Characterizing the cost of introducing secure programming patterns and practices in Ethereum

N'Da, Aboua Ange Kevin; Matalonga, Santiago; Dahal, Keshav

Published in:
WorldCist'20 - 8th World Conference on Information Systems and Technologies

DOI:
10.1007/978-3-030-45691-7_3

Published: 08/06/2020

Document Version
Peer reviewed version

Characterizing the cost of introducing secure programming patterns and practices in Ethereum

Aboua Kevin N'Da, Santiago Matalonga, Keshav Dahal

University of the West of Scotland

{abouaangekevin.n'da, santiago.matalonga, keshav.dahal}@uws.ac.uk

Abstract. Ethereum is blockchain-based platform which enables the development and deployment of smart contracts. Smart contracts are computer programs that provide automation for the governance of decentralized autonomous organizations (DAO). However, while the Blockchain technology is secure, smart contracts are only as secure as the programmers has designed it to be. Therefore, smart contract exposes vulnerabilities that can be exploited by attackers and threaten the viability of the DAOs. This study presents a case study which investigated how security programming patterns and practices from other programming languages can be applied in Solidity – Ethereum programming language. We have characterized the cost of introducing these patterns and practices. We identified 30 security programming patterns and practices from C++, JAVA which can be applicable to Solidity and implemented ten in a representative smart contract. The results show that the application of the ten security patterns and practices identified and implemented increases the cost of the smart contract (when compared to the baseline). Furthermore, we argue that this difference is not significant and should not deter any programmers into introducing the security patterns and practices into their smart contracts.

Keywords: Secure programming, Blockchain, Ethereum, Smart contract.

1. Introduction

Ethereum is a decentralized platform based on Blockchain technology that enables the implementation and deployments of software programs called smart contract. Ethereum provides environment to write and run smart contract. Solidity is the programming language of choice for Ethereum smart contracts, and the environment provides an Ethereum Virtual Machine (EVM) where these smart contracts can run. Yet, as any software program, smart contract in Solidity, can expose vulnerabilities. These vulnerabilities can be exploited by attackers to take control of the assets being governed by the smart contract. Smart contract vulnerabilities can therefore lead to the loss of the user asset such financial asset. For instance, The DAO smart contract, that once held almost a third of Ethereum assets, was hacked and led the loss of USD 50 million [1]. Therefore, we argue actions to mitigate vulnerabilities in smart contract must be taken. Our approach differs from most of the literature reviewed. While most research focus

on vulnerability detection [2][3], this research focus on developing a sound theory for the development of software systems on top of Blockchain platforms.

In this paper we present a case study that explores if security patterns and practices from other programming languages (mainly C++ and Java) can be applied to Solidity. Furthermore, we perform a characterization of the transaction cost (roughly the cost of deploying and executing methods of a smart contract in Ethereum) of implementing these security patterns and practices. As a result, we identified 30 security programming patterns and practices from C++, JAVA which can be applied to Solidity. Our running example, enable us to implement ten of these patterns and practices, and deploy them in Ethereum. Regarding transaction cost, the result show that the application of security patterns and practices identified increases the cost of the baseline smart contract. But this difference of cost is not significant (estimated to be between USD 0,6 and USD 3,0). As a result and given that smart contract can potentially hold millions of USD in assets, we argue that practitioners must introduce these practices into the development process of smart contract.

2. Literature review

2.1. Blockchain, Ethereum, Smart contract, Transaction cost

Blockchain is the technology that underpins the current interest in cryptocurrencies like Bitcoin. A Blockchain is a decentralized ledger that guarantees its records through cryptographic capabilities and decentralized control [4]. It allows for transactions to be anonymous, immutable and transparent [4].

Ethereum is a decentralized platform based on the Blockchain technology. It supports money transfer using Ether and supports decentralized application execution [5]. Ethereum provides a programming environment to write and run computer program called smart contracts. Solidity is the Turing complete programming language used by Ethereum to write the smart contract. The smart contracts run in a virtual machine called EVM. The Blockchain technology allows to decentralize the execution of the smart contract program [5] and save smart contract resources (code, state) securely. The smart contract code is immutable whereas its states are alterable. Though data transacted through smart contract cannot be tampered, smart contracts are still vulnerable to poor programming practices [1].

The transaction cost is a cost used to perform smart contract operations. This cost is evaluated based on the resources (measured in units of GAS [6]) needed to execute the operation.

$$Tx\ Cost = Gas\ Cost \times Gas\ Price \qquad (I)$$

The transaction cost is calculated according to Equation (I). Gas Cost is the cost (in Gass) that the EVM needed to perform the functions in the transaction. GAS price acts as a motivator for miners to execute and validate a transaction. If the price is high enough, transaction will be executed sooner [7]. In addition, the user provides a Gas

limit for the transaction execution to protect him from depleting his funds due to defects in the smart contract source code [7]. Finally, Token value is
Chen et al. [8] investigated on Solidity and revealed that it fails to optimize gas-costly programming patterns. They identified 7 gas-costly programming patterns. Among them 3 such as dead code is presented in 4,240 real smart contracts. Eliminating these gas-cost patterns can therefore reduce the cost of contract creators and the cost of contract users. Luciano et al. [8] proposed an optimized method of execution of business process on top of commodity Blockchain technology. Their method is focused on initialization cost for process instances, task execution cost privileging a space-optimized data structure, and improved runtime components for maximized throughput.

2.2. Security patterns and practices

Security is an ongoing process involving people, practices, for ensuring application integrity, confidentiality and availability [9].Developing secured software is a big challenge, and is worsened on a language like Solidity. Solidity, is still in constant evolution, which pushed the programmer for continues updates on the languages structures. Furthermore, Solidity provides a familiar style for programmers accustomed to languages like JavaScript, Java and C++ [10] which would potentially give novice Blockchain developers a false sense of confidence (i.e. the language looks familiar, but the underlying platform is not). We argue that these are some of the main causes for the unnecessary introduction of vulnerabilities in Solidity smart contracts.

Nonetheless, this work is based on the assumption that the familiarity of the traditional languages can be exploited to introduce into Solidity time-tested security pattern and practices. The security programming practices are applications or methods of developing software in a way that guards against the accidental introduction of security vulnerabilities [11].Unlike to traditional security solutions which simply treat the symptoms, not the problem, usually do so in a reactive way, Security programming practices like Security programming patterns treat the security problem [12].

A security programming pattern is general reusable solution to a commonly occurring problem in creating and maintaining security [13]. The security pattern is not only included a solution of a problem, but also the context for which the solution is used [14] [15]. One of the general formats for documenting security pattern consist of: Name, Context, Problem, Solution, Structure, Dynamics, Implementation, Example Resolved, Variants, Known Uses, and Consequences [14].

An important number of security patterns and practices guidelines exists for the most used programming languages. For example, the SEI CERT Coding Standards web platform, supports the development of set of guidelines for commonly used programming languages such as C, C++, JAVA [16]. David and Wheeler have provided a set of guidelines for writing secure programs for Linux and UNIX system [17]

For the solidity, a programming language that is still evolving, the security patterns and practices might help to prevent smart contract vulnerabilities. The amount of existing guidelines of security patterns and practices can help in the decision of implementing the security patterns and practices from those programming languages to Solidity.

2.3. Works undertaken for the smart contract security

Smart contracts play a crucial role in the Ethereum Blockchain. As any software program, the free-defect cannot be guaranteed. Even though, Ethereum is a secure platform, security of smart contract cannot be guaranteed. Vulnerabilities can be injected into the code of the smart contact. As a result, Ethereum has been the target of several attacks through smart contract [18]. For instance, the DAO, a crow founding smart contract which has been hacked auto the security flaws into the contract [1].

Daniele et al [2] investigated the verification and validation of smart contract concerning the smart contract security, given the vulnerabilities to which they are exposed. Through their study they shown the importance of automating the verification and validation of smart contract for assuring the smart contract correctness. Sukrit et al. [3] proposed a framework (known as Zeus) that automate solution to verify the correctness and validate the fairness of smart contract. In the same way, Sergei et al [19] provided a comprehensive classification of code issues in Solidity and implemented an extensible static analysis tool (knows as Smart checks) that detects issues. Instead of focusing to the detection of vulnerability, Whorer and Zdun [20] proposed security patterns for smart contract programming in solidity. Given the difficulty of the creation process of writing well performing and secure contracts in Ethereum, they developed six security patterns based on an analysis of collected data with Grounded Theory techniques.

The literature review conducted shows that many researches have been done to find solutions to smart contract security and on another side, to reduce the transaction cost in Ethereum blockchain. But none of those researches pays attention to how costly is the implementation of security to smart contracts.

3. Characterization of transaction cost for security programming practices and patterns

This section describes the research methodology and results of our study. We followed the case study guidelines from Runerson et al. [21].

3.1. Design and Planning

Research Goal: The goal of this research is to characterise the cost of security programming patterns and practices application into the development of smart contract for the Ethereum Platform.

Case definition and unit of measurement: This case concerns the measurement of the change of **transaction cost** after implementing security patterns and practices on the E-voting smart contract. The E-voting smart contract was selected because it is the example smart contract provided by the Ethereum development platform (www.remix.com). The cost is measured in WEI or ETHER and can be equated to dollar value through Cryptocurrency exchange marketplace. As a reference, for this work, the exchange rate from Wei to USD was fixed in October 2019 at $5.5 * 10^{-15}$ (Captured from crypts.info on 8th October 2019). To measure transaction cost, we rely on

the Remix platform when performing a transaction into the Ethereum Blockchain, which provides a measurement of the transactions processes within its platform.

Research question and hypotheses: We have defined the following research questions:

RQ-1: Can security patterns and practices from other programming languages be implemented in Solidity?

RQ-2: How is transaction cost affected by the implementation of security practices and patterns?

For the purposes of our analysis, the following hypothesis has been defined to answer RQ-2.

Hp-1: Transaction cost is increased by the implementation of security practices and patterns.

Instrumentation: We have set up a sandboxed development environment to run this case study. The environment consists of a local Ethereum deployment (supported by Ganache personal Blockchain https://www.trufflesuite.com/ganache). The local Ethereum is managed via Remix online development platform. An unmodified version of the E-voting is used as baseline for measuring Transaction costs. The E-voting smart contract consist of 4 methods. Transaction cost is measured by the Remix platform. For each security pattern or practice within the scope (see Table 2), judgement was made whether the vulnerability was present on the E-voting smart contract, or whether the pattern could be applied in the Ethereum platform. If deemed applicable, a new version of the E-Voting smart contract was created which implemented the selected patterns, and transaction cost was measured each of the method of the smart contract. Table 1 presents the measurement of transaction cost for each method in the E-Voting smart contract.

Analysis procedure: From the collected measurement of transaction cost for each security pattern and practices, we have compared the transaction cost of each method on which the security patterns and practices have been applied to the baseline transaction cost.

The CERT secure coding guidelines [16] were selected as the source of the secure coding guidelines, and the results of an unstructured literature review is the source of the security patterns.

Table 1 Baseline measurement for E-Voting smart contract transaction cost

E-Voting method	Baseline Transaction Cost (Ether)
Constructor	$1.26*e^{-2}$
Delegate	$1.10*e^{-3}$
Vote	$1.37*e^{-3}$
GiveRightToVote	$8.74*e^{-4}$
Winning Proposal	0

3.2. Result

*Answer to **RQ-1:** Can security patterns and practices from other programming languages be implemented in Solidity?* Table 2 presents the results of our review of security patterns and practices. In the interest of space, we only present those which were deem applicable to Solidity. We found that the context of software coding in

JAVA and C++ for the application of security patterns and practices in the Table 2 are similar to the context of smart contract coding with Solidity. The others patterns and practices explored from our sources and not listed in table 2 were deemed not applicable given their application context and also features which are not present in Solidity. For instance one of the security practices reviews was SQL injection (both from Java and C++). We claim, that this security practice cannot be applied to the Solidity given Ethereum has not features for SQL Database communication. In Table 2, the first ten represent those which the E-Voting smart contract provided implementation opportunities.

Table 2 List of security patterns and practices applicable in solidity

No	Security programming patterns and practices applicable to Solidity
1	Conditionally executed blocks should be reachable
2	defensively copy mutable input and mutable internal component
3	emergency stop
4	guard-check
5	mutex
6	detect prevent integer overflow
7	Ensure actively held locks are released on exceptional conditions
8	Speed bump
9	Ensure that unsigned integer operations do not wrap
10	Ensure that compound operation on shared are atomic
11	Prevent initialization cycle
12	public identifier from the Standard library of language
13	Ensure conversion of numeric types to narrower types do not result in lost or misinterpreted data
14	Limit accessibility of fields
15	Use integer types that can fully represent the possible range of unsigned data
16	Never use assertion or validate method arguments
17	Methods that perform a security check must be declared private or final
18	Ensure that constructors do not call overridable methods
19	Synchronize access to static fields that can be modified by untrusted code
20	Do not override thread-safe methods with methods that are not thread-safe
21	Do not let this reference escape during object construction
22	Do not publish partially initialized objects
23	Do not leak memory
24	Do not declare variables inside a switch statement before the first case label
25	Do not depend on the order of evaluation for side effects
26	Do not access a volatile object through a non-volatile reference
27	Do not read uninitialized memory
28	Checks-Effects-Interaction
29	Rate Limit
30	Balance Limit

To sum up and answer RQ1, security patterns and practices from other programming languages can be implemented in Solidity, as shown by Table 2, and those that we could implement in the E-Voting smart contract.

*Answer to **RQ-2**: How is transaction cost affected by the implementation of security practices and patterns*? Table 3 presents the results of measuring transaction cost for all security programming patterns and practices applicable in our case study.

Table 3 Transaction cost in ETHER *(1 ETHER = 10^18 WEI)* of each pattern and practice applicable in the E-voting smart contract. *(NA= same cost as the baseline due to pattern or practice not being applicable on these methods)*

Security practices/patterns	constructor	vote	delegate	GiveRighTovote
Conditionally executed blocks should be reachable	1.27E-02	1.37E-03	1.10E-03	NA
defensively copy mutable input and mutable internal component	1.59E-02	1.61E-03	1.34E-03	NA
emergency stop	1.41E-02	1.38E-03	1.12E-03	8.80E-04
guard check	1.29E-02	1.38E-03	NA	NA
mutex	1.39E-02	1.51E-03	1.62E-03	1.39E-03
detect prevent integer overflow	1.30E-02	1.38E-03	NA	NA
Ensure actively held locks are released on exceptional conditions	1.35E-02	1.59E-03	1.32E-03	NA
speed bump	1.34E-02	NA	1.13E-03	NA
Ensure that unsigned integer operations do not wrap	1.38E-02	1.39E-03	NA	1.12E-03
Ensure that compound operation on shared are atomic	1.60E-02	2.13E-03	1.64E-03	NA

The figure below presents the transaction cost with patterns and practices compared to the baseline cost of E-voting methods. As shown in Figure 1, the transaction cost after implementation of security patterns or practices is typically greater than the baseline. However, to evaluate **Hyp-1,** we computed the difference between the baseline values presented in Table 1 and the transaction cost measurements in Table 3. Our results show that the minimum difference is $3{,}50*e^{-7}$, the mean difference is $1{,}27*e^{-4}$ and maximum difference is $3{,}32*e^{-3}$. Therefore, according to our measurement, our hypothesis holds and we can claim that our *evidence* shows that Transaction cost is increased by the implementation of security practices and patterns.

To sum up and answer RQ2, transaction cost is affected by the implementation of security patterns by increasing and observable increase in the cost.

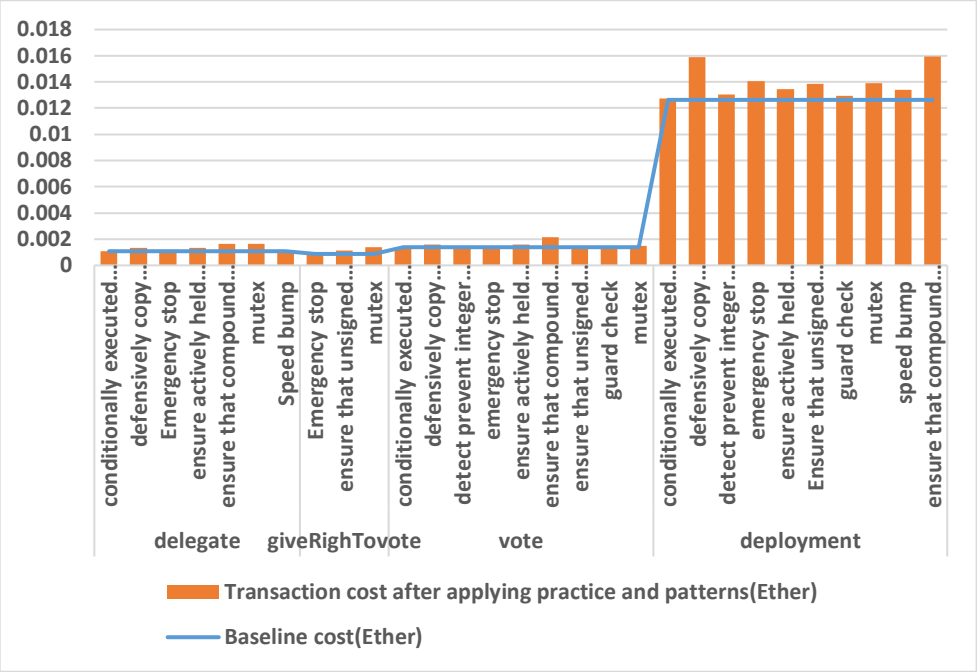

Figure 1. Transaction cost before and after implementing of each security practice and pattern of each method.

4. Discussion

- *Concerning the applicability of security patterns and practices:* The result shows that patterns and practices from other programming languages can be applied to the development of smart contract in Solidity (see Table 2). However, applicability entails opportunity of application. Not all security patterns and practices in the literature can be applied to Solidity (see section 3.2). And specifically, to the E-Voting smart contract. As described by Yoshioka et al. [22], implementing a security patterns requires that the context and the environment must be taken into consideration. The E-Voting smart contract provided a good initial context for evaluating the transferability of security practices and patterns. In contrast, it does not provide opportunities for evaluating the applicability of security patterns and practices that aim at mitigating vulnerabilities that arise due to the communication/collaboration of software components.
- *Concerning the increase of transaction cost when implementing security patterns*: Table 3 shows that the implemented security patterns and practices affect the cost. In line with the hypothesis 1, for each pattern and practice applied, the transaction cost is greater than the baseline cost. Security patterns and practices are implemented

to minimize vulnerabilities. In general, this implies that new code has to be added, that is not necessarily needed by the business. Thus, it is reasonable to expect an increase in computation time, which in Ethereum, is translated into cost. Our empirical evidence shows that the mean cost of the implemented security patterns and practices in the E-Voting smart contract was 1,27*e-4 which corresponds to USD 0,023. From a business perspective, this is not a significant cost. If we were to introduce all of the evaluated security practices into the E-voting smart contract, this cost difference can be estimated to between 3,81*e-3 and 1,6*e-2 Ether (equivalent USD 0,6 to USD 3,0). Given those costs, and the relative values of the information currently stored in Ethereum Smart Contracts, it is hard to argue against implementing security patterns and practices.

5. Conclusion

In this paper, we have characterised the transaction cost of smart contracts using security programming practices and patterns. The characterisation has been conducted through a case study. The study has shown the practices and patterns from other languages can be applicable to Solidity. We have identified 30 security patterns and practices that can be applicable to Solidity, and implemented 10 in a representative Smart Contract.

Based on this case study, we conclude that the application of security practices and patterns increases the transaction cost. Nonetheless, our results suggest that the cost in USD of implementing these secure programming patterns and practices should be affordable for any enterprise looking to develop Smart Contracts as part of their business. In contrast, the literature has shown examples of smart contracts that have been hacked through vulnerabilities that should have been mitigated. Given the findings of this research, we call on practitioners to introduce secure programming patterns and practices into their Solidity development process.

For the future work we will extend this case study while looking at the application of secure programming patterns and practices that deal with the communication between smart contracts and components not within the Ethereum EVM.

References

1. Michael del Castillo. The DAO Attacked: Code Issue Leads to $60 Million Ether Theft, 2016. https://www.coindesk.coom/dao-attacked-code-issue-leads-60-million-ether-theft/, last accessed 2019/05/25.
2. Magazzeni, D., McBurney, P. and Nash, W., 2017. Validation and verification of smart contracts: A research agenda. Computer, 50(9)
3. Kalra, S., Goel, S., Dhawan, M. and Sharma, S., 2018, February. Zeus: Analyzing safety of smart contracts. In 25th Annual Network and Distributed System Security Symposium, NDSS.
4. Nakamoto, Satoshi. "Bitcoin: A peer-to-peer electronic cash system." available at http://bitcoin.org/bitcoin.Pdf , last accessed 2019/04/30
5. Buterin, V. (2014). "A next-generation smart contract and decentralized application

6. Żuchowski, Ł. (2017). "Ethereum: Everything you want to know about Gas." From https://blog.softwaremill.com/ethereum-everything-you-want-to-know-about-the-gas-b7c8f5c17e7c, last accessed 2019/06/12.
7. Chen, T., Li, X., Luo, X. and Zhang, X., 2017, February. Under-optimized smart contracts devour your money. In *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)* (pp. 442-446). IEEE.
8. García-Bañuelos, L., Ponomarev, A., Dumas, M. and Weber, I., 2017, September. Optimized execution of business processes on blockchain. In *International Conference on Business Process Management* (pp. 130-146). Springer, Cham.
9. Pfleeger, C.P. and Pfleeger, S.L., 2002. Security in computing. Prentice Hall Professional Technical Reference
10. Dannen, C., 2017. Bridging the Blockchain Knowledge Gap. In Introducing Ethereum and Solidity (pp. 1-20). Apress, Berkeley, CA
11. Wikipedia "Secure coding" https://en.wikipedia.org/wiki/Secure_coding/, last accessed 2019/05/12
12. Viega, J. and McGraw, G., 2011. Building Secure Software: How to Avoid Security Problems the Right Way (paperback)(Addison-Wesley Professional Computing Series). Addison-Wesley Professional
13. Schumacher, M., Fernandez-Buglioni, E., Hybertson, D., Buschmann, F. and Sommerlad, P., 2013. *Security Patterns: Integrating security and systems engineering*. John Wiley & Sons.
14. Steel, C. and Nagappan, R., 2006. *Core Security Patterns: Best Practices and Strategies for J2EE", Web Services, and Identity Management*. Pearson Education India
15. Andrews, M. and Whittaker, J.A., 2006. *How to Break Web Software: Functional and Security Testing of Web Applications and Web Services*. Addison-Wesley Professional.
16. SEI CERT Code standard https://wiki.sei.cmu.edu/confluence/display/seccode/SEI+CERT+Coding+Standards, last accessed 2019/05/15
17. D. A. Wheeler, "Secure Programming for Linux and Unix HOWTO," 1999. http://www.dwheeler.com/ secure-programs/.
18. Atzei, N., Bartoletti, M. and Cimoli, T., 2017, April. A survey of attacks on ethereum smart contracts (sok). In *International Conference on Principles of Security and Trust* (pp. 164-186). Springer, Berlin, Heidelberg
19. Tikhomirov, S., Voskresenskaya, E., Ivanitskiy, I., Takhaviev, R., Marchenko, E. and Alexandrov, Y., 2018, May. Smartcheck: Static analysis of ethereum smart contracts. In 2018 IEEE/ACM 1st International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB) .IEEE.
20. Wohrer, M. and U. Zdun (2018). Smart contracts: security patterns in the ethereum ecosystem and solidity. 2018 International Workshop on Blockchain Oriented Software Engineering (IWBOSE), IEEE.
21. Runeson, P., Host, M., Rainer, A.W., Regnell, B.: Case Study Research in Software ¨ Engineering. Guidelines and Examples. Wiley, Hoboken (2012)
22. Yoshioka, N., Washizaki, H. and Maruyama, K., 2008. A survey on security patterns. Progress in informatics, 5(5), pp.35-47.