WAFSmith: An advanced WAF Rule Governance Agent

WAFSmith

An advanced WAF Rule Management Agent

“LLM-based Rule Management Framework to Create Rules that Simply Work”

Github: https://github.com/tankeehock/wafsmith

An open-source tool by Kee Hock Tan with guidance from Sean Yeoh

Leveraging LLM's abilities to mimic cognitive human agents, WAFSmith aims to reduce the friction of Web Application Firewall rule management from rule creation to deployment in minutes. It is designed as a highly disruptive tool to augment Blue Team operations in a rapidly evolving threat landscape. It was developed to enhance Blue Team's capabilities to respond to threats quickly and effectively, without compromising business operations. The solution is the first of its kind, especially in the open source landscape, a novel approach to solving a challenging WAF rule management problem.

In this whitepaper, we attempt to discuss in depth the key challenges of WAF Rule Management and a new approach towards improving Rule Management practices by empowering Blue Team with a novel, advanced Rule Management AI agent - WAFSmith.

Abstract

Rule management has been a challenging issue that enterprises face to maintain a highly effective WAF. Poor maintenance of WAF rulesets results in productivity loss and reduced WAF effectiveness which ultimately translates to an enlarged attack surface. WAFSmith was developed to reduce the friction of rule management by leveraging the capabilities of intelligent AI agents to perform tasks to reduce the cognitive burden placed upon human operators. It is a highly advanced LLM-based agent that can perform payload extraction, rule writing, testing, and deployment. Evaluation performed with proven rulesets such as ModSecurity’s CRS, shows significant security improvements introduced by WAFSmith, demonstrating real-world impact. The methodology adopted by WAFSmith proved to be robust and reliable which can be easily retrofitted to suit other rule-management use cases such as IDPS Rules. WAFSmith is designed to enhance Blue Team’s cyber defense investments through a high degree of intelligent operations in the domain of rule management. The by-products in the development of WAFSmith such as the Prompts and ModSecurity rules serve as valuable contributions to the Open-Source community.

Problem Statement

Background

Web Application Firewall (WAF) is one of the most critical security controls against web-related threats. It is often one of the first layers of defense against web-related attacks as part of a defense-in-depth strategy that enterprises employ. However, maintaining a highly effective WAF continues to be a prevailing challenge as WAF Rule Management continues to be a difficult problem to solve in enterprise settings. This can be attributed to: dynamically changing business needs, rapidly evolving attack payload landscape, and efficiency requirements necessitating pruning of stale rules.

In this paper, we define WAF Rule Management as active management of the WAF rulesets to achieve two core goals:

- Creating useful rules
- Simplifying ruleset maintenance and governance

Default Rulesets are not enough, finetuning is needed

Using the default WAF ruleset is not enough to mitigate the constantly evolving threat landscape. A report by AppTrana reveals that 59% of the malicious traffic blocked is attributed to custom rules. The observation indicates the need to continually review and create new WAF rules to meet changing business needs. Popular Open Source WAF applications such as ModSecurity by OWASP, released ModSecurity Rule which often served as a baseline for WAF application performance.

WAF rules have to play the "catch-up" game as the threat landscape continuously evolves. For instance, new vulnerability discoveries such as Log4J back in 2021, WAF operators scrambled to quickly deploy WAF rules to protect against potential exploitation of Log4J.

Writing Good WAF rule(s) is hard

WAF rule creation often relies on a good understanding of regular expression (depending on the WAF technology) and its associated rule execution engine. Even with the most experienced human operator, time is required to develop high-quality rules that target specific web vulnerability ranges vastly. The complexity is further exacerbated by WAF solutions adopting different flavors of regular expressions such as POSIX Basic Regular Expressions, POSIX Extended Regular Expressions (ERE), Perl-Compatible Regular Expressions (PCRE), and many other variants.

Bad Rules Break Business Operations

WAF rules can potentially disrupt business operations by accidentally blocking normal traffic (False Positives). WAF with high false positive rates are indicative of a poor rule design process which does not add value to the security posture and may potentially result in business disruption. Thus, most Blue Teams will adopt a more conservative approach towards WAF Rule Development and perform rigorous testing to ensure that the new rule(s) do not disrupt

business operations. This often results in delayed responses to threats and increases the Mean Time To Respond (MTTR).

Current Solutions

With the developments in Large Language Models, many leading WAF solution providers adopted LLMs as part of their product strategy to provide assisted WAF rule writing capabilities to assist human operators in navigating the challenges of rule management. The list below are popular WAF rule-writing agents or tools that are available on the market (non-exhaustive list) to assist in the Rule Management process:

- Cloudflare's AI Assistant for WAF Rule Builder
- Impart's Rule Architect
- Imperva's Cloud WAF AI Explainability

Identified Gaps in the Market

Limited Access to Advanced Rule Writing Agents in the Market

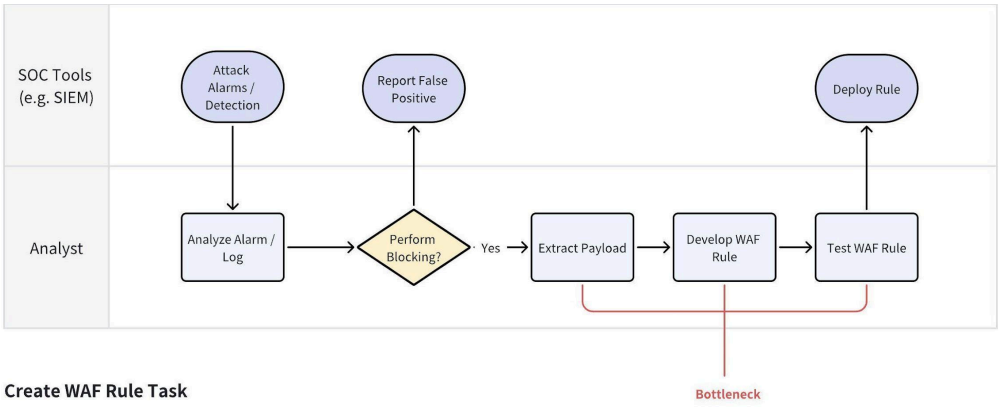
Access to such capabilities is often limited as these agents are designed specifically for a single product. WAF is a highly competitive and fragmented market, with known dominant players such as AWS, Cloudflare, Imperva, and many other lesser-known players serving different segments of the market. These players often offer a unique blend of solutions to meet the increasing demands of their customers. With the developments in Artificial Intelligence (AI), these players have invested in AI capabilities to improve their service offerings. This creates the potential for "vendor lock-in".

- However, the distribution of these capabilities is largely uneven. Not all dominant WAF players offer a flavor of AI in their service offerings. This is evident in the open source space, the market is largely under-served.
- The capabilities of such tools can be easily witnessed in generally more accessible AI-based service offerings such as ChatGPT. Refer to Appendix A: Case Study of Using OpenAI GPT4 for ModSecurity Rule Generation.

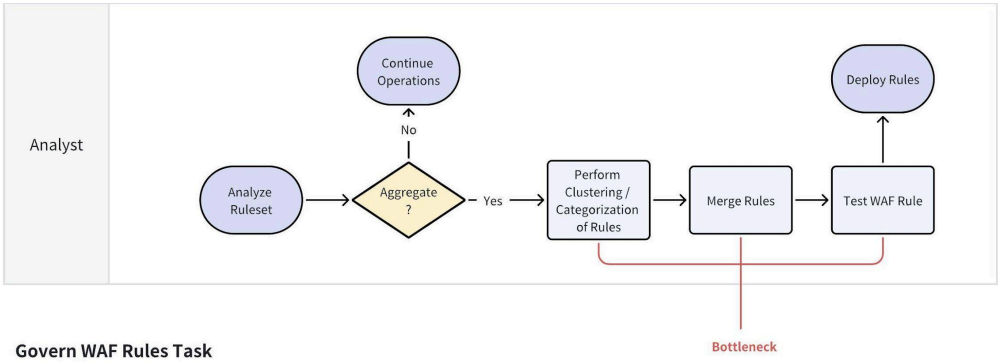
WAF Rule Management remains a highly manual and laborious process

Often, managing WAF rulesets requires a human-in-the-loop, which draws resources away from Blue Team's ability to react to ongoing threats. The two diagrams below are two common tasks relating to rule management. The workflows are highly abstracted and generic. However, they serve as a "good enough" representation of activities carried out by typical Blue Teams in most enterprises.

Activities that are identified to be bottlenecks (commonly identified as laborious and manual) in the workflow are highlighted in RED.



Create WAF Rule Task



Govern WAF Rules Task

Proposed Solution - WAFSmith

Develop a LLM based Rule Management Framework to reliably develop rules that are highly robust and vendor-agnostic

Definition(s):

- LLM: Large Language Model, such as OpenAI's GPT 4 series
- Rule Management Framework: A structured set of tasks to achieve Rule Management goals
- Reliably develop rules: Ability to create rules (WAF) that do not impact business operations
- Robust and vendor agnostic: The solution should not work only for a particular vendor and offer a high level of interoperability across products

Leveraging LLM's abilities to emulate cognitive human agents, WAFSmith is a highly disruptive tool capable of augmenting Blue Team operations in a rapidly evolving threat landscape. WAFSmith enhances Blue Team's capabilities to respond to threats in a fast and effective manner, without compromising business operations. The solution is the first of its kind, especially in the open source landscape, a novel approach to solving a challenging problem of WAF rule management.

It is developed as a standalone Command-Line-Interface (CLI) that is platform and product agnostic. It can be rapidly deployed in any environment and adapted to the Web Application Firewall (WAF) engine to provide reliable rule management capabilities. It allows Blue Teams to leverage advanced Artificial Intelligence (AI) agents to reliably and intelligently perform WAF-related activities, from payload extraction to WAF rule development, testing, and deployment through battle-tested workflows. It can be used in all kinds of WAF-related scenarios, such as using it as a rapid-response tool during security incidents to rapidly develop WAF rules or as part of routine WAF maintenance tasks to perform rule management (aggregation).

It is designed to ingest text content such as Web Server Logs, and extract potentially malicious payload which can be used to trigger rule management-related workflow(s). For instance, it will attempt to create a WAF rule to block these potentially malicious payloads. It enforces testing workflows to reliably assess the quality of the rules through simulated environments (docker containers) to ensure that the rules do not impact business operations.

WAFSmith leverages LLMs ability to reason, learn, create, and make-decisions to perform resource-intensive tasks to a similar or better output level than its human counterparts. WAFSmith reduces the friction of rule management by empowering Large Language Models (LLMs) to augment and orchestrate these activities. Allowing for enhanced security responses

with little to no human intervention, saving resource costs and allowing human operators to focus on higher-order tasks.

Design Motivation

The key motivation behind this solution is to demonstrate the capabilities of mature LLM offerings being able to value-add cybersecurity initiatives in any enterprise. The solution takes advantage of the advanced cognitive capabilities witnessed in today's LLM offerings and shows through empirical evidence how LLM-based agents can reliably perform tasks that are of importance.

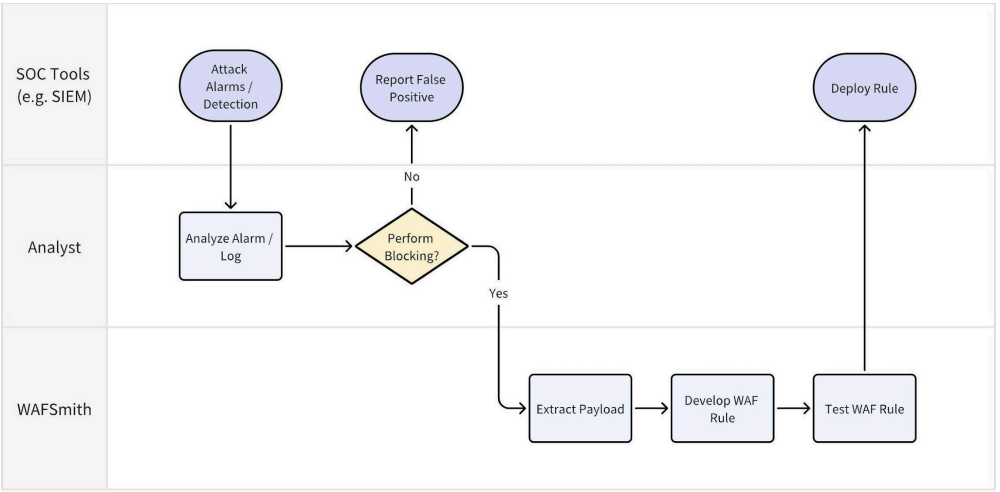
The solution aims to be a solution that can be adopted by enterprises to empower their cyber defense strategy and capabilities through low-cost and highly effective innovations as seen in today's market.

WAFSmith is currently developed using ModSecurity as the WAF engine.

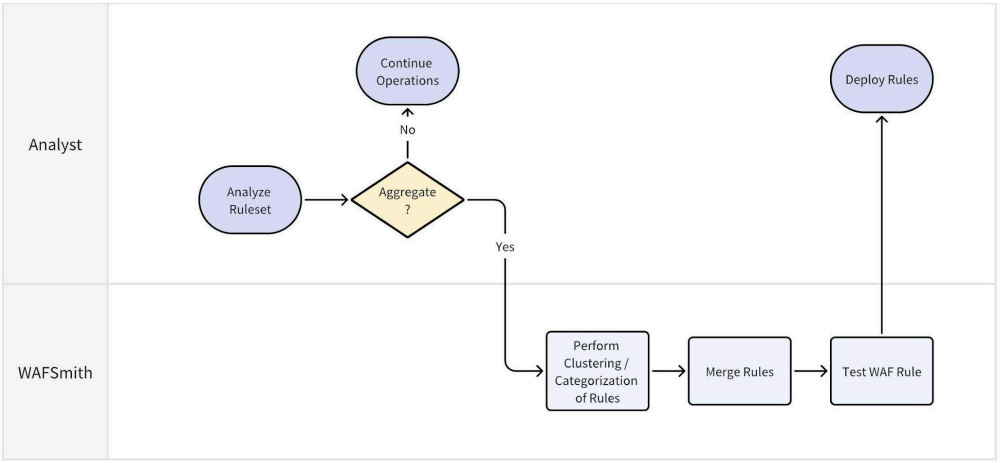
- CoreRuleSet (CRS) is available as a useful benchmark and base ruleset

Revised Workflows with WAFSmith

WAFSmith will attempt to automate the identified bottleneck activities as highlighted in the "Identified Gaps in the Market" section. The revised workflow is shown below.



Create WAF Rule Task



Govern WAF Rules Task

Key Feature(s)

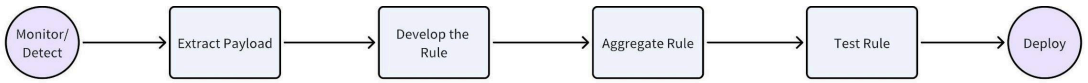
Ruleset Self-Management Capabilities

In this whitepaper, Self-Management capability is defined as the tool's ability to perform tasks that are crucial in maintaining an effective and easy-to-maintain ruleset. There are four capabilities identified that are critical for achieving Self-Management:

- **Extraction of Payload(s)**
    - Based on the data source (e.g. Log Entry), the tool provides the means to extract potentially malicious payload that it should block.
- **Creation of Rule(s)**
    - Based on the payload(s), the tool creates a relevant rule(s) to block these payload(s)
- **Aggregating Rule(s)**
    - If possible, merge rules to reduce the size of the ruleset, improving runtime efficiency and rule sprawl
- **Testing the Rule(s)**
    - Test the rule against the payload, along with simulated business traffic to ascertain the potential business impact

The orchestration of these tasks provides adopters the capability to perform self-governance of the ruleset.

In which provides the core functionality of each key phase of the Rule Creation Lifecycle as illustrated below:



Typical Rule Creation Workflow

In practice, rules are hardly decommissioned, unless they can be aggregated or covered by newer rules.

The tool provides the 4 capabilities in an integrated workflow to ease ruleset governance.

Functional Specification(s)

The framework is to be encapsulated as a NodeJS CLI tool that can be easily adopted and modified by end users.

Functional Requirements

- Generate ModSecurity Rule
- Deploy a simulated Web Application for testing using Docker Containers

Non-Functional Requirements

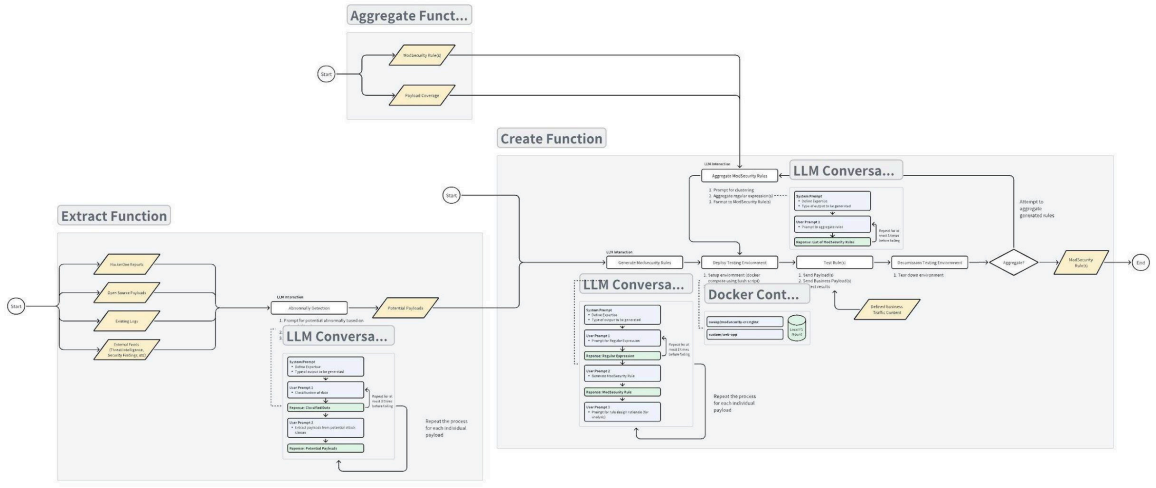
- Fail gracefully for docker-related deployment tests
- Ease of use in adopting the CLI tool

- Test the generated rule against business traffic and the payload

Technical Design

Proposed WAFSmith Workflow(s)

The diagram below provides a high-level blueprint of the inner workings of WAFSmith. An enlarged diagram can be found in the Appendix F: WAFSmith Workflow(s).



In the following subsections, the workflow's design specification will be described.

Extract Workflow

	Specification
Capability	The extract function in WAFSmith intelligently extracts payloads from text sources at scale using LLM.
Scenario	Blue Team performs a periodic review of web server logs to identify abnormalities. WAFSmith's extract function is used to process these logs at scale to produce a list of potentially malicious content / potential payloads.
Technical Inputs	Open Source Payloads Web Server Logs HackerOne Reports Threat Intelligence Feed

Technical Output	Payload Lists

Create Workflow

	Specification
Capability	The `create` function in WAFSmith creates an aggregated ruleset designed to block the maximum number of payloads (based on the given input).
Business Use Case	After the extraction of payloads, Blue Team proceeds to create WAF rules to block these payloads as part of ongoing cyber defense efforts.
Technical Inputs	Payloads
Technical Output	ModSecurity Rules

Aggregate Workflow

	Specification
Capability	The `aggregate` function in WAFSmith provides Blue Teamers the capability to intelligently merge rules
Business Goal	Blue Team periodically uses WAFSmith to perform rule aggregation to maintain a sizable ruleset.
Technical Inputs	ModSecurity Rules Payload Lists (Coverage)
Technical Output	ModSecurity Rules

Prompts

Please view the prompts in the GitHub

`evaluate` workflow does not require LLM.

Evaluation Metrics

To understand the performance of WAFSmith, we want to be able to answer two key questions:

- How well the rules are created

  With relevance to WAF, the quality of the rules created can be measured by its ability to act on the payload. A good rule that is created can catch the payload that is designed for. To measure how well rules are created, the use of the Evasion Rate is adopted.

  Evasion Rate

  Number of payloads that were not blocked by the ModSecurity Rules as a ratio against the total number of payloads sent

- How easy it is to govern the rules

  Ease of governance is a subjective measure. In this metric, we identified three areas of indications that tell us how easy it is to govern rules:

  1. Ease of Creating Rules
     - This can be indicated by the time taken to create rule(s)
  2. Ease of Rule Maintenance (Aggregation)
     - Aggregation of rules refers to consolidating and improving the ruleset to reduce the size of the ruleset to a minimal size without compromising the protection it can offer. This can be observed by the number of rules in the ruleset
  3. Payload Coverage
     - This can be observed through the amount of payloads that the ruleset can detect

  Thus, to simplify the measurement, we used 2 key statistical measures as primitive indicators (below) to measure the three goals above. This leads us to coin the term "Ease of Management".

  Ease of Management

  The score will be using a custom metric that is based on two statistical measures in this experiment:

  Ease of Management = Processing Rate * Block Rate

  - **Processing Rate**

    Total number of payloads that the party/tool is tasked to process over the total time taken in seconds. The design of this metric helps us to indicate how efficiently the tool/party is processing the task.

The higher the **Processing Rate**, the more likely the party/tool is effective in processing payloads.

Example - WAFSmith vs Operator

It takes 120 seconds for Operator 1 to develop modsecurity rules targeted at catching 120 payloads.

Processing Rate: 120 / 120 = 1

e.g. It takes 30 seconds for WAFSmith to develop modsecurity rules targeted at catching 120 payloads

Processing Rate: 120 / 30 = 4

Processing Rate indicates how effective the tool/party is in performing the task of rule generation along with aggregation for the given set of payloads. The better Processing Rate indicates the ease of governing these rules!

- **Block Rate**

The number of payloads blocked over the number of active rules

e.g. 10 rules blocked 100 different payloads for Operator 1

Block Rate: 100 / 10 = 10

e.g. 5 rules blocked 100 different payloads for WAFSmith

Block Rate: 100 / 5 = 20

Block rate simply tells us how effective the rules are in blocking the payloads.

Calculating the Ease of Management Score

Ease of Management (Operator 1): 1 * 10 = 10

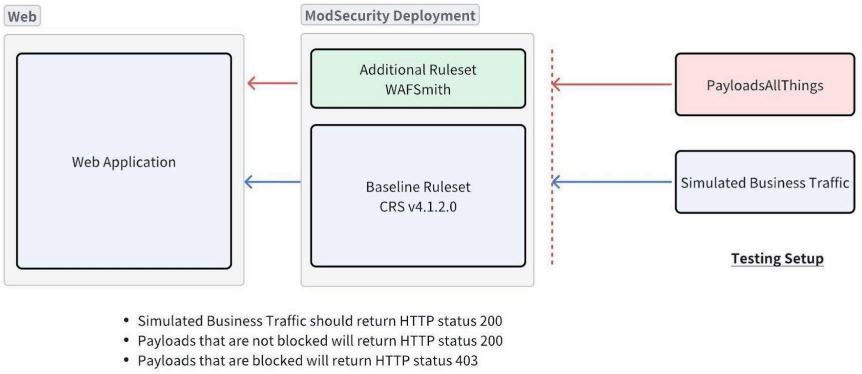
Ease of Management (WAFSmith): 4 * 20 = 80

Interpretation: The higher the Ease of Management score, the easier it is to govern the rules. In this example, using WAFSmith is much easier to govern by around 8 times compared to the human counterpart.

WAFSmith is focused on generating new WAF Rules and governing them. The focus of the evaluation is to compare **how much security has improved after using WAFSmith** from a set of baseline rules. In this setup, **CRS 4.12.0 ModSecurity rules** are used as **baseline** benchmarks in which WAFSmith is adopted to improve and govern the ruleset.

Benchmark / Evaluation Setup

Evasion Rate



Payloads from a well-known Open Source repository, PayloadsAllThings are used to measure the performance of ModSecurity rules.

1. Benchmarking is performed by measuring the evasion rate from the CRS ruleset
2. After the creation of the rules from WAFSmith, the same set of payloads are sent again.
3. Compare the difference in evasion rates before and after rules from WAFSmith were deployed.

In addition, simulated business traffic was sent to determine if the rules were blocking supposedly legitimate traffic. This is an additional simple test to ensure the reliability of the ruleset.

Evaluation Data

Type of Attack	**Number of Payloads**	**Source**
Cross-Site Scripting	2655	https://github.com/swisskyrepo/PayloadsAllTheThings/tree/master/XSS%20Injection/Intruders
Open Redirect	325	https://github.com/swisskyrepo/PayloadsAllTheThings/tree/master/Open%20Redirect/Intruder
SQL Injection	1455	https://github.com/swisskyrepo/PayloadsAllTheThings/tree/master/SQL%20Injection/Intruder
File Inclusion	7943	https://github.com/swisskyrepo/PayloadsAllTheThings/tree/master/File%20Inclusion/Intruders
Command Injection	531	https://github.com/swisskyrepo/PayloadsAllTheThings/tree/master/Command%20Injection/Intruder

HTTP Methods

In the benchmarking exercise, both GET Request and POST Request will be evaluated.

GET Request

Payload is embedded in the URL parameter

```
GET /?payload=<input%2btype%3dtext%2bvalue%3d"XSS"> HTTP/1.1
Host: 127.0.0.1
```

POST Request

Payload is embedded in the body as an URL Encoded Form Data

```
POST / HTTP/1.1
Host: 127.0.0.1
Content-Length: 37
Content-Type: application/x-www-form-urlencoded

payload=<input+type=text+value="XSS">
```

Ease of Management

Humans vs WAFSmith

Human operators will be invited to participate in a controlled test to create ModSecurity Rules for a given set of payloads. The following data will be collected during the controlled test:

- (A) Time Taken to develop regular expression
- (B) Time Taken to write the ModSecurity Rule(s) after developing the regular expression
- (C) Time Taken to deploy the ModSecurity Rule(s) and perform testing

Human operators are allowed to use any tools online except LLM-based agents.

The purpose of the controlled test is to provide benchmarking of the WAFSmith against human operators on the Ease of Management metric. The human operators involved in the test are of varying experience:

- Human Operator 1: Junior Cyber Security Engineer (1-2 YoE)
- Human Operator 2: Mid Level Cyber Security Engineer (2-4 YoE)
- Human Operator 3: Senior Cyber Security Engineer (6-8 YoE)

The difference in seniority provides a more fair and varied test against WAFSmith.

Evaluation Data

The list below contains a set of payloads that the human operators are tasked to work on.

```
navigator.vibrate(500)
;system('/usr/bin/id')
<A HREF="http://www.gohttp://www.google.com/ogle.com/">XSS</A>
/etc/httpd/logs/error_log
onwheel
&#X000003C;
\x3c
```

```
1' ORDER BY 1,2,3--+
/////example.com
etc%2fpasswd%00
```

A template ModSecurity Rule is provided to the human operators as well.

```
SecRule ARGS_GET "@rx x='<%'" "id:6558757574341,phase:2,deny,status:403,severity:2,tag:'Catch X Open Angle Bracket Payload in GET URL Params'"
```

WAFSmith at Scale

Data collection is performed during the benchmarking exercise for the Evasion Rate. The data collection aims to derive an Ease of Management score for larger payload sets.

Results

Disclaimer and Assumptions

- By default, the code base for WAFSmith has a certain degree of performance optimization such as multithreading
- Human operators have varying degrees of experience and skill sets which provide data variation
- Popular LLM offerings such as OpenAI, Claude, Germini, etc are likely to be indifferent in terms of performance related to WAFSmith.

Evasion Rate

Type of Attack	Number of Payloads	GET *Evasion Rate*		POST *Evasion Rate*	
		CRS 4.12.0 (Before)	WAFSmith (After)	CRS 4.12.0 (Before)	WAFSmith (After)
Cross Site Scripting	2655	428 (16.12%)	19 (0.72%)	428 (16.12%)	21 (0.79%)
Open Redirect	325	227 (69.85%)	29 (8.92%)	225 (69.23%)	10 (3.08%)
SQL Injection	1455	242 (16.63%)	45 (3.09%)	242 (16.63%)	42 (2.89%)
File Inclusion	7943	445 (5.60%)	103 (1.30%)	445 (5.60%)	101 (1.27%)
Command Injection	531	148 (27.87%)	13 (2.45%)	148 (27.87%)	22 (4.14%)

Improvement Analysis

GET				**POST**			
CRS 4.12.0	WAFSmith	Change in Evaded Payloads	(%) Change	CRS 4.12.0	WAFSmith	Change in Evaded Payloads	(%) Change
428	19	409	96%	428	21	407	95%
227	29	198	87%	225	10	215	96%
242	45	197	81%	242	42	200	83%
445	103	342	77%	445	101	344	77%
148	13	135	91%	148	22	126	85%
		Avg. Change	**86%**			Avg. Change	**87%**

Results Analysis

From the evasion rates collected before and after the use of WAFSmith, there are clear and significant drops in evasion rates. This concludes that WAFSmith can create rules that block payloads effectively.

- Based on the performance of the other payload types, it is conclusive that the underlying LLM is indifferent toward the rule creation of either POST or GET requests. However, the evasion rate for Open Redirect attacks has a significant difference between POST and GET requests. This is likely due to the temperature configuration of the LLM as it does tend to produce varying results even with the same prompts.

Ease of Management Benchmarking

Human Operator vs WAFSmith

Review the sample set of ModSecurity Rules generated in Appendix E: Data samples from Ease of Management Benchmarking

Legend

- Block Task: Number of payloads the rules are developed by the entities are supposed to block
- (A) Time Taken to develop regular expression
- (B) Time Taken to write the ModSecurity Rule(s) after developing the regular expression
- (C) Time Taken to deploy the ModSecurity Rule(s) and perform testing
- E.O.M Score: Ease of Management Score

	Block Task	(A)	(B)	(C)	Time	Processing Rate	Payloads Blocked	No. Rules	Block Rate	E.O.M Score
Human Operator 1	10	1533	510	134	2177	0.004593477	4	9	0.4	0.001837
Human Operator 2	10	1164	150	613	1927	0.005189414	10	10	1	0.005189
Human Operator 3	10	195	240	270	705	0.014184397	10	10	1	0.014184
WAFSmith	**10**		**121**			**0.082644628**	**10**	**7**	**1**	**0.082645**

Remarks

- Time measurement for WAFSmith to perform Column (A), (B), and (C) activities are captured in a single execution flow. Thus, the value is an aggregated value.
- WAFSmith is configured for 10 threads (minimally).

Result Analysis

WAFSmith emerged as a clear winner in the benchmarking exercise with a score that is nearly 80 times better than Human Operator 1. However, it is worth noting that compared to a skilled operator like Human Operator 3, WAFSmith performed only 5.85 times better. While skills matter, the difference in score is likely to increase as the scale of the test enlarges (e.g. a larger payload set).

WAFSmith at Scale

GET Request

	Number of Payloads Designed to Block	Time Taken (s)	Processing Rate	Number of Payloads Blocked	Number of ModSecurity Rules	Block Rate	Ease of Management Score
XSS	428	2755.356	0.15533383	382	68	5.617647059	**0.873**
Open Redirect	227	1320.276	0.171933747	195	96	2.03125	**0.349**
SQL Injection	242	1098.8	0.220240262	197	19	10.36842105	**2.284**
File Inclusion	445	1993.115	0.223268602	327	96	3.40625	**0.761**
Command Injection	148	864.066	0.171283212	133	11	12.09090909	**2.071**

Remarks

- Configured for 50 threads, payload positioned in URL Parameter

POST Request

	Number of Payloads Designed to Block	Time Taken (s)	Processing Rate	Number of Payloads Blocked	Number of ModSecurity Rules	Block Rate	Ease of Management Score
XSS	428	2738.899	0.156267172	407	38	10.71052632	**1.674**
Open Redirect	225	1377.747	0.1633101	213	20	10.65	**1.739**
SQL Injection	242	1008.873	0.239871619	200	11	18.18181818	**4.361**
File Inclusion	445	1699.554	0.261833399	327	78	4.192307692	**1.098**

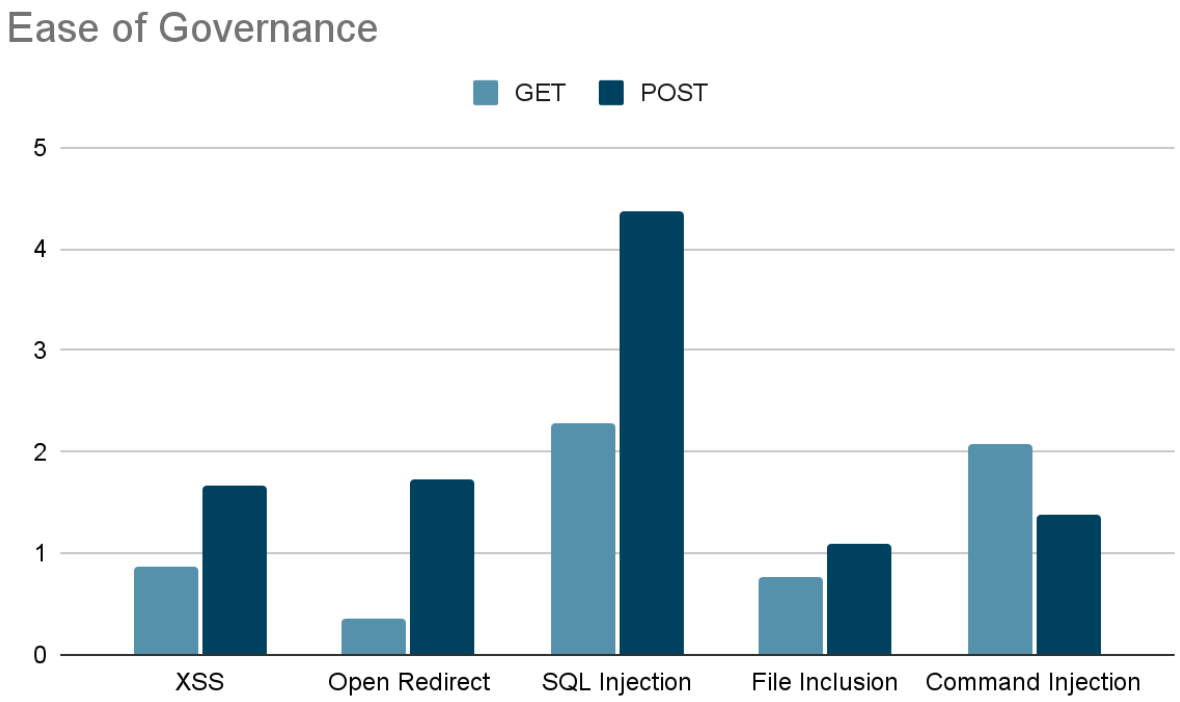
Command Injection	148	897.811	0.164845385	126	15	8.4	**1.385**

Remarks

- Configured for 50 threads, payload positioned in URL Parameter

Results Analysis

The Ease of Management scores are visualized in a bar chart below to understand how WAFSmith performs at scale.



Visually, we can observe that POST Requests, except Command Injection payloads, WAFSmith finds it easier to govern! However, POST request for SQL Injection is a case of an outlier in which it outperforms significantly compared to other attack types.

Discussion

Benefits

The benchmarking exercise concludes that WAFSmith can perform its intended design goals of Rule Management:

- Creating useful rules
- Simplifying ruleset maintenance and governance

Ultimately, it was developed to reduce the cost of Rule Management in enterprises. Blue Team can easily adopt this tool in their workflows to improve the efficacy of rule management. The reduced cost of Rule Management will translate to:

- Stronger security posture
- Increased enterprise agility to respond to threats
- Reduced manpower consumption

Creating useful rules

The Evasion Rate benchmarking demonstrated how much the attack surface can be reduced through the adoption of WAFSmith. On average, the evasion rate improved by 86.5% after the adoption of WAFSmith.

Ease of governing ruleset

In the evaluation against human operators, it is evident that WAFSmith can reliably scale and develop rules that are of quality (maybe even better) in a shorter amount of time. The level of automation allows enterprises to be unrivaled. In the data collected for the Evasion Rate benchmarking, WAFSmith scaled relatively well to develop ModSecurity rules that are designed to block hundreds (>100~400) payloads.

Limitations

Accountability for AI Agent

As the capabilities of AI agents develop, it is more likely that tasks that require lower cognitive workload will be replaced by these AI agents. However, enterprises have different risk appetites and trust in AI agents. While WAFSmith has proven to be a reliable agent in developing high-quality ModSecurity Rules, enterprises may adopt a human-in-the-loop approach to ensure accountability for WAFSmith output.

Novelty of WAFSmith

Advanced Self-Governing Rule Capabilities based on AI agents

WAFSmith is the first of its kind, aimed at empowering Blue-Teamers across the world with access to advanced LLM capabilities. Current developments in LLM allow unparalleled levels of automation and intelligence to perform difficult and laborious tasks. This capability has yet to be widely accessible in the WAF Rule Management space.

Proven Methodology for Different Kinds of Rule Management Scenarios

The methodology adopted in the solution is highly robust and meticulously designed to handle different kinds of rule-making scenarios. For instance, the tool can be easily modified to support Perl-Compatible Regular Expressions (PCRE) to develop ModSecurity rules or, easily modified to create SNORT rules (refer to Appendix G: Adaptation of WAFSmith for IDPS Solution).

The methodology emphasizes automation and testing to ensure high-quality rules and reliability. It is designed to operate as a standalone Command Line Interface (CLI) that is not reliant on any other products. This allows Blue Teamers to easily plug and play the tool in various kinds of Rule Management scenarios.

Developing open-source rulesets and tailored prompts for wider adoption

The by-products of the development process such as the ModSecurity Rules and Prompts are released for community usage. These by-products are tested and customized for the solution's proposed use case. Proposed use cases of the by-products:

- ModSecurity Rulesets (as shared in the repository) produced to improve CRS baselines are available for everyone to adopt.
- The prompts used in the solution can be adopted by security engineers in LLM-related application development

Conclusion

The current benchmarking exercises provide positive reinforcement of the novelty and practicality of WAFSmith. It is the first step in a novel approach toward WAF rule management and represents the potential in harnessing Artificial Intelligence in cyber-defence strategies.

WAFSmith demonstrated the ability of LLMs to orchestrate reactive action at a fraction of the cost and time. The cost savings introduced by adopting this work aim to:

- Shift of cognitive burden for human operators to higher-order tasks
- Introduce cost savings through gained productivity
- Reduce Mean Time To Respond (MTTR) for security-related incidents
- Allow bootstrapping of WAF rules

Future Works

On the longer horizon, it is part of a larger initiative to develop and improve security tools within the open-source space. It is also part of an active research project that is in the domain of AI and Cybersecurity.

Currently, WAFSmith is part of a Honeypot operation in which ModSecurity is deployed in the wild to collect data and perform a high degree of WAF governance automation (detecting attacks to automatically deploy generated rules to protect against these attacks).

The projects below are directions in which WAFSmith will pursue:

Automated WAF Management

WAFSmith represents a significant level of automation that can be introduced to the entire WAF Management operation. The current benchmarks show that a fully automated WAF solution using AI agents is feasible and potentially a highly practical solution.

Support for other types of Rule Management Use Cases

A similar use case in which WAFSmith is highly suitable to be adopted is for Snort Rule Management. Snort Rule Management is similar to how WAF rules are traditionally governance. In addition, the following use cases can be adopted:

- SAST Tools such as SemGrep
- Host-Based Firewall Rule Management

AI-Based Security Operations Center

As AI continues to make advances, especially in areas of reasoning and decision-making, it is foreseeable that activities within any cybersecurity organization can be automated and improved through AI, allowing humans to focus on more important tasks or decision-making processes. An article by RadiantSecurity further enforces such an idea. WAFSmith represents a small portion of this AI-driven dream!

Release(s)

Version	Expected Timeline	Release Notes
v1.0.0	17 March 2025 ● Aligned with Blackhat Arsenal of Tools	Functional Release ● v1.0 ○ extract ○ create ○ aggregate ○ evaluate Ruleset ● v1.0 (XSS/SQLi/Command Injection/Open-Redirect) Prompts ● v1.0 (extract/create/aggregate) Notes The initial release is targeted at the specific content: ● `extract` supports payload extraction on NGINX logs ● `create` supports creation of ModSecurity rules ● `evaluate` supports testing of Web Application Attacks
v1.1	By the end of April 2025 *Ongoing legal and administrative discussion	Prompts ● v1.1 Enhanced `extract` prompts for various types of text content (e.g. Apache Logs, Web Content, etc)

Acknowledgments

I would like to acknowledge Hitesh Yadav and Sean Yeoh's efforts to participate in reviewing both the technical and administrative elements of this project. Their guidance and insights into how human operators work in high-pressure scenarios, especially in cyber incidents proved invaluable to the project's motivation and roadmap. Not to forget Jordan Vaughn's continued support for this project to benefit the open-source community.

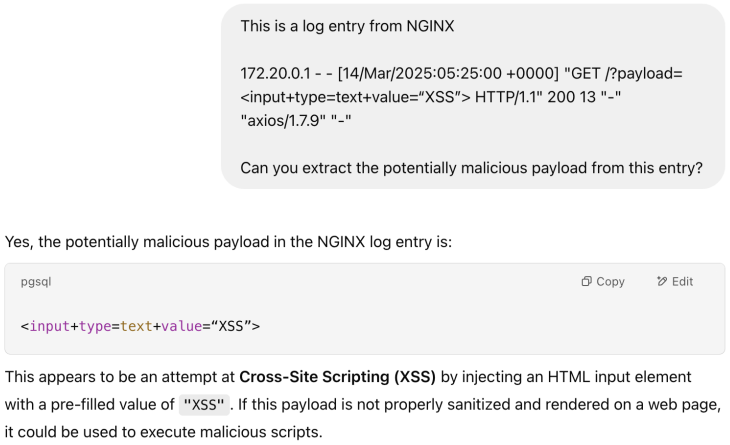
To other supportive folks, you know who you are! Thank you!

Appendix A: Case Study of Using OpenAI GPT4 for Regular Expression Generation

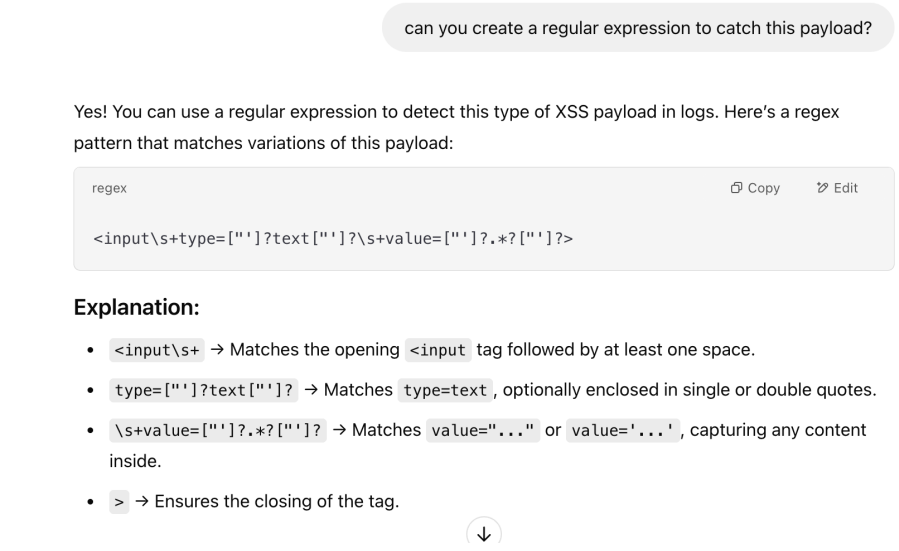
Conversation History: https://chatgpt.com/share/67d75289-8ff4-800f-9eea-d06b764b3727

The following case study is a simple evaluation performed on a popular Large Language Model such as OpenAI's ChatGPT ability to mimic a junior cybersecurity analyst writing a ModSecurity Rule.

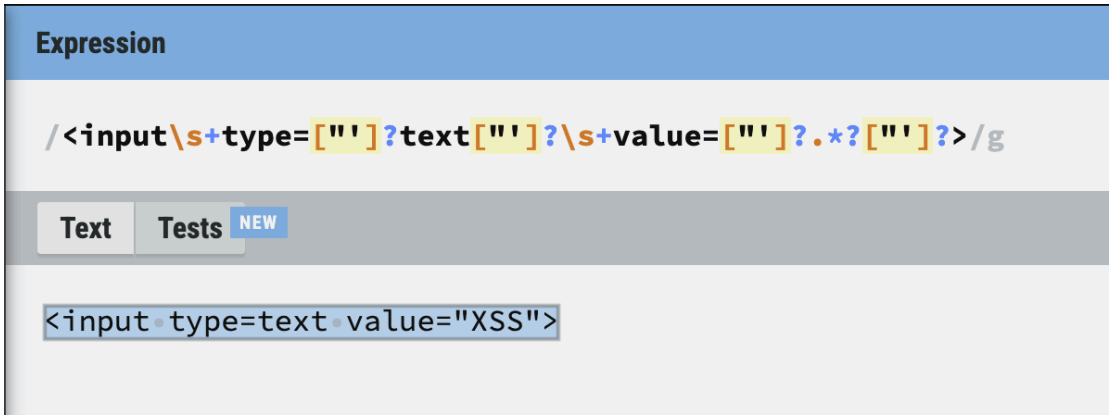
1. **Extracting the payload from a sample NGINX Log entry**



2. **Creating a Regular Expression**



3. Validating the Regular Expression



4. Prompting LLM to convert the Regular Expression to a ModSecurity Rule

can you write a modsecurity rule using that regular expression?

Sure! Below is a **ModSecurity** rule that detects potential **XSS attempts** using your specified regex pattern:

apache Copy Edit

```
SecRule ARGS "@rx <input\s+type=[\"']?text[\"']?\s+value=[\"']?.*?[\"']?>" \
    "id:1000001,\
    phase:2,\
    block,\
    msg:'Potential XSS Attack Detected',\
    log,\
    tag:'attack-xss',\
    severity:'CRITICAL'"
```

Breakdown of the Rule:

- `SecRule ARGS` → Inspects all request parameters (`GET`, `POST`, etc.).
- `@rx <input\s+type=[\"']?text[\"']?\s+value=[\"']?.*?[\"']?>` → Uses the regex to detect malicious input fields.
- `id:1000001` → Unique rule ID.
- `phase:2` → Executes in Phase 2 (after request body is parsed).
- `block` → Blocks the request if it matches.
- `msg:'Potential XSS Attack Detected'` → Logs a message if triggered.
- `log` → Ensures the request is logged.
- `tag:'attack-xss'` → Tags it for easier tracking.
- `severity:'CRITICAL'` → Marks it as a critical security event.

Optional Enhancements:

- If you want to inspect **headers** as well, replace `ARGS` with `REQUEST_HEADERS|ARGS|ARGS_NAMES`.
- If you prefer **only logging** without blocking, replace `block` with `log,pass`.

5. Testing the ModSecurity Rule

Before the implementation of the rule, a `curl` request was sent to observe the server response.

```
keehock@keehock-ubuntu:~/Workspace/wafsmith/cli-app$ curl '127.0.0.1/?payload=<input+type=text+value="XSS">
'
Hello, World!keehock@keehock-ubuntu:~/Workspace/wafsmith/cli-app$
```

The rule is then implemented in ModSecurity in a file named `custom.conf`.

custom.conf

cli-app / infra / rules / custom.conf

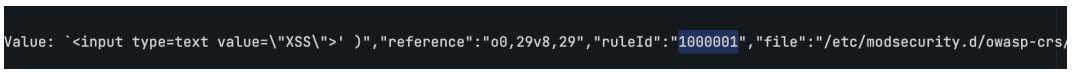
```
SecRule ARGS "@rx <input\s+type=[\"']?text[\"']?\s+value=[\"']?.*?[\"']?>" \
    "id:1000001,\
    phase:2,\
    block,\
    msg:'Potential XSS Attack Detected',\
    log,\
    tag:'attack-xss',\
    severity:'CRITICAL'"

```

Upon deploying the rule, ModSecurity service is restarted and the same `curl` request is sent again. The response is the same as what was observed previously.

```
keehock@keehock-ubuntu:~/Workspace/wafsmith/cli-app$ curl '127.0.0.1/?payload=<input+type=text+value="XSS">
'
Hello, World!keehock@keehock-ubuntu:~/Workspace/wafsmith/cli-app$
```

However, since the rule is configured for `block` mode, it will not deny the request. Instead, it should appear in ModSecurity's Audit Log.



A quick search in the logs with the Rule ID shows that the entry appeared! Thus, it shows that the proof of concept works!

Appendix D: Techniques used to improve WAFSmith

Improving Rule Generation

Generating Random Value instead of relying on LLM

Issue	LLM agents are observed not to be very good with random value generation during evaluation trials
Impact	LLM agents are unable to generate ModSecurity rules at scale due to ID collision when deployed
Solution	A random value is generated and substituted into the ModSecurity rule that was generated by LLM.

Adopting Chain of Thought Prompting

Issue	LLM agents may have difficulty processing a complex task such as creating a valid ModSecurity rule
Impact	The rules developed by LLM are not syntactically correct or with valid semantic meaning.
Solution	Provide step-by-step guidance on deriving the output. In addition, be explicit in the output which the LLM should not produce. For example "Respond in JSON format only."

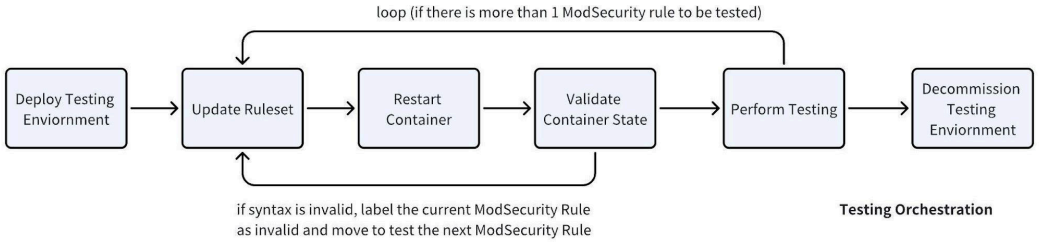
Perform preprocessing of data

Issue	During payload extract attempts, raw data might be encoded or formatted. For example, the following Nginx Log entry has been encoded: 172.20.0.1 - - [14/Mar/2025:05:25:00 +0000] "GET /?payload=%3Cinput+type%3Dtext+value%3D%E2%80%9CXSS%E2%80%9D%3E HTTP/1.1" 200 13 "-" "axios/1.7.9" "-"
Impact	The LLM agent generated a regular expression that catches the payload in its encoded form. However, it does not work in real-world attacks as WAF engines like ModSecurity will perform URL decoding.
Solution	Perform preprocessing steps such as URL encoding, before supplying the data to the LLM to perform the task. E.g.

172.20.0.1 - - [14/Mar/2025:05:25:00 +0000] "GET /?payload=<input+type=text+value="XSS"> HTTP/1.1" 200 13 "-" "axios/1.7.9" "-"

Improving interaction with Docker Containers

Docker containers are used to orchestrate a consistent testing environment. Various state checks were implemented to ensure the health of the containers. It is worth noting that an invalid ModSecurity rule can result in the container being unable to run properly. It is common for LLM to occasionally produce an invalid ModSecurity rule.



Appendix E: Data samples from Ease of Management Benchmarking

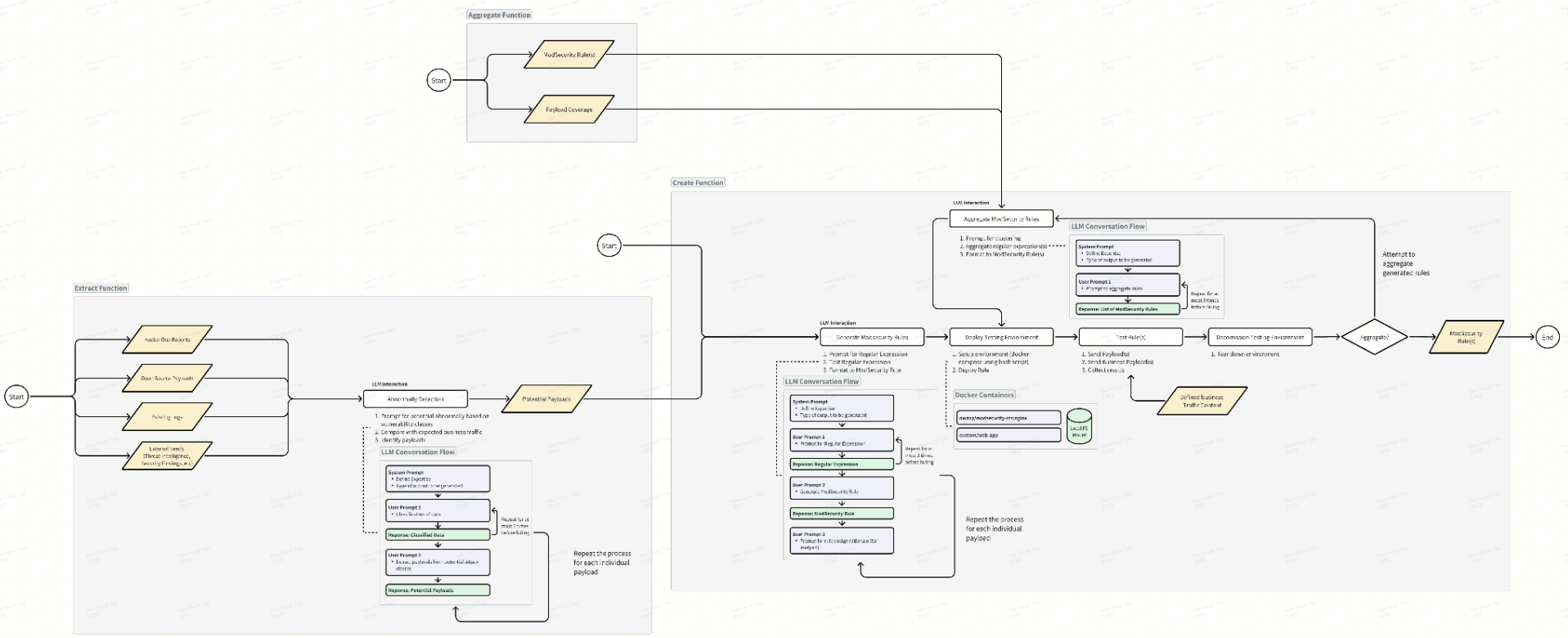
The payload below is the payload in which the subsequent ModSecurity Rules are designed to catch

```
;system('/usr/bin/id')
```

The table below shows the a sampled set of ModSecurity Rule(s) from the experiment for the given payload

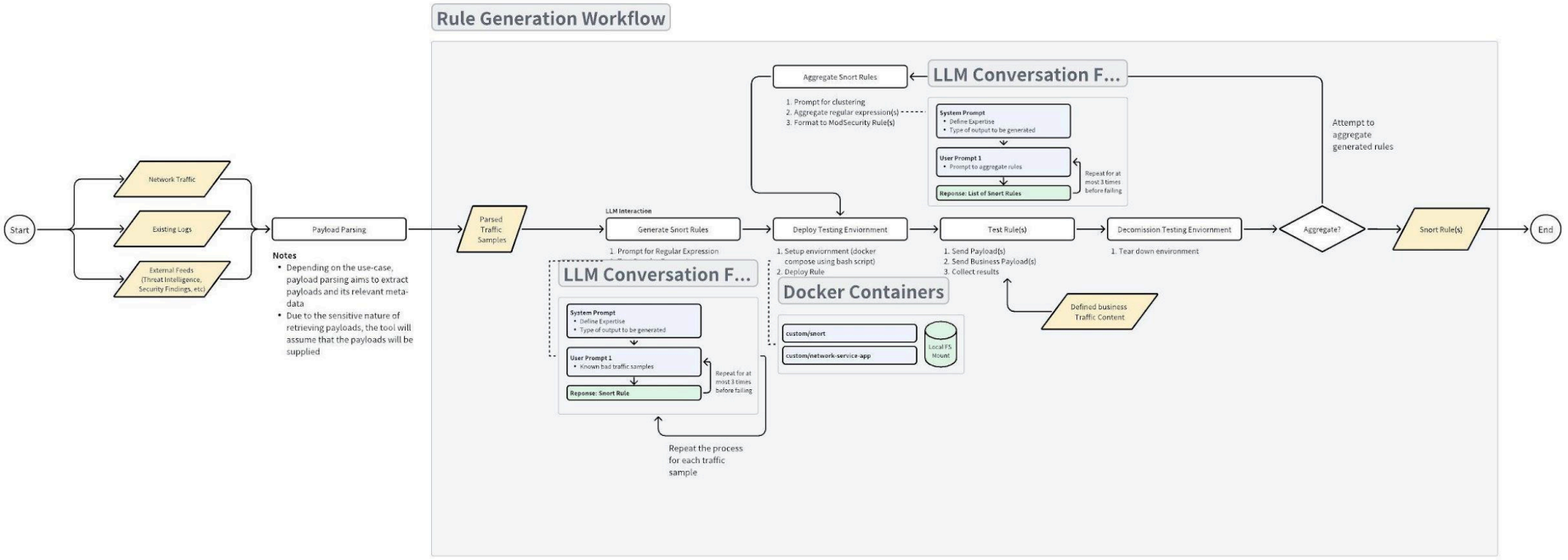
	ModSecurity Rule
Human Operator 1	SecRule ARGS_GET "@rx '\;system\(\'\/usr\/bin\/[A-z]+'" "id:6558757574342,phase:2,deny,status:403,severity:2,tag:'Catch Command injection'"
Human Operator 2	SecRule ARGS_GET "@rx ;(system|SYSTEM)\('\/.*\/.*'\)" "id:6558752571341,phase:2,deny,status:403,severity:2,tag:'hacked'"
Human Operator 3	SecRule ARGS_GET "@rx ;system\('.*?'\)" "id:45362358,phase:2,deny,status:403,severity:2,tag:'hehexd'"
WAFSmith	SecRule ARGS "|;system\s*\(\s*['\"]?.*?['\"]?\s*\)|" "id:7778889123,phase:2,deny,status:403,t:normalizePathWhitespace,t:urlDecodeUni,tag:'command-injection'"

Appendix F: WAFSmith Workflow(s)



WAFSmith: LLM based Rule Management Framework to Create Rules that Simply Work

Appendix G: Adaptation of WAFSmith for IDPS Solution



WAFSmith: LLM based Rule Management Framework to Create Rules that Simply Work