

Blockchain & Smart Contract Attack Vectors

Jorden Seet



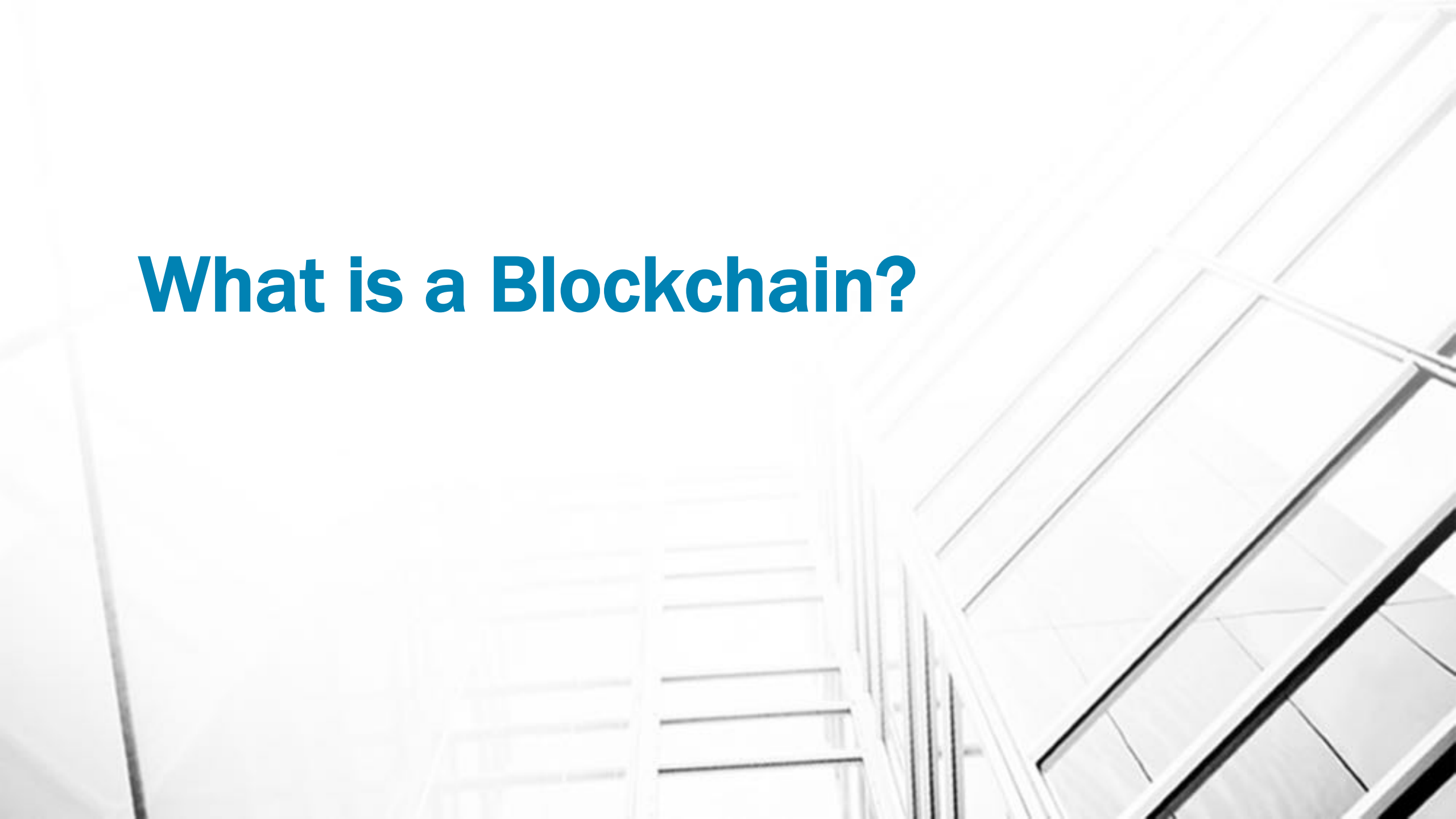
Who am I?

- Jorden Seet
- Year 3 Undergraduate, SMU, BSc Information Systems & Analytics double major
- Blockchain Teaching Assistant for SMU
- Passionate about Cybersecurity, from Pentesting to Cryptography to Network Topology
- Recently completed an internship at Cyber Security Agency of Singapore (CSA)
 - Penetration Testing Department

Why am I here?

- Blockchain is a buzzword
 - Foreign?
 - Unhackable?
- It's not that different!
- Compilation of various attack vectors
 - Usually, presentations focus on a single scope
- Take home something new
 - Smart Contract Attack Vectors
 - Consensus Mechanism Attack Vectors
 - Topology Attack Vectors

What is a Blockchain?



Blockchain is a special kind of Database

Centralized



Decentralized



Distributed Ledgers



Permissioned



Permissionless

Blockchain is a Distributed Ledger

Centralized



Decentralized



Distributed Ledgers



Permissioned



Permissionless

Benefits of Blockchain

- Data Integrity
 - Consensus rejects nodes with inconsistent data
 - Requires “hacking” into 51% or 33% of nodes to carry attacks
- High Availability
 - As long as one node is up, information is available
 - Hard to DDoS network when there is no single source
 - Distribution of bandwidth can also mitigate DDoS
- High Auditability
 - Information in blockchain is transparent to all nodes in network
 - All information is linked to each other

Smart Contracts



What are Smart Contracts?

- Cryptographic computer protocol that is designed to
 - Facilitate
 - Monitor
 - Verify
 - Enforce
- The performance of an agreement in an immutable and exact manner.
- Without third parties like
 - Lawyers
 - Auditors
 - Witnesses
 - Banks
 - Governments

Why do Smart Contracts need Blockchain?

- Smart Contracts require witnesses
 - Honest nodes operating (anonymously) in a distributed network
- Nodes may not necessarily be honest
- Smart Contracts need a Byzantine Fault Tolerant environment
 - Consensus must be reached amidst failure of nodes
 - Ensured (to a higher degree) through the Blockchain

Smart Contract Attack Vectors

Integer Underflow/Overflow

- When one puts in a value that exceeds the limit, the value becomes something else.
- Uints have a max value of $2^{256} - 1$, in Solidity it wraps after max value to 0

```
function batchTransfer(address[] _receivers, uint256 _value) public whenNotPaused returns (bool) {
    uint cnt = _receivers.length;
    uint256 amount = uint256(cnt) * _value;
    require(cnt > 0 && cnt <= 20);
    require(_value > 0 && balances[msg.sender] >= amount);

    balances[msg.sender] = balances[msg.sender].sub(amount);
    for (uint i = 0; i < cnt; i++) {
        balances[_receivers[i]] = balances[_receivers[i]].add(_value);
        Transfer(msg.sender, _receivers[i], _value);
    }
    return true;
}
```

What happens if we call $(2^{256})/2 = 8 * 16^{63}$ with 2 addresses?

Timestamp Dependence

- Dangerous to attempt pseudo-random number generation via block timestamps
 - Deterministic, attackers can determine the “random” variable

```
contract CoinFlip {
    uint256 public consecutiveWins;
    uint256 lastHash;
    uint256 FACTOR = 57896044618658097711785492504343953926634992332820282019728792003956564819968;

    function CoinFlip() public {
        consecutiveWins = 0;
    }

    function flip(bool _guess) public returns (bool) {
        uint256 blockValue = uint256(block.blockhash(block.number-1));

        if (lastHash == blockValue) {
            revert();
        }

        lastHash = blockValue;
        uint256 coinFlip = blockValue / FACTOR;
        bool side = coinFlip == 1 ? true : false;
```

Denial of Service

- Smart contracts can indicate if they are “payable”
 - Non-payable functions/contracts cannot receive cryptocurrency

```
3 contract Auction {  
4     address highestBidder;  
5     uint highestBid;  
6  
7     function bid() {  
8         if (msg.value < highestBid) throw;  
9  
10        if (highestBidder != 0) {  
11            highestBidder.transfer(highestBid);  
12        }  
13  
14        highestBidder = msg.sender;  
15        highestBid = msg.value;  
16    }  
17 }
```


Reentrancy Attack

```
function payout() public payable {  
  
    checkPermissions(msg.sender);  
  
    if (game.originator.status == STATUS_TIE && game.taker.status == STATUS_TIE) {  
        game.originator.addr.transfer(game.betAmount);  
        game.taker.addr.transfer(game.betAmount);  
    } else {  
        if (game.originator.status == STATUS_WINNER) {  
            game.originator.addr.transfer(game.betAmount*2);  
        } else if (game.taker.status == STATUS_WINNER) {  
            game.taker.addr.transfer(game.betAmount*2);  
        } else {  
            game.originator.addr.transfer(game.betAmount);  
            game.taker.addr.transfer(game.betAmount);  
        }  
    }  
}  
  
resetGame();  
getBetOutcome();  
}
```

When a payment is made, it calls the fallback function of the payee contract

The fallback function can call the payment function again recursively

Phishing using tx.origin

```
interface Wallet {  
    function transferTo(address to, uint amount);  
}  
contract Exploit {  
    address owner;  
    constructor() public {  
        owner = msg.sender; {  
    }  
    function getOwner() public returns (address) {  
        return owner;  
    }  
    function() payable public {  
        Wallet(msg.sender).transferTo(owner, msg.sender.balance);  
    }  
}
```

Parity Wallet attacks

- Bad coding left them vulnerable to attacks on two occasions

```
contract WalletLibrary is WalletEvents {
    function initWallet(address[] _owners, uint _required, uint _daylimit) only_uninitialized {
        initDaylimit(_daylimit);
        initMultiowned(_owners, _required);
    }
    function execute(address _to, uint _value, bytes _data) external onlyowner returns (bytes32 o_hash) {
        //sends money to the address
    }
}

contract Wallet is WalletEvents {

    function Wallet(address[] _owners, uint _required, uint _daylimit) {
        function() payable {
            // just being sent some cash?
            if (msg.value > 0)
                Deposit(msg.sender, msg.value);
            else if (msg.data.length > 0)
                _walletLibrary.delegatecall(msg.data);
        }
    }
}
```

Short Address Attack

- The fault here lies in the way EVM handles underflows, it pads it with 0s

[illegible]

2<<4 = 32 coins
= 30 new coins

```
contract MyToken {
    mapping (address => uint) balances;

    event Transfer(address indexed _from, address indexed _to, uint256 _value);

    function MyToken() {
        balances[tx.origin] = 10000;
    }
    // 0x62bec9abe373123b9b635b75608f94eb86441630
    function sendCoin(address to, uint amount) returns(bool sufficient) {
        // 2
        if (balances[msg.sender] < amount) return false;
        balances[msg.sender] -= amount;
        balances[to] += amount;
        Transfer(msg.sender, to, amount);
        return true;
    }

    function getBalance(address addr) constant returns(uint) {
        return balances[addr];
    }
}
```

Honeypots

```
contract MultiplierX3 {
    address public Owner = msg.sender;

    function() public payable{}

    function withdraw() payable public{
        require(msg.sender == Owner);
        Owner.transfer(this.balance);
    }

    function Command(address adr,bytes data) payable public{
        require(msg.sender == Owner);
        adr.call.value(msg.value)(data);
    }

    function multiply(address adr) public payable{
        if(msg.value>=this.balance){
            adr.transfer(this.balance+msg.value);
        }
    }
}
```

this.balance is updated before the multiply method is called

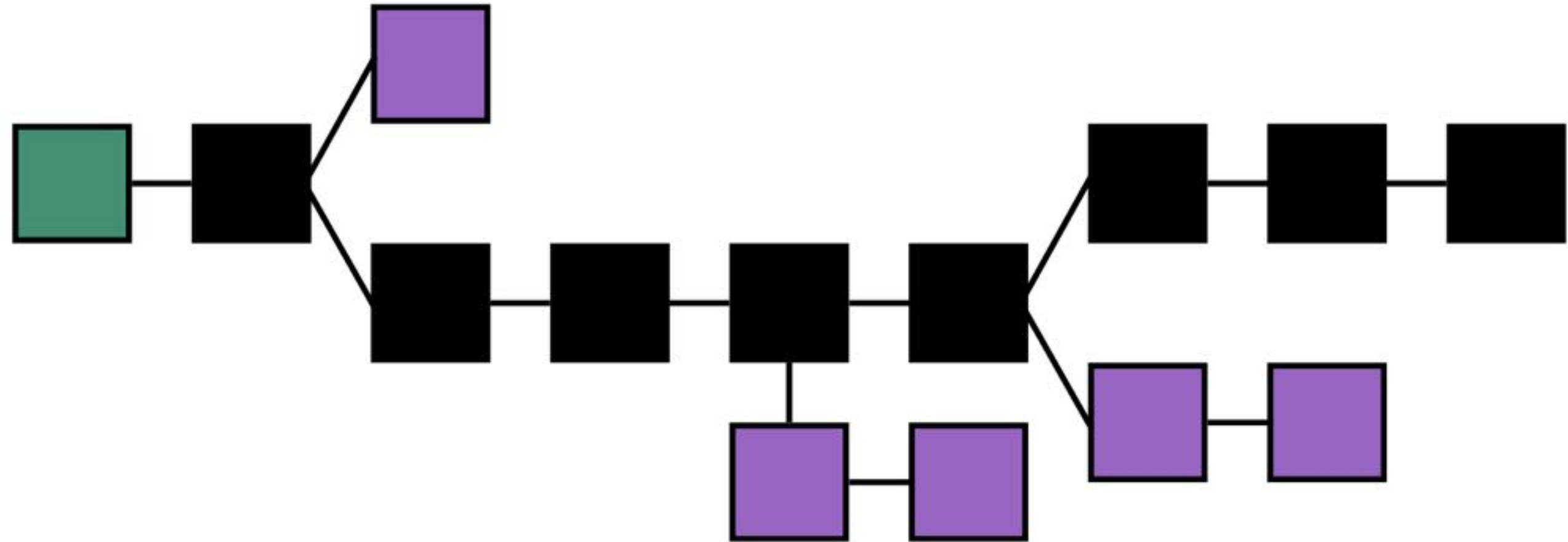
Hence, (msg.value>=this.balance) will always be false and transfer will not run (unless initial balance is 0)

Blockchain Attack Vectors

Consensus Mechanism Attack Vectors

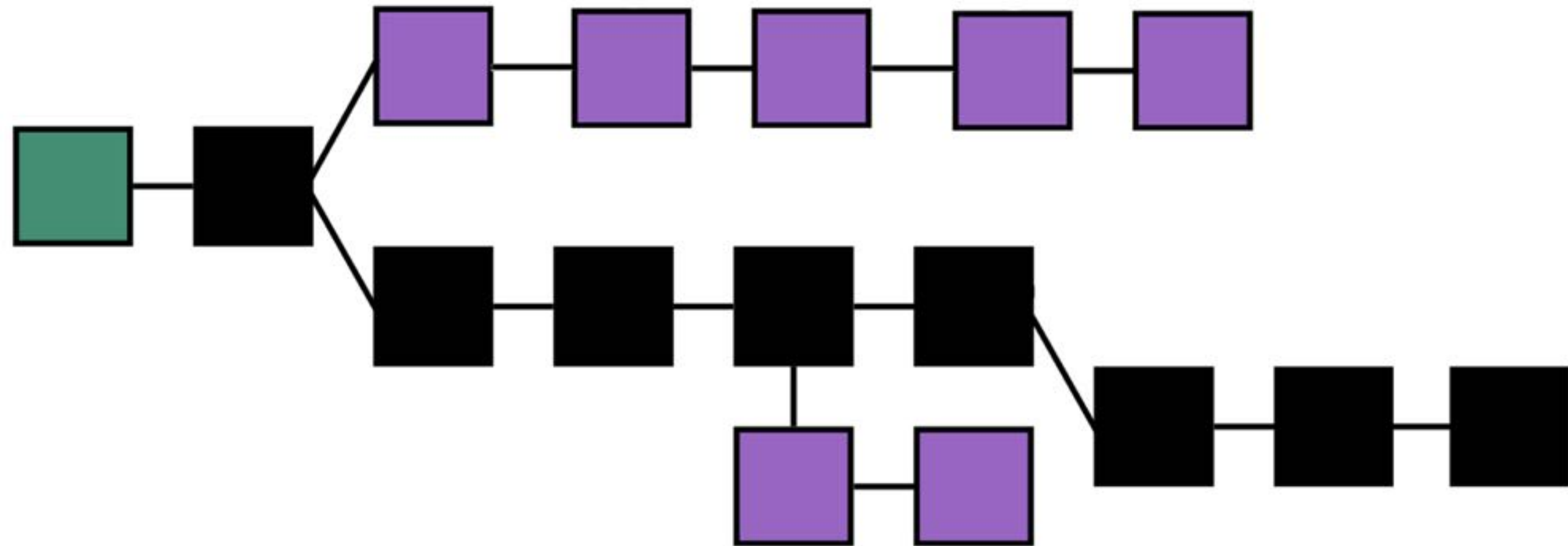
51% Attack (Proof of Work)

- Blocks are validated through mining, which is dependent on hashing power



Long Range Attack (Proof of Stake)

- Blocks are validated through validators, which is dependent on wealth (amount staked)



TEE-based consensus

- Proof of Elapsed Time (Intel's Hyperledger Sawtooth)
- Based on Intel Software Guard eXtensions (Intel SGX)
- Utilises Trusted Execution Environments to generate randomness
 - Data is protected from even malicious or hacked kernels
- Speculative Execution
 - Microarchitecturally, processor might speculatively guess values from memory
 - Faulty guesses disturbs other parts of the processor like the cache contents
 - Speculative Execution detects and measures such disturbances to infer in-memory values
- Meltdown, Spectre, Foreshadow

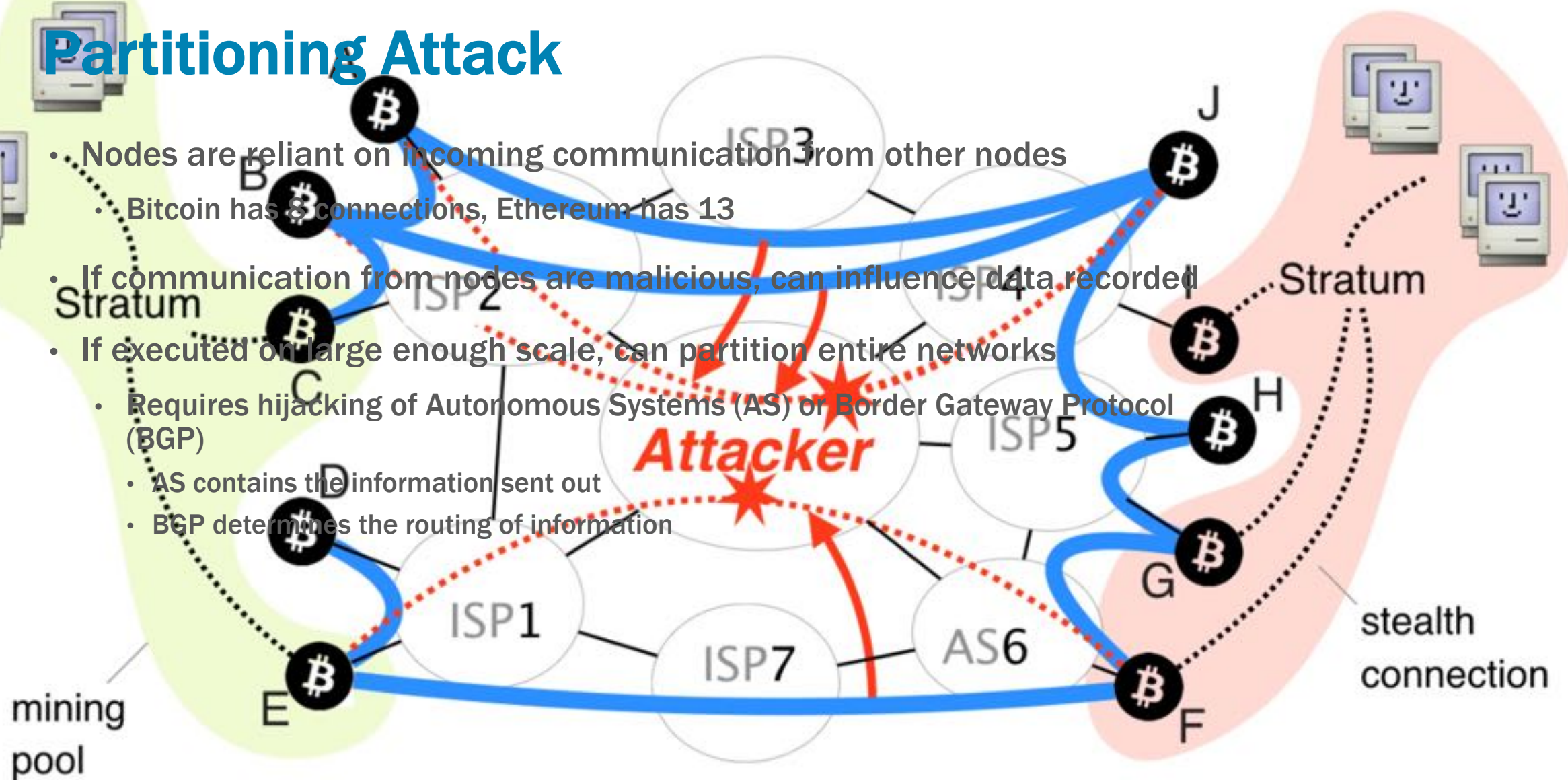
Blockchain Attack Vectors

Topology Attack Vectors

miners

Partitioning Attack

- Nodes are reliant on incoming communication from other nodes
 - Bitcoin has 8 connections, Ethereum has 13
- If communication from nodes are malicious, can influence data recorded
- If executed on large enough scale, can partition entire networks
 - Requires hijacking of Autonomous Systems (AS) or Border Gateway Protocol (BGP)
 - AS contains the information sent out
 - BGP determines the routing of information



mining
pool

stealth
connection

A Delay Attack

