Ethereum Testnets

- Ethereum can handle a large number of transactions. However, it is also a very expensive blockchain.
- Because Ethereum has protocol features that must be followed, it can be mocked up to be used in a testnet.
- **Testnet**: is a collection of nodes that are used to test the Ethereum protocol. Tests are run on the testnets to ensure that the protocol is working as expected.

Use of Testnets

- Writing smart contract and deploying them on the mainnet is a much more expensive operation than writing tests and deploying them on the testnet.

  - **Deploy Smart Contracts :** Test the functionality and security of your smart contracts without spending valuable ETH.
  - **Test Applications :** Ensure your decentralized applications (DApps) work as expected with other tools and services.
  - **Evaluate Network Upgrades :** Test new features and protocol changes in a simulated environment before they are implemented on the main network.

Examples of Testnets

- **Sepolia** : A popular, widely used Ethereum testnet.
- **Holesky** : Another independent testnet for testing applications and network upgrades.
- **Goerli** : A previously popular testnet, now largely deprecated in favour of Sepolia and Holesky.

Ethereum Testnets - Sepolia

- Sepolia is a test network for Ethereum launched in 2021 that operates independently of the main network (mainnet) but closely simulates its conditions.
- **Key features**:
  - Ease of access
  - Free to use
  - Serves to run realistic tests
- It provides decentralized application and developer tooling builders with an environment for testing their solutions before moving to the Ethereum mainnet.

Ethereum Testnets –Working of Sepolia

- The test network mirrors the mainnet's functionality, including transaction processing, block production and smart contract deployment.
  - ❖ It lets developers see how their code will behave in a live environment.
  - ❖ The testnet has its own block explorers that allow users to view on-chain data specific to Sepolia.
  - ❖ It used to test protocol upgrades and hard forks (like upcoming Pectra) before they go live on mainnet so developers can test how these changes affect dApps and smart contracts.
  - ❖ Supported by popular Ethereum development frameworks and tools like Remix, Hardhat and Foundry.
- Developers can interact with Sepolia using testnet Ethereum RPC (Remote Procedure Call) endpoints, similar to how they would on the mainnet.

Sepolia ETH

- Sepolia uses test Ether (ETH) which serves the same purpose as ETH on the mainnet but has no actual value. Developers can request test ETH from faucets that provide them for free.
- These tokens allow developers to interact with the network as they would on the mainnet — sending transactions, paying for gas fees, deploying contracts, and more.

- An **Ethereum faucet** is a service (usually a website or dApp) that gives out small amounts of **free ETH** (test Ether) to users.

Ethereum Testnets - Holesky

- Holesky was launched on September 2023, replacing the Goerli test network
- This new network uses Holesky ETH tokens and serves as a test environment for consensus staking, infrastructure and protocols
- As part of Ethereum's post- Dencun upgrade plans, Goerli was deprecated in April 2024 and fully replaced by Holesky testnet.
- Existing Goerli validators were shut down and prompted to migrate to either Sepolia or Holesky.

Key Differences : Sepolia vs. Holesky

Validator Sets

- **Sepolia**: Sepolia has a closed validator set, making it easy to sync, and requiring minimal disk space to run a node.
- **Holesky**: Holesky is open to new validators, with over 1.5 million validators running on the testnet.

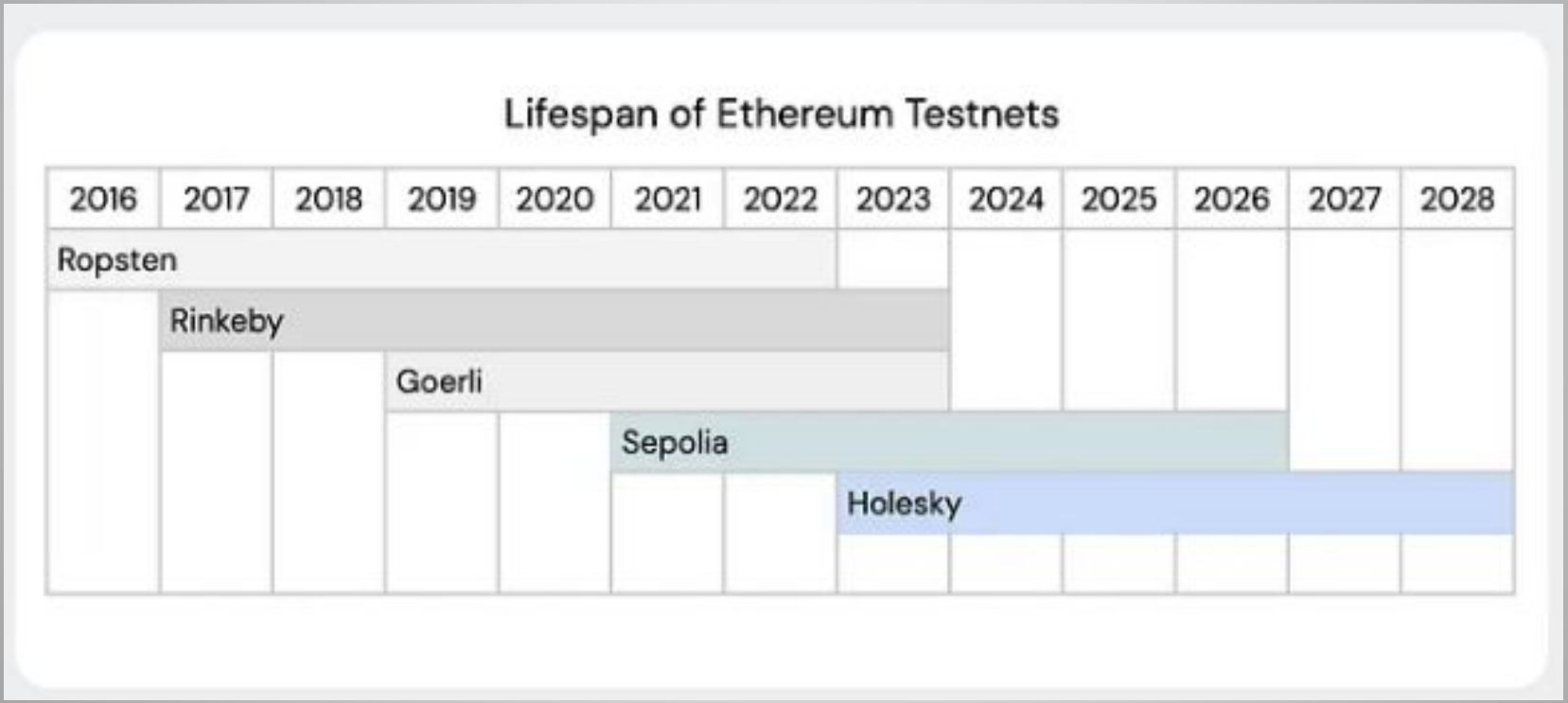
Test Token Supply Mechanics

- **Sepolia**: Sepolia has an uncapped token supply.
- **Holesky**: While not unlimited, Holesky's has a capped supply of holETH at 1.6 billion, more than twice the amount issued on Goerli.

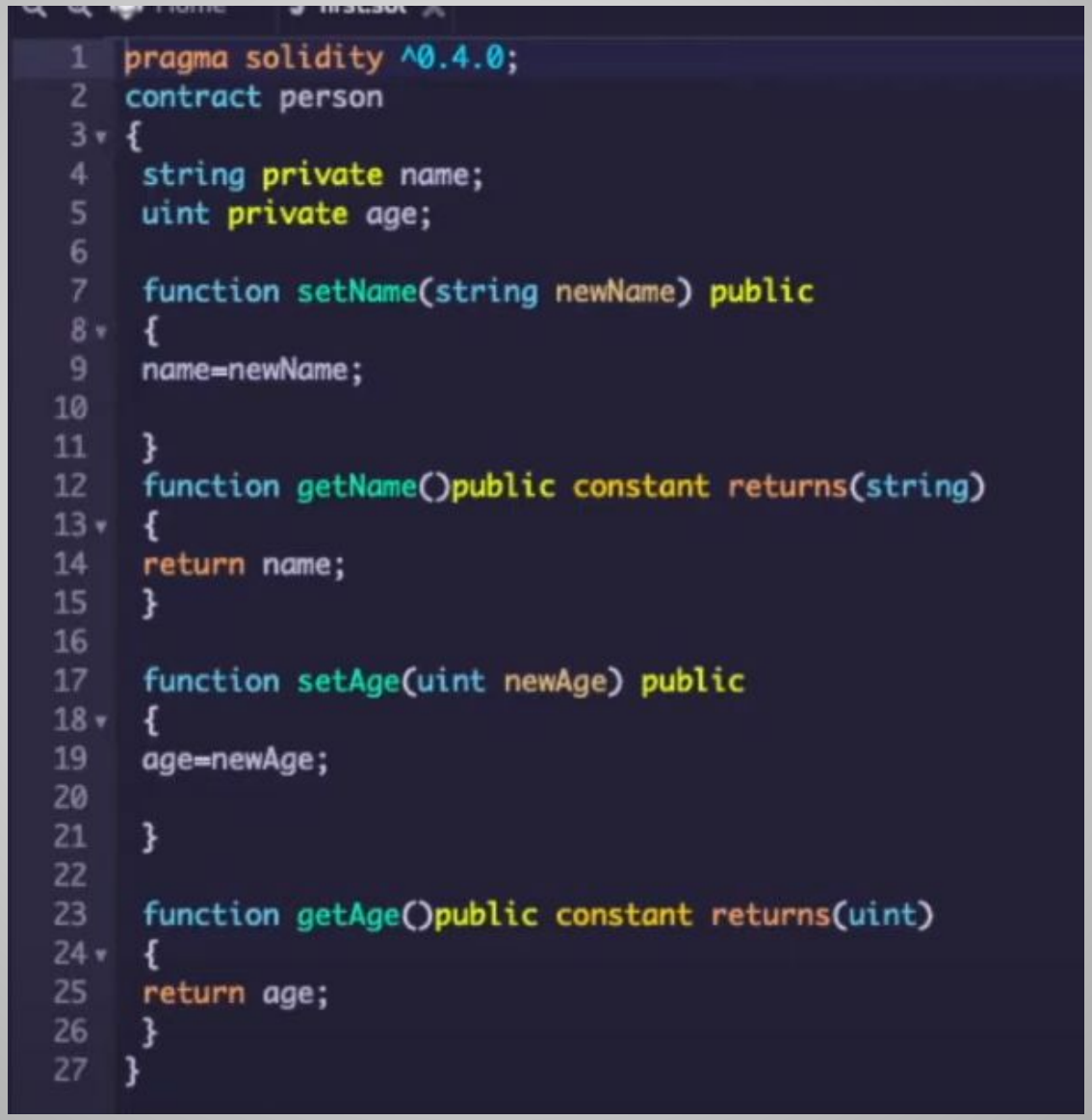
Intended Use Case

- **Sepolia**: Sepolia is the best place to test smart contracts and dApps.
- **Holesky:** The intent of Holesky is testing validators and staking.

Lifespan of Ethereum Testnets



SC-Solidity

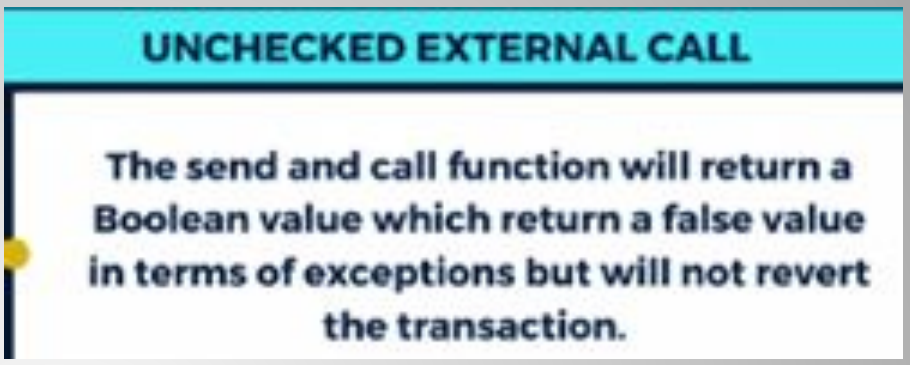


```
1  pragma solidity ^0.4.0;
2  contract person
3  {
4   string private name;
5   uint private age;
6
7   function setName(string newName) public
8   {
9   name=newName;
10
11  }
12  function getName()public constant returns(string)
13  {
14  return name;
15  }
16
17  function setAge(uint newAge) public
18  {
19  age=newAge;
20
21  }
22
23  function getAge()public constant returns(uint)
24  {
25  return age;
26  }
27 }
```

Solidity issues

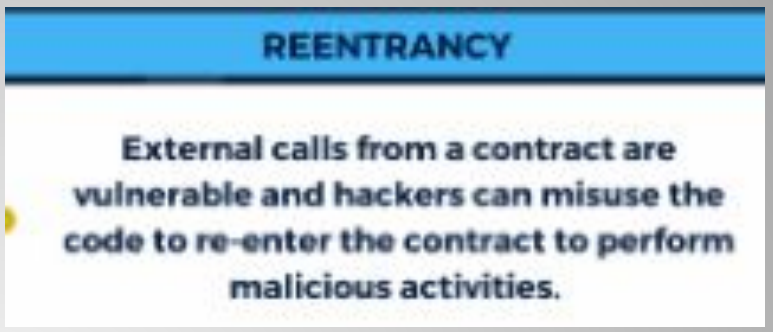
1. Unchecked External Call

- This is **a major solidity issue.** The most commonly used function is **the transfer function that can send Ether to any external account.**



- Other way is to **use the call() and send() function.**
- Developers use the call() function for more external calls.
- Unfortunately, the **send and call function** will only return a Boolean value which will tell you whether the call was successful or not.
- But if **these functions face any exceptions**, in which case they can't perform the task, they only return a false value rather than reverting.
- So, if you don't check the return value, you won't know if the transaction was a success or not.
- The developer may just expect to get a revert rather than check the value.
- **Thus, it's completely necessary to check the return value of unsuccessful transfers.**

Solidity issues



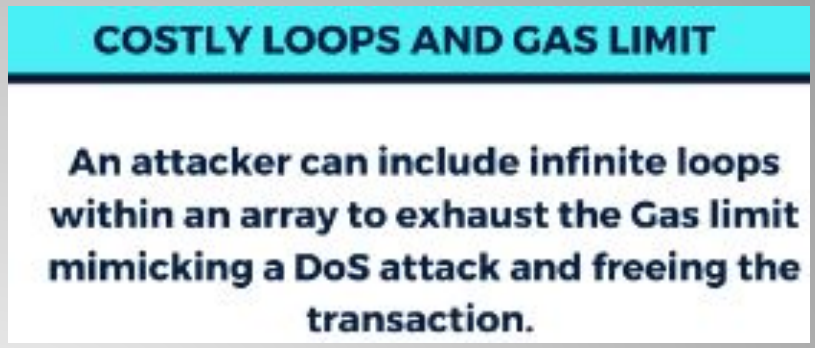
2. Reentrancy

- In reality, Ethereum smart contracts come with a special feature that can call or even use codes from other **external contract**s.
- Typically, contracts use Ether and send Ether to other external contract addresses.
- However, to perform all of these functions, the contract needs to use an external call.
- Actually, these **external calls don't have adequate safety features**; thus, the hacker can attack these codes and ensure to execute more tasks with the contract calling it.
- Therefore, **the code can re-enter the contract, misuse the information, or even send tokens to other addresses**.
- In reality, this is exactly what happened in the DAO attack. Mainly this attack can happen when you send Ether to an unknown contract.

Solidity issues

- The attacker can create a **malicious program in an external address**, and this will introduce the f**allback function.**
- Therefore, when a contract sends any amount of Ether to that address, it will start the malicious program.
- Fallback function is a special function available to a contract. It has following features −
  - It is called when a **non-existent function** is called on the contract.
  - It is required to be **marked external.**
  - It has **no name** and **no arguments**
  - It **can not return any thing**.
  - If **not marked payable, it will throw exception** if contract receives plain ether without data.

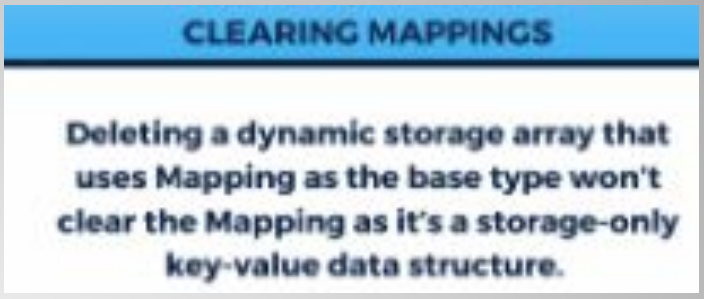
Solidity issues



- **3. Costly Loops and Gas Limit**
- The computational power in the Ethereum blockchain is not for free.
- You need to pay **Ether to buy Gas** in order to get the **amount of Computational power** for your transaction to execute.
- So, if you can somehow **reduce the number of computational** steps, it will also **save time and even save you a lot of money**.
- **Adding loops into the program is one way to increasing the costs.** In reality, an array already comes with a lot of loops. However, if the elements increase, more integration is required to complete that specific loop.
- So, if an **attacker can include infinite loops**, he/she can single-handedly exhaust all available Ethereum Gas.
- In this case, the attacker can influence the elements of the array length, which will create a DOS issue and won't allow the system to jump out of the loop.
- It will ensure that the contract is stalled as every contract comes with a **Gas limit**.
- Even though this solidity issue is not as prominent as other issues, more than 8% of smart contracts seem to have this problem.

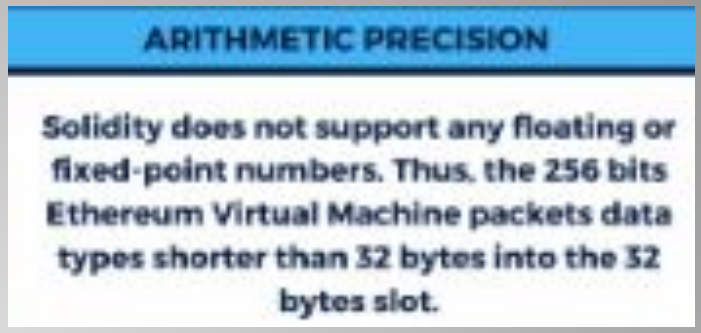
Solidity issues

4. Clearing Mappings



- Clearing mappings is a huge solidity issue due to having some limitations of this programming language.
- In reality, the **Solidity type mapping offers a key-value data structure**, which is storage-only for blockchain platforms.
- Thus, it will not track all the keys with a value other than zero.
- **you can’t clean a mapping without adding any extra information**,.
- In a dynamic storage array, if you want to use Mapping as the base type, if you delete or even pop the array, that won’t affect the Mapping.
- More so, if you use Mapping in a struct's member field as a type in a dynamic storage array, here the Mapping will also be ignored even if you delete the array.

Solidity issues



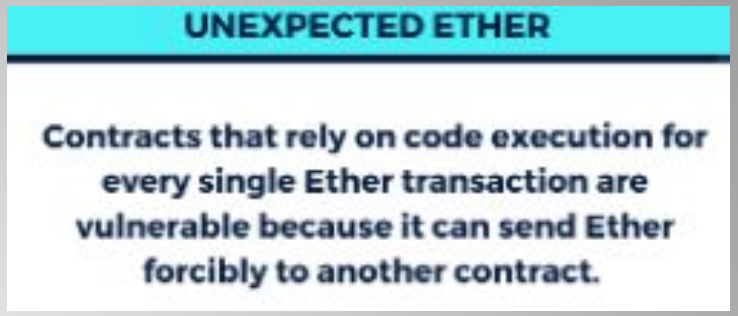
5. Arithmetic Precision

- Solidity **does not support any floating- or fixed-point numbers**.

  Therefore, to represent these floating points ,you need to use an integer

  type in the Solidity.
- In reality, many developers can make errors or create loopholes if they can't implement this correctly.
- Therefore, developers **need to implement their very own fixed-point** data type **using the standard integer data type**.
- This **process is not easy and quite complex** and can pose a lot of problems if not pulled off correctly.
- More so, the 256 bits Ethereum Virtual Machine can create a lot of issues for the data types as types shorter than 32 bytes are assembled together into the 32 bytes slot.
- Therefore, it affects the precision of any calculations as you can't expect a proper rounding when you perform any division before multiplication.

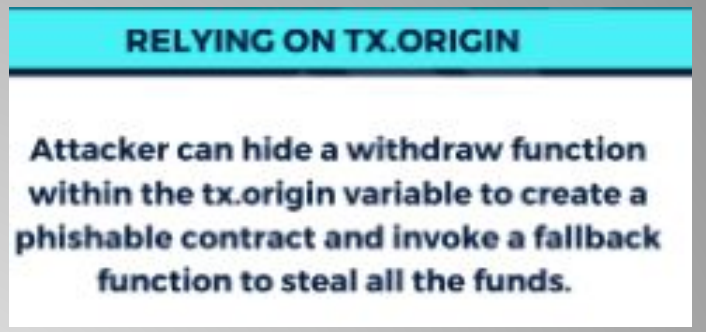
Solidity issues



6.Unexpected Ether

- Usually, when you are sending Ether to an address, it needs to execute the fallback function or other functions needed for the contract.
- However, there are certain exceptions to these rules. In this case, the Ether can stay in the contract without executing any code.
- Thus, **contracts that rely on programs for every single Ether transaction** are vulnerable because it can **send Ether forcibly to another contract.**
- Typically, there are 2 ways one can send Ether forcibly to another contract, and it **will not use any payable function** for that.
- In most cases, **the blockchain developers don't realize that you can accept or even obtain** Ether via other means aside from the payable function.
- It can lead to contracts having a false Ether balance, and it will lead to vulnerability.
- It's best **to check the current Ether stored in the contract** when before calling any functions

Solidity issues



7. Relying on tx.origin

- This one is a global variable and will scan the call stack and give you the address of the account that sent the transaction.
- The problem is that if you use **this variable for authentication purposes, it can make your smart contracts vulnerable.**

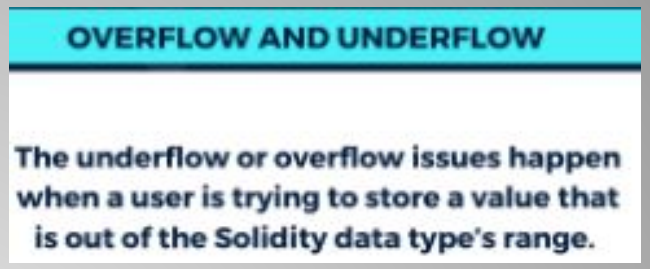
  Now your contract will face a phishing attack. Basically, the attacker can trick the user into using the tx.origin variable to authenticate the attacker to the contract.
- Here, the **attacker can hide the function withdrawAll()** within the tx.origin variable and create a malicious code for the phishable contract.
- Usually, the attacker will deploy the code and convince the user to send some Ether to this contract.
- The user will think that it's a typical authentication contract and will send the Ether to the contract (attackers address disguised within the contract).
- Thus, once the victim does that, it will invoke a fallback function which will call the withdrawAll() function.
- As a result, **all the funds from the victim will go to the attacker's address.**
- **Thus, users should not depend on tx.origin for authentication purpose only** as it's one of the major blockchain risks.

Solidity issues

8. Default Visibilities

- In Solidity, the functions have visibility to the public unless you are assigning how anyone can call that function.
- Therefore, the **visibility can determine who can call the function** – external users, derived contracts, internal users, etc.
- Any function which is default to public viewing can be **called by users externally**.
- In reality, many times, the developers tend to use incorrect specifiers, which will severely **damage the integrity of the smart contract**.
- As the **default is public**, so always make sure to specify the visibility if you want to change it.
- For example, if you are offering a reward to users who can perform a certain task make sure to keep the **visibility of the function private**.
- If **it's public**, then any user can just **see the function and call it and take the reward without performing any single task**.
- It's really easy to misuse this type of blockchain security concerns as many newcomers don't understand the concept fully.

Solidity issues



9. Overflow and Underflow

- In reality, Ethereum Virtual Machine can **only support a fixed size for integer data types.**

  So, blockchain professionals can only use a certain range of number to represent an integer variable.

  For example, in unit 8, you can only store values from [0,255]. So, here, if you want to store 256, you will only get 0 as a result.

  Thus, using variables in Solidity needs to take precautions as attackers can exploit this issue.

  Typically, the underflow or overflow issues happen when you are trying to store a value that is out of the data type's range.
- An underflow issue occurs when you are trying to subtract 1 from a unit 8 variable that had 0 as the initial value. In this case, it will only **return the number 255.**
- On the other hand, if you are trying to add a value that is larger than the data type's range will create an overflow issue.
- So, if you are **adding 257 to a unit 8 0 value**, it will give you a result of 1. it can create vulnerabilities, and attackers can use it to misuse the code and even create unexpected logic that results in an infinite loop.

Solidity issues

- **10. Timestamp Manipulation**

- In reality, blockchain timestamps are one of the most important elements in blockchain applications.
- You can use it for various purposes, for example, locking funds, setting a timer for funds to be released, entropy for numbers, state-changing conditions and so on.
- However, a miner has the freedom to change the timestamp, which is risky if they're using it incorrectly in the smart contract.
- In times of any rewards, with the use of enough Ether in the contract, the miner can change the timestamp and win the reward along with the pooled Ether. Obviously, it's not a fair trade.