

SWC-based Smart Contract Development Guide Research

Jong-Hoon Lee, SeongHo Yoon, Hyuk Lee

School of Cybersecurity, Korea University, Seoul 02841

ehm94@korea.ac.kr, shyoon17@korea.ac.kr, fmlab_hlee@korea.ac.kr

Abstract— Smart contracts are consensus protocols in which contracts are automatically concluded by code. Despite the growing popularity of smart contracts recently, no guidelines have been presented for developers to refer to when developing smart contracts. Therefore, using SWC, which classifies the weaknesses of smart contracts, guidelines that developers can use in the development process from the perspective of security, maintenance, and reuse are presented based on the weaknesses. The presented indicators can be used to reduce costs incurred later during development.

Keywords— Smart Contracts, Software Engineering, Secure Software, Block Chain, Solidity

I. INTRODUCTION

Since the launch of Bitcoin in 2009, decentralized cryptocurrency has gained considerable interest and adoption [1]. Cryptocurrency is managed by automated agreement between network users, not by a central agency.[2] Blockchain technology has been rapidly developing beyond cryptocurrency in recent years, and smart contracts are currently in the spotlight among the technologies that can be used as blockchain.

The term smart contract is generally an agreement protocol in which contracts are automatically concluded by code, and was first introduced by Nick szabo in 1997 for automation contracts[3]. It was not used due to lack of reliability at the time of introduction, but with the recent advent of blockchain technology, reliability has been guaranteed, increasing interest in smart contracts[4]. Blockchain has advantages such as transparency, invariance, security, and safety based on decentralized distributed ledger technology[5]. Currently, Ethereum supports smart contracts and is the most used blockchain with more than 1300 million smart contracts built, and about 1 million transactions are made every day[6].

Despite the growing popularity of smart contracts, no guidelines have been presented for developers to refer to when developing smart contracts. Currently, smart contracts are developed using the language of solidity [7]. Solidarity is also changing rapidly as the popularity of smart contracts increases, which can cost a lot in terms of maintenance and reuse.[8]. Therefore, this paper presents guidelines that minimize cost aspects when developing smart contracts and map them to security perspectives, maintenance perspectives, and reuse

perspectives based on SWC to allow developers to consider cost aspects when developing smart contracts.

II. BACKGROUND

A. Smart Contract

Smart contracts are not paper contracts, but protocols for digital implementation such as electronic documents and electronic contracts made with electronic signatures[9]. The first smart contract concept was proposed by Nick Szabo in 1996, and it was an innovative concept that changed from analog to digital[3]. However, before blockchain technology came out, there was no transaction intermediary, which could lead to forgery and alteration, so trust in the contract could not be guaranteed, so it was not used properly[4]. After that, blockchain technology emerged and reliability problems were solved. Currently, the Ethereum network creates a smart contract by developing a language called solidity[7]. In order to deploy a smart contract, it can only be used by paying a fee called GAS[10]. You have to pay a lot of fees to distribute smart contracts. The reason is that other nodes accessing the network must proceed with verification to verify smart contracts.

Solidity is a contract-oriented programming language for developing smart contracts[7]. It is used to develop smart contracts on various blockchain platforms[11]. Solidity is a static form of programming language for developing smart contracts running on Ethereum Virtual Machine (EVM), compiled into[12], Bytecode, and executed on EVM. Solidity can define hierarchical mapping and structure as member variables and supports multiple inheritance. It provides an Application Binary Interface (ABI)[13] function to support multiple types of secure access to functions of the developed contract. Developers can develop smart contracts that contain their own business logic using solidity, and after the completion of the contract, they keep unrejectable and reliable transactions between the parties.

B. SWC

In software, it is a widely accepted practice to classify bugs and weaknesses into standardized systems such as Common Weakness Enumeration (CWE) and common vulnerabilities exposures (CVE) [14][17]. Although this does not prevent new vulnerabilities in software from appearing, it has helped educate and recognize the nature of software's security-related

problems in various fields[16]. Also, can quickly understand weakness to perform root cause analysis or classify the results of various security analysis sources. Currently, Smart Contract Weakness Classification (SWC) exists as a weakness classification system related to Ethereum smart contracts, such as Common Weakness Enumeration (CWE)[14][15].

CWE does not explain the weaknesses related to smart contracts, but explains the weaknesses related to higher abstraction layers. Using this, SWC was built to suit smart contracts based on CWE[15]. SWC consists of a total of 37 weaknesses, identifiers, weakness titles, related CWE items, and test case codes[15]. When developing a smart contract, you can proceed with secure coding using SWC.

III. SMART CONTRACT DEVELOPMENT GUIDELINE BASED ON SWC

A. Motivation and Approach

Smart contracts are discovering security issues where new syntax is evolving very rapidly and newly discovered[18]. Recently, there are studies on the life cycle of smart contract development such as DevOps and SCSVS for blockchain smart contracts, but it is difficult to proceed with development only with the contents. And since maintenance costs the most, maintenance and reuse aspects are important to minimize costs, as shown as in Fig 1.

Therefore, SWC is used to create specific indicators. The SWC has 37 items, each consisting of an identifier, a weak point name, a related CWE, and an example. Based on the

relevant CWE, each item in the SWC was analyzed in terms of confidentiality, integrity, and availability, and further mapped, including maintenance and reuse aspects, considering the cost issues required for future development.

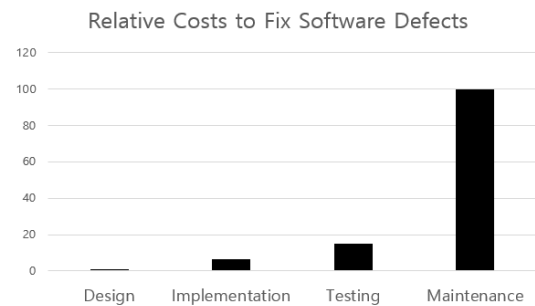


Fig 1. Relative Costs to Fix Software Defects

B. Results

Indicators such as Table 1 were derived focusing on security, maintenance, and reusability so that they can be applied when developing rapidly evolving smart contracts. Based on this, developers can reduce correction costs related to security weaknesses and possible maintenance costs in the future by calculating priorities using the table when developing smart contracts.

TABLE 1. SWC WITH CONFIDENTIALITY, INTEGRITY, AVAILABILITY, MAINTAINABILITY AND RESUABILITY

SWC-ID	Title	Confidentiality	Integrity	Availability	Maintainability	Reusability
SWC-100	Function Default Visibility	✓	✓		✓	
SWC-101	Integer Overflow and Underflow	✓	✓	✓		
SWC-102	Outdated Compiler Version		✓	✓	✓	✓
SWC-103	Floating Pragma			✓	✓	✓
SWC-104	Unchecked Call Return Value		✓	✓		
SWC-105	Unprotected Ether Withdrawal	✓	✓			
SWC-106	Unprotected SELFDESTRUCT Instruction	✓	✓			
SWC-107	Reentrancy		✓			
SWC-108	State Variable Default Visibility	✓			✓	
SWC-109	Uninitialized Storage Pointer	✓	✓	✓	✓	
SWC-110	Assert Violation		✓			
SWC-111	Use of Deprecated Solidity Functions	✓	✓	✓	✓	✓

SWC-112	Delegatecall to Untrusted Callee	✓	✓	✓		
SWC-113	DoS with Failed Call	✓	✓	✓		
SWC-114	Transaction Order Dependence	✓	✓	✓	✓	✓
SWC-115	Authorization through tx.origin	✓	✓	✓		✓
SWC-116	Block values as a proxy for time	✓	✓	✓		
SWC-117	Signature Malleability	✓	✓	✓		
SWC-118	Incorrect Constructor Name	✓	✓			
SWC-119	Shadowing State Variables	✓		✓	✓	
SWC-120	Weak Sources of Randomness from Chain Attributes	✓				
SWC-121	Missing Protection against Signature Replay Attacks	✓	✓			
SWC-122	Lack of Proper Signature Verification		✓			
SWC-123	Requirement Violation			✓	✓	
SWC-124	Write to Arbitrary Storage Location	✓	✓	✓	✓	
SWC-125	Incorrect Inheritance Order		✓		✓	
SWC-126	Insufficient Gas Griefing			✓		
SWC-127	Arbitrary Jump with Function Type Variable		✓	✓	✓	
SWC-128	DoS With Block Gas Limit			✓		
SWC-129	Typographical Error		✓		✓	✓
SWC-130	Right-To-Left-Override control character (U+202E)			✓		
SWC-131	Presence of unused variables		✓		✓	✓
SWC-132	Unexpected Ether balance			✓		
SWC-133	Hash Collisions With Multiple Variable Length Arguments	✓	✓	✓	✓	
SWC-134	Message call with hardcoded gas amount	✓		✓	✓	✓
SWC-135	Code With No Effects		✓		✓	✓
SWC-136	Unencrypted Private Data On-Chain		✓			

IV. CONCLUSIONS

Smart contracts, which refer to overall automation of contracts, are widely used as blockchain technology has recently developed. However, according to the results of the survey, there is a lack of research on how to develop smart contracts compared to the rapid development of smart contracts. In this study, indicators that developers can use

during development were presented using SWC, which classified the weaknesses of smart contracts.

In this paper, based on the weakness provided by SWC, weakness that may occur in smart contracts are classified into security perspectives, maintenance perspectives, and reuse perspectives. In terms of security, it was largely divided into confidentiality, integrity, and availability, which are the main elements of security. The items in the classification and SWC were mapped and provided as a table. Through the provided table, guidelines for calculating development priorities in the

direction of minimizing development costs in consideration of security, maintenance, and reuse aspects when developing.

ACKNOWLEDGMENT

This work was supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government(MSIT) (No.2021-0-00177, High Assurance of Smart Contract for Secure Software Development Life Cycle)

REFERENCES

- [1] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," Working Paper, pp. 1–9, 2008.
- [2] Cryptocurrency, <https://www.investopedia.com/terms/c/cryptocurrency.asp>
- [3] Smart Contracts: neither contracts nor smart ,<https://lawahead.ie.edu/smart-contracts-neither-contracts-nor-smart/>
- [4] Smart Contract History and Law, <https://jqrgen.medium.com/smart-contract-history-and-law-9b2f3331d1a>
- [5] What Is a Blockchain?, <https://www.investopedia.com/terms/b/blockchain.asp>
- [6] Etherscan, <https://etherscan.io/>
- [7] Allison Ian, "PwC blockchain expert pinpoints sources of ambiguity in smart contracts", IBTimes, 12 August 2016.
- [8] <https://github.com/ethereum/solidity/blob/develop/Changelog.md>
- [9] Smart Contracts, <https://www.investopedia.com/terms/s/smart-contracts.asp>
- [10] WHAT IS GAS?, <https://ethereum.org/en/developers/docs/gas/>
- [11] Alyssa Hertig, "Blockchain Veterans Unveil Secure Smart Contracts Framework", CoinDesk, 15 September 2016.
- [12] Mougayar, William, The Business Blockchain: Promise, Practice, and Application of the Next Internet Technology, Wiley Publishing, 2016
- [13] ABI(Application Binary Interface), https://en.wikipedia.org/wiki/Application_binary_interface, retrieved 19 February 2018.
- [14] CWE, <https://cwe.mitre.org/>
- [15] SWC, <https://swcregistry.io/>
- [16] ITU-T Rec. X.1524 (03/2012) Common weakness enumeration
- [17] CVE, <https://cve.mitre.org/>
- [18] Design Patterns for Ethereum Smart Contracts



Hyuk LEE received the B.S. degree from the University of Technology Sydney, Sydney, Australia, in 2006, and M.S. degree in 2009 and the Ph.D. degree in 2019 from the Korea University, Seoul, Korea. He is currently a research professor with the Graduate School of Information Security, Korea University, Seoul, Korea. His current research interests are in formal methods, constraint problem solving, and secure software engineering.



Jong-Hoon Lee received B.S. degree from the School of Computer Engineering, Dongguk University, Korea in 2020. He is currently an M.S. candidate in the School of Cybersecurity, Korea University. His current research interests are in secure coding and secure software engineering



Seong Ho Yoon received B.S. degree from the School of Computer Engineering, Dongguk University, Korea in 2020. He is currently an M.S. candidate in the School of Cybersecurity, Korea University. His current research interests are in Network Security and secure software engineering.