Another Brick off The Wall: Deconstructing Web Application Firewalls Using Automata Learning

George Argyros, Ioannis Stais

Joint Work with:
Suman Jana, Angelos D. Keromytis, Aggelos Kiayias
Overview

- A journey in the world of:
  - Code Injection attacks.
  - Web Application Firewalls.
  - Parsers.
  - Learning algorithms.
- And newly discovered vulnerabilities :)

Code Injection Attacks

• SQLi, XSS, XML, etc…

• Not going anywhere anytime soon.

• 14% increase in total web attacks in Q2 2016 [1]

• 150% - 200% increase in SQLi and XSS attacks in 2015 [2]

Code Injection is a Parsing Problem

Input data → Web Application → Injection attack → Language Runtime
Code Injection is a Parsing Problem

Input data is parsed incorrectly

Web Application

Injection attack

Language Runtime
Code Injection is a Parsing Problem

Web application parsers are doing a really bad job in parsing user inputs.

Diagram:
- Input data → Web Application
- Web Application parses input data incorrectly → Injection attack → Language Runtime

Legend:
- Input data
- Injection attack
- Language Runtime

Explanatory Text:
Web application parsers are doing a really bad job in parsing user inputs.
Web Application Firewalls
(or solving parsing problems with parsing)
Web Application Firewalls

- Monitor traffic at the Application Layer: *Both HTTP Requests and Responses*.
- Detect and Prevent Attacks.
- Cost-effective compliance with PCI DSS requirement 6.6 [1]
WAFs Internals

User Input

Normalization

Rulesets Matching

Attack Mitigation

<Script>alert(1);</Script>
WAFs Internals

User Input → Normalization → Rulesets Matching → Attack Mitigation

- `<ScRipt>alert(1);</ScRipt>`
- `<script>alert(1);</script>` Lower Case
WAFs Internals

User Input → Normalization → Rulesets Matching → Attack Mitigation

Matched Rule:

<script>alert(1);</script>
WAFs Internals

User Input -> Normalization -> Rulesets Matching -> Attack Mitigation

<ScRipt>alert(1);</ScRipT>

<scrpit>alert(1);</script>
Lower Case

<scrpit>alert(1);</script>
Matched Rule: <script>.*</script>
WAFs Internals

User Input

Normalization

Rulesets Matching

Attack Mitigation

Tokenising

Event Correlation

<ScRipt>alert(1);</ScRipt>

<ScRipt>alert(1);</ScRipt>

Lower Case

<ScRipt>alert(1);</ScRipt>

Matched Rule:
<ScRipt>.*</ScRipt>
WAFs Internals

User Input → Normalization → Rulesets Matching → Attack Mitigation

1. `<script>`
2. `alert(1);`
3. `</script>`

Lower Case

Matched Rule: `<script>`.*`</script>`
WAFs Internals

1. `<script>alert(1);</script>`
2. `<script>alert(1);</script>`
3. `<script>alert(1);</script>`
4. `<script>alert(1);</script>`

- Lower Case Matched Rule: `<script>.*</script>`
- 4 Rules Matched
- 1. Session/User history

User Input → Normalization → Rulesets Matching → Event Correlation → Attack Mitigation
WAF Rulesets

- **Signatures:** Strings or Regular Expressions

_E.g., [PHPIDS Rule 54] Detects Postgres pg_sleep injection, waitfor delay attacks and database shutdown attempts:_

```
(?:select\s*pg_sleep)|(?:waitfor\s*delay\s?"\+\s?d)|(?:\;\s*shutdown\s*\;\;\;\;\;\--\#|\|\|\|\|\|\|\|\|\;)
```
WAF Rulesets

- **Signatures:** *Strings or Regular Expressions*

- **Rules:** Logical expressions and Condition/Control Variables

E.g., ModSecurity CRS Rule 981254:

```
SecRule REQUEST_COOKIES|\!REQUEST_COOKIES:__/\_utm/\|\!REQUEST_COOKIES:/ _pk_ref/\|\!REQUEST_COOKIES_NAMES|ARGS_NAMES|ARGS|XML:/\* "(?i:\(?:select\s*? pg_sleep)\)?(?:waitfor\s*?delay\s?\|\"\"]+\s?\d)\)?;\s*?shutdown\s*?\(?:\;|--|#|\/{)}}" "phase: 2,capture,t:none,t:urlDecodeUni,block, setvar:tx.sql_injection_score=+1,setvar:tx.anomaly_score=+%{tx.critical_anomaly_score},setvar:'tx.%{tx.msg}-OWASP_CRS/WEB_ATTACK/SQLI-%{matched_var_name}=%{tx.0}"
WAF Rulesets

• **Signatures:** Strings or Regular Expressions

• **Rules:** Logical expressions and Condition/Control Variables

• **Virtual Patches:** Application Specific Patches

E.g., ModSecurity: Turns off autocomplete for the forms on login and signup pages

SecRule REQUEST_URI "^(/login|/signup)" "id:1000,phase:4,chain,nolog,pass"
SecRule REQUEST_METHOD "@streq GET" "chain"
SecRule STREAM_OUTPUT_BODY "@rsub s/<form /<form autocomplete="off" /"
WAF Rule sets

- **Signatures:** Strings or Regular Expressions
- **Rules:** Logical expressions and Condition/Control Variables
- **Virtual Patches:** Application Specific Patches
- **PHPIDS has more than 420K states**
- **Shared between different WAFs and Log Auditing Software:** PHPIDS, Expose, ModSecurity
Why Bypasses Exist
Why Bypasses Exist

- Simple hacks:
  
  • Lack of support for different protocols, encodings, contents, etc

  • Restrictions on length, character sets, byte ranges, types of parameters, etc
Why Bypasses Exist

- Rulesets sharing mistakes:
  - Normalisation and Rulesets Failure
Why Bypasses Exist

- Rulesets sharing mistakes:
  
  • Normalisation and Rulesets Failure

User Input → Normalization → Rulesets Matching

PHPIDS 0.7.0
Why Bypasses Exist

- Rulesets sharing mistakes:
  
  • Normalisation and Rulesets Failure

User Input → Normalization → Rulesets Matching

\"s*(src|style|on\w+)[^\w]*=\"s*\")

\"" onclick='a()'>

PHPIDS 0.7.0

MATCHED!
Why Bypasses Exist

- Rule sets sharing mistakes:
  
  • Normalisation and Rulesets Failure

```
x' onclick='a()'>

MATCHED!
```

```
"\s*(src|style|on\w+)\s*=\s*"
```

User Input

Normalization

Rulesets Matching
Why Bypasses Exist

- **Rulesets sharing mistakes:**

  • Normalisation and Rulesets Failure

User Input → Normalization → Rulesets Matching

```
x' onclick='a()'>
'' "s*(src|style|on\w+|s*=s*)"
MATCHED!
```
Why Bypasses Exist

- Rule sets sharing mistakes:
  
  • Normalisation and Rule sets Failure
Why Bypasses Exist

- Rulesets sharing mistakes:
  
  • Normalisation and Rulesets Failure

  ```
  x' onclick='a()'>
  
  "\s*(src|style|on\w+)\s*=\s*"
  ```

  Expose 2.4.0

  BYPASS!
Why Bypasses Exist

- Critical WAF components are not being updated:

  • E.g, ModSecurity *libinjection* library
Why Bypasses Exist

- Critical WAF components are not being updated:
  
  • E.g, ModSecurity *libinjection* library
Why Bypasses Exist

- The Real Fundamental Reasons:
  
  • Insufficient Signatures & Weak Rules
  
  • Detecting vulnerabilities without context is HARD
Our Goal

1. Formalize knowledge in code injection attacks variations using context free grammars and automata.

2. Use Learning algorithms to expand this knowledge by inferring system specifications.
Using parsers to break parsers
Regular Expressions and Finite Automata

Every regular expression can be converted to a Deterministic Finite Automaton.

\((.*)\)man
Context Free Grammars

- Superset of Regular Expressions.
- Mostly used to write programming languages parsers.
- Equivalent to a DFA with a stack.
- Can be used to count.
  - Example: matching parentheses.

\[
\begin{align*}
E & \rightarrow N \\
E & \rightarrow E \text{ Op } E \\
E & \rightarrow ( E ) \\
N & \rightarrow N \ N \\
\text{Op} & \rightarrow + \\
\text{Op} & \rightarrow - \\
\text{Op} & \rightarrow * \\
N & \rightarrow [0-9]
\end{align*}
\]
_ATTACK OF THE GRAMMARS_

- Context Free Grammars can be used to encode attack vectors.
- Assume we would like to inject code into the query:
  - "SELECT * FROM users WHERE id=$id;"
- The valid suffixes (injections) for this query can be encoded as a CFG!
Cross checking regular expressions with grammars is easy!

Why should I care?

Find an SQL Injection attack in the Grammar $G$ which is not rejected by the filter $F$. 
However…

- In reality, we do not know the language parsed by most implementations.
  - MySQL is parsing a different SQL flavor than MS-SQL.
  - Browsers are **definitely not** parsing the HTML standard.
  - WAFs are doing much more than a simple RE matching.
Learning to Parse

• **Our Approach:** Use Learning algorithms in order to infer the specifications of parsers and WAFs.
  - Cross check the inferred models for vulnerabilities.

• By using learning we can actively figure out important details of the systems.
Learning Automata
Learning Automata

- **Active Learning** algorithm.
  - Instead of learning from corpus of data, query the program with input of his choice.
- Eventually a model is generated.
- Discovered inconsistencies of the model is used to refine it.
Learning Model

- Learning Algorithm
- Parser P
Learning Model

Membership Query

Learning Algorithm

Parser P
Learning Model

Membership Query

Learning Algorithm

Is $s$ accepted by $P$?

Parser $P$

string $s$
Learning Model

Membership Query

- Model H
- Learning Algorithm
  - string $s$
- Parser P
  - Is $s$ accepted by P?
Learning Model

Model H → Learning Algorithm

string $s$ → Parser P

Is $s$ accepted by P?
Learning Model

Equivalence Query

Model H

Learning Algorithm

string s

Is s accepted by P?

Parser P
Learning Model

Equivalence Query

Is \( s \) accepted by \( P \)?

Is \( H \) a correct model of \( P \)?
Yes, or provide counterexample.
Learning Model

- Model $H$
- Learning Algorithm
- Parser $P$
- Is $s$ accepted by $P$?
- Is $H$ a correct model of $P$?

Yes, or provide counterexample.
Learning DFAs

• Angluin’s algorithm is an active learning algorithm for learning DFAs.

• Learns the target DFA using a table data structure called the observation table.

• Let’s use it to learn the regular expression (.*)<a(.*)

  - Aggressive filtering of anchor tags.
Learning DFAs

<table>
<thead>
<tr>
<th>εT</th>
<th>ε</th>
</tr>
</thead>
<tbody>
<tr>
<td>ε</td>
<td></td>
</tr>
<tr>
<td>ε&lt;</td>
<td></td>
</tr>
<tr>
<td>εa</td>
<td></td>
</tr>
</tbody>
</table>
Learning DFAs

Empty string
Learning DFAs

Strings for “testing” states. (Distinguishing strings)

Empty string
Learning DFAs

Strings accessing different states in the target automaton. (Access Strings)

Strings for “testing” states. (Distinguishing strings)

Empty string
Learning DFAs

Strings accessing different states in the target automaton. (Access Strings)

Strings which transition from the above states.

Empty string

Strings for “testing” states. (Distinguishing strings)
Learning DFAs

Strings accessing different states in the target automaton. (Access Strings)

Strings which transition from the above states.

Empty string

Strings for “testing” states. (Distinguishing strings)

An entry is filled by concatenating the row and column string and filling with the output of the automaton.
Learning DFAs

Model:

Target:

\[
\begin{array}{c}
q_0 \\
< \\
a \\
<,a \\
\end{array}
\]

\[
\begin{array}{c}
\varepsilon \\
< \\
a \\
<,a \\
\end{array}
\]

\[
\begin{array}{c|c}
\text{OT} & \varepsilon \\
\varepsilon & < \\
\varepsilon & \varepsilon a \\
\end{array}
\]
Learning DFAs

Model:

Target:

<table>
<thead>
<tr>
<th>State</th>
<th>OT</th>
<th>ε</th>
</tr>
</thead>
<tbody>
<tr>
<td>q_0</td>
<td>ε</td>
<td>0</td>
</tr>
<tr>
<td>ε&lt;</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>εa</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Learning DFAs

Model:

Target:

<table>
<thead>
<tr>
<th>OT</th>
<th>ε</th>
</tr>
</thead>
<tbody>
<tr>
<td>ε</td>
<td>0</td>
</tr>
<tr>
<td>ε&lt;</td>
<td>0</td>
</tr>
<tr>
<td>εa</td>
<td>0</td>
</tr>
</tbody>
</table>

q_0

trans.
Learning DFAs

Model:

Target:

<table>
<thead>
<tr>
<th>OT</th>
<th>ε</th>
</tr>
</thead>
<tbody>
<tr>
<td>ε</td>
<td>0</td>
</tr>
<tr>
<td>ε&lt;</td>
<td>0</td>
</tr>
<tr>
<td>εa</td>
<td>0</td>
</tr>
</tbody>
</table>
Equivalence Query

Counterexample analysis

CE: <aaa<<a

Add a new column with character “a” in the OT.
Learning DFAs

Model:

Target:

<table>
<thead>
<tr>
<th>OT</th>
<th>ε</th>
<th>a</th>
</tr>
</thead>
<tbody>
<tr>
<td>q_0 trans.</td>
<td>ε</td>
<td>0</td>
</tr>
<tr>
<td>ε&lt;</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>εa</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
Learning DFAs

Model:

Target:

<table>
<thead>
<tr>
<th>OT</th>
<th>ε</th>
<th>a</th>
</tr>
</thead>
<tbody>
<tr>
<td>ε</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ε&lt;</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>εa</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Learning DFAs

Model:

Target:

Must be a new state

<table>
<thead>
<tr>
<th>OT</th>
<th>ε</th>
<th>a</th>
</tr>
</thead>
<tbody>
<tr>
<td>ε</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ε&lt;</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>εa</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Learning DFAs

Model:

Target:
Learning DFAs

Model:

Target:

<table>
<thead>
<tr>
<th>OT</th>
<th>ε</th>
<th>a</th>
</tr>
</thead>
<tbody>
<tr>
<td>q_0</td>
<td>ε</td>
<td>0</td>
</tr>
<tr>
<td>q_1</td>
<td>ε&lt;</td>
<td>0</td>
</tr>
<tr>
<td>q_0</td>
<td>εa</td>
<td>0</td>
</tr>
<tr>
<td>q_1</td>
<td>ε&lt;</td>
<td>0</td>
</tr>
<tr>
<td>Trans.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>q_0</td>
<td>q_1</td>
<td>1</td>
</tr>
<tr>
<td>Trans.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Learning DFAs

Model:

Target:

Must be a new state

<table>
<thead>
<tr>
<th>OT</th>
<th>ε</th>
<th>a</th>
</tr>
</thead>
<tbody>
<tr>
<td>q_0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>q_1</td>
<td>ε (&lt;</td>
<td>1</td>
</tr>
<tr>
<td>ε&lt;</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>εa</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ε&lt;</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>&lt;a</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>&lt;=</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Learning DFAs

Model:

Target:

\[
\begin{align*}
\text{OT} & \\ q_0 & \varepsilon & 0 & 0 \\
\varepsilon & 0 & 1 \\
\varepsilon < & 1 & 1 \\
\langle a & 0 & 0 \\
\langle a & 0 & 1 \\
\langle a & 1 & 1 \\
\langle a & 0 & 1 \\
\langle a & 0 & 1 \\
\end{align*}
\]
Learning DFAs

Model:

Target:

<table>
<thead>
<tr>
<th>OT</th>
<th>ε</th>
<th>a</th>
</tr>
</thead>
<tbody>
<tr>
<td>ε</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ε&lt;</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>&lt;a</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>εa</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ε&lt;</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>&lt;a</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>&lt;&lt;</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>&lt;aa</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>&lt;a&lt;</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Learning DFAs

Model:

Target:

states

<table>
<thead>
<tr>
<th>OT</th>
<th>ε</th>
<th>a</th>
</tr>
</thead>
<tbody>
<tr>
<td>q_0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>q_1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>q_2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>&lt;a</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ε&lt;</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>&lt;a</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>&lt;aa</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>&lt;a&lt;</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Learning DFAs

Model:

Target:

states

transitions

<table>
<thead>
<tr>
<th>OT</th>
<th>ε</th>
<th>a</th>
</tr>
</thead>
<tbody>
<tr>
<td>q_0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>q_1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>q_2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>&lt;a</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>εa</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>ε&lt;</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>&lt;&lt;</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>&lt;aa</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>&lt;a&lt;</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Learning DFAs

Model:

Target:

states

<table>
<thead>
<tr>
<th>OT</th>
<th>ε</th>
<th>a</th>
</tr>
</thead>
<tbody>
<tr>
<td>q_0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>q_1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>q_2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>q_0</td>
<td>εa</td>
<td>0</td>
</tr>
<tr>
<td>q_1</td>
<td>ε&lt;</td>
<td>0</td>
</tr>
<tr>
<td>q_2</td>
<td>&lt;a</td>
<td>1</td>
</tr>
<tr>
<td>trans.</td>
<td>&lt;a</td>
<td></td>
</tr>
</tbody>
</table>

transitions

ε  a
ε  εa
ε< εa
ε< εa
ε< εa
<   εa
<   εa
<   εa
<   εa
<   εa
Learning DFAs

Model:

Target:

states

transitions

ε
<

<
a

<,a

q_0
ε
0
0

q_1
ε<
0
1

q_2
<εa
1
1

ε
q_0
trans.
εa
0
0

ε<
q_1
trans.
ε<
0
1

<εa
q_2
trans.
<εa
1
1

<aa
0
1

<aa
1
1

<aa
1
1

ε
<
a

<,a
Learning DFAs

• This algorithm is inefficient for large alphabets/automata.

• For just one PHPIDS Rule (id. 72):
  
  • \(((\{s\}^{n}*(top|this|window|content|self|frames|\_content))(V\{s\}^{n}*[gimx]*\{s\}^{n}*[\(\)]\}))\(([^\ ]{s}^{n}={s}^{n}*script))\((\{s\}^{n}*constructor)\((default\{s\}+xml\{s\}+namespace\{s\}^{n}*[}=)\((V\{s\}^{n}+[\^\+]+\{s\}^{n}+[\{s\}]^{n}\}))\)

  - 72 states when represented as a DFA.
  
  - The OT will have ~650k entries.

• We need a faster algorithm in order to check real systems!
Symbolic Finite Automata

✓ Efficient modeling of large alphabets.

✓ We designed a novel, efficient learning algorithm.

✓ Details in the whitepaper!
Bootstrapping Automata Learning

• Similar concept with seed inputs in fuzzers.
  - Provide sample inputs and learning algorithm will discover additional states in the parser.

• Utilize previously inferred models, specifications, etc.

• Seed inputs are guiding the learning algorithm.

• Details in the white paper!
Grammar Oriented Filter Auditing (GOFA)
Assume that we are given a grammar with attacks.

How do we utilize it with the learning algorithm?

Main idea:
Use the grammar to drive the learning procedure.
Grammar Oriented Filter Auditing

Context Free Grammar $G$

... 

select_exp: SELECT name
any_all_some: ANY | ALL
column_ref: name
parameter: name

Learning Algorithm
Grammar Oriented Filter Auditing

Step 1:
Learn a model of the WAF.
Grammar Oriented Filter Auditing

Step 1:
Learn a model of the WAF.

Learning Algorithm
Grammar Oriented Filter Auditing

Step 1:
Learn a model of the WAF.

select_exp: SELECT name
any_all_some: ANY | ALL
column_ref: name
column parameter: name

Context Free Grammar $G$

WAF Model → Learning Algorithm → Firewall
Grammar Oriented Filter Auditing

Step 2:
Find a vulnerability in the model using the grammar.

Context Free Grammar $G$ vs WAF Model
Grammar Oriented Filter Auditing

Context Free Grammar $G$ vs WAF Model

Learning Algorithm
Grammar Oriented Filter Auditing

Step 3:
Verify WAF vulnerability.

Context Free Grammar $G$

WAF Model

Learning Algorithm

VS

WAF Model
Grammar Oriented Filter Auditing

Step 3:
Verify WAF vulnerability.

Context Free Grammar $G$ vs WAF Model

Learning Algorithm

Candidate Bypass
Grammar Oriented Filter Auditing

Context Free Grammar G vs WAF Model Learning Algorithm vs Candidate Bypass
Grammar Oriented Filter Auditing

Step 4:
- or refine model and repeat.

Context Free Grammar $G$ vs WAF Model

Candidate Bypass

Learning Algorithm
Grammar Oriented Filter Auditing

Step 4:

or refine model and repeat.

counterexample (false positive)
Vulnerabilities
GOFA SQL Injections

• Grammar for extending search conditions:
  
  ```sql
  select * from users where user = admin and email = ${_GET[c]}
  ```
GOFA SQL Injections

• Grammar for extending search conditions:

```
select * from users where user = admin and email = $_GET[c]
```
GOFA SQL Injections

• Authentication bypass using the vector: `or exists (select 1)`

Example:

select * from users where username = `$_GET['u']` and password = `$_GET['p']`;

select * from users where username = `admin` and password = `a` or exists (select 1)

Affected: ModSecurity Latest CRS, PHPIDS, WebCastellum, Expose
GOFA SQL Injections

• Authentication bypass using the vector:  

1 or a = 1

1 or a like 1

Example:

select * from users where username = $_GET['u'] and password = $_GET['p'];

select * from users where username = admin and password = 1 or isAdmin like 1

Affected: ModSecurity Latest CRS, PHPIDS (only for statement with ‘like’), WebCastellum, Expose
GOFA SQL Injections

• Columns/variables fingerprinting using the vectors:  
  \[ \text{and exists (select a)} \]
  \[ \text{a or a > any select a} \]

Example:

select * from users where username = admin and id = $_GET['u']);

select * from users where username = admin and id = 1 \text{ and exists (select email)}

Affected: ModSecurity Latest CRS, PHPIDS, WebCastellum, Expose
GOFA SQL Injections

• Grammar for extending select queries:
  
  \[
  \text{select} \ast \text{ from users where user = } \$_\text{GET}[c] \]
  

GOFA SQL Injections

• Grammar for extending select queries:

```
select * from users where user = ${_GET[c]}
```

S: A main
main: query_exp
query_exp: groupby_exp | order_exp | limit_exp | procedure_exp | into_exp | for_exp | lock_exp | ; select_exp | union_exp | join_exp

groupby_exp: GROUP BY column_ref ascdesc_exp
order_exp: ORDER BY column_ref ascdesc_exp
limit_exp: LIMIT intnum
into_exp: INTO output_exp intnum
procedure_exp: PROCEDURE name ( literal )
literal: string | intnum
select_exp: SELECT name
union_exp: UNION select_exp
ascdesc_exp: ASC | DESC
column_ref: name
join_exp: JOIN name ON name
for_exp: FOR UPDATE
lock_exp: LOCK IN SHARE MODE
output_exp: OUTFILE | DUMPFILE
string: name
intnum: 1
name: A
GOFA SQL Injections

• Data retrieval bypass using the vector: 

\[
\text{1 right join a on a = a}
\]

Example:

select * from articles left join authors on author.id=$\_\text{GET['id']}$

select * from articles left join authors on author.id= \text{1 right join users on author.id = users.id}

Affected: ModSecurity Latest CRS, WebCastellum
GOFA SQL Injections

- Columns/variables fingerprinting using the vectors: a group by a asc

Example:

```sql
select * from users where username = @$_GET['u'];
```

```sql
select * from users where username = admin group by email asc
```

Affected: ModSecurity Latest CRS, PHPIDS, WebCastellum, Expose
GOFA SQL Injections

• Columns/variables fingerprinting using the vectors:

Example:

select * from users where username = $_GET['u'];

select * from users where username = admin procedure analyze()

Affected: libInjection
SFADiff: Learning Attack Vectors
Available grammars are not always good for finding vulnerabilities.

Most XSS bypasses result from attack vectors deviating from the HTML standard.

- `<IMG SRC="javasscript:alert('XSS');">`

- Tons of other examples.

Use the same learning approach to infer the HTML parser specification!
SFADiff

WAF

Browser
SFADiff

- Automata Learner
- WAF
- Automata Learner
- Browser
SFADiff

Automata Learner → WAF

Automata Learner → Browser
SFADiff
SFADiff

WAF model

VS

Automata Learner

WAF

Automata Learner

Browser

HTML Model
SFADiff

VS

WAF model

Automata Learner

Candidate bypasses

WAF

Browser

Automata Learner

Candidate bypasses

HTML Model
SFADiff

VS

WAF model

Automata Learner

Automata Learner

Candidate bypasses

candidate bypasses

counterexamples

HTML Model

WAF

Browser
SFADiff

VS

Bypasses

WAF model

HTML Model

Automata Learner

Automata Learner

WAF

Browser

candidate bypasses

counterexamples

candidate bypasses
This site can't be reached

Starting diff_browser_waf:
Initializing learning procedure.
Starting WebSocket Server at port 8080: OK
Starting HTTP Server at port 8080: OK
Please connect your Browser at http://localhost:8088

Checking for rst module: OK
Checking for python-pdu module: OK
Entering module diff_browser_waf

George Argyros, Ioannis Stais
This site can't be reached

192.168.254.254

George Argyros, Ioannis Stais

Checking for rst module: OK
Checking for pythonpd module: OK
Entering module diff_browser_waf

Starting diff_browser_waf:
Initializing learning procedure.
Starting WebSocket Server at port 8000: OK
Starting HTTP Server at port 8080: OK
Please connect your Browser at http://localhost:8080
Verifying Web Socket connection: OK
Awaiting initialization command: OK
Initializing learning procedure.

Initialized from DFA em_vector table is the following:

- Processing counterexample `<p onclick=() with length 15.`
- Processing counterexample `<p onclick=a() with length 26.`
- Generated conjecture machine with 26 states.
- Generated conjecture machine with 24 states.
- Processing counterexample `<p onclick=a() with length 15.`
- Processing counterexample `<p onclick=a() with length 15.`
- Generated conjecture machine with 29 states.
- Generated conjecture machine with 24 states.
- Processing counterexample `<p onclick=a();p0=a() with length 26.`
- Processing counterexample `<p onclick=a() with length 16.`
- Generated conjecture machine with 29 states.
- Generated conjecture machine with 24 states.
- Processing counterexample `<p onclick=a();nclick=a() with length 26.`
- Processing counterexample `<p onclick=a();p0=a() with length 15.`
- Generated conjecture machine with 29 states.
- Generated conjecture machine with 24 states.
- Processing counterexample `<p onclick=a();nclick=a() with length 26.`
- Processing counterexample `<p onclick=a();p0=a() with length 15.`
- Generated conjecture machine with 29 states.
- Generated conjecture machine with 24 states.
- Processing counterexample `<p onclick=a();nclick=a() with length 26.`
- Processing counterexample `<p onclick=a();p0=a() with length 15.`
- Generated conjecture machine with 29 states.
- Generated conjecture machine with 24 states.
- Processing counterexample `<p onclick=a();nclick=a() with length 26.`
- Processing counterexample `<p onclick=a();p0=a() with length 15.`
- Generated conjecture machine with 29 states.
- Generated conjecture machine with 24 states.
- Processing counterexample `<p onclick=a();nclick=a() with length 26.`
- Processing counterexample `<p onclick=a();p0=a() with length 15.`
- Generated conjecture machine with 29 states.
- Generated conjecture machine with 24 states.

Generated conjecture machine with 22 states.
SFADiff XSS Bypass

• XSS Attack vectors in PHPIDS 0.7/ Expose 2.4.0

```xml
<p onmouseover=-a() ></p>
<p onmouseover=(a()) ></p>
<p onmouseover=;a() ></p>
<p onmouseover=!a() ></p>
```

• Other types of events can also be used for the attack (e.g. "onClick").

• Rules 71, 27, 2 and 65 are related to this insufficient pattern match.
Bonus:
Fingerprinting WAFs
Generating Program Fingerprints

\[ P_1 \quad P_2 \quad \ldots \quad P_N \]
Generating Program Fingerprints

P_1, P_2, ..., P_N

Which program is running in the Black-box?

P_T
Generating Program Fingerprints

\[ P_1 \quad P_2 \quad \ldots \quad P_N \]

\[ P_T \]
Generating Program Fingerprints

\[ \text{SFADiff} \]

\[
\begin{array}{ccc}
P_1 & P_2 & \ldots & P_N \\
\end{array}
\]
Generating Program Fingerprints

\[ P_1 \rightarrow P_2 \rightarrow \ldots \rightarrow P_N \rightarrow SFADiff \rightarrow P_T \]
Generating Program Fingerprints

SFADiff

Input causing difference in P_1, P_2

P_T
Generating Program Fingerprints

P_1 -> SFADiff -> P_i

Input causing difference in P_1, P_2

SFADiff

P_2

...  

P_N

P_T
Generating Program Fingerprints

\[
P_T \leftarrow SFADiff \leftarrow P_1 \rightarrow \ldots \rightarrow P_N \rightarrow P_i
\]
Generating Program Fingerprints

\[ P_1 \rightarrow SFADiff \rightarrow P_i \rightarrow SFADiff \rightarrow P_T \]
Generating Program Fingerprints

\[
\begin{align*}
\text{SFADiff} & \quad \text{P}_1 \quad \text{P}_2 \quad \ldots \quad \text{P}_N \\
\end{align*}
\]

\[
\begin{align*}
\text{SFADiff} & \quad \text{P}_i \\
\end{align*}
\]

\[
\begin{align*}
\text{P}_T
\end{align*}
\]
Generating Program Fingerprints

- \( P_1 \)
- \( P_2 \)
- \( ... \)
- \( P_N \)

SFADiff

Input causing difference

\( P_T \)
Generating Program Fingerprints

P_1 \rightarrow \text{SFADiff} \rightarrow P_i \rightarrow P_T

P_2 \rightarrow \text{SFADiff} \rightarrow P_i \rightarrow P_T

\ldots

P_N \rightarrow \text{SFADiff} \rightarrow P_i \rightarrow P_T

Input causing difference
Generating Program Fingerprints

![Diagram of program fingerprints generation using SFADiff]
Generating Program Fingerprints

\[
P_1 \quad P_2 \quad \ldots \quad P_N
\]

SFADiff

\[
P_i \quad P_j
\]
Generating Program Fingerprints

\[ \text{SFADiff} \]

- \( P_1 \)
- \( P_2 \)
- \( \ldots \)
- \( P_N \)

\[ \text{SFADiff} \]

- \( P_i \)
- \( P_j \)

\[ P_T \]
Generating Program Fingerprints

\[ \text{Input causing difference} \]
Generating Program Fingerprints

\[ P_1, P_2, \ldots, P_N \]

\[ SFADiff \]

\[ P_i \]

\[ SFADiff \]

\[ P_j \]

\[ SFADiff \]

\[ P_T \]

Input causing difference

\[ P_T \]
Generating Program Fingerprints

\[ \text{SFADiff} \]

\[ P_1 \rightarrow \text{SFADiff} \rightarrow \text{SFADiff} \rightarrow P_T \]

\[ P_2 \rightarrow \text{SFADiff} \rightarrow \text{SFADiff} \rightarrow P_T \]

\[ \ldots \]

\[ P_N \rightarrow \text{SFADiff} \rightarrow \text{SFADiff} \rightarrow P_T \]
LightBulb
Modular Design

• **Core Modules:**
  • Use automata models and operations
  • Extend the SFA learning algorithm

• **Built-in Query Handlers:**
  • Perform membership queries

• **Modules (and Built-in Modules):**
  • Use the Built-in Query Handlers
  • Extend the Core Modules: GOFA, SFADiff

• **Library:**
  • Set of grammars, filters, fingerprints trees and configurations
Core Modules

- Extend SFA Learning algorithm:
  - Accept the Alphabet, a Seed and/or a Tests file and a Query handler.
  - Initialise learning and manage results and models

- **The Alphabet**: Set of characters to be used

- **The Seed File**: Knowledge of what the examined inputs should look like

- **The Tests File**: Knowledge of specialised attacks

- **The Query Handler/Function**: Knowledge of how to perform queries for selected inputs
Core Modules

- **GOFA:**
  - Grammar Oriented Filter Auditing.

- **SFADiff:**
  - A black-box differential testing framework based on Symbolic Finite Automata (SFA) learning.

Simple Structure: *Class with five (5) basic functions: setup(), learn(), query(), getresults(), stats()*
Built-in Query Handlers

- **HTTP Request Handler:**
  - Perform queries on WAF filters and Sanitizers

- **SQL Query Handler:**
  - Perform queries on MySQL Parser

- **Browser Parser Handler:**
  - Perform queries on Browser JavaScript Parsers

- **Browser Filter Handler:**
  - Perform queries on Browser Anti-XSS Filters
HTTP Request Handler

- Targets WAF Filter
- Requires URL, HTTP Request Type, Parameter and Block or Bypass Signature
MySQL Query Handler

- Targets MySQL Database Parser
- Requires Database Credentials
- Requires Prefix Query: e.g, “SELECT a FROM a WHERE a=**”
Browser Parser Handler

- Targets the Browser HTML and JavaScript Parsing Engine
- Requires web sockets port, web browser port, host and trigger delay
- Inputs must trigger function a() (e.g., <script>a();</script>)
Browser Filter Handler

- Targets the Browser Anti-XSS Filter, HTML and JavaScript Parsing Engine
Using GOFA module and HTTP Handler
Using GOFA module and HTTP Handler

```plaintext
use HTTPHandler as my_query_handler
define URL http://83.212.105.5/PHPIDS07/
define BLOCK impact
back
```
Using GOFA module and HTTP Handler

use HTTPHandler as my_query_handler
define URL http://83.212.105.5/PHPIDS07/
define BLOCK impact back

Query Handler was created. We now can perform membership requests.
Using GOFA module and HTTP Handler

use HTTPHandler as my_query_handler
define URL http://83.212.105.5/PHPID507/
define BLOCK impact back

use GOFA as my_gofa
define TESTS_FILE {library}/regex/PHPID507/12.y
define HANDLER my_query_handler back

Query Handler was created. We now can perform membership requests.
Using GOFA module and HTTP Handler

**HTTP Handler**

```plaintext
use HTTPHandler as my_query_handler
define URL http://83.212.105.5/PHPIDS07/
define BLOCK impact
back
```

**Algorithm was selected and populated.**
Know we can learn application states.

**Query Handler was created.**
We now can perform membership requests.

**GOFA**

```plaintext
use GOFA as my_gofa
define TESTS_FILE {library}/regex/PHPIDS070/12.y
define HANDLER my_query_handler
back
```
Using GOFA module and HTTP Handler

**Query Handler was created.**
We now can perform membership requests.

**Algorithm was selected and populated.**
Know we can learn application states.
Built-in Modules

• **WAF Fingerprints Tree Generator:**
  • Automatically generates a fingerprints tree for a set of WAFs

• **WAF Distinguisher:**
  • Identifies a WAF using a set of fingerprints trees

• **Model Operations:**
  • Perform automata operations on stored models, input filters and grammars

• **Browser and WAF Differential Testing:**
  • Queries both Browser and WAF using a predefined set of strings
Built-in Rulesets Library

• **Regular Expressions**
  • Set of WAF filters, and attack models in the form of regular expressions

• **Grammars:**
  • Set of grammars that can be used for GOFA algorithm.

• **Fingerprints Trees:**
  • Set of fingerprints trees for a predefined number of WAFs.

• **Configurations:**
  • Sample configurations for WAF distinguish tree generation
Grub LightBulb:
https://github.com/lightbulb-framework/
Future Work

• Currently building many optimizations.
  - Learning will be much faster in the next months.
  - Cross checking models is also getting better.

• Incorporate fuzzers to improve models.

• New ideas?
Conclusions

- Current state of WAFs is still (very) ugly.
  - Many low hanging fruits.

- Our vision is to enforce a standard for such products.
  - WAFs must effectively defend against inferred language specifications.
  - Learning can run continuously with the assistance of fuzzers.

- We have a similar line of work on sanitizers.
Another Brick off The Wall: Deconstructing Web Application Firewalls Using Automata Learning

George Argyros, Ioannis Stais

Joint Work with:
Suman Jana, Angelos D. Keromytis, Aggelos Kiayias