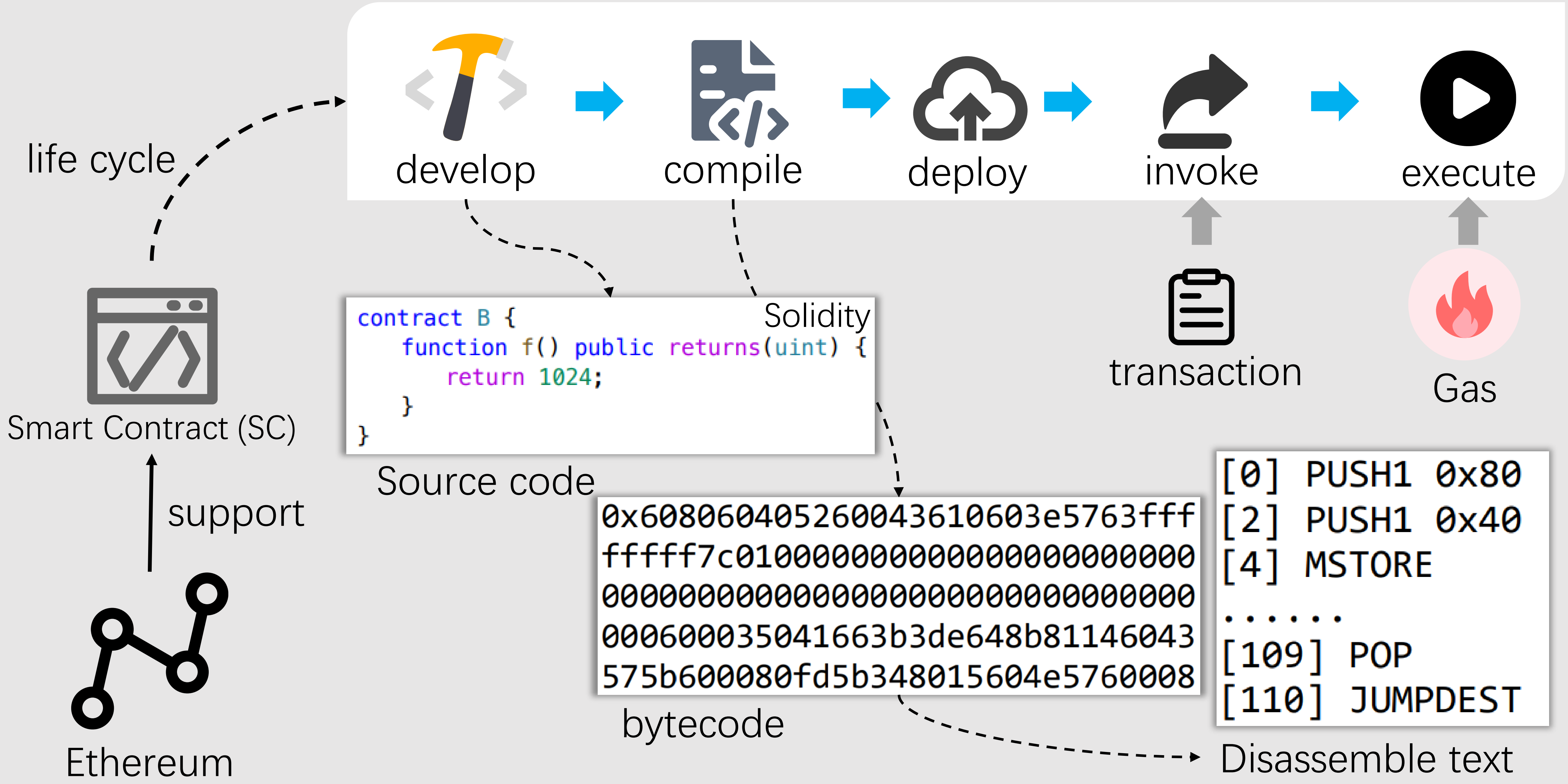
# Smart Contract Vulnerabilities: Vulnerable Does Not Imply Exploited

Daniel Perez, Benjamin Livshits

Imperial College London

USENIX Security 2021

# Background



life cycle
develop
compile
deploy
invoke
execute
Smart Contract (SC)
support
Ethereum
contract B {
function f() public returns(uint) {
return 1024;
}
}
Solidity
Source code
transaction
Gas
0x6080604052600043610603e5763fff
fffff7c0100000000000000000000000
00000000000000000000000000000000
00060003504166 3b3de648b81146043
575b600080fd5b348015604e5760008
bytecode
[0] PUSH1 0x80
[2] PUSH1 0x40
[4] MSTORE
......
[109] POP
[110] JUMPDEST
Disassemble text

# Background

- Immutability: Once a smart contract is deployed, its code cannot be modified, which means the vulnerability cannot be fixed.
- Vulnerabilities: such as Re-Entrancy, Integer Overflow, Locked Ether …
- Re-Entrancy:

```
Contract A {
    数据结构 记录每个用户向本合约存入了多少ETH;
    ...
    function withdraw(uint256 amount) {
        检查msg.sender的余额，当余额大于amount时 {
            向msg.sender转amount个ETH;
            msg.sender的余额 -= amount;
        }
    }
    ...
}
```

如果向智能合约发送ETH，即使没有指定要调用该合约的函数，也会尝试调用该合约的fallback函数（如果它有fallback函数的话）。

# Background

- Immutability: Once a smart contract is deployed, its code cannot be modified, which means the vulnerability cannot be fixed.
- Vulnerabilities: such as Re-Entrancy, Locked Ether, Integer Overflow …
- Re-Entrancy:

```
Contract A {
    数据结构 记录每个用户向本合约存入了多少ETH;
    …
    function withdraw(uint256 amount) {
        检查msg.sender的余额，当余额大于amount时 {
            向msg.sender转amount个ETH;
        }
        msg.sender的余额 -= amount;
    }
    …
}
```

```
Contract B {
    function attack() {
        A.withdraw(100);
    }

    // fallback
    function() {
        A.withdraw(100);
    }
}
```

# Research Gap & Problems

- A great deal of both academic and practical interest in the topic of vulnerabilities in smart contracts.

- Most of the work has focused on detecting vulnerable contracts:
  Source code/bytecode ➔ vulnerabilities

- Problem: it is frequently difficult to estimate what fraction of discovered **vulnerabilities are exploited in practice**.

- Why is this problem important?
  —— It can support analysis tool development efforts by helping to understand what type of exploitation is happening in the wild.

# Goals and Challenges

- Goal
  - vulnerabilities reported (23,327 SCs) VS. actual exploitation (unkonwn)

- Challenges
  - The number of contracts is very large, how to detect them automatically?
  - Scalability —— Need to detect multiple types of vulnerabilities.

# Dataset

- The authors analyze the vulnerable contracts reported by the following six academic papers.
- The dataset is comprised of a total of 821,219 contracts, of which 23,327 contracts have been flagged as vulnerable.

| **Name** | **Contracts analyzed** | **Vulnerabilities found** | **Ether at stake** at time of report |
|---|---|---|---|
| Oyente | 19,366 | 7,527 | 1,287,032 |
| Zeus | 1,120 | 861 | 671,188 |
| Maian | NA | 2,691 | 15.59 |
| Securify | 29,694 | 9,185 | 724,306 |
| MadMax | 91,800 | 6,039 | 1,114,958 |
| teEther | 784,344 | 1,532 | 1.55 |

**Figure 2:** Summary of the contracts in our dataset.

| **Name** | **Vulnerabilities** | | | | | | **Report month** | **Citation** |
|---|---|---|---|---|---|---|---|---|
| | RE | UE | LE | TO | IO | UA | | |
| Oyente | ✓ | ✓ | | ✓ | ✓ | | 2016-10 | [35] |
| ZEUS | ✓ | ✓ | ✓ | ✓ | ✓ | | 2018-02 | [31] |
| Maian | | | ✓ | | | ✓ | 2018-03 | [39] |
| SmartCheck | ✓ | ✓ | ✓ | | ✓ | | 2018-05 | [48] |
| Securify | ✓ | ✓ | ✓ | ✓ | | ✓ | 2018-06 | [51] |
| ContractFuzzer | ✓ | ✓ | | | | | 2018-09 | [30] |
| teEther | | | | | | ✓ | 2018-08 | [32] |
| Vandal | ✓ | ✓ | | | | | 2018-09 | [15] |
| MadMax | | | ✓ | | ✓ | | 2018-10 | [24] |

**Figure 1:** A summary of smart contract analysis tools presented in prior work.

# Dataset

- The authors find a lot of contradiction in the analysis of the different tools.

| Tools | Total | Agreed | Disagreed | % agreement |
|---|---|---|---|---|
| Oyente/Securify | 774 | 185 | 589 | 23.9% |
| Oyente/Zeus | 104 | 3 | 101 | 2.88% |
| Zeus/Securify | 108 | 2 | 106 | 1.85% |

**Figure 4:** Agreement among tools for re-entrancy analysis.

- This became another motivation for this study.
  —— "this gives us yet another motivation to find out the impact of the reported vulnerabilities."

# Methodology

- Main Idea:
  perform bytecode-level **transaction analysis** to check for potential exploits.

1. Trace recording
   Smart Contracts in Dataset ➔ Transaction ➔ Execution trace
   Trace: <TxHash, invoked SC, data, **executed instraction**>

2. Encode traces into a **Datalog** representation

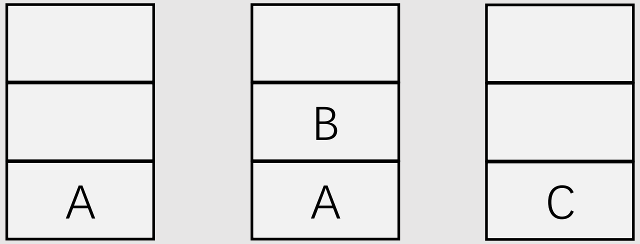
3. Datalog queries for detecting different vulnerability classes

# Datalog: express instructions in an abstract way

Datalog facts:

Case 1:

PUSH A; PUSH B; ADD;




Datalog: is_output(C, A), is_output(C, B)

Case 2:

PUSH A; PUSH B; SDIV;

Datalog: is_signed(C), is_output(C, A), is_output(C, B)

| **Fact** | **Description** |
|---|---|
| is_output($v_1 \in V$, $v_2 \in V$) | $v_1$ is an output of $v_2$ |
| size($v \in V$, $n \in \mathbb{N}$) | $v$ has $n$ bits |
| is_signed($v \in V$) | $v$ is signed |
| in_condition($v \in V$) | $v$ is used in a condition |
| call($a_1 \in A$, $a_2 \in A$, $p \in \mathbb{N}$) | $a_1$ calls $a_2$ with $p$ Ether |
| create($a_1 \in A$, $a_2 \in A$, $p \in \mathbb{N}$) | $a_1$ creates $a_2$ with $p$ Ether |
| expected_result($v \in V$, $r \in \mathbb{Z}$) | $v$'s expected result is $r$ |
| actual_result($v \in V$, $r \in \mathbb{Z}$) | $v$'s actual result is $r$ |
| call_result($v \in V$, $n \in \mathbb{N}$) | $v$ is the result of a call and has a value of $n$ |
| call_entry($i \in \mathbb{N}$, $a \in A$) | contract $a$ is called when program counter is $i$ |
| call_exit($i \in \mathbb{N}$) | program counter is $i$ when exiting a call to a contract |
| tx_sstore($b \in \mathbb{N}, i \in \mathbb{N}, k \in \mathbb{N}$) | storage key $k$ is written in transaction $i$ of block $b$ |
| tx_sload($b \in \mathbb{N}, i \in \mathbb{N}, k \in \mathbb{N}$) | storage key $k$ is read in transaction $i$ of block $b$ |
| caller($v \in V, a \in A$) | $v$ is the caller with address $a$ |
| load_data($v \in V$) | $v$ contains transaction call data |
| restricted_inst($v \in V$) | $v$ is used by a restricted instruction |
| selfdestruct($v \in V$) | $v$ is used in SELFDESTRUCT |

**(a)** Datalog facts.

# Datalog

Datalog rules:

The Datalog is further abstracted according to the rules.

- is_output(v1, v2) ➔ depends(v1, v2)
- call(a1, a2, p) ➔ call_flow(a1,a2,p)
- call(a1, a3, p) + call(a3, a2, _) ➔call_flow(a1, a2, p)

| **Datalog rules** |
|---|
| $\texttt{depends}(v_1 \in V,\ v_2 \in V)\ \texttt{:-}\ \texttt{is\_output}(v_1,\ v_2).$<br>$\texttt{depends}(v_1,\ v_2)\ \texttt{:-}\ \texttt{is\_output}(v_1,\ v_3),\ \texttt{depends}(v_3,\ v_2).$ |
| $\texttt{call\_flow}(a_1 \in A,\ a_2 \in A,\ p \in \mathbb{Z})\ \texttt{:-}\ \texttt{call}(a_1,\ a_2,\ p).$<br>$\texttt{call\_flow}(a_1 \in A,\ a_2 \in A,\ p \in \mathbb{Z})\ \texttt{:-}\ \texttt{create}(a_1,\ a_2,\ p).$<br>$\texttt{call\_flow}(a_1,\ a_2,\ p)\ \texttt{:-}\ \texttt{call}(a_1,\ a_3,\ p),\ \texttt{call\_flow}(a_3,\ a_2,\ \_).$ |
| $\texttt{inferred\_size}(v \in V,\ n \in \mathbb{N})\ \texttt{:-}\ \texttt{size}(v,\ n).$<br>$\texttt{inferred\_size}(v,\ n)\ \texttt{:-}\ \texttt{depends}(v,\ v_2),\ \texttt{size}(v_2,\ n).$ |
| $\texttt{inferred\_signed}(v \in V)\ \texttt{:-}\ \texttt{is\_signed}(v).$<br>$\texttt{inferred\_signed}(v)\ \texttt{:-}\ \texttt{depends}(v,\ v_2),\ \texttt{is\_signed}(v_2).$ |
| $\texttt{condition\_flow}(v \in V,\ v \in V)\ \texttt{:-}\ \texttt{in\_condition}(v).$<br>$\texttt{condition\_flow}(v_1,\ v_2)\ \texttt{:-}\ \texttt{depends}(v_1,\ v_2),\ \texttt{in\_condition}(v_2).$ |
| $\texttt{depends\_caller}(v \in V)\ \texttt{:-}\ \texttt{caller}(v_2,\ \_),\ \texttt{depends}(v, v_2).$ |
| $\texttt{depends\_data}(v \in V)\ \texttt{:-}\ \texttt{load\_data}(v_2,\ \_),\ \texttt{depends}(v, v_2).$ |
| $\texttt{caller\_checked}(v \in V)\ \texttt{:-}\ \texttt{caller}(v_2, \_),$<br>$\texttt{condition\_flow}(v_2, v_3),\ v_3 < v.$ |

**(b)** Datalog rule definitions.

# Datalog

Datalog queries:

for detecting different vulnerability classes

| Vulnerability | Query |
|---|---|
| Re-Entrancy | call_flow($a_1$, $a_2$, $p_1$), call_flow($a_2$, $a_1$, $p_2$), $a_1 \neq a_2$ |
| Unhandled Excep. | call_result($v$, 0), ¬condition_flow($v$,_) |
| Transaction Order Dependency | tx_sstore($b$,$t_1$,$i$), tx_sload($b$,$t_2$,$i$), $t_1 \neq t_2$ |
| Locked Ether | call_entry($i_1$,$a$), call_exit($i_2$), $i_1 + 1 = i_2$ |
| Integer Overflow | actual_result($v$, $r_1$), expected_result($v$, $r_2$), $r_1 \neq r_2$ |
| Unrestricted Action | restricted_inst($v$), depends_data($v$), ¬depends_caller($v$), ¬caller_checked($v$) ∨ selfdestruct($v$), ¬caller_checked($v$) |

(c) Datalog queries for detecting different vulnerability classes.

For example (Re-Entrancy):

call_flow(a1, a2, p1)

➔Contract a1 invokes contract a2

call_flow(a2, a1, p2)

➔Contract a2 invokes contract a1

a1≠a2

➔a1 and a2 are different contracts

There are false positives!

# Experimental results

(Use re-entry as an example and ignore the remaining 5 vulnerabilities)

**Results:**

- Vulnerable: 4,337 contracts (457,073 transactions);
- Actual exploitation: 116 contracts;
- Ether exploited: <= 6,076 ETH

**Manual analysis**

The top contracts in terms of fund lost were analyzed manually and it was confirmed that they were indeed being exploited.

**Sanity checking**

# Experimental results

Summary:

- the number of contracts exploited is non negligible (2% to 4%);
- However, it is important to note that the percentage of Ether exploited is an order of magnitude lower (<0.4%);
- **This indicates that exploited contracts are usually low-value.**

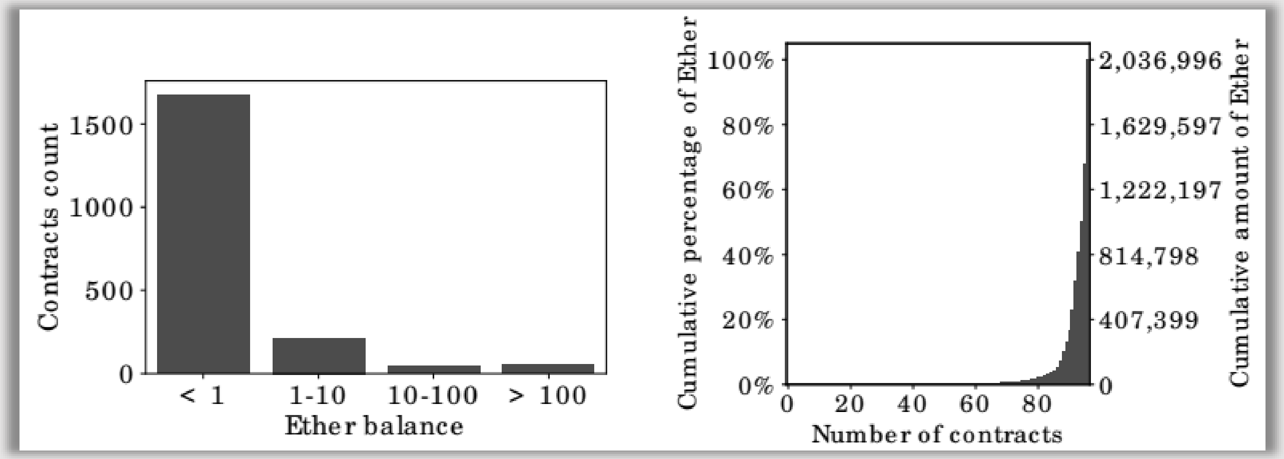
| Vulnerable | | | | Exploited contracts | | Exploited Ether | |
|---|---|---|---|---|---|---|---|
| **Vuln.** | **Vulnerable contracts** | **Total Ether at stake** | **Transactions analyzed** | **Contracts exploited** | **% of contracts exploited** | **Exploited Ether** | **% of Ether exploited** |
| RE | 4,337 | 1,518,067 | 457,073 | 116 | 2.68% | 6,076 | 0.40% |
| UE | 11,427 | 419,418 | 3,400,960 | 264 | 2.31% | 271.9 | 0.068% |
| LE | 7,285 | 1,416,086 | 10,660,066 | 0 | 0% | 0 | 0% |
| TO | 1,881 | 302,679 | 3,002,304 | 54 | 3.72% | 297.2 | 0.091% |
| IO | 2,492 | 602,980 | 1,295,913 | 62 | 2.49% | 1,842 | 0.31% |
| UA | 5,163 | 580,927 | 3,871,770 | 42 | 0.813% | 0 | 0% |
| **Total** | 23,327 | 3,124,433 | 20,241,730 | 463 | 1.98% | 8,487 | 0.27% |

**Figure 11:** Understanding the exploitation of potentially vulnerable contracts.

# Discussion

Some of the factors impacting the actual exploitation of smart contracts:

- The distribution of Ether among contracts (Top 10 SCs owns 95% ETH): the top contract is not exploited ➔ Not much Ether is actually at stack




- Manual inspection of **high value** contracts: The top 6 contracts seemed quite secure and the vulnerabilities flagged were definitely not exploitable.
- This dataset follows the same trend as the whole Ethereum blockchain: a very small amount of contracts hold most of the wealth.

# Conclusion

| | | | |
|---|---|---|---|
| 背景 | 智能合约不能修改 | 涌现出很多工具，意图检测合约漏洞 | 23327个合约被标记为有漏洞 |
| 问题 | 各个工具存在分歧 | 实际上有多少合约的漏洞真的被利用 | 通过利用漏洞，有多少以太币被盗 |
| 方法 | 分析交易Trace | Datalog表示交易，抽象、可扩展 | 人工分析实验结果，提出潜在因素 |
| 结论 | Vulnerable Does Not Imply Exploited | | |