I. J. Mathematical Sciences and Computing, **2022, 3, 30-36**
Published Online August 2022 in MECS (http://www.mecs-press.org/)
DOI: 10.5815/ijmsc.2022.03.03

Smart Contract Obfuscation Technique to Enhance Code Security and Prevent Code Reusability

Kakelli Anil Kumar
Associate Professor, School of Computer Science and Engineering (SCOPE), Vellore Institute of Technology, Vellore, India, 632014
E-mail: anilsekumar@gmail.com

Aena Verma
School of Computer Science and Engineering (SCOPE), Vellore Institute of Technology, Vellore, India, 632014
E-mail: vermaaena@gmail.com

Hritish Kumar
School of Computer Science and Engineering (SCOPE), Vellore Institute of Technology, Vellore, India, 632014
E-mail: hritish42@gmail.com,

Received: 28 December 2021; Accepted: 25 February 2022; Published: 08 August 2022

Abstract: Along with the advancements in blockchain technology, many blockchain-based successful projects have been done mainly on the ethereum platform, most of which deal with transactions. Still, it also carries various risks when it comes to security, as evident from past attacks. Most big projects like uniswap, decentraland, and others use smart contracts, deployed on the ethereum platform, leading to similar projects via code reuse. Code reuse practice is quite frequent as a survey suggests 26% of contract code deployed is via code reuse. Smart contract code obfuscation techniques can be used on solidity code that is publicly verified, published (in the case of Ethereum), and on the deployment address. All the above techniques work by replacing characters with their random counterpart, known as statistical substitution. A statistical substitution is a process of transforming an input string into a new string where each character has been replaced by a random character drawn from a stock of all possible 'random' characters. Therefore, we proposed numerous methods in this paper to solve the above problems using various smart contract code obfuscation techniques. These techniques can be really useful in blockchain projects and can save millions of dollars to investors & companies by enhancing code security and preventing code reusability. Techniques mentioned in this paper when compared with other techniques. Our methods are not expensive to implement, very easy to use, and provide a developer-friendly selective increment in code complexity.

Index Terms: Smart contract, Code Reuse, Security, and Obfuscation.

1. Introduction

Any product that has ever been created took much effort. There is a fixed lifecycle for every product that has ever been created in this world. For example, there is just the idea, motivation, and then people get funding and investment in the early development phase. After that, slowly build the project, which takes time, money, motivation, dedication, and risk. Then, there is always a chance of failure, and if the product is successful and has launched in public. Then people copy the idea and code. Then within months or weeks, there are similar products in the market. In contrast, it takes years of work, resources, and billions of dollars to develop, maintain a particular product, test it, and secure it from reverse engineering and attacks. Similarly, in blockchain, the projects are mostly deployed on mainnet networks, which are nothing but an independent blockchain running its network with a particular technology and protocol. After the deployment of smart contracts, people can verify and publish smart contract codes. Then it will be publicly available on the blockchain network. Without formal legal contracts, copyright ownership, or rights, companies and developers reuse the publicly available smart contracts code in their projects. This leads to piracy, violates several laws, and disrespects the original project owner [1].

Uniswap is one of the biggest decentralized cryptocurrency exchange platforms that work on uni swap protocols [2]. With an exchange market of over $7.6 billion and rising every day. It works on the ethereum platform and has around 1600 crypto tokens listed, making it one of the most popular token exchange platforms. Hayden Adams created it on November 2, 2018, who came up with a new idea [7]. Uniswap code is open-source, and anyone can read the code and understand how uniswap works, the logic behind the code, which can lead to security attacks, and copyright or code reuse. Another decentralized exchange platform launched on September 20, 2020, is PancakeSwap, which is just a copy of the uniswap code. The only difference is that it runs on the Binance smart chain. PancakeSwap's exchange market is over $6.5 billion, which is a considerable amount compared to uniswap as it took long enough to make their exchange market this successful. Pancakeswap uses the Binance chain, which charges low gas fees on transactions, attracting most users [6]. Actually, In a decentralized exchange like Sushiswap and others swap, almost all simulator clones of uniswap Source code. It even disrespects intellectual property.

Table 1. The Topmost reused smart contracts

Sub-contracts Types	Reused
ERC-20	9000+
ERC-20 Basic	5500+
ERC-20 Basic	4600+
Ownable	4600+
Token	4000+

Below are the techniques that are currently in use to prevent source code reusability and provide code security but, these techniques are unreliable, expensive to implement, and difficult to maintain.

1. Source code protection policy
2. Use Encryption and monitoring
3. Endpoint security tools
4. Network security solution
5. Copyrights and Patents
6. Access controls policies

As we can see most code thieves would not care by following a protection policy, patents, and copyrights. Thus, we have created a better solution to solve the above issues.

Here obfuscation techniques can help to prevent code-reuse by obfuscating the open-source code. It will turn it into a format by reordering the code or encoding some necessary address that is more difficult to read, understand, and analyze. Still, no one can use it directly in their projects. Ultimately developers and companies will keep doing such things and will never stop because it fastens their work and uses very few resources. They are gaining high profits on low investment by code reusability. People are getting well developed, tested, and secured products directly due to this. With only a few changes, it is ready to get deployed in the market. Nevertheless, this is not the only issue with publicly available code. There are security issues too. Publicly available codes are usually user-friendly and easily understandable, allowing security researchers, hackers, and bad guys to analyze, detect, and exploit security flaws in a contract that'll lead to business and trust loss [7, 8].

Recently in August 2021, there was a heist of 610 million USD. A hacker exploited a vulnerability in smart contracts of PolyNetworks that were available publicly. People usually make their code publicly so that the community can contribute, and the development of the application would take less time and resources to develop. Nevertheless, everything comes with a price [9, 10]. Open-source code is like showing our house blueprint to a thief for burglarizing. This is not the only case; There are many cases in which the attackers have exploited open-source applications that lead to such massive breaches. With the upcoming crypto demand, if all the critical codes were publicly available, it would be very dangerous if some hacker found another critical vulnerability and stole cryptocurrency like in the above case. This would ultimately result in people losing trust in crypto and technology [11,12].

Table-2 showcases the top 5 vulnerabilities that can be easily found in the code that is open source. If the code is not easily understandable, it will be hard to detect such flaws and exploit them. The above vulnerabilities are critical and if exploited can produce a devastating impact on the end-users. Therefore, if there is good code obfuscation, it will be difficult to read and understand the code, thus increasing security, diversity and, ultimately, preventing the above problem [13, 14].

Table 2. Emerging Vulnerabilities of Blockchain, Software, and SmartContract

Blockchain Vulnerabilities	**Software Vulnerabilities**	**SmartContract Vulnerabilities**
1. Complexity 2. Immutability 3. Sequential Execution 4. Human Errors 5. Transparency	1. Buffer Overflows 2. Race Conditions 3. Improper Input Validations 4. Hardcoded credentials & APIs 5. Command Injection & Privilege Escalation	1. Block timestamp dependency 2. Call stack depth limitation 3. Re-entrancy problem 4. Greedy contracts 5. Transaction ordering

2. Proposed Solution

A blockchain explorer mainly uses API and nodes to fetch data from the blockchain and show it in software in a presentable and user-friendly way like etherscan. That is used to query about the blocks. In blockchain explorer, we can search any smart contract that has been deployed using its address or transaction hash, and it will display all the details related to all the transactions that had been made until now. We can even read verified and published smart contract codes that are publicly available. The obfuscator program can be applied to verified and published code directly after submitting the code to the server for verification and before showing the code to the user. The whole process of obfuscation will be done in the blockchain explorer's backend. So, for implementation, some of the obfuscation techniques are used in smart contracts.

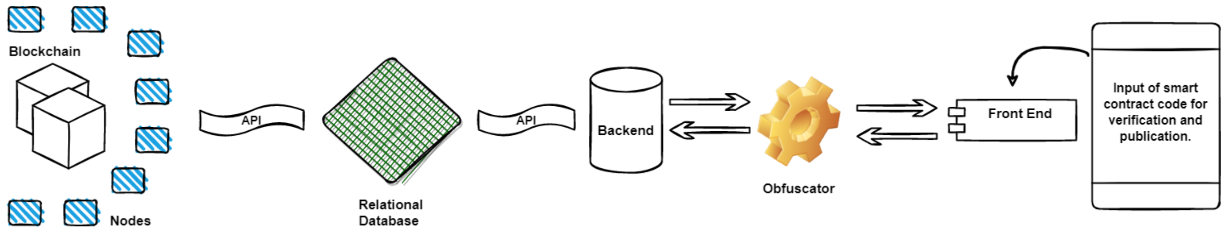

Fig .1. The working model of Blockchain explorer

2.1. Reformat technique

This technique usually renames the functions, methods, classes, variables, and object names to something random and unpredictable. Also, in the process, we remove the comments and indent/whitespaces of the code. We can change integer type variable values to a formula that gives fixed output after passing a hardcoded value. It'll behave like a small function. Also, the variables can be merged into a single variable and can be mathematically calculated when in use. This will help us to confuse the attackers. We applied the Reformat technique in the sample code, as shown below. Similarly, all the methods can be used to smart contract codes.

2.2. Data Structure transformation technique

In this technique, we alter the way of storing data in memory. There are many ways it can be done like to change the way data is placed in the program [4] by shuffling variables from local to global storage, we can encrypt all the addresses in smart contract or encrypt variable function related to the transaction, to hide critical information that can be vulnerable or flaws by masking it by changing its name or identity. For example, let us take arrays. We can split a big array into multiple parts to confuse the reader about its design. We can also merge multiple arrays and combine a single array. With that, to increase the obfuscating factor, even more, we can increase the dimensions or flatten it, shift the matrix elements left side or right side as we do in cryptographic algorithms, reverse the elements and insert decoy data inside the arrays [5].

2.3. Decoy data insertion.

In this technique, we will insert a dummy working code into the working code. That will act as a decoy and puzzle the attacker or the reader. The code inserted will not affect the working or the code logic. This can also work like an Opaque predicate in which an expression results in either true(1) or false(0). The output will be known to the programmer but needs to happen only at runtime, confusing the attacker with reading dummy code. Also, we can start by splitting classes and functions then inserting multiple bogus classes and functions into split functions or classes to make it look crowded and sophisticated. We could add identifiers in the code to make the compiler skip these classes and functions. As shown above, probabilistic control flow or opaque predicates will maintain the overall code complexity while increasing the obfuscation factor.

Table 3. The comparison between original and obfuscated smart contract code

Original	**Obfuscated**

contract deal{
address public seller;
address payable public buyer;
address payable public buyer;
uint public pay_amount;
constructor(address _seller, address payable _buyer, uint _amount)
Public{
seller=_seller;
buyer= _buyer;
lawyer =msg.sender;
pay_amount= _amount;
}
//to deposit money
function deposit() payable public {
require(msg.sender == payer) ;
require(address(this). balance <= amount);}
}
..
.
	contract arm99765422{
	address public c9891eb76c;
	address payable public buyer;
	address payable public buyer;
	uint public amy5876tot;
	constructor(address i7654311t,address
	payable b6152e176a, uint p87ga3139)
	public{
	payer =i7654311t;
	payee =b6152e176a;
	09091eb76c =msg.sender;
	amy5876tot = p87ga3139;
	}
	
	function 0676548x11p() payable public {
	require(msg.sender==payer) ;
	require(address(this). balance
	<= amy5876tot);}
	}
	..
	.

Table 4. The Data Structure transformation technique

Original	**Obfuscated**

int main()
{
int arr1 = {1,2,3};
int arr2 = {4,5,6};
int arr3 = {7,8,9};
..
.
}
	int main()
	{
	
	int arr1 = {2,3,$,4,5,6,$,7,8,9,$,1};
	..
	.
	.
	}

2.4. String manipulation and Encryption technique

In this technique, we can do cryptographic operations on strings like converting the characters to ASCII and [3] then doing xor with some values and then reversing it or doing encryption. Also, we can split the variables and use them, when necessary, at runtime; this would ultimately hide the strings and only show the strings when the string is called. For example, we can convert the static strings to procedural strings to evade reverse engineering software and debuggers. This is like creating a small program or function with a program that generates strings, like H(1), H(2), H(3), H(4). Could mean H(1) = “ABC,” H(2) = “BBB,” H(3) = “BCF”.. Adding all of this to a single function would not make it stealthy; it will be easy to crack the code and break the obfuscation algorithm. Therefore, the function should be broken up into smaller parts and split across the program [8].

Table 5. The Decoy data insertion

Original	**Obfuscated**

contract mapEXP {
mapping (address => uint) public balance;
function upated_bal(uint new_bal) {
balance[msg.sender] = new_bal;
}
}
 	
contract mapuser {
function f() returns (uint) {
mapEXP m= new mapEXP();
m.upated_bal(100);
return m. balance (this);
}
…..
.
	contract mapEXP {
	mapping (address => uint) public balance;
	mapping (address => uint) public balances;
	
	function upated_bal(uint new_bal) {
	balance[msg.sender] = new_bal;
	}
	
	function upted_bal(uint new_bal) {
	balances[msg.sender] = new_bal;
	}
	}
	
	contract mapuser {
	function f() returns (uint) {

.
.
 	
}
	mapEXP m= new mapEXP();
	m.upated_bal(100);
	return m. balance (this);
	}
	
	function f1() returns (unit){
	mapping (address => uint) public balances;
	}
	….
	.
	
	}

Table 6. The String manipulation and Encryption techniques

Original	Obfuscated

pragma solidity ^0.4.24;
contract String {
string store = "mark_account";
 	
function getStore() public view returns (string) {
return store;
}
 	
function setStore(string_value) public {
store = _value;
}
….
.
}
	pragma solidity ^0.4.24;
	contract String {
	string store = "9-3147-5-3-1-111171016";
	
	function getStore() public view returns (string) {
	return store;
	}
	
	function setStore(string_value) public {
	store = _value;
	}
	….
	.
	.
	}
	Encryption logic(turn string to ASCII-100)

2.5. Reordering of code technique.

This technique can reorder various functions, methods, and other code snippets without changing the program's logic. We can also split a single class and functions into multiple classes and functions or vice versa.

Table 7. The reordering of code technique

Original	Obfuscated

contract deal{
address public seller;
address payable public buyer;
address public dealer;
uint public pay_amount;
 	
constructor(address _seller, address payable _buyer,uint _pay_amount)public{
 	
seller = _seller;
buyer = _ buyer;
dealer = msg.sender;
pay_amount = _pay_amount;
 	
}
 	
function deposit() payable public {
require(msg.sender seller) ;
require(address(this). balance <= pay_amount);
}
 	
function release() public {
require(address(this). balance pay_amount);
require(msg.sender dealer);
buyer.transfer (pay_amount);
}
 	
function balance() view public returns (uint){
return address (this).balance;
	contract deal{
	
	address public dealer;
	uint public pay_amount;
	address public seller;
	address payable public buyer;
	
	constructor(uint _pay_amount, address payable _buyer,address _seller)public{
	dealer = msg.sender;
	pay_amount = _pay_amount;
	seller = _seller;
	buyer = _ buyer;
	}
	
	modifier checkdealer{
	require(msg.sender == dealer);
	_;
	}
	modifier checkseller{
	require(msg.sender == seller);
	_;
	}
	
	function balance() view public returns (uint){
	return address (this).balance;
	}
	
	function release() public checkdealer{

}
….
.
.
 	
}
	require(address(this). balance pay_amount);
	buyer.transfer (pay_amount);
	}
	
	function deposit() payable public checkseller{
	require(address(this). balance <= pay_amount);
	}
	…..
	.
	
	}

All the above techniques could be combined as a single program for a particular blockchain that can generate perfect obfuscated code for smart contracts and be used in the blockchain explorer. After implementing the above techniques, there can be a drawback, like if the obfuscated code gets broken or cracked and if then people can read it via automated tools, it will be a problem, and the security gets compromised. So, to avoid it, the obfuscation program should always be in improvement mode and getting new updates.

3. Conclusion

This paper has highlighted why obfuscation is necessary and should be implemented in public smart contracts. Due to its flexibility, ease of use, low cost, improves security with a great factor, and keeps the attackers away from easily exploiting the smart contracts, obfuscation should be implemented in blockchain explorers. Also, in this paper, we have presented various case studies on code reuse and blockchain security that moreover proposed multiple techniques for obfuscating and implementing smart contracts. We have concluded five ways that are best for obfuscating code. We believe that our research work would make some contribution to blockchain security and prevent code reusability, which would make blockchain more secure, reliable and users would trust it & adopt the technology quicker for all applications.

References

[1] Chen X, Liao P, Zhang Y, Huang Y, Zheng Z. Understanding Code Reuse in Smart Contracts. In2021 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER) 2021 Mar 9 (pp. 470-479). IEEE.
[2] Sebastian SA, Malgaonkar S, Shah P, Kapoor M, Parekhji T. A study & review on code obfuscation. In2016 World Conference on Futuristic Trends in Research and Innovation for Social Welfare (Startup Conclave) 2016 (pp. 1-6). IEEE.
[3] Behera CK, Bhaskari DL. Different obfuscation techniques for code protection. Procedia Computer Science. 2015 Jan 1;70:757-63.
[4] Zhang M, Zhang P, Luo X, Xiao F. Source Code Obfuscation for Smart Contracts. In2020 27th Asia-Pacific Software Engineering Conference (APSEC) 2020 Dec 1 (pp. 513-514). IEEE.
[5] Karnick, M., MacBride, J., McGinnis, S., Tang, Y., Ramachandran, R. (2006). A Qualitative analysis of Java Obfuscation, Proceedings of 10th IASTED International Conference on Software Engineering and Applications, Dallas TX, USA, November 13-15, 2006.
[6] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil Vadhan, and Ke Yang. "On the (im) possibility of obfuscating programs." In J. Kilian, editor, Advances in Cryptology:CRYPTO 2001, 2001. LNCS 2139.
[7] Aigner AA, Dhaliwal G. UNISWAP: Impermanent Loss and Risk Profile of a Liquidity Provider. arXiv preprint arXiv:2106.14404. 2021 Jun 28.
[8] Collberg C, Thomborson C, Low D. Breaking abstractions and unstructuring data structures. In Proceedings of the 1998 International Conference on Computer Languages (Cat. No. 98CB36225) 1998 May 16 (pp. 28-38). IEEE.
[9] Praitheeshan P, Pan L, Yu J, Liu J, Doss R. Security analysis methods on ethereum smart contract vulnerabilities: a survey. arXiv preprint arXiv:1908.08605. 2019 Aug 22.
[10] Liu Z, Qian P, Wang X, Zhuang Y, Qiu L, Wang X. Combining graph neural networks with expert knowledge for smart contract vulnerability detection. IEEE Transactions on Knowledge and Data Engineering. 2021 Jul 7.
[11] Wang Z, Jin H, Dai W, Choo KK, Zou D. Ethereum smart contract security research: survey and future research opportunities. Frontiers of Computer Science. 2021 Apr;15(2):1-8.
[12] Rameder H. Systematic Review of Ethereum Smart Contract Security Vulnerabilities, Analysis Methods and Tools (Doctoral dissertation, Wien).
[13] Huang Y, Bian Y, Li R, Zhao JL, Shi P. Smart contract security: A software lifecycle perspective. IEEE Access. 2019 Oct 11;7:150184-202.
[14] Guida L, Daniel F. Supporting reuse of smart contracts through service orientation and assisted development. In2019 IEEE International Conference on Decentralized Applications and Infrastructures (DAPPCON) 2019 Apr 4 (pp. 59-68). IEEE.

Authors' Profiles

Dr. Kakelli Anil Kumar is an Associate Professor of the School of Computer Science and Engineering at the Vellore Institute of Technology (VIT), Vellore, TN, India. He earned his Ph.D. in Computer Science and Engineering from Jawaharlal Nehru Technological University (JNTUH) Hyderabad in 2017, and graduated in 2009 and under-graduated in 2003 from the same university. He started his teaching career in 2004 and worked as an Assistant Professor, Associate Professor, and HOD in various reputed institutions of India. His current research includes wireless sensor networks, the internet of things (IoT), cybersecurity and digital forensics, Malware analysis, block-chain, and crypto-currency. He has published over 50 research articles in reputed peer-reviewed international journals and conferences.

Aena Verma is from Gujrat, India. She is a B.Tech. undergraduate in Computer Science from the Department of Computer Science and Engineering (CSE), Vellore Institute of Technology. Her main interests lie primarily in Blockchain, NTF's and Information Security. She is currently working as a software engineer at Walmart. She previously worked on many blockchain projects like Polkadot, ethereum, NFT's, and Cryptocurrency mainly on the development side. In her leisure time, she likes to listen to music and do cooking.

Hritish Kumar is from Gurgaon, India. He is a Security Researcher, Content Developer, Infosec Consultant, and a B.Tech. undergraduate in Computer Science from the Department of Computer Science and Engineering (CSE), Vellore Institute of Technology. His research interests lie primarily in the area of Information Security and Blockchain Security. He currently serves as an offensive security engineer and a content developer. In his leisure time, he likes to listen to music and play video games.

How to cite this paper: Kakelli Anil Kumar, Aena Verma, Hritish Kumar, "Smart Contract Obfuscation Technique to Enhance Code Security and Prevent Code Reusability", International Journal of Mathematical Sciences and Computing(IJMSC), Vol.8, No.3, pp. 30-36, 2022. DOI:10.5815/ijmsc.2022.03.03