

# Week 8

Introduction of smart contract audit & security



- - 
  - 
  - 
  - 
  -
- - 
  - 
  - 
  - 
  -

# Acknowledgement of Country

We respectfully acknowledge the Wurundjeri People of the Kulin Nation, who are the Traditional Owners of the land on which Swinburne’s Australian campuses are located in Melbourne’s east and outer-east, and pay our respect to their Elders past, present and emerging.

We are honoured to recognise our connection to Wurundjeri Country, history, culture, and spirituality through these locations, and strive to ensure that we operate in a manner that respects and honours the Elders and Ancestors of these lands.

We also respectfully acknowledge Swinburne’s Aboriginal and Torres Strait Islander staff, students, alumni, partners and visitors.

We also acknowledge and respect the Traditional Owners of lands across Australia, their Elders, Ancestors, cultures, and heritage, and recognise the continuing sovereignties of all Aboriginal and Torres Strait Islander Nations.

- -
- -

- - 
  - 
  - 
  - 
  - 
  - 
  - 
  - 
  - 
  - 
  - 
  - 
  -
- - 
  - 
  - 
  - 
  - 
  - 
  - 
  - 
  - 
  - 
  - 
  - 
  -



# Smart Contract Audit

# Smart Contract

- **Smart contracts** are one of the key components of many blockchain-based ecosystems, and an especially important element of many application-focused blockchains like Ethereum.
- These digital contracts are autonomous, decentralized, and transparent — and are usually irreversible and unmodifiable once deployed.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract Telephone {

    address public owner;

    constructor() {
        owner = msg.sender;
    }

    function changeOwner(address _owner) public {
        if (tx.origin != msg.sender) {
            owner = _owner;
        }
    }
}
```

# Smart Contract Use Cases

Digital  
Identity

Cross Border  
Payments

Loans and  
Mortgages

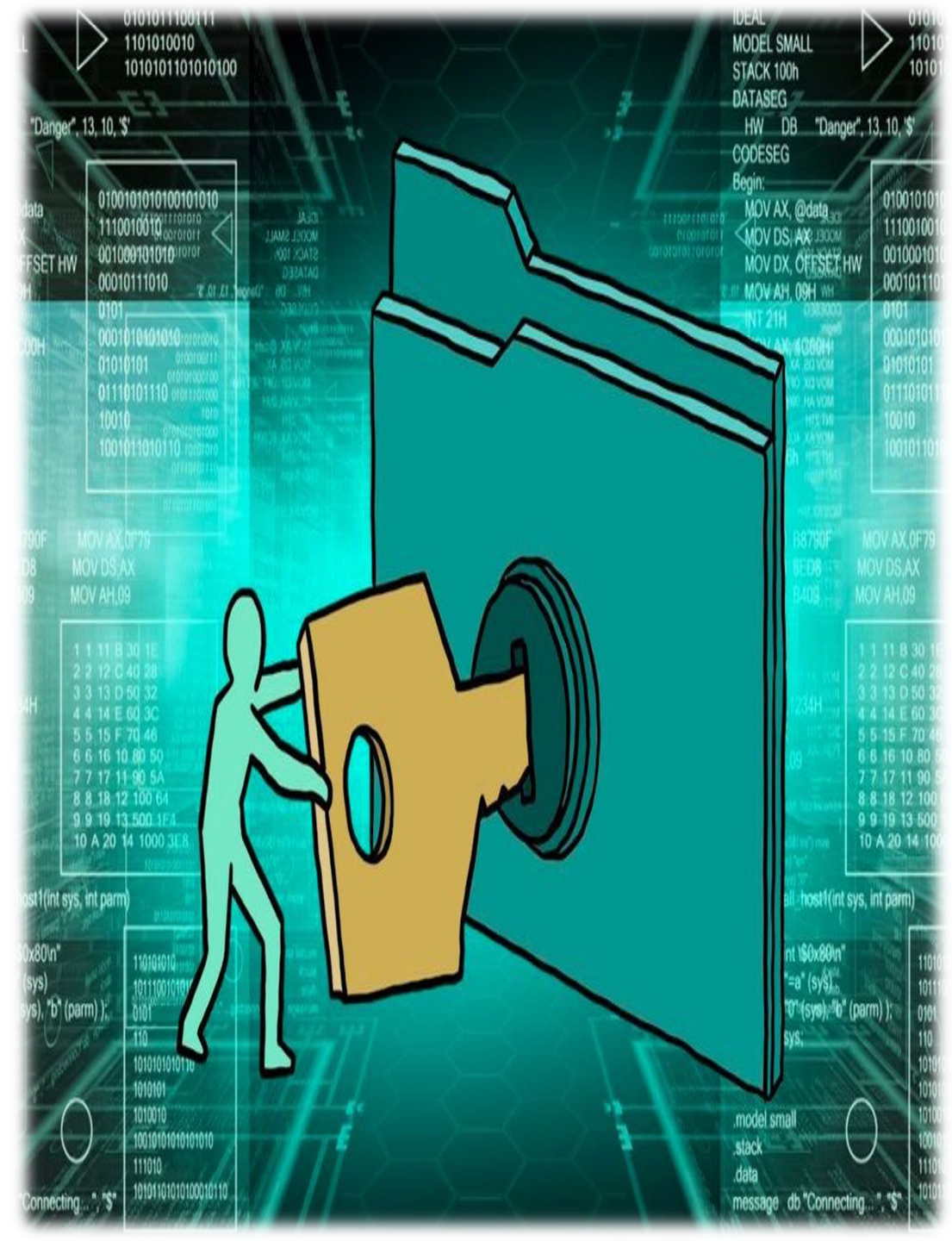
Financial Data  
Recording

Supply Chain  
Management

Insurance

# Smart Contract Security

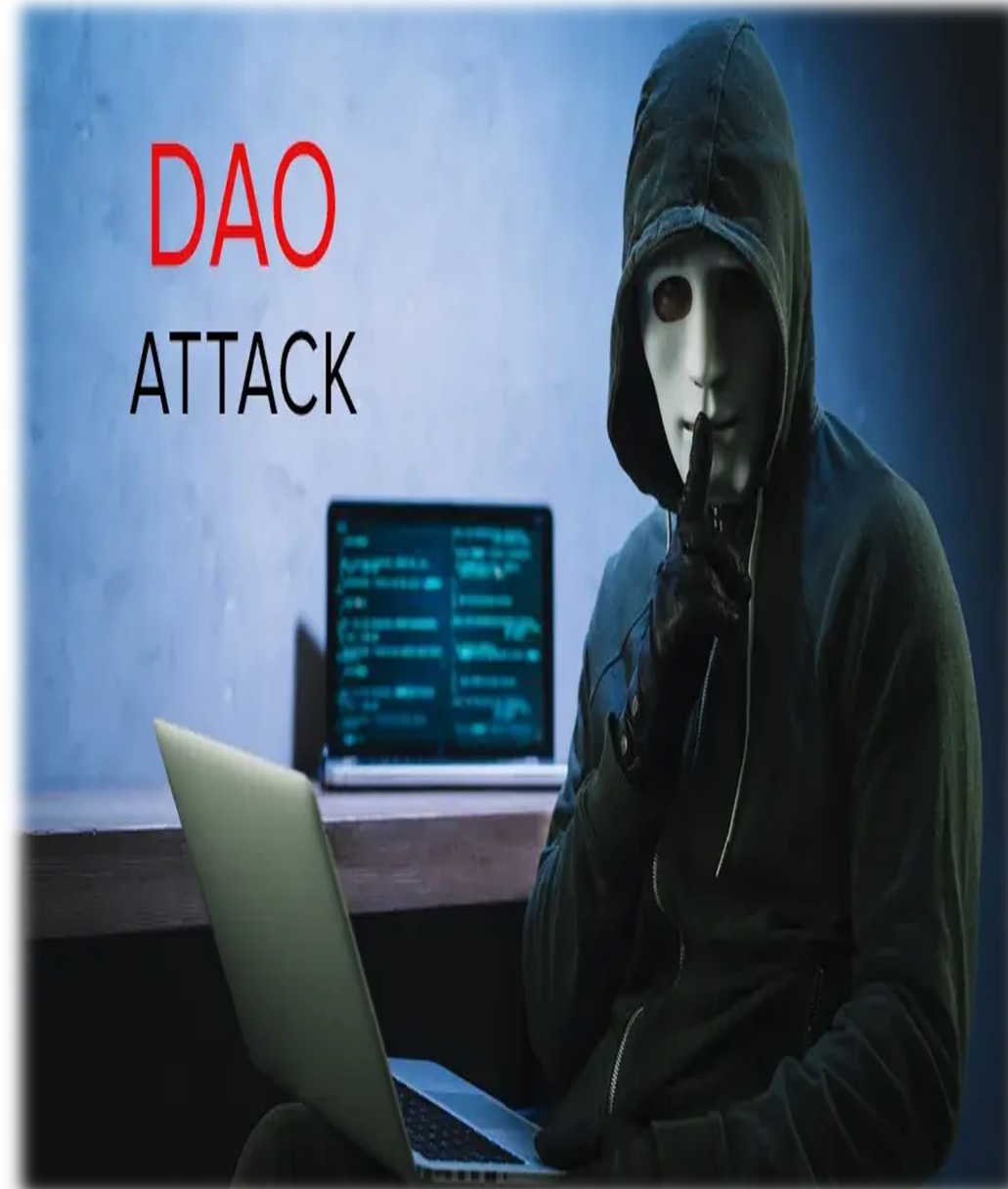
- Because smart contracts have the authority to **allocate high-value resources** between complicated systems and are, for the most part, autonomous, **security and consistency are critical**.
- The DAO breach on Ethereum results in losses amounting to **\$60 million**.
- Another example of a significant smart contract problem is the theft of **over \$3 million** from The Tinyman exchange on the Algorand blockchain in January 2022.





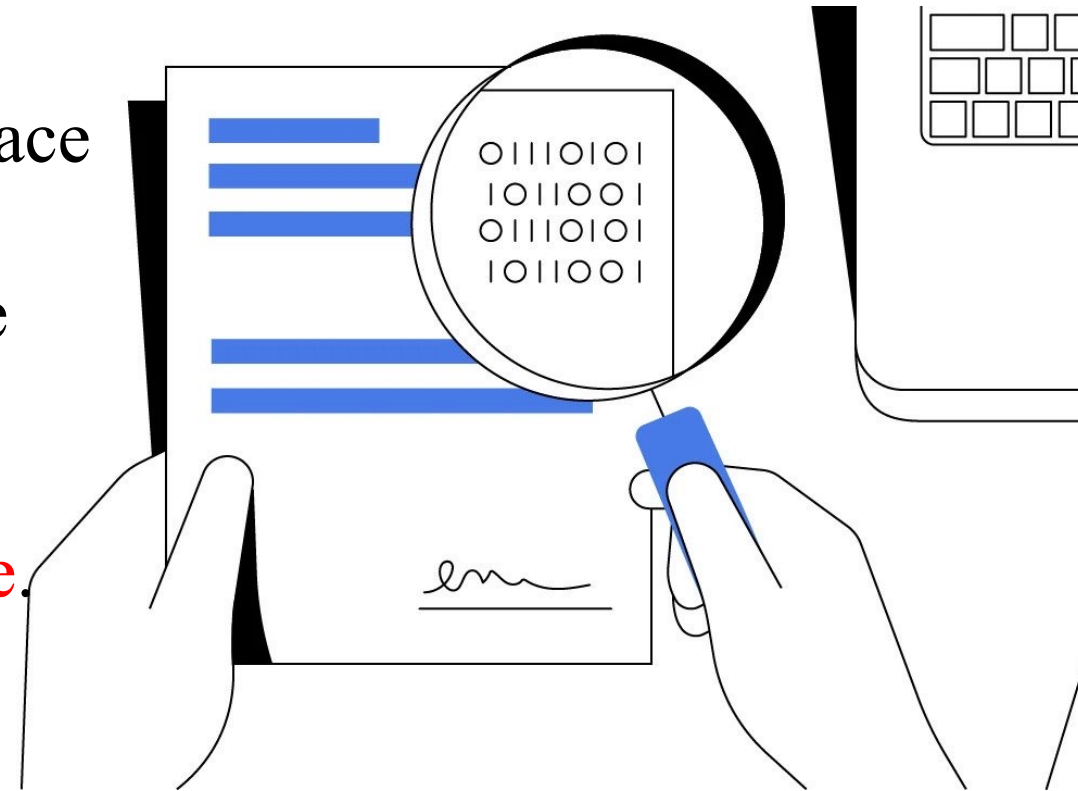
# Smart Contract Common Security Flaws

- **Re-entrancy:** Attackers using this bug can **withdraw balances multiple times** before their balance is set to 0.
- **Over/Under Flows:** These vulnerabilities allow attackers to create **unexpected logic flows**.
- **Frontrunning:** Because transactions are collected into blocks and added to the ledger as a part of various blocks, bad actors can buy large sums of tokens in response to large transactions that will **swing the token's price**.
- **Incorrect calculations:** Incorrect **decimal handling and fee calculations** can result in the loss of funds or funds being locked indefinitely.



# Smart Contract Audit

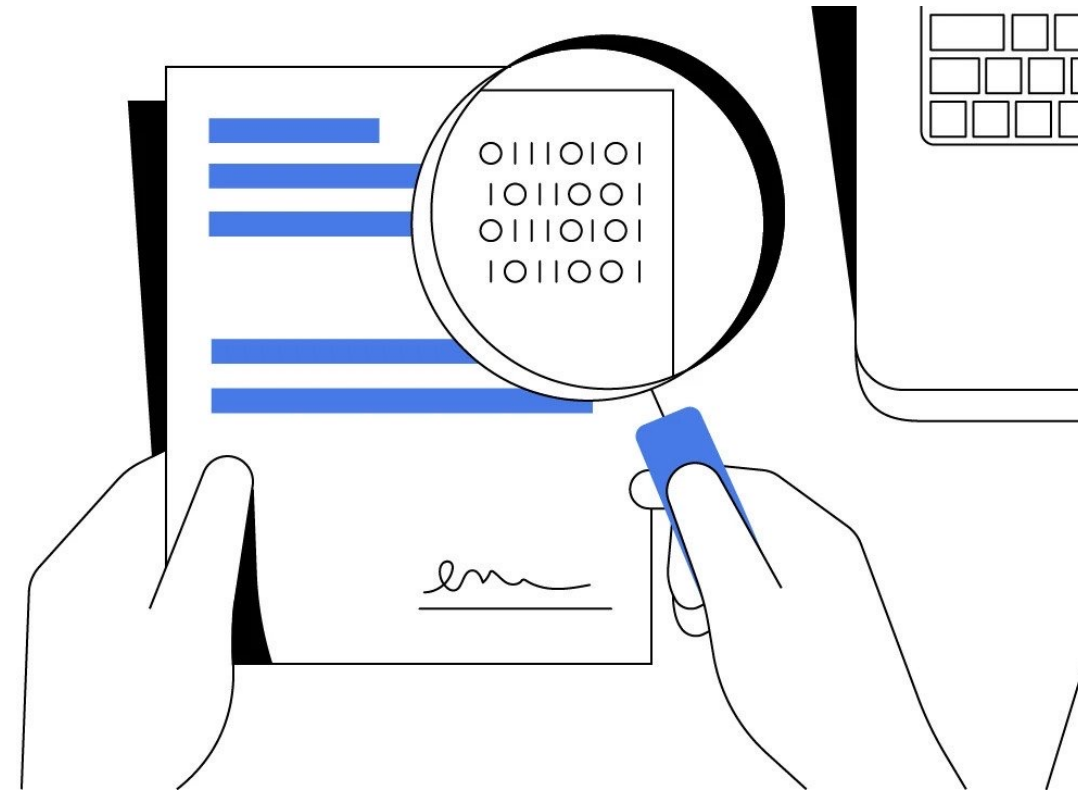
- Most smart contract security measures take place during the development process.
- Unlike traditional systems, smart contracts are nearly **impossible to patch once deployed**.
- Once you write the smart contract to the blockchain, **it is impossible to change the code**.
- For this reason, it's **essential to perform smart contract audit**.





# Smart Contract Audit

- The audit process for a smart contract focuses on **scrutiny of the code** used for underwriting the terms and conditions in the smart contract.
- With the help of such an audit, smart contract developers could **easily identify the vulnerabilities** and bugs before the deployment of smart contracts.



# Why Smart Contract Audit?

- Better optimization of the code
- Improved performance of smart contracts
- Enhanced security of wallets
- Security against hacking attacks



**SMART CONTRACT**  
**AUDIT**

# Responsibilities of Smart Contract Auditors

- **Collecting Code Specifications**

The foremost responsibility of smart contract auditors points to the assessment of a smart contract project's documentation. The evaluation of project documentation could help in developing a comprehensive understanding of the project. You can learn about the use cases, design, and architecture of the smart contract.





# Responsibilities of Smart Contract Auditors

- **Assessment of Code for Vulnerabilities**

The next important addition to the responsibilities outlined by smart contract audit companies would focus on the assessment of the smart contract code. Smart contract auditors have to check the smart contract code line by line and compare it with a list of common vulnerabilities expected in smart contract code.



# Responsibilities of Smart Contract Auditors

- **Testing**

The responsibilities of smart contract auditors also emphasize testing, which helps in the precise identification of code errors and bugs. Auditors can implement unit testing or integration testing, depending on the scale of assessment. For example, unit testing could serve as useful for targeting specific functions.






# Responsibilities of Smart Contract Auditors

- **Reporting**

The final addition to the list of responsibilities for smart contract auditors would refer to reporting. After the audit process is complete, the auditors must work on developing a detailed report for providing specifications of the assessment. Auditors have to create a vulnerability report before publishing the final audit report.






# How to Become a Smart Contract Security Auditor

## 1. Programming Knowledge

- One of the most basic skills required for smart contract audits is programming.
- How will you audit smart contract code if you don't know how to read it?





# How to Become a Smart Contract Security Auditor

## 2. Ethereum and Solidity

- The second step in the journey to becoming a smart contract auditor emphasizes knowledge of [Ethereum](#) and [Solidity](#).
- Ethereum is the most popular blockchain platform for developing [smart contracts](#), and Solidity is the programming language that helps in achieving the objective.

# How to Become a Smart Contract Security Auditor

## 3. Practical Experience with Smart Contracts

- All smart contract audit companies look for professionals with practical experience in smart contract audits.
- Aspiring smart contract auditors must interact with the most popular and commonly used smart contracts.
- As a smart contract auditor, you would encounter such contracts, algorithms, and patterns frequently in your career.
- Therefore, an in-depth understanding of how smart contracts work and their other intricacies can boost your career prospects.



# Solidity

## How to get started?

- As a beginner, you find great tutorials, resources and tools that help you get started building with Solidity on the **ethereum.org** (<https://ethereum.org/en/developers/>) developer portal.
- Alternatively, you can start by learning the basics about blockchain, smart contracts and the Ethereum Virtual Machine (EVM) in the **Solidity docs** (<https://docs.soliditylang.org/en/latest/introduction-to-smart-contracts.html>).



# Some Security Issues

# Integer Overflow

When a single numerical computation is performed, the output **exceeds the maximum** that a register or memory can store or represent.

- In the Solidity programming language, uint8 may represent 256 values ranging from 0 to 255.
- When the uint8 type is used to compute  $255 + 1$  in the actual operation, an overflow occurs, and the calculated result is 0, i.e., the uint8 type's minimal value.

# Integer Overflow

This is how the contract time table works: it allows us to deposit funds, but we have to wait for a minimum of one week before we can withdraw, as well as increase the lock time by calling the “increased lock time” function.

- uint:  $0 \sim (2^{256})-1$
- Initially, lockTime > 1 week
- line 15 may exceed the maximum value of uint
- lockTime will be less than a week
- The contract then fails to maintain the waiting time

```
1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.7.6;
3
4
5 contract TimeLock {
6     mapping(address => uint) public balances;
7     mapping(address => uint) public lockTime;
8
9     function deposit() external payable {
10         balances[msg.sender] += msg.value;
11         lockTime[msg.sender] = block.timestamp + 1 weeks;
12     }
13
14     function increaseLockTime(uint _secondsToIncrease) public {
15         lockTime[msg.sender] += _secondsToIncrease;
16     }
17
18     function withdraw() public {
19         require(balances[msg.sender] > 0, "Insufficient funds");
20         require(block.timestamp > lockTime[msg.sender], "Lock time not expired");
21
22         uint amount = balances[msg.sender];
23         balances[msg.sender] = 0;
24
25         (bool sent, ) = msg.sender.call{value: amount}("");
26         require(sent, "Failed to send Ether");
27     }
28 }
```

# Integer Underflow

Underflow occurs when the computation result is **less than the minimum capacity** of the register or memory to store or represent.

- For instance, using the uint8 type to calculate 0–1 in Solidity will result in underflow.
- So, the calculated value will be 255, which is the highest value that the uint8 type can represent.

# Integer Underflow

The calculation may underflow

```
contract Attack {
    TimeLock timeLock;

    constructor(TimeLock _timeLock) {
        timeLock = TimeLock(_timeLock);
    }

    fallback() external payable {}

    function attack() public payable {
        timeLock.deposit{value: msg.value}();

        timeLock.increaseLockTime(
            type(uint).max + 1 - timeLock.lockTime(address(this))
        );
        timeLock.withdraw();
    }
}
```



# Function Default Visibility

- A function's visibility can be set to **external**, **public**, **internal** or **private**
- **external**: External functions can only be called from outside the contract in which they were declared.

```
contract RandomContract {  
    function externalFunction() external  
    {  
        //performs a task  
    }  
}
```



My Contract

Inherited  
Contract

External  
Contract

# Function Default Visibility

- **internal**: Functions and variables declared with the internal keyword are only accessible within the contract in which they were declared, although they can be accessed from derived contracts.

```
contract RandomContract {  
    function externalFunction() internal  
    {  
        //performs a task  
    }  
}
```

My Contract

Inherited  
Contract

External  
Contract

# Function Default Visibility

- **private**: Functions declared with the private keyword are only accessible within the contract in which they were declared. Private functions are also the only functions that cannot be inherited by other functions.

```
contract RandomContract {  
    function externalFunction() private  
    {  
        //performs a task  
    }  
}
```



My Contract

Inherited  
Contract

External  
Contract

# Function Default Visibility

- **public**: Public functions and variables can be accessed by all parties within and outside the contract. **When the visibility is not specified, the default visibility of a function is public.**

```
contract RandomContract {  
    function externalFunction() public  
    {  
        //performs a task  
    }  
}
```

My Contract

Inherited  
Contract

External  
Contract

# Function Default Visibility

- When the visibility is not specified, the **default visibility of a function is public.**
- Public function can be accessed **by all parties**
- This can lead to a vulnerability if a developer **forgot to set the visibility** and a malicious user is able to make unauthorized or unintended state changes.

```
contract RandomContract {  
    function nFunction() {  
        //performs a task  
    }  
}
```

Attack



```
interface IRandomContract {  
    function nFunction();  
}  
contract Attack {  
    function functionIsExternalCall(address addr) external {  
        IRandomContract(addr).nFunction();  
    }  
}
```