The Mathematics behind an Automated Penetration Testing Framework

Dimitris E. Simos
SBA Research

Security Forum 2014
University of Applied Sciences Upper Austria
Hagenberg Campus, Austria
April 9, 2014
• **Stefanie Falkner** and **Hagenberger Kreis** for the invitation and hospitality

• **Researchers and penetration testers** contributed in various ways to the material of this talk
  - SBA Research: Bernhard Garn
  - Security Research: Peter Aufner, Andreas Bernauer, Severin Winkler
  - IST/TU Graz: Franz Wotawa, Josip Bozic
Outline of the Talk

Penetration Testing
  Concept
  Combinatorial Testing
Outline of the Talk

Penetration Testing
   Concept
   Combinatorial Testing

Testing a Web Application
   The Application
   The Connection to the Designs
Outline of the Talk

Penetration Testing
  Concept
  Combinatorial Testing

Testing a Web Application
  The Application
  The Connection to the Designs

Automation of XSS Attack Vectors
  A Combinatorial Grammar
  Evaluation and Results
  Test Execution and Framework
Outline of the Talk

Penetration Testing
  Concept
  Combinatorial Testing

Testing a Web Application
  The Application
  The Connection to the Designs

Automation of XSS Attack Vectors
  A Combinatorial Grammar
  Evaluation and Results
  Test Execution and Framework

Research Problems
Penetration Testing

Software Testing

- Testing is an important but expensive part of the software development process
- The problem is to design a test plan for a software system

Penetration Testing: Problem Specification

- The process of simulating attacks on a network and its systems, at the request of the owner or senior management
- Reference: CISSP certification guide
You cannot test Everything
But you can select a Few Tests..
..and still achieve Good Coverage
How to Design a Test Plan?

The Design Process

- To design a test plan the tester identifies possible output values from each of the stages of the software system or *system under test* (SUT).
- It is important to find a test plan that is not too large, yet tests for most of the interactions among the possible outputs in the modules of a software system.
- Software developers have begun using combinatorial designs to test for these interactions.
Coverage vs. Interactions

  http://dx.doi.org/10.1109/MITP.2008.54
How many Interactions to Test?

Motivations

1. Detection rate increases rapidly with interaction strength
2. Within the NASA database application:
   - 67 percent of the failures were triggered by only a single parameter value
   - 93 percent by two-way combinations
   - and 98 percent by three-way combinations

Challenges

- Four- to six-way combinations reach 100 percent detection
- Generating test suites to cover all \( t \)-way interactions is a difficult mathematical problem studied for nearly a century
- Modelling and automating penetration testing procedures
Components of a Testing Framework

This Talk

- **Automation** of test cases for penetration testing
- **Structure** of an (automated) penetration testing framework
Combinatorial Test Design Process

1. Model the input space; The model is expressed in terms of stages and stage values
2. The model is input to a combinatorial design procedure to generate a combinatorial object which is simply an array of symbols
3. Every column of the generated array is used to output a test plan of the software system

Benefit

Steps 2 and 3 can be automated
The Application

Toy Example for Testing a Web Application

1. User Input can be capital or small letters and quotes
2. Validation for SQL injection with values yes and no
3. User database interaction with values true and false
4. Validation for cross-site scripting (XSS) with values yes and no

Goal

- Vulnerabilities are caused by two-way interactions
- Generate test plans to identify these vulnerabilities
Different Scenarios

Analysis
We have three binary stages and one with three stage values therefore a total of $3 \times 2^3 = 24$ different scenarios

Goal
1. Reduce the number of test scenarios
2. Still test all of the two-way interactions
   - The interaction of two-variables, i.e. (user input, validation for SQL-I)
The Connection to the Designs

Mixed-level Covering Arrays

1. A mixed-level covering array $\text{MCA}(t, k, (g_1, \ldots, g_k))$ is an $k \times N$ array in which the entries of the $i$-th row arise from an alphabet $S_i$ of size $g_i$. 

2. Let $\{i_1, \ldots, i_t\} \subseteq \{1, \ldots, k\}$ and consider the subarray of size $t \times N$ by selecting rows of the MCA.

3. There are $\prod_{i=1}^{t} g_i$ possible $t$-tuples that could appear as columns, and an MCA requires that each appears at least once.
Definition too Mathematical?

"I think you should be more explicit here in step two."
In Practical Terms

\( \text{MCA}(t, k, (g_1, \ldots, g_k)) \) Covering Arrays for Mixed-Test Plans

- The strength \( t \) of the MCA refers to the number of interactions we are checking.
- The rows \( k \) of the MCA give the different stages of the model.
- The number \( |S_i| = g_i, i = 1, \ldots, k \) gives the number of different stage values.
- Coverage is ensured from the property that in every combination of \( t \) rows, each pair of symbols from \( S_i \) appears at least once.
How to Design a Mixed Test Plan

Formulation of the Problem

- The testing for the web-application has 4 stages therefore the MCA must have 4 rows of which:
  1. One row must take values \{0, 1, 2\}
  2. Remaining rows are encoded from \{0, 1\}
- If we take into account only two-way interactions we seek a \(MCA(2, 4, (2, 2, 2, 3))\)
The Mixed Test Plan

A Mixed-level Covering Array $MCA(2, 4, (2, 2, 2, 3))$

- Such an array with 6 columns exists:

```
  0 1 2 0 1 2
  0 1 0 1 0 1
  0 0 1 1 1 0
  0 1 1 1 0 0
```

Understanding the Two-Way Interactions of the MCA

- Selecting any two columns with binary encoding each the following 2-tuples must appear at least once: 00, 01, 11
- Selecting any two columns with binary and ternary encoding, respectively, the following 2-tuples must appear at least once: 00, 01, 02, 11, 12
### The Test Configuration

#### Test Scenarios

<table>
<thead>
<tr>
<th></th>
<th>caps</th>
<th>small</th>
<th>quotes</th>
<th>caps</th>
<th>small</th>
<th>quotes</th>
</tr>
</thead>
<tbody>
<tr>
<td>User Input</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>SQL Injection</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>User DB Inter.</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>XSS</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

#### Test Configurations

<table>
<thead>
<tr>
<th></th>
<th>caps</th>
<th>small</th>
<th>quotes</th>
<th>caps</th>
<th>small</th>
<th>quotes</th>
</tr>
</thead>
<tbody>
<tr>
<td>User Input</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>SQL Injection</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>User DB Inter.</td>
<td>no</td>
<td>true</td>
<td>true</td>
<td>no</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>XSS</td>
<td>yes</td>
<td>false</td>
<td>false</td>
<td>yes</td>
<td>no</td>
<td>no</td>
</tr>
</tbody>
</table>
## Still not Convinced?

### Some Applications of Combinatorial Design Testing so Far

- Air traffic avoidance system module
- Fault detections in F-16 Locheed Martin software
- Calibration of Industrial plant switches
- Android smart phone configuration testing
- Attack-pattern combinatorial testing for web-applications

### Popular Test Suites

- Advanced Combinatorial Testing System (ACTS) - NIST project
- The AETG System - An approach to testing based on combinatorial design
Generation of XSS Attack Vectors

Cross-Site-Scripting (XSS)

- Inject client-side script into web-pages viewed by other users
- Bypass access controls

Sample of XSS Attack Vectors

```html
<src>"onclick""alert("hacked")"</src>
<script>'"onclick"alert(document.cookie)'</script>
```

Our Goal

Employ combinatorial testing to (automatically) generate XSS attack vectors
A Combinatorial Structure for XSS Attack Vectors

Input Parameter Modelling

The designer chooses the possible parameter values for the SUT

Structure of an XSS Attack Vector

- 12 discrete parameters
- 6 of them are single-valued or have to satisfy certain constraints

Form of an XSS Attack Vector

\[
\text{AV} :=< \text{FOBRACKET}, \text{TAG}, \text{FCBRACKET}, \text{QUOTE1}, \text{SPACE}, \text{EVENT}, \text{SPACE2}, \\
\text{QUOTE2}, \text{PAYLOAD}, \text{LOBRACKET}, \text{CLOSINGTAG}, \text{LCBRACKET} >
\]
A BNF Grammar for XSS Attack Vectors

The Grammar

FOBRACKET ::= <
TAG ::= img | frame | src | script | body | HEAD | BODY | iframe | IFRAME |
FCBRACKET ::= >
QUOTE1 ::= ' | null
SPACE ::= \n | \t | \r | a | b | c | null
EVENT ::= onclick | onmouseover | onerror | onfire | onbeforeload |
SPACE2 ::= \n | \t | \r | a | b | c | null
QUOTE2 ::= ' | null
PAYLOAD ::= alert(1) | alert(0) | alert(document.cookie) | alert("hacked") |
LOBRACKET ::= </
CLOSINGTAG ::= img | frame | src | script | body | HEAD | BODY | iframe |
LCBRACKET ::= >

Table: Modelling the XSS Attack Vectors

<table>
<thead>
<tr>
<th>covering arrays</th>
<th>attack vectors</th>
</tr>
</thead>
<tbody>
<tr>
<td>rows symbols</td>
<td>types</td>
</tr>
<tr>
<td>strength</td>
<td>derivation rules per type interactions</td>
</tr>
</tbody>
</table>
Case Study

Test Applications

- **BodgeIt**: 1 vulnerable field parameter for XSS
- **DVWA & Mutillidae**: 2 vulnerable field parameters for XSS

Combinatorial coverage vs. field coverage

- **Combinatorial coverage**: How many of the generated XSS vectors exploit a vulnerability
- **Field coverage**: How many of the field parameters are vulnerable to XSS with a specified input

Combinatorial test oracle

1. First testing **all** two pairwise interactions ($t = 2$)
2. Then **increase** the interaction strength $t$ **until** no further errors (injections in our case) are detected by the $t$-way tests
3. Finally test $t + 1$ and ensure that **no** additional errors are detected
<table>
<thead>
<tr>
<th>Strength</th>
<th>Application</th>
<th>Vulnerability</th>
<th>Difficulty Level</th>
<th>Execution time (s)</th>
<th># of successful injections</th>
<th>% coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Bodgelt RXSS</td>
<td>-</td>
<td></td>
<td>30.00</td>
<td>69</td>
<td>60.53</td>
</tr>
<tr>
<td></td>
<td>SXSS</td>
<td>-</td>
<td></td>
<td>31.80</td>
<td>56</td>
<td>49.12</td>
</tr>
<tr>
<td>3</td>
<td>RXSS</td>
<td>-</td>
<td></td>
<td>244.60</td>
<td>619</td>
<td>60.04</td>
</tr>
<tr>
<td></td>
<td>SXSS</td>
<td>-</td>
<td></td>
<td>242.80</td>
<td>512</td>
<td>49.66</td>
</tr>
<tr>
<td>4</td>
<td>RXSS</td>
<td>-</td>
<td></td>
<td>1788.70</td>
<td>4991</td>
<td>59.90</td>
</tr>
<tr>
<td></td>
<td>SXSS</td>
<td>-</td>
<td></td>
<td>1950.80</td>
<td>4157</td>
<td>49.89</td>
</tr>
<tr>
<td>2</td>
<td>DVWA RXSS</td>
<td>1</td>
<td></td>
<td>27.40</td>
<td>69</td>
<td>60.53</td>
</tr>
<tr>
<td></td>
<td>SXSS</td>
<td>1</td>
<td></td>
<td>28.50</td>
<td>69</td>
<td>60.53</td>
</tr>
<tr>
<td></td>
<td>RXSS</td>
<td>2</td>
<td></td>
<td>30.70</td>
<td>56</td>
<td>49.12</td>
</tr>
<tr>
<td></td>
<td>SXSS</td>
<td>2</td>
<td></td>
<td>30.60</td>
<td>0</td>
<td>0.00</td>
</tr>
<tr>
<td>3</td>
<td>RXSS</td>
<td>1</td>
<td></td>
<td>281.80</td>
<td>619</td>
<td>60.04</td>
</tr>
<tr>
<td></td>
<td>SXSS</td>
<td>1</td>
<td></td>
<td>284.90</td>
<td>619</td>
<td>60.04</td>
</tr>
<tr>
<td></td>
<td>RXSS</td>
<td>2</td>
<td></td>
<td>269.30</td>
<td>512</td>
<td>49.66</td>
</tr>
<tr>
<td></td>
<td>SXSS</td>
<td>2</td>
<td></td>
<td>289.20</td>
<td>0</td>
<td>0.00</td>
</tr>
<tr>
<td>4</td>
<td>RXSS</td>
<td>1</td>
<td></td>
<td>2284.30</td>
<td>4991</td>
<td>59.90</td>
</tr>
<tr>
<td></td>
<td>SXSS</td>
<td>1</td>
<td></td>
<td>3200.60</td>
<td>4991</td>
<td>59.90</td>
</tr>
<tr>
<td></td>
<td>RXSS</td>
<td>2</td>
<td></td>
<td>2684.19</td>
<td>4157</td>
<td>49.89</td>
</tr>
<tr>
<td></td>
<td>SXSS</td>
<td>2</td>
<td></td>
<td>3010.10</td>
<td>0</td>
<td>0.00</td>
</tr>
<tr>
<td>2</td>
<td>Mutillidae RXSS</td>
<td>1</td>
<td></td>
<td>56.60</td>
<td>69</td>
<td>60.53</td>
</tr>
<tr>
<td></td>
<td>SXSS</td>
<td>1</td>
<td></td>
<td>61.10</td>
<td>41</td>
<td>35.96</td>
</tr>
<tr>
<td></td>
<td>RXSS</td>
<td>2</td>
<td></td>
<td>74.20</td>
<td>69</td>
<td>60.53</td>
</tr>
<tr>
<td></td>
<td>SXSS</td>
<td>2</td>
<td></td>
<td>80.80</td>
<td>41</td>
<td>35.96</td>
</tr>
<tr>
<td>3</td>
<td>RXSS</td>
<td>1</td>
<td></td>
<td>514.00</td>
<td>619</td>
<td>60.04</td>
</tr>
<tr>
<td></td>
<td>SXSS</td>
<td>1</td>
<td></td>
<td>536.30</td>
<td>302</td>
<td>29.29</td>
</tr>
<tr>
<td></td>
<td>RXSS</td>
<td>2</td>
<td></td>
<td>625.90</td>
<td>619</td>
<td>60.04</td>
</tr>
<tr>
<td></td>
<td>SXSS</td>
<td>2</td>
<td></td>
<td>700.30</td>
<td>302</td>
<td>29.29</td>
</tr>
<tr>
<td>4</td>
<td>RXSS</td>
<td>1</td>
<td></td>
<td>3990.90</td>
<td>4991</td>
<td>59.90</td>
</tr>
<tr>
<td></td>
<td>SXSS</td>
<td>1</td>
<td></td>
<td>4451.70</td>
<td>2398</td>
<td>28.78</td>
</tr>
<tr>
<td></td>
<td>RXSS</td>
<td>2</td>
<td></td>
<td>5246.00</td>
<td>4991</td>
<td>59.90</td>
</tr>
<tr>
<td></td>
<td>SXSS</td>
<td>2</td>
<td></td>
<td>5586.50</td>
<td>2405</td>
<td>28.86</td>
</tr>
</tbody>
</table>
Evaluation in terms of Penetration Testing

Conclusions

1. For DVWA not a single breakthrough was reported on the medium and hard levels
2. Mutillidae and Bodgelt have been hacked on both levels for both types of XSS
3. Properties revealed for designing grammars and payloads for XSS vectors
   - High combinatorial coverage results when no filtering was applied
   - Usage of constraints (i.e. TAG == CLOSINGTAG)
4. No testing quality can be achieved by simply increasing the test inputs as long as the combinatorial grammar is fixed
Evaluation in terms of Combinatorial Testing

Conclusions

1. For every combinatorial interaction strength we get the same coverage of positive results, i.e. vulnerability detections.

2. Interaction of the involved parameters is independent of the SUTs we evaluated.

3. 99.93% reduction of the search space.
   - test all 2-way interactions with 114 inputs out of all 158799 possible inputs (exhaustive search).
   - while still being able to *penetrate* the SUTs.

4. Cost and volume of XSS attack vectors depend on the combinatorial grammar.
   - Adding more types in the grammar increases in logarithmic scale the number of vectors.
   - Adding more derivation rules per type will result in an exponential growth.
What about Test Execution?

**YAKINDU**
- Represent test execution paths as UML state machine diagrams
- Eclipse open-source toolkit

**BURP Suite**
- Commercial penetration testing framework
- Customized in-house with python injector scripts

**Test Oracle**
- Check the server response for payload strings
  - Distinction between general and custom policies
  - Plus the combinatorial oracle
Overview of an XSS Penetration Testing Framework

<table>
<thead>
<tr>
<th>Modelling Phase</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Discretizing the search space // Designer</td>
</tr>
<tr>
<td>• Devise payload grammars // black-box testing</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Test Generation Phase</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Generate CAs from a combinatorial test design tool // ACTS automation</td>
</tr>
<tr>
<td>• Translate abstract tests to XSS attack vectors // bash scripts</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Test Execution Phase</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preprocessing ( \textbf{urls} := \texttt{SPIDER(webpage)} ) // BURP automation</td>
</tr>
<tr>
<td>Injection ( \texttt{XSSINJECTOR(urls,attack vectors)} ) // python scripts</td>
</tr>
<tr>
<td>Oracle Check whether an attack vector is reflected on webpage</td>
</tr>
</tbody>
</table>
# Research Problems

## MOdel-Based SEcurity Testing in Practice (MoBSeTIP)

- **Bridge-FFG Project**
- **Partnership** between SBA Research, Security Research and IST/TU Graz
  - Generate test cases for penetration testing
  - Characterize vulnerabilities
  - Automated detection of such vulnerabilities

## Milestones

- Formalizing security knowledge in terms of policies
- Novel development of combinatorial security testing
- 2nd largest repo for XSS payloads
  - Behind IBM AppScan and ahead of OWASP Xenotix Framework
- Large-scale testing and evaluation of 32 SUTs
Summary

Highlights

1. Automated test cases for penetration testing
2. Combinatorial design approach
   - Combinatorial grammar for XSS attack vectors
   - Mixed-level covering arrays for variable size of parameters values
3. Evaluation and high-coverage results for combinatorial security testing

Future Work

Solve (some) of the research problems..!


Questions - Comments

Thanks for your Attention!

dsimos@sba-research.org