By: Troy Sargent

Slither: a Vyper and Solidity static analyzer

Background

Security Engineer at Trail of Bits:

- Work on smart contracts, blockchain nodes, rollups, VMs
- Core contributor to Slither
- @0xalpharush on Twitter/Github

Trail of Bits:

- Combine manual review with practical program analysis
- Apply fuzzing and static analysis to Golang, Rust, Cairo, Solana, etc

Slither

- **Static analysis framework for smart contracts**
  - Vulnerability detection
  - Optimization detection
  - Assisted code review

https://github.com/crytic/slither

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

Slither now supports Vyper!

- Vyper is a pythonic smart contract language
- Initial support for Vyper 0.3.7 (Aug. 2023)
  - Worked with Vyper Foundation to support 3 codebases (Yearn, Curve, and Lido)
- Very little changes required for the 90+ existing detectors
- Bonus: Vyper can be fuzzed with Echidna/ Medusa

Agenda

- Find bugs and explore Vyper codebases
- 2 tips to use Slither effectively

Not in this talk:

- Lowering Vyper to Slither’s intermediate representation

Finding Vulnerabilities and Understanding Code

Vulnerability Detectors (reentrancy)

```
balances: HashMap[address, uint256]

@external
def withdraw():
    raw_call(msg.sender, b"", value=self.balances[msg.sender])
    self.balances[msg.sender] = 0
```

```
(env) alpharush@macbook slither % slither test.vy --detect reentrancy-eth
INFO:Detectors:
Reentrancy in test.withdraw() (test.vy#4-6):
        External calls:
        - msg.sender.raw_call{value: self.balances[msg.sender]}(0x) (test.vy#5)
        State variables written after the call(s):
        - self.balances[msg.sender] = 0 (test.vy#6)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities
INFO:Slither:test.vy analyzed (1 contracts with 1 detectors), 1 result(s) found
```

What Slither offers

Understand code

```
@payable
@external
@nonreentrant('lock')
def add_liquidity(amounts: uint256[N_COINS], min_mint_amount: uint256) -> uint256:
    assert not self.is_killed  # dev: is killed
    amp: uint256 = self._A()
    old_balances: uint256[N_COINS] = self._balances(msg.value)
    D0: uint256 = self.get_D(old_balances, amp)

    lp_token: address = self.lp_token
    token_supply: uint256 = ERC20(lp_token).totalSupply()
    new_balances: uint256[N_COINS] = old_balances
    for i in range(N_COINS):
        if token_supply == 0:
            assert amounts[i] > 0
        new_balances[i] += amounts[i]

    D1: uint256 = self.get_D(new_balances, amp)
    assert D1 > D0

    fees: uint256[N_COINS] = empty(uint256[N_COINS])
    mint_amount: uint256 = 0
    D2: uint256 = 0
    if token_supply > 0:
```

Understand code (continued)

- **Slither's printers provide quick insights into functions and contracts with out-of-the-box analyses**
- **For example, the "vars-and-auth" printer will show:**
  - What state variables each function updates
  - Uses of msg.sender (e.g. is the sender the owner?)

```
Function: add_liquidity(uint256[2],uint256) returns(uint256)
        Sends ETH: no
        State variables written:
                -admin_balances
        Conditions on msg.sender:
                -assert ERC20(self.coins[1]).transferFrom(msg.sender, self, amounts[1])
```

Code comprehension (continued)

How can we make this content digestible for large codebases?

Command line magic:

```
slither steth.vy --print vars-and-auth | awk -v RS= -v ORS='\n\n' '/-is_killed/'
```

```
Function: kill_me() returns()
        Sends ETH: no
        State variables written:
                -is_killed
        Conditions on msg.sender:
                -assert msg.sender == self.owner

Function: unkill_me() returns()
        Sends ETH: no
        State variables written:
                -is_killed
        Conditions on msg.sender:
                -assert msg.sender == self.owner
```

2 Tips to Effectively Use Slither

Tip 1: Run specific detectors

```
@view
@internal
def _convertToShares(assetAmount: uint256) -> uint256:
    totalSupply: uint256 = self.totalSupply
    totalAssets: uint256 = self.asset.balanceOf(self)
    if totalAssets == 0 or totalSupply == 0:
        return assetAmount  # 1:1 price

    # NOTE: `assetAmount = 0` is extremely rare case, not optimizing for it
    return assetAmount * totalSupply / totalAssets
```

```
(env) alpharush@macbook slither % slither test.vy --detect divide-before-multiply
INFO:Detectors:
test._convertToShares() (test.vy#119-126) performs a multiplication on the result of a division:
        - assetAmount / totalSupply * totalAssets (test.vy#126)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply
```

The best use of static analysis results is to identify concerns and see if relevant queries surface anything

Tip 2: Sarif

What is a productive way to triage static analysis results?

- **Use sarif (standard file format) with powerful editor integrations**
  - IDE diagnostics (Microsoft sarif viewer VSCode extension)
  - Marking whether finding is valid
  - Note taking
  - Share triage results with collaborators

Usage:

```
slither test.vy --sarif test.sarif
```

Tip 2: Sarif (continued)

```
@external
def withdraw():
    raw_call(msg.sender, b"", value=self.balances[msg.sender])
```

```
Reentrancy in test.withdraw() (test.vy#4-6):
        External calls:
        - msg.sender.raw_call{value: self.balances[msg.sender]}(0x) (test.vy#5)
        State variables written after the call(s):
        - self.balances[msg.sender] = 0 (test.vy#6)
```

The best medium to review static analysis results is in your editor with context

Tip 3: Github Action

```
 7
 8      function withdraw() external{
 9          msg.sender.call{value: balances[msg.sender]}("");
10           balances[msg.sender] = 0;
11      }
```

```
Reentrancy in Test.withdraw():
External calls:
- msg.sender.call{value: balances[msg.sender]}()
State variables written after the call(s):
- balances[msg.sender] = 0
```

Slither

```
12
13  }
```

Tool	Rule ID
Slither	0-1-reentrancy-eth

Apply the `check-effects-interactions pattern`.

- **The best time to review a static analysis result is when the code is fresh in your mind (slither-action)**

Future Work

- **Fix and test any gaps in the initial language support**
  - Vyper has a greater number of builtins (also more complex)
- **Add support for newer Vyper as the language evolves**
  - 0.4 module system
- **Pursue upstream improvements in the semantic info contained within Vyper's AST**
  - Referenced declarations (could also benefit language server implementations)
- **Write Vyper-specific detectors**
  - Side effects in lazily evaluated contract like (x in [f(), g()]).

Future Work

-
- **Improve support for Solidity and Vyper interoperability**
  - Retrieve AST from frameworks like Ape (pending fixes upstream)
  - At each callsite to an interface, instantiate candidates and gather information from their source code (cross-contract analysis)
- **Increase effectiveness for developers and researchers**
  - Each potential vulnerability should suggest how to triage/ drive the decision
  - Documentation and tutorials on writing detectors

Reach out

- **Have questions or ideas?**
  - @0xalpharush / troy.sargent@trailofbits.com
- **Find these slides:**
  https://github.com/trailofbits/publications#blockchain