Toward LLM-Driven Rule Generation for Enforcement Systems: An Exploratory Study on WAF

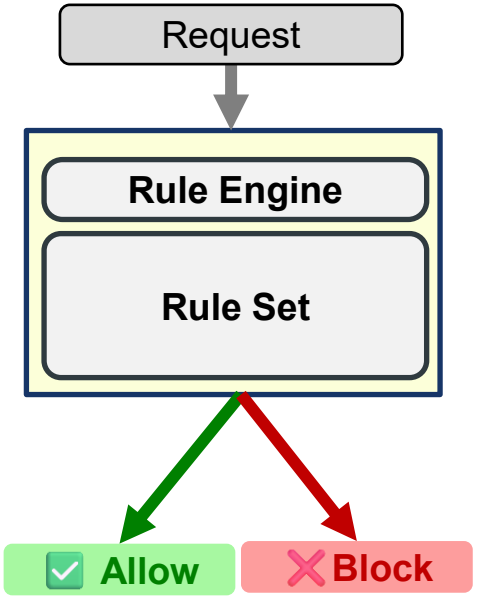
Quanzhi Fu, Dan Williams
Virginia Tech

Rule-based Enforcement System

System that **evaluates operations** against a set of **predefined rules** and permits or denies the operation.

- SELinux
- Network Firewalls
- Web Application Firewalls (WAF)

Architecture: Rule Set + Rule Engine



Human Response Slow to New Attacks: WAF example

Attacks keep evolving, existing rule sets fail to handle new attacks

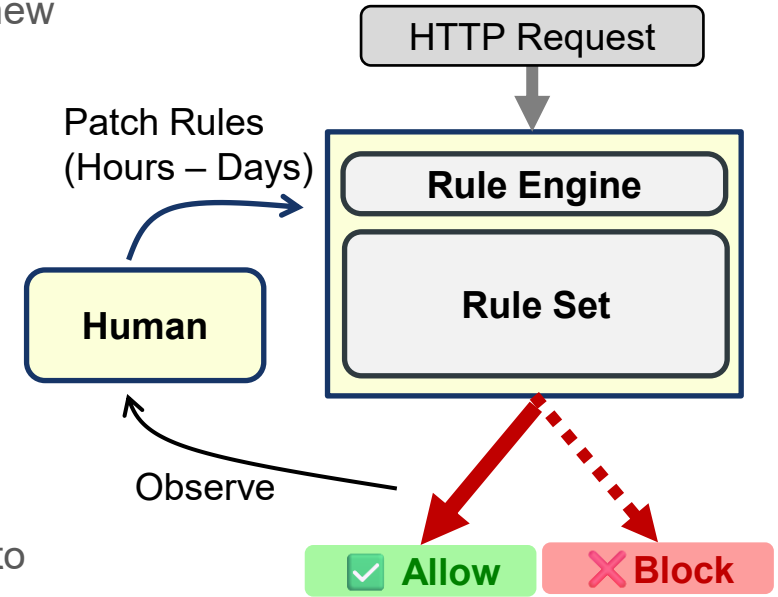
Iterative process:

New attacks get through -> Human patch rule set

Human response is slow:

2021 Log4Shell Incident:

hours to release new patch, weeks for end users to adopt



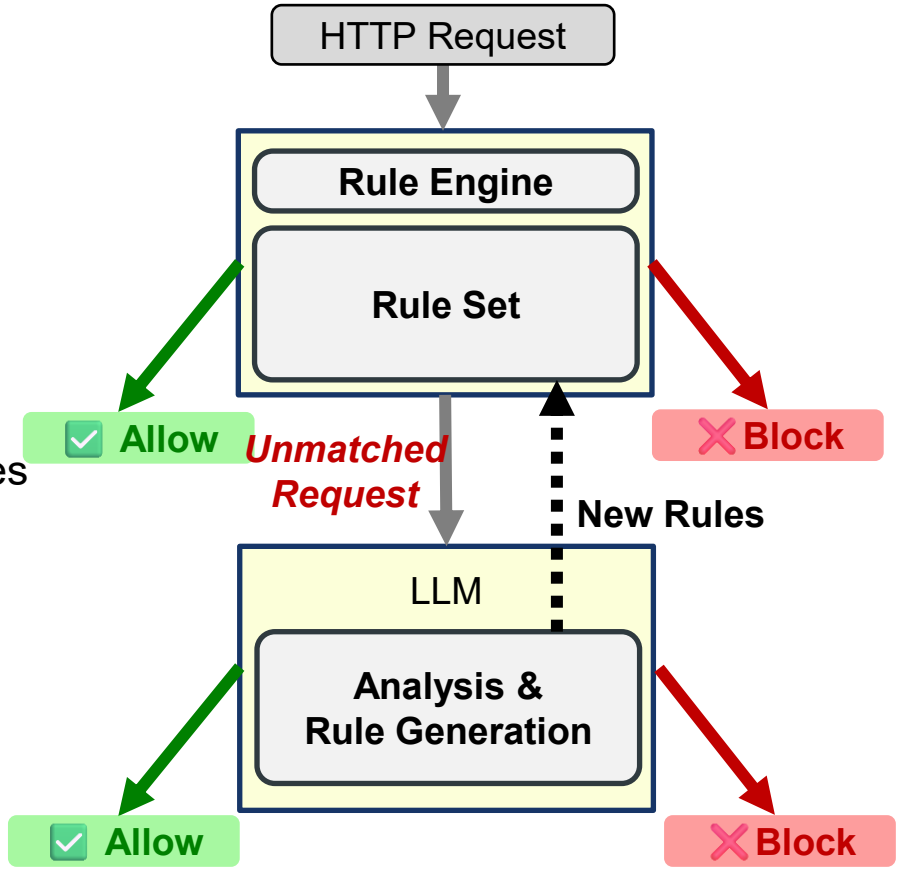
Overview of Takeaways

A hybrid architecture：VibeWAF

Combine slow LLM with fast rule engine

Takeaways:

- ✅ Comparable perf with human crafted rules
- ✅ Rule set converges well
- ⚠️ Existing rules need reevaluate regularly
- ⚠️ ~10% requests still hit LLM



Roadmap

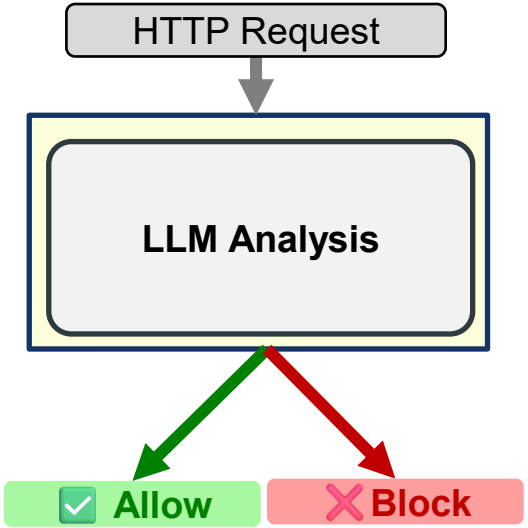
- The Powerful but Slow LLM
- Hybrid architecture: VibeWAF prototype
- Evaluation questions:
  - Is this a feasible architecture
  - The dynamics of the system
- Takeaways
- Future work

LLM is Powerful, but Slow

Sampled 200 requests from SR-BH 2020 dataset

Prompting LLM to classify traffic as normal/malicious

Comparing LLM with open sourced rule set OWASP CRS



Similar F1: 0.80 vs 0.79
High Latency: 6000x slower

Impractical to directly use LLM with WAF

Method	P	R	F1	Latency
OWASP CRS	0.83	0.78	0.80	<1ms
Claude Sonnet 4.5	0.75	0.85	0.79	~6s

Table 3: Classification performance comparison.

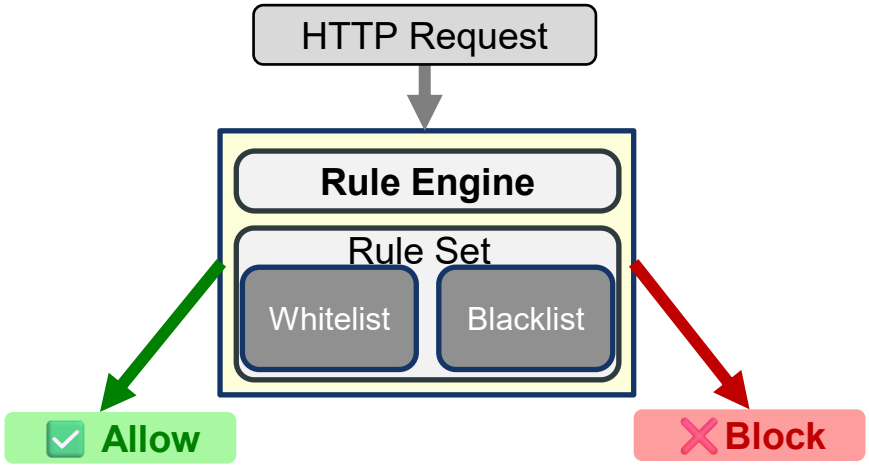
Hybrid Architecture: VibeWAF

VibeWAF prototype: combining slow LLM with fast rule matching

Phase I rule matching

Blacklist: match malicious pattern

Whitelist: match normal pattern



Hybrid Architecture: VibeWAF

VibeWAF prototype: combining slow LLM with fast rule matching

Phase I Rule Matching

Blacklist: match malicious pattern

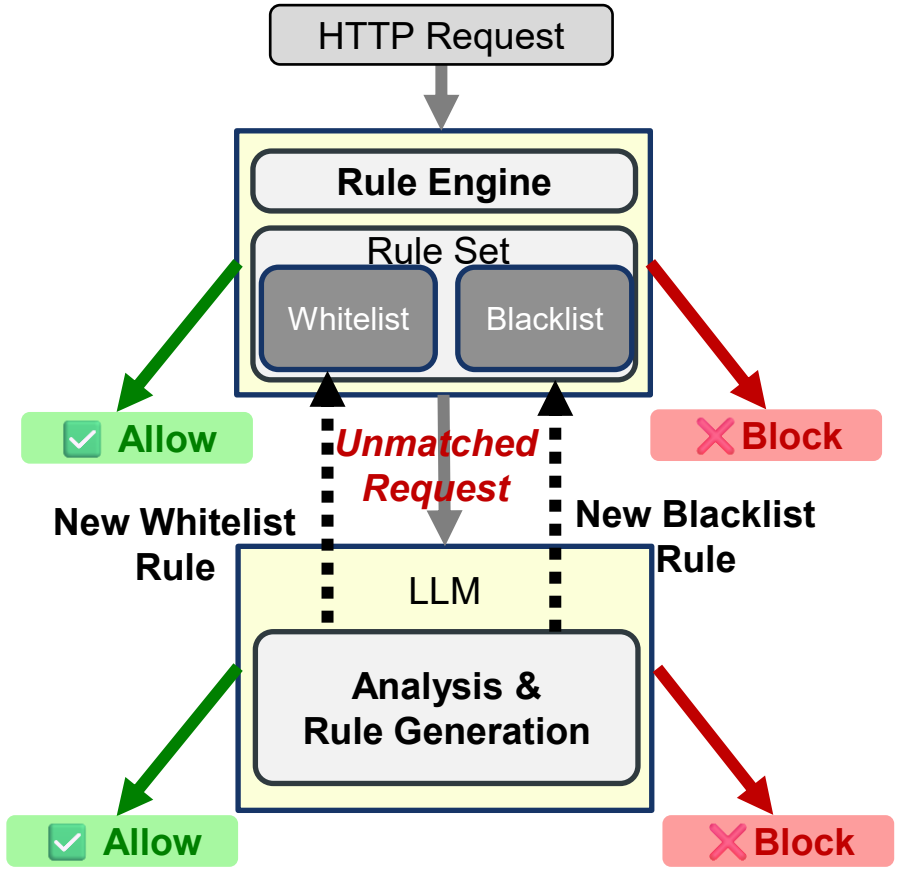
Whitelist: match normal pattern

Phase II LLM analysis & Generate Rule:

LLM analyze unmatched request individually

Rule set is append-only

Progressive Offloading: With the rule set accumulating, LLM only needs to handle “new attack”



Evaluation Questions

Does this architecture work?

- What is the overall detection capacity of the system?
- Will the rule set converge?
- Can the system adapt to novel attack?

What is the dynamics of the architecture?

- How rule strategy affect detection?
- How does the rule set scale?

Does the Architecture Work?

1. What is the overall detection capacity of VibeWAF

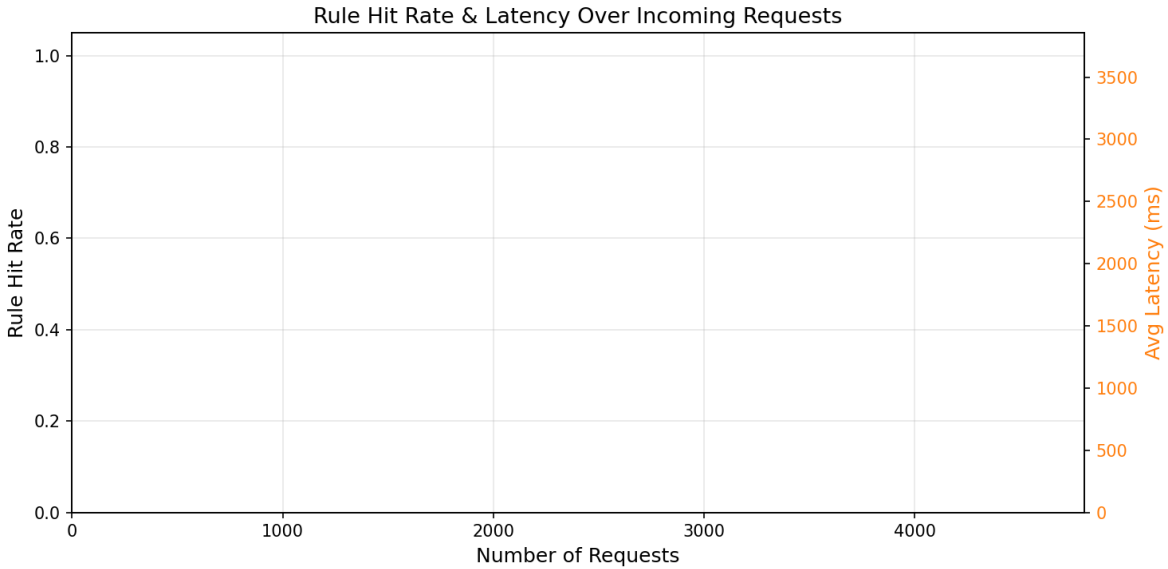
   We evaluate the system on subset of SR-BH 2020

Method	Acc.	Prec.	Rec.	F1
CRS	0.866	0.917	0.793	0.850
LLM-only	0.828	0.807	0.843	0.825
VibeWAF	0.806	0.736	0.928	0.821

Takeaway: Comparable performance to CRS; Low FN rate

Does the Architecture Work?

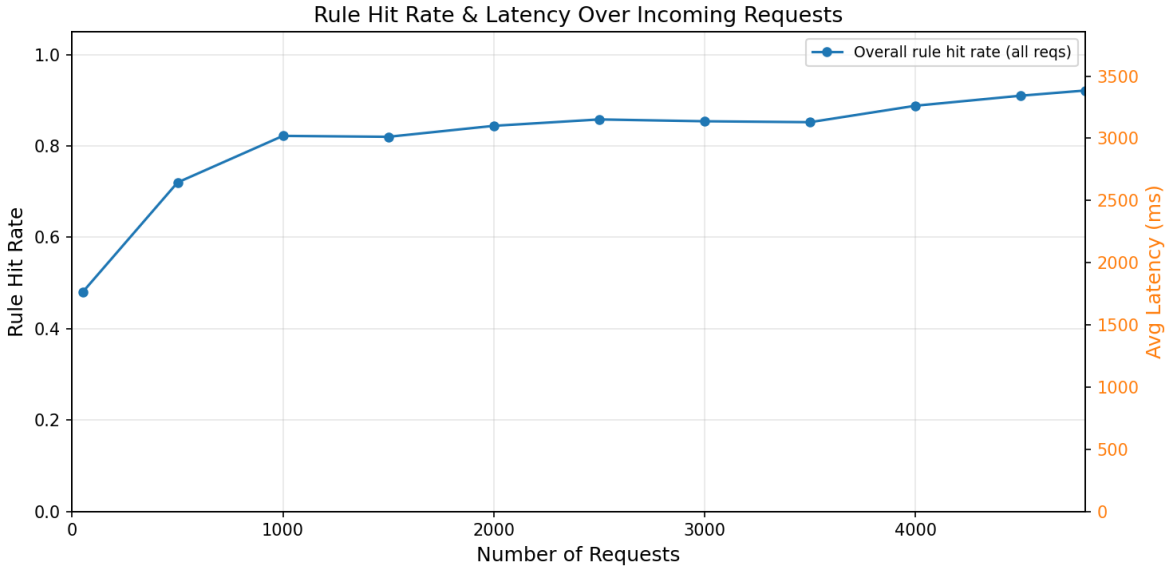
2. Will the rule set converges?



Does the Architecture Work?

2. Will the rule set converges?

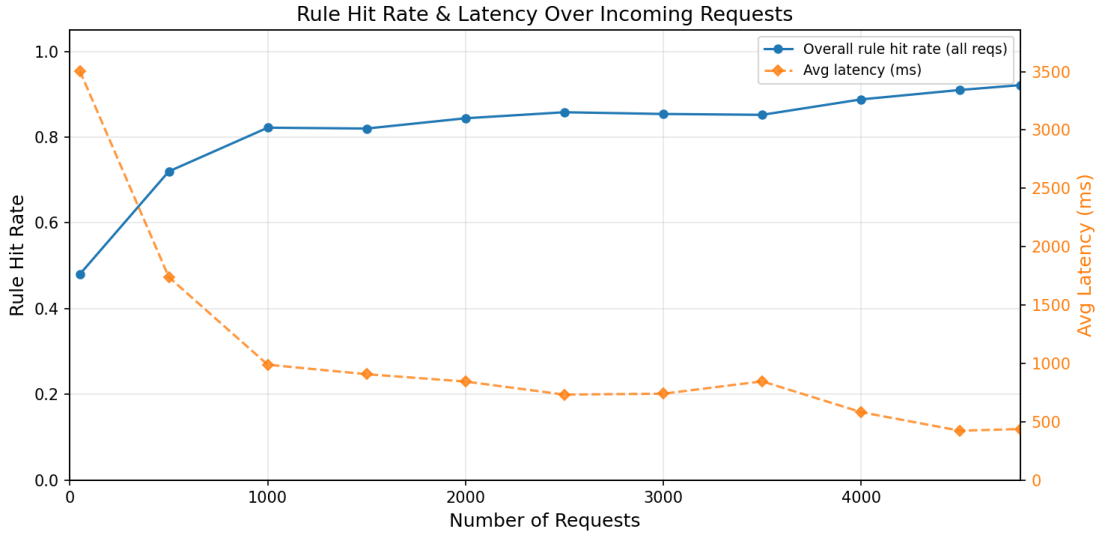
Rule hit rate 0% - ~90%



Does the Architecture Work?

2. Will the rule set converges?

Rule hit rate 0% - ~90%, average latency from 6000ms ~400ms

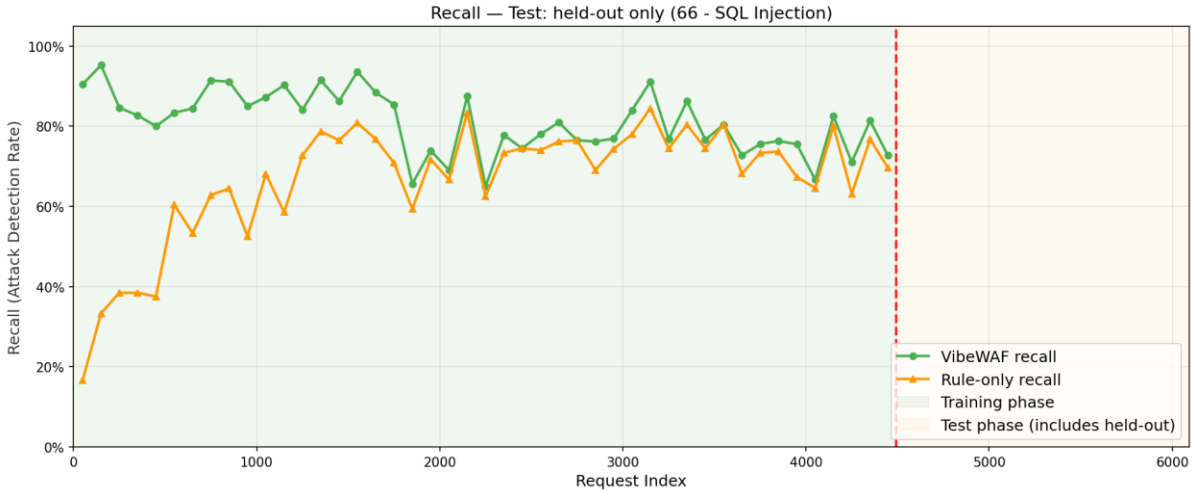


Takeaway: Rule hit rate ↑, latency ↓ Progressive Offloading Work

Does the architecture work?

3. Can VibeWAF adapt to new attacks?

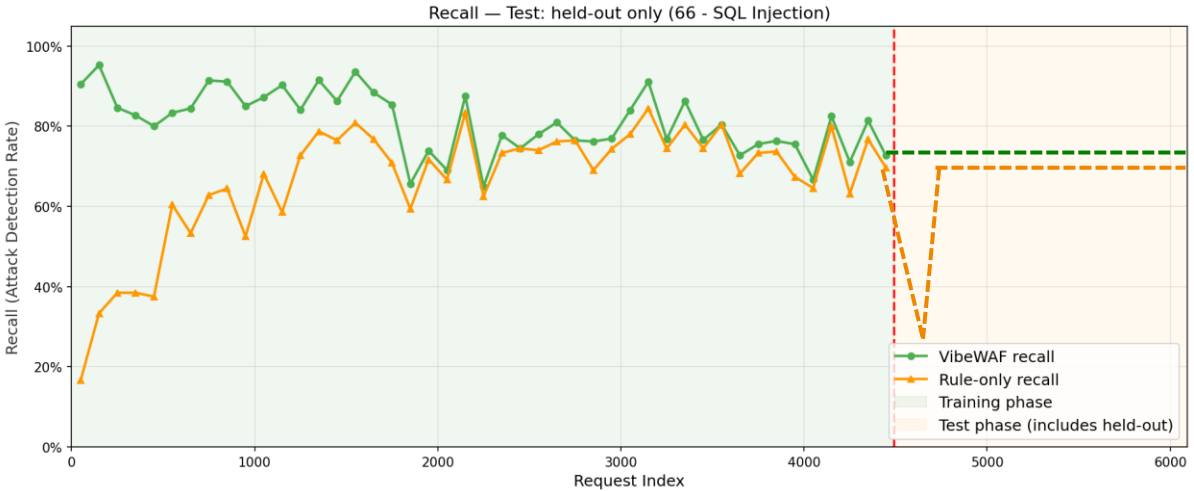
hold-out experiment: working on regular traffic(training) + add a new attack later (testing) **The training phase: rule only**



Does the architecture work?

3. Can VibeWAF adapt to new attacks?

hold-out experiment: working on regular traffic(training) + add a new attack later (testing) **The training phase: rule only**

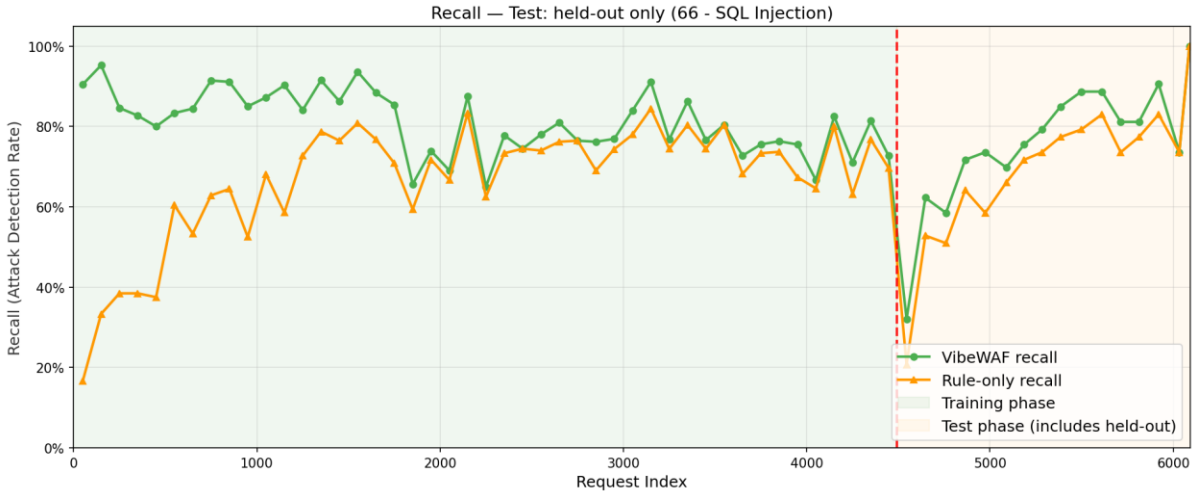


Does the architecture work?

3. Can VibeWAF adapt to new attacks?

held-out experiment: working on regular traffic + add a new attack later

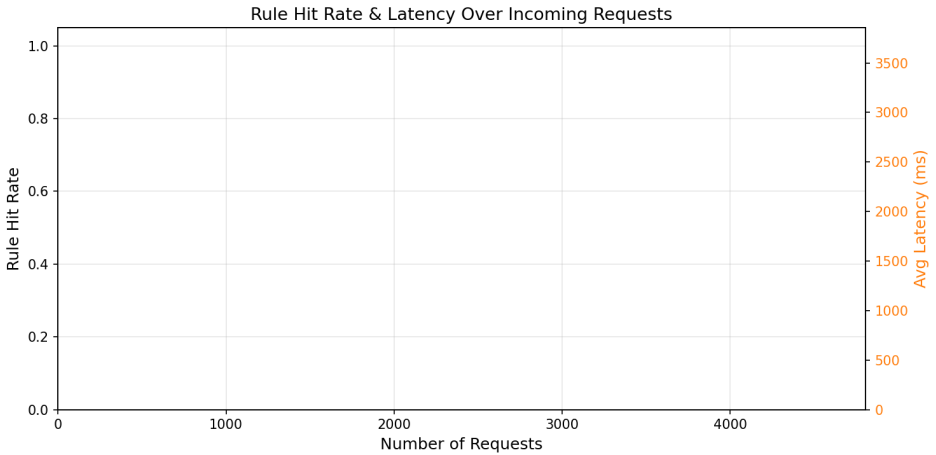
The testing phase… adapt but not as ideal as expected



Takeaway: whitelist rule silently absorbs malicious traffic

Dynamics of the Architecture

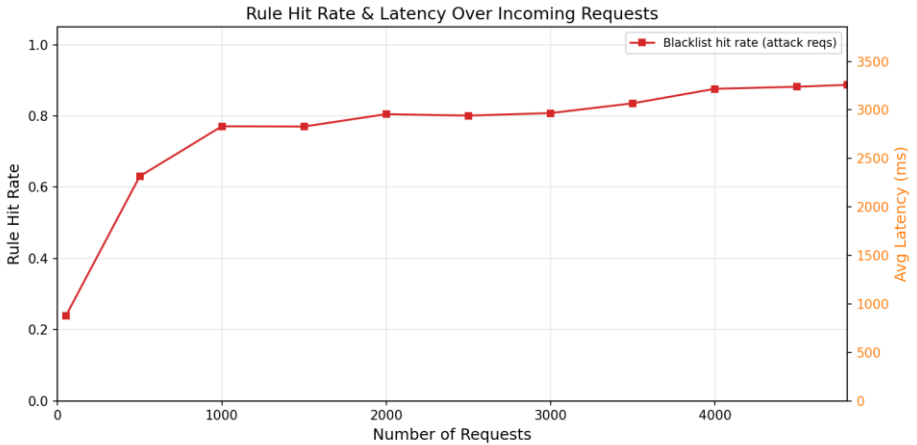
1. What is the difference between blacklist vs. whitelist rule



Dynamics of the Architecture

1. What is the difference between blacklist vs. whitelist rule

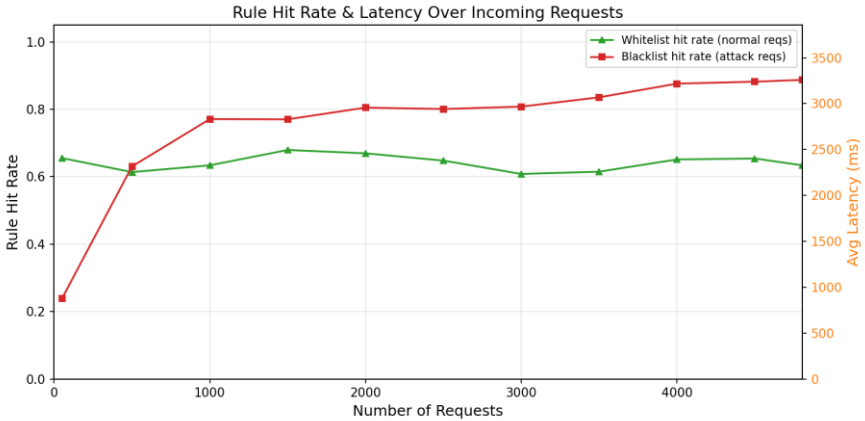
blacklist rule: converge smoothly



Dynamics of the Architecture

1. What is the difference between blacklist vs. whitelist rule

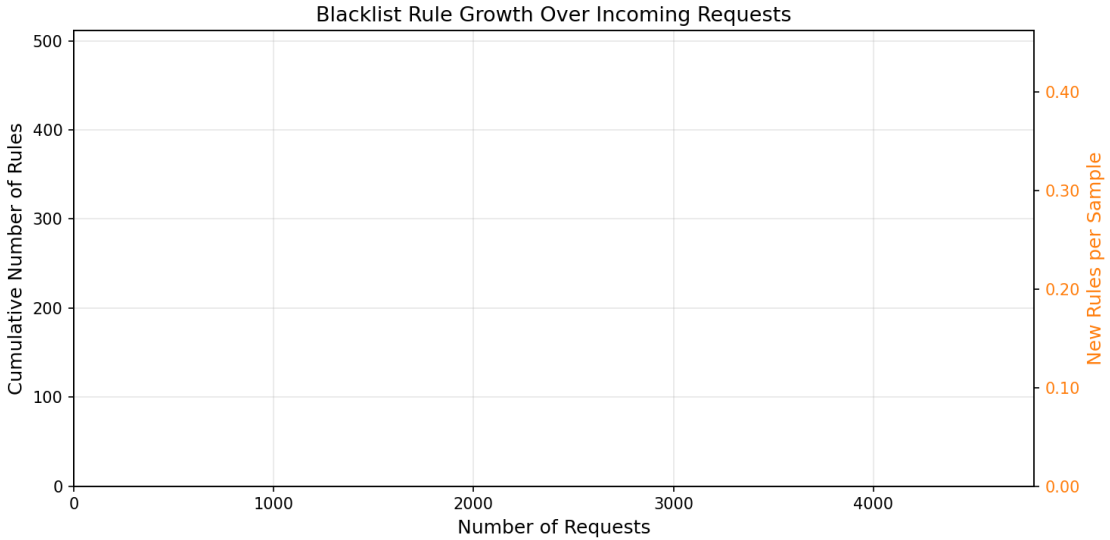
blacklist rule: converge smoothly & **whitelist rule**: not converge



Takeaway: Whitelist rules do not converge

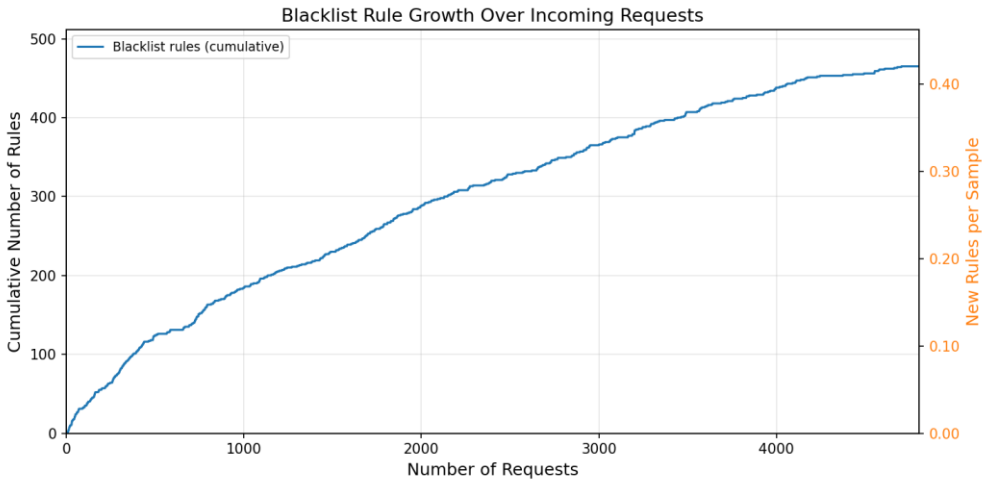
Dynamics of the Architecture

2. How does the rule set scale?



Dynamics of the Architecture

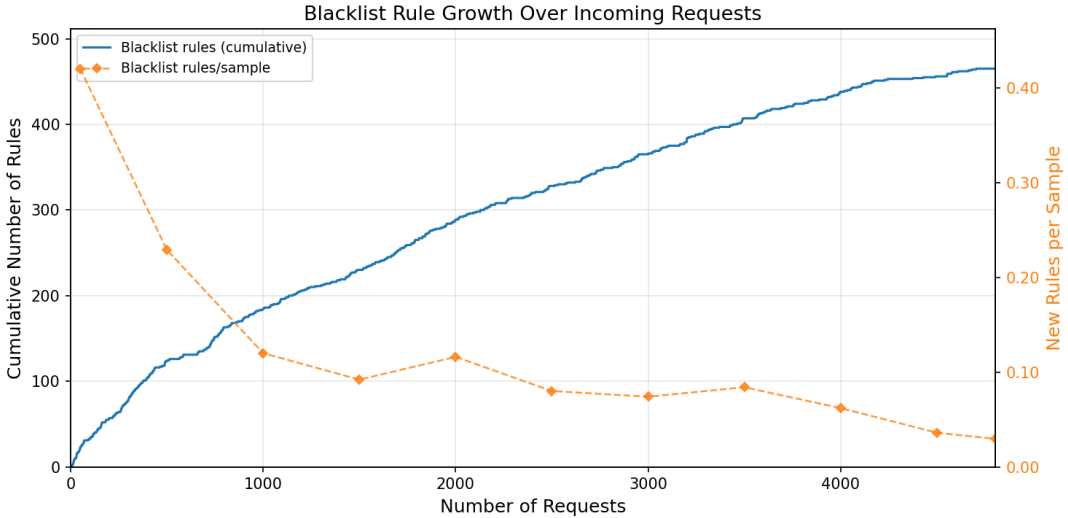
2. How does the rule set scale?



Dynamics of the Architecture

2. 2. How does the rule set scale?

In append-only setting, # of rules grows unbounded



Takeaway: the # of rules grows unbounded

Evaluation takeaway

Hybrid architecture is a feasible direction:

- ✅ Similar performance with CRS; Suitable scenario: low FN tolerance
- ✅ Rule set converges, latency drops significantly
- ✅ able to respond to new attacks

Challenges for more :

- ⚠️ Whitelist rule is problematic: fail to converge & silently admit new traffics
- ⚠️ # of rules grows unboundedly with append-only

Future Work

Identified Challenges	Design Indication	Potential Solutions
Whitelist rule non-converge	LLM should not stay online (All normal traffic will hit LLM)	async LLM + filtering method to identify problematic traffic
Unbounded rule accumulation with append only Whitelist rule admit bad traffic	Ruleset should be managed in finer grained	Mechanisms for rule refinement (deduplication, expiration, validation) Mechanisms to generate better rules (different context, traffic)

Questions?

Hybrid architecture is a feasible direction:

- ✅ Similar performance with CRS; Suitable scenario: low FN tolerance
- ✅ Rule set converges, latency drops significantly
- ✅ able to respond to new attacks

Challenges for more :

- ⚠️ Whitelist rule is problematic: fail to converge & silently admit new traffics
- ⚠️ # of rules grows unboundedly with append-only