

Automatically Test (and Sometimes Bypass) Web Application Firewalls

Luca Demetrio, Andrea Valenza



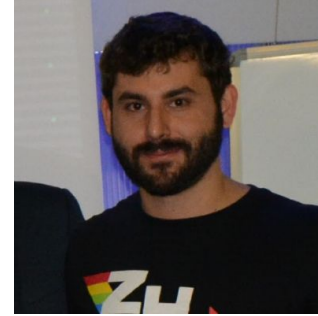
OWASP Italy March Meetup

About us



Luca Demetrio

Assistant Professor @ UniGe
Collaborator @ Pluribus One



Andrea Valenza

Security Engineer @ Prima Assicurazioni
PhD @ UniGe



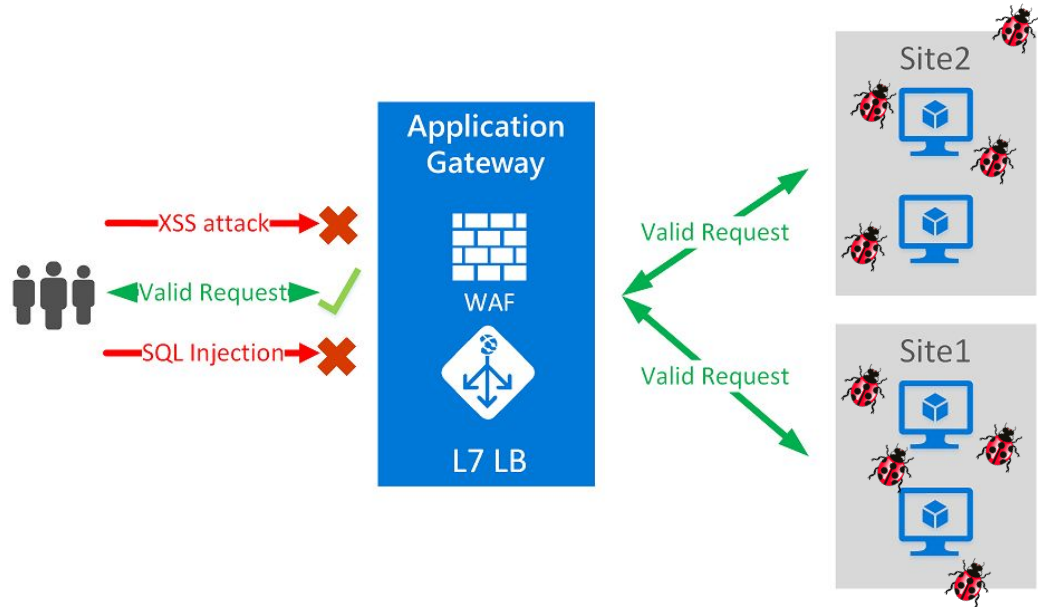
Research included in this presentation is part of the **TESTABLE EU Project** on testing technologies

Web Application Firewalls

Deployed “in front” of web applications to protect them from attacks

WAFs are a quick and easy solution, but they *DO NOT* remove vulnerabilities, just hide them under the rug

Very useful to “patch” applications, also block some “unexpected” attacks, but far from perfect



Web Application Firewalls

What they *CAN* do

Block Common Attacks

- Cross-Site Scripting (XSS)
- SQL Injection (SQLi)
- Remote File Inclusion (RFI)
- Command Injection
- Directory Traversal

What they *CAN'T* do

Address All Vulnerabilities

- Most Business Logic vulnerabilities
- Broken Access Control
- (some) Zero-Day Exploits

- Understand the context of your application

Example: SQL Injections

admin



```
SELECT * FROM users WHERE 'admin' AND password='...'
```

Example: SQL Injections

admin'--

✗ `SELECT * FROM users WHERE 'admin'-- ' AND password='...'`

How can I block this?

- Detect (and block) incoming payloads via “signatures”
 - Most commonly, regular expressions
- ModSecurity + CoreRuleSet

`admin' --`

Signature-based WAF

- Detect (and block) incoming payloads via “signatures”
 - Most commonly, regular expressions
- ModSecurity + CoreRuleSet

admin'--

```
SecRule ... "@rx /\*!?\| \*/|[';]--|--(?:[\s\v]|[\^\\-]*?-)|[\^&\-]#.*?[\s\v]|;?\x00" ...
```


Shortcomings of signature-based WAFs

Rules are created to match **specific payloads**, they can't foresee all their possible variants!

`admin'--` → `admin'` OR `'1'='1`

How do you bypass signature-based WAFs?

- Find alternatives to the payload you are currently
 - Possibly maintaining the “meaning” of the attack

admin'-- → admin' OR '1'='1

```
SecRule ... "@rx /\*!?\| \*/|[';]--|--(?:[\s\v]|[\^\\-]*?-)|[\^&\\-]#.*?[\s\v]|;?\x00" ...
```

How do you bypass signature-based WAFs?

- Find alternatives to the payload you are currently
 - Possibly maintaining the “meaning” of the attack
- Trial-and-error process (*LOTS* of errors, especially if you don't see the rules)
 - If the alternative is too common, another rule will match it

admin' -- → admin' OR '1'='1

```
SecRule ... "@rx
```

```
(?i)\b(?:or\b(?:[\s\v]?(?:[0-9]{1,10}|\\"'\"'^={1,10}|\\"'\"'))[\s\v]?[<->]+|[\s\v]+(?:[0-9]{1,10}|\\"'\"'^={1,10}')(?:[\s\v]*?[<->])?)|xor\b[\s\v]+(?:[0-9]{1,10}|\\"'\"'^={1,10}')(?:[\s\v]*?[<->])?)|'[\s\v]+x?or[\s\v]+.{1,20}[\s\v]+<->'" ...
```

Testing Problems

Testing with the same old payloads

Developers *only* use well-known attacks from ZAP and SQLmap, but attackers are not restricted to them!

Manual testing is hard

Crafting payloads manually is effective, but it does not scale since it requires plenty of human time and resources!

(Let's face it: if you had this time, you could use it to actually fix the vulnerability instead 😏)

WAF Problems

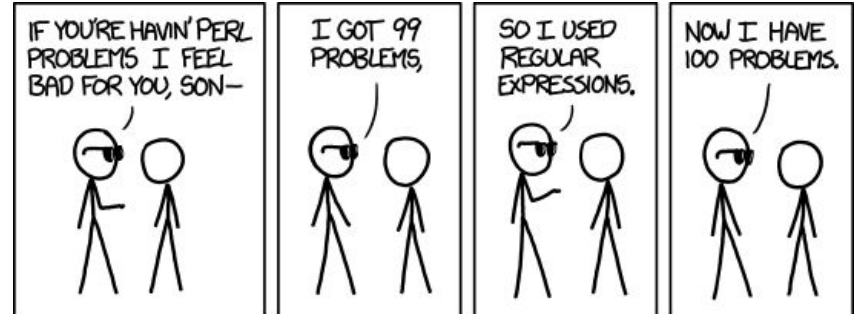
Testing with the same old payloads

It's impossible to find *ALL* attacks to create generic rules, so we often end up optimizing for known patterns

“One-size-fits-all” approach for shared rulesets (like CRS)

Bypasses still exist

Rules should block attacks, but let legitimate traffic through



“Next Generation WAF” == ML-based WAF

Learning threats from data

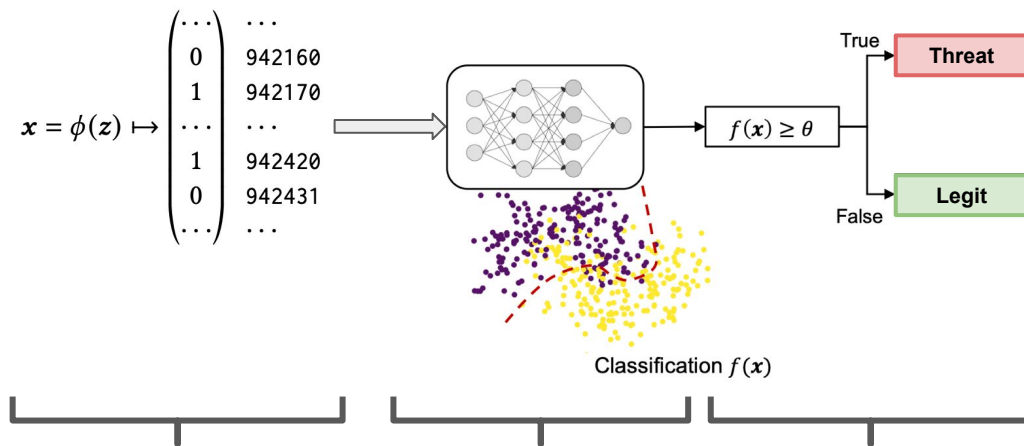
Directly train a model to understand what is malicious and what is legitimate

Generalizing across variants

No ad-hoc rules, the model generalizes among variations of the same payload



How ML works



Extract measures from data known as "features", to create a training set

Compute parameters of the model function f on training data that correctly generalize on future data

In production, if the output score is higher than a threshold, then the input is a threat



Are ML-based WAFs really effective?



Promising results of ML-based WAFs

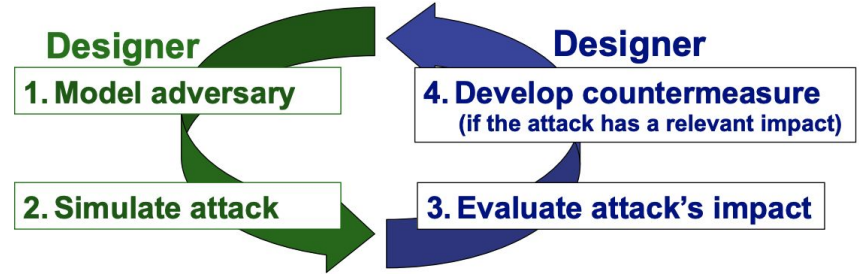
		A	R	P
WAF-Brain	RNN	98.27%	96.73	99.8%
Token-based	Naive Bayes	50.16%	98.71%	50.08%
	Random forest	98.33%	98.33%	100%
	Linear SVM	98.75%	98.76%	100%
	Gaussian SVM	97.82%	97.82%	100%
SQLiGoT	Dir. Prop.	90.61%	97.30%	85.82%
	Undir. Prop.	96.38%	97.31%	95.54%
	Dir. Unprop.	90.52%	97.12%	85.80%
	Undir. Unprop.	96.25%	97.05%	95.53%

Testing ML – Adversarial Machine Learning

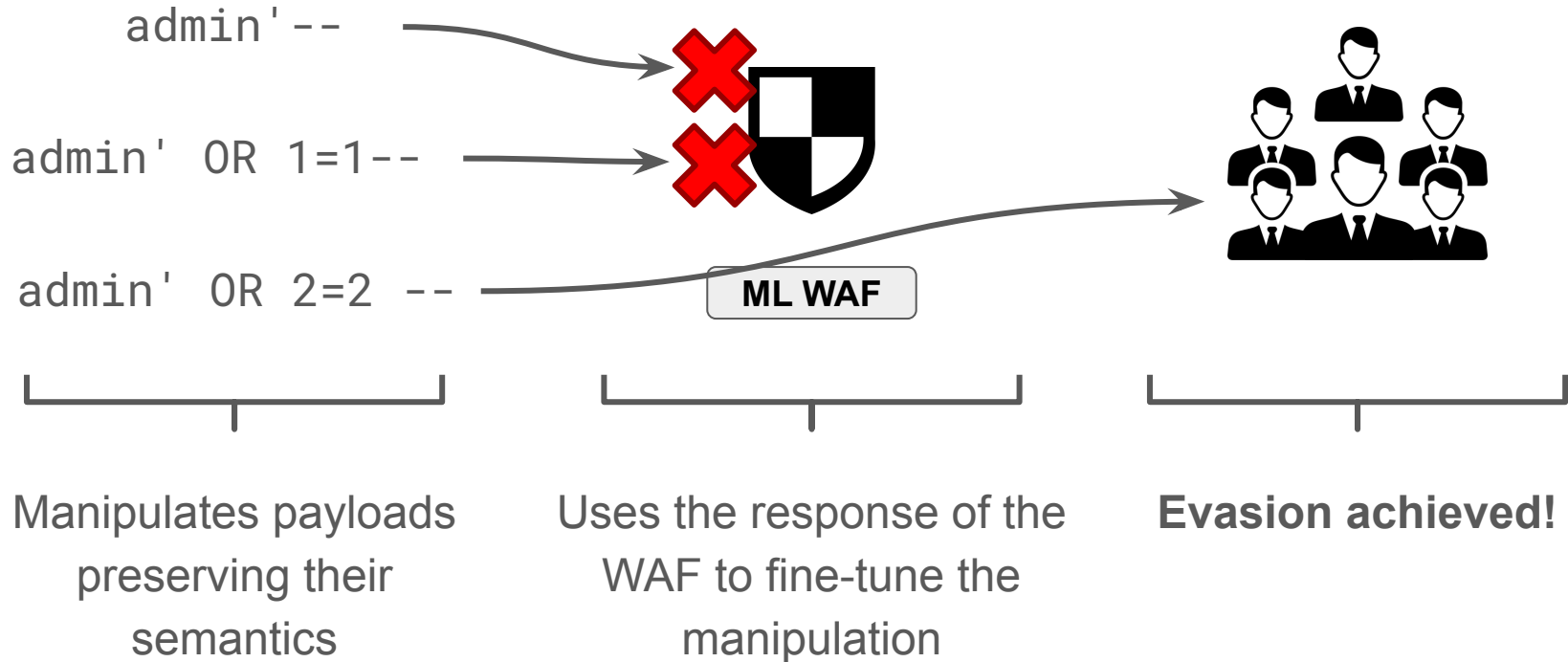
Considering an attacker that actively wants to dismantle your model, by proactively mimicking its behavior

If WAFs block 99% of payloads, the attacker can **systematically** generate that 1% that is missed

Payloads should be computed and refined, but without the scaling issues




WAF-A-MoLE: evading ML-based WAF



Components

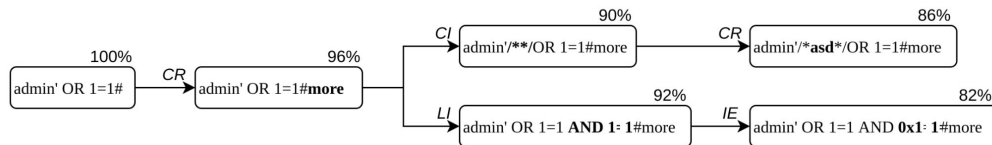
SQLi Manipulations

Semantically-equivalent transformations that change the representation of the payload

admin' --

admin' OR 2=2 --

Optimization Algorithms

Guide the testing process by prioritizing manipulations that lower the confidence score of the target model the most



SQLi Manipulations

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

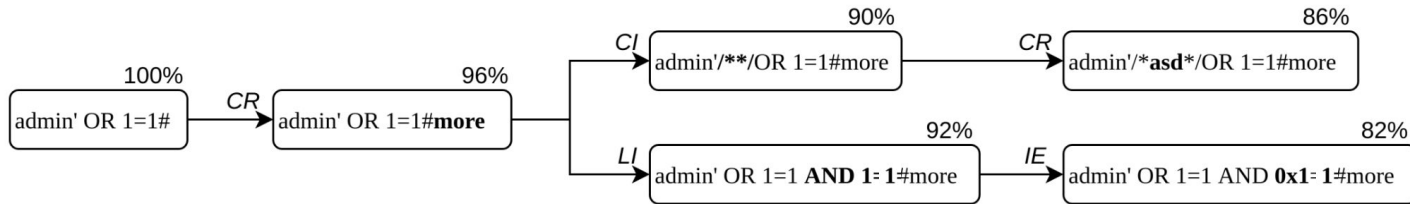
Manipulations can be combined in any order, since
they all maintain the original semantics of the
injection

Optimization Algorithms

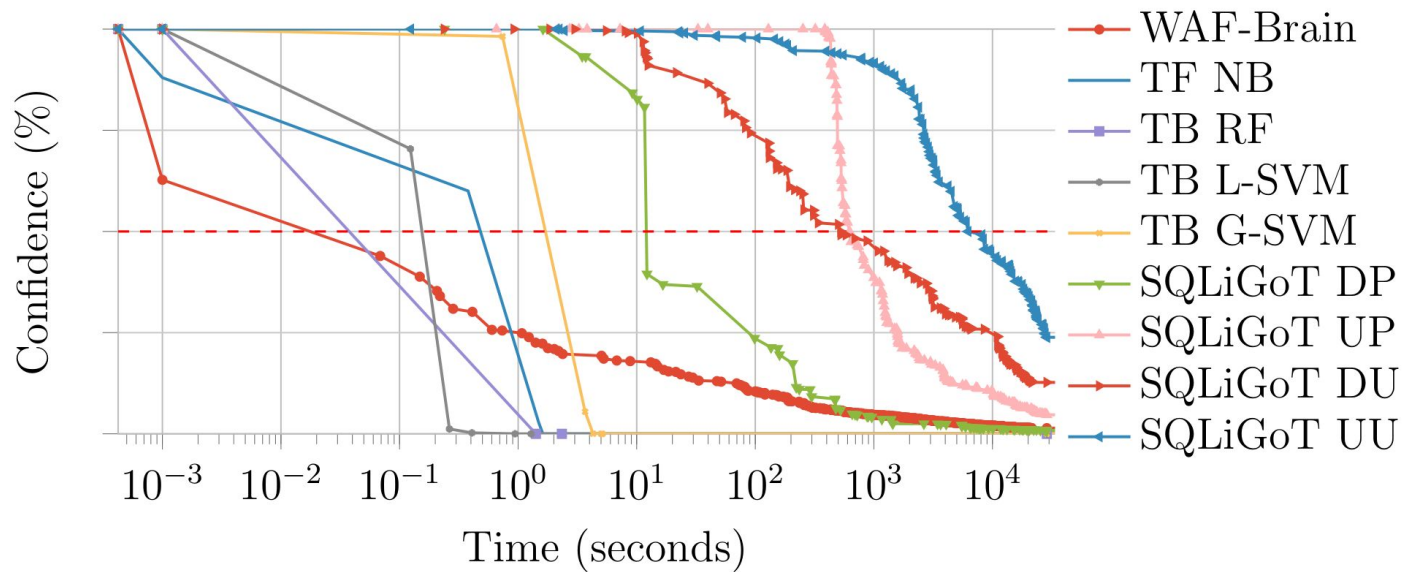
Guide the application of manipulations to reduce the score attributed by the model

Example: save the history of previously-applied manipulations, and prioritize the one whose score is the lowest possible

Note: some manipulations could lead to “dead ends”. This is why payloads that have higher scores have a lower priority, but are not discarded



Example results



In a few instants, WAF-A-MoLE can automatically find successfully-bypassing payloads against different ML-based WAFs!

So... should I stick with ModSecurity?



Manipulations interfere with signatures

Rules are stronger, but still bypassable

We highlighted that they can be manually bypassed,
this process can be automated as well!

Manipulations can remove the pattern detected by rules

Adversarial manipulations also apply to signature-based

BUT!

WAF-A-MoLE needs the score of the model to use as a “compass”,
it cannot be applied “as-is” to ModSecurity

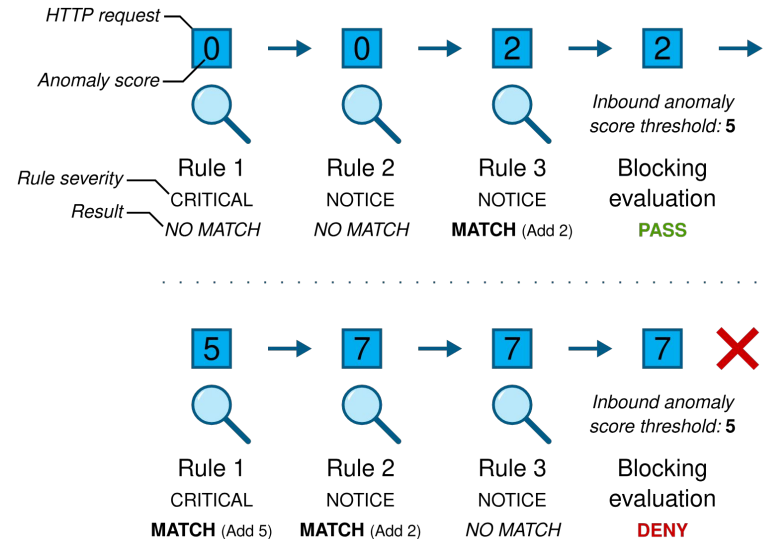
A closer look to the internals of ModSecurity

Expectation: pass/block binary decision

Reality: signatures are associated with a severity that heuristically quantifies their danger

ModSecurity computes an “Anomaly Score” as the sum of severities of all matched rules

This score can be used by WAF-A-MoLE!

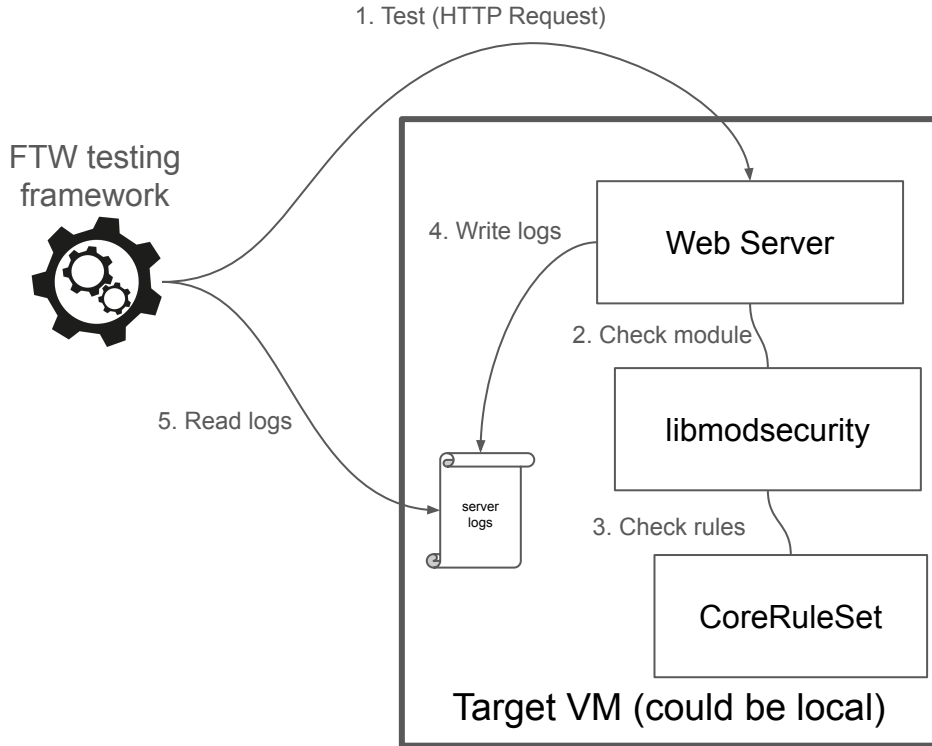


Problem: Testing Rulesets requires a whole environment

Testing the ruleset requires a full-fledged environment

- Web Server (Apache/nginx)
- ModSecurity
- Configured CoreRuleSet
- [A mock web application]

Workaround: a Python wrapper for libmodsecurity – Basically a CLI access to the ruleset!



pymodsecurity

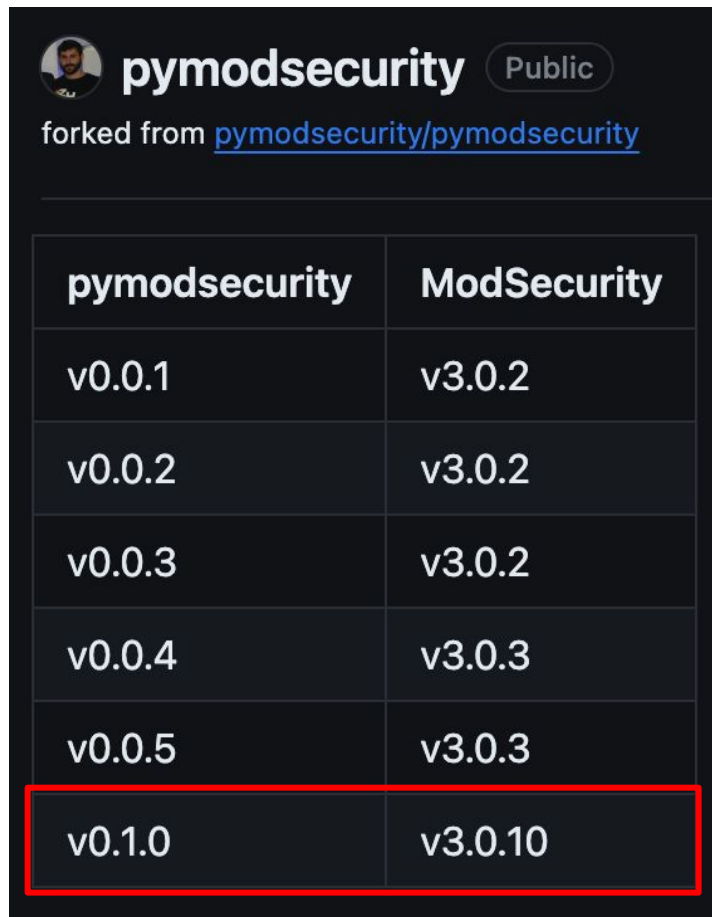
Python wrapper of ModSecurity, originally developed to support up to 3.0.3 version, abandoned few years ago

We forked it, and make it available also for ModSecurity latest!


<https://github.com/AvalZ/pymodsecurity>

Now, ModSecurity can be reached easily by Python tools

<https://github.com/AvalZ/modsecurity-cli>



The image shows a screenshot of a GitHub repository page for 'pymodsecurity'. The repository is public and was forked from 'pymodsecurity/pymodsecurity'. Below the repository information, there is a table comparing versions of 'pymodsecurity' and 'ModSecurity'. The table has two columns: 'pymodsecurity' and 'ModSecurity'. The rows show the following version mappings: v0.0.1 to v3.0.2, v0.0.2 to v3.0.2, v0.0.3 to v3.0.2, v0.0.4 to v3.0.3, v0.0.5 to v3.0.3, and v0.1.0 to v3.0.10. The last row, containing v0.1.0 and v3.0.10, is highlighted with a red border.

 pymodsecurity Public	
forked from pymodsecurity/pymodsecurity	
pymodsecurity	ModSecurity
v0.0.1	v3.0.2
v0.0.2	v3.0.2
v0.0.3	v3.0.2
v0.0.4	v3.0.3
v0.0.5	v3.0.3
v0.1.0	v3.0.10

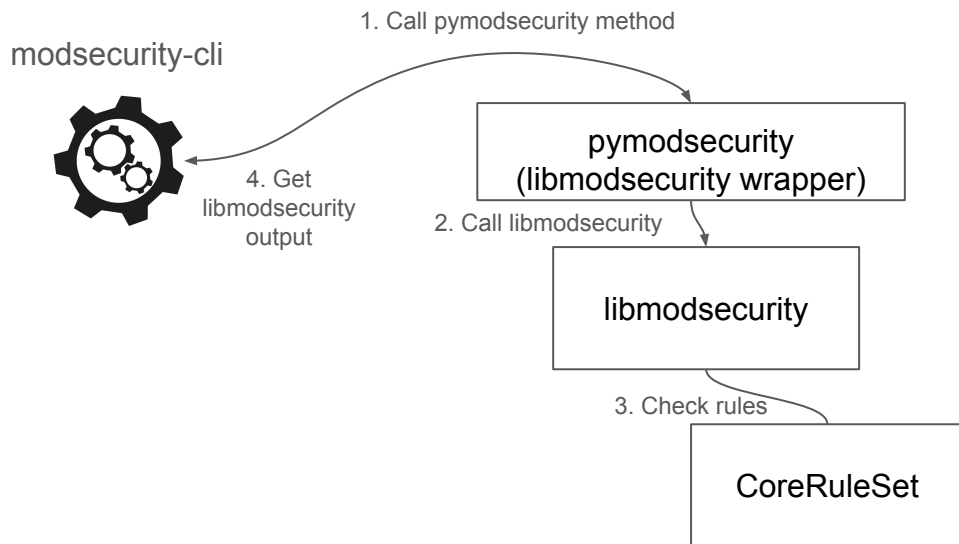
pymodsecurity

Access libmodsecurity directly
(via bindings)

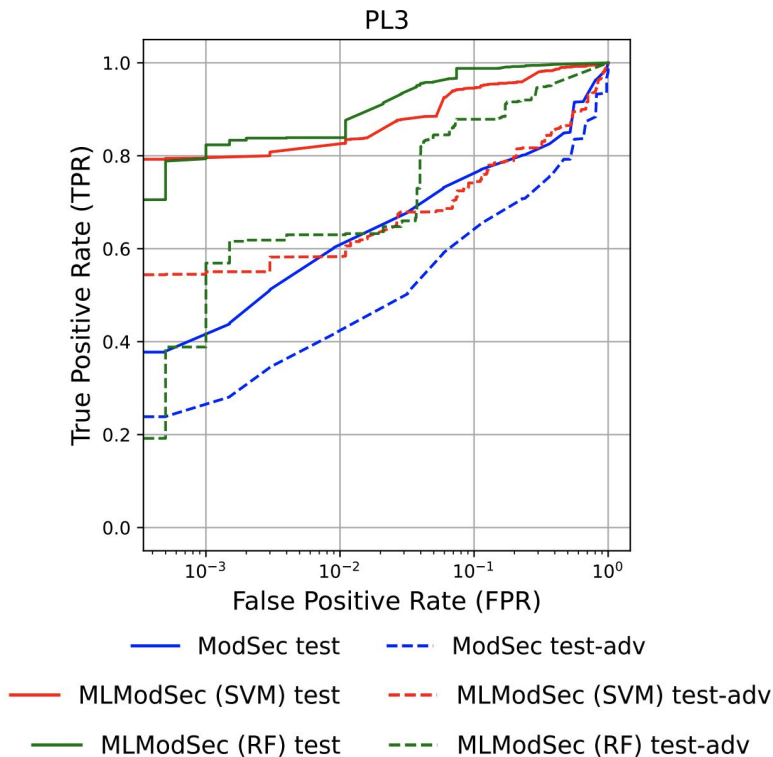
No web server setup, log parsing, etc.

Completely removes the overhead of
having to run HTTP requests!

Now it's WAF-A-MoLE-time :)



Severity scores are not “calibrated”



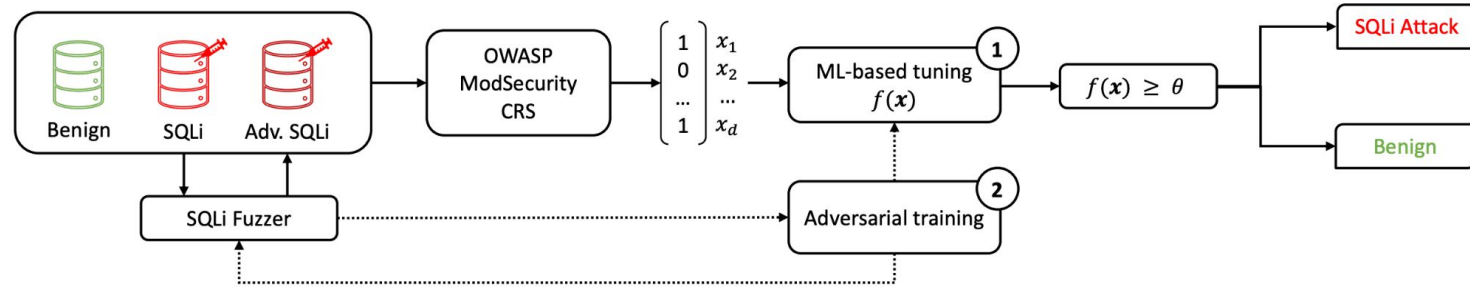
ModSecurity accuracy is not incredible

Higher Paranoia Levels don't solve the problem, since it leads to blocking more benign traffic → More False Positives

Adversarial attacks are a challenge for ModSecurity (CRS)

Results suggest that ModSecurity **is even worse** than ML models!

AdvModSec – Hardening against Adversarial SQLi

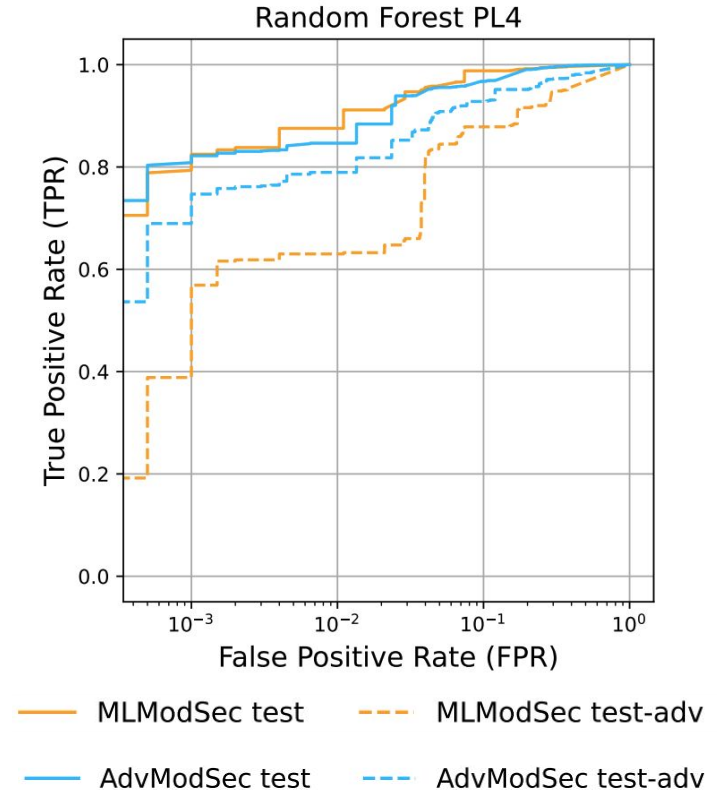


Train ML model on CRS signatures, and use as training data both regular and adversarial SQLi

AdvModSec – Hardening against Adversarial SQLi

Still vulnerable, but more robust and accurate than ModSecurity alone

Easy to implement and deploy, since it leverages the best of both worlds (specificity of rules, and generalization of ML models)



Limitations

- Mutations are only for SQL Injections
 - Currently looking into XSS, drop a mail if you have ideas and want to join us :D
- We're implementing better optimization algorithms
 - We only use guided fuzzing, but better strategies can be applied
- The prototype is not great yet – Contribute!
 - It's likely full of bugs! We caught some, but most of them are probably still in the code
 - More ML- and signature-based WAF can be included directly in the codebase

Work in progress – Contributions are welcome!

- Improve WAF-A-MoLE mutations
 - Wanna add more domains, like XSS or Path Traversal?
Drop a mail or just fork the repo and add your PR :D
- Systematic payload generation
 - Generate payloads that match rules by “reversing” rules themselves
 - Some super-early work using Z3 at <https://github.com/avalz/regrets>
- Super recent work: **Autospear**
 - Builds on WAF-A-MoLE using a different optimization algorithm, could be included inside the prototype as well (feel free to open a PR and include it!)

Thank you! Questions?



[WAF-A-MoLE](#)



[WAF-A-MoLE \(paper\)](#)



[Adversarial ModSecurity](#)



Luca Demetrio



luca.demetrio@unige.it



zangobot



@zangobot

Andrea Valenza



avalenza89@gmail.com



AvalZ



avalz_



<https://avalz.it>