Smart Contract Bug Hunting - 101

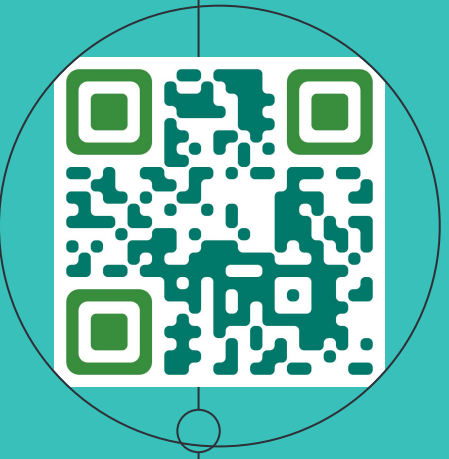
Ethereum Edition

Table of Contents

1

Intro & Basics

Theory

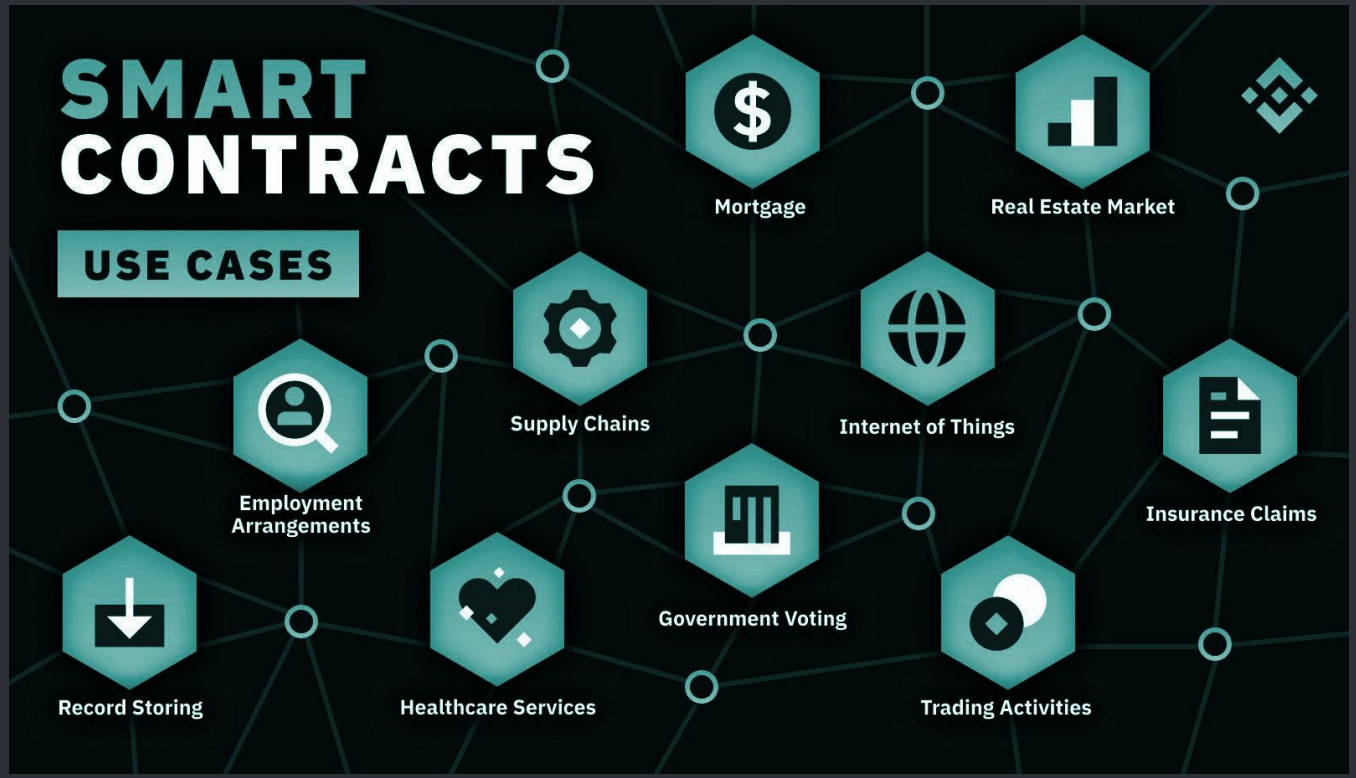


msg.sender ==

Samandeep Singh

- Organizer - BSides Singapore
- Approx. 10 years in Offsec
- Smart contract security since 2021
- Tweet @samanl33t

Smart Contracts Today...



Ref: https://twitter.com/binance/status/1428926886382604288/photo/1

WHY?

PLATYPUS FINANCE - REKT

Thursday, February 16, 2023
Platypus Finance - Avalanche - REKT

Evolution works in mysterious ways. Platypus Finance lost $8.5M to a flash loan attack on its recently-launched stablecoin. It's a jungle out there... and, as ever, it's survival of the fittest.

DFORCE NETWORK - REKT

Monday, February 13, 2023
dForce Network - Arbitrum - Optimism - REKT

dForce Network was hit for $3.65M on both Arbitrum and Optimism. This attack on two fronts exploited a common reentrancy vulnerability. How much more will be lost to this bug?

MORE

ORION PROTOCOL - REKT

Saturday, February 4, 2023
Orion Protocol - REKT

The hunter has become the hunted. Orion Protocol fell prey to a $3M reentrancy exploit on ETH and BSC. The loss was contained to an internal broker account and user funds are safe. Let's hope they take a more Sirius approach in future.

MORE

BONQDAO - REKT

Friday, February 3, 2023
BonqDAO - AllianceBlock - REKT

BonqDAO got bonked for $120M, but the anonymous attacker got away with less than $2M. The hacker was able to manually update the price feed of collateral by staking just $175 worth of TRB tokens.

MORE

List of DeFi Hacks

DEFI APPLICATION	DATE OF HACK	CAUSE OF HACK	AMOUNT STOLEN (USD)
Beanstalk	18 April 2022	Flash loans were used to manipulate governance mechanisms	$182 million
Ronin	29 March 2022	A connection to the developers of Axie Infinity, Sky Mavis, was exploited	$552 million
Wormhole	3 February 2022	The bridge between Ethereum and Solana was manipulated	$326 million
Vulcan Forged	13 December 2021	User's private keys were stolen	$140 million
BadgerDAO	2 December 2021	Malicious code was added into the application, changing user's wallet permissions	$120 million

Last updated on Feb 17th 2023 at 00:00 UTC

	Name	Whitehat Score	Total Earnings	Paid Reports
1	Barracuda3172	959	$13,010,000	4
2	RetailDdene2946	735	$10,020,000	2
3	PwningEth	606	$8,000,000	8
4	Bobface	253	$3,301,000	4
5	gegul	210	$2,705,632	6
6	Oralie96051	193	$2,386,250	8

What is a Blockchain?

BlockChain

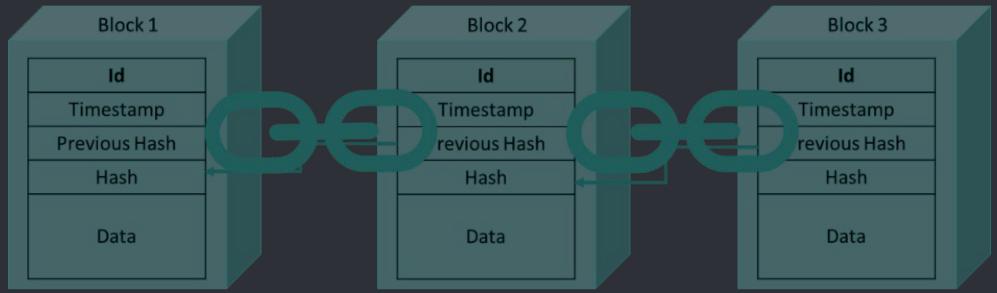
Type of Distributed Ledger Technology (DLT)

Immutable Chain of Blocks

- Each block containing:
  - data
  - hash(data)
  - hash(previous block)

Distributed among peers/users.

Maintains Integrity



Ref: https://www.paiementor.com/wp-content/uploads/2020/02/Pictorial-representation-of-a-blockchain-1024x388.png

Ethereum

Ethereum is a technology for building apps and organizations, holding assets, transacting and communicating without being controlled by a central authority. There is no need to hand over all your personal details to use Ethereum - you keep control of your own data and what is being shared. Ethereum has its own cryptocurrency, Ether, which is used to pay for certain activities on the Ethereum network.

- **Turing Complete -** World Computer
- **Programmable** - Allows building of DApps
- **DApps** use the blockchain to store data/code
- Transaction Fee paid in **Ether**
- Uses **Ethereum Virtual Machine (EVM)** to execute code (**Contracts**)

Ref: https://ethereum.org/en/what-is-ethereum/

Ethereum(Web3) ‘Terminology’

Term	Definition
Accounts	The addresses/entity with Ether that can send transactions (Types: EOA or Contract address)
Wallet	Application/Hardware that stores the private keys for blockchain assets.
EVM	Ethereum Virtual Machine
Gas	Fee paid for each transaction on the blockchain
Block Explorer	Tool(s) for browsing information on the blockchain

What are Smart Contracts and How they work?

A Program

- that runs on a Blockchain
- is a collection of code and data
- that is meant to automatically execute based on the transactions received.
- may contain balance

Broadly consists of

- Data
- Functions
- Events

Languages:

- Solidity/Viper/Huff - Ethereum EVM
- Rust/C/C++ - Solana
- Plutus - Cardano

```
pragma solidity ^0.8.14;

UnitTest stub | dependencies | uml | draw.io
contract IncDec {
    uint public count;

    ftrace | funcSig
    function increment() public {
        count++;
    }

    ftrace | funcSig
    function decrement() public {
        count--;
    }

    ftrace | funcSig
    function getCount() public view returns (uint) {
        return count;
    }
}
```

Lab Environment

Setup

We will work with:

- Ziion OS (https://www.ziion.org/download)

Lab Notes, Sample code and Slides

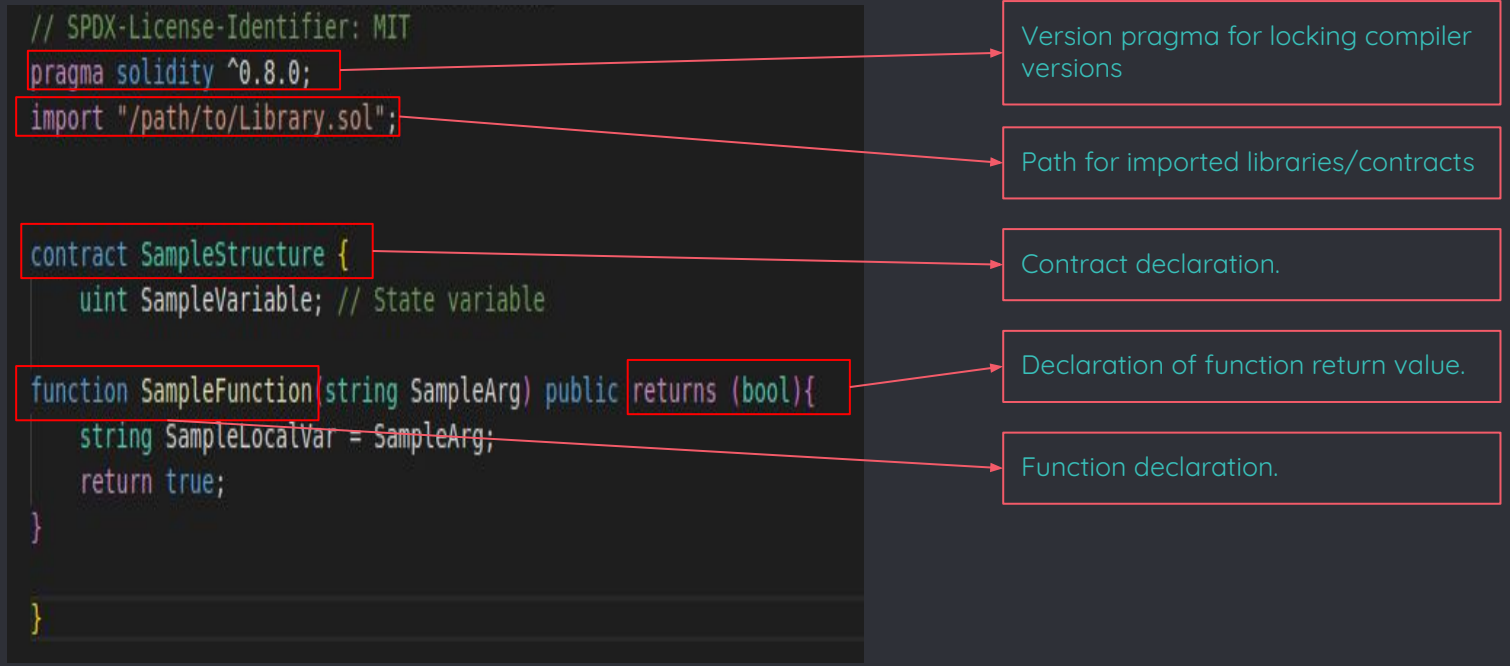
- https://github.com/samanL33T/Disobey2023_Smart_Contract_Bug_Hunting_101

2

Solidity Primer

Theory + Practical

Structure of Solidity Smart contract



Solidity - Types

Usual

- *string*
- signed/unsigned integers - *int/uint/utin8/uint256...*
- bool: *true/false*

Address:

- Holds a 20 byte Ethereum address
- ***address payable***
  - *to receive Ether.*
  - Members: ***transfer*** & ***send***
- Members:
  - ***balance*** & ***transfer***
  - ***call/delegatecall/staticcall***

```
contract SolidityTypes {
    int IamSignedInt;
    uint256 Userbalance;
    bool alwaysTrue = true;

    address UselessWallet;
    UselessWallet = 0xac0974bec39a17e36ba4a6b4d238ff944bacb478cbed5efcae784d7bf4f2ff80;

    address payable UsefulWallet;
    UsefUllWallet = 0x5de4111afa1a4b94908f83103eb1f1706367c2e68ca870fc3fb9a804cdab365a;

function TestAddress(uint256 amount) public returns (bool){
    address payable MakeitUsefull;
    MakeitUsefull = payable(UselessWallet);
    Userbalance = balance.MakeitUsefull;
    MakeitUsefull.transfer(amount);
    return true;
}
}
```

Solidity - Types (cont..)

Enums:

- To create user defined data-types
- Max 256 members allowed
- Members:
  - type(<EnumName>).min
  - type(<EnumName>).max

Function types:

- ***internal*** (default)/**external**
- ***pure/view/payable***

```
contract SolidityTypes2 {
    enum ActionChoices { GoLeft, GoRight, GoStraight, SitStill }
    ActionChoices choice;
    ActionChoices constant defaultChoice = ActionChoices.GoStraight;

    function setGoStraight() external pure {
        choice = ActionChoices.GoStraight;
    }
}
```

Solidity - Types (cont..)

Reference type with data location:

- Reference type variable declaration needs a data location to be specified
- Data location can be:
  - ***memory***
  - ***storage***
  - ***calldata***

Mappings

- ***mapping (KeyType KeyName => ValueType ValueName)***

```
contract SolidityTypes3 {
    uint val;  //data location is storage  - state variable

    // The data location of memoryArray is memory.
    function datalocations(uint[] memory memoryArray) public {
        val = memoryArray;
        uint[] storage y = x; //data location of y is storage
        ....
        ....

    mapping(address userWallet => uint256 balance) public StoredFunds;

    function mappingtype(address user) public returns(uint256){
        uint256 userbalance = StoredFunds[user];
        return userbalance;

    }
```

Solidity - Units

Ether Units

- wei
- 1 gwei == 1e9 wei
- 1 ether == 1e18 wei

Time Units

- 1 seconds
- 1 minutes == 60 seconds
- 1 hour == 60 minutes
- 1 days == 24 hours
- 1 weeks == 7 days

```
function SolidityUnits(uint start, uint daysAfter, address payable recipient) public {
    if (block.timestamp >= start + daysAfter * 1 days) {
        recipient.call{value: 1 ether}(""); // 1000000000000000000 wei
    }
}
```

Solidity - Special variables & Functions

Transaction properties:

- ***msg.sender (address)*** - sender address
- ***msg.value (uint)*** - wei sent
- ***msg.data (bytes calldata) -*** *full calldata*
- ***msg.sig (bytes4)*** - 4 bytes of calldata/function identifier
- ***tx.origin*** (address) sender address / full-chain

Variables for block properties

- **block.timestamp (uint)** - *current block timestamp*
- **block.number (uint)** - *current block number*
- **blockhash(uint blockNumber)** - *hash of the block (For last 256 blocks)*

Error Handling:

- ***assert (bool condition)***
  - *reverts if condition is not met // for internal errors*
- ***require (bool condition, string memory "message")***
  - *reverts if condition is not met// for inputs & external components/errors*
- ***revert (string memory "reason")***
  - *abort execution, revert state changes*

Solidity - Special variables & Functions

```
contract SpecialVariables {
    function SpecialTest() public {
        address sender = msg.sender;
        uint256 value = msg.value;

        uint256 currentTimestamp = block.timestamp;
        uint256 blockNumber = block.number;
        bytes memory blockHash = blockhash(blockNumber - 1);
        require(currentTimestamp >= value, "Timestamp is not current");

        assert(value > 0, "Value of transfer must be greater than zero");

        bytes memory data = msg.data;

        bytes4 functionSignature = msg.sig;

        address transactionOrigin = tx.origin;

    }
}
```

Lab 01

- Getting familiar with Remix IDE
- Writing and deploying your first smart contract

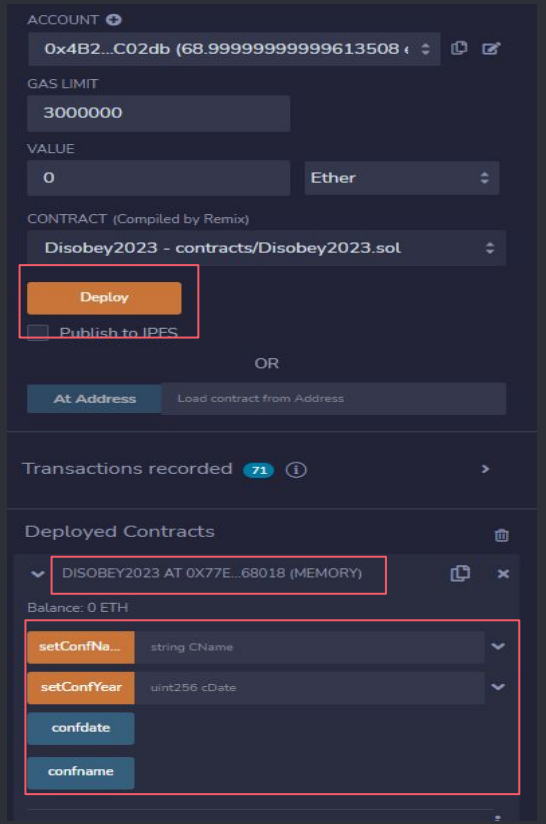
Contract Creation/Deploying

Contract can be created on the Ethereum blockchain:

- From outside
  - Web3.js / ether.js etc.
  - Remix IDE
  - Dev frameworks - foundry, Hardhat, Brownie etc.
- From other on-chain contracts

On Creation:

- Returns the address of deployed contract
- Automatically creates **Getter functions** for ***public state variables***



Solidity - Functions

function name (<parameter types>) {<visibility>} [<mutability>] [returns (<return types>)]

E.g:

```
function namePrinter (string memory abc) public view returns (string){
return abc;
}
```

Solidity - Functions (cont..)

State mutability

- ***pure*** - state modification
- ***view*** - no state modification

Special Functions

- **receive ()**
  - executed when contract receives **ether** without a specified function call with empty **calldata**
  - Called when there's no payable **fallback** function available
  - At most 1 receive() function per contract
  - No '*function*' keyword for declaration
  - Declared as ***'external payable'***

```
contract SampleReceiveFallback {
    receive() external payable {
        // Perform some action with the value sent to this contract
        // ...
    }

    fallback() external payable {
        // Perform some action as a default function when this contract
        // is called without a specified function

    }
}
```

- ***fallback ()***
  - executed when:
    - no other functions match the provided function signature
    - contract receives **ether and no calldata** without a specified function call.
    - there's no **receive()** function

Solidity - Function Modifiers

Modifiers

- Can be used to change the behaviour of a function during execution
- For e.g for Access Control checks

```
contract OWnerRulez {
    modifier onlyOwner {
            require(msg.sender == owner,"Only owner can call this function." );
            _;
    }

    function destroy() public onlyOwner {
        selfdestruct(owner);
    }
```

Solidity Smart Contract Dev Frameworks

Hardhat

- JavaScript based
- Allows writing tests in JS
- Good to simulate front-end web app calls to smart contracts
- Has its own ‘Hardhat Network’ - local node

Brownie

- Python based
- Can write tests in Python
- Uses ganache for local node

Foundry

- Purely Solidity based
- Allows writing tests in solidity
- No language switching
- Supports property based fuzzing

Foundry

Foundry tools

- **forge** - For building/compiling, testing and deploying the smart contracts
- **cast** - For making direct RPC calls to Ethereum blockchain.
- **anvil** - For locally deploying a test Ethereum blockchain/ testnode
- **chisel** - Solidity REPL


```
❯ forge test --match-test testsetConfYear
[⠊] Compiling...
No files changed, compilation skipped

Running 1 test for test/Disobey2023.t.sol:Disobey2023Test
[PASS] testsetConfYear(uint256) (runs: 256, μ: 7659, ~: 7659)
Test result: ok. 1 passed; 0 failed; finished in 14.49ms
```

Ref: https://book.getfoundry.sh/

Lab 02

Getting familiar with Foundry

- Foundry tooling (forge, cast, anvil)
- Creating a foundry project
- Building a project
- Writing and running a simple test
- Using existing foundry project

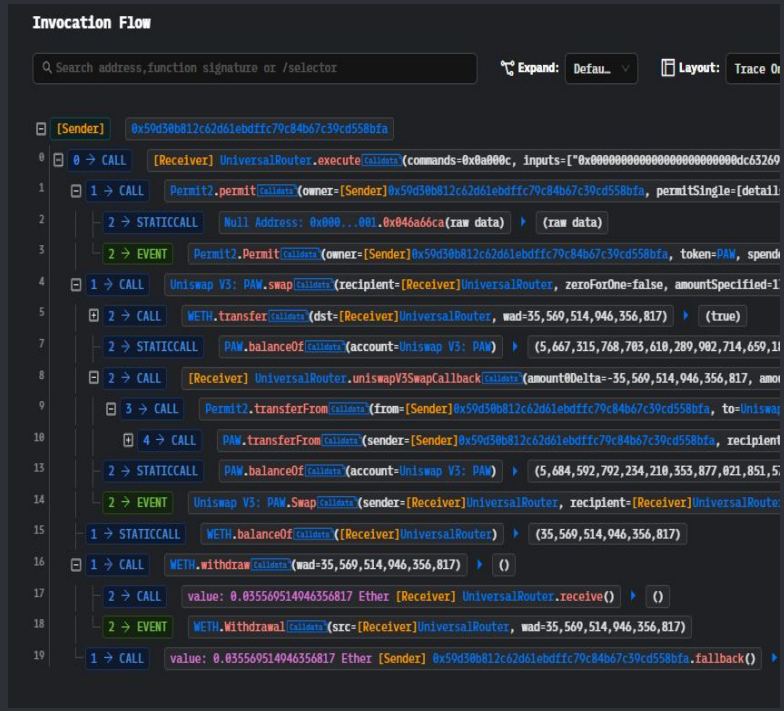
OnChain- Block/Transaction Explorer(s)

Block Explorer

- To view contracts and user addresses on the block
- Provides transaction information as well
- Specific for each chain
- Eg: Etherscan (https://etherscan.io/)

Transaction Explorers:

- Detailed information on each transaction
- Some allow debugging of old transactions
- Supports multiple chains
- Good for Incident response/On-Chain Forensics
- Eg:
  - Phalcon (https://phalcon.blocksec.com/)
  - Tx.viewer (https://openchain.xyz/trace)



3

Smart Contract Vulnerabilities

Theory

Vulnerability categories you might already know of

Insecure Design

Missing or Insecure Access Control

- No checks on authorized function calls
- Incorrect function visibility

Insecure Arithmetic operations

- Integer overflows
- Integer underflows
- Floating points and precision errors

Insecure Business Logic

- Missing checks on multi-step transactions
- Broken Contract logic (for a game/voting/DAO/DeFi etc.)

Insecure Storage of data

- Sensitive data hardcoded in the contract code.
- Sensitive data saved on blockchain

Missing Input validation

- AllowList vs DenyList
- Missing validation on untrusted user input

Smart Contract specific vulnerabilities

Vulnerabilities/Attacks

- **Reentrancy** - No proper checks on recursive function calls
- Insecure **delegatecall**
- Insecure Proxies
- No check for **zero address**
- **Gas usage** related vulnerabilities
- Backdoors and over privileged contract owners/deployers
- Denial-of-Service
  - Locking of funds
  - Locking of users out of the contract
- Race-conditions : Frontrunning
- Insecure randomness
- DeFi:
  - Oracle price manipulation

The impact

- Huge financial loss from even minor bugs

Reentrancy

What?

- When a contract allows repeated calls to a function(s).
- And there is no mechanism to prevent reentrant calls or lock the state.
- Contract does not follow **check-effect-interaction** pattern
- Can lead to draining of contract funds

Real-world attacks

- DAO Hack (2016) - 3 mil +
- Orion Protocol (2023) - 3 mil +

```solidity
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.11.0;

UnitTest stub | dependencies | uml | draw.io
contract Reentrant {
    uint256 public balance;
    mapping (address => uint256) public deposits;

    ftrace | funcSig
    function deposit() public payable {
        deposits[msg.sender] += msg.value;
        balance += msg.value;
    }

    ftrace | funcSig
    function withdraw(uint256 amount) public {
        require(deposits[msg.sender] >= amount, "Not enough deposit");
        payable(msg.sender).transfer(amount);
        deposits[msg.sender] -= amount;

    }
}
```

Hundred Finance (Reentrancy)

```
function transfer(address _to, uint256 _value) public returns (bool) {
        require(superTransfer(_to, _value));
        callAfterTransfer(msg.sender, _to, _value); //vulnerable point
        return true;
    }

    function transferFrom(address _from, address _to, uint256 _value) public returns (bool) {
        require(super.transferFrom(_from, _to, _value));
        callAfterTransfer(_from, _to, _value); //vulnerable point
        return true;
    }
```

Ref: https://web3sec.notion.site/web3sec/ba459372dc434341b99ec92a932f98dc?v=7fceca7b3da74aa8a99b49c44a2a3916&p=e4eb32be4f3a4a5d873b31901c07ceb1&pm=s

OnChain Private data

What?

- ‘private’ variables are private to the current contracts
- But can still be read outside of the contract scope.
- Any data on chain is publicly readable.

Real-world attacks

-

```
pragma solidity ^0.8.11;

UnitTest stub | dependencies | uml | draw.io
contract Privdata {
    struct User {
        address addr;
        uint256 secret;
    }

    mapping(address => User) private users;

    ftrace | funcSig
    function addUser(uint256 _secret) public {
        users[msg.sender].addr = msg.sender;
        users[msg.sender].secret = _secret;
    }

    ftrace | funcSig
    function getUserSecret(address _user) public view returns (uint256) {
        return users[_user].secret;
    }
}
```

Insecure Access Control

What?

- When a contract allows repeated calls to a function(s).
- And there is no mechanism to prevent reentrant calls or lock the state.
- Contract does not follow **check-effect-interaction** pattern
- Can lead to draining of contract funds

Real-world attacks

- Parity Hack (2017) - 514k ETH

Code snippet from - HPay (2022)

```
function setToken(address _addr) public { //vulnerable point
    configuration.stakingToken = ERC20(_addr);
    configuration.rewardsToken = ERC20(_addr);
}
```

Ref: https://web3sec.notion.site/web3sec/ba459372dc434341b99ec92a932f98dc?v=7fceca7b3da74aa8a99b49c44a2a3916&p=fb2894e0ae804b9fb9c35e76f55db4d0&pm=s

Integer Overflows & Underflows

What?

- Without proper checks, an integer can wrap around to a higher or lower number
- For example:
- Safe by default in Solidity 0.8.x onward
- Unchecked{} can still be used and leave it vulnerable

Real-world attacks

- Umbrella Network (2022)
- Beauty Chain (2018) - 900 mil +

```
pragma solidity ^0.8.11;

contract OverUnderFlow {
    uint8 public totalSupply;

    constructor() {
        totalSupply = 255;
    }

    function increaseSupply(uint8 _value) public {
        unchecked{totalSupply += _value;}  // overflow
    }

    function decreaseSupply(uint8 _value) public {
        unchecked{totalSupply -= _value;}  // underflow
    }

}
```

Umbrella Network (Integer Overflow)

```
/// @param amount tokens to withdraw
    /// @param user address
    /// @param recipient address, where to send tokens, if we migrating token address can be zero
    function _withdraw(uint256 amount, address user, address recipient) internal nonReentrant updateReward(user) {
        require(amount != 0, "Cannot withdraw 0");
        _totalSupply = _totalSupply - amount;
        _balances[user] = _balances[user] - amount; //vulnerable point, overflow
        require(stakingToken.transfer(recipient, amount), "token transfer failed");

        emit Withdrawn(user, amount);
    }
```

Ref: https://web3sec.notion.site/web3sec/ba459372dc434341b99ec92a932f98dc?v=7fceca7b3da74aa8a99b49c44a2a3916&p=e344c6b2a9a14e8083892dcdc6a02e86&pm=s

Insecure Random number generation

What?

- Difficult to implement PRNG in Smart contracts
- All Block related values are insecure and can be predicted

Real-world attacks

- Multiple gambling Smart contracts (2018)
- LuckyTiger (2022)

Code snippet from - LuckyTiger (2022)

```
function _getRandom() private returns(bool) {
        uint256 random = uint256(keccak256(abi.encodePacked(block.difficulty, block.timestamp)));
        uint256 rand = random%2;
        if(rand == 0){return lucky = false;}
        else         {return lucky = true;}
    }
```

Insecure Delegatecall

What?

- Generally, message call allows a contract to call another contract(s)
- **delegatecall** is a variant of message call
- It allows the code at target contract to be executed in the context of calling contract
- Meaning: Caller's storage, balance and current address (msg.sender) is being used
- Only the code from **callee** contract is used.
- Can allow **User controlled callee contracts** to overtake the **ownership of caller contract**
- Can allow attacker to drain **User controlled state variables**

Real-world attacks

- Parity Multisig Wallet (2017)

Example (Insecure Delegate Call)

```
pragma solidity ^0.8.11;

UnitTest stub | dependencies | uml | draw.io
contract Delegator {
    address public owner;
    ftrace
    constructor() {
        owner = msg.sender;
    }

    ftrace | funcSig
    function execute(address callee, bytes memory _code) public {
        (bool success, bytes memory ret) = callee.delegatecall(_code);
        require(success, "Delegate call failed");
    }
}
```

Denial of Service

What?

- Smart contracts can be rendered useless causing DoS
- Failed external calls can cause DoS
- For example with **Unexpected Revert**

Code snippet from - Akutar NFT (2022)

```
function processRefunds() external {
      require(block.timestamp > expiresAt, "Auction still in progress");
      uint256 _refundProgress = refundProgress;
      uint256 _bidIndex = bidIndex;
      require(_refundProgress < _bidIndex, "Refunds already processed");

      uint256 gasUsed;
      uint256 gasLeft = gasleft();
      uint256 price = getPrice();

      for (uint256 i=_refundProgress; gasUsed < 5000000 && i < _bidIndex; i++) {
          bids memory bidData = allBids[i];
          if (bidData.finalProcess == 0) {
            uint256 refund = (bidData.price - price) * bidData.bidsPlaced;
            uint256 passes = mintPassOwner[bidData.bidder];
            if (passes > 0) {
                refund += mintPassDiscount * (bidData.bidsPlaced < passes ? bidData.bidsPlaced : passes);
            }
            allBids[i].finalProcess = 1;
            if (refund > 0) {
                (bool sent, ) = bidData.bidder.call{value: refund}(""); // low-lecel call
                require(sent, "Failed to refund bidder");
```

Ref: https://web3sec.notion.site/web3sec/ba459372dc434341b99ec92a932f98dc?v=7fceca7b3da74aa8a99b49c44a2a3916&p=a6113455c79546a588ba7ce3dea04225&pm=s

tx.origin - Phishing attack

What?

- Contract uses **tx.origin** instead of **msg.sender** to verify authentication
- **tx.origin** is insecure
- Attacker can write a phishing contract that tricks the owner to call a function
- Which indirectly calls the function in vulnerable contract.
- Since the transaction was initiated by **owner,** the **tx.origin** authentication check passes.

```
pragma solidity ^0.8.11;

UnitTest stub | dependencies | uml | draw.io
contract Vulnerable {
    address public owner;

    ftrace
    constructor() {
        owner = msg.sender;
    }

    ftrace | funcSig
    function changeOwner(address newOwner) public {
        if (tx.origin == owner) {
            owner = newOwner;
        }
    }
}
```

4

Bug Hunting

Practical

Challenge

Find the Bug

Static Analysis

Solhint

- Linter
- Mainly a style guide validator
- Supports some security validations as well
- Supports custom rules

solhint

Slither

- Static Analysis Framework
- Specifically for finding security vulnerabilities
- Highlights the vulnerabilities based on their severity
- Supports solidition >=0.4
- Fast

Lab 03

Static Analysis

- Manual Code Reviews
- **Slither** & **solhint** for static code analysis

5

What next?

Further studying and practice

Further Learning ideas and resources

Learn Solidity in detail

- Cryptozombies
- Buildspace

Solve CTF challenges

- Ethernaut
- Damn Vulnerable DeFi
- Paradigm CTF
- Quill CTF challenges
- Ciphershastra CTF - https://ciphershastra.com/

Learn to Reproduce real-world attacks

- DeFiHackLabs - https://github.com/SunWeb3Sec/DeFiHackLabs

Participate in audit competitions and bug bounties

- Code4rena
- SpearbitDao
- Immunefi

Additional resources for learning solidity bug classes

- Secureum :
  - Bootcamp
  - Monthly RACE(s)

Thank you

@samanl33t/saman.J.L33T@protonmail.ch

Appendix A: Resources & References

- SCSVS - https://github.com/securing/SCSVS
- SWC Registry - https://swcregistry.io/
- Echidna - https://github.com/crytic/building-secure-contracts/tree/master/program-analysis/echidna
- DEFi VulnLabs - https://github.com/SunWeb3Sec/DeFiVulnLabs
- DeFiHackLabs - https://github.com/SunWeb3Sec/DeFiHackLabs
- DeFI Hack Analysis - https://web3sec.notion.site/web3sec/ba459372dc434341b99ec92a932f98dc?v=7fceca7b3da74aa8a99b49c44a2a3916
- Smart Contract attack vectors - https://github.com/harendra-shakya/smart-contract-attack-vectors
- Awesome Foundry - https://github.com/crisgarner/awesome-foundry
- Code4rena Audit Reports
- Sherlock Audit Reports