Blockchain and Smart Contract Security Design & Code Audit

Mukhtar Serikbayev

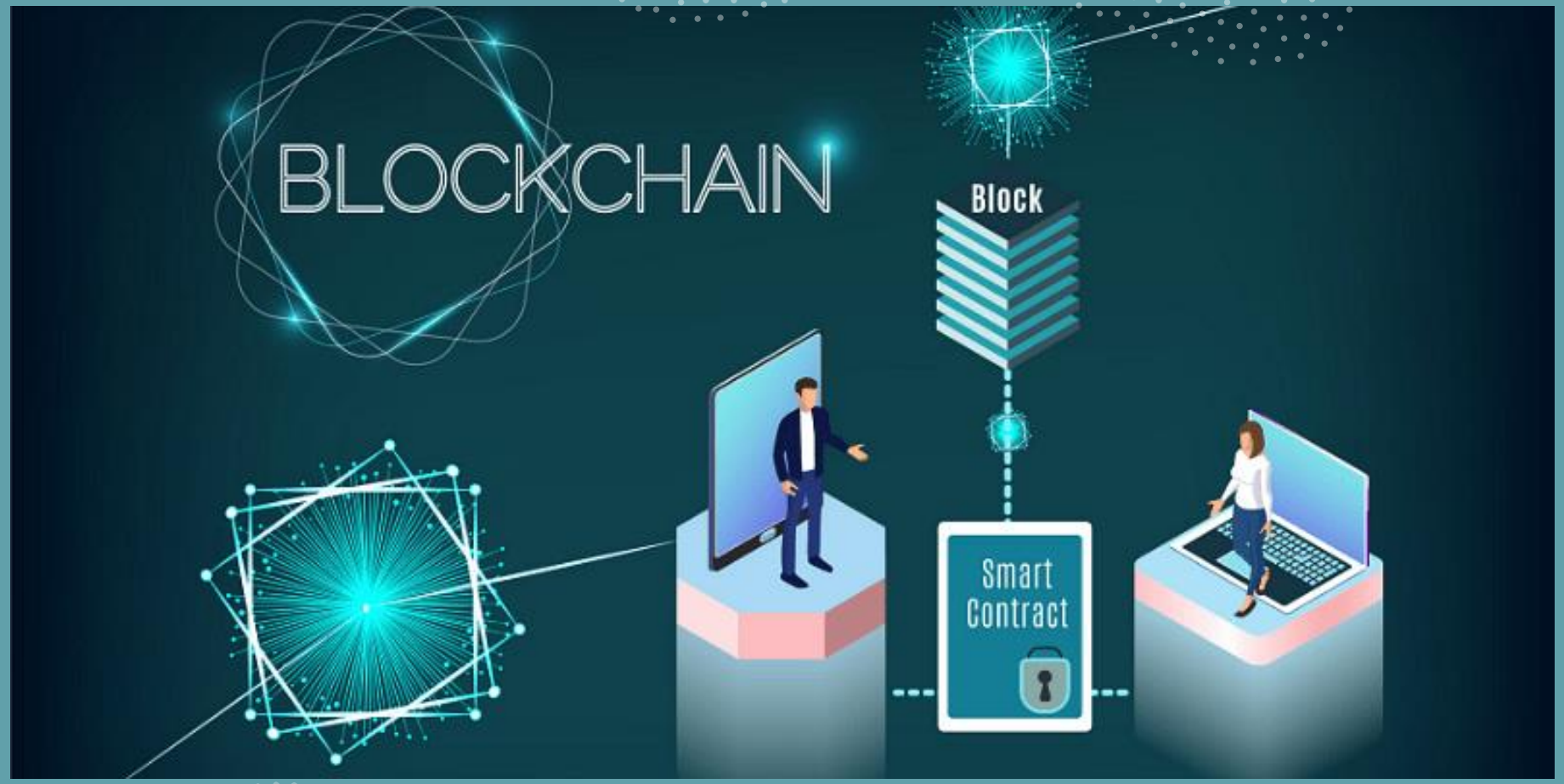
Agenda

- About me
- Blockchain Technology & Smart Contract 101
- Smart Contracts vs Traditional software development
  - Code Sample
  - Ethereum incentive model
- Front-running vulnerability
  - Commitment Scheme
- Code Auditing
  - Code Auditing tools
  - Real world code audit exercise
- Smart Contract Vulnerabilities
- Resources
- References

#whoami

- Mukhtar Serikbayev
  - Offensive Security Consultant @ Help AG
  - Application Layer Security – Mobile, Web & Source Code Audit

Blockchain technology & Smart Contract 101



Smart Contracts:

New paradigm of software development

Simple Application, or is it?

How do you fetch an API endpoint?

Python:

```python
import urllib2
contents = urllib2.urlopen("http://example.com/foo/bar").read()
```

Solidity?

```
3  pragma solidity >=0.7.0 <0.9.0;
4  import requests;
5
6  contract sampleFetcher {
7
8      bytes32 public content;
9
10     function fetch() public {
11         content = requests.get("http://example.com/foo/bar");
12     }
13
14 }
```

Simple Application, or is it?

- Not executing at the exact same time
- Closed System
- All Deterministic execution (EVM) No random data
- Requires consensus on the final state to be valid

- Oracles supposed to be a solution for external data

GLOBAL BITCOIN NODES DISTRIBUTION
Reachable nodes as of Wed Sep 29 09:58:40 2021 +04.

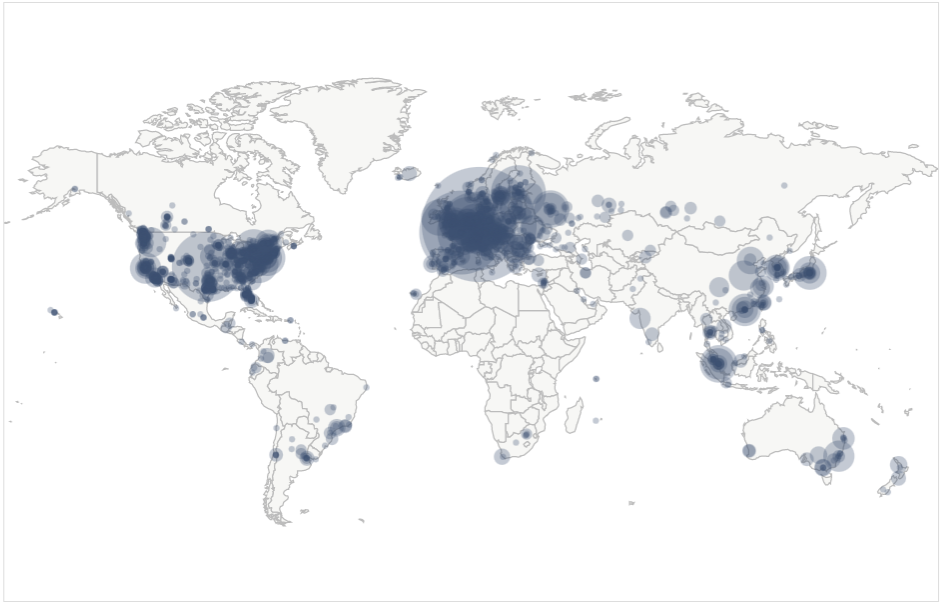
11792 NODES

24h | 90d | 1y

Top 10 countries with their respective number of reachable nodes are as follow.

RANK	COUNTRY	NODES
1	n/a	4412 (37.42%)
2	United States	1862 (15.79%)
3	Germany	1777 (15.07%)
4	France	547 (4.64%)
5	Netherlands	377 (3.20%)
6	Canada	321 (2.72%)
7	United Kingdom	253 (2.15%)
8	Russian Federation	193 (1.64%)
9	Finland	187 (1.59%)
10	China	136 (1.15%)

More (88) »



Map shows concentration of reachable Bitcoin nodes found in countries around the world.

LIVE MAP

Yet Another Simple Application, or is it?

How do you password protect an application?

- e.g. Password protected vault

- All Data in blockchain is public.
- No variable can store a secret

```
6  contract Vault {
7
8      bool public locked;
9      bytes32 private password;
10
11     function Vault(bytes32 _password) public {
12         locked = true;
13         password = _password;
14     }
15
16     function unlock(bytes32 _password) public {
17         if (password == _password) {
18             locked = false;
19         }
20     }
21 }
```

Yet Another Simple Application, or is it?

Password protected vault

Traditional security best practice:

Store the hash of the password

- Tip: Ethereum has a built-in secure hash function:

Kecceck256() ~= SHA3

```
23  contract Vault {
24
25      bool public locked;
26      bytes32 private hash;
27
28      function Vault(bytes32 _hash) public {
29          locked = true;
30          hash = _hash;
31      }
32
33      function unlock(bytes32 _password) public {
34          if (hash == keccak256(_password)) {
35              locked = false;
36          }
37      }
38  }
```

Yet Another Simple Application, or is it?

Password protected vault

that pays 100 Ether

```
23 ▾ contract Vault {
24
25       bool public locked;
26       bytes32 private hash;
27
28 ▾     function Vault(bytes32 _hash) public {
29           locked = true;
30           hash = _hash;
31       }
32
33 ▾     function unlock(bytes32 _password) public {
34 ▾         if (hash == keccak256(_password)) {
35               locked = false;
36               msg.sender.transfer(100 Ether)
37           }
38       }
39   }
```

Enter the Ethereum

Decentral, public, permission-less blockchain

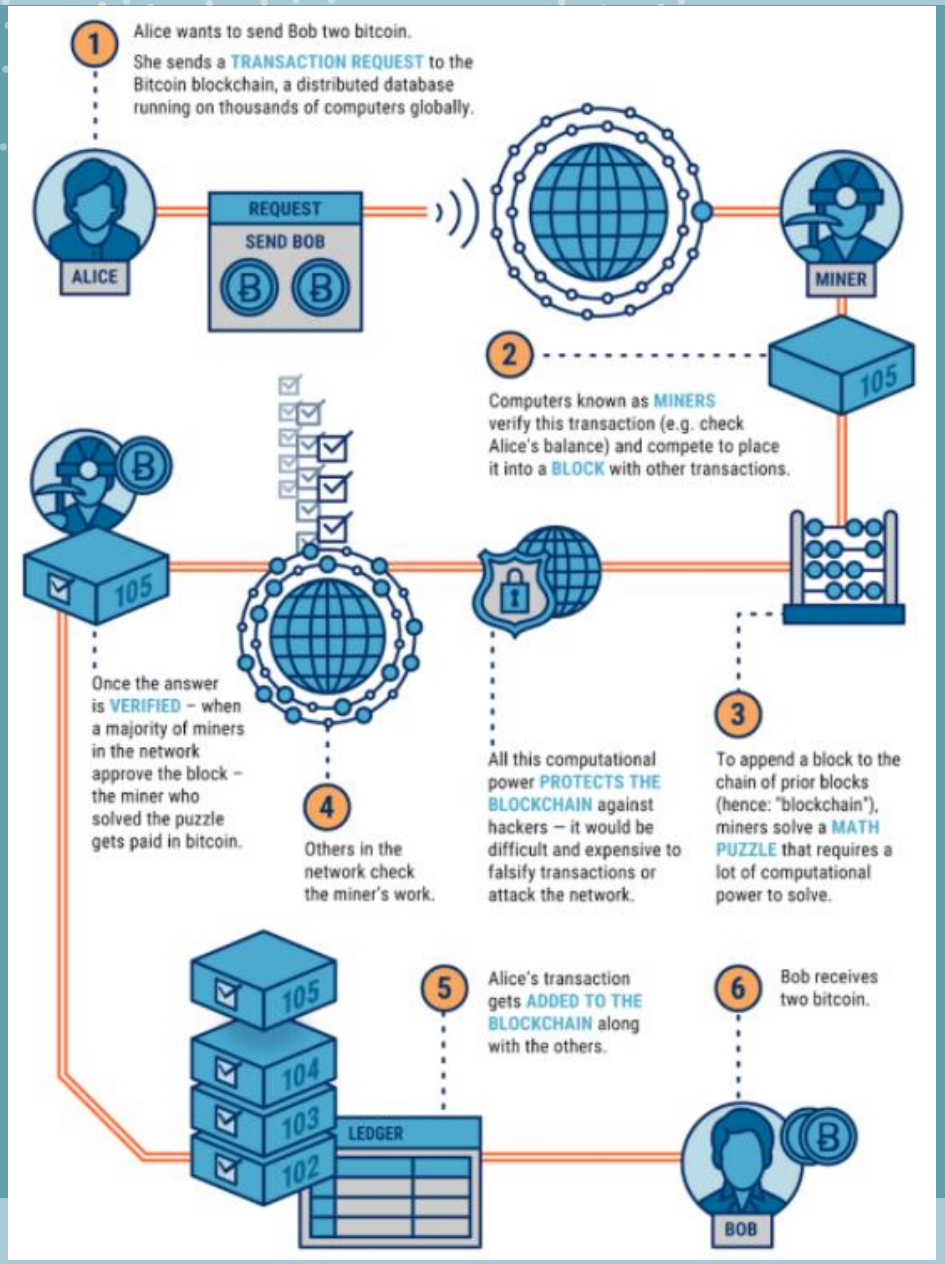
Technically different from cloud /online services

Enter the Ethereum

Request/Transactions have stages until they are final:

1. Locally **create** a transaction
2. **Broadcast** it to the decentral network
    a) Everyone else are doing the same
    b) Recent transactions are kept in the nodes' **mempool**

Enter the Ethereum



Ethereum Incentive Model

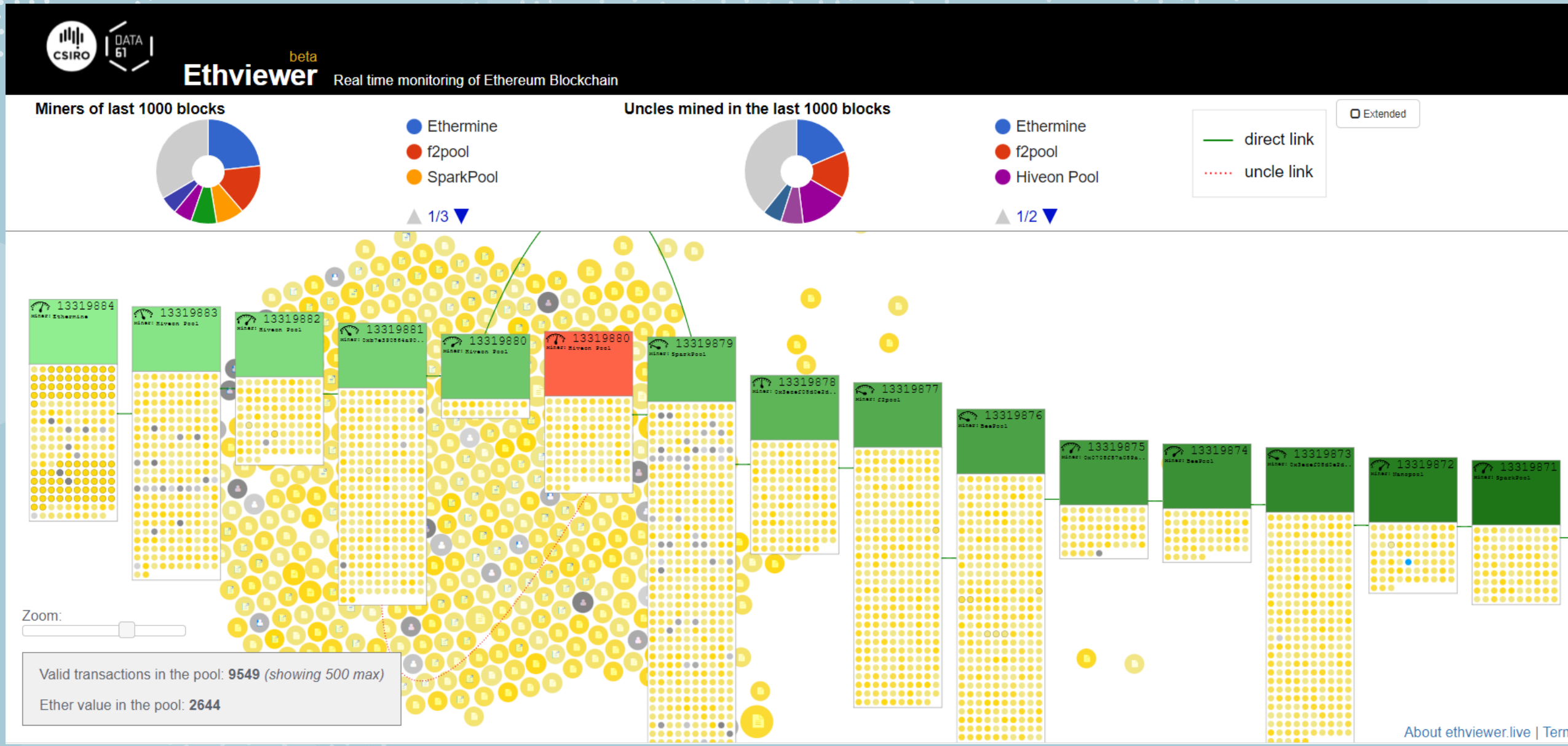
- **Blocks are scarce, block size is limited**
  - Computation, Storage, Network
- **Gas**: Unit of Payment for the computation
  - **Gas Cost**: The amount of gas required to execute the transaction
  - **GasPrice**: The price of each unit in Ether
  - **GasLimit**: The maximum amount willing to pay

- One can change their transaction priority by paying higher price ← Miners behave based on economic incentives

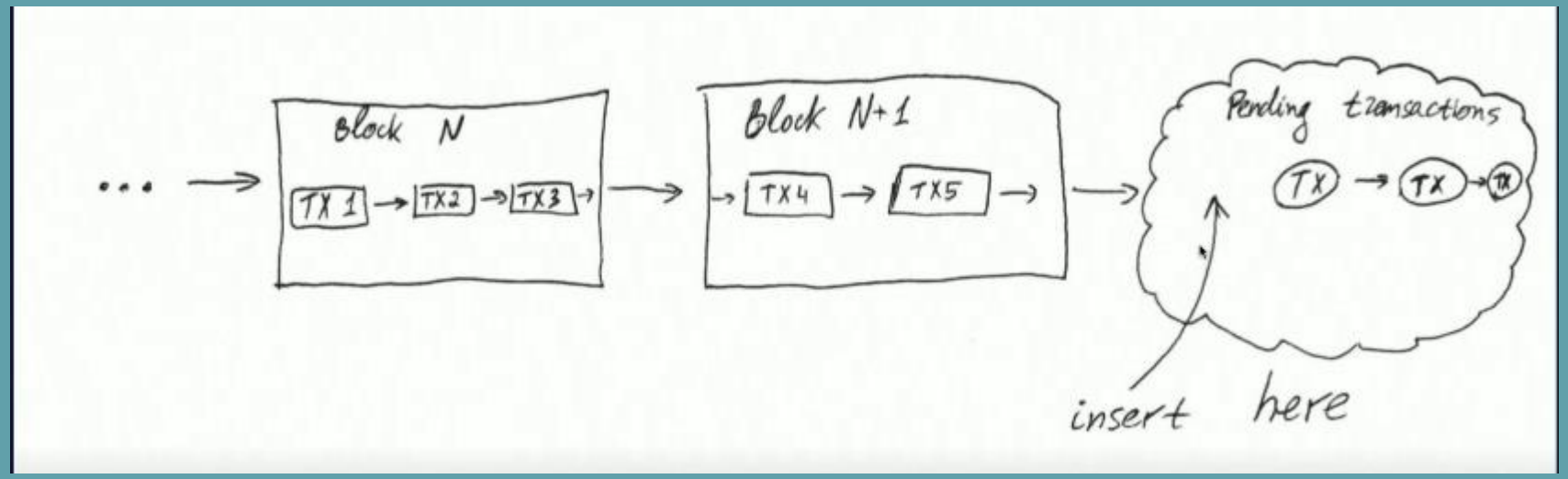
Front-running

(Traditional/Stock Market) Front-running is a course of action where an entity:

- Benefits from early access to market information about upcoming transactions and trades
  - Typically, because of a privileged position along the transmission of this information


CSIRO
DATA 61
beta
Ethviewer
Real time monitoring of Ethereum Blockchain
Miners of last 1000 blocks
Ethermine
f2pool
SparkPool
1/3
Uncles mined in the last 1000 blocks
Ethermine
f2pool
Hiveon Pool
1/2
Extended
direct link
uncle link
13319884
13319883
13319882
13319881
13319880
13319880
13319879
13319878
13319877
13319876
13319875
13319874
13319873
13319872
13319871
Zoom:
Valid transactions in the pool: 9549 (showing 500 max)
Ether value in the pool: 2644
About ethviewer.live | Term

Front-running in Blockchain



Yet Another Simple Application (cont...)

Password protected vault

that pays 100 Ether

Anyone can **front-run** **unlock(password)** transaction by sending it with higher **gasPrice** and **get 100 Ether**

```
23 ▾ contract Vault {
24
25       bool public locked;
26       bytes32 private hash;
27
28 ▾     function Vault(bytes32 _hash) public {
29           locked = true;
30           hash = _hash;
31       }
32
33 ▾     function unlock(bytes32 _password) public {
34 ▾         if (hash == keccak256(_password)) {
35               locked = false;
36               msg.sender.transfer(100 Ether)
37           }
38       }
39   }
```

Commitment scheme

1. Send **Commitment Transaction** with **hash of the data**

   (Grace Period, usually a few blocks)

2. Send **Reveal Transaction with the data**

Code Auditing

- Manual code review and in-depth understanding of the system and its documentations
- Security tools can help you find common misuse patterns and security hotspots
- Much more work and expertise needed than just running tools and writing reports
- It is becoming industry standard to hire external code auditors

Real world Application

- **Ethereum Name Service**
  - Human readable names to identify addresses on the Ethereum network
    - Similar to DNS for IP
      - E.g.
        - DNS: google.com --> 172.217.19.3
        - ENS: google.eth --> 0x57f1887a8bf19b14fc0df6fd9b2acc9af147ea85
    - On-chain transactions to purchase domains
        - Used to be Blind auctions

Real world Application

- **Ethereum Name Service**
  - 4th of July 2019, switched to “instant” domain purchase with rent-based model called EthRegisterar
    - **Holds funds** (rents) in the smart contract
    - Requires commit-reveal to prevent front-running or you will lose the domain you are interested in

Code Audit

- Fire up ***VSCode***
  - Make sure you have ***Solidity*** and ***Solidity Visual Auditor*** extension installed

Code Audit - ENS

- Download the code
  - http://bit.ly/ENSRegistrar

Open /contracts/ETHRegistrarController.sol

- Access Control and fund transfers are very critical
  - *onlyOwner(), isOwner() modifier*
  - *Address.transfer()*

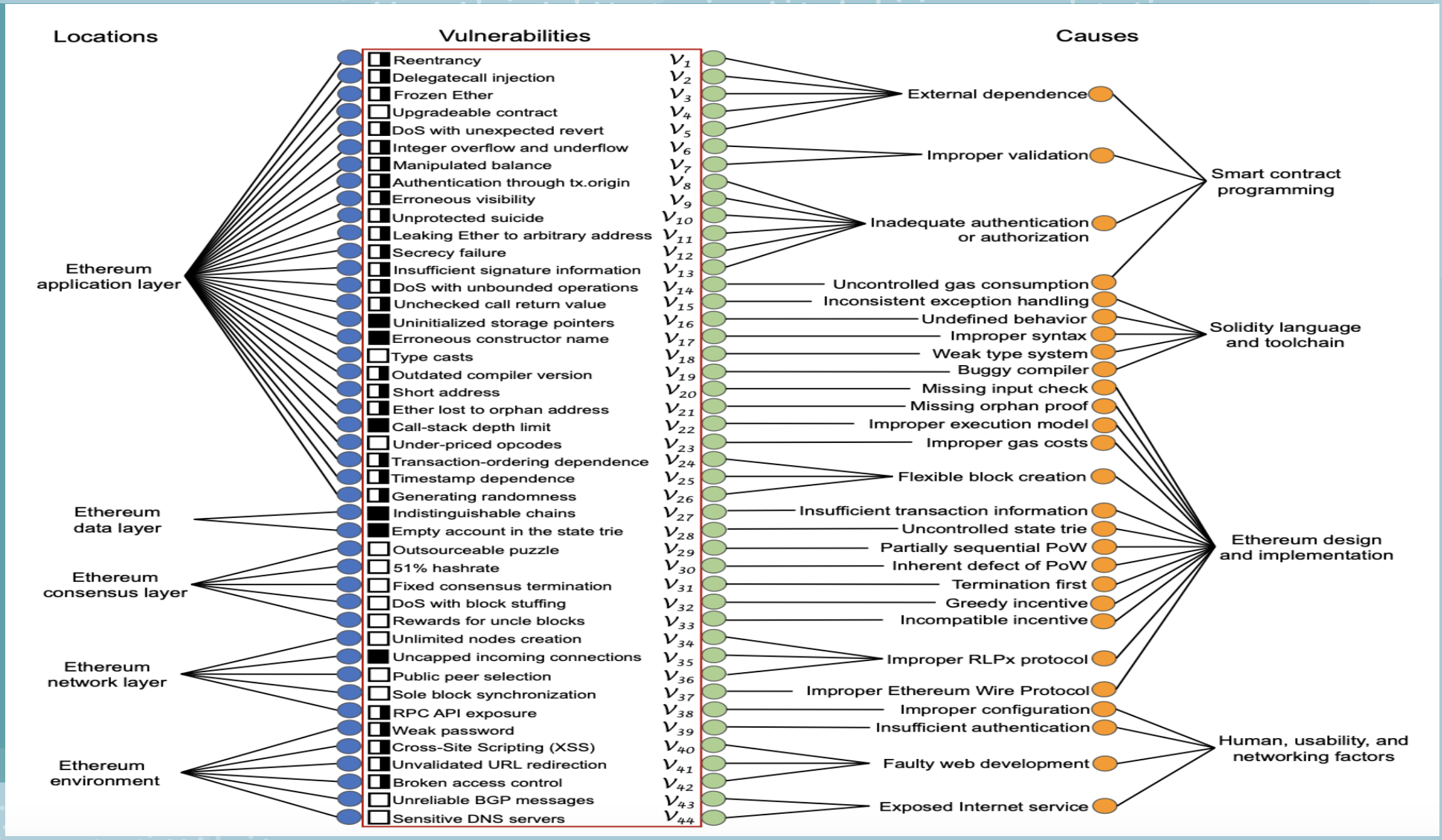
Code Audit - ENS

- ETHRegistrarController.register() is **vulnerable to front-running**
  - **Commitment** is not tied to a specific **owner**
  - **Attack Scenario:**
    i. **Alice** calls ***commit(makeCommitment("mydomain",<secret>)).***
    ii. 10 minutes later ***Alice*** sends a transaction to ***register("mydomain",Alice, …, <secret>)***
    iii. **Eve** **observes** this transaction in the transaction pool
    iv. **Eve** submits ***register("mydomain",Eve, …, <secret>)*** with a ***higher gas price*** and wins the race

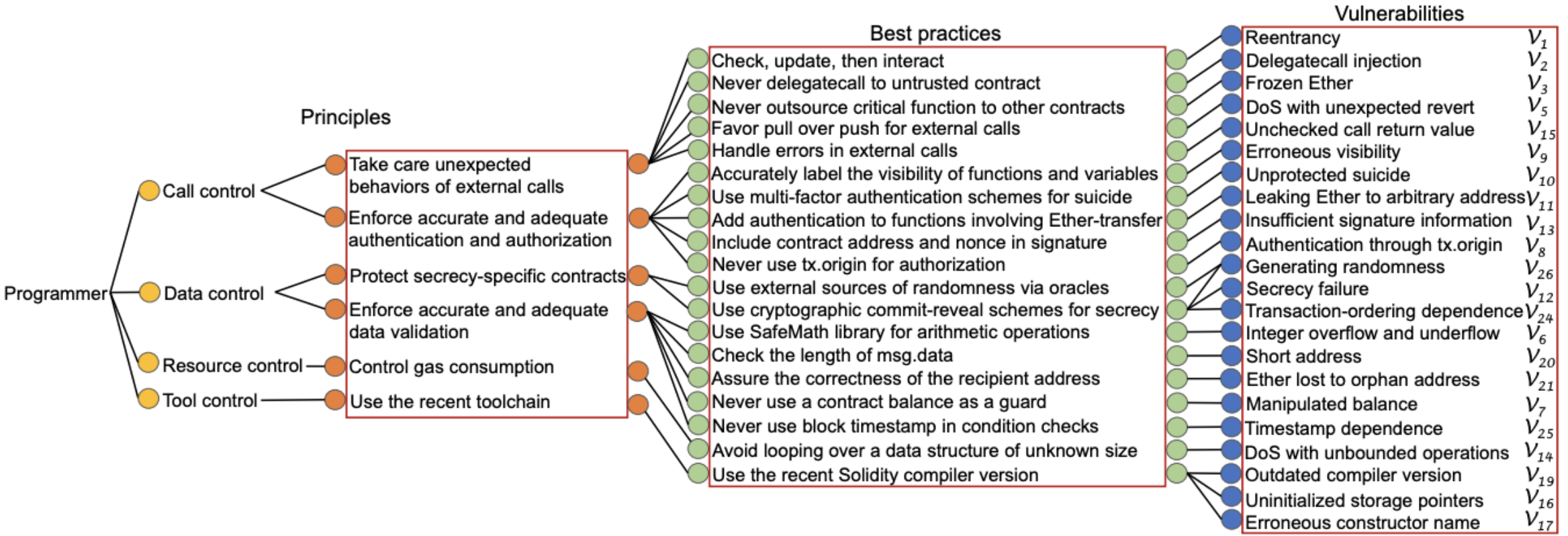
Smart contract Vulnerabilities

- Traditional common vulnerabilities
  - Integer Overflow/Underflow (Arithmetic Attacks)
  - Transaction Order Dependence (Race Conditions and Front Running Attacks)
  - Uninitialized storage/function (Parity Frozen Eth Hack)
- Smart Contract Specific vulnerabilities
  - Reentrancy (DAO)
  - Front Running
  - Trusted Actors (Trust & Key Compromise)
  - Upgradability
  - Unprotected Selfdestruct (Parity Attack)
  - Frozen Ether/Tokens
  - Timestamp Manipulation
  - No Randomness

Smart contract Vulnerabilities



Smart contract Vulnerabilities



Resources

- Useful code security/quality tools
  - Linter: ETHLINT, Solhint
  - Smart Contract Code Analysis: Myth/mithril, Slither, teEther, Oyente, Securify
  - Remix IDE plugins
  - Truffle tests
- Ethereum Smart Contract Security Best Practices
  - https://swcregistry.io/
- Read Public Smart Contract Audit Report
- Smart Contract Security Challenges
  - Capturetheether.com
  - https://www.damnvulnerabledefi.xyz/
  - https://iphelix.medium.com/damn-vulnerable-defi-setup-and-challenge-1-walkthrough-1ea16ea09709