Planning-based Security Testing of Web Applications

Josip Bozic and Franz Wotawa
Graz University of Technology
Institute for Software Technology
jbozic@ist.tugraz.at
AST@ICSE, May 28th, 2018

Outline

1. Introduction
2. Planning
3. WebTesting
4. Conclusion

Introduction

- Why security testing?
  - Cyber security is a global issue.
  - Web application vulnerabilities still represent a major challenge. [1]
  - Security leaks indicate vulnerability against attacks.

- Requirements:
  - Data confidentiality
  - Secure authentication
  - Secure communication

Introduction

- Notable targets:
  - The United Arab Emirates Invest Bank
  - White House
  - eBay
- Negative consequences:
  - Vulnerable programs cause costs.
  - Negatively impacts trust in applications, companies and people.

Contribution

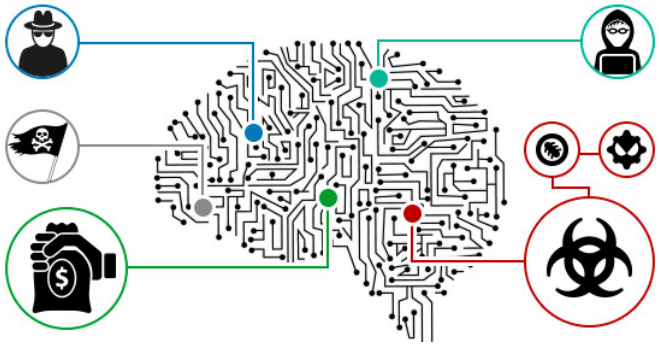
- Planning-based approach for modeling and security testing of web applications.
- Automated execution and detection of SQL injection (SQLI) and reflected and stored cross-site scripting (XSS).
- Goal: Cover standard exploitation attempts and uncover new ones.

Automated Planning and Scheduling

- Applications:
  - Artificial intelligence (AI) in testing. [2,3]
  - Initially used for intelligent agents, robotics etc. [4]

- Characteristics:
  - Plan: Sequence of actions with pre- and postconditions.
  - Conditions guide the planning process.
  - Planner: Program that provides a solution to the problem according to an algorithm. [5,6,7,8]

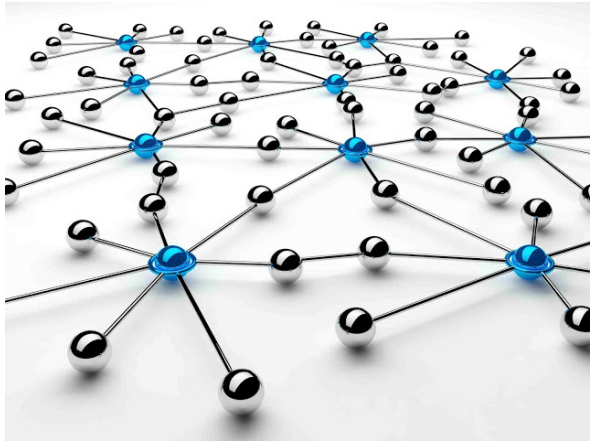
Planning in Security Testing



- Why planning ?
  - Attack: Sequence of actions that lead to exploitation.
  - Plan: Blueprint for an attack.
  - By automating the test case generation and execution, the attacker is emulated in an iterative manner.

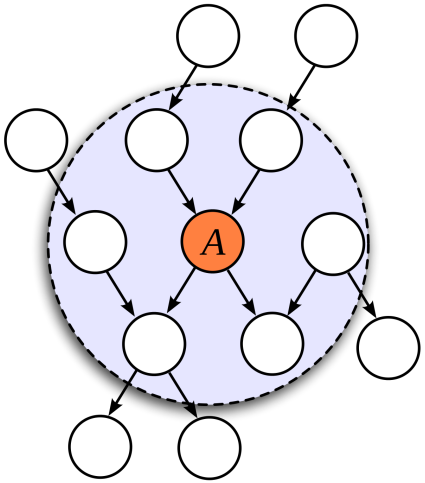
Planning in Security Testing

- Every interaction between a client and a system can be represented as sequence of actions.
- A test case is a sequence of interactions with the SUT.
- An **initial state**: Starting point of the attack.
- **Final state**: Condition where an attack was successful.
- **Result**: FAIL or PASS.



Planning Model

- No graphical model of the SUT.
- Planning specification in the Planning Domain Definition Language (PDDL).
- Data definitions in order for the planner to generate a plan.
- Multiple solutions for one planning problem.



Planning Model

```
(:objects x)

(:predicates
   (inInitial ?x)
   (inGotSite ?x)
   (inAttackedSQLI ?x)
   (inAttackedXSS ?x)
   (inFinished ?x))

(:action GetSite
   :parameters (?x)
   :precondition ()
   :effect (inGotSite ?x))

(:action AttackXSSGet
   :parameters (?x)
   :precondition (and (inGotSite ?x)
      (inAttackedSQL ?x)))
   :effect (and (inAttackedXSS ?x)
      (inFinished ?x)))
```

```
(:action AttackXSSPost
   :parameters (?x)
   :precondition (and (inGotSite ?x)
      (inAttackedSQL ?x))
   :effect (and (inAttackedXSS ?x)
      (inFinished ?x)))

(: action AttackSQLGet
    :parameters (?x)
    :precondition (inGotSite ?x)
    :effect (and (inAttackedSQL ?x)
       (inFinished ?x)))

(: action AttackSQLPost
    :parameters (?x)
    :precondition (and (inGotSite ?x)
       (inAttackedXSS ?x))
    :effect (and (inAttackedSQL ?x)
       (inFinished ?x)))
```

Planning Model

Problem description:

```
(:init (inInitial x))

(:goal (inFinished x))
```

Generated plan:

```
0: GetSite(x)
1: AttackSQLGet(x)
2: GetSite(x)
3: AttackSQLPost(x)
4: AttackXSSPost(x)
5: GetSite(x)
6: AttackXSSGet(x)
```

Planning Model

- Advantages:
  - Extendibility.
  - Configurability (e.g. no conditions).
  - Every change in the model results in different plans.
  - Model explosion is avoided.
  - Follow execution traces that are not given in specification of SUT.

- Disadvantages:
  - No possibility to interact with SUT.
  - (Almost) No concrete values.

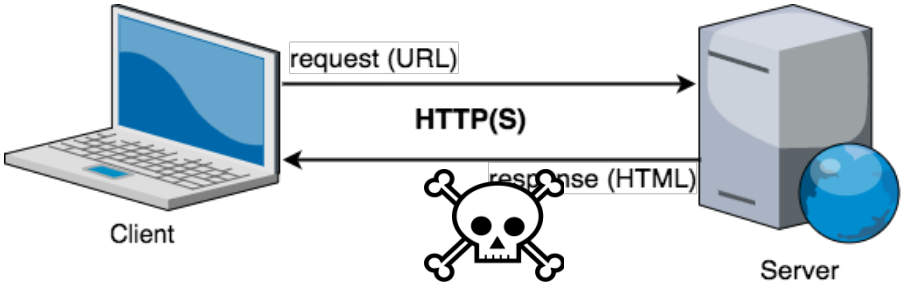
Planning System

- Automated planners:
  - Planners usually return only one plan for a planning problem. (E.g. [5])
  - Java implementation of the Graphplan algorithm, JavaGP. [8]
  - Configurable number of plans.
- Motivation:
  - Generate plans with a broad diversity.
  - Cause unintended behavior, eventually confusing the SUT.

Planning System

1	GetSite(x), AttackSQLGet(x), AttackXSSGet(x), AttackSQLPost(x)
2	GetSite(x), AttackSQLGet(x), GetSite(x), AttackXSSGet(x), AttackSQLPost(x)
3	GetSite(x), GetSite(x), AttackSQLGet(x), AttackXSSGet(x), AttackSQLPost(x), GetSite(x), AttackXSSPost(x)
4	GetSite(x), AttackSQLGet(x), GetSite(x), AttackXSSGet(x), GetSite(x), AttackSQLPost(x)
5	GetSite(x), AttackSQLGet(x), GetSite(x), AttackXSSGet(x),GetSite(x), GetSite(x), AttackSQLPost(x), AttackXSSGet(x)
6	GetSite(x), GetSite(x), AttackSQLGet(x), AttackXSSGet(x), GetSite(x), GetSite(x), AttackSQLPost(x), AttackXSSPost(x)
7	GetSite(x), AttackSQLGet(x), AttackXSSPost(x), GetSite(x), AttackSQLPost(x)
8	GetSite(x), AttackSQLGet(x), GetSite(x), AttackXSSPost(x), AttackSQLPost(x), GetSite(x), AttackXSSPost(x)

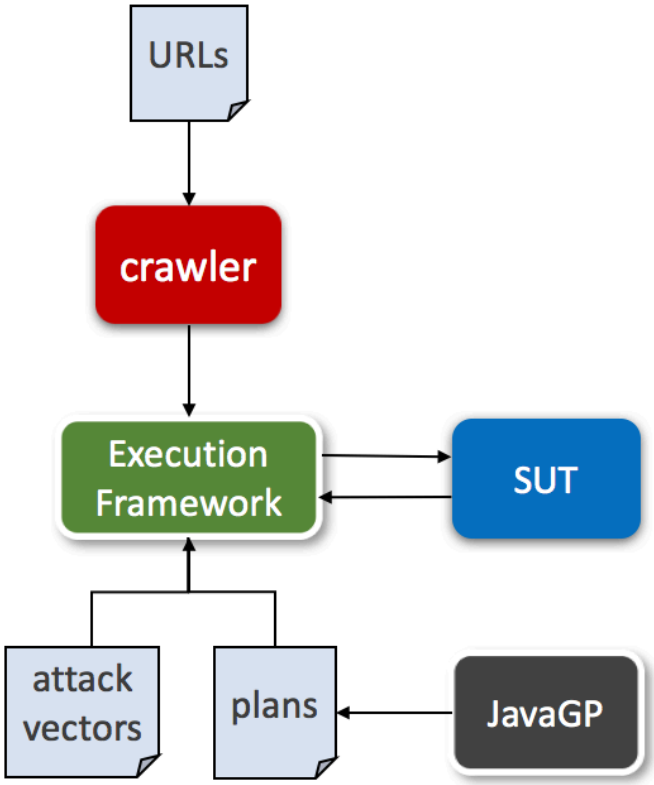
Planning for Web Applications



- PDDL specification of the client's side.
- Checking the server's response.
- HTTP methods: GET, POST,…

GET /site/login.php?username=john&password=!js123 HTTP/1.1 POST /site/login.php HTTP/1.1 Host: www.company.com username=john&password=!js123

Test Execution Framework



Test Execution Framework

- Java-based execution framework: Resembles the communication between client and server.
- HttpClient [9] for HTTP
- Parser: jsoup [10]
- Plan: Abstract test case; Guidance of the execution.
- Test sets: XSS and SQLI; for concretization.
- Concretization phase: During execution, concrete values are assigned to the parameters of the individual actions.
- Implemented oracles: *PASS* | *FAIL*.
- Crawler: Ensures that a SUT is tested completely.

Test Execution Framework

- Examples: <script>alert(0)</script>

```
POST /site/login_form.php HTTP/1.1
Host: w3company.com
username=<script>alert(0)</script>&password=
```

```
POST /site/login_form.php HTTP/1.1
Host: w3company.com
username=&password=<script>alert(0)</script>
```

Test Execution Framework

- Concretization:

1. GetSite(x)
2. AttackXSSGet(x)
3. AttackXSSPost(x)
4. AttackSQLGet(x)
5. AttackSQLPost(x)

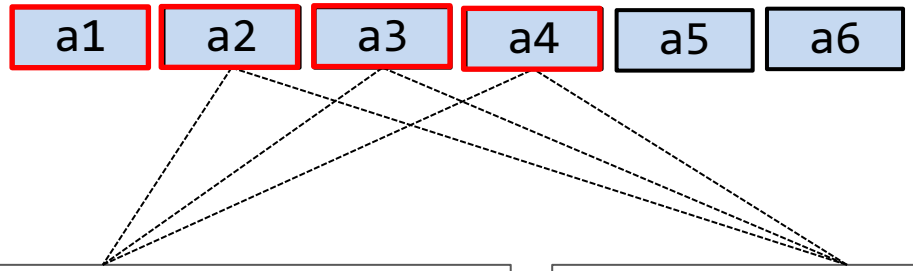
```
public static int[] get()
{
public static boolean attackXss(...)
{
    HttpResponse response =
    httpclient.execute(httpGet);
    HttpEntity entity = response.getEntity();
    BasicNameValuePair(Submit.get(0),
    Submit.get(1)));
    httpPost.setEntity(new
    UrlEncodedFormEntity(list));
    response = httpclient.execute(httpPost);

}
```

…

Test Execution Framework

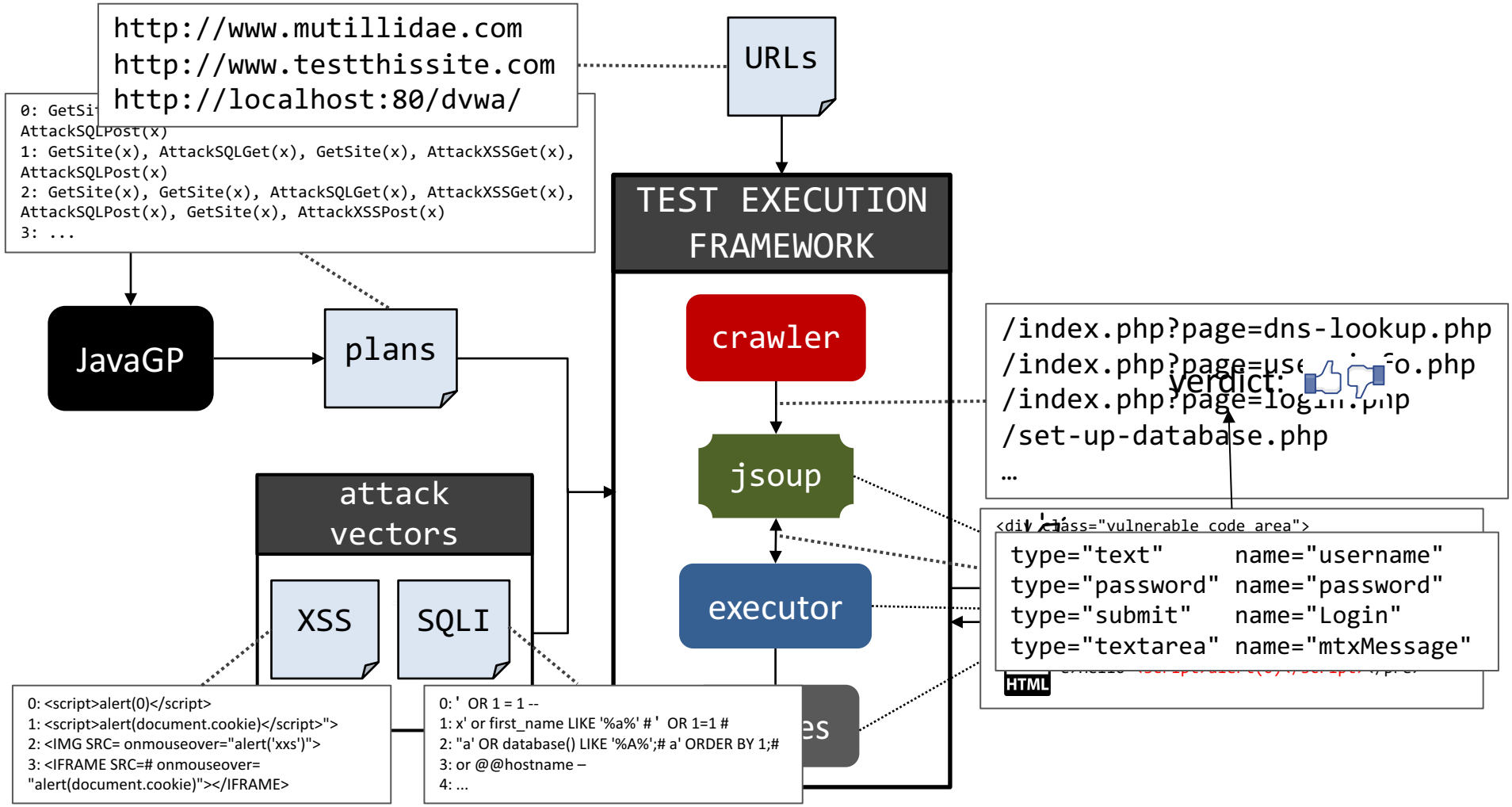
- Concretization:




```
0: <script>alert(0)</script>
1: <script>alert(document.cookie)</script>">
2: <IMG SRC= onmouseover="alert('xxs')">
3: <IFRAME SRC=# onmouseover=
"alert(document.cookie)"></IFRAME>
```

```
0: ' OR 1 = 1 --
1: x' or first_name LIKE '%a%'
# ' OR 1=1 #
2: "a' OR database() LIKE
'%A%';# a' ORDER BY 1;#
3: or @@hostname –
4: ...
```

Test Execution Framework



Conclusion/Future Work

- Planning-based security testing approach.
- Planning model for XSS and SQLI.
- Crawler support.
- Advantages:
  - Planning models: Keep the representation small but achieve many test cases with variety.
  - Configurability
- Disadvantages:
  - PDDL knowledge.

Conclusion/Future Work

- Future:
  - Extend the planning model (i.e. incorporate more attacks).
  - Combine attacks.
  - Refine test oracles.
  - Test real-world applications.
  - Comparison with other approaches.
  - Optimum: Trigger new vulnerabilities.

References

[1] “OWASP Top Ten Project,”
https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project.

[2] K. Durkota and V. Lisy: *Computing Optimal Policies for Attack Graphs with Action Failures and Costs*. In: 7th European Starting AI Researcher Symposium (STAIRS’14), 2014.

[3] M. Backes, J. Hoffmann, R. Kunnemann, P. Speicher and M. Steinmetz: *Simulated Penetration Testing and Mitigation Analysis*. In: CoRR abs/1705.05088 (2017), 2017.

[4] S. J. Russell and P. Norvig: *Artificial Intelligence: A Modern Approach*.
In: Prentice Hall, 1995.

[5] "Metric-FF," http://fai.cs.uni-saarland.de/hoffmann/metric-ff.html.

[6] "Fast Downward," http://www.fast-downward.org/.

[7] ”LPG," http://lpg.unibs.it/lpg/.

[8] “JavaGP,” http://emplan.sourceforge.net/.

[9] “Apache HttpComponents - HttpClient,” https://hc.apache.org/httpcomponents-client-ga/.

[10] “jsoup: Java HTML Parser,” https://jsoup.org/.

[11] “OWASP Mutillidae 2 Project,”
https://www.owasp.org/index.php/OWASP_Mutillidae_2_Project

THANK YOU !