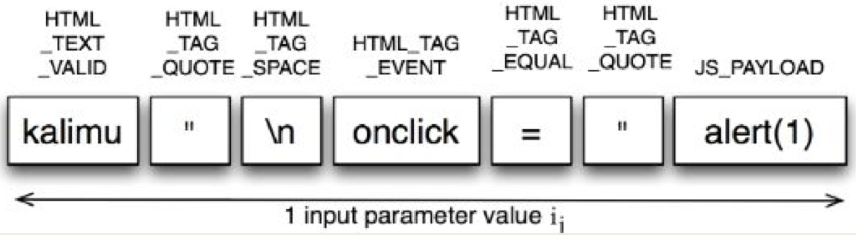
# Deeper technical insight

## XSS Fuzzing attack grammer

HTML_XSS_FIELD ::= HTML_TEXT_SIMPLE HTML_TAG_QUOTE
    HTML_TAG_SPACE HTML_TAG_EVENT HTML_TAG_EQUAL
    HTML_TAG_QUOTE JS_PAYLOAD
HTML_TAG_QUOTE ::= ' | "
HTML_TAG_SPACE ::= \n | \t | \r | ␣
HTML_TAG_EQUAL ::= =
HTML_TAG_EVENT ::= onabort | ... | onclick | ... | onwaiting

Grammar fragment 1. Injecting into an HTML attribute field value

Figure 3 shows an input parameter value $i_l$ (thus a subset of an input sequence) generated using that grammar:




← an extract of the written attack grammar for guiding input mutations

# Deeper technical insight

## Creating first generation

Individuals of the first generation are created from the attack grammar and known attack inputs

# Deeper technical insight

## Character classes

- Exploiting an injection is about sending data and instructions to the SUT that does not use them in a safe way and assumes those inputs as only data.
- First submit only data for inferring a SUT formal model. Then during the fuzzing step, a combination of data and instructions is sent to the SUT.

- $C_0$: HTML Spaces: ␣ || \r || \t || \n
- $C_1$: HTML Attribute delimiter: " || ` || '
- $C_2$: HTML Tag delimiter: $< \| > \| />$
- $C_3$: HTML Equal sign: $=$
- $C_4$: JavaScript code: ( || ) || ; || { || }
- $C_5$: URL related: / || : || ? || &
- $C_6$: Escaping character: \
- $C_7$: HTML_TEXT_SIMPLE: [a-Z] $\cup$ [0-9]

Here our grammar **G**0 is HTML

$$C_{valid} = C_0 \cup C_7.$$

During the fuzzing step, input parameter values from $C_{fuzzy} = \cup_{i=0}^{7} C_i$ are submitted to the SUT.

# Deeper technical insight

## Detecting XSS attacks

If the attacker succeeds in crafting an input **i**/ such that

$\tilde{q}_2 = \texttt{kalimu" onclick="alert(1)}$ then **q**2 is not syntactically confined w.r.t **G**0 (HTML)

# Deeper technical insight

## Evolutionary fitness function for XSS

How well a given individual **I** is close to detect an XSS.

$$Fit(I) = \frac{S(I)}{S_{total}} + \frac{C_{injected}(I)}{C_{sent}(I)} + W_{ell}(I) + N(I)$$

1. States reachable within few transitions from the initial state are more likely to be sanitized than deeper ones In *Fig. 2.*, if $I = ((i_1^1 = a), (i_1^2 = b), (i_1^3 = \texttt{kalimu}))$, then $S_{reached}(I) = 3$
2. Fitness should be an increasing function of it.
3. A well formed output will be more likely be executed by the client. Well(I) = 1 if q is well formed 0 otherwise.

# Deeper technical insight

4. metric $N(I)$ that represents the improvement of $I$ in terms of HTML nodes that are reflected from $i^{m}_{l_{fuzzed}}$ w.r.t. its predecessors $Pred(I)$. If $q(I_m) = q_1 \cdot q_2 \cdot \ldots \cdot q_k$, then $A(I) = \max_{j \in 1..k} Nodes_{G_O}(q_j(i^{m}_{l_{fuzzed}}))$
For instance, in *Fig. 4.*, $A(I) = 3$.

$$N(I) = \frac{A(I) - \frac{\sum_{P \in Pred(I)} A(P)}{\|Pred(I)\|}}{\max_{E \in Gen} A(E)}$$

How much of the reflections will be interpreted as instructions.

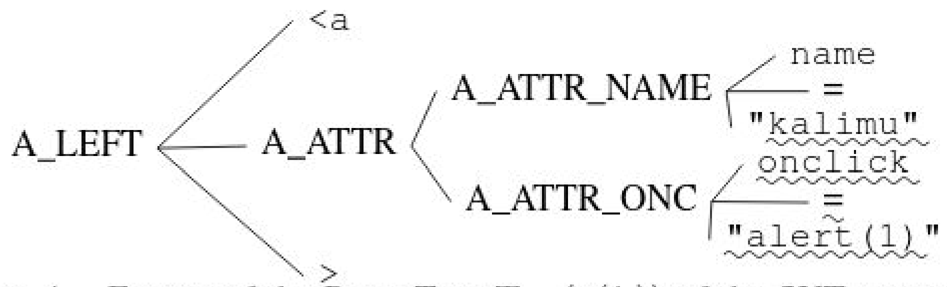



Fig. 4. Extract of the Parse Tree $T_{G_0}(q(i_l))$ of the SUT output $q$

# Deeper technical insight

## Evolving the population

- Following we define mutation and crossover operations that tend to respect the Attack Input Grammar $G_{AI}$
- Let $I = (I_1, ...I_m)$ and $J = (J_1, ..., J_w)$ be two individuals with potential reflection.
- Crossover performed and a child would be: $(I_1, ..., I_{m-1}, (i_1^m, ..., i_{x-1}^m, j_y^w))$ Where i,j are input parameters.