Middle East Technical University
Informatics Institute

Vulnerability Detection on Solidity Smart Contracts by Using Convolutional Neural Networks

Advisor Name: Asst. Prof. Dr. Aybar Can Acar
(METU)

Student Name: Barış Cem Bektaş
(Cyber Security)

January 2023

TECHNICAL REPORT
METU/II-TR-2023-

Orta Doğu Teknik Üniversitesi
Enformatik Enstitüsü

Solidity Akıllı Sözleşmelerinde Evrişimli Sinir Ağları Kullanarak Güvenlik Açığı Tespiti

Danışman Adı: Asst. Prof. Dr. Aybar Can Acar
(ODTÜ)

Öğrenci Adı: Barış Cem Bektaş
(Siber Güvenlik)

Ocak 2023

TEKNİK RAPOR
ODTÜ/II-TR-2023-

REPORT DOCUMENTATION PAGE

1. AGENCY USE ONLY (Internal Use)	**2. REPORT DATE** 26.01.2023
3. TITLE AND SUBTITLE **Vulnerability Detection on Solidity Smart Contracts by Using Convolutional Neural Networks**	
4. AUTHOR (S) Barış Cem Bektaş	**5. REPORT NUMBER (Internal Use)** **METU/II-TR-2023-**
6. SPONSORING/ MONITORING AGENCY NAME(S) AND SIGNATURE(S) Non-Thesis Master's Programme,, Department of Cyber Security, Informatics Institute, METU Advisor: **Asst. Prof. Dr. Aybar Can Acar**	Signature:
7. SUPPLEMENTARY NOTES	
8. ABSTRACT (MAXIMUM 200 WORDS)	
9. SUBJECT TERMS **Blockchain, Smart Contracts, Vulnerability Detection, Deep Learning, CNN**	**10. NUMBER OF PAGES** **55**

8. ABSTRACT (MAXIMUM 200 WORDS)

Smart contracts, which are self executing contracts with the terms of the agreement between buyer and seller being directly written into lines of code, have the potential to revolutionize many industries by automating complex processes and reducing the need for intermediaries. However, the immutability of smart contracts also means that vulnerabilities cannot be easily fixed once they are deployed, making it crucial to detect and prevent vulnerabilities before deployment. In this project, we focus on the problem of vulnerability detection in smart contracts, specifically the reentrancy vulnerability, which allows an attacker to repeatedly call an external contract in a malicious manner. To address this problem, we introduce four-layer convolutional neural network (CNN) for reentrancy vulnerability scanning. We compare our method to other vulnerability scanning tools which are using machine learning approaches, including long short-term memory (LSTM) and graph neural network (GNN), and show that our method outperforms on dataset of real-world smart contracts. Our results demonstrate the effectiveness of using deep learning for vulnerability detection in smart contracts and provide a promising direction for further research in this area.

Vulnerability Detection on Solidity Smart Contracts by Using Convolutional Neural Networks

ABSTRACT

Smart contracts, which are self executing contracts with the terms of the agreement between buyer and seller being directly written into lines of code, have the potential to revolutionize many industries by automating complex processes and reducing the need for intermediaries. However, the immutability of smart contracts also means that vulnerabilities cannot be easily fixed once they are deployed, making it crucial to detect and prevent vulnerabilities before deployment. In this project, we focus on the problem of vulnerability detection in smart contracts, specifically the reentrancy vulnerability, which allows an attacker to repeatedly call an external contract in a malicious manner. To address this problem, we introduce four-layer convolutional neural network (CNN) for reentrancy vulnerability scanning. We compare our method to other vulnerability scanning tools which are using machine learning approaches, including long short-term memory (LSTM) and graph neural network (GNN), and show that our method outperforms on dataset of real-world smart contracts. Our results demonstrate the effectiveness of using deep learning for vulnerability detection in smart contracts and provide a promising direction for further research in this area.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

ETH	Ether (Cryptocurrency of Ethereum)
EVM	Ethereum Virtual Machine
GAS	Cost Of Transfer in Ethereum Network
PoW	Proof of Work
PoS	Proof of Stake
DAO	Decentralized Autonomous Organization
OWASP	The Open Web Application Security Project
ML	Machine Learning
DNN	Deep Neural Network
ANN	Artificial Neural Network
SVM	Support Vector Machine
RNN	Recurrent Neural Network
CNN	Convolutional Neural Network
LSTM	Long-Short Term Memory
BLSTM	Bidirectional Long-Short Term Memory
GNN	Graph Neural Network
AST	Abstract Syntax Tree
NLP	Natural Language Process
DoS	Denial of Service

BTC	Bitcoin
CFG	Control Flow Graph
DASP	Decentralized Application Security Project
ASIC	Application-specific Integrated Circuit

CHAPTER 1

INTRODUCTION

Until Satoshi Nakamoto introduced Bitcoin in 2009, digital currencies were used in a centralized manner. (Nakamoto 2008) He developed a decentralized electronic cash system using cryptographic functions in a clever way. After Bitcoin gained increasing popularity, other cryptocurrencies began to be developed by enthusiasts. Ethereum was one of them and today it is the second most popular blockchain platform, after Bitcoin. (Shen et al. 2018) The Ethereum platform gained popularity through the use of smart contracts, which are computer programs running on the blockchain. Smart contracts contain automatically executing scripts that trigger certain actions or results when the terms of the contract are met. (Röscheisen et al. 1998) They have a non-reversible or changeable structure once they are uploaded to the blockchain. Since anyone can write and upload smart contracts through the Ethereum network, vulnerabilities can occur when amateur programmers write them. This makes smart contracts vulnerable to exploitation by malicious people. When a malicious actor discovers a backdoor in the code, they can abuse the smart contract and take advantage of the opportunity to withdraw people's money from the contract. As a result, smart contract programmers and the blockchain security community have started seeking solutions to prevent these exploitations. Since smart contracts cannot be changed after they are uploaded, vulnerabilities must be addressed before they are put on the blockchain. Therefore, vulnerable statements in the code must be carefully examined during the development phase. Some well-known methodologies, such as automated software testing (fuzzing), have been applied to smart contracts for this purpose. (Jiang et al. ,2018) Static and dynamic analysis tools have been developed to efficiently find bugs in smart contracts. However, in some cases, these tools do not provide the expected high accuracy results in vulnerability testing. In order detect to tricky vulnerabilities that cannot be detected using tools and increase accuracy, more advanced methods need to be used. Machine learning techniques have been used in software vulnerability detection since the early 2000s. (Chernis et al. ,2018) Because smart contracts are also a small size of software, the use of machine learning methods in vulnerability detection for them has become prominent in this area. Many different types of ML methods, such as Convolutional Neural Network (CNN), Recurrent Neural Network (RNN), Long-Short Term Memory (LSTM), and Graph Neural Network (GNN), have been used to scan smart contract code for vulnerabilities. Vulnerabilities in smart contracts are also divided into different categories. The Decentralized Application Security Project (DASP), a recognized organization in blockchain security, has created a Top 10 list of the most risky and common smart contract code vulnerabilities. (Forbes Business Council, 2022) According to the DASP list, an attack called reentrancy is the top vulnerability on the current Top 10 list for 2018. (Forbes Business Council, 2022) In fact, one of the most impactful attacks on the Ethereum DAO, which resulted in a 60 million dollar loss, was conducted by exploiting a reentrancy bug. (Alchemy,2008) Since the reentrancy bug is more common and causes more harm than any other vulnerabilities, we focus on detecting reentrancy bugs in smart contracts in this work. After evaluating the advantages and disadvantages of deep learning models, we decided to use a Convolutional Neural Network (CNN) for vulnerability detection purposes. To train our model, we downloaded open-source contracts labeled as having or not having a reentrancy bug. We used the Word2vec technique as a Natural Language Processing (NLP) model to feed our neural network with code statements. In this paper, we provide information about two distinct topics. First, we will discuss blockchain technology, and second, we will give an overview of machine learning basics and methods. After that, we will examine static, dynamic, and ML tools and their accuracy rates in vulnerability detection. Then we will develop a machine learning method for detecting vulnerabilities in blockchain technology (specifically Ethereum smart contracts) and compare its results to tools using different methods

CHAPTER 2

BACKGROUND

Our research is based on two main pillars. In this section, we examine these two pillars and their subtopics in order to provide the necessary knowledge for understanding the subsequent sections of this paper. The first main subject is blockchain technology, and we also provide necessary knowledge on the subtopics of the Ethereum platform and smart contract basics. The second main topic in our research is machine learning, and we will provide information on the subtopics of machine learning such as deep learning, neural network models, and finally the ML model that we will use in this work, which is CNN.

2.1. Blockchain Technology

A blockchain is a distributed ledger that contains a record of all transactions since the beginning of its existence. (Alharby et al., 2017) Copies of this ledger are held by the participants in the blockchain network, known as nodes. This allows the transaction records on these ledgers to be verifiable with each other and ensures that all nodes can trust the authenticity and verifiability of these records. As a result, network participants do not need a trusted third party. This technology enables peer-to-peer money transactions by its users. (Halas, 2019) The shared ledger is divided into blocks in the blockchain. Blocks are chained together using cryptographic hash functions and are continuously added to the blockchain network. Each block contains the hash of the previous block, so that is allowing the blocks to be chained.

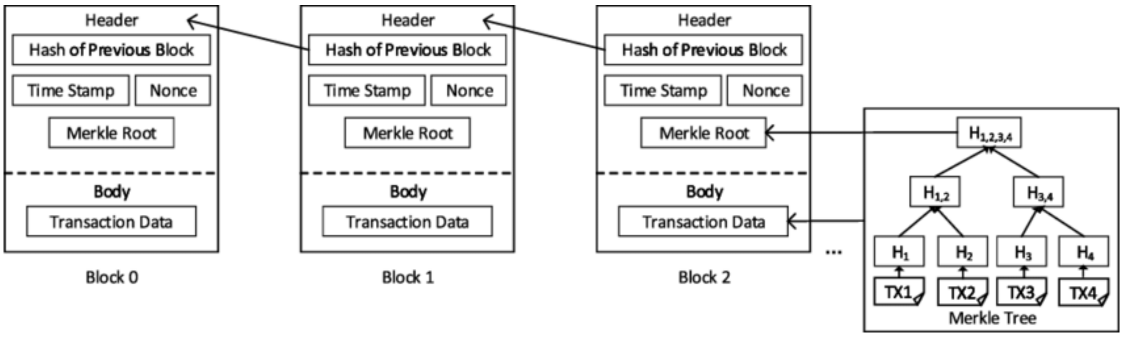


Figure 1: The Block Structure of Blockchain

In Bitcoin, first block called *genesis block* created by Satoshi Nakamoto who founder of Bitcoin. (Nakamoto, 2008) After first block had created six blocks(approx.) have being continuously created in every hour. (Hammami, 2017)Block creation process called mining being caried out by miners and it works through consensus mechanism. Consensus mechanism is a validation mechanism which is used hinder to multiple spending a cryptocurrency and it makes tamper proof of a blockchain system. Consensus mechanism relies on Byzantine Fault Tolerance algorithm which states that as long as majority nodes are honest, a blockchain system can be secure against malicious attacks. (Zhang et al., 2019) Most popular consensus mechanisms are Proof of Work (PoW) and Proof of Stake (PoS). (Halas, 2019)

While Proof of Work based on solving hash puzzles by dedicated CPU power of miners, Proof of Stake based on number of staked coins miners have in the system. (Investopedia) PoW based mining is time consuming and computationally expensive as well as its difficulty level is increasing after every 2.016 blocks are created. (Coindesk, n.d.) First blockchain system which is Bitcoin has been using PoW consensus algorithms.

First blockchain system Bitcoin tried to provide benefits on user below;

1- Decentralization: Bitcoin ledger is distributed through nodes which means one need to send cash to others, does not need to trust third parties.

2- Non-Reversible: After consensus is formed on last six blocks (apprx.1 hour pasts), it is no longer computationally feasible to reverse the transaction.

3- Fault Tolerance: Since many nodes sharing the same ledger, even if some of them get downed, blockchain system would work seamlessly. Thus, bitcoin is resistant to SoF(Single Point Of Failure)

4- Tamper proof: Bitcoin consensus algorithm PoW rely on Byzantine General Problem. If majority (%51) nodes are honest, system is safe against malicious attacks.

5- Open source: Anyone can download bitcoin source code and bitcoin blocks from blockchain. Hence, Bitcoin working principle and backbone are transparent to public. (Nakamoto, 2008)

After Bitcoin has gained popularity other blockchain systems has appeared. One of the most popular ones is Ethereum which is using Proof of Stake consensus algorithm today.

2.1.1. Ethereum

Ethereum is the second most popular blockchain system, introduced by Vitalik Buterin in 2014 and launched in 2015. (Ethereum,2022) Unlike the completely ownerless structure of Bitcoin, Ethereum is backed by the Ethereum Foundation, a non-profit organization that aims to support and develop the Ethereum blockchain network and ecosystem. (Ethereum Foundation,2022) Ethereum used Proof of Work (PoW) consensus algorithms until September 2022, after which it switched to Proof of Stake (PoS) with the ETH2.0 upgrade. In Ethereum's PoS system, nodes can become validators if they hold a specific amount of Ether in their accounts. This allows them to contribute to the Ethereum network and earn rewards.

(Ethereum ,2022) PoS does not require the CPU power or electricity consumption that PoW does, making it more energy-efficient and accessible to anyone without the need to buy and set up specialized hardware such as GPUs and ASICs. This makes it easier to achieve decentralization compared to the Bitcoin blockchain network structure. (Ethereum,2022) Ethereum also has an unlimited supply, unlike Bitcoin's hard-coded limit of 21 million BTC. This makes Ether slightly inflationary compared to Bitcoin. In addition to these differences, the main advantage of Ethereum over Bitcoin is that it is both a computing platform and an electronic cash system. Ethereum has the Solidity programming language, which is Turing complete and runs on the Ethereum Virtual Machine on the Ethereum blockchain network. (Arkangelo, 2019) Thanks to Solidity, not only can cash be transferred between parties, but data can also be transferred over the network. This feature has led the Ethereum development team to call it "The World Computer." (Arkangelo, 2019)

2.1.2. Smart Contracts

A Smart Contract is kind of a virtual contract that formed by compact-sized code that is loaded and running on blockchain network. It has two distinct feature which are terms or conditions and actions or results. When conditions are met, Smart Contract code is executed by automatically and give some results or actions predetermined. (Alharby et al., 2017) Smart Contract idea was first suggested by Nick Szabo in 1996. (Szabo, 1996) He mention about smart contract as "*a set of promises, specified in digital form, including protocols within which the parties perform on these promises*" Until Nakamoto introduce Bitcoin in 2009 it had not been getting attention much.

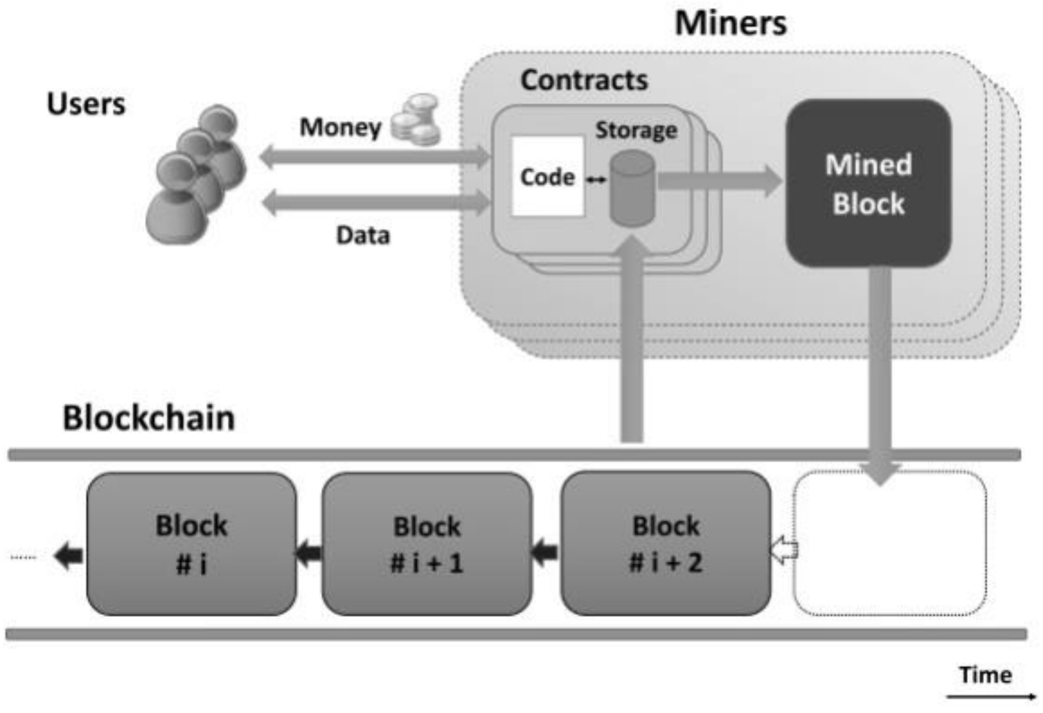


Figure 2: Smart contract system

Smart contracts contain validation counter, account balance and data storage. Counter is integer value that serves to validate transaction, account balance is amount of Ether that contracts have a specific time and data storage is code information of Contract. When a transaction smart contract conditions are triggered by transfer or data which comes from blockchain network, bytecode executed through EVM (Ethereum Virtual Machine), and account balance of contract updates. Each running of smart contract code needs transaction payment in terms of Ether. (Alharby et al., 2017) This payment called as GAS , this GAS money goes to Miners in order motivate them to keep storage of Ethereum blockchains and maintain to the network. (King of the Ether, 2022) One key feature of smart contracts is that they cannot be altered or changed on their running principle once they uploaded on blockchain. Smart contract codes are open to public as blockchain transactions are, hence, anyone who wants to, download these codes to himself/herself own devices and examines them.

Besides use cases of smart contracts mentioned above, there are new blockchain application areas has came after smart contracts started to be widely used. One of the most important ones is Dapp (Decentralized Application). A Dapp is an application that code is written by Solidity executing on Ethereum blockchain network. Contrary to classic applications which are running on single server Daap's do not have single point of failure since its backend holding through blockchain. (Margaritis, 2021)

2.1.3. Smart Contract Vulnerabilities

One important feature of Smart Contracts is that their code is open to the public, as mentioned in section 2.3. The open-source structure makes them easily exploitable. Since anyone can access and analyze the code, malicious individuals can discover bugs and software vulnerabilities in it. When these vulnerabilities are discovered, they can be exploited to take control of the Smart Contract or withdraw its account balance. Since Smart Contracts running on the blockchain network are non-modifiable and non-reversible, their insecure code needs to be cleaned of bugs and insecure code structures before being uploaded to the network. (David et al., 2022)

2.1.4. The DASP Top 10 Attacks

DASP (Decentralized Application Security Project) is an initiative conducted by NCC Group which aiming to identify most impactful and frequently seen vulnerabilities. For this purpose they uncovered Top 10 smart contract vulnerabilities list in 2018. (Currencyrate.today, 2022) We are going to examine these vulnerabilities in order understand to their types and effect.

2.1.4.1. Reentrancy

Reentrancy is the most important vulnerability a smart contract may have. It is a bug that allow an external contract making multiple calls to itself over victim contract while other calls itself are continuing. (DASP,2022)

One of the worst Smart Contract hacks in history was DAO hack. Definition of DAO is Decentralized Autonomous Organization. DAO is a running software which is uploaded on blockchain, and it has owned by anyone. Participants have vote rights over decisions will be taken by proportion to the fund they invested on. (Szabo, 1996) As is valid for all smart contracts a DAO as well is automatically executing once it runs. Hence if there is a vulnerability or a bug in its code. It is wide open to be attacked by malicious actors. In DAO hack it was happened. The DAO members could produce child DAO's if they wanted to split from main code. Hacker(s) discovered bug known as infinite loop in DAOs' code and they made recursive call to split function. (Hsieh, 2018) Consequently, they made to stole 3.6 million Ether which is $60 million dollar at the time was incident happened. It makes roughly $4.3 billion dollar in today prices'. (Coindesk, 2016)

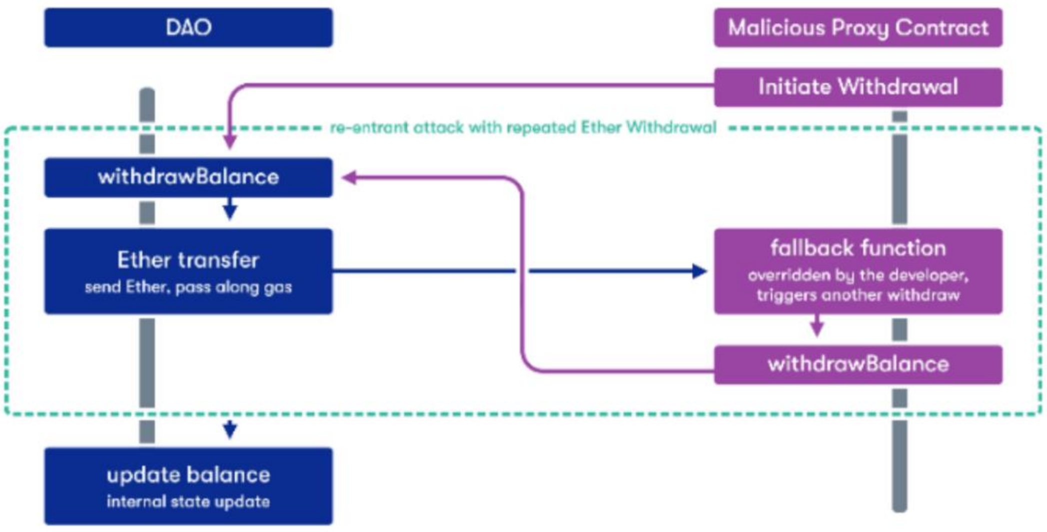
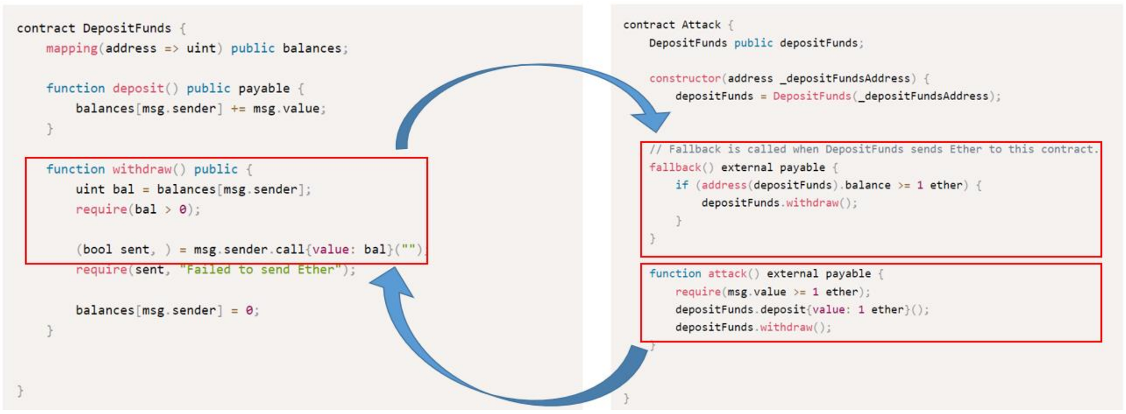


Figure 30: Representation of Reentrancy Attack



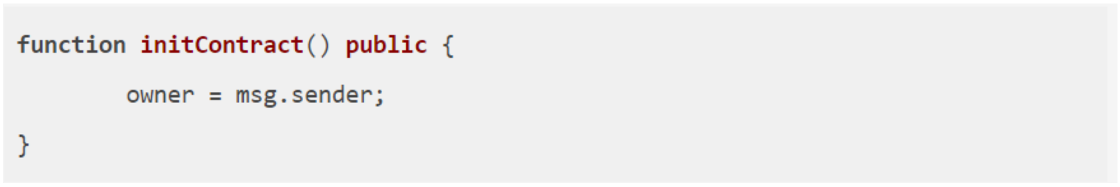
```
contract DepositFunds {
    mapping(address => uint) public balances;

    function deposit() public payable {
        balances[msg.sender] += msg.value;
    }

    function withdraw() public {
        uint bal = balances[msg.sender];
        require(bal > 0);

        (bool sent, ) = msg.sender.call{value: bal}("");
        require(sent, "Failed to send Ether");

        balances[msg.sender] = 0;
    }


}
```

```
contract Attack {
    DepositFunds public depositFunds;

    constructor(address _depositFundsAddress) {
        depositFunds = DepositFunds(_depositFundsAddress);


    // Fallback is called when DepositFunds sends Ether to this contract.
    fallback() external payable {
        if (address(depositFunds).balance >= 1 ether) {
            depositFunds.withdraw();
        }
    }

    function attack() external payable {
        require(msg.value >= 1 ether);
        depositFunds.deposit{value: 1 ether}();
        depositFunds.withdraw();
    }


}
```

Figure 4: Re-entrance bug showing in Solidity code

2.1.4.2. Access Control

Access Control is not only Smart Contract vulnerability problem but also general cyber security issue which OWASP (Open Web Application Security Project) listed as Top 1 Cyber Security Problem as of 2021. Access Control refers which user can reach to which data or specific partition of a software program. (David et al., 2022) Since, user may have different level of authorization such as regular user or admin, different types of users have different access rights to reach resources of a software potentially hold. If an attacker tries to gain more access right than he/she has, we can mention about an Access Control Attack. Through this way an attacker' escalate of his privilege and his main aim is to capture resources more than he should.(Leander, 2022) In Smart Contract, if a user can capture ownership of contract by making call to this contract, he can withdraw all money contract holds.



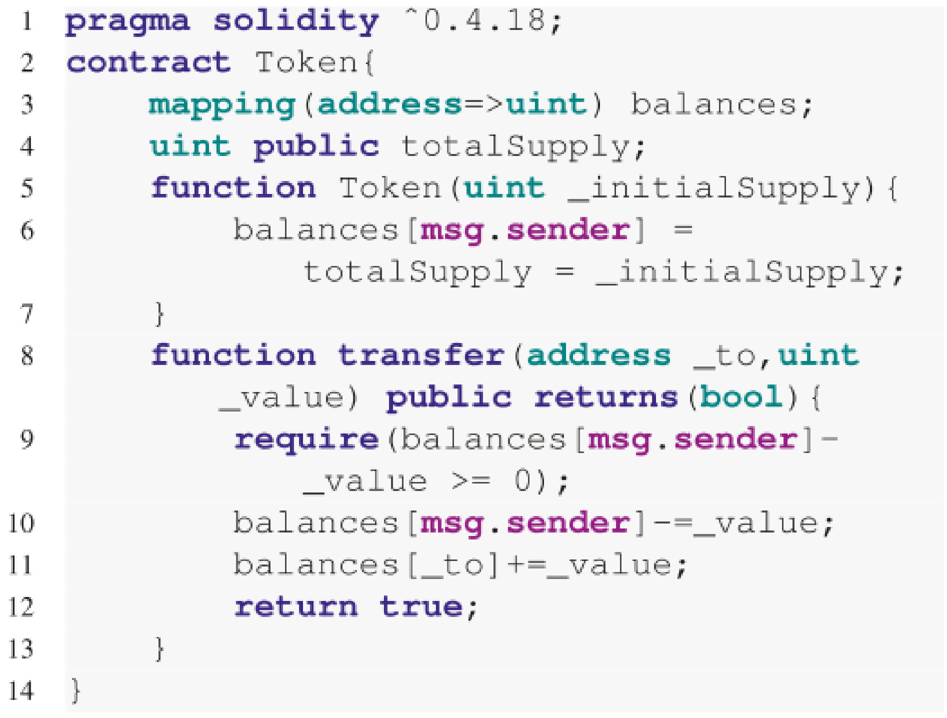
```
function initContract() public {

        owner = msg.sender;

}
```

Figure 4: Broken Access Control on a Smart Contract

As it seen on Figure 4, msg sender is attended as contract owner. A malicious actor may get over control of this control because of this code failure.

2.1.4.3. Arithmetic Issues

Arithmetic vulnerabilities are a common type of vulnerability. They occur when an integer overflow or integer underflow happens. In Solidity, there is an 8-bit unsigned number that can hold a maximum of 256 integer values between 0 and 255. If the result of an operation is more than 255, an integer overflow occurs. On the other hand, if the result of an operation is less than 0, an integer underflow occurs. (Dingman et al., 2021) Since 255+1 results in 0 and 0-1 results in 255 in Solidity, overflows and underflows can cause the software to behave differently than intended. This can allow attackers to manipulate the smart contract to behave as they want. (Redfoxsec, 2022)



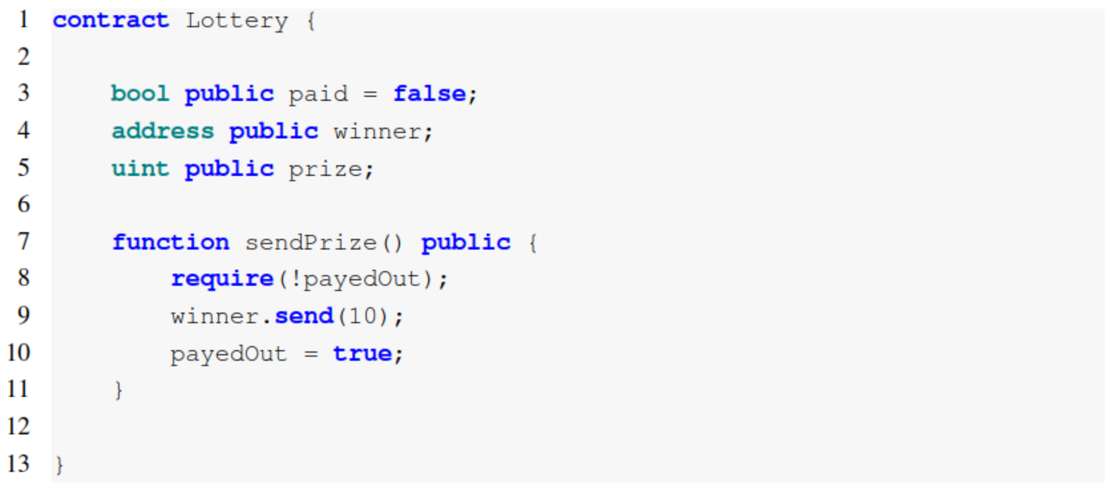
```
pragma solidity ^0.4.18;
contract Token{
    mapping(address=>uint) balances;
    uint public totalSupply;
    function Token(uint _initialSupply){
        balances[msg.sender] =
            totalSupply = _initialSupply;
    }
    function transfer(address _to,uint
        _value) public returns(bool){
        require(balances[msg.sender]-
            _value >= 0);
        balances[msg.sender]-=_value;
        balances[_to]+=_value;
        return true;
    }
}
```

Figure 5: Integer Underflow Vulnerability in Smart Contract

On line 9, the contract requires that the account balance of the caller (msg.sender) should be greater than the number of tokens they want to transfer (_value). Normally, if the account balance of the caller is 0 and they want to transfer 1 token, 0-1 results in -1. Thus, they cannot meet the condition on line 9, which states that the balance of the caller minus the number of tokens they want to transfer must be a positive integer. However, due to the integer underflow bug, 0-1 results in 255 in Solidity, and the condition on line 9 can be met.Dingman et al., 2021) As seen in the example above, arithmetic problems make Solidity contracts vulnerable to exploitation by malicious actors.

2.1.4.4. Unchecked Return Values for Low Level Calls

In Solidity, functions like address.call(), address.send(), and others are referred to as low-level functions, as they use the same opcode called "call()". (Wikipedia, 2022) This opcode is used to transfer funds from one contract to another. If call() fails, the function returns false. However, even if the return value is false, the contract's execution will not be reverted. This means that if a contract is executed without checking the return value, the contract's balance will be decreased even if the return value is false.



```
contract Lottery {

    bool public paid = false;
    address public winner;
    uint public prize;

    function sendPrize() public {
        require(!payedOut);
        winner.send(10);
        payedOut = true;
    }

}
```

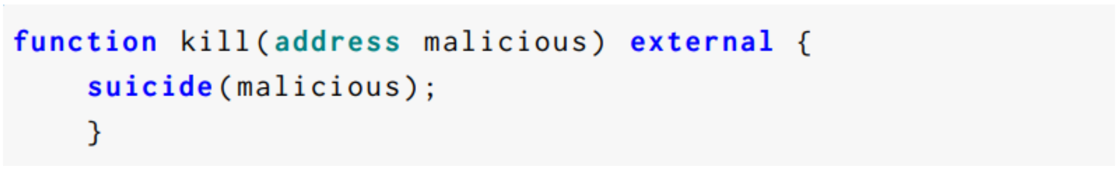
Figure 6: Unchecked low level calls example

In the example above, there is a prize of 10 ETH being sent to a winner. The payedOut() function is set to true without waiting for the response of the contract being called. As a result, the sendPrize() function is executed and the balance is decreased, locking the prize money inside the contract irreversibly. However, even if the transaction fails and the winner.send() function returns false, the balance will still be decreased, as if the transfer process was completed successfully. (David et al., 2022) This vulnerability, known as unchecked return values for low-level calls, can be prevented by checking the return value of the send() opcode and throwing an exception if it is false. In other words, this vulnerability is a missing exception failure.

2.1.4.5. Denial of Service

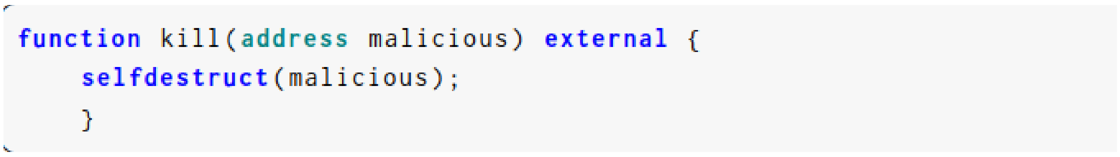
Denial of Service Attack (DoS Attack) is making unavailable an information system such as server, network, or other machines for their users.(Raikwar et al., 2022) It is well known attack, and we face this on web frequently. An attacker overwhelms a website server or a service for a specific time frame such as one day or a week. Then, we apply necessarily mitigation technique such as writing on new firewall rules or blocking attackers Ip’s or some regions that attacks come from and through these arrangements we find solution eventually and we can take up service. On the other hand, we mentioned that Smart Contracts cannot be modifiable after they upload on blockchain on Section 2.3. Hence, when a Smart Contract is subject of DoS attack, it can be irreversibly lost its functionality. *“Denial of service is deadly in the world of Ethereum: while other types of applications can eventually recover, smart contracts can be taken offline forever by just one of these attacks.”* (Currencyrate.today, 2022) There are some known ways that is applied DoS attack to a Smart Contract. Most important three ones are; First, asking result of computationally time taken operation by calling another smart contract in order make to unavailable it. Second, making out of gas it by asking multiple refund to multiple addresses. Third, cancelling refund by using fallback function of Smart Contract. (Samreen et al., 2020)

Sometimes, smart contracts developers put suicide() or selfdestruct() function code inside of smart contract so that they can alter it if malicious activities happens on there. However, if ownership of contract is not asked by smart contract in solid way, malicious actors can trigger this functionality of contract and terminate Smart Contract. (Eskandari et al., 2019)



```
function kill(address malicious) external {
    suicide(malicious);
    }
```

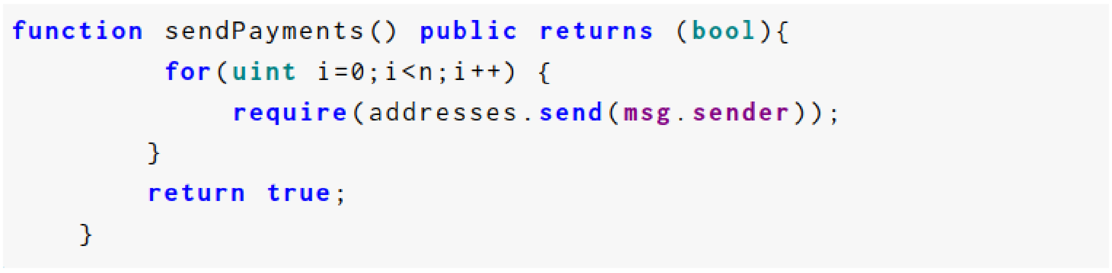
Figure 7: Suicide() function example



```
function kill(address malicious) external {
    selfdestruct(malicious);
    }
```

Figure 8: Selfdestruct() function example

As it can be seen on Figure 7 and Figure 8, Smart Contract kill functions which are suicide() and selfdestruct() are not questioning ownership of the contract. Hence, malicious actors can trigger this function by making call to the contract by another contract that they may use. It can be results with killing this contract.



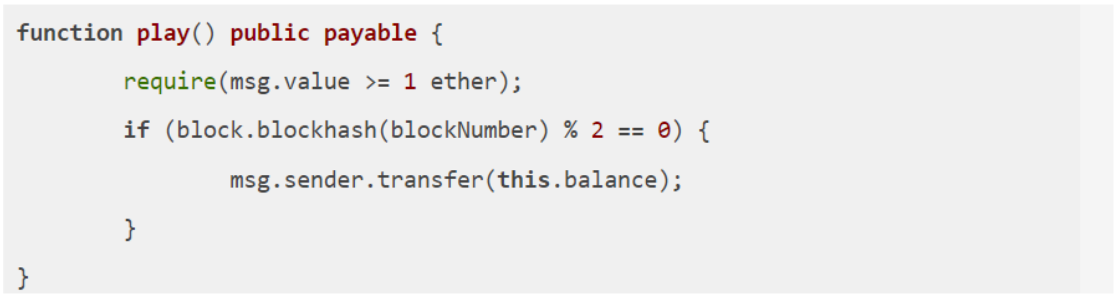
```
function sendPayments() public returns (bool){
        for(uint i=0;i<n;i++) {
            require(addresses.send(msg.sender));
        }
        return true;
    }
```

Figure 9: Loophole Vulnerability for DoS Attack

In example of Figure 9 we see send() opcode in a for loop. If this transaction fails, for loop locks and it makes smart contract unavailable. This code flaw can be used for applying DDoS attack for malicious actors.

2.1.4.6. Bad Randomness

Bad randomness becomes a vulnerability when a smart contract uses a random generator to produce a number for giving prize money or making a transaction to a winner in a game. Since smart contract codes are open to the public, anyone can see the method of generating pseudo-random numbers. When an attacker figures out the next result of the random generator, they can manipulate the smart contract and take money from it through this method. (David et al., 2022)



```
function play() public payable {
        require(msg.value >= 1 ether);
        if (block.blockhash(blockNumber) % 2 == 0) {
                msg.sender.transfer(this.balance);
        }
}
```

Figure 10: Bad random generator in a Smart Contract

In Figure 10, if random generator of block.blockhash() is using current block number or number of before than 256 block it will be safe as randomness. However, if it uses previous block number for instance, an attacker contract can find previous block’s number and can calculate its hash eventually he/she can find random number generator’s result. Attacker can gain advantage over this result by winning a lottery for instance.

In order hinder to this vulnerability, random number should come from outsources (i.e. A web server) to the smart contract, mechanism underlying of random generator can be hidden thanks to this way. (David et al., 2022)

2.1.4.7. Front Running

Ethereum miners take Ether, which is called Gas, as a fee in return for their efforts in running codes on the Ethereum Blockchain Network. Therefore, users who transfer cash or send code to smart contracts can determine the fee price for their own payload. They do this so that their code will be run on the network before others' code. Naturally, miners are more eager to process codes that have higher fees. This is called a gas auction. (Alharby et al., 2017) Transactions and their payloads on the Ethereum blockchain network can be seen by everyone. If a code output is valuable information, a malicious actor can replicate the code and upload it onto the network as if it were his own code, with a higher gas fee. As mentioned above, codes that promise higher Gas fees are processed before others' code. In our example, the attacker's replicated code will be run by miners before the copied code, allowing the attacker to receive the reward of the code output.

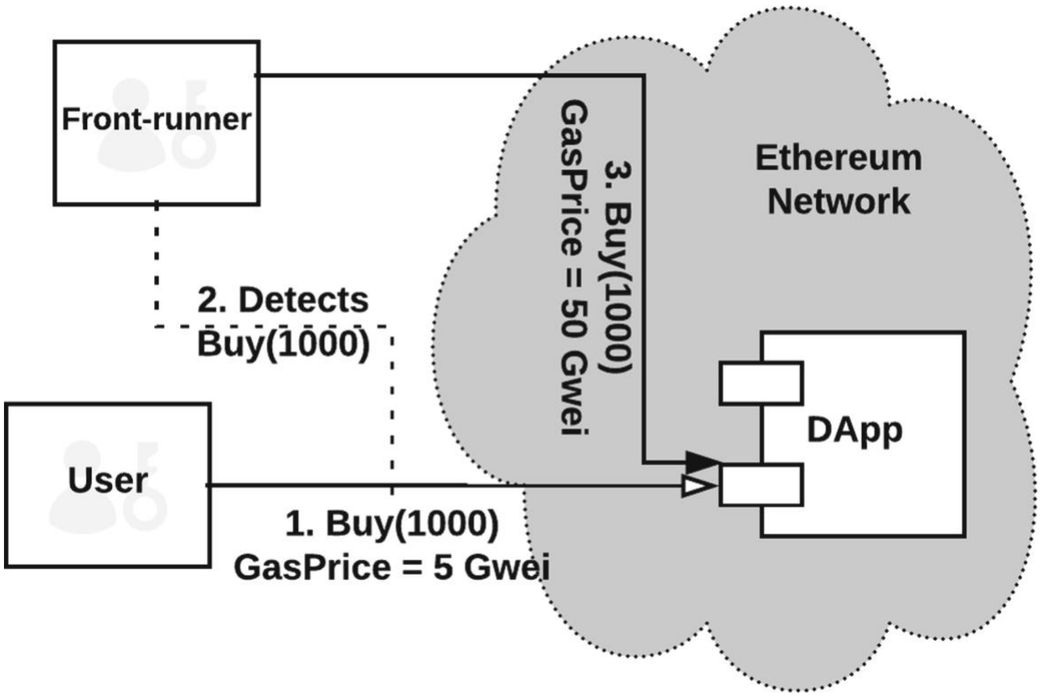
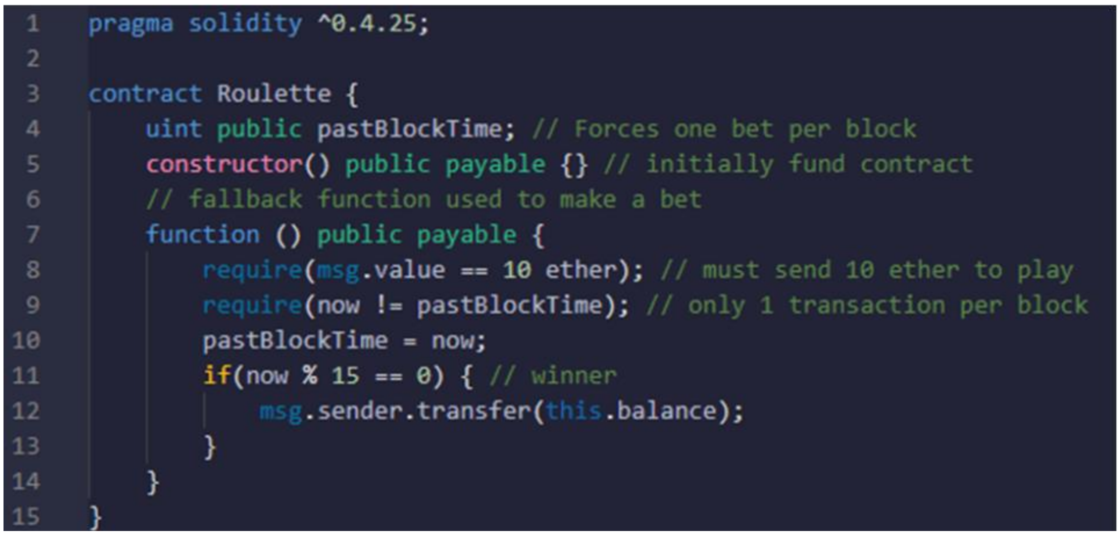

Front-runner
Ethereum Network
3. Buy(1000) GasPrice = 50 Gwei
2. Detects Buy(1000)
DApp
User
1. Buy(1000) GasPrice = 5 Gwei

Figure 11: A Front runner Attack representation on Ethereum Network.

In example above, Front-runner attacker steal user's code from network and he/she upload to the network back with promising higher fee. Consequently, attacker gets prize money of code output which is 1000 ETH.

2.1.4.8. Time Manipulation

In blockchain network, blocks have timestamp indicates the time when a block is mined. A block timestamp is determined by the local time zone of the miner who mined the block. However, miners have elasticity of about 15 minutes to declare when they mined a block. (SecureWorld, 2022) When a smart contract uses opcode such as timestamp.block(), now() or similar ones that is pointing to timestamp of a block in order select to sending money to another address, timestamp vulnerability occurs. This vulnerability can be exploited by dishonest miners.



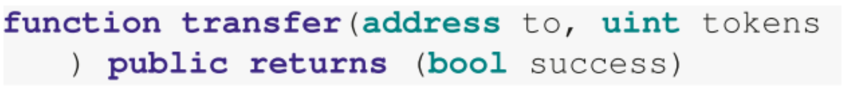
```
pragma solidity ^0.4.25;

contract Roulette {
    uint public pastBlockTime; // Forces one bet per block
    constructor() public payable {} // initially fund contract
    // fallback function used to make a bet
    function () public payable {
        require(msg.value == 10 ether); // must send 10 ether to play
        require(now != pastBlockTime); // only 1 transaction per block
        pastBlockTime = now;
        if(now % 15 == 0) { // winner
            msg.sender.transfer(this.balance);
        }
    }
}
```

Figure 12: Timestamp dependence example in a Smart Contract

In example above, a smart contract determines winner of Roulette game according to timestamp of block which players send ether. When a players' transaction block time (now) modulo 15 equals to 0, that player wins the game and gets all money on the contract. The vulnerability of this code is that a miner can determines timestamp a block 900 second forward or backward from right now. Hence, he/she can arrange timestamp of block he mines equals 0 when it is divided by 15. So that he can win the game and withdraw all money on Smart Contract. (Chen et al., 2019)

2.1.4.9. Short Address Attack

Short Address attack is based on explosion of an EVM (Ethereum Virtual Machine) vulnerability. EVM expects 32 byte long character as transferring address [38] When the address is not given as 32 byte, EVM automatically adds padding as zero (0) in order make to the address suitable its format. However, EVM is not checking that if address is valid or not. Thus, a malicious actor can generate special address when is padded turn into explosion. In example below, transfer function accepts any address as a valid address.



```
function transfer(address to, uint tokens
    ) public returns (bool success)
```

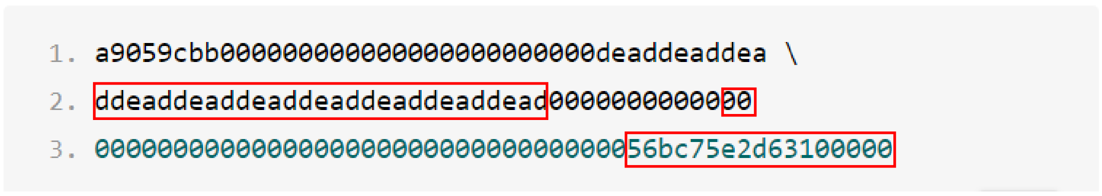
Figure 13: Example of Short Address Attack

A transfer normally occurs two parts. First one is address part which is assuming 32byte and second one is amount part which is 32 bytes also. Since, address part and tokens that transferring encodes together, it can be manipulated if address parts' one zero will be deleted.

We assume that Alices' address is

(`0xdeaddeaddeaddeaddeaddeaddeaddeaddeaddead`)

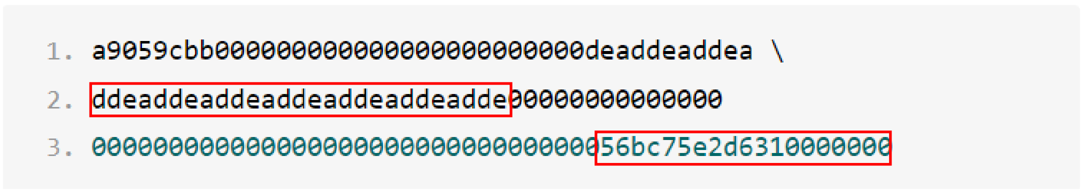
Alice wants to withdraw her 100 tokens from an exchange web site to her address as stated above. Here is below a normal non -tricky transfer.



```
1. a9059cbb000000000000000000000000deaddeaddea \
2. ddeaddeaddeaddeaddeaddeaddead0000000000000
3. 000000000000000000000000000000000056bc75e2d63100000
```

Figure 14: Non tricky transfer

In line 1, first part which is until zeros start is trigger transfer function. Right part contains both address and transfer amount together. When its decoded, line 2 and line 3 is extracted. Hence, line 2 represents address and line 3 represents transfer amount which is 100 Ether in this example.



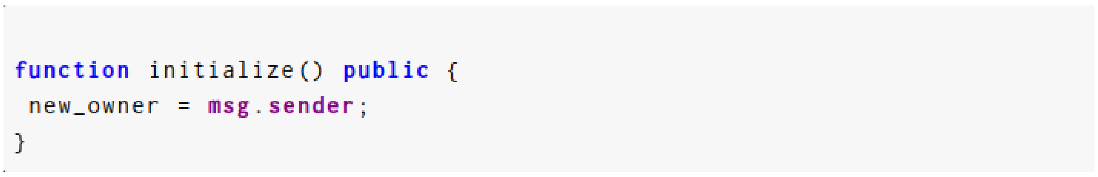
```
1. a9059cbb000000000000000000000000deaddeaddea \
2. ddeaddeaddeaddeaddeaddeadde00000000000000
3. 000000000000000000000000000000000056bc75e2d6310000000
```

Figure 15: Showing that when one zero is deleted

Since, address part and transfer amount part are encoded and decoded together. When is deleted one zero from the address part, EVM is applying padding on transferring amount part instead of address part. Thus, when one zero is deleted, EVM adds two zero to amount part and this value makes 25600 ethers instead of 100 ethers, since its multiplied with 256. As a result, while manipulating address value, a malicious actor can withdraw way more money that he shows to smart contract. (Ethereum Book, 2022) One simple and effective way hinder to this vulnerability is that checking address longevity if it is 32 bytes long or not before accepting it as a valid address.

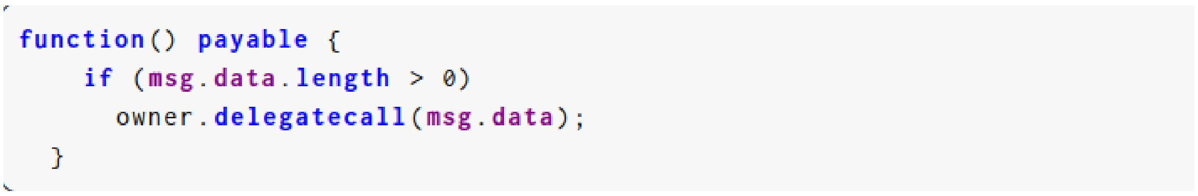
2.1.4.10. Calls to the Unknown Vulnerability

Smart Contracts are not only called by another user for transferring his/her funds, but they can also call other smart contracts', or they can called by other smart contracts. The problem is about that, malicious actors can write malicious smart contracts. A honest smart contract codes can be triggered in malicious way. For instance, when a contract runs delegatecall() to call other smart contract, second contract code gets run. Malicious contract can repeat this process many times and it can lock vulnerable contract code by it gets running repeatedly. (Eskandari et al., 2019)



```
function initialize() public {
 new_owner = msg.sender;
}
```

Figure 16: Calls to the Unknown Vulnerability



```
function() payable {
    if (msg.data.length > 0)
      owner.delegatecall(msg.data);
  }
```

Figure 17: Calls to the Unknown Vulnerability Example 2

On above, delegetacall() function runs on initialize function of contract. Thus, malicious contract can get control of vulnerable contract through this way.

2.2. Machine Learning

Machine learning is the study of computer science that enables computers to learn automatically from data through experience. (Mitchell, 1997) Machine learning algorithms build a model by extracting information from training data, which is given to them. Machine learning approaches are divided into two categories: supervised learning and unsupervised learning. (IBM, 2022) While supervised learning works with labeled data, unsupervised learning finds its own labels. Supervised learning is the method of training a machine learning algorithm through example inputs and their corresponding labels. The goal of the algorithm is to increase the accuracy of labeling input data as much as possible. The more training data the algorithm is provided, the better the results of the model. (Liu et al., 2012) According to their tasks and output variables, supervised learning algorithms are divided into two subsets: classification and regression. The output variables of classification tasks should be categorized, such as the brand of a car, the color of a pencil, or the marriage status of a person. The output should be limited to a set of values. On the other hand, the output of regression tasks must be a continuous variable, such as the degree of air temperature, the price of a bicycle, or the height of a person. In essence, supervised learning algorithms are used to detect relationships and differences of degree between two objects. (Apaydin et al., 2020) On the other hand, in unsupervised learning, input data is unlabeled and given to an algorithm, which then builds patterns and discovers relationships within the data. There are two main types of unsupervised learning tasks: clustering and association. Association tasks focus on discovering patterns and rules related to the input data, such as predicting political tendencies of people. Clustering tasks, on the other hand, involve grouping data based on specific features, like grouping similar customer profiles. [46]

2.2.1. Human Neural Networks and Artificial Neural Networks

Artificial Neural Networks (ANNs) are computational models inspired by the structure and function of the human nervous system. In the human neural system, cells called soma serve as the basic unit of information processing. These units receive input through their dendrites, process it, and send the processed information to other cells through axons. (Mehtab et al.2017) Similarly, in ANNs, neurons are connected to each other in a network structure. The units in ANNs are called neurons or nodes. Each node has input connections, called dendrites, and output connections, called axons, to connect with adjacent nodes. These connections are used to transmit information between neurons in the network. The architecture of the ANN and the way the neurons are connected to each other determines the capabilities and performance of the network. ANNs have been successful in a wide range of applications including image recognition, natural language processing, and decision-making.

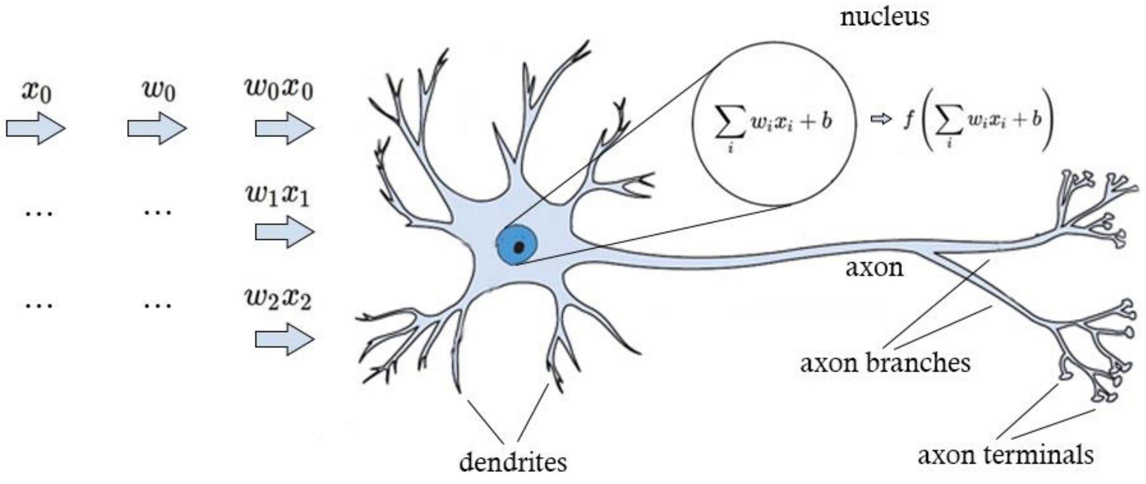


Figure 18: A Biological Neuron

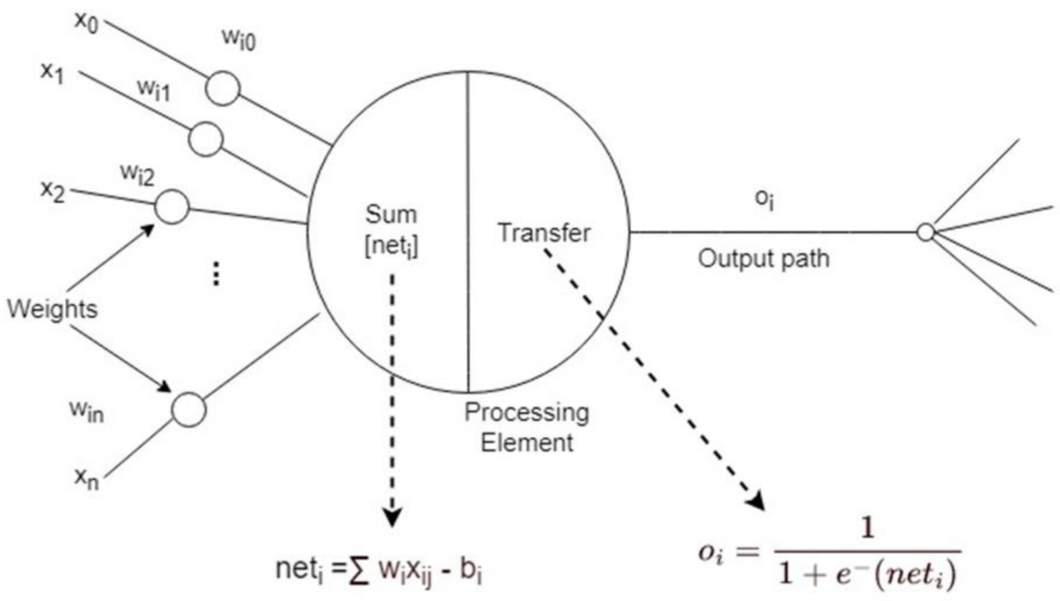


Figure 19: An Artificial Neuron

Inputs received by an Artificial Neural Network (ANN) have specific weights, which can be either positive or negative. Positive values activate a node, while negative values inhibit it. The neuron sums the received values of signals by multiplying them by their weight. The output of the summing process is passed through a transfer function, typically a logistic function, to be processed, and the final output is sent to other neurons. (Laakso, 2022)A back-propagation function was developed for the ANN structure. The aim of the back-propagation algorithm is to minimize error in a feed-forward neural network. The back-propagation algorithm takes the output value from the network's output layer and feeds it back to the network's input layer to decrease the error rate from the previous process. In each iteration, the weights of connections are determined again until the output error rate is acceptable. (Buscema, 1998) In a classic ANN, there are three layers formed by groups of neurons: the input layer, the hidden layer, and the output layer. This earliest form of ANN model is sufficient for most calculations, such as predicting house prices or classifying objects.

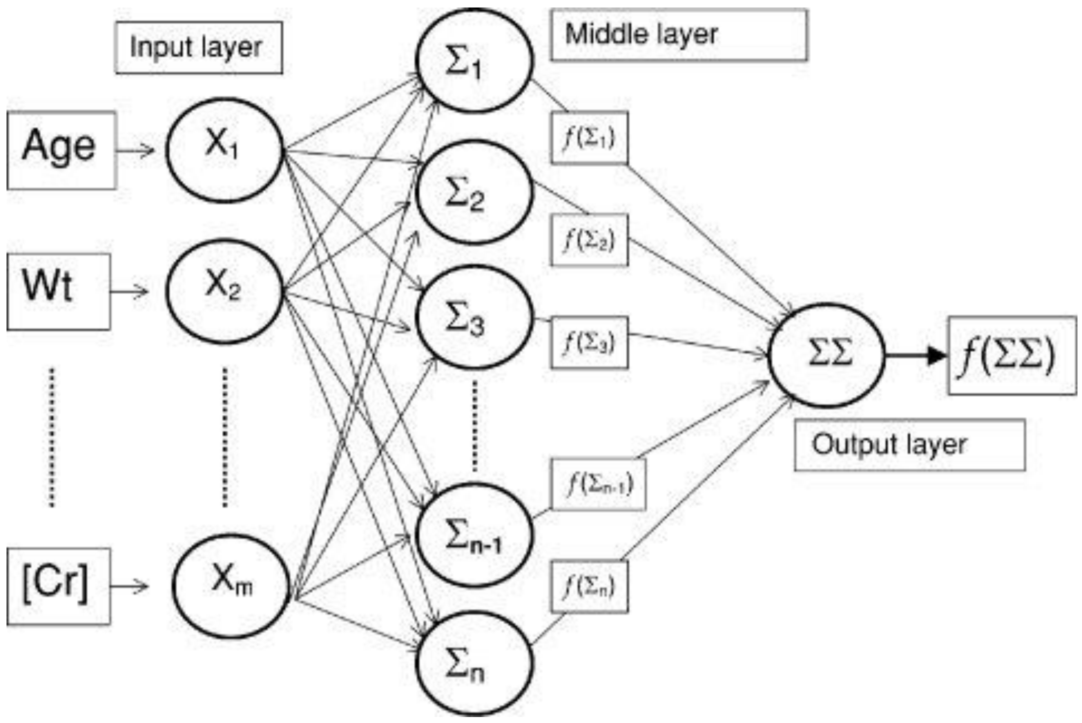


Figure 20: Simple form of ANN

2.2.2. Deep Learning and Deep Neural Networks

Deep learning is a subset of machine learning in artificial intelligence (AI) that is capable of unsupervised learning using multiple layers of artificial neural units on unlabeled data. Deep learning algorithms have shown success in a variety of tasks, including image and speech recognition, natural language processing, and predictive modeling (Bengio, 2015) These algorithms are able to extract complex features and patterns from data automatically through a neural network. When confronted with the need to learn a complex model, process thousands of values, or make future predictions in a time series, a simple form of artificial neural network (ANN) may become incapable. In such cases, a deep neural network (DNN) can be used to address these complex problems. Essentially an advanced form of ANN, a DNN is developed in its "deep" structure, consists of multiple hidden layers (ranging from two to hundreds) of artificial neural units. The additional layers and connections in a DNN enable it to process and learn from more complex data than a single-hidden-layer ANN. Research has shown that DNNs can outperform other machine learning methods in certain tasks, such as image classification (Krizhevsky et al., 2012)

However, the training of DNNs can be computationally intensive and requires large amounts of labeled data, as well as require good optimization of hyperparameters. (Bengio et al., 2007) Because of these challenges, the use of DNNs in various fields, including computer vision, natural language processing etc. are expected to grow in future.

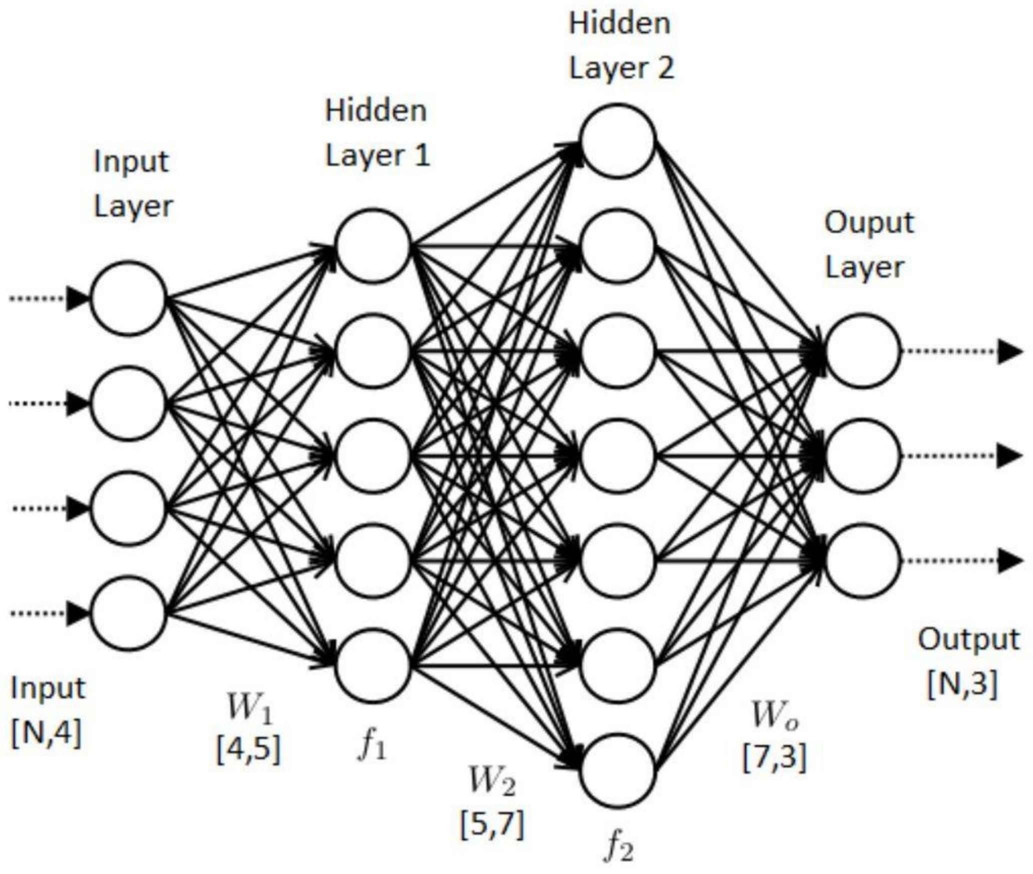


Figure 21: Deep Neuron Network

2.2.3. Convolutional Neural Networks

Convolutional neural networks (CNN) is a type of RNN in artificial neural network that are particularly well suited for image classification and recognition tasks. (LeCun et al., 1998) CNN occurs of multiple artificial neurons layers and it uses backpropagation algorithm. One key feature of CNN is its usage of convolutional layers, which apply a convolution operation to the input data. Convolutional layers extracts features of input data and it builds up a pattern which contains hierarchical representation of it. (Goodfellow et al., 2016) This method provide CNN to automatically learn and extract important features from the input data, hence, it does not requires to get manually label to input data. (Bengio et al., 2013)

Another feature of CNN need to be emphasized is that its use of pooling layers. İnput data turn into feature maps after it processed by convolutional layers. Then, this feature maps goes through pooling layers. These special layers gets high dimension of given input then it reduces dimensionality of the data. Thanks to this way, pooling layers provide increasing the robustness of the model and prevent from deformations in the input data (LeCun et al., 1998) CNNs have been successful in a wide range of image recognition and classification tasks. (Krizhevsky et al., 2012) CNN have also been applied to other domains such as natural language processing and speech recognition (Collobert et al., 2011) In this work, we will implement a CNN model to analyzing vulnerabilities on the smart contract code. We decide to choose a CNN model due to its success on NLP (Natural Language Processing) tasks. Details of our method will be explained in next chapters.

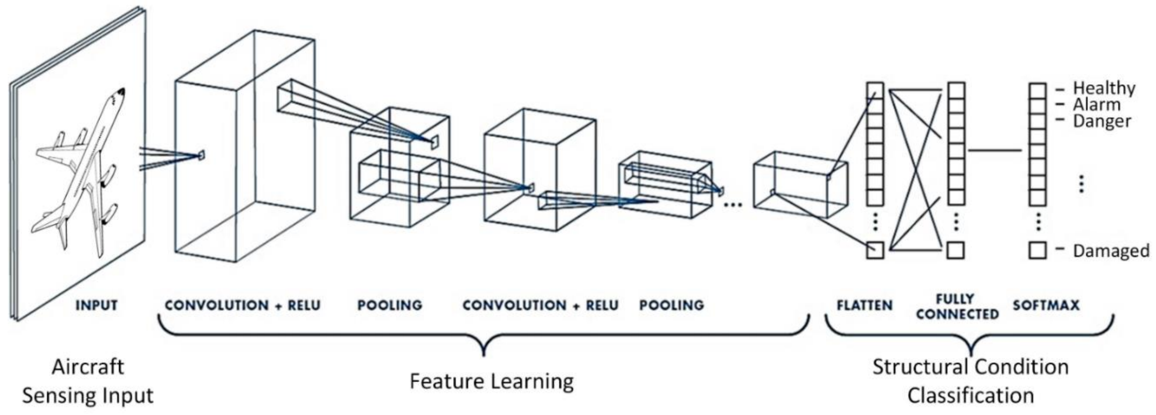


Figure 22: Representation of Convolutional Neural Network

2.2.4. Graph Neural Networks

Graph neural networks (GNN) are a type of neural network model designed to process on graph structured data (Bruna et al., 2014) GNN model is capable of processing of the complex dependencies between nodes and edges in a graph, and have been successful in a variety of tasks, including node classification and graph generation (Kipf and Welling, 2017) There are different types of GNN, including convolutional GNN, recurrent GNN, and attention-based GNN (Wu et al., 2019) Convolutional GNNs are inspired by convolutional neural networks (CNN) and use a "*localized neighborhood aggregation scheme*" to learn graph level representations. (Kipf and Welling, 2017) On the other hand, Recurrent GNNs are inspired by recurrent neural networks (RNN) and use a sequential data passing scheme to update the node representations. (Li et al., 2016) Attention-based GNNs use self-attention mechanisms to weight the importance of different nodes and edges in the graph. (Velickovic et al., 2018)One of the most difficulties in training GNN is the efficient computation of the graph convolutions. Since graph convolutions require the summation of the all features which belong neighbor nodes, its cost very expensive regarding with computation power. To solve this issue, researchers develop some methodologies, such as sampling-based and spectral methods. (Chen et al., 2018) GNN has achieved high accuracy results on a wide range of tasks. Hence, it has been used in medicine, social media analysis, fraud analysis, anomaly detection (Wang et al., 2019)

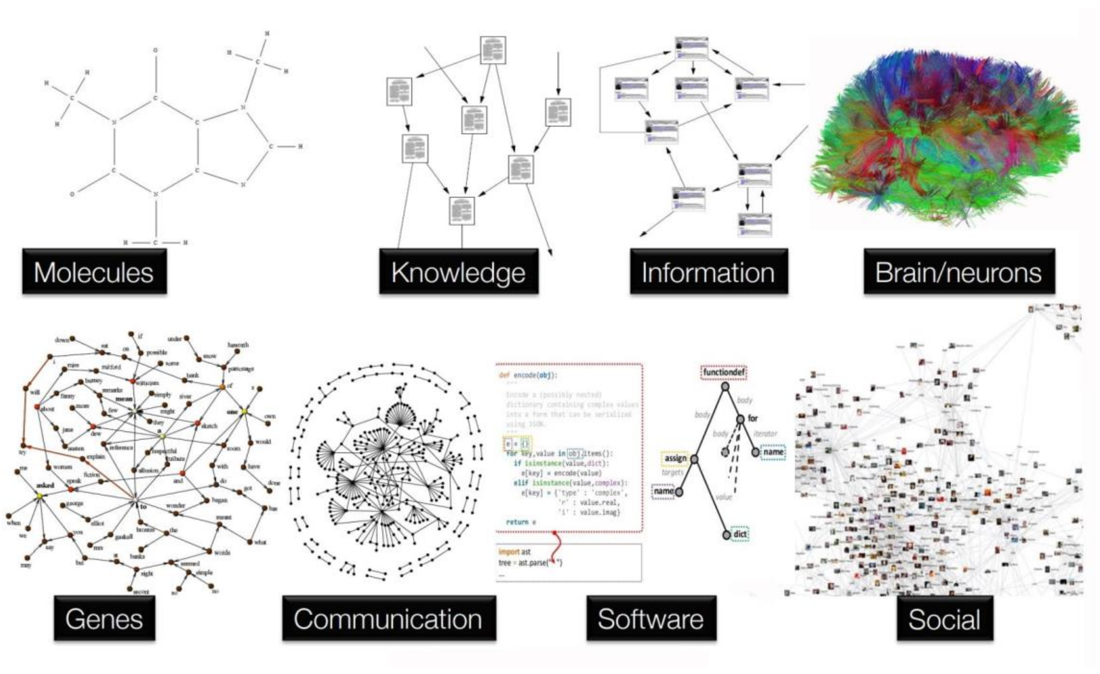


Figure 23 : Graph Neural Network Representation

CHAPTER 3

RELATED WORK

Both dishonest smart contract developers and external malicious actors try to find vulnerable code parts in Solidity to exploit them for their own benefit. To detect these bugs, many initiatives and independent researchers have developed a large amount of automatically executed tools. The methods these tools use for detecting vulnerabilities are different from each other. Some of them try to find predefined code snippets on smart contracts' source code (Static Vulnerability Tools) while others look into executing results and search for vulnerabilities based on scenarios (Dynamic Vulnerability Detection Tools). These two methods use traditional techniques to find vulnerabilities in smart contract codes. On the other hand, thanks to improving Machine Learning techniques, some tools that use Deep Learning and Artificial Neural Networks are beginning to be used for finding vulnerabilities in smart contracts. We will examine all three methods and their advantages and disadvantages over each other.

3.1. Static Vulnerability Detection Tools

Static analysis of software code is a well-known technique applied to software programs. It involves searching for bug-prone code snippets or vulnerable code structures within the code. There are many tools available for static analysis of smart contract code, such as Securify, Slither, Smartcheck, and Oyente.

Securify: It is a static analysis tool supported by the Ethereum Foundation. It checks the dependency graph of the code and extracts semantic code snippets from the smart contract's source code, searching for vulnerable patterns. It runs automatically and does not require expert skills to use. (Securify, 2022)

Slither: Slither is a vulnerability analysis framework that includes data flow and taint tracking to detect buggy code in smart contracts. It uses a framework called Static Single Assessment form to reduce high-level Solidity code to an instruction set, while preserving the semantic information of the code. (Feist et al., 2022) (Slither, 2022)

Oyente: Oyente differs from other static analysis tools in its approach. It uses symbolic execution to extract the control flows of the code, focusing on the execution paths of the software rather than the input data. This allows it to predict how the code will behave when it is run. Oyente can work with bytecode and can detect reentrancy, callstack depth, and integer overflow vulnerabilities. (Oyente, 2022) (Lutz, 2020)

Smart Check: Smart Check is another popular analysis tool developed for smart contract vulnerability analysis. It extracts the Abstract Syntax Tree (AST) of the contract code and analyzes the semantic features of the code, using XML to save these structures. It then checks for denial of service (DoS) vulnerabilities, reentrancy vulnerabilities, and timestamp dependencies in the code using XML Path Language (XPath). (SmartCheck, 2018) (Huang et al., 2022) (XPath, 2022)

Mythrill is one of the most advanced smart contract vulnerability scanning tools available. In addition to analyzing human-readable Solidity code or bytecode like other tools, it can also analyze the intermediate representation (IR) of the code. It can detect various vulnerabilities, including those related to gas consumption and the handling of external calls. Mythrill uses control flow graph execution engine which named as LASER by its producer firm. It executes Symbolic analyze on extracted control flow graph to find bugs. Since, it can find most of the DASP10 vulnerability types such as Integer Overflow/Underflow, Reentrancy Vulnerability, Unchecked Call Return, Denial of Service Vulnerability, Arithmetic Issues etc. its coverage rate is more than other static analyze tools (Jaggi, 2020) Mythrills' output is in JSON file extension. Hence, it is not practical for developers to validate its results. One who wants to check Mythrill results need to use JSON parser which needs extra effort as well as expert contribution requires for using it. (Jaggi, 2020) (Aryal, 2022)

3.2. Dynamic Vulnerability Detection Tools

Dynamic vulnerability detection tools differ from static ones in their working principle. While static tools scan software when it is not running, dynamic tools scan it during execution. As detecting every malicious code structure is a difficult task, static analysis cannot catch all vulnerabilities in the code. On the other hand, analysis tools that use dynamic methods can check the actual output of a code structure and determine if it is acting in a malicious way. However, dynamic analysis can be more expensive. (Cervantes et al., 2007)

Manticore: Manticore is an open-source analysis tool that can perform symbolic execution on smart contracts. It uses external solvers such as Yices, Z3, and CVC4 to symbolically trace code paths on the smart contract and the Manticore core engine can scan for bugs in these code snippets. (Cervantes et al., 2007) (Manticore, 2022) Manticore can analyze both smart contract EVM bytecodes and Linux binaries. (Lutz, 2020) It can detect reentrancy vulnerabilities, timestamp dependencies, external call to sender vulnerabilities, among others. However, its analysis can be time-consuming. (Aryal, 2022)

Maian: It aims to find greedy, prodigal, or suicidal contracts by scanning a set of smart contracts. More specifically, it looks for contracts that can be locked by malicious actors, contracts that provide a possibility for malicious actors to steal their funds, or contracts that can be terminated by malicious actors. (Maian, 2018) Maian uses two different methods for this: symbolic analysis and concrete validation to detect bugs. Concrete analysis involves executing a smart contract on a fork of the Ethereum blockchain, allowing Maian to trace a smart contract's behavior in its real environment. Maian can work with bytecode instead of Ethereum source code. (Ivicanikolicsg, 2022)

teEther: It is developed by researchers at Saarland University and focuses on codes that can transfer funds to another address. (Norta et al. 2023) It works at a low level (with bytecode). If the bytecode of a contract is not available, teEther can translate Solidity code into bytecode using its dissembly support. After extracting the bytecode, teEther uses an SMT solver to run the bytecode on a private blockchain created for testing smart contracts in a controlled and safe environment. One drawback of teEther is that it requires expert knowledge to be used, as it does not have a frontend. A developer or analyst must load the smart contract into the Python environment to analyze it with teEther's code. (Krupp & Rossow, 2018) (Nescio007, 2018)

MythX: It is a security analysis platform for Ethereum smart contracts developed by the team at ChainSecurity. It provides a suite of tools for analyzing and testing smart contracts for vulnerabilities and other security issues, including static analysis, dynamic analysis, symbolic execution, a debugger, and a testing framework. (MythX, 2022)

3.3. Vulnerability Detection Tools Using Deep Learning

Although both Static Vulnerability Detection Tools and Dynamic Detection Tools give accurate results mostly on detection vulnerable code parts and bugs on Smart Contracts, these tools rely on predefined models. In other words, they simply scanning Smart Contracts if they have any predefined malicious code structure or not. That work in most of the scenarios however, since these scanners are trying to find predefined codes they cannot catch similarity of a code snippet may have. A vulnerable code can hide from static and dynamic scanners since it has not identified onto scanner before. Power of Machine Learning has came to the fore on this point. Since Machine Learning algorithms are extracting model their self in regarding with labeled data, they can find hidden vulnerabilities in higher percentage of accuracy compare with other methods. (Huang et al., 2022) There are bunch of tools which are using different machine learning techniques out there, however we examined widely used and well-known ones.

SCSCAN: It was developed by Xiaohan Hao et al. It uses support vector machines (SVMs) to detect vulnerabilities, including reentrancy, DoS attacks, and access control vulnerabilities. The developers claim that SCSCAN has a success rate of over 90% for identifying these vulnerabilities. (Hao & Ren, 2020)

BGNN4VD: It was introduced by Sicong Gao et al. It uses bidirectional graph neural networks to detect a variety of vulnerabilities in Solidity code, including integer overflows/underflows, reentrancy, and callstack attacks. BGNN4VD achieved an average precision of 87.8% and an average recall of 81.5% on a dataset of vulnerable and non-vulnerable contracts. (Cao et al., 2021) (Li & Zou, 2018)

VulDeePecker: VulDeePecker introduced by Zhen Li et al., uses bidirectional long-short term memory to detect vulnerabilities. It has been evaluated on a dataset of real-world Ethereum smart contracts and has demonstrated good performance in terms of precision and recall. (Yu et al., 2021)

DeeSCVHunter: DeeSCVHunter developed by Yu X et al., uses a technique called vulnerability candidate slicing (VCS) to improve the accuracy of its deep learning model. This model can only detect reentrancy and timestamp vulnerabilities. [88]

CBGRU: CBGRU is a hybrid model that combines a word embedding model and a convolutional neural network (CNN). (Goswami et al., 2021)

BLTM-ATT: BLTM-ATT introduced by Qian P. et al., combines bidirectional long-short term memory with an attention mechanism to scan for vulnerable contracts.

TokenCheck: TokenCheck introduced by Goswami S et al., uses long short term memory for smart contract code analysis. (Zeng et al., 2022)

EtherGIS: EtherGIS, developed by Zeng et al., uses a graph neural network (GNN) to detect vulnerabilities. It extracts graph features from the control flow graph (CFG) and then applies GNN to these features after converting them into vectors. (Ashizawa & Yanai, 2021)

Eth2Vec, Eth2Vec developed by Ashizawa et al., uses natural language processing (NLP) to extract semantic features of smart contract code, which are then used in a graph neural network to detect vulnerable smart contracts. (Hacken.io, 2022)

CHAPTER 4

METHODOLOGY

Smart contracts have many vulnerabilities, which have resulted in significant economic losses and a loss of public confidence in the blockchain system as a whole. These vulnerabilities can occur due to inexperienced developers or malicious actors. There is a wide range of vulnerability types, and foundations such as DASP have attempted to detect and classify them. DASP has created a top 10 list of the most impactful vulnerabilities, and according to this list, reentrancy vulnerabilities are the most common and harmful type of smart contract vulnerability. They have directly caused losses of almost $200 million dollars. (Mohd. Ishrat et al., 2012) Given the significance of this issue, we have decided to develop an analysis tool that focuses on finding reentrancy vulnerabilities in smart contracts.

In chapter 4, we discussed the tools and methodologies used for detecting vulnerabilities in smart contracts. While static analysis tools are widely used by smart contract developers and the cyber security community, they can struggle to detect rare and malicious code patterns because they can only detect code snippets that exist in their database. Dynamic analysis tools also have limitations, as they can only find vulnerabilities when they execute the smart contract program. It is not always possible to know how the smart contract will behave in different execution scenarios. (Wang & Xu, 2020) Research has shown that the most promising results, with accuracy rates above 90%, can be obtained by using machine learning in vulnerability detection tools. In recent years, many such tools based on ML algorithms have been developed by researchers for software vulnerability scanning. However, some of these tools suffer from issues such as using the wrong methods, such as working with bytecodes that result in bytecode loss during the extraction phase, or not being well-configured, such as using incorrect or missing hyperparameters.

On the other hand, tools that use state-of-the-art deep learning models, such as graph neural networks, can be require a high level of knowledge in the field of machine learning. In addition, GNN models may not be suitable for balancing ease of use with good results, as they often produce results similar to those of other deep learning tools. Therefore, their acceptance among smart contract developers may be limited.

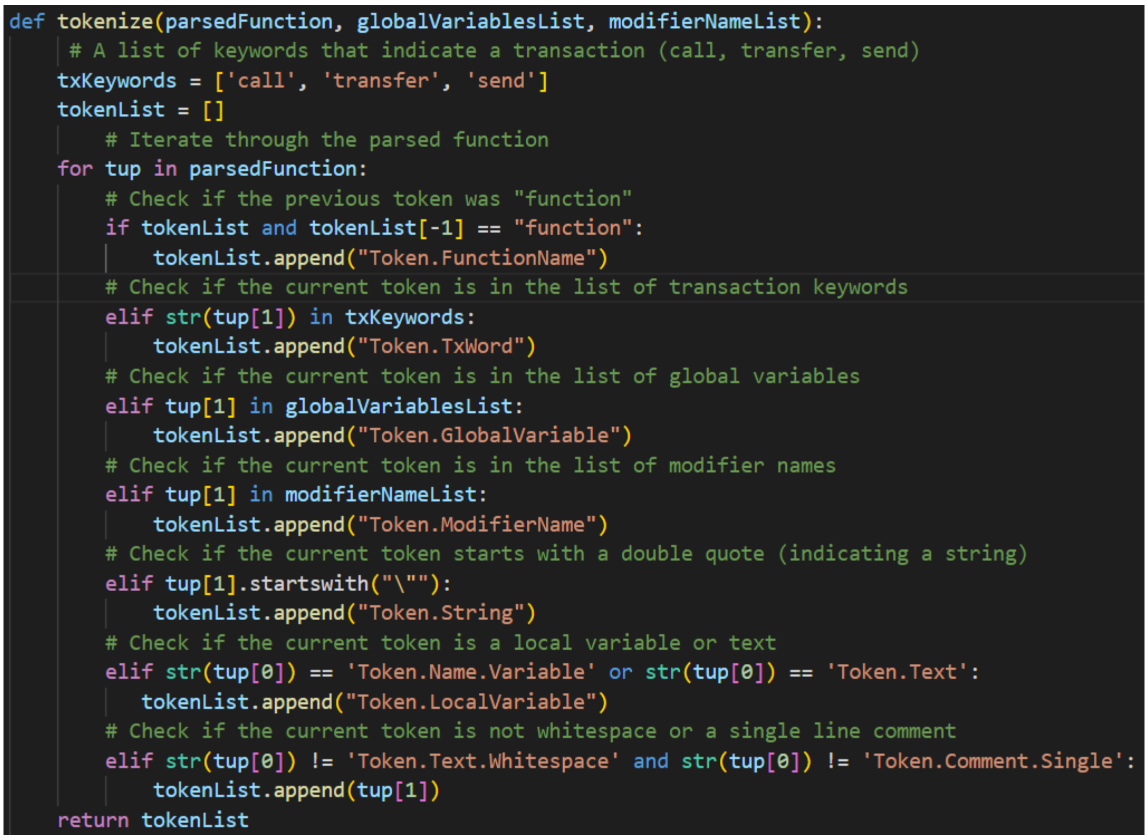
Considering features that are both compactness and produce good results, we introduce our Four Layer CNN (Convolutional Neural Network) Model for the task of detecting reentrancy vulnerabilities in smart contracts. Our model works on extracted code snippets provided by the Word2Vec model. It takes the extracted features as input and uses four layers to analyze these features according to their labels. Below, we will provide more detail and describe the phases of our work.

4.1. Data Acquisition

Deep learning models need to be trained with labeled data before they can be used. The data must be accurately labeled in order to use our model. There are works [95] (Liu et al., 2023) (Saastamoinen, 2020) that use training data labeled by other analysis tools, such as Oyente and Smartbugs. However, these approaches are flawed at their root. If these analysis tools were reliable 100% of the time, there would be no need to develop deep learning models for vulnerability analysis of smart contracts. Therefore, the labeling process needs to be done by hand or under human supervision. We used a labeled dataset created by another researcher (Zhuang et al., 2020) The original reentrancy dataset has 273 smart contract examples that are labeled as 0 or 1 based on their reentrancy vulnerability status. We picked 200 labeled smart contracts. While, 64 of these contracts which are labeled as “1”, have reentrancy vulnerability, 136 of them which are labeled as “0” are not vulnerable . We divided our dataset into 80% for training and 20% for testing as it is 160 number of contracts for training and 40 number of contracts for test.

4.2. Implementing of Word2Vec

Word2Vec is a popular word embedding model that uses machine learning to provide natural language processing. Its principle of operation involves representing words as vectors in space based on their meanings in sentences. "The idea is that these word embeddings contain information derived from the contexts of each target word, i.e., from the words frequently occurring near each target word, and are therefore more informative than the plain words by themselves." (Hao & Ren, 2020)



```python
def tokenize(parsedFunction, globalVariablesList, modifierNameList):
    # A list of keywords that indicate a transaction (call, transfer, send)
    txKeywords = ['call', 'transfer', 'send']
    tokenList = []
        # Iterate through the parsed function
    for tup in parsedFunction:
        # Check if the previous token was "function"
        if tokenList and tokenList[-1] == "function":
            tokenList.append("Token.FunctionName")
        # Check if the current token is in the list of transaction keywords
        elif str(tup[1]) in txKeywords:
            tokenList.append("Token.TxWord")
        # Check if the current token is in the list of global variables
        elif tup[1] in globalVariablesList:
            tokenList.append("Token.GlobalVariable")
        # Check if the current token is in the list of modifier names
        elif tup[1] in modifierNameList:
            tokenList.append("Token.ModifierName")
        # Check if the current token starts with a double quote (indicating a string)
        elif tup[1].startswith("\""):
            tokenList.append("Token.String")
        # Check if the current token is a local variable or text
        elif str(tup[0]) == 'Token.Name.Variable' or str(tup[0]) == 'Token.Text':
           tokenList.append("Token.LocalVariable")
        # Check if the current token is not whitespace or a single line comment
        elif str(tup[0]) != 'Token.Text.Whitespace' and str(tup[0]) != 'Token.Comment.Single':
            tokenList.append(tup[1])
    return tokenList
```

Figure 24: Implementing of word2vec model on python language

We use the Word2Vec model to extract code snippets from smart contract source codes and represent these snippets as vectors in a graph. This method allows us to build a model that takes into account the relationships between code parts, enabling us to identify vulnerable code parts using our CNN model.

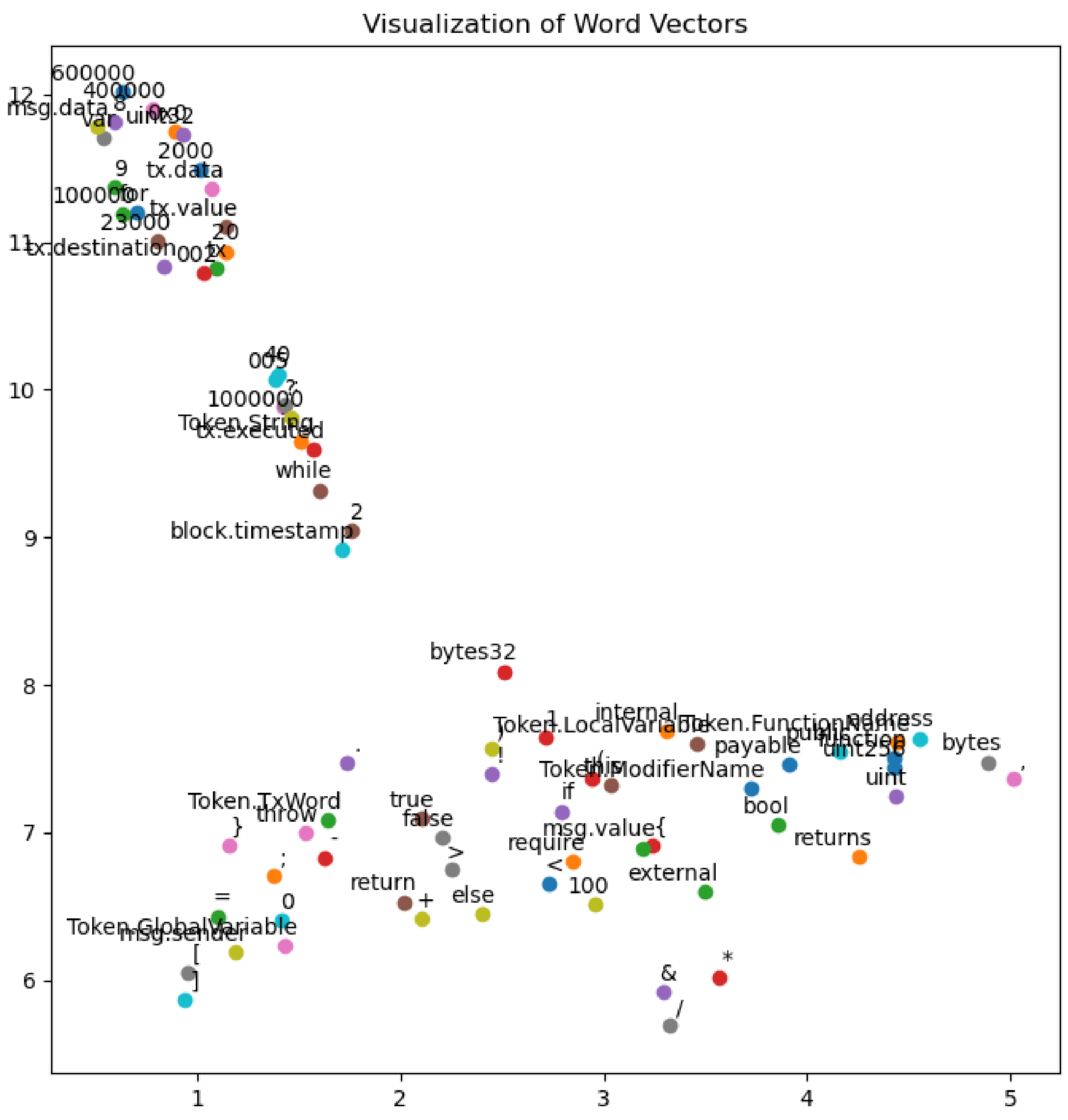
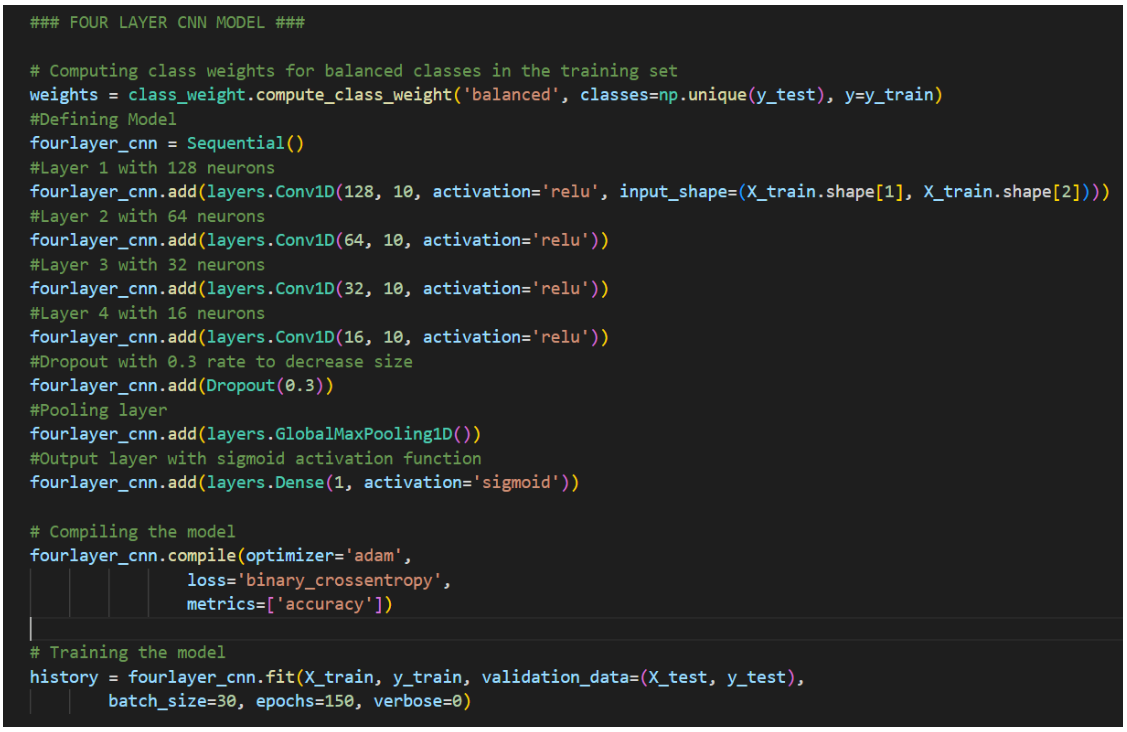


Figure 25: Code snippets' vector extracted by out Word2Vec Model

As mentioned above, the Word2Vec model represents code snippets based on their co-occurrence frequency. This method will provide semantic meaning for the code parts to our deep learning model in the subsequent stages of our work.

4.3. Creating CNN Model

We examined a large number of research to find the best model for our purpose. There are works that use RNN, LSTM, GRU, etc. We chose to work with a Convolutional Neural Network because research shows that it gives better results than LSTM (Hao & Ren, 2020) and is more practical to implement compared to GNN (Zhuang et al., 2020)



```python
### FOUR LAYER CNN MODEL ###

# Computing class weights for balanced classes in the training set
weights = class_weight.compute_class_weight('balanced', classes=np.unique(y_test), y=y_train)
#Defining Model
fourlayer_cnn = Sequential()
#Layer 1 with 128 neurons
fourlayer_cnn.add(layers.Conv1D(128, 10, activation='relu', input_shape=(X_train.shape[1], X_train.shape[2])))
#Layer 2 with 64 neurons
fourlayer_cnn.add(layers.Conv1D(64, 10, activation='relu'))
#Layer 3 with 32 neurons
fourlayer_cnn.add(layers.Conv1D(32, 10, activation='relu'))
#Layer 4 with 16 neurons
fourlayer_cnn.add(layers.Conv1D(16, 10, activation='relu'))
#Dropout with 0.3 rate to decrease size
fourlayer_cnn.add(Dropout(0.3))
#Pooling layer
fourlayer_cnn.add(layers.GlobalMaxPooling1D())
#Output layer with sigmoid activation function
fourlayer_cnn.add(layers.Dense(1, activation='sigmoid'))

# Compiling the model
fourlayer_cnn.compile(optimizer='adam',
                loss='binary_crossentropy',
                metrics=['accuracy'])

# Training the model
history = fourlayer_cnn.fit(X_train, y_train, validation_data=(X_test, y_test),
        batch_size=30, epochs=150, verbose=0)
```

Figure 26: CNN Model Code in Python

Our CNN model consists of four layers with 128, 64, 32, and 16 neurons, respectively. We use "ReLu" as the activation function for the hidden layers and sigmoid for the output (dense) layer. Additionally, we use a 0.3 dropout for each epoch. After experimenting with various values, we decided to use a batch size of 30 and determined 150 as epoch size to achieve the best results.

CHAPTER 5

EVALUATION

We trained and test our model as we mention above in detail and we got promising results which are better than many analyzing tools.

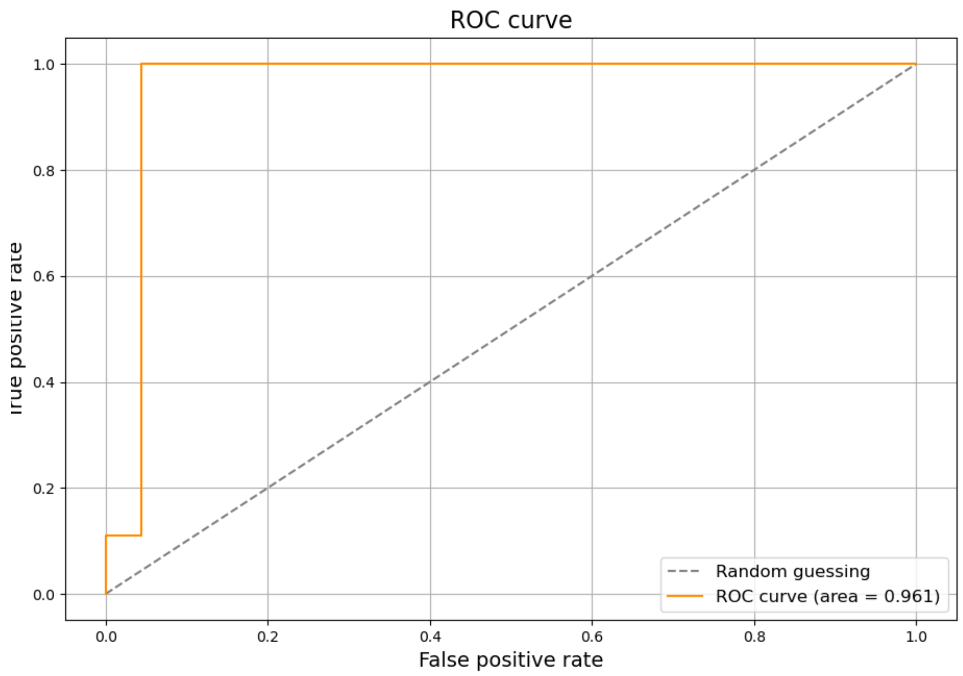
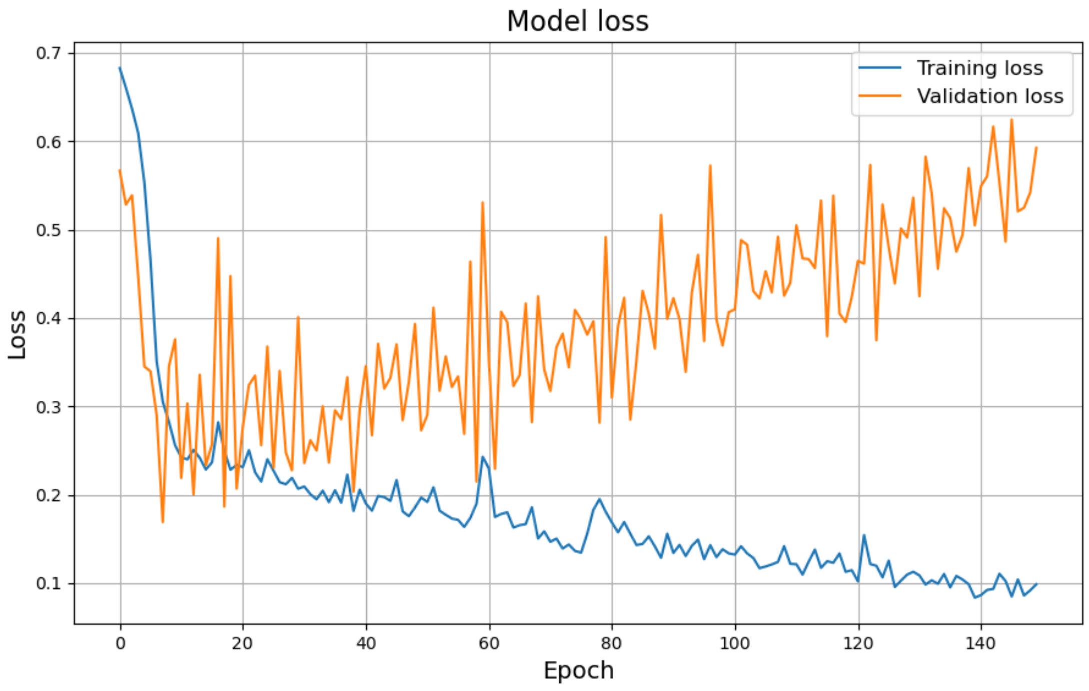


Figure 27: ROC Curve of Results



- *Figure 28: Model loss of Results*

	Precision	***Recall***	***F1 Score***	***Support***
0	1.00	0.93	0.97	30
1	0.83	1.00	0.91	10
Accuracy			0.95	40
Macro Avg	0.92	0.97	0.94	40
Weighted Avg	0.96	0.95	0.95	40
F1 Score				0.90
Recall Score				1.0
Precision Score				0.83
Accuracy Score				0.95

Table 1: Result Values of Our Model

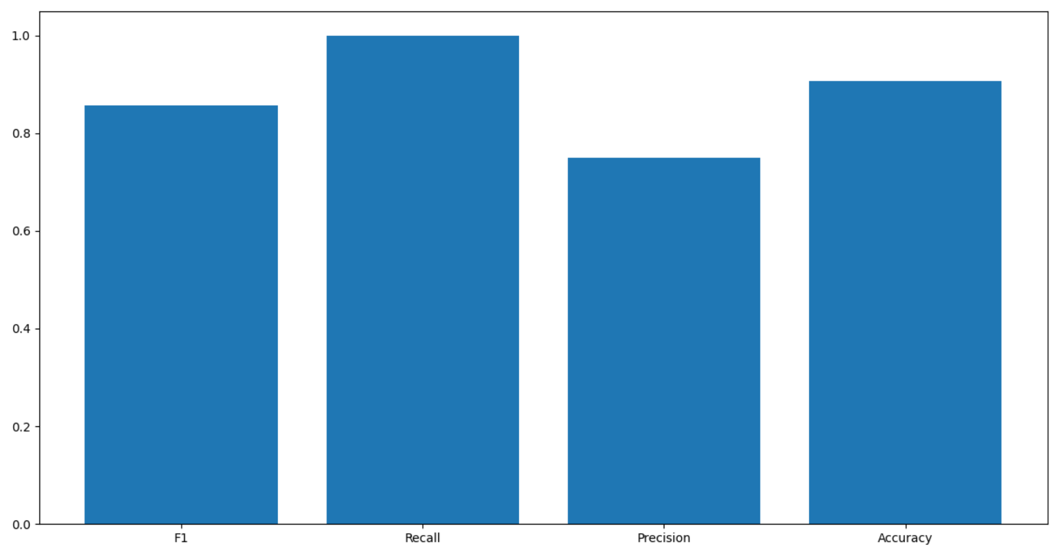


Figure 29: Demonstration of Result Values on Table 2

5.1. Comparison Results with Other Tools

We compared result of our model and some analyze tools that we select.

Tools	***Accuracy(%)***	***Recall(%)***	***Precision(%)***	***F1(%)***	***Acc(%)***
Smartcheck	52.9	32.08	25.00	28,00	44.32
Oyente	61.62	54.71	38.16	44.96	59.45
Mythrill	60.54	71.69	39.58	51.02	61.08
Securify	71.89	56.60	50.85	53.57	-
DR-GCN	81.47	80.89	72.36	76.39	-
Our Model	95	1.0	83.33	90.90	-

Table 2: Comparison with other Tools

CHAPTER 5

CONCLUSION

Since blockchain ecosystem is economically growing day by day, people who do not know much thing about blockchain and its working principle, put their big amount of money with hoping of making profit. However, both malicious smart contract developers and external attackers exploit smart contracts’ software by using Solidity vulnerabilities. Even though, there are some initiatives that audit bugs on smart contracts and warn people against smart contracts’ vulnerabilities, there is no widely used, accepted by public and at the same time effective and easy to use smart contract vulnerability solution.

In this context, our proposed method using CNN (Convolutional Neural Network) for smart contract vulnerability detection offers a promising solution. By leveraging the power of machine learning, CNNs can accurately and efficiently identify reentrancy vulnerability which is most seen and impactful one in Solidity contracts. Our results demonstrate the effectiveness of this approach, with an average precision and accuracy of 0.83% and 0.95%, respectively. Additionally, CNN-based approaches have the potential to scale and adapt to new types of vulnerabilities as they emerge, making them a valuable tool for ensuring the security and reliability of smart contracts in the long term.

Overall, our research demonstrates the potential of convolutional neural networks as a valuable tool for detecting vulnerabilities in smart contracts. While more research is needed to fully realize the potential of this approach, we believe that CNNs offer a promising solution for improving the security and reliability of smart contracts in the rapidly growing blockchain ecosystem.

REFERENCES

[1] Nakamoto, S. (2008). Bitcoin: A peer-to-peer electronic cash system. Retrieved from http://bitcoin.org/bitcoin.pdf

[2] Shen, L. (2018, January 8). Ethereum Regains Title as Second Most Valuable Cryptocurrency Behind Bitcoin. Fortune. Retrieved from https://www.fortune.com/2018/01/08/ethereum-second-most-valuable-cryptocurrency/

[3] Röscheisen, M., Baldonado, M., Chang, K., Gravano, L., Ketchpel, S., & Paepcke, A. (1998). The Stanford InfoBus and its service layers: Augmenting the internet with higher-level information management protocols. In Digital Libraries in Computer Science: The MeDoc Approach (pp. 173-187). Springer.

[4] Jiang, B., Liu, Y., & Chan, W. K. (2018). ContractFuzzer: Fuzzing Smart Contracts for Vulnerability Detection. arXiv preprint arXiv:1807.03932.

[5] Chernis, B., & Verma, R. (2018). Machine Learning Methods for Software Vulnerability Detection.

[6] Forbes Business Council. (2022, March 17). Smart Contracts and the Law: What You Need to Know. Forbes. Retrieved from https://www.forbes.com/sites/forbesbusinesscouncil/2022/03/17/smart-contracts-and-the-law-what-you-need-to-know/

[7] Alchemy. Reentrancy Attack in Solidity. Retrieved from https://www.alchemy.com/overviews/reentrancy-attack-solidity

[8] Alharby, M., & van Moorsel, A. (2017). Blockchain-based smart contracts: A systematic mapping study. arXiv preprint arXiv:1710.06372.

[9] Halas, M. (2019). A Survey of Vulnerabilities and Attacks on Smart Contracts. Master's Thesis, LUT University. Retrieved from https://lutpub.lut.fi/bitstream/handle/10024/160115/Milan_Halas_Masters_Thesis_2019LUTUniversity.pdf?sequence=1

[10] Hammami, M. (2017, March 3). Bitcoin and Blockchain mechanism. GOV-SUR.

[11] Zhang, R., Xue, R., & Liu, L. (2019). Security and Privacy on Blockchain. ACM Computing Surveys, 1(1), 1-19. https://doi.org/10.1145/3316481

[12] Investopedia. Proof-of-Stake (PoS). Retrieved from https://www.investopedia.com/terms/p/proof-of-stake-pos.asp

[13] Coindesk. Bitcoin Mining Difficulty: Everything You Need to Know. Retrieved from https://www.coindesk.com/learn/bitcoin-mining-difficulty-everything-you-need-to-know/

[14] Ethereum. Ethereum Whitepaper. Retrieved from https://ethereum.org/en/whitepaper/

[15] Ethereum. Ethereum Foundation. Retrieved from https://ethereum.org/en/foundation/

[16]Ethereum. Proof-of-Stake (PoS). Retrieved from https://ethereum.org/en/developers/docs/consensus-mechanisms/pos/

[17] Arkangelo, T. S. J. (2019). Research on Malicious Smart Contracts Detection in Blockchain. Retrieved from: https://www.academia.edu/46856731/Title_Research_on_Malicious_Smart_Contracts_Detection_in_Blockchain

[18] Alharby, M., & van Moorsel, A. (2017). Blockchain-based smart contracts: A systematic mapping study. arXiv preprint arXiv:1710.06372.

[19] Szabo, N. Smart Contracts. Retrieved from https://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/smart.contracts.html

[20] King of the Ether. Postmortem. Retrieved from https://www.kingoftheether.com/postmortem.html

[21] Margaritis, A. (2021). Decentralized applications: Development of a blockchain-based academic documents verification platform. University of Macedonia. Retrieved from https://dspace.lib.uom.gr/bitstream/2159/25923/1/MargaritisArgyriosMsc2021.pdf

[22] David, I. M., & Tallinn University of Technology. (2022). An Evaluation Framework for Smart Contract Vulnerability Detection Tools on the Ethereum Blockchain.

[23] Szabo, N. (1996). Smart Contracts: Building Blocks for Digital Markets. EXTROPY: The Journal of Transhumanist Thought, 18(2).

[24] Hsieh, Y. Y. (2018). The Rise of Decentralized Autonomous Organizations: Coordination and Growth within Cryptocurrencies. Electronic Thesis and Dissertation Repository. https://ir.lib.uwo.ca/etd/5393

[25] Coindesk. (2016, June 17). The DAO Attacked: Code Issue Leads to $60 Million Ether Theft. Retrieved from https://www.coindesk.com/markets/2016/06/17/the-dao-attacked-code-issue-leads-to-60-million-ether-theft/

[26] Currencyrate.today. How much is 3600000 ETH (Ethereums) in USD (US Dollars). Retrieved from https://eth.currencyrate.today/convert/amount-3600000-to-usd.html

[27] DASP. DASP. Retrieved from https://dasp.co/

[28] Cryptodevops Academy. DASP 10: The top 10 smart contract vulnerabilities in Solidity. Retrieved from https://medium.com/cryptodevopsacademy/dasp-10-the-top-10-smart-contract-vulnerabilities-in-solidity-3e4365634717

[29] Leander, B. (2022). Access Control Models to Secure Industry 4.0 Industrial Automation and Control Systems. Retrieved from http://www.divaportal.org/smash/get/diva2:1470155/FULLTEXT02.pdf

[30] Redfoxsec. Integer Overflow in Smart Contract. Retrieved from https://redfoxsec.com/blog/integer-overflow-in-smart-contract/

[31] Dingman, W., Cohen, A., Ferrara, N., Lynch, A., Jasinski, P., Black, P. E., Deng, L. (2021). Defects and Vulnerabilities in Smart Contracts, a Classification Using the NIST Bugs Framework. Security and Communication Networks, 2021. https://doi.org/10.1155/2021/5798033

[32]Wikipedia. (Denial-of-service attack. Retrieved from https://en.wikipedia.org/wiki/Denial-of-service_attack

[33] Raikwar, M., & Gligoroski, D. ,2022. DoS Attacks on Blockchain Ecosystem. Retrieved from https://www.researchgate.net/publication/360887571_DoS_Attacks_on_Blockchain_Ecosystem/link/6290632bc660ab61f8487ee4/download

[34] Samreen, N. F., & Alalfi, M. H. ,2018, VOLCANO: Detecting Vulnerabilities of Ethereum Smart Contracts Using Code Clone Analysis. Retrieved from https://arxiv.org/pdf/2203.00769.pdf

[35] Eskandari, S., Moosavi, S., & Clark, J. (2019). SoK: Transparent Dishonesty: Front-Running Attacks on Blockchain. Retrieved from https://users.encs.concordia.ca/~clark/papers/2019_wtsc_front.pdf

[36] Alharby, M., & van Moorsel, A. (2017). Blockchain-based smart contracts: A systematic mapping study. arXiv preprint arXiv:1710.06372.

[37] SecureWorld. What is Timestamp Dependence Vulnerability? Retrieved from https://www.getsecureworld.com/blog/what-is-timestamp-dependence-vulnerability/

[38] Chen, H., Pendleton, M., Njilla, L., & Xu, S. (2019). A Survey on Ethereum Systems Security: Vulnerabilities, Attacks and Defenses. Retrieved from https://arxiv.org/pdf/1908.04507.pdf

[39] Security and Communication Networks. (2021). Hindawi. https://doi.org/10.1155/2021/5798033

[40] Ethereum Book. Smart Contracts: Consensus. Retrieved from https://www.bookstack.cn/read/ethereumbook-en/spilt.10.c2a6b48ca6e1e33c.md

[41] https://www.bookstack.cn/read/ethereumbook-en/spilt.10.c2a6b48ca6e1e33c.md

[42] Mitchell, T. (1997). Machine learning. McGraw Hill. Retrieved from http://www.cs.cmu.edu/~tom/files/MachineLearningTomMitchell.pdf

[43] IBM. Supervised vs. unsupervised learning. Retrieved from https://www.ibm.com/cloud/blog/supervised-vs-unsupervised-learning

[44] Liu, Q., & Wu, Y. (2012). Supervised learning. ResearchGate. Retrieved from https://www.researchgate.net/publication/229031588_Supervised_Learning/link/551c22380cf2909047ba23f5/download

[45] Halit Apaydin, Hajar Feizi, Mohammad Taghi Sattari, Muslume Sevba Colak, Shahaboddin Shamshirband, Kwok-Wing Chau, Comparative Analysis of Recurrent Neural Network Architectures for Reservoir Inflow Forecasting, Water ,12, 1500, 2020 https://www.mdpi.com/2073-4441/12/5/1500

[46] Sidra Mehtab, Jaydip Sen, Abhishek Dutta, Stock Price Prediction Using Machine

[47] Laakso, S. (2022). Artificial neural networks and deep learning. Retrieved from https://www.theseus.fi/bitstream/handle/10024/779806/Laakso%20Seila.pdf?sequence=2

[48] Sathelly, B. (2018). An artificial neural network approach to predict liver failure likelihood. The University of Toledo. Retrieved from https://etd.ohiolink.edu/apexprod/rws_etd/send_file/send?accession=toledo1535112414430542&disposition=inline

[49] Buscema, M. (1998). Back propagation neural networks. ResearchGate. Retrieved from https://www.researchgate.net/publication/13731614_Back_Propagation_Neural_Networks/link/554dca5408ae93634ec5a619/download

[50] Bengio, Y. (2015). Deep learning. Nature. https://www.researchgate.net/publication/277411157_Deep_Learning/link/55e0cdf908ae2fac471ccf0f/download

[51] Krizhevsky, A., Sutskever, I., & Hinton, G. (2012). Improving neural networks by preventing co-adaptation of feature detectors. https://arxiv.org/abs/1207.0580

[52] Bengio, Y., LeCun, Y., & Hinton, G. (2007). Deep learning. https://www.researchgate.net/publication/277411157_Deep_Learning/link/55e0cdf908ae2fac471ccf0f/download

[53] LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. IEEE Transactions on Neural Networks, 11(11), 1529-1554. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=726791

[54] Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep learning. MIT Press.

[55] Bengio, Y., Courville, A., & Vincent, P. (2013). Representation learning: A review and new perspectives. IEEE Transactions on Pattern Analysis and Machine Intelligence, 35(8), 1798-1828.

[56] Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In Advances in Neural Information Processing Systems (pp. 1097-1105).

[57] Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., & Kuksa, P. (2011). Natural language processing with deep recurrent neural networks. In Proceedings of the International Conference on Machine Learning and Data Mining (pp. 169-176). Springer, Berlin, Heidelberg.

[58] Bruna, J., Micheli, A., Zaremba, W., & Szlam, A. (2014). Spectral networks and deep locally connected networks on graphs.

[59] Kipf, T., & Welling, M. (2017). Semi-supervised classification with graph convolutional networks. Retrieved from https://openreview.net/forum?id=SJU4ayYgl

[60] Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., & Yu, P. S. (2019). A comprehensive survey on graph neural networks. IEEE Transactions on Neural Networks and Learning Systems.

[61] Li, Y., Tarlow, D., Brockschmidt, M., & Zemel, R. (2016). Gated graph sequence neural networks. In Advances in Neural Information Processing Systems.

[62] Velickovic, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., & Bengio, Y. (2018). Graph attention networks. Retrieved from https://openreview.net/forum?id=rJXMpikCZ

[63] Chen, Y., Ma, T., & Barzilay, R. (2018). Fastgcn: Fast learning with graph convolutional networks via importance sampling.

[64] Wang, X., Li, Y., & Tao, D. (2019). Neural recommendation system based on graph attention network.

[65] securify. GitHub. https://github.com/eth-sri/securify

[66] Feist, J., Grieco, G., & Groce, A. , 2019 Slither: A static analysis framework for smart contracts. IEEE Explore. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8823898

[67] slither. GitHub. https://github.com/crytic/slither

[68] oyente. GitHub. https://github.com/melonproject/oyente

[69] Lutz, O. (2020). Detection of software vulnerabilities in smart contracts using deep learning. Department of Computer Science. 19 October 2020.

[70] SmartCheck. (2018). https://tool.smartdec.net/.

[71] Huang, J., Zhou, K., Xiong, A., & Li, D. (2022). Smart contract vulnerability detection model based on multi-task learning. Sensors, 22, 1829. https://doi.org/10.3390/s22051829

[72] XPath cover page. W3C. https://www.w3.org/TR/xpath/all/

[73] Jaggi, M. (2020). Exploring deep learning models for vulnerabilities detection in smart contracts. Machine Learning and Optimization Laboratory, EPFL. https://www.comp.nus.edu.sg/~hobor/Publications/2018/Maian.pdf

[74] Aryal, B. (2022). Comparison of Ethereum smart contract vulnerability detection tools. Cyber Security Master's Degree Programme in Information and Communication Technology, Department of Computing, Faculty of Technology, Master of Science in Technology Thesis. https://www.utupub.fi/bitstream/handle/10024/152760/Comparison%20of%20Ethereum%20Smart%20Contract%20Vulnerability%20Detection%20Tools.pdf?sequence=1&isAllowed=y

[75] mythril. GitHub. https://github.com/ConsenSys/mythril

[76] Cervantes, J., Li, X., Yu, W., & Li, K. (2007). Support vector machine classification for large data sets via minimum enclosing ball clustering. https://reader.elsevier.com/reader/sd/pii/S0925231207002962?token=0DAB3A5D72EBA3CB633D2385D3108B0C33FA3CEA6E55ADC8060159F853C49C31FAEB1B132BAE513B48C841965C60B843&originRegion=eu-west-1&originCreation=20221227190754

[77] Manticore. GitHub. https://github.com/trailofbits/manticore

[78] Maian, P. (2018). Finding the greedy, prodigal, and suicidal contracts at scale. Retrieved from https://www.comp.nus.edu.sg/~hobor/Publications/2018/Maian.pdf

[79] Ivicanikolicsg. MAIAN. Retrieved from https://github.com/ivicanikolicsg/MAIAN

[80] An Evaluation Framework for Smart Contract Vulnerability Detection Tools on the Ethereum Blockchain. (2023). Retrieved from Pp.54

[81] Krupp, J., & Rossow, C. (2018). teEther: Gnawing at Ethereum to automatically exploit smart contracts. In 27th USENIX Security Symposium (USENIX Security 18). USENIX Association. Retrieved from https://publications.cispa.saarland/2612/

[82] Nescio007. teether. Retrieved from https://github.com/nescio007/teether

[83] MythX.. Retrieved from https://mythx.io/

[84] Huang, J., Zhou, K., Xiong, A., & Li, D. (2022). Smart contract vulnerability detection model based on multi-task learning. Sensors, 22, https://doi.org/10.3390/s22051829

[85] Cao, S., Sun, X., Bo, L., Wei, Y., & Li, B. (2021). BGNN4VD: Constructing bidirectional graph neural-network for vulnerability detection. Information and Software Technology.

[86] Cao, S., Sun, X., Bo, L., Wei, Y., & Li, B. (2021). BGNN4VD: Constructing bidirectional graph neural-network for vulnerability detection. Information and Software Technology. Retrieved from https://reader.elsevier.com/reader/sd/pii/S0950584921000586?token=69D38D0A66B8B28BB02817A7712B422B0362041B7D4C9EB323BEFA8862DB38F35FBD0CF660FAD563F66F24F6A2A2216D&originRegion=eu-west-1&originCreation=20221228211014

[87] Li, Z., & Zou, D. (2018). VulDeePecker: A deep learning-based system for vulnerability detection. arXiv:1801.01681 [cs].

[88] Yu, X., Zhao, H., Hou, B., Ying, Z., & Wu, B. (2021). DeeSCVHunter: A deep learning-based framework for smart contract vulnerability detection. In International Joint Conference on Neural Networks (IJCNN). Shenzhen, China.

[89] Zhang, L., Chen, W., Wang, W., Jin, Z., Zhao, C., Cai, Z., & Chen, H. (2022). CBGRU: A detection method of smart contract vulnerability based on a hybrid model. Sensors, 22, 3577. https://doi.org/10.3390/s22093577

[90] Goswami, S., Singh, R., Saikia, N., Bora, K. K., & Sharma, U. (2021). TokenCheck: Towards deep learning based security vulnerability detection in ERC-20 tokens. In IEEE Region 10 Symposium (TENSYMP). Jeju, Korea.

[91] Zeng, Q., He, J., Zhao, G., & Yanai, N. (2022). EtherGIS: A vulnerability detection framework for Ethereum smart contracts based on graph learning features. IEEE Transactions on Services Computing. https://doi.org/10.1109/TSC.2022.9842713

[92] Ashizawa, N., & Yanai, N. (2021). Eth2Vec: Learning contract-wide code representations for vulnerability detection on Ethereum smart contracts. arXiv:2101.02377 [cs].

[93] Hacken.io. Reentrancy attacks. Retrieved from https://hacken.io/discover/reentrancy-attacks/

[94] Mohd. Ishrat, M., Saxena, M., & Mohd. Alamgir, D. (2012). Comparison of static and dynamic analysis for runtime monitoring. International Journal of Computer Science & Communication Networks, 2(5), 615-617. https://www.academia.edu/35952183/Comparison_of_Static_and_Dynamic_Analysis_for_Runtime_Monitoring

[95] Wang, W., & Xu, G. (2020). ContractWard: Automated vulnerability detection models for Ethereum smart contracts. IEEE Transactions on Network Science and Engineering, 7(1), 94-108. https://doi.org/10.1109/TNSE.2020.2968505

[96] Zhang, L., Chen, W., Wang, W., Jin, Z., Zhao, C., Cai, Z., & Chen, H. (2022). CBGRU: A detection method of smart contract vulnerability based on a hybrid model. Sensors, 22(9), 3577. https://doi.org/10.3390/s22093577

[97] Liu, Z., Qian, P., Yang, J., Liu, L., Xu, X., He, Q., & Zhang, X. (2023).Rethinking Smart Contract Fuzzing: Fuzzing With Invocation Ordering and Important Branch Revisiting. arXiv preprint arXiv:2301.03943.

[98] Saastamoinen, T. (2020). Word2vec and its application to examining the changes in word contexts over time. Master's thesis, University of Helsinki, Faculty of Social Sciences. https://helda.helsinki.fi/bitstream/handle/10138/323724/Saastamoinen_Taneli_m_soc_sc_2020.pdf?sequence=2&isAllowed=y

[99] Zhuang, Y., Liu, Z., Qian, P., Liu, Q., Wang, X., & He, Q. (2020). Smart contract vulnerability detection using graph neural networks. In Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence (IJCAI-20).

[100] Hao, X., & Ren, W. (2020). SCscan: A SVM-based Scanning System for Vulnerabilities in Blockchain Smart Contracts. 2020 IEEE 19th International

Conference on Trust, Security and Privacy in Computing and Communications (TrustCom). https://doi.org/10.1109/TrustCom49850.2020.9343119

References of Figures

Figure 1: ResearchGate The structure of a Blockchain: A block is composed of a header and a body, where a header contains metadata and a body contains transactions Retrieved from https://www.researchgate.net/figure/The-structure-of-a-Blockchain-A-block-is-composed-of-a-header-and-a-body-where-a-header_fig1_337306138/download

Figure 2: Delmolino, K., Arnett, M., Kosba, A., Miller, A., & Shi, E. (2016). Step by step towards creating a safe smart contract: Lessons and insights from a cryptocurrency lab. In International Conference on Financial Cryptography and Data Security (pp. 79-94). Springer.

Figure 3: Zhang, P., Schmidt, D. C., & White, J. (2020). A pattern sequence for designing blockchain-based healthcare information technology systems. Security and Communication Networks, 2021, 5798033. https://doi.org/10.1155/2021/5798033

Figure 4: dasp.co Retrieved from https://dasp.co/

Figure 5: Hindawi (2021). Security and Communication Networks, 2021, Article ID 5798033, 12 pages, https://doi.org/10.1155/2021/5798033

Figure 6: David, I. M., & Tallinn University of Technology. (2022). An Evaluation Framework for Smart Contract Vulnerability Detection Tools on the Ethereum Blockchain.

Figure 7,8,9,16,17: Eskandari, S., Moosavi, S., & Clark, J. (2019). SoK: Transparent Dishonesty: Front-Running Attacks on Blockchain. Retrieved from https://users.encs.concordia.ca/~clark/papers/2019_wtsc_front.pdf

Figure 10: dasp.co Retrieved from https://dasp.co/

Figure 11: Alharby, M., & van Moorsel, A. (2017). Blockchain-based smart contracts: A systematic mapping study. arXiv preprint arXiv:1710.06372.

Figure 12: GetSecureWorld What is timestamp dependence vulnerability? [Blog post]. Retrieved from https://www.getsecureworld.com/blog/what-is-timestamp-dependence-vulnerability/

Figure 13: Hindawi (2021). Security and Communication Networks, 2021, Article ID 5798033, page 4 https://doi.org/10.1155/2021/5798033

Figure14:bookstack.cn,Retrieved from https://www.bookstack.cn/read/ethereumbook-en/spilt.10.c2a6b48ca6e1e33c.md

Figure15: bookstack.cn Retrieved from https://www.bookstack.cn/read/ethereumbook-en/spilt.10.c2a6b48ca6e1e33c.md

Figure 18: Varol, Temel & Canakci, Aykut & Ozsahin, Sukru. (2014). Prediction of the influence of processing parameters on synthesis of Al2024-B4C composite powders in a planetary mill using an artificial neural network. Sci. Eng. Compos. Mater.. 21. 411-420. 10.1515/secm-2013-0148. https://www.researchgate.net/figure/Artificial-neural-cell-artificial-neuron_fig3_264881135.

Figure 19: Figure caption: "Artificial Neuron," WikiDocs, https://wikidocs.net/165313.

Figure 20: Tangri, Navdeep & Ansell, David & Naimark, David. (2008). Predicting technique survival in peritoneal dialysis patients: Comparing artificial neural networks and logistic regression. Nephrology, dialysis, transplantation : official publication of the European Dialysis and Transplant Association - European Renal Association. 23. 2972-81. 10.1093/ndt/gfn187. https://www.researchgate.net/publication/5411405_Predicting_technique_survival_in_peritoneal_dialysis_patients_Comparing_artificial_neural_networks_and_logistic_regression/citation/download

Figure 21: ResearchGate The structure of a Deep Neural Network with three hidden layers ,Retrieved from https://www.researchgate.net/figure/The-structure-of-a-Deep-Neural-Network-with-three-hidden-layers_fig2_352996743

Figure 22: Tabian I, Fu H, Sharif Khodaei Z. A Convolutional Neural Network for Impact Detection and Characterization of Complex Composite Structures. Sensors. 2019; 19(22):4933. https://doi.org/10.3390/s19224933

Figure 23: NVIDIA (2022, October 24). What are graph neural networks? [Blog post]. Retrieved from https://blogs.nvidia.com

Figure 30: https://quantstamp.com/blog/what-is-a-re-entrancy-attack