GUILDAUDITS WEEK 1 & 2 REPORT

Submitted By: M D Sathees Kumar Aka Monstersec
Website: https://blog.mdsathees.In

Table of Contents

Mastering Ethereum Book Summary

Mastering Ethereum Flow chart is available in the below URL.

URL: https://github.com/Monst3rSec/Smart-Contract-Auditor-Resources/blob/main/Mastering%20Ethereum-Mastering%20Ethereum.png

Chapter 1 – Ethereum

Ethereum is proposed as a generic programmable blockchain that run a virtual state machine for executing arbitrary code (smart contracts codes). Ethereum has overcome the resource bounding problem by utilising fuel as gas to run the virtual state machine. Hence, Ethereum is Turing incomplete.

The Ethereum blockchain network relies on a small number of interconnected components, including a p2p network, consensus rules, transactions, state machines, data structure, consensus algorithm, clients, and economic security components, among others.

Chapter 2 – Ethereum Basis

Like our traditional banks, we can use Ethereum to trade and commerce purpose. But how to measure the value? So, Ethereum has introduced their own currencies units to measure the values based on the gas consumption to operate in the Ethereum blockchain networks.

Value (in wei)	Exponent	Common name	SI name
1	1	wei	Wei
1,000	10^3	Babbage	Kilowei or femtoether
1,000,000	10^6	Lovelace	Megawei or picoether
1,000,000,000	10^9	Shannon	Gigawei or nanoether
1,000,000,000,000	10^{12}	Szabo	Microether or micro
1,000,000,000,000,000	10^{15}	Finney	Milliether or milli
1,000,000,000,000,000,000	*10^{18}*	*Ether*	*Ether*
1,000,000,000,000,000,000,000	10^{21}	Grand	Kiloether
1,000,000,000,000,000,000,000,000	10^{24}		Megaether

The following question that arises is how to transfer these currencies. Like a traditional banking, we must open an account to holding, transfer, and receive Eth currencies by using wallet software.

As one of the most popular software wallets, Meta Mask is accessible as a browser extension and on mobile devices. After installing the wallets, you may create an account by ordering the mnemonic codes, which will generate the account's public and private keys. Additionally, you must keep your private keys for protecting your account.

After settling the account, you may link the wallet to the preferred blockchain network (Ethereum mainnet or testnet for testing) to send and receive Eth currency.

Chapter 3 – Ethereum Clients

An Ethereum client is a software application that implements the Ethereum specification and communicates over the peer-to-peer network with other Ethereum clients.

Ethereum client (for convenience, we call an Ethereum Full Node) running on the live mainnet network that keeps all ledger data in the blockchain. Full Node hardware will be required more often as the ledger expands since it will do the computing tasks necessary for adding the next block to the network.

The Ethereum Remote Client has the ability to communicate with a live blockchain network via API calls (web.js) and wallet functionality. Ex: Metamask, Geth etc.

Reference: https://www.coindesk.com/learn/ethereum-nodes-and-clients-a-complete-guide/

Chapter 4 – Cryptography

Ethereum uses an asymmetric algorithm to protect the privacy of its users. Just like with bank account numbers, Ethereum account addresses are formed from a combination of both public and private keys. There are two types of Account addresses in Ethereum, such as Externally owned account addresses and contract addresses. This Ethereum address will be generated as described below.

Ethereum uses the elliptic curve algorithm to make public and private keys. The public key is then hashed with the keccak-256 hash algorithm and turned into an Ethereum address format.

Elliptic curve algorithm -> public and private key -> keccak256(public) -> Ethereum Address [0x + (20 bytes of keccak256 result hash)]

Chapter 5 – Wallets

Wallets is a software application used to manage and store account keys. These wallets are available in two varieties.

- Non-deterministic Wallets - Keys are created based on distinct random numbers and are unrelated. It can no longer regenerate.
- Deterministic wallets — keys are produced based on seeds, and if hierarchical deterministic wallets are utilised, each key is tied to the others. It is regenerable by the use of seeds.

Account keys are often maintained and managed by the user (non-custodian wallets) or a third party (custodian wallets), which is depends on the user's preference.

To create keys from seeds, the hierarchical deterministic wallets that adopted Ethereum used a standard key derivation mechanism like BIP-32/BIP-44. The following procedure can produce these seeds.

Step1 – Generate the mnemonic code

Step2 – Generate the seeds using mnemonic code.

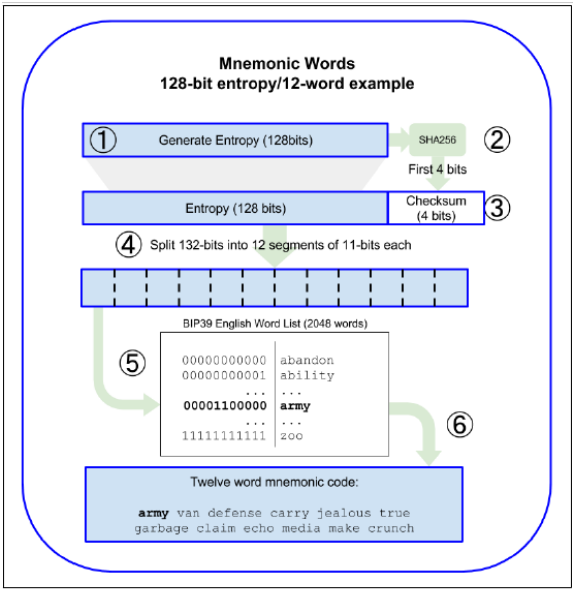

Figure 5-2. Generating entropy and encoding as mnemonic words

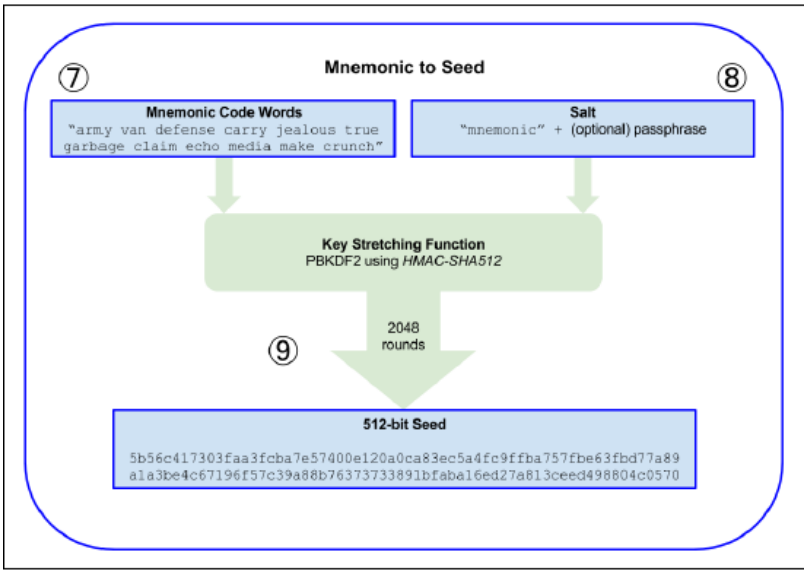

Figure 5-3. From mnemonic to seed

As previously described, how can deterministic wallets renew private keys using seeds? Using the application shown below, we could generate keys using seeds.

Application: https://allprivatekeys.com/mnemonic-code-converter

Process Reference: https://github.com/SilentCicero/ethereum-wallet-management/blob/master/ethereum-wallet-basics/using-seed-phrases-to-create-ethereum-accounts.md

Chapter 6 – Transactions

A digitally signed message created by an external owned account and transmitted over the Ethereum network to be stored on the Ethereum blockchain.

Ethereum blockchains use a unique transaction format that is easily understood by other nodes in peer-to-peer networks, which allows them to reliably record and verify transactions across those networks. Here are some of the parameters that may be included in a transactional message.

Nonce	**Transaction ID** which is used to avoid false transaction and ordering the transaction sequence.
Gas Price	**Gas Price** need to be paid by EOA for making the successful transaction.
Gas Limit	**Max. Gas price** need to be paid by EOA
Recipient	**Destination address**
Value	**Ether Amount** will be transfer from source to destination.
Data	**Data as function orientated invocation** will used to trigger a contract function and their parameter.

ECDSA signature	**Signature Parameter** (V, R, S) used for digital signature algorithm

Contract creation is also considered as special transaction where the destination address is zero address.

Chapter 7 – Smart contracts and Solidity

Smart contracts are immutable computer program that run in the blockchain network. There are multiple programming language used to write the smart contracts. Ex: Solidity, Rust etc.

Solidity is an object-oriented programming language used to implement smart contracts on Ethereum blockchain networks. In this chapter, we can learn more about solidity data types, ABI, object-oriented concepts, functions, and error handling associated with contracts.

In addition, this chapter offers a few recommendations on general guideline to cut down on the amount of gas that is used in blockchain.

Chapter 8 – Smart contracts and Vyper

When compared to other smart contract programming languages, Vyper was created for simplicity and security in a way that is not seen in other smart contract programming languages.

In this chapter, we can learn the multiple object-oriented concepts such as modifier, class inheritance, function overloading, decorators etc.

Chapter 9 – Smart contracts Security

Solidity implementation is not as safe as we believe, and it has many security flaws that have been exploited to steal tokens/ether from deployed contracts.

The most typical types of security flaws are re-entrancy, delegate call, default visibility, default origin, DOS, front running, contact libraries, and so on.

Note: Few vulnerabilities are outdated (ex: overflow vulnerabilities) which is mentioned in this chapter.

Chapter 10 – Token

Token means blockchain-based abstractions that can be owned and that represent assets, currency, or access rights.

There are two types of tokens.

1. **Fungible Token – ERC 20**

   ERC 20 Tokens are fungible when we can substitute any single unit of the token for another without any difference in its value or function and ICO (Initial Coin Offering) can be decided when the contract is deployed, and if necessary, we can change the ICO value.

ERC20 required functions and events.

- totalSupply - Returns the total units of this token that currently exist. ERC20 tokens can have a fixed or a variable supply.
- balanceOf - Given an address, returns the token balance of that address.
- transfer - Given an address and amount, transfers that amount of tokens to that address, from the balance of the address that executed the transfer.
- transferFrom - Given a sender, recipient, and amount, transfers tokens from one account to another. Used in combination with approve.
- approve - Given a recipient address and amount, authorizes that address to execute several transfers up to that amount, from the account that issued the approval.
- allowance - Given an owner address and a spender address, returns the remaining amount that the spender is approved to withdraw from the owner.
- Transfer - Event triggered upon a successful transfer (call to transfer or transferFrom) (even for zero-value transfers).
- Approval - Event logged upon a successful call to approve.

ERC20 optional functions:

- name - Returns the human-readable name (e.g., "US Dollars") of the token.
- symbol - Returns a human-readable symbol (e.g., "USD") for the token.
- decimals - Returns the number of decimals used to divide token amounts. For example, if decimals is 2, then the token amount is divided by 100 to get its user representation.

2. Non-fungible Token (deeds Standards) – ERC 721

ERC 721 are non-fungible tokens that track an ownership of a unique thing. The thing owned can be a digital item, such as an in-game item or digital collectible; or the thing can be a physical item whose ownership is tracked by a token, such as a house, a car, or an artwork.

ERC 721 Required Functions:

- balanceOf - Returns the number of NFTs in `owner's account.
- ownerOf - Returns the owner of the NFT specified by tokenId.
- safeTransferFrom - Transfers a specific NFT (tokenId) from one account (from) to another (to).
- transferFrom - Transfers a specific NFT (tokenId) from one account (from) to another (to).
- approve - Approves another address to transfer the given token ID
- getApproved - Gets the approved address for a token ID
- setApprovalForAll - Sets or unsets the approval of a given operator
- isApprovedForAll - Tells whether an operator is approved by a given owner.
- safeTransferFrom - Transfers the ownership of a given token ID to another address.
- supportsInterface - Safely transfers the ownership of a given token ID

ERC 721 Required Events

- Transfer - Emitted when tokenId token is transferred from from to to.
- Approval - Emitted when owner enables approved to manage the tokenId token.
- ApprovalForAll - Emitted when owner enables or disables (approved) operator to manage all of its assets.

ERC 721 optional Function

- name - Returns the token collection name.
- symbol - Returns the token collection symbol.
- tokenURI - Returns the Uniform Resource Identifier (URI) for tokenId token.

ERC20 & ERC 721 implementations:

- Consensys EIP20
- OpenZeppelin StandardToken

Resources:

- https://docs.openzeppelin.com/contracts/2.x/erc20
- https://docs.openzeppelin.com/contracts/2.x/erc721
- https://docs.openzeppelin.com/contracts/4.x/api/token/erc20
- https://docs.openzeppelin.com/contracts/2.x/api/token/erc721

Chapter 11 – Oracle

Oracle is a concept for a system that can communicate external data sources to Ethereum smart contracts. Oracle data, which is external data, is sent to smart contracts as signed messages. These messages are used as inputs for the next operations. But why do we have to send data from outside the smart contract?

Keeping all the data in the on-chain is a costly and inefficient way to do things.

In the actual world, we employ sensors, physical equipment, and medical gadgets, among other things, to tackle real-life issues. In order to solve the problem quickly, we need to connect these data to the ethereum blockchain.

Take a weather Dapp as an example. All the weather sensor data will be sent to the smart contract as signed messages, and once it has been processed, it will be sent back to the application frontend. So, the user can look at the weather report in his apps.

There are three categories that served an oracle data into on chain.

1. Request and response - two-way communications model
2. Publish and subscribe - subscriptions based communications model.
3. Immediate read – on-demand request and response model

Chapter 12 – Decentralized Apps

Decentralized Applications (also known as web3 application) are application that built on the decentralized blockchain. There are five components in the decentralized application.

1. Backend software – smart contract programmable languages used to write the backend logic, operations, and rules.
2. Frontend software – Similar to web2 technologies, we are using HTML, CSS and JavaScript language used to create the frontend web application.

3. Data storage – except on-chain data, we need to store and retrieve data in the off-chain storage. IPFS, Swarm are protocols developed for data storage.
4. Network communication – Peer to Peer communication used to transfer the message between blockchain nodes.
5. Name resolution –Like DNS, ENS used for name resolution purpose. (Domain to Eth address)

Chapter 13 – Ethereum Virtual Machine

The EVM is the part of Ethereum that is in full control of deploying and running smart contracts. Ethereum EVM is like a Quasi Turing state machine, which can only do a limited number of computations based on how much gas is available to execute a smart contract. EVM uses a stack-based architecture, to store all in memory values during the execution and operations.

The EVM instruction are supporting the below operations.

- ✓ Arithmetic and bitwise logic operations
- ✓ Execution context inquiries
- ✓ Stack, memory, and storage access
- ✓ Control flow operations
- ✓ Logging, calling, and other operators

Please check the below URLs for more information about the EVM instruction sets.

URL: https://github.com/crytic/evm-opcodes

Chapter 14 – The consensus

Consensus is intended to produce a system of strict rules without rulers. This implies that the consensus algorithm and its rules are crucial to the functioning of the blockchain. There is multiple consensus algorithm that used to build the blockchain. Ethereum has built it by proof of work consensus algorithm. Due to high energy consumption, Ethereum and many blockchain has being rebuilt their blockchain by using proof of work consensus algorithm.

Except above consensus algorithms, few more consensus algorithm is used it to build their blockchain based on their use case requirements.

1. Proof of authority
2. Proof of Burn
3. Proof of Elapsed time
4. Delegated proof of stacks.

Please check the resource in the below URL:

https://thecrypto.app/knowledge/the-main-types-of-consensus-algorithms/

Appendix content - Appendix pages give further information on Ethereum specs, EVM opcodes, Web.js, and frameworks, among other topics.

Secureum Solidity 201 summary

Object-oriented features, EVM storage and memory architecture, value layout, and third-party libraries are among the advanced solidity topics covered in Secureum Solidity 201 articles.

Contract Types

Contracts in Solidity are similar to classes in object-oriented languages. There are three types of contracts.

Abstract Contracts: Contracts need to be marked as abstract when at least one of their functions is not implemented. They use the abstract keyword.

Interfaces: They cannot have any functions implemented.

There are further restrictions:

1) They cannot inherit from other contracts, but they can inherit from other interfaces

2) All declared functions must be external

3) They cannot declare a constructor

4) They cannot declare state variables. They use the interface keyword.

Libraries: They are deployed only once at a specific address and their code is reused using the DELEGATECALL opcode. This means that if library functions are called, their code is executed in the context of the calling contract. They use the library keyword.

EVM Storage

EVM Storage is a key-value store that maps 256-bit words to 256-bit words and is accessed with EVM's SSTORE/SLOAD instructions. All locations in storage are initialized as zero.

EVM Storage layout packing and ordering process used to fill storage slots with data. The value will be stored in a single storage location if it does not exceed 32 bytes. Otherwise, the subsequent storage space will be used.

The storage structure will vary depending on the data type, and each storage slot point will be determined using the Keccak256 hash algorithm.

- Fixed Size variable - Variables of fixed size are stored in a single storage slot.
- Dynamic size variable - store in numerous storage slots, but begins with a single storage slot.
- Structs and arrays - Always begin filling a new storage slot and store items sequentially according to the requirements of the data type.
- Mapping – The key and value will be saved in separate storage locations.
- Inheritance – Ordering the storage slot according to the C3 linearization contract rules and reusing the same storage slot if the contract utilised the same variable.

EVM Memory

EVM memory is linear and can be addressed at byte level and accessed with MSTORE /MSTORE8 /MLOAD instructions. All locations in memory are initialized as zero.

Solidity uses free memory pointer object and it's initialized at 0x80 pointer. Free memory pointer will move the exact pointer location for updating the value in the memory.

Solidity reserves four 32-byte slots, with specific byte ranges (inclusive of endpoints) being used as follows:

- 0x00 - 0x3f (64 bytes): scratch space for hashing methods
- 0x40 - 0x5f (32 bytes): currently allocated memory size (aka. free memory pointer)
- 0x60 - 0x7f (32 bytes): zero slot (The zero slot is used as initial value for dynamic memory arrays and should never be written to)

In EVM memory, all the dynamic sized arrays are stored one by one and also it reversed few keywords such as *after, alias, apply, auto etc.*

Inline Assembly

Inline assembly is a way to access the Ethereum Virtual Machine at a low level. This bypasses several important safety features and checks of Solidity. You should only use it for tasks that need it, and only if you are confident with using it.

Yul is an inline assembly language used to access External Variables, Functions, and Libraries from memory. It also has its own syntax for accessing memory for various data kinds, functions, control structures, and more.

Uniswap

Uniswap is an automated liquidity mechanism powered by a constant product formula and deployed on the Ethereum blockchain as a system of non-upgradable smart contracts.

x*y=k is the algorithm for automated market making, where x and y represent a token pair. Few fundamental concepts include Pool, swaps, flash swaps, and oracles is used in the Uniswap.

Uniswap V3 is the most recent version of Uniswap. Uniswap V3 adds features such as concentrated liquidity, Multiple charge levels, and TWAPs that provide oracles, among others.
