Slither Training

Presenter: Josselin Feist, josselin@trailofbits.com

Requirements:

- Basic Python knowledge
- git clone https://github.com/trailofbits/trufflecon-2019

The goal of this document is to see a basic introduction to Slither API through the creation of two utilities that extend Solidity security.

Need help? Slack: https://empireslacking.herokuapp.com/ #ethereum

Installation

Slither requires >= python 3.6. It can be installed through pip or using docker.

Slither through pip

```
pip install --user slither-analyzer
```

Slither through docker

```
docker pull trailofbits/eth-security-toolbox
docker run -it -v "$PWD":/home/trufflecon trailofbits/eth-security-toolbox
```

The last command runs eth-security-toolbox in a docker that has access to your current directory. You can change the files from your host, and run the tools on the files from the docker

Inside docker, run:

```
solc-select 0.5.3
cd /home/trufflecon/
```

To run a python script with python 3:

```
python3 script.py
```

Slither: API Basics

The following details how to manipulate a smart contract through the Slither API:

- How to explore the contracts of a project
- How to explore the functions of a contract
- The basic attributes of the contracts and the functions

Loading a project

The first thing to do on a script is to initiate a new blockchain:

```
from slither import Slither
slither = Slither('coin.sol')
```

Exploring contracts and functions

A Slither object has:

- `contracts (list(Contract)`: list of contracts
- `contracts_derived (list(Contract)`: list of contracts that are not inherited by another contract (subset of contracts)
- `get_contract_from_name (str)`: Return a contract from its name

A Contract object has:

- `name (str)`: Name of the contract
- `functions (list(Function))`: List of functions
- `modifiers (list(Modifier))`: List of functions
- `all_functions_called (list(Function/Modifier))`: List of all the internal functions reachable by the contract
- `inheritance (list(Contract))`: List of inherited contracts
- `get_function_from_signature (str)`: Return a Function from its signature
- `get_modifier_from_signature (str)`: Return a Modifier from its signature

- `get_state_variable_from_name (str)`: Return a StateVariable from its name

A Function or a Modifier object has:

- `name (str)`: Name of the function
- `contract (contract)`: the contract where the function is declared
- `nodes (list(Node))`: List of the nodes composing the CFG of the function/modifier
- `entry_point (Node)`: Entry point of the CFG
- `variables_read (list(Variable))`: List of variables read
- `variables_written (list(Variable))`: List of variables written
- `state_variables_read (list(StateVariable))`: List of state variables read (subset of variables`read)
- `state_variables_written (list(StateVariable))`: List of state variables written (subset of variables`written)

Example: Print Basic Information

print_basic_information.py shows how to print basic information about a project.

Exercice 1 : Function overridden protection

The goal is to create a script that fills a missing feature of Solidity: function overriding protection.

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

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

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

Proposed algorithm

```
Get the coin contract
For each contract of the project:
      If Coin is in the list of inherted contract:
            Get the mint function
            If the contract declaring the mint function is != Coin:
                  A bug is found.
```

Hints:

- To get a specific contract, use `slither.get_contract_from_name`
- To get a specific function, use `contract.get_function_from_signature`

Exercice 2 (Bonus) : onlyOwner whitelist

slither/exercises/exercise2/coin.sol possess an access control with the onlyOwner modifier. A frequent mistake is to forget to add the modifier to a critical function. We are going to see how to implement a conservative access control approach with Slither.

The goal is to create a script that will ensure that all the public and external function calls onlyOwner, except for the functions whitelisted.

Proposed algorithm

```
Create a whitelist of signatures
Explore all the functions
      If the function is in the whitelist of signatures:
            Skip
      If the function is public or external:
            If onlyOwner is not in the modifiers:
                  A bug is found
```