Informações de Debug do contrato `SpyErcBridge`

Esses são os argumentos que foram passados para a função analisada.

```python
def execute_command(
    disassembler: MythrilDisassembler,
    address: str,
    parser: ArgumentParser,
    args: Namespace,
):
```

```python
address=
'0x0000000000000000000000000000000000000000'
```

```python
parser=
{'add_help': True,
 'allow_abbrev': True,
 'argument_default': 'None',
 'conflict_handler': 'error',
 'description': 'Security a...',
 'epilog': 'None',
 'exit_on_error': True,
 'fromfile_prefix_chars': 'None',
 'prefix_chars': '-',
 'prog': 'myth',
 'usage': 'None'}
```

```python
args=
{'address': 'None',
```

```
'attacker_address': 'None',
'beam_search': 'None',
'bin_runtime': False,
'call_depth_limit': 3,
'code': 'None',
'codefile': 'None',
'command': 'analyze',
'create_timeout': 30,
'creator_address': 'None',
'custom_modules_directory': 'None',
'disable_coverage_strategy': False,
'disable_dependency_pruning': False,
'disable_iprof': False,
'disable_mutation_pruner': False,
'enable_physics': False,
'enable_state_merging': False,
'enable_summaries': False,
'epic': False,
'execution_timeout': 3600,
'graph': 'None',
'infura_id': 'None',
'loop_bound': 3,
'max_depth': 128,
'modules': 'None',
'no_onchain_data': False,
'outform': 'text',
'parallel_solving': False,
'phrack': False,
'pruning_factor': 'None',
'query_signature': False,
'rpc': 'infura-mai...',
'rpctls': False,
'solc_args': 'None',
'solc_json': 'None',
'solidity_files': ['/home/mat/workspace/mythril/sContract.sol'],
'solv': 'None',
'solver_log': 'None',
'solver_timeout': 25000,
'statespace_json': 'None',
'strategy': 'bfs',
'transaction_count': 2,
'transaction_sequences': 'None',
'unconstrained_storage': False,
'v': 2}
```

Python

```
disassembler=
{'contracts': [<mythril.solidity.soliditycontract.SolidityContract object at
0x7f35547af4d0>],
 'eth': 'None',
 'sigs': {'path': '...', 'solidity_sigs': '...'},
 'solc_binary': 'None',
 'solc_settings_json': 'None',
 'solc_version': '0.8.0'}
```

Python

```
disassembler.contracts[0]=
{'bytecode_hash' :
'0x2f0d3fcb8eefe094eb4c1c5620d1366171b7eaad08242d1596c149617cd10fa3'
'code': '6080604052...',
 'constructor_mappings': [<mythril.solidity.soliditycontract.SourceMapping
object at 0x7f3553d79d60>,
                          <mythril.solidity.soliditycontract.SourceMapping
object at 0x7f3553d79d30>,
                                                  '...'],
'creation_bytecode_hash':
'0x6cf118e3f56e0809fb3f250ee872ade704f7913adaaf1c3c6ef262f1289e7a25'
 'creation_code': '6080604052...',
 'creation_disassembly': {'address_to_function_name': '...',
                          'bytecode': '...',
                          'func_hashes': '...',
                          'function_name_to_address': '...',
                          'instruction_list': '...'},
 'disassembly': {'address_to_function_name': '...',
                 'bytecode': '...',
                 'func_hashes': '...',
                 'function_name_to_address': '...',
                 'instruction_list': '...'},
 'features': {'': '...',
              'allowance': '...',
              'approve': '...',
              'balanceOf': '...',
              'burn': '...',
              'mint': '...',
              'totalSupply': '...',
              'transfer': '...',
              'transferAnyERC20Token': '...',
              'transferFrom': '...',
              'transferOwnership': '...',
```

```
              'transferTokenOwnership': '...',
              'vtransfer': '...'},
 'input_file': '/home/mat/...',
 'mappings': [<mythril.solidity.soliditycontract.SourceMapping object at
0x7f3553d36cf0>,
              <mythril.solidity.soliditycontract.SourceMapping object at
0x7f3553d36c90>,

              '...'],
 'name': 'SpyErcBrid...',
 'solc_indices': {0: '...', 1: '...'},
 'solc_json': {'contracts': '...', 'sources': '...'}}
```

Python

```
disassembler.contracts[0].constructor_mappings[0]=
{
{'length': 1839,
 'lineno': 'None',
 'offset': 1146,
 'solc_mapping': '1146:1839:...',
 'solidity_file_idx': 0}

}
```

Python

```
disassembler.contracts[0].creation_disassembly=
{
{'address_to_function_name': {150: '...',
                              191: '...',
                              232: '...',
                              293: '...',
                              336: '...',
                              379: '...',
                              420: '...',
                              461: '...',
                              504: '...'},
 'bytecode': '6080604052...',
 'func_hashes': ['0xaffed0e0',
                 '0xd493b9ac',
```

```
                '0xf2fde38b',
                '0xf851a440',
                '0xfc0c546a',
                '0x21e6b53d',
                '0x8326acce',
                '0x91dae519',
                '0xa0905880'],
 'function_name_to_address': {'admin()': '...',
                              'feepayer()': '...',
                              'nonce()': '...',
                              'processedNonces(uint256)': '...',
                              'token()': '...',

'transferAnyERC20Token(address,address,uint256)': '...',
                              'transferOwnership(address)': '...',
                              'transferTokenOwnership(address)': '...',
                              'vtransfer(address,uint256,uint256)': '...'},
 'instruction_list': [{...},
                      {...},
                      {...},
                      {...},
                      {...},
                      {...},
                      {...},
                      {...},
                      {...},
                      {...},
                      '...']}
}
```

Python

```
disassembler.contracts[0].disassembly=
{'address_to_function_name': {150: '...',
                              191: '...',
                              232: '...',
                              293: '...',
                              336: '...',
                              379: '...',
                              420: '...',
                              461: '...',
                              504: '...'},
 'bytecode': '6080604052...',
 'func_hashes': ['0xaffed0e0',
```

```
                  '0xd493b9ac',
                  '0xf2fde38b',
                  '0xf851a440',
                  '0xfc0c546a',
                  '0x21e6b53d',
                  '0x8326acce',
                  '0x91dae519',
                  '0xa0905880'],
 'function_name_to_address': {'admin()': '...',
                              'feepayer()': '...',
                              'nonce()': '...',
                              'processedNonces(uint256)': '...',
                              'token()': '...',

'transferAnyERC20Token(address,address,uint256)': '...',
                              'transferOwnership(address)': '...',
                              'transferTokenOwnership(address)': '...',
                              'vtransfer(address,uint256,uint256)': '...'},
 'instruction_list': [{...},
                      {...},
                      {...},
                      {...},
                      {...},
                      {...},
                      {...},
                      {...},
                      {...},
                      {...},
                      '...']}
```

Python

```
disassembler.contracts[0].features=
{'': {'all_require_vars': '...',
      'contains_assert': '...',
      'contains_call': '...',
      'contains_callcode': '...',
      'contains_delegatecall': '...',
      'contains_selfdestruct': '...',
      'contains_staticcall': '...',
      'has_owner_modifier': '...',
      'is_payable': '...',
      'transfer_vars': '...'},
```

```
'allowance': {'all_require_vars': '...',
              'contains_assert': '...',
              'contains_call': '...',
              'contains_callcode': '...',
              'contains_delegatecall': '...',
              'contains_selfdestruct': '...',
              'contains_staticcall': '...',
              'has_owner_modifier': '...',
              'is_payable': '...',
              'transfer_vars': '...'},
'approve': {'all_require_vars': '...',
            'contains_assert': '...',
            'contains_call': '...',
            'contains_callcode': '...',
            'contains_delegatecall': '...',
            'contains_selfdestruct': '...',
            'contains_staticcall': '...',
            'has_owner_modifier': '...',
            'is_payable': '...',
            'transfer_vars': '...'},
'balanceOf': {'all_require_vars': '...',
              'contains_assert': '...',
              'contains_call': '...',
              'contains_callcode': '...',
              'contains_delegatecall': '...',
              'contains_selfdestruct': '...',
              'contains_staticcall': '...',
              'has_owner_modifier': '...',
              'is_payable': '...',
              'transfer_vars': '...'},
'burn': {'all_require_vars': '...',
         'contains_assert': '...',
         'contains_call': '...',
         'contains_callcode': '...',
         'contains_delegatecall': '...',
         'contains_selfdestruct': '...',
         'contains_staticcall': '...',
         'has_owner_modifier': '...',
         'is_payable': '...',
         'transfer_vars': '...'},
'mint': {'all_require_vars': '...',
         'contains_assert': '...',
         'contains_call': '...',
         'contains_callcode': '...',
         'contains_delegatecall': '...',
         'contains_selfdestruct': '...',
         'contains_staticcall': '...',
         'has_owner_modifier': '...',
```

```
        'is_payable': '...',
        'transfer_vars': '...'},
 'totalSupply': {'all_require_vars': '...',
                 'contains_assert': '...',
                 'contains_call': '...',
                 'contains_callcode': '...',
                 'contains_delegatecall': '...',
                 'contains_selfdestruct': '...',
                 'contains_staticcall': '...',
                 'has_owner_modifier': '...',
                 'is_payable': '...',
                 'transfer_vars': '...'},
 'transfer': {'all_require_vars': '...',
              'contains_assert': '...',
              'contains_call': '...',
              'contains_callcode': '...',
              'contains_delegatecall': '...',
              'contains_selfdestruct': '...',
              'contains_staticcall': '...',
              'has_owner_modifier': '...',
              'is_payable': '...',
              'transfer_vars': '...'},
 'transferAnyERC20Token': {'all_require_vars': '...',
                           'contains_assert': '...',
                           'contains_call': '...',
                           'contains_callcode': '...',
                           'contains_delegatecall': '...',
                           'contains_selfdestruct': '...',
                           'contains_staticcall': '...',
                           'has_owner_modifier': '...',
                           'is_payable': '...',
                           'transfer_vars': '...'},
 'transferFrom': {'all_require_vars': '...',
                  'contains_assert': '...',
                  'contains_call': '...',
                  'contains_callcode': '...',
                  'contains_delegatecall': '...',
                  'contains_selfdestruct': '...',
                  'contains_staticcall': '...',
                  'has_owner_modifier': '...',
                  'is_payable': '...',
                  'transfer_vars': '...'},
 'transferOwnership': {'all_require_vars': '...',
                       'contains_assert': '...',
                       'contains_call': '...',
                       'contains_callcode': '...',
                       'contains_delegatecall': '...',
                       'contains_selfdestruct': '...',
```

```
                      'contains_staticcall': '...',
                      'has_owner_modifier': '...',
                      'is_payable': '...',
                      'transfer_vars': '...'},
 'transferTokenOwnership': {'all_require_vars': '...',
                            'contains_assert': '...',
                            'contains_call': '...',
                            'contains_callcode': '...',
                            'contains_delegatecall': '...',
                            'contains_selfdestruct': '...',
                            'contains_staticcall': '...',
                            'has_owner_modifier': '...',
                            'is_payable': '...',
                            'transfer_vars': '...'},
 'vtransfer': {'all_require_vars': '...',
               'contains_assert': '...',
               'contains_call': '...',
               'contains_callcode': '...',
               'contains_delegatecall': '...',
               'contains_selfdestruct': '...',
               'contains_staticcall': '...',
               'has_owner_modifier': '...',
               'is_payable': '...',
               'transfer_vars': '...'}}
```

Python

```
disassembler.contracts[0].mappings[0]=

{'length': 1839,
 'lineno': 'None',
 'offset': 1146,
 'solc_mapping': '1146:1839:...',
 'solidity_file_idx': 0}
```

Finalidade de cada dado de depuração

---

address

- **Finalidade:** Representa o endereço do contrato no blockchain Ethereum que está sendo analisado. No caso específico de `'0x0000000000000000000000000000000000000000'`, indica que o contrato não está implantado na rede.
- **Contexto da análise:** Utilizado para identificar e carregar os códigos associados a contratos já implantados.

`parser`

- **Finalidade:** É uma instância de `ArgumentParser` que analisa os argumentos de linha de comando fornecidos pelo usuário ao executar o Mythril.
- **Contexto da análise:** Configura o comportamento da ferramenta com base nos parâmetros fornecidos pelo usuário, como nível de log, modo de saída e opções específicas de análise.

`args`

- **Finalidade:** Um objeto `Namespace` que contém os valores dos argumentos processados pelo `parser`.
- **Contexto da análise:** Define quais ações o Mythril executará, como análise, desmontagem ou outras operações, além de parâmetros específicos (por exemplo, profundidade máxima, estratégia de execução simbólica, arquivos de entrada).

`disassembler`

- **Finalidade:** Uma instância de `MythrilDisassembler`, responsável por carregar e desmontar o código do contrato Solidity ou bytecode EVM.
- **Contexto da análise:** Converte o código fonte ou bytecode em uma representação interna para análise simbólica e detecção de vulnerabilidades.

Momento em que os dados são gerados

- **`args` e `parser`:** Criados no início da execução do programa, durante o processamento dos argumentos de linha de comando.
- **`disassembler`:** Inicializado após o processamento dos argumentos e antes da análise principal.
- **`address`:** Determinado durante o carregamento do código do contrato, seja de um endereço na blockchain, um arquivo de bytecode ou código fonte Solidity.

Uso posterior

Construção da análise e geração do espaço de estados

1. O `disassembler` utiliza `address` e `args` para carregar o contrato e desmontar o código em instruções EVM.
2. Estruturas como `disassembly`, `features`, `mappings` e `constructor_mappings` são usadas para criar uma representação detalhada do contrato, incluindo instruções, mapeamentos de origem e características.
3. Essa representação é utilizada pelo analisador simbólico para explorar o espaço de estados do contrato, simulando execuções possíveis e identificando vulnerabilidades.

Identificação de vulnerabilidades

- **Funções (`func_hashes`, `instruction_list`):** Definem pontos de entrada e caminhos de execução.
- **Características (`features`):** Identificam padrões específicos (por exemplo, uso de chamadas externas ou autodestruição) que podem indicar vulnerabilidades.
- **Mapeamentos (`mappings` e `constructor_mappings`):** Relacionam partes do bytecode ao código fonte original, permitindo localizar vulnerabilidades no código Solidity.

Integração com outras ferramentas

- **Interação com módulos internos:**
  - O `disassembler` fornece ao analisador as instruções e estruturas necessárias para a análise.
  - As características (`features`) podem ativar ou desativar módulos específicos de detecção de vulnerabilidades.
- **Integrações externas (como SMT solver):**
  - Durante a análise simbólica, condições e restrições são enviadas a um SMT solver para verificar a viabilidade de execuções ou identificar vulnerabilidades.

Relação entre os elementos principais

1. **`parser`** analisa os argumentos e preenche o objeto `args`.
2. **`args`** determina o comportamento do `disassembler`:
   - Se `args.solidity_files` contém arquivos, o `disassembler` compila esses arquivos.

    - Se `args.address` contém um endereço, o `disassembler` busca o bytecode na rede.
3. **`address`** é usado pelo `disassembler` para carregar e associar a análise ao contrato correto.

Estruturas de análise

- **`disassembly`:** Representa o bytecode desmontado do contrato, incluindo lista de instruções (`instruction_list`), hashes de funções (`func_hashes`) e mapeamentos entre endereços e nomes de funções.
- **`constructor_mappings`:** Relaciona o bytecode de criação com o código fonte.
- **`features`:** Identifica padrões relevantes no código (como `call`, `delegatecall`, verificações `require`).
- **`mappings`:** Relaciona bytecode com o código fonte, permitindo correlacionar vulnerabilidades com linhas específicas no Solidity.

Propriedades específicas

- **`instruction_list`:** Define as instruções EVM que o contrato executa, fundamentais para a análise simbólica.
- **`func_hashes`:** Identifica pontos de entrada no contrato por meio dos hashes de assinaturas de funções.
- **`creation_code` e `bytecode`:**
    - `creation_code`: Bytecode para criar o contrato, incluindo o constructor.
    - `bytecode`: Bytecode executado após a implantação.

Insights sobre o mapeamento bytecode e código-fonte

1. O Mythril compila o código-fonte Solidity em bytecode usando o compilador Solidity (`solc`).
2. O `disassembler` converte o bytecode em uma representação legível (`disassembly`).
3. Os `mappings` mantêm a relação entre bytecode e código fonte, permitindo que vulnerabilidades identificadas no bytecode sejam mapeadas para linhas do Solidity.

Conclusão

O Mythril combina argumentos do usuário, desmontagem de bytecode, mapeamento com o código-fonte e extração de características para realizar análises aprofundadas de contratos Solidity. Apesar de limitado por abstrações e possíveis erros de transformação, ele fornece insights acionáveis para a identificação e correção de vulnerabilidades em contratos inteligentes.

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

```python
def execute_command(
    disassembler: MythrilDisassembler,
    address: str,
    parser: ArgumentParser,
    args: Namespace,
):
    """
    Execute command
    :param disassembler:
    :param address:
    :param parser:
    :param args:
    :return:
    """
    if getattr(args, "beam_search", None): # f
        strategy = f"beam-search: {args.beam_search}"
    else:
        strategy = getattr(args, "strategy", "dfs") # v

    if args.command == READ_STORAGE_COMNAND: # f
        storage = disassembler.get_state_variable_from_storage(
            address=address,
            params=[a.strip() for a in
args.storage_slots.strip().split(",")],
```

```
        )
        print(storage)

    elif args.command in DISASSEMBLE_LIST: # f
        if disassembler.contracts[0].code:
            print("Runtime Disassembly: \n" +
disassembler.contracts[0].get_easm())
        if disassembler.contracts[0].creation_code:
            print("Disassembly: \n" +
disassembler.contracts[0].get_creation_easm())

    elif args.command == SAFE_FUNCTIONS_COMMAND: # f
        args.no_onchain_data = args.disable_dependency_pruning = (
            args.unconstrained_storage
        ) = True
        args.pruning_factor = 1
        function_analyzer = MythrilAnalyzer(
            strategy=strategy, disassembler=disassembler, address=address,
cmd_args=args
        )
        try:
            report = function_analyzer.fire_lasers(
                modules=(
                    [m.strip() for m in args.modules.strip().split(",")]
                    if args.modules
                    else None
                ),
                transaction_count=1,
            )
            print_function_report(disassembler, report)
        except DetectorNotFoundError as e:
            exit_with_error("text", format(e))
        except CriticalError as e:
            exit_with_error("text", "Analysis error encountered: " +
format(e))

    elif args.command in ANALYZE_LIST + FOUNDRY_LIST: # v
        analyzer = MythrilAnalyzer(  # v
            strategy=strategy, disassembler=disassembler, address=address,
cmd_args=args
        )

        if not disassembler.contracts: # f
            exit_with_error(
                args.outform, "input files do not contain any valid
contracts"
            )
```

```
        if args.attacker_address: # f
            try:
                ACTORS["ATTACKER"] = args.attacker_address
            except ValueError:
                exit_with_error(args.outform, "Attacker address is invalid")

        if args.creator_address: #  f
            try:
                ACTORS["CREATOR"] = args.creator_address
            except ValueError:
                exit_with_error(args.outform, "Creator address is invalid")

        if args.graph: # f
            html = analyzer.graph_html(
                contract=analyzer.contracts[0],
                enable_physics=args.enable_physics,
                phrackify=args.phrack,
                transaction_count=args.transaction_count,
            )

            try:
                with open(args.graph, "w") as f:
                    f.write(html)
            except Exception as e:
                exit_with_error(args.outform, "Error saving graph: " +
str(e))

        elif args.statespace_json: # f

            if not analyzer.contracts:
                exit_with_error(
                    args.outform, "input files do not contain any valid
contracts"
                )

            statespace =
analyzer.dump_statespace(contract=analyzer.contracts[0])

            try:
                with open(args.statespace_json, "w") as f:
                    json.dump(statespace, f)
            except Exception as e:
                exit_with_error(args.outform, "Error saving json: " +
str(e))

        else:
            try:
                report = analyzer.fire_lasers( # v
```

```
                    modules=(
                        [m.strip() for m in args.modules.strip().split(",")]
                        if args.modules # f
                        else None
                    ),
                    transaction_count=args.transaction_count,
                )

                outputs = {
                    "json": report.as_json(),
                    "jsonv2": report.as_swc_standard_format(),
                    "text": report.as_text(),
                    "markdown": report.as_markdown(),
                }
                print(outputs[args.outform])
                if len(report.issues) > 0:
                    exit(1)
                else:
                    exit(0)
            except DetectorNotFoundError as e:
                exit_with_error(args.outform, format(e))
            except CriticalError as e:
                exit_with_error (
                    args.outform, "Analysis error encountered: " + format(e)
                )

    else:
        parser.print_help()
```