SlithIR, An Intermediate Representation of Solidity to enable High Precision Security Analysis

RunEVM 2019

Who am I?

- Josselin Feist (josselin@trailofbits.com, @Montyly)
- Trail of Bits: trailofbits.com
  - We help organizations build safer software
  - R&D focused: we use the latest program analysis techniques
    - https://github.com/trailofbits/manticore
    - https://github.com/cyric/echidna/
    - https://github.com/cyric/evm_cfg_builder

Plan

- What is Slither
- What is SlithIR
- Conclusion and roadmap

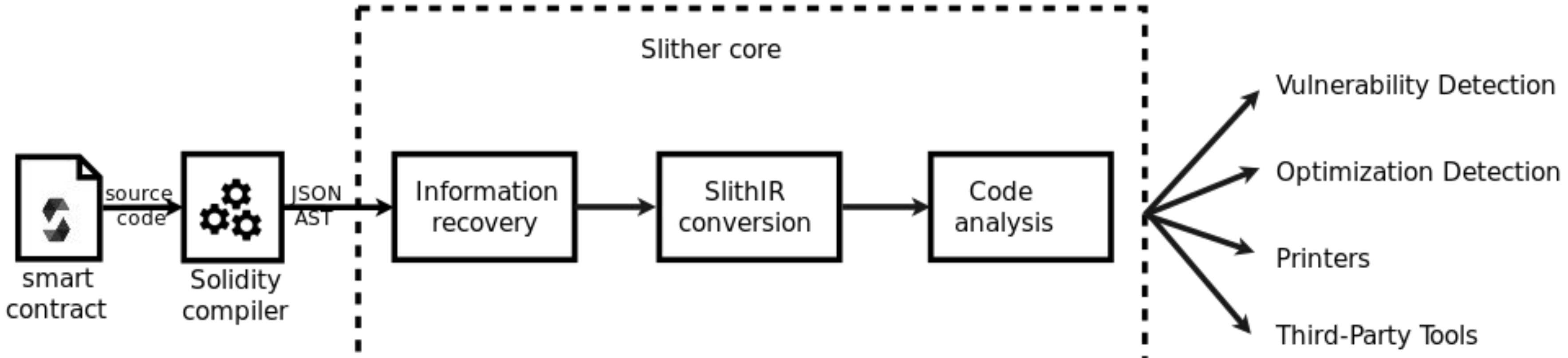
Slither

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

https://github.com/crytic/slither

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

Slither



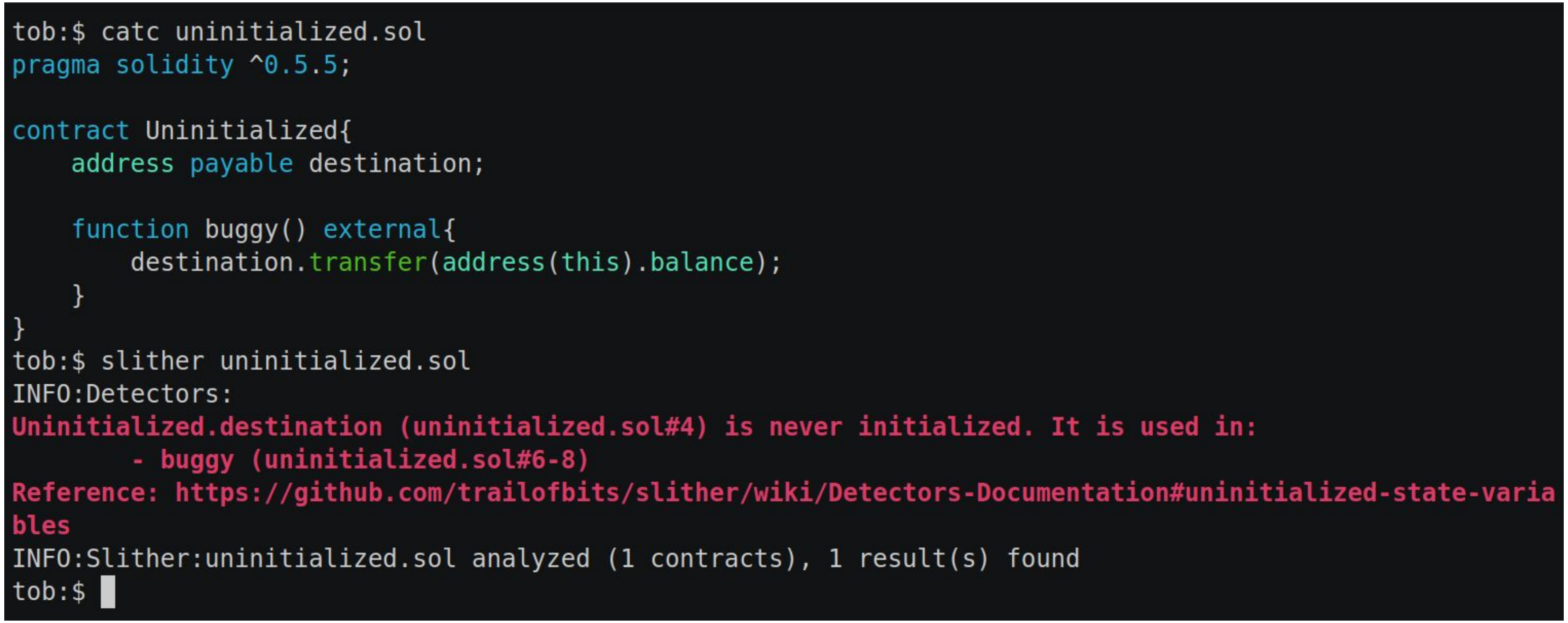
smart contract
source code
Solidity compiler
JSON AST
Slither core
Information recovery
SlithIR conversion
Code analysis
Vulnerability Detection
Optimization Detection
Printers
Third-Party Tools

Detectors

Detectors

- **~30 public vulnerability detectors**
- **From critical issues:**
  - Reentrancy
  - Shadowing
  - Uninitialized variables
  - …
- **To optimization issues**
  - Variables that should be constant
  - Functions that should be external
  - …
- **Private detectors with more complex patterns**

Vulnerability Detection



```
tob:$ catc uninitialized.sol
pragma solidity ^0.5.5;

contract Uninitialized{
    address payable destination;

    function buggy() external{
        destination.transfer(address(this).balance);
    }
}
tob:$ slither uninitialized.sol
INFO:Detectors:
Uninitialized.destination (uninitialized.sol#4) is never initialized. It is used in:
        - buggy (uninitialized.sol#6-8)
Reference: https://github.com/trailofbits/slither/wiki/Detectors-Documentation#uninitialized-state-varia
bles
INFO:Slither:uninitialized.sol analyzed (1 contracts), 1 result(s) found
tob:$
```

https://asciinema.org/a/eYrdWBvasHXelpDob4BsNi6Qg

Vulnerability Detection

- Fast (1-2 seconds)
- No configuration
- Low # false alarms
- Easy integration into CI (Truffle/Embark/...)

Generic Static Analysis Framework

Code Understanding

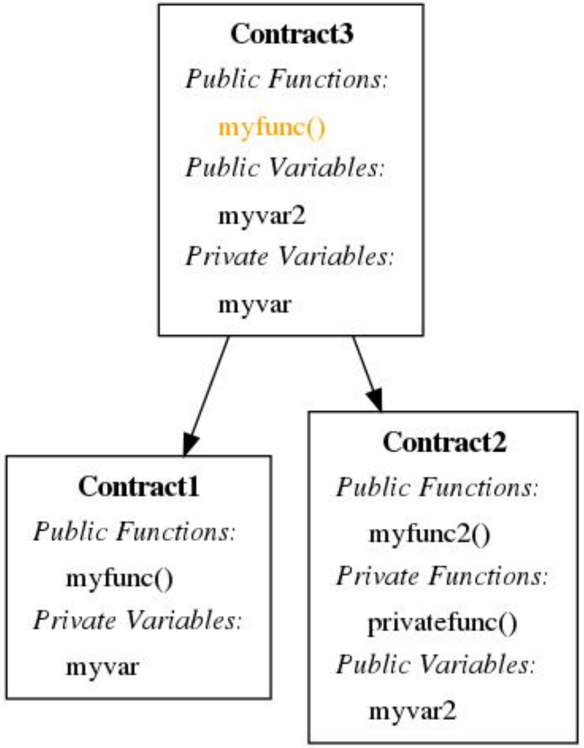
- **Printers: visual representations**
- **Examples:**
  - Graph-based representations (inheritance graph, CFG, call-graph)
  - Read/Write/Call summary
  - Access control summary
  - Human-readable summary (code complexity, minting restrictions, ..)
- **https://github.com/crytic/slither/#printers**

Printers: Inheritance Graph

```
contract Contract1{
    uint myvar;
    function myfunc() public{}
}

contract Contract2{
    uint public myvar2;

    function myfunc2() public{}
    function privatefunc() private{}
}
contract Contract3 is Contract1, Contract2{
    function myfunc() public{} // override myfunc
}
```



Assisted Code Review

- **Library for tooling**
    - slither-check-upgradability: Help to review delegatecall proxy contract
    - slither-find-paths: Find all the paths that can reach a given function

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

Assisted Code Review

Ex: What functions can modify a state variable:

```
slither = Slither('function_writing.sol')
contract = slither.get_contract_from_name('Contract')
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 Engine

- Designed as a compiler
- Input: solc AST
- Use refinement parsing (joern)
    - Parse through multiple stages/layers

Slither Layers

- **Contracts**
  - Inheritance, state variables, functions
- **Functions**
  - Attributes, CFG
- **Control Flow Graphs**
  - Nodes
- **Nodes**
  - Expressions as AST -> SlithIR

SlithIR

- **Slither Intermediate Representation**
    - Solidity -> Human usage
    - SlithIR -> Code analysis usage

SlithIR

- Less than 40 instructions
- Linear IR (no jump)
- Based on Slither CFG
- Flat IR
- Code transformation/simplification
    - Ex: removal of ternary operator

SlithIR Instructions

- Binary/Unary
  - `LVALUE = RVALUE + RVALUE`
  - `LVALUE = ! RVALUE`
  - `...`
- Index
  - `REFERENCE -> LVALUE [ RVALUE ]`

SlithIR Instructions

- Member
  - `REFERENCE -> LVALUE . RVALUE`
- New
  - `LVALUE =` **`NEW_ARRAY`** `ARRAY_TYPE DEPTH`
  - `LVALUE =` **`NEW_CONTRACT`** `CONSTANT`
  - `LVALUE =` **`NEW_STRUCTURE`** `STRUCTURE`

note: no new_structure operator in Solidity

SlithIR Instructions

Expression: allowance[_from][msg.sender] -= _value

IRs:

```
REF_1 -> allowance[_from]
REF_2 -> REF_1[msg.sender]
REF_2 -= _value
```

SlithIR SSA

- **SSA (Static Single Assignment) form**
  - A variable is assigned only one time
  - Needed for precise data dependency analysis
  - Usually, ϕ indicates multiple definitions of a variable

```
a = 0
if(){
    a = b;
}

a = a + 1;
```

```
a_0 = 0
if(){
    a_1 = b_0;
}

a_2 = ϕ(a_0, a_1)
a_3 = a_2 + 1;
```

SlithIR SSA

- **SlithIR SSA features**
  - Include:
    - State variables
    - Alias analysis on *storage* reference pointers
  - Inter-procedural
    - Track internal calls
  - Inter-transactional
    - Take in consideration the state-machine aspect of smart contracts

SSA Inter-Transactional Example

```
uint my_var_A;

uint my_var_B;

function direct_set(uint i) public {
    my_var_A = i;
}


function indirect_set() public {
     my_var_B = my_var_A;
}
```

SSA Inter-Transactional Example

```
uint my_var_A;

uint my_var_B;

function direct_set(uint i) public {
    my_var_A = i;
}


function indirect_set() public {
     my_var_B = my_var_A;
}
```

SSA within one transaction context

```
my_var_A_0;
my_var_B_0;
```

SSA Inter-Transactional Example

```
uint my_var_A;

uint my_var_B;

function direct_set(uint i) public {
    my_var_A = i;
}


function indirect_set() public {
     my_var_B = my_var_A;
}
```

SSA within one transaction context

```
my_var_A_0;
my_var_B_0;


direct_set(uint i):

    my_var_A_1 := i_0
```

SSA Inter-Transactional Example

```
uint my_var_A;

uint my_var_B;

function direct_set(uint i) public {
    my_var_A = i;
}


function indirect_set() public {
    my_var_B = my_var_A;
}
```

SSA within one transaction context

```
my_var_A_0;
my_var_B_0;


direct_set(uint i):

    my_var_A_1 := i_0



indirect_set():

    my_var_B_1 := my_var_A_0
```

SSA Inter-Transactional Example

```
uint my_var_A;

uint my_var_B;

function direct_set(uint i) public {
    my_var_A = i;
}


function indirect_set() public {
     my_var_B = my_var_A;
}
```

SSA within one transaction context

```
my_var_A_0;
my_var_B_0;


direct_set(uint i):

    my_var_A_1 := i_0



indirect_set():

    my_var_B_1 := my_var_A_0
```

- Lack of precision, we don't know that my_var_B can be controlled by i

SSA Inter-Transactional

- **Inter-Transactional SSA**
  - Entry point of function: add phi operators
  - Late binding of the phi parameters
  - Represent a fix-point over all the potential transactions

SSA Inter-Transactional Example

```
uint my_var_A;

uint my_var_B;

function direct_set(uint i) public {
    my_var_A = i;
}


function indirect_set() public {
    my_var_B = my_var_A;
}
```

```
my_var_A_0;
my_var_B_0;


direct_set(uint i):

    my_var_A_1 := i_0


indirect_set():

    my_var_A_2 := ϕ(my_var_A_1,
                    my_var_A_0)

    my_var_B_1 := my_var_A_2
```

Data dependency

```
uint my_var_A;

uint my_var_B;

function direct_set(uint i) public {
    my_var_A = i;
}


function indirect_set() public {
     my_var_B = my_var_A;
}
```

Dependencies:

- `my_var_A depends on i`
- `my_var_B depends on my_var_A, i`

SlithIR: Code Analysis

- **Data dependency**
  - Pre-computed, free for analyses
  - Level: function/contract
- **Read/Write of variables**
  - Level: node/function/contract
- **Protected functions**
  - What functions need ownership?

Conclusion

Conclusion

- **Vulnerability and optimization detection**
  - Fast and precise
  - No configuration
  - CI support
- **Code review**
  - In-depth information about the codebase through Printers and API
- **A foundation for research**
  - Generic library for static analysis

Slither Roadmap

- More detectors!
- Improve developer integration
  - Visual Studio plugin (90)
  - slither-format: automatic patching (150)
- New language support
  - Vyper (39)

SlithIR Roadmap

- **SSA improvements**
- **Formal semantics**
    - SlithIR -> K
    - Symbolic Computation/Symbolic Execution/Abstract Interpretation
- **Code generation**
    - SlithIR -> LLVM
    - SlithIR -> YUL/EVM

Slither

- **https://github.com/crytic/slither**

- **Crytic: SaaS to ensure safe contracts**
  - https://crytic.io/
  - Includes Slither private detectors and formal verification
  - For more information: Dan Guido (dan@trailofbits.com)

- **Need Help?**
  - Slack: https://empireslacking.herokuapp.com (#ethereum)
  - Office Hours: free 1-hour consultation on Hangouts every two weeks