Formal and smart contracts—or maybe not

Not a presentation of results but of terrible problems opportunities

Carlos E. Budde
Security group @ Dipartimento di Ingegneria e Scienza dell'Informazione

carlosesteban.budde@unitn.it

Smartitude — 23.10.2023
Blockchain... what? which? why?

Crypto market

7 Major Blockchains – Discover the Reviews

1. Ethereum
   - Ethereum Network
   - Ethereum Blockchain
   - Ethereum.org
   - Layer 1 Blockchain
   - Rating: 5/5

2. Binance
   - Binance Chain
   - Binance Smart Chain (BSC)
   - www.binance.org
   - Layer 1 Blockchain
   - Rating: 5/5

3. Bitcoin
   - Bitcoin Blockchain
   - www.bitcoin.org
   - Layer 1 Blockchain
   - Rating: 5/5

4. Solana
   - Solana Network
   - Solana.org
   - Layer 1 Blockchain
   - Rating: 5/5

5. Litecoin
   - Litecoin Blockchain
   - Litecoin.org
   - Layer 1 Blockchain
   - Rating: 5/5

6. Avalanche
   - Avalanche Network
   - www.avalanche.org
   - Layer 1 Blockchain
   - Rating: 5/5

7. Cardano
   - Cardano Network
   - Cardano.org
   - Layer 1 Blockchain
   - Rating: 5/5

Cryptocurrencies Overview

<table>
<thead>
<tr>
<th>Coin</th>
<th>Price</th>
<th>Change % 24h</th>
<th>Market cap</th>
<th>Volume in USD 24h</th>
<th>Circulating supply</th>
</tr>
</thead>
<tbody>
<tr>
<td>BTC</td>
<td>$28533.18</td>
<td>0.04%</td>
<td>$556.621B</td>
<td>$15.724B</td>
<td>$19.508M</td>
</tr>
<tr>
<td>ETH</td>
<td>$1564.08</td>
<td>-1.75%</td>
<td>$187.682B</td>
<td>$5.42B</td>
<td>$119.995M</td>
</tr>
<tr>
<td>Tether</td>
<td>$1.00162500</td>
<td>-0.09%</td>
<td>$83.667B</td>
<td>$39.617B</td>
<td>$83.531B</td>
</tr>
<tr>
<td>BNB</td>
<td>$211.32114500</td>
<td>-1.52%</td>
<td>$32.059B</td>
<td>$360.667M</td>
<td>$151.706M</td>
</tr>
<tr>
<td>XRP</td>
<td>$0.49395</td>
<td>-0.68%</td>
<td>$26.397B</td>
<td>$783.996M</td>
<td>$53.441B</td>
</tr>
<tr>
<td>USD Coin</td>
<td>$1.000037</td>
<td>0.18%</td>
<td>$25.21B</td>
<td>$3.083B</td>
<td>$25.209B</td>
</tr>
<tr>
<td>Solana</td>
<td>$24.1310</td>
<td>0.38%</td>
<td>$10.046B</td>
<td>$542.831M</td>
<td>$416.321M</td>
</tr>
<tr>
<td>Cardano</td>
<td>$0.246644</td>
<td>-1.43%</td>
<td>$8.61B</td>
<td>$135.898M</td>
<td>$34.91B</td>
</tr>
<tr>
<td>Dogecoin</td>
<td>$0.0589201</td>
<td>-1.79%</td>
<td>$8.342B</td>
<td>$159.159M</td>
<td>$141.5858</td>
</tr>
</tbody>
</table>
Blockchain... what? which? why?

7 Major Blockchains – Discover the Reviews

1. **ethereum**
   - Ethereum Network - Ethereum.org
   - Layer 1 Blockchain
   - Rank: 5.57

2. **BNB Chain**
   - Binance Chain
   - BNB.org
   - Rank: 5.57

3. **bitcoin**
   - Bitcoin Blockchain
   - Bitcoin.org
   - Rank: 5.57

4. **Solana Network**
   - Solana Network
   - Solana.org
   - Rank: 9.29

5. **litecoin**
   - Litecoin Network
   - LTC.org
   - Rank: 9.29

6. **Avalanche Network**
   - Avalanche Network
   - Avesta.org
   - Rank: 9.43

7. **Cardano Network**
   - Cardano.org
   - Rank: 4.43

---

Ethereum Total Value Locked

**Overview**

**Total value locked**

<table>
<thead>
<tr>
<th>Coin</th>
<th>Total value locked rank</th>
<th>DeFi coins</th>
</tr>
</thead>
<tbody>
<tr>
<td>BTC</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>ETH</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>USDT</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>BNB</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>XRP</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>XDC</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>SOL</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>ADA</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>DOGE</td>
<td>9</td>
<td></td>
</tr>
</tbody>
</table>

---

**Performance**

<table>
<thead>
<tr>
<th>Coin</th>
<th>Volume (USD)</th>
<th>TVL (USD)</th>
<th>Market Cap (USD)</th>
</tr>
</thead>
<tbody>
<tr>
<td>BTC</td>
<td>$65,888,888</td>
<td>$63,141,388</td>
<td>$7,678,123</td>
</tr>
<tr>
<td>ETH</td>
<td>$3,248,488</td>
<td>$3,248,488</td>
<td>$1,706,088</td>
</tr>
<tr>
<td>USDT</td>
<td>$1,706,088</td>
<td>$1,706,088</td>
<td>$1,706,088</td>
</tr>
<tr>
<td>BNB</td>
<td>$4,16,008</td>
<td>$4,16,008</td>
<td>$4,16,008</td>
</tr>
<tr>
<td>XRP</td>
<td>$9,719,719</td>
<td>$9,719,719</td>
<td>$9,719,719</td>
</tr>
<tr>
<td>XDC</td>
<td>$5,35,353</td>
<td>$5,35,353</td>
<td>$5,35,353</td>
</tr>
<tr>
<td>SOL</td>
<td>$5,00,005</td>
<td>$5,00,005</td>
<td>$5,00,005</td>
</tr>
<tr>
<td>ADA</td>
<td>$5,54,544</td>
<td>$5,54,544</td>
<td>$5,54,544</td>
</tr>
<tr>
<td>DOGE</td>
<td>$6,52,526</td>
<td>$6,52,526</td>
<td>$6,52,526</td>
</tr>
</tbody>
</table>

---

**Valuation**

<table>
<thead>
<tr>
<th>Coin</th>
<th>Market Cap (USD)</th>
<th>TVL (USD)</th>
<th>Total Value Locked</th>
</tr>
</thead>
<tbody>
<tr>
<td>BTC</td>
<td>$7,678,123</td>
<td>$63,141,388</td>
<td>$65,888,888</td>
</tr>
<tr>
<td>ETH</td>
<td>$1,706,088</td>
<td>$3,248,488</td>
<td>$3,248,488</td>
</tr>
<tr>
<td>USDT</td>
<td>$1,706,088</td>
<td>$1,706,088</td>
<td>$1,706,088</td>
</tr>
<tr>
<td>BNB</td>
<td>$4,16,008</td>
<td>$4,16,008</td>
<td>$4,16,008</td>
</tr>
<tr>
<td>XRP</td>
<td>$9,719,719</td>
<td>$9,719,719</td>
<td>$9,719,719</td>
</tr>
<tr>
<td>XDC</td>
<td>$5,35,353</td>
<td>$5,35,353</td>
<td>$5,35,353</td>
</tr>
<tr>
<td>SOL</td>
<td>$5,00,005</td>
<td>$5,00,005</td>
<td>$5,00,005</td>
</tr>
<tr>
<td>ADA</td>
<td>$5,54,544</td>
<td>$5,54,544</td>
<td>$5,54,544</td>
</tr>
<tr>
<td>DOGE</td>
<td>$6,52,526</td>
<td>$6,52,526</td>
<td>$6,52,526</td>
</tr>
</tbody>
</table>
Ethereum’s Blockchain Smart Contracts (BSC) in Solidity

A common misconception is that developers must write smart contracts in order to build on Ethereum. This is false. One of the beauties of the Ethereum community is that you’re able to participate in just about any project.

Ethereum and its community embrace open source. You can find projects - client implementations, APIs, development frameworks, etc. - in a wide variety of languages.

CHOOSE YOUR LANGUAGE

Select your programming language of choice to find projects, resources and communities:

- Ethereum for Dart developers
- Ethereum for Delphi developers
- Ethereum for .NET developers
- Ethereum for Go developers
- Ethereum for Java developers

https://etherscan.io/contractsVerified/

https://ethereum.org/en/developers/docs/programming-languages/
Common vulnerabilities

Security vulnerabilities in BSC written in Solidity

execution of (Solidity) BSC in Ethereum
Common vulnerabilities

Security vulnerabilities in BSC written in Solidity

- Reentrancy
- Access Control
- Arithmetic
- Unchecked external call
- Denial of Service
- Bad Randomness
- Front Running
- Time Manipulation
- Short Addresses

- Bad randomness
- Denial of service
- Forced Ether reception
- HoneyPots
- Incorrect interface
- Integer overflow
- Race condition
- Reentrancy
- Unchecked external call
- Unprotected function
- Variable shadowing
- Wrong constructor name

https://dasp.co/
https://github.com/crytic/not-so-smart-contracts
Unchecked external call

Certain Solidity operations known as “external calls”, require the developer to check that the operation succeeded—in contrast to operations which throw an exception on failure.

*If an external call fails, the contract will continue execution “as if the call succeeded.”*
Unchecked external call

Certain Solidity operations known as “external calls”, require the developer to check that the operation succeeded—in contrast to operations which throw an exception on failure.

*If an external call fails, the contract will continue execution “as if the call succeeded.”*
Unchecked external call

```solidity
pragma solidity ^0.4.24;

/**
 * Easy Investment 25% Contract
 * - GAIN 25% PER 24 HOURS (every 5900 blocks)
 * - NO COMMISSION on your investment (every ether stays on contract's balance)
 * - NO FEES are collected by the owner, in fact, there is no owner at all (just look at the code)
 *
 * How to use:
 * 1. Send any amount of ether to make an investment
 * 2a. Claim your profit by sending 0 ether transaction (every day, every week, i don't care... OR:
 * 2b. Send more ether to reinvest AND get your profit at the same time
 *
 * RECOMMENDED GAS LIMIT: 70000
 * RECOMMENDED GAS PRICE: https://ethgasstation.info/
 *
 * Contract reviewed and approved by pros!
 */

contract EasyInvest25 {
    address owner;
    function EasyInvest25 () { owner = msg.sender; }
    mapping (address => uint256) invested; // records amounts invested
    mapping (address => uint256) atBlock; // records blocks at which investments were made
    function() external payable {...} // this function called every time anyone sends a transaction to this contract
}
```

https://etherscan.io/address/0x06faa4d8157ba45baf2da5e7d02384225948d54f#code
**Example 1’**

Unchecked external call

```solidity
pragma solidity ^0.4.24;

/**
 * Easy Investment 25% Contract
 * - GAIN 25% PER 24 HOURS (every 5900 blocks)
 * - NO COMMISSION on your investment (every ether stays on contract's balance)
 * - NO FEES are collected by the owner, in fact, there is no owner at all (just look at the code)
 *
* How to use:
* 1. Send any amount of ether to make an investment
* 2a. Claim your profit by sending 0 ether transaction (every day, every week, i don't care… OR:
* 2b. Send more ether to reinvest AND get your profit at the same time
*
* RECOMMENDED GAS LIMIT: 70000
* RECOMMENDED GAS PRICE: https://ethgasstation.info/
*
* Contract reviewed and approved by pros!
*/

contract EasyInvest25 {
    address owner;
    function EasyInvest25 () { owner = msg.sender; }
    mapping (address => uint256) invested; // records amounts invested
    mapping (address => uint256) atBlock; // records blocks at which investments were made

    function() external payable {
        // this function called every time anyone sends a transaction to this contract
        // if sender (aka YOU) is invested more than 0 ether
        if (invested[msg.sender] != 0) {
            // calculate profit amount as such:
            address kashout = msg.sender;
            // amount = (amount invested) * 25% * (blocks since last transaction) / 5900
            // 5900 is an average block count per day produced by Ethereum blockchain
            uint256 getout = invested[msg.sender] * 25 / 100 * (block.number - atBlock[msg.sender]) / 5900;
            // send calculated amount of ether directly to sender (aka YOU)
            kashout.send(getout);
        }
        // record block number and invested amount (msg.value) of this transaction
        atBlock[msg.sender] = block.number;
        invested[msg.sender] += msg.value;
    }
}
```

[https://etherscan.io/address/0x06faa4d8157ba45baf2da5e7d02384225948d54f#code](https://etherscan.io/address/0x06faa4d8157ba45baf2da5e7d02384225948d54f#code)
Denial of Service

Denial of service is deadly in the world of Ethereum: while other types of applications can eventually recover,

smart contracts can be taken offline forever by just one of these attacks.

// Caller decides who will be rewarded by next call to function.
// Passing a very large _largestWinner value can make the
// *** next call infeasible *** due to gas limitations in Ethereum.

function selectNextWinners(uint256 _largestWinner) {
    for (uint256 i = 0; i < largestWinner, i++) {
        // heavy code, such gas, wow
    }
    largestWinner = _largestWinner;
}

if largestWinner ≠ 0...
Denial of Service

Parity Multisig Hacked. Again

Tony Kent - Follow
Published in ChainCloud company blog - 3 min read - Nov 8, 2017

Yesterday, Parity Multisig Wallet was hacked again:
https://paritytech.io/blog/security-alert.html

“This means that currently no funds can be moved out of the [ANY Parity] multi-sig wallets”

A lot of people/companies/ICOs are using Parity-generated multisig wallets. About $300M is frozen and (probably) lost forever.

Disclaimer: I lost little money (about $1000) but my friends lost about $300K.

Who hacked it?
Some guy with a nickname @devops199 (not a member of the Parity team)

All legit execution (perhaps should be “access control”)
Poor guy even did it accidentally!

How @devops199 hacked it?

1. All Parity Multisig wallets use single library at
   0x863DF6BFA4469F3ead0BEB89F2AAE51c91A907b4

2. Library contract was not initialized properly. That allowed anyone to become its owner and selfdestruct it.

3. @devops199 “accidentally” called initWallet() method to own the library
   https://etherscan.io
   /tx/0x05f71e1b2cb4f03e547739db15d080fd30c989eda04d37ce6264e586e0722c9

4. @devops199 “accidentally” called kill() method to selfdestruct it
   https://etherscan.io
   /tx/0x47f7cf7a5e671884629c93b368cb18f58a993f4b19c2a53a8662e3f1482f690

5. As a result, ALL Parity multisig wallets became useless. If you had any funds or tokens in the Parity multisig -> they are frozen forever (not yet an official position of Parity or Ethereum team, but mine) and you won’t be able to withdraw anything out of it.

https://medium.com/chain-cloud-company-blog/parity-multisig-hack-again-b46771eaa838
Example 2’

Denial of Service

Parity Multisig Hacked. Again

Yesterday, Parity Multisig Wallet was hacked again.

https://paritytech.io/blog/security-alert.html

“This means that currently no funds can be moved out of the [All multi-sig wallets]"

A lot of people/companies/ICOs are using Parity-generated multisig wallets. About $300M is frozen and (probably) lost forever.

Disclaimer: I lost little money (about $1000) but my friends lost about $300K.

Who hacked it?

Some guy with a nickname @devops199 (not a member of the Parity team)

All legit execution (perhaps should be “access control”)

devops199 @devops199
@AnthonyAkentiev most of my kills on contracts are failed... i though this one too because parity is a very big org..

Francisco Giordano @frangio
@devops199 sorry you’re going through this. i believe you’re innocent but you should probably get a lawyer

Anthony Akentiev @AnthonyAkentiev
@alathon I think that person that IS CALLING initWallet with parameters and then kill methods should be responsible for what he did.

@devops199 Why didn’t you contacted Parity when you found that initWallet finished with no exception? You "accidentally" called kill? })

devops199 @devops199
bye

5. As a result, ALL Parity multisig wallets became useless. If you had any funds or tokens in the Parity multisig -> they are frozen forever (not yet an official position of Parity or Ethereum team, but mine) and you won’t be able to withdraw anything out of it.

https://medium.com/chain-cloud-company-blog/parity-multisig-hack-again-b46771eaa838
Is all lost?
State-of-the-art in security for BSC

- Symbolic execution
- Formal methods
- Fuzz testing
- Deep learning (ML)

Static / dynamic code analysis:
- Build Control Flow Graph (CFG)
- Variables inputs as symbolic expressions in CFG
- Symbolic path has condition over those expressions
- Feed full thing to SMT solver, e.g. Z3
- Profit $$

```solidity
function selectNextWinners(uint256 _largestWinner) {
  for (uint256 i = 0; i < largestWinner; i++) {
    // heavy code, such gas, wow
    /*...
    */
    largestWinner = _largestWinner;
  }
}
```

```solidity
// if sender (YOU!) is invested more than 0 ether
if (invested[msg.sender] != 0) {
  address kashout = msg.sender; // calculate profit amount as such:
  uint256 getout = invested[msg.sender] * 25/100 * (block.number-atBlock[msg.sender]) / 5900;
  kashout.send(getout); // send calculated amount of ether directly to sender (YOU!)
}
```
State-of-the-art in security for BSC

- Symbolic execution
- Formal methods
- Fuzz testing
- Deep learning (ML)

Theorem proving & model checking (mainly)
- Create specification/model of desired behaviour
- Create model available implementation
- Prove/check whether current implementation “refines”/“simulates” the specification
- Some flavours:
  - Formal code semantics (denotational, small step, ...)
  - Abstract interpretation e.g. via function decorations
  - EVM bytecode to Prolog (!)
  - Etc.

Correctness
- Mathematically guaranteed!
- Pretty hard (if not impossible) to automate
State-of-the-art in security for BSC

- Symbolic execution
- Formal methods
- Fuzz testing
- Deep learning (ML)

Generate invalid input and monitor execution
State-of-the-art in security for BSC

- Symbolic execution
- Formal methods
- Fuzz testing
- Deep learning (ML)

Usual arsenal of black-box methods, now with BSC

- Supervised-learn bytecode of buggy contracts
- Detect fishy patterns in source code
- ...

Write an exploit for this contract, that transfers ETH to address 0x863DF6BFa4...
Security-enhancing tools

What about implementing those approaches?

<table>
<thead>
<tr>
<th>#</th>
<th>Tools</th>
<th>Tool URLs</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>E-EVM [33]</td>
<td><a href="https://github.com/pisocrob/E-EVM">https://github.com/pisocrob/E-EVM</a></td>
</tr>
<tr>
<td>3</td>
<td>Echidna</td>
<td><a href="https://github.com/cryptic/echidna">https://github.com/cryptic/echidna</a></td>
</tr>
<tr>
<td>4</td>
<td>Erays [44]</td>
<td><a href="https://github.com/teammsrg/erays">https://github.com/teammsrg/erays</a></td>
</tr>
<tr>
<td>5</td>
<td>Ether [26]</td>
<td>N/A</td>
</tr>
<tr>
<td>6</td>
<td>Ethersplay</td>
<td><a href="https://github.com/cryptic/ethersplay">https://github.com/cryptic/ethersplay</a></td>
</tr>
<tr>
<td>7</td>
<td>EtherTrust [19]</td>
<td><a href="https://www.netidee.at/ethertrust">https://www.netidee.at/ethertrust</a></td>
</tr>
<tr>
<td>8</td>
<td>EthIR [1]</td>
<td><a href="https://github.com/costa-group/EthIR">https://github.com/costa-group/EthIR</a></td>
</tr>
<tr>
<td>9</td>
<td>FSolidM [28]</td>
<td><a href="https://github.com/anmavrid/smart-contracts">https://github.com/anmavrid/smart-contracts</a></td>
</tr>
<tr>
<td>10</td>
<td>Gasper [9]</td>
<td>N/A</td>
</tr>
<tr>
<td>11</td>
<td>HoneyBadger [41]</td>
<td><a href="https://github.com/christofoforses/HoneyBadger">https://github.com/christofoforses/HoneyBadger</a></td>
</tr>
<tr>
<td>12</td>
<td>KEVM [21]</td>
<td><a href="https://github.com/kframework/evm-semantics">https://github.com/kframework/evm-semantics</a></td>
</tr>
<tr>
<td>13</td>
<td>MadMax</td>
<td><a href="https://github.com/nevillegrech/MadMax">https://github.com/nevillegrech/MadMax</a></td>
</tr>
<tr>
<td>16</td>
<td>Mythril [31]</td>
<td><a href="https://github.com/ConsenSys/mythril-class">https://github.com/ConsenSys/mythril-class</a></td>
</tr>
<tr>
<td>17</td>
<td>Octopus</td>
<td><a href="https://github.com/quoscient/octopus">https://github.com/quoscient/octopus</a></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>#</th>
<th>Tools</th>
<th>Tool URLs</th>
</tr>
</thead>
<tbody>
<tr>
<td>18</td>
<td>Osiris [40]</td>
<td><a href="https://github.com/christofoforses/Osiris">https://github.com/christofoforses/Osiris</a></td>
</tr>
<tr>
<td>19</td>
<td>Oyente [27]</td>
<td><a href="https://github.com/melonproject/oyente">https://github.com/melonproject/oyente</a></td>
</tr>
<tr>
<td>20</td>
<td>Porosity [38]</td>
<td><a href="https://github.com/comaeio/porosity">https://github.com/comaeio/porosity</a></td>
</tr>
<tr>
<td>21</td>
<td>rattle</td>
<td><a href="https://github.com/cryptic/rattle">https://github.com/cryptic/rattle</a></td>
</tr>
<tr>
<td>22</td>
<td>ReGuard [25]</td>
<td>N/A</td>
</tr>
<tr>
<td>23</td>
<td>Remix</td>
<td><a href="https://github.com/ethereum/remix">https://github.com/ethereum/remix</a></td>
</tr>
<tr>
<td>24</td>
<td>SASC [43]</td>
<td>N/A</td>
</tr>
<tr>
<td>25</td>
<td>sCompile [6]</td>
<td>N/A</td>
</tr>
<tr>
<td>26</td>
<td>Securify [42]</td>
<td><a href="https://github.com/eth-sri/securify">https://github.com/eth-sri/securify</a></td>
</tr>
<tr>
<td>29</td>
<td>Solgraph</td>
<td><a href="https://github.com/raineorshine/solgraph">https://github.com/raineorshine/solgraph</a></td>
</tr>
<tr>
<td>30</td>
<td>Solhint</td>
<td><a href="https://github.com/protofire/solhint">https://github.com/protofire/solhint</a></td>
</tr>
<tr>
<td>32</td>
<td>teEther [23]</td>
<td><a href="https://github.com/nescio007/teether">https://github.com/nescio007/teether</a></td>
</tr>
<tr>
<td>35</td>
<td>Zeus [22]</td>
<td>N/A</td>
</tr>
</tbody>
</table>

* Durieux et al.: "Empirical Review of Automated Analysis Tools on 47,587 Ethereum Smart Contracts" (ICSE 2020)
Has all been done?
Vulnerabilities in Solidity BSC

Collect (and classify) true-positive vulnerabilities

Table 3: Categories of vulnerabilities available in the dataset sb\textsuperscript{CURATED}. For each category, we provide a description, the level at which the attack can be mitigated, the number of contracts available within that category, and the total number of lines of code in the contracts of that category (computed using cloc 1.82).

<table>
<thead>
<tr>
<th>Category</th>
<th>Description</th>
<th>Level</th>
<th>Contracts</th>
<th>Vulns</th>
<th>LoC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access Control</td>
<td>Failure to use function modifiers or use of tx.origin</td>
<td>Solidity</td>
<td>17</td>
<td>19</td>
<td>899</td>
</tr>
<tr>
<td>Arithmetic</td>
<td>Integer over/underflows</td>
<td>Solidity</td>
<td>14</td>
<td>22</td>
<td>295</td>
</tr>
<tr>
<td>Bad Randomness</td>
<td>Malicious miner biases the outcome</td>
<td>Blockchain</td>
<td>8</td>
<td>31</td>
<td>1,079</td>
</tr>
<tr>
<td>Denial of service</td>
<td>The contract is overwhelmed with time-consuming computations</td>
<td>Solidity</td>
<td>6</td>
<td>7</td>
<td>177</td>
</tr>
<tr>
<td>Front running</td>
<td>Two dependent transactions that invoke the same contract are included in one block</td>
<td>Blockchain</td>
<td>4</td>
<td>7</td>
<td>137</td>
</tr>
<tr>
<td>Reentrancy</td>
<td>Reentrant function calls make a contract to behave in an unexpected way</td>
<td>Solidity</td>
<td>7</td>
<td>8</td>
<td>778</td>
</tr>
<tr>
<td>Short addresses</td>
<td>EVM itself accepts incorrectly padded arguments</td>
<td>EVM</td>
<td>1</td>
<td>1</td>
<td>18</td>
</tr>
<tr>
<td>Time manipulation</td>
<td>The timestamp of the block is manipulated by the miner</td>
<td>Blockchain</td>
<td>4</td>
<td>5</td>
<td>76</td>
</tr>
<tr>
<td>Unchecked low level calls</td>
<td>call(), callcode(), delegatecall() or send() fails and it is not checked</td>
<td>Solidity</td>
<td>5</td>
<td>12</td>
<td>225</td>
</tr>
<tr>
<td>Unknown Unknowns</td>
<td>Vulnerabilities not identified in DASP 10</td>
<td>N/A</td>
<td>3</td>
<td>3</td>
<td>115</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td></td>
<td>69</td>
<td>115</td>
<td>3,799</td>
</tr>
</tbody>
</table>

* Durieux et al.: "Empirical Review of Automated Analysis Tools on 47,587 Ethereum Smart Contracts" (ICSE 2020)
Collect (and classify) **true-positive** vulnerabilities

<table>
<thead>
<tr>
<th>Category</th>
<th>Description</th>
<th>Solidity</th>
<th>Blockchain</th>
<th>EVM</th>
<th>Blockchains</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access Control</td>
<td>False positive</td>
<td></td>
<td>14</td>
<td>22</td>
<td>295</td>
</tr>
<tr>
<td>Arithmetic</td>
<td>Integer over/underflows</td>
<td></td>
<td>8</td>
<td>31</td>
<td>1,079</td>
</tr>
<tr>
<td>Bad Randomness</td>
<td>Malicious miner biases the outcome</td>
<td></td>
<td>6</td>
<td>7</td>
<td>177</td>
</tr>
<tr>
<td>Denial of service</td>
<td>The contract is overwhelmed with time-consuming computations</td>
<td></td>
<td>4</td>
<td>7</td>
<td>137</td>
</tr>
<tr>
<td>Front running</td>
<td>Two dependent transactions that invoke the same contract are included in one block</td>
<td></td>
<td>7</td>
<td>8</td>
<td>778</td>
</tr>
<tr>
<td>Reentrancy</td>
<td>Reentrant function calls make a contract to behave in an unexpected way</td>
<td></td>
<td>1</td>
<td>1</td>
<td>18</td>
</tr>
<tr>
<td>Short addresses</td>
<td>EVM itself accepts incorrectly padded arguments</td>
<td></td>
<td>4</td>
<td>5</td>
<td>76</td>
</tr>
<tr>
<td>Time manipulation</td>
<td>The timestamp of the block is manipulated by the miner</td>
<td></td>
<td>5</td>
<td>12</td>
<td>225</td>
</tr>
<tr>
<td>Unchecked low level calls</td>
<td>call(), callcode(), delegatecall() or send() fails and it is not checked</td>
<td></td>
<td>3</td>
<td>3</td>
<td>115</td>
</tr>
<tr>
<td>Unknown Unknowns</td>
<td>Vulnerabilities not identified in DASP 10</td>
<td>N/A</td>
<td>3</td>
<td>3</td>
<td>115</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td>69</td>
<td>115</td>
<td>3,799</td>
<td></td>
</tr>
</tbody>
</table>

* Durieux et al.: "Empirical Review of Automated Analysis Tools on 47,587 Ethereum Smart Contracts" (ICSE 2020)
**Table 3:** Categories of vulnerabilities available in the dataset \( sb^{CURATED} \). For each category, we provide a description, the level at which the attack can be mitigated, the number of contracts available within that category, and the total number of lines of code in the contracts of that category (computed using cloc 1.82).

<table>
<thead>
<tr>
<th>Category</th>
<th>Description</th>
<th>Level</th>
<th>Contracts</th>
<th>Vulns</th>
<th>LoC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access Control</td>
<td>Failure to use function modifiers or use of ( \text{tx.origin} )</td>
<td></td>
<td>19</td>
<td>899</td>
<td></td>
</tr>
<tr>
<td>Arithmetic</td>
<td>Integer over/underflows</td>
<td></td>
<td>22</td>
<td>295</td>
<td></td>
</tr>
<tr>
<td>Bad Randomness</td>
<td>Malicious miner biases the outcome</td>
<td></td>
<td>31</td>
<td>1,079</td>
<td></td>
</tr>
<tr>
<td>Denial of service</td>
<td></td>
<td></td>
<td>7</td>
<td>177</td>
<td></td>
</tr>
<tr>
<td>Front running</td>
<td>Two dependent transactions that invoke the same contract are included in one block</td>
<td></td>
<td>7</td>
<td>137</td>
<td></td>
</tr>
<tr>
<td>Reentrancy</td>
<td>Reentrant function calls make a contract to behave in an unexpected way</td>
<td></td>
<td>8</td>
<td>778</td>
<td></td>
</tr>
<tr>
<td>Short addresses</td>
<td>EVM itself accepts incorrectly padded arguments</td>
<td></td>
<td>1</td>
<td>18</td>
<td></td>
</tr>
<tr>
<td>Time manipulation</td>
<td>The timestamp of the block is manipulated by the miner</td>
<td></td>
<td>5</td>
<td>76</td>
<td></td>
</tr>
<tr>
<td>Unchecked low level</td>
<td></td>
<td></td>
<td>12</td>
<td>225</td>
<td></td>
</tr>
<tr>
<td>calls</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Unknown Unknowns</td>
<td>Vulnerabilities not identified in DASP 10</td>
<td></td>
<td>3</td>
<td>115</td>
<td></td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td></td>
<td>69</td>
<td>115</td>
<td>3,799</td>
</tr>
</tbody>
</table>

* Durieux et al.: "Empirical Review of Automated Analysis Tools on 47,587 Ethereum Smart Contracts" (ICSE 2020)
Vulnerability introduction-discovery correlations

Relies on quantitative data like CVSS of CVEs
Security in BSC

Do smart contracts really need Turing completeness?

function selectNextWinners(uint256 _largestWinner) {
    for (uint256 i = 0; i < _largestWinner, i++) {
        // heavy code, such gas, wow
    }
    largestWinner = _largestWinner;
}

do {
    break(havoc);
} while (still_works);

But now they have already tasted blood...

//=== @requires { @GAS_LIMIT > 2100*_largestWinner; }
function selectNextWinners(uint256 _largestWinner) {
    // !@if (@GAS_LEFT < 2100) { throw(); }
    for (uint256 i = 0; i < _largestWinner, i++) {
        // heavy code, such gas, wow
    }
    largestWinner = _largestWinner;
}
Formal and smart contracts—or maybe not

Not a presentation of results but of terrible problems opportunities

Carlos E. Budde
Security group @ DISI

carlosesteban.budde@unitn.it

Smartitude