Informações de Debug do contrato `SpyErcBridge`

Esses são os argumentos que foram passados para o método fire_lasers da classe MythrilAnalyzer.

```python
 def fire_lasers(
        self,
        modules: Optional[List[str]] = None,
        transaction_count: Optional[int] = None,
    ) -> Report:
```

```python
modules= None
```

```python
self=
{'address': '0x00000000...',
 'contracts': [<mythril.solidity.soliditycontract.SolidityContract object at
0x7f2dafe01550>],
 'create_timeout': 30,
 'custom_modules_directory': '',
 'disable_dependency_pruning': False,
 'eth': 'None',
 'execution_timeout': 3600,
 'loop_bound': 3,
 'max_depth': 128,
 'strategy': 'bfs',
 'use_onchain_data': True}
```

```python
transaction_count= 2
```

Finalidade de cada dado de depuração

`self` no Contexto da Função `fire_lasers`

- **Representação**:
  Representa a instância da classe `MythrilAnalyzer` que está executando o método `fire_lasers`.
  Contém informações sobre o contrato a ser analisado, estratégia de análise e configurações relevantes.
- **Contribuição para o Sucesso da Análise**:
  Armazena o estado interno necessário para conduzir a análise, incluindo a lista de contratos e parâmetros configurados.

Contexto da Função `fire_lasers`

- **Descrição**:
  É o método principal que inicia a análise simbólica e a detecção de vulnerabilidades.
  Recebe parâmetros como `modules` (módulos de análise a serem executados) e `transaction_count` (número de transações a serem simuladas).

Parâmetros

- **`modules = None`**:
  Indica que todos os módulos padrão serão executados.
- **`transaction_count = 2`**:
  O Mythril irá simular sequências de até 2 transações, permitindo detectar vulnerabilidades que requerem múltiplas interações.

`self` (Instância de `MythrilAnalyzer`)

- **Configurações Importantes**:
  - `strategy = 'bfs'` (Breadth-First Search): Define como o espaço de estados é explorado.
  - `use_onchain_data = True`: Permite que o Mythril obtenha dados diretamente da blockchain, enriquecendo a análise com informações reais.

Código - Observações

- Locais marcados com `# f`:
  Indicam que, durante a execução do contrato analisado, a condição **não foi satisfeita** e, portanto, **não entrou** na condicional associada.

- Locais marcados com `# v`:
  Indicam que a condição **foi satisfeita** e, durante a execução, o contrato **entrou** na condicional associada.

Essas marcações ajudam a identificar o comportamento do contrato em diferentes cenários de execução.

Python

```
def fire_lasers(
        self,
        modules: Optional[List[str]] = None,
        transaction_count: Optional[int] = None,
    ) -> Report:
        """
        :param modules: The analysis modules which should be executed
        :param transaction_count: The amount of transactions to be executed
        :return: The Report class which contains the all the
issues/vulnerabilities
        """
        all_issues: List[Issue] = []
        SolverStatistics().enabled = True
        exceptions = []
        execution_info: Optional[List[ExecutionInfo]] = None
        for contract in self.contracts:
            StartTime()  # Reinitialize start time for new contracts
            try:
                sym = SymExecWrapper( # v
                    contract,
                    self.address,
                    self.strategy,
                    dynloader=DynLoader(self.eth,
active=self.use_onchain_data),
                    max_depth=self.max_depth,
                    execution_timeout=self.execution_timeout,
                    loop_bound=self.loop_bound,
                    create_timeout=self.create_timeout,
                    transaction_count=transaction_count,
                    modules=modules,
                    compulsory_statespace=False,
```

```
disable_dependency_pruning=self.disable_dependency_pruning,
                    custom_modules_directory=self.custom_modules_directory,
                )
                issues = fire_lasers(sym, modules) # v
                execution_info = sym.execution_info # v
            except DetectorNotFoundError as e:
                # Bubble up
                raise e
            except KeyboardInterrupt:
                log.critical("Keyboard Interrupt")
                issues = retrieve_callback_issues(modules)
            except Exception:
                log.critical(
                    "Exception occurred, aborting analysis. Please report
this issue to the Mythril GitHub page.\n"
                    + traceback.format_exc()
                )
                issues = retrieve_callback_issues(modules)
                exceptions.append(traceback.format_exc())
            for issue in issues: # v
                issue.add_code_info(contract)

            all_issues += issues
            log.info("Solver statistics:
\n{}".format(str(SolverStatistics())))

        source_data = Source() # v
        source_data.get_source_from_contracts_list(self.contracts) # v

        # Finally, output the results
        report = Report( # v
            contracts=self.contracts,
            exceptions=exceptions,
            execution_info=execution_info,
        )
        for issue in all_issues:
            report.append_issue(issue)

        return report
```