Combinatorial XSS Attack Grammars

XSS Vectors for Everywhere

Bernhard Garn
bgarn@sba-research.org
SBA Research

April 10, 2015
SBA Research, Vienna
Outline

Introduction
  XSS
  Input parameter modelling
  Challenges

Evolution of grammars
  Global grammars
  Subgrammars

Brief note about oracles
Overview of Cross-Site-Scripting (XSS)

XSS vulnerabilities are caused by insufficient input sanitizing/parsing of parameter values of web applications.

- XSS remains one of the top vulnerabilities in OWASP Top 10 Web Application Security Risks:
  - 2010: 2nd
  - 2013: 3rd

Threat:
Execution of malicious JavaScript in the victim’s browser!
Testing overview – bird’s eye view

- SUT
- Test suite
- Model
- Test suite generator
- Check output
- Policy

PASS
FAIL
Cross-Site-Scripting (XSS)

Scope: We focus on reflected and stored XSS

Example: The response sent from a server contains parts of the submitted request-url reflected identically the body

Goal: Make injected JavaScript executable!

- High quality XSS vectors are of utmost importance to find/reveal vulnerabilities!
- Various generation methods (fuzzing, manually crafted list, learning approaches, CT)
Increase in web-apps: injection possibilities everywhere!
Structure of an XSS Attack Vector

Valid URLs vs Attack Vectors


Sample of XSS attack vectors

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

Input Parameter Model

- Parameters ⇒ parts of the url
- Parameter value selection: Input parameter modelling via categories

Combinatorial form of an XSS attack vector

AV := (parameter₁, parameter₂, ..., parameterₖ)
Challenges

Model creation can be considered as multi-objective optimization problem

- Network latency
- Avoidance of DoSing
- Sizes of the arrays (CAN)

Aim:

High quality, highly diverse attack vectors
Resulting Test Vectors

Generation parameters:

- Class of array (MCA)
  - Algorithms (IPO-family)
  - ACTS (Courtesy of NIST)
  - THANK YOU VERY MUCH
- Strength $t$
- Constraint solver
- New/extend/base choice (corresponds to modelling)
- Priorization

Yields files with XSS attack vectors, one attack vector per line, corresponding to rows in arrays.
Example
Definition pos_vecs

3,1,4,1,1,1,1,1,4,1,4

6,1,7,2,3,1,1,2,7,3,7

9,1,10,3,2,1,1,3,10,2,1

11,1,12,2,1,2,1,1,1,2,3

14,2,1,2,3,2,1,3,4,2,6

15,3,2,1,1,3,1,1,5,3,7

1,1,3,3,1,1,2,2,6,1,8

\[\begin{array}{c}
3,1,4,1,1,1,1,1,4,1,4 \\
6,1,7,2,3,1,1,2,7,3,7 \\
9,1,10,3,2,1,1,3,10,2,1 \\
11,1,12,2,1,2,1,1,1,2,3 \\
14,2,1,2,3,2,1,3,4,2,6 \\
15,3,2,1,1,3,1,1,5,3,7 \\
1,1,3,3,1,1,2,2,6,1,8
\end{array}\]
Evalution of grammars – criteria

- Test suite size

$$ER := \frac{\# \text{pos_vecs}}{\# \text{test-suite}}$$

- Simple t-way combination coverage of passing tests (CCM tool)
- Correlation of parameter values and specific sanitizing functions for input checking
- Data mining / machine learning

Recent results
See our contribution to IWCT 2015
Milestones in the Grammar Development

<table>
<thead>
<tr>
<th>Global grammars</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Global = generic attack grammars for XSS</td>
</tr>
<tr>
<td>• Publications:</td>
</tr>
<tr>
<td>▶ Proof of Concept [AST 2014]</td>
</tr>
<tr>
<td>▶ Global grammar $G$ [JAMAICA 2014]</td>
</tr>
<tr>
<td>▶ Global grammar constrained $G_c$ [IWCT 2015]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Subgrammars [UNPUBLISHED]</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Optimized grammars to attack specific contexts in an HTML page</td>
</tr>
<tr>
<td>• Multiple refinement iterations</td>
</tr>
</tbody>
</table>
• Inception of attack pattern + CT for Security Testing
• Important differences: generation, structure and execution of test cases
• Algorithm: IPOG
• Sizes of attack suites:
  ▶ $t = 2$: 114
  ▶ $t = 3$: 1031
  ▶ $t = 4$: 8332

Grammar:

MCA ($t$, 12, (1, 1, 1, 1, 3, 3, 6, 9, 9, 10, 10, 10))

FOBRACKET(1) ::= <
TAG(10) ::= img | frame | src | script | body | HEAD | ...
FCBRACKET(1) ::= >
QUOTE1(3) ::= | | null
SPACE(9) ::= \n | \t | \r | \r
 | \a | \b | \c | ...
EVENT(10) ::= onclick | onmouseover | onerror ... 
SPACE2(9) ::= \n | \t | \r | \r
 | \a | \b | \c | ... 
QUOTE2(3) ::= | | null 
PAYLOAD(6) ::= alert(1) | alert(0) | ...
LOBRACKET(1) ::= </
CLOSINGTAG(10) ::= img | frame | src | script | body | ...
LCBRACKET(1) ::= >

Constraint

tag = closingtag
• Configuration:  
  **MCA** \((t, 11, (3, 3, 3, 3, 3, 3, 9, 11, 14, 15, 23))\)

  ▶ 11 parameters  
  ▶ whitespace modeling  
  ▶ no constraints

• Algorithm: IPOG

• Comparison of tools (ZAP, Burp)

• Sizes of attack suites:
  ▶ \(t = 2\): 345  
  ▶ \(t = 3\): 4875  
  ▶ \(t = 4\): 54706

---

**Grammar G**

\[
\begin{align*}
  JSO(15) &::= <script> \mid <img \mid \ldots \\
  WS1(3) &::= \text{tab} \mid \text{space} \mid \ldots \\
  INT(14) &::= ';' \mid '>>' \mid \ldots \\
  WS2(3) &::= \text{tab} \mid \text{space} \mid \ldots \\
  EVH(3) &::= \text{onLoad(} \mid \text{onError(} \mid \ldots \\
  WS3(3) &::= \text{tab} \mid \text{space} \mid \ldots \\
  PAY(23) &::= \text{alert('XSS')} \mid \text{ONLOAD=alert('XSS')} \mid \ldots \\
  WS4(3) &::= \text{tab} \mid \text{space} \mid \ldots \\
  PAS(11) &::= ')' \mid '>' \mid \ldots \\
  WS5(3) &::= \text{tab} \mid \text{space} \mid \ldots \\
  JSE(9) &::= </script> \mid > \mid \ldots 
\end{align*}
\]
• Configuration: \( \text{MCA} \left( t, 11, (3, 3, 3, 3, 3, 3, 9, 11, 14, 15, 23) \right) \)

  ▶ 11 parameters
  ▶ whitespace modeling
  ▶ CONSTRAINTS

• Algorithm: IPOG, IPOG-F
• Comparison of algorithms

### Some constraints of \( G_c \)

- \((\text{JSO}=5) \Rightarrow (\text{JSE}=5 \mid \text{JSE}=6 \mid \text{JSE}=7 \mid \text{JSE}=8 \mid \text{JSE}=9)\)
- \((\text{EVH}=1) \Rightarrow (\text{PAY}=12 \mid \text{PAY}=14 \mid \text{PAY}=16 \mid \text{PAY}=17 \mid \text{PAY}=18 \mid \text{PAY}=19)\)
- \((\text{WS}_1=\text{WS}_2 \land \text{WS}_2=\text{WS}_3 \land \text{WS}_3=\text{WS}_4 \land \text{WS}_4=\text{WS}_5)\)

### Table: Different sizes of test suites

<table>
<thead>
<tr>
<th>Str.</th>
<th>( G )</th>
<th>( G_c )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>IPOG</td>
<td>IPOG-F</td>
</tr>
<tr>
<td>2</td>
<td>345</td>
<td>345</td>
</tr>
<tr>
<td>3</td>
<td>4875</td>
<td>4830</td>
</tr>
<tr>
<td>4</td>
<td>53706</td>
<td>53130</td>
</tr>
</tbody>
</table>
Subgrammars [UNPUBLISHED]

- Can be thought of as “specific attacks”
- From 10 to 4 to 3 subgrammars (including new constraints each):
  - inside `element`
  - inside `attribute`
  - inside `JavaScript`
- Parameter values specific to context of subgrammar
- Evaluation pending, so far highly effective
10 Subgrammars

@element

sg1
MCA (t, 8, (2, 6, 5, 3, 2, 5, 6, 3))

sg2
MCA (t, 12, (2, 6, 5, 3, 2, 1, 2, 5, 3, 6, 3, 3))

sg3
MCA (t, 15, (2, 5, 3, 2, 2, 2, 5, 3, 1, 1, 2, 6, 1, 3, 3))

sg10
MCA (t, 9, (2, 2, 2, 5, 4, 1, 6, 4, 3))

@attribute

sg4
MCA (t, 7, (4, 3, 3, 2, 5, 6, 3))

sg5
MCA (t, 8, (4, 3, 2, 5, 3, 6, 3, 3))

sg6
MCA (t, 3, (1, 2, 6))

sg7
MCA (t, 4, (4, 2, 5, 6))

sg8
MCA (t, 5, (4, 3, 1, 6, 1))

sg9
MCA (t, 3, (2, 6, 2))

@JavaScript

sg4
MCA (t, 7, (4, 3, 3, 2, 5, 6, 3))

sg7
MCA (t, 4, (4, 2, 5, 6))

sg8
MCA (t, 5, (4, 3, 1, 6, 1))

sg9
MCA (t, 3, (2, 6, 2))
# 3 Subgrammars

<table>
<thead>
<tr>
<th>Subgrammar</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>@element</strong></td>
<td>MCA ((t, 5, (3, 3, 7, 3, 7)))</td>
</tr>
<tr>
<td>SG01</td>
<td>delimiter01, closing-angle-bracket, opening-tag, payload, closing tag</td>
</tr>
<tr>
<td>Constr</td>
<td>opening-tag=2 =&gt; closing-tag=2</td>
</tr>
<tr>
<td><strong>@attribute</strong></td>
<td>MCA ((t, 4, (3, 6, 3, 6)))</td>
</tr>
<tr>
<td>SG02</td>
<td>delim01, attribute, payload, delim02</td>
</tr>
<tr>
<td>Const</td>
<td>(delim01=1) =&gt; (delim02=1</td>
</tr>
<tr>
<td><strong>@JavaScript</strong></td>
<td>MCA ((t, 3, (3, 3, 3)))</td>
</tr>
<tr>
<td>SG03</td>
<td>delim03, payload, delim04</td>
</tr>
<tr>
<td>Constr</td>
<td>(delim03=3) =&gt; (delim04=3)</td>
</tr>
</tbody>
</table>
Oracle Consideration

- Reflection oracle
  - via string matching
  - false positives, false negatives
  - tools seem not to agree

- Browser oracle
  - zero false positives (!)

Change of oracle might require change of some parameters/parameter values
Change of payload

• Reflection oracle
  ▶ possible parameter values: alert(‘1’), alert(document.cookie)
  ▶ any malicious code possible

• Browser oracle
  ▶ at this location we call home
  ▶ coordination with logging infrastructure

Testing with different oracles can require re-generation of attack suites
• Everything shown is completely scripted (i.e. automated)!

Thank you very much for your attention!

Questions?