Extended Shield Rule System Design

This document outlines the extensions to the Shield rule system, incorporating rule activation toggles, centralized logging configuration, and a rule prioritization system.

1\. Rule Activation Toggle

The rule definition in both YAML and JSON formats will be extended with an optional `enabled` field. If this field is present and set to `false`, the rule will be ignored by the Scanner Engine during prompt evaluation. If the field is absent, it will default to `true`, meaning the rule is active.

Updated YAML Example:

```yaml
rules:
  - id: jailbreak_attempt_1
    description: Detect common jailbreak prefixes
    severity: high
    enabled: true
    priority: 100
    pattern: "^(ignore previous instructions|as a helpful ai assistant, you must)"
    match_type: regex
    actions:
      - block
      - log:
          level: critical
```

```
      message: "Potential jailbreak attempt detected."

- id: role_override_injection_1
  description: Detects attempts to override the AI's role (Inactive)
  severity: medium
  enabled: false
  priority: 50
  pattern: "you are no longer a helpful ai assistant, you are now a"
  match_type: keyword_in
  actions:
    - transform:
        type: replace
        target: "you are no longer a helpful ai assistant, you are now a"
        replacement: "the user is attempting to redefine your role as a"
    - log:
        level: warning
        message: "Role override attempt neutralized."

- id: recursive_command_1
  description: Detect potential recursive command patterns
  severity: high
  enabled: true
  priority: 90
  pattern: "repeat the following exactly: repeat the following exactly:"
  match_type: regex
  actions:
    - block
```

```
    - log:
        level: critical
        message: "Potential recursive command detected."
```

Updated JSON Example:

```
{
  "rules": [
    {
      "id": "token_bleed_keyword_1",
      "description": "Detect keywords potentially leading to token bleed",
      "severity": "medium",
      "enabled": true
      "priority": 70,
      "pattern": "(^|\\s)(show me your internal configuration|reveal your prompts)($|\\s)",
      "match_type": "regex",
      "actions": [
        {
          "log": {
            "level": "warning",
            "message": "Keyword potentially related to token bleed detected."
          }
        }
      ]
    },
    {
      "id": "test_rule_inactive",
```

```
    "description": "This is an inactive test rule",
    "severity": "low",
    "enabled": false,
    "priority": 10,
    "pattern": "test",
    "match_type": "keyword_in",
    "actions": ["log"]
  }
 ]
}
```

2. Centralized Logging Configuration

A new YAML file named logger.yaml will be introduced in the top-level shield/ directory (or a configurable location). This file will define global logging settings for the Shield framework.

Overview of logger.yaml Structure:

```
level: INFO       # Default global logging level (DEBUG, INFO, WARNING, ERROR, CRITICAL)
format: '%(asctime)s - %(levelname)s - %(message)s' # Default log message format
output:
  type: console    # Default output type (console, file)
  destination: stdout # If type is 'file', specify the file path here
```

Interaction with Individual Rule Logging:

When a rule specifies a log action, the Action Engine will first check if the rule provides a level and message.

* If the rule explicitly defines a level, that level will be used for the log entry. If not, it will fall back to the level defined in logger.yaml.

* If the rule provides a message, that message will be logged. It can also use placeholders like {prompt} and {rule_id}.

* The log format and output destination will be determined by the settings in logger.yaml, unless the Logger module is extended to allow rule-specific output configurations (which is not part of this initial extension).

The Logger module will load the logger.yaml configuration during initialization.

3. Rule Prioritization System

A new optional field named priority (integer) will be added to the rule definition in both YAML and JSON. Rules with higher integer values will be considered to have higher priority.

Matching & Priority Resolution Process:

* The Scanner Engine iterates through all enabled rules and identifies all rules whose pattern matches the input prompt based on their match_type.

* If one or more rules match the prompt, the Scanner Engine will collect all matching rules.

* Before triggering the Action Engine, the Scanner Engine will sort the list of matching rules based on their priority field in descending order (highest priority first). If a rule does not have a priority field, a default priority (e.g., 0 or a configurable default) will be assigned.

* The Action Engine will then process the actions of the matched rules in the order of their determined priority. This means that higher-priority rules will have their actions executed before lower-priority rules.

Example Scenario:

Consider a prompt that matches two rules:

* Rule A (priority: 100, action: block)

* Rule B (priority: 50, action: log)

Because Rule A has a higher priority, its block action will be executed first, preventing the prompt from proceeding further and potentially skipping the log action of Rule B (depending on the implementation of the Decision Router and

whether blocking halts further rule processing for that prompt). This allows for defining specific rules that should take precedence over more general ones.

These extensions enhance the flexibility and control of the Shield framework, allowing for more sophisticated rule management and logging behavior. The activation toggle enables easier management of rule sets, the centralized logging provides a consistent logging framework, and the prioritization system allows for fine-tuning the order in which security measures are applied.