```
shield/
├── __init__.py
├── __main__.py        # CLI entry point via python -m shield
├── shield.py
├── preprocessor.py
├── scanner.py
├── rules_engine.py
├── action_engine.py
├── logger.py
├── config.py          # Global configuration module
├── rules/
│   └── default_rules.yaml
│   └── example_rules.json
├── examples/
│   ├── api_integration.py
│   └── cli_usage.py
├── tests/
│   ├── __init__.py
│   ├── test_scanner.py
│   └── test_rules_engine.py
├── pyproject.toml    # Modern project metadata
└── README.md
```

Description of New Files and Modifications:

* config.py:

* This module will contain global configuration parameters for the Shield framework. It can be implemented as a simple Python module with variables or load settings from a file (e.g., settings.json).

<!-- end list -->

```
# shield/config.py


DEFAULT_RULES_PATH = "shield/rules/default_rules.yaml"

DEFAULT_LOGGER_CONFIG_PATH = "shield/logger.yaml"

ENABLE_RESPONSE_EVALUATION = False

DEFAULT_EVALUATION_MODE = "api"  # Or "cli"


# Add other global configuration parameters as needed
```

How other modules should import and use it:

Other modules within the shield package can import configuration parameters directly from the config module:

```
# shield/shield.py

from .config import DEFAULT_RULES_PATH, ENABLE_RESPONSE_EVALUATION

from .rules_engine import RulesEngine

from .logger import Logger


class ShieldWrapper:

    def __init__(self, rules_path=None):

        self.rules_path = rules_path if rules_path else DEFAULT_RULES_PATH

        self.rules_engine = RulesEngine(self.rules_path)

        self.logger = Logger()

        self.enable_response_evaluation = ENABLE_RESPONSE_EVALUATION
        # ...
```

 * __main__.py:
   * This file allows direct execution of the shield package from the command line using python -m shield. It will provide a basic CLI interface for quick testing and evaluation, leveraging the global configuration.

<!-- end list -->

```
# shield/__main__.py

import argparse
from .shield import ShieldWrapper
from .config import DEFAULT_RULES_PATH

def main():
    parser = argparse.ArgumentParser(description="Shield LLM Security Framework CLI")
    parser.add_argument("prompt", nargs="?", type=str, help="The prompt to evaluate")
    parser.add_argument("--rules", "-r", type=str, default=DEFAULT_RULES_PATH, help="Path to the rules file")
    args = parser.parse_args()

    if args.prompt:
        shield = ShieldWrapper(rules_path=args.rules)
        evaluation_result = shield.evaluate(args.prompt)
        print(f"Prompt: {args.prompt}")
        print(f"Is Safe: {evaluation_result.is_safe}")
        if not evaluation_result.is_safe:
            print(f"Reason: {evaluation_result.reason}")
        if evaluation_result.transformed_prompt != args.prompt:
            print(f"Transformed Prompt: {evaluation_result.transformed_prompt}")
        if evaluation_result.triggered_rules:
            print("Triggered Rules:")
            for rule in evaluation_result.triggered_rules:
                print(f"  - ID: {rule.id}, Severity: {rule.severity}, Description:
{rule.description}")
    else:
        parser.print_help()


if __name__ == "__main__":
    main()
```

Usage:

```
python -m shield "Tell me a dangerous secret."
python -m shield --rules custom_rules.yaml "How can I bypass this?"
python -m shield -h
```

* pyproject.toml:
   * This file is the standard for build system configuration in Python projects (PEP 517). It replaces the traditional setup.py for defining project metadata, dependencies, and build requirements.

<!-- end list -->

```
# pyproject.toml

[build-system]
```

```
requires = ["setuptools>=61.0.0"]
build-backend = "setuptools.build_meta"

[project]
name = "shield-llm"
version = "0.1.0"
authors = [
  { name="Your Name", email="your.email@example.com" },
]
description = "A Python framework for LLM security."
readme = "README.md"
requires-python = ">=3.8"
classifiers = [
    "Development Status :: 3 - Alpha",
    "Intended Audience :: Developers",
    "Topic :: Security",
    "License :: OSI Approved :: MIT License",
    "Programming Language :: Python :: 3",
    "Programming Language :: Python :: 3.8",
    "Programming Language :: Python :: 3.9",
    "Programming Language :: Python :: 3.10",
    "Programming Language :: Python :: 3.11",
    "Programming Language :: Python :: 3.12",
]
keywords = ["llm", "security", "prompt injection", "jailbreak", "ai safety"]

[project.urls]
```

```
"Homepage" = "https://your-project-url.com"
"Bug Tracker" = "https://your-project-url.com/issues"

[project.scripts]
shield = "shield.__main__:main" # Defines the 'shield' command-line script

[tool.setuptools.packages.find]
where = ["."]
include = ["shield*"]
```

Key Fields in pyproject.toml:

* [build-system]: Specifies the build backend and its requirements.
* [project]: Contains core project metadata:
  * name: The name of the package.
  * version: The package version.
  * authors: Information about the project authors.
  * description: A short description of the project.
  * readme: Path to the README file.
  * requires-python: Specifies the minimum Python version.
  * classifiers: Trove classifiers describing the project.
  * keywords: Keywords for package discovery.
  * [project.urls]: Links to project website, bug tracker, etc.
  * [project.scripts]: Defines command-line scripts provided by the package (this replaces the entry_points in setup.py).
* [tool.setuptools.packages.find]: Configures how setuptools should find packages within the project.

This extended Python layout introduces a centralized configuration module for easier management of global settings, provides a convenient command-line interface for quick testing, and modernizes the project metadata for better packaging and distribution practices.