TRAIL OF BITS

Slither: API walkthrough

Who am I?

- **Josselin Feist (@montyly)**

- **Trail of Bits: trailofbits.com**
    - We help developers build safer software
    - R&D focused: we use the latest program analysis techniques
    - Slither, Echidna, Tealer, Caracal, solc-select, ..

ToB Twitter list

Hackathon rule

- https://github.com/crytic/ethdam
- **Up to $2k for the best project(s)**
- **Themes**
  - UX/UI
    - Impression us with Slither-lsp + vscode
  - On-chain monitoring
    - Show the state variables evolution over time/block number
  - Machine learning
    - Build a RAG with langchain to do code understand / QA bot on solidity
- **Criteria**
  - Novelty
  - Reliance on Slither (the more the better)

Agenda

- What is Slither
- Slither internals & API
- SlithIR

Slides & hackathon details: https://github.com/crytic/ethdam

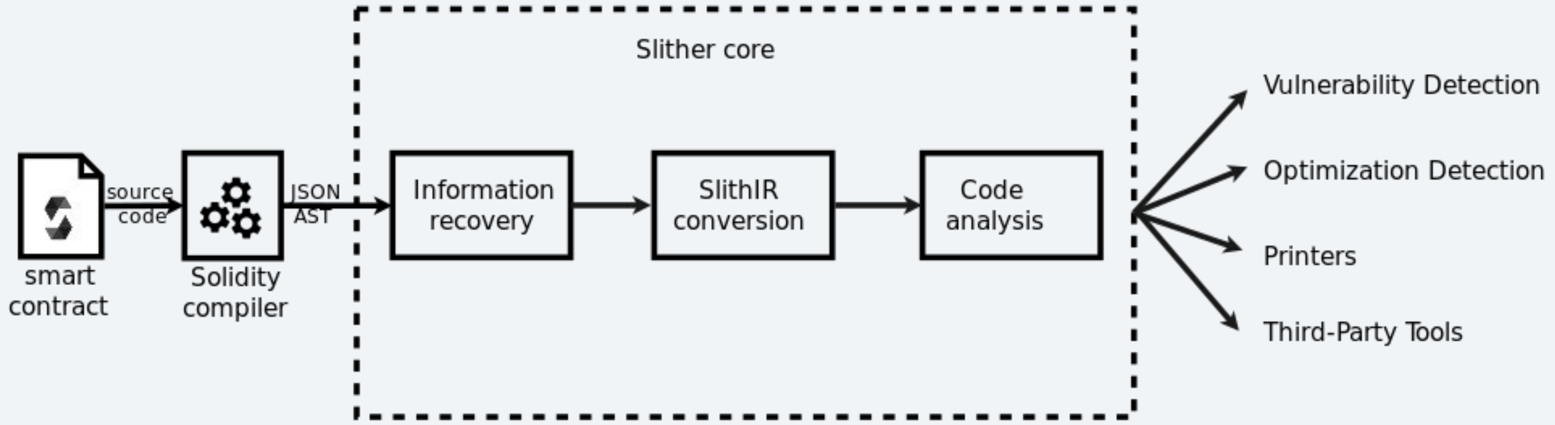
Slither

- **Static analysis framework for Solidity & Vyper**
  - Vulnerability detection
  - Optimization detection
  - Code understanding
  - Assisted code review

https://github.com/crytic/slither

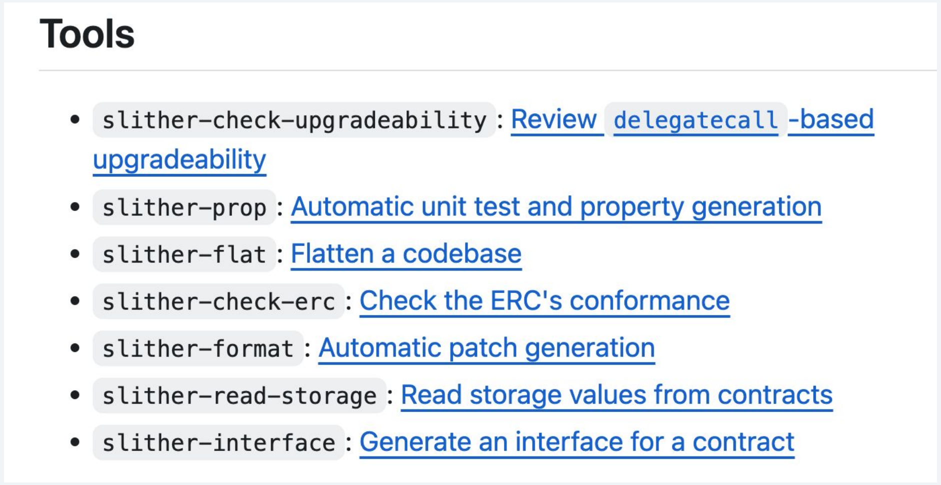
```
pip3 install -u slither-analyzer
```

Slither


source
code
JSON
AST
smart
contract
Solidity
compiler
Slither core
Information
recovery
SlithIR
conversion
Code
analysis
Vulnerability Detection
Optimization Detection
Printers
Third-Party Tools

Generic Static Analysis Framework

Assisted code review


Tools

- `slither-check-upgradeability`: Review `delegatecall`-based upgradeability
- `slither-prop`: Automatic unit test and property generation
- `slither-flat`: Flatten a codebase
- `slither-check-erc`: Check the ERC's conformance
- `slither-format`: Automatic patch generation
- `slither-read-storage`: Read storage values from contracts
- `slither-interface`: Generate an interface for a contract

Python API

- **Python API to help during a code review**
  - Inspect contract information
  - Including data dependency/taint analysis

Python API

- **Ex: What functions can modify a state variable:**

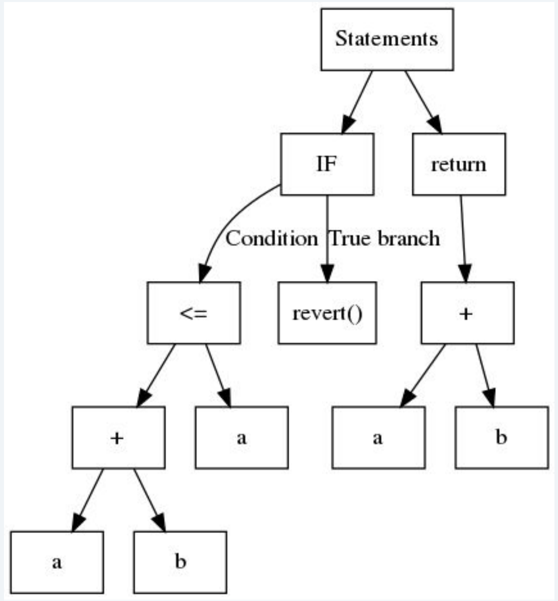
```
slither = Slither('function_writing.sol')
contract = slither.get_contract_from_name('Contract')[0]
var_a = contract.get_state_variable_from_name('a')

functions_writing_a = contract.get_functions_writing_variable(var_a)

print('The function writing "a" are {}'.format([f.name for f in functions_writing_a]))
```

Slither Internals

Slither Internals

- **Input: solc AST**

```
function safeAdd(uint256 a, uint256 b) ...
    if (a + b <= a) {
        revert();
    }
    return a + b;
}
```



Slither Layers

- **Compilation units**
  - ~ group of files used by one call to solc
- **Contracts**
  - Inheritance, state variables, functions
- **Functions**
  - Attributes, CFG
- **Control Flow Graphs**
  - Nodes
- **Expression & IR**
  - Operations

Slither object

```
from slither import Slither

# Create the slither object

sl = Slither("test.sol")


# Works with the other supported target, ie :

sl = Slither("0xdac17f958d2ee523a2206206994597c13d831ec7") # Load USDT

# Add etherscan_api_key for rate limit

sl = Slither("0xdac17f958d2ee523a2206206994597c13d831ec7", etherscan_api_key="..")
```

Compilation unit

- ~ group of files used by one call to solc
- Most targets have 1 compilation, but not always true
  - Partial compilation for optimization
  - Multiple solc version used
  - Etc..

```
sl = Slither("test.sol")

sl.compilation_units # array of SlitherCompilationUnit
```

Compilation unit

- **Why compilation unit matters?**
    - Some APIs might be not intuitive
    - Ex: looking for a contract based on the name?
        - Can have multiple contracts
- **For hacking you can *(probably)* use the first compilation unit**
    - compilation_unit = sl.compilation_units[0]

Compilation unit - cheatsheet

- slither/core/compilation_unit.py
- **contracts: List[Contract]**
  - List of all the contracts
- **contracts_derived(self): List[Contract]**
  - List of the most derived contracts. I.e. contract not inherited
- **get_contract_from_name(contract_name): List[Contract]**
  - *Usually: returns one contract*
- **Top level objects**
  - [structures | enums | events | variables | functions]_top_level

Compilation unit - example

```
from slither import Slither

sl = Slither("0xdac17f958d2ee523a2206206994597c13d831ec7")

compilation_unit = sl.compilation_units[0]


# Print all the contracts from the USDT address

print([str(c) for c in compilation_unit.contracts])


# Print the most derived contracts from the USDT address

print([str(c) for c in compilation_unit.contracts_derived])
```

Compilation unit - example

```
% python test.py

['SafeMath', 'Ownable', 'ERC20Basic', 'ERC20', 'BasicToken', 'StandardToken', 'Pausable',
'BlackList', 'UpgradedStandardToken', 'TetherToken']


['SafeMath', 'UpgradedStandardToken', 'TetherToken']
```

Contract - cheatsheet

- slither/core/declarations/contract.py
- **name: str**
- **Inheritance**
  - inheritance: List[Contract]: c3 linearization order
  - derived_contracts: List[Contract]: contracts derived from it
- **General objects**
  - enums | events | structures
- **Variables**
  - state_variables: List[StateVariable]: list of accessible variables
  - state_variables_ordered: List[StateVariable]: all variable ordered by declaration
- **get_function_from_signature(sig)**

Contract - example

```
from slither import Slither

sl = Slither("0xdac17f958d2ee523a2206206994597c13d831ec7")

compilation_unit = sl.compilation_units[0]
```

```
# Print all the state variables of the USDT token

contract = compilation_unit.get_contract_from_name("TetherToken")[0]

print([str(v) for v in contract.state_variables])
```

Contract - example

```
% python test.py

['owner', 'paused', '_totalSupply', 'balances', 'basisPointsRate', 'maximumFee',
'allowed', 'MAX_UINT', 'isBlackListed', 'name', 'symbol', 'decimals', 'upgradedAddress',
'deprecated']
```

Function - cheatsheet

- core/declarations/function.py
- **solidity_signature: str**
- **entry_point: Node**
- **Elements**
  - expressions, variables, nodes, modifiers
- **Operations**
  - [state |local]_variable_[read |write]
  - All can be prefixed by “all_” for recursive lookup
    - Ex: all_state_variable_read: return all the state variables read in internal calls
  - slithir_operations

Function - example 1

```
from slither import Slither
sl = Slither("0xdac17f958d2ee523a2206206994597c13d831ec7")
compilation_unit = sl.compilation_units[0]
contract = compilation_unit.get_contract_from_name("TetherToken")[0]
```

```
# Print all the state variables read by the totalSupply function
totalSupply = contract.get_function_from_signature("totalSupply()")
print([str(v) for v in totalSupply.state_variables_read])
```

Function - example 1

```
% python test.py
['_totalSupply', 'deprecated', 'upgradedAddress']
```

Function - example 2

```
[..]

transfer = contract.get_function_from_signature("transfer(address,uint256)")

# Print all the state variables read by the transfer function
print([str(v) for v in transfer.state_variables_read])
# Print all the state variables read by the transfer function and its internal calls
print([str(v) for v in transfer.all_state_variables_read])
```

Function - example 2

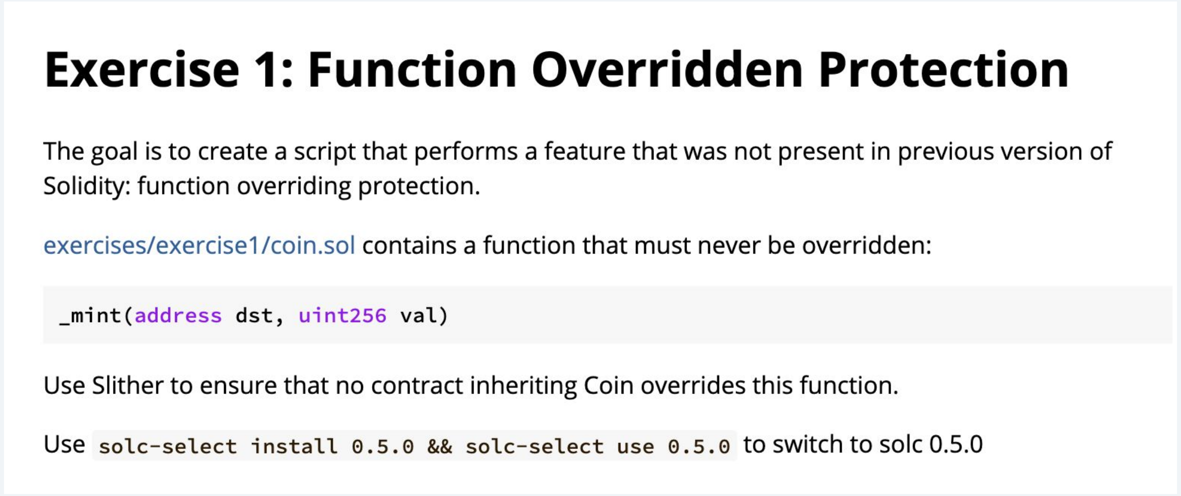
```
% python test.py

['deprecated', 'isBlackListed', 'upgradedAddress']

['owner', 'basisPointsRate', 'deprecated', 'paused', 'isBlackListed', 'maximumFee', 'upgradedAddress', 'balances']
```

```
function transfer(address _to, uint _value) public whenNotPaused {
    require(!isBlackListed[msg.sender]);
    if (deprecated) {
        return UpgradedStandardToken(upgradedAddress).transferByLegacy(msg.sender, _to, _value);
    } else {
        return super.transfer(_to, _value);
    }
}
```

Exercises

Exercise 1

- https://secure-contracts.com/program-analysis/slither/exercise1.html



Exercise 1: Function Overridden Protection

The goal is to create a script that performs a feature that was not present in previous version of Solidity: function overriding protection.

exercises/exercise1/coin.sol contains a function that must never be overridden:

```
_mint(address dst, uint256 val)
```

Use Slither to ensure that no contract inheriting Coin overrides this function.

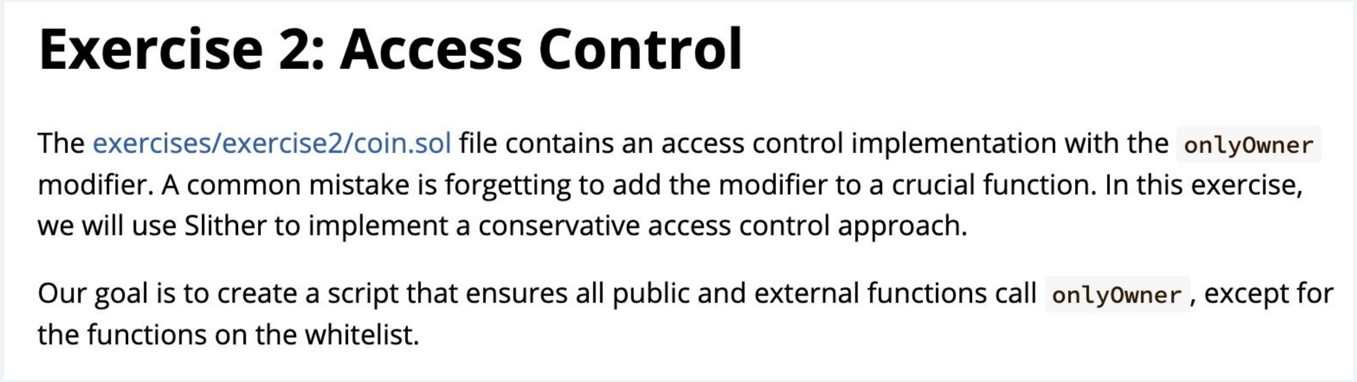
Use `solc-select install 0.5.0 && solc-select use 0.5.0` to switch to solc 0.5.0

Exercise 1

```
# Iterate over all the contracts
for contract in slither.contracts:
   # If the contract is derived from MyContract
   if coin in contract.inheritance:
      # Get the function definition
      mint = contract.get_function_from_signature('_mint(address,uint256)')
      # If the function was not declared by coin, there is a bug !
      # Detect error only for contracts overriding the '_mint' function
      if mint.contract_declarer == contract:
           print(f'Error, {contract} overrides {mint}')
```

Exercise 2

- https://secure-contracts.com/program-analysis/slither/exercise2.html


Exercise 2: Access Control

The exercises/exercise2/coin.sol file contains an access control implementation with the `onlyOwner` modifier. A common mistake is forgetting to add the modifier to a crucial function. In this exercise, we will use Slither to implement a conservative access control approach.

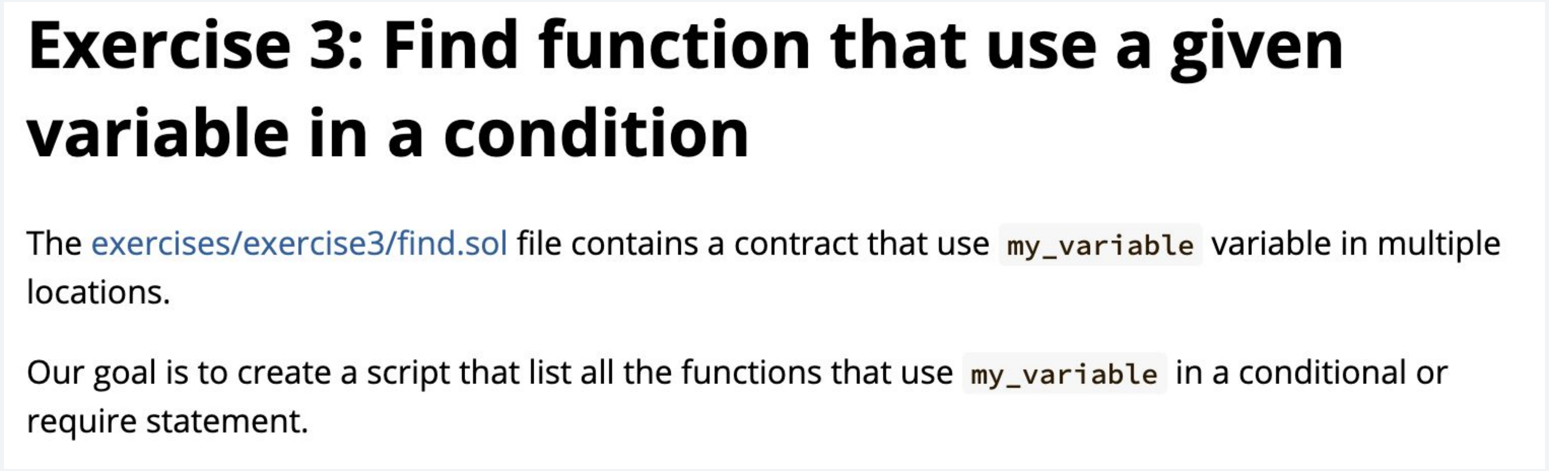
Our goal is to create a script that ensures all public and external functions call `onlyOwner`, except for the functions on the whitelist.

Exercise 2

```python
slither = Slither('coin.sol')

whitelist = ['balanceOf(address)']

for contract in slither.contracts:
    for function in contract.functions:
        if function.full_name in whitelist:
            continue
        if function.is_constructor:
            continue
        if function.visibility in ['public', 'external']:
            if not 'onlyOwner()' in [m.full_name for m in function.modifiers]:
                print(f'{function.full_name} is unprotected!')
```

Exercise 3

- https://secure-contracts.com/program-analysis/slither/exercise3.html



Exercise 3: Find function that use a given variable in a condition

The exercises/exercise3/find.sol file contains a contract that use `my_variable` variable in multiple locations.

Our goal is to create a script that list all the functions that use `my_variable` in a conditional or require statement.

Exercise 3

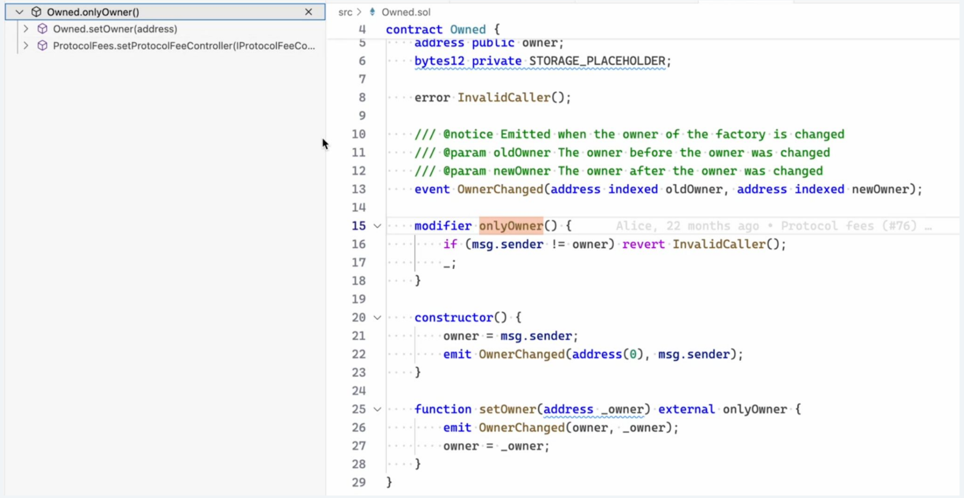
```python
slither = Slither('find.sol')
find = slither.get_contract_from_name('Find')[0]

assert find

# Get the variable
my_variable = find.get_state_variable_from_name("my_variable")
assert my_variable


function_using_a_as_condition = [
    f
    for f in find.functions
    if f.is_reading_in_conditional_node(my_variable) or f.is_reading_in_require_or_assert(my_variable)
]

# Print the result
print(f'The function using "a" in condition are {[f.name for f in function_using_a_as_condition]}')
```

contract-explorer

contract-explorer

- **New VScode plugin that leverages slither**
  - Go to implementations/definitions
  - Find all references
  - Show call / type hierarchy
  - ...
- **Backend: slither-lsp**
  - Full access to slither’s power

contract-explorer



Owned.onlyOwner()
Owned.setOwner(address)
ProtocolFees.setProtocolFeeController(IProtocolFeeCo...
src > Owned.sol
Alice, 22 months ago • Protocol fees (#76) ...

```solidity
contract Owned {
    address public owner;
    bytes12 private STORAGE_PLACEHOLDER;

    error InvalidCaller();

    /// @notice Emitted when the owner of the factory is changed
    /// @param oldOwner The owner before the owner was changed
    /// @param newOwner The owner after the owner was changed
    event OwnerChanged(address indexed oldOwner, address indexed newOwner);

    modifier onlyOwner() {
        if (msg.sender != owner) revert InvalidCaller();
        _;
    }

    constructor() {
        owner = msg.sender;
        emit OwnerChanged(address(0), msg.sender);
    }

    function setOwner(address _owner) external onlyOwner {
        emit OwnerChanged(owner, _owner);
        owner = _owner;
    }
}
```

contract-explorer

- **Easy to hack on top of it**
    - See slither-lsp’s README to add a new command
        - https://github.com/crytic/slither-lsp
    - See contract-explorer’s DEV.md page
        - https://github.com/crytic/contract-explorer/blob/master/DEV.md

Where to start

Where to start

- ***secure-contracts.com***
  - *program-analysis/slither*
  - *Base API + exercises*
- ***Demo project***
  - *git clone git@github.com:crytic/slither.git*
  - cd slither/tools/demo/
    - Default folder with argument parsing + Slither object creation
- **Read slither/detectors code**
  - APIs' usage

Conclusion

Slither

- **Open source framework to build custom analysis**

- **Hackathon: https://github.com/crytic/ethdam**
  - $2k to win
- **Need help?**
  - EthDam Discord (*#trail-of-bits*) (*@josselin_trailofbits*)
  - Github (issues , discussions)
  - Slack (https://slack.empirehacking.nyc/, #ethereum)