PAGE NO.
DATE / /

Topic : .....

→ Better than fuzzing Sometimes

F.C.

(1) Formal Verification

↳ Mathematically prove or disprove a situation can occur to break an invariant of our protocol.

Vyper & Solidity → High level lang
Yul & Huf → Low level Lang so more gas efficient

1) Solidity: Frees Memory Pointer.
2) ~~Vpr~~ Vyper & Solidity: Checks on eth & data size
3) Slight differences in opcode use

opcodes = Machine readable code

Cast to-base 0x60 dec

~~Huf~~ Function Dispatching

contract eg if you deploy or call function we get output like this in remix.

0xe026c0170000...

↳ called call data.

In every function in Solidity has signature - unique identifier formed

Never Stop Lea

Slither - Smart Contract Static Analysis framework

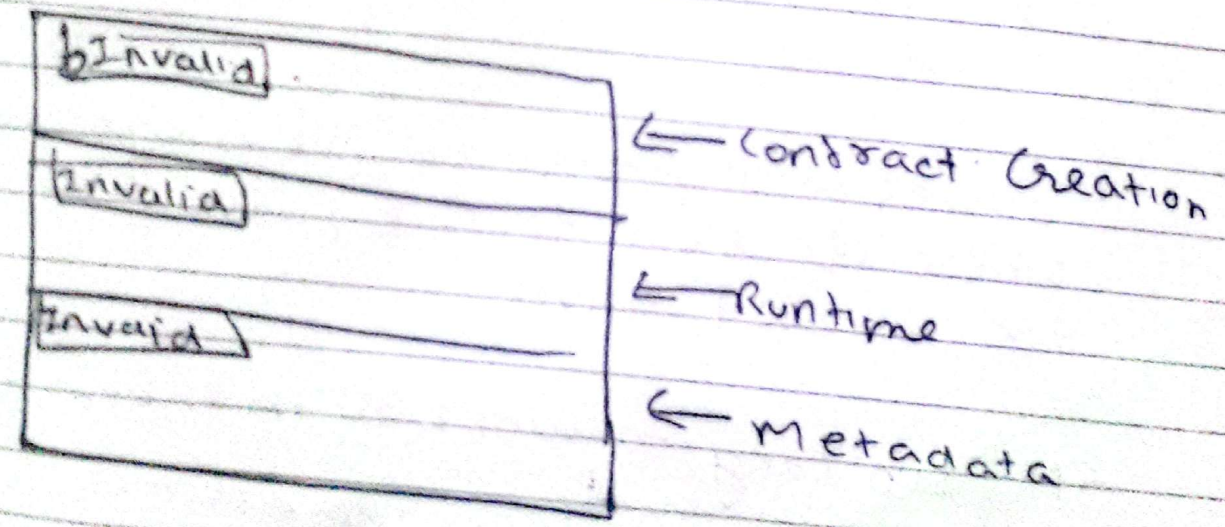
Topic :

PAGE NO. DATE 10/8/K

by hashing its name & input types. The first 4 bytes of the calldata correspond to function selector. So when you call function, Remix sends a selector 0xe026c017 at start of calldata. Like GOT table there is dispatch table or algo used to determine which function/commands to run in response to message.

1. Your calldata is sent to smart contract.
2. Our smart contract sees function selector in first 4 bytes
3. Dispatcher uses selector to route call to correct function
4. Function executes based on the calldata.

Solidity opcode format



Symbolic execution is a technique for formal verification Eg. $y = mx + b$, $y = mx/b$ creates diff versions for each path.

Fuzzing catchs 99% of issues.

Topic : .....

PAGE NO. 4
DATE / /

Symbolic Executor

```
func f(uint256 a) public returns (uint256){
        a = a+1;
        return a;
    }
}
```

~~if we~~

~~1st~~

1st path
↳ If we give max size of uint256 & we try to add 1 to it, function will revert.

2nd
↳ If we give any other numbers the function will add 1 to it & normaly return.

Solver will figure out for us expression where its both true & false like SMT or SAT solver.

SMTLib Lang to work with these solvers. Like Boolean Expression.

Topic :

23 = High performance Satisfiability Modulo Theories (SMT) solver ~~using~~ (Theorem Prover).

PAGE NO.

DATE

Code → ① Conver code 2 math ② Run through solver → Satisfy / → unsatisfy.

① Explore paths
② Convert paths to set of booleans.
③ See if path are reachable

using Solidity compiler (Builtin SMT checker).

```
solc --model-checker-engine chc
--model-checker-targets overflow
Smallsol.sol.
```

Property will matter.

Combo: SMT + fuzzer

① Stateless fuzz

② Statefull fuzz

③ Statefull fuzz with conditon (handle)

④ Formal verification.
↳ Tool → Certora → uses lang its own ↓ CVL

Topic : .....

PAGE NO.
DATE / /

Halmos (Symbolic Testing)

↳ F.V Suite

↳ pip install halmos

Complex steps to small → Modularization

(Modular Verification)

Path explosion problem

↳ If too many paths to solve the problem.