UDC 004.056

Y. Baryshev
D. Zarezenko

SECURE APPLICATIONS OF CREATION DESIGN PATTERNS FOR SMART CONTRACTS

Vinnytsia National Technical University

Анотація
Розглянуто задачу застосування твірних шаблонів проєктування під час розробки смарт-контрактів для розподілених застосунків. Обґрунтовано актуальність аналізу безпеки застосування цих шаблонів перед їх реалізацією засобами мови Solidity. Розглянуто основні твірні шаблони. Визначено аспекти кібербезпеки, на які потрібно звертати увагу під час їх реалізації. Наведено рекомендації щодо застосування. Запропоновано приклади доречного використання шаблонів при розробці смарт-контрактів. Виявлено спільні проблеми безпеки твірних шаблонів при застосування для реалізації смарт-контрактів та визначено перспективи подальших досліджень для пом'якшення їх негативного впливу на безпеку розподілених застосунків.
Ключові слова: блокчейн, кібербезпека, смарт-контракти, захист даних, шаблони проєктування, розподілені застосунки.

Abstract
The problem of creation design patterns utilizing when developing smart contracts for distributed applications, is considered. The importance of analyzing the security of these patterns' usage before implementing them by the means of Solidity language is substantiated. The main creation patterns are considered. The cybersecurity aspects that need to be paid attention to during their implementation are identified. Recommendations for their application are given. Examples of appropriate usage of the patterns at the smart contracts development are proposed. Common security problems of creation patterns application for the smart contracts implementation are identified and prospects for further research are determined in order to mitigate their negative impact on the security of distributed applications.
Keywords: blockchain, cybersecurity, distributed technologies, smart contracts, data protection, design patterns, distributed applications.

Introduction

Design patterns aid software developers at their task solving in many ways. For instance, they facilitate communication, can be used as a best or worst practices for certain tasks solutions etc [1-3]. Some of them are relatively programming language independent, while others heavily depend on certain language peculiarities [2, 3]. Skills of successfully applying the former ones helps software developers to switch languages and quickly learn new ones [2, 3]. However design patters usage at the distributed applications (DApps) development, that involves Solidity programming language [4] for blockchain stored smart contracts creation, is a tricky one due to security reasons [1, 5, 6]. Despite Solidity being the object-oriented language [4] which technically allows software developers to use the most of language independent patterns, resulting software execution won't be the one, which was assumed by the GoF [3], who originally introduced them, and the consecutive works those further develop the original idea [2, 7]. Those are caused by the stored data openness and distributed nature of execution process at different nodes. Therefore an important task appears to analyze security impact of language independent creation patterns implementation at the smart contracts development process using Solidity.

The aim of the research is to enhance DApps cybersecurity by analyzing and drawing recommendations of secure language independent design patterns application for the smart contracts development.

Creation Design Patterns Security Analysis

Creation patterns — ones used to create instances of the object/class/smart contract. They are very useful for the smart contract cases, because they allow to create new smart contracts to the DApps after the initial code deployment to the blockchain. This allows to provide flexibility to created DApp. The well-known language independent creation patterns are the following:

- ***Factory*** – generates an instance of the smart contract without providing any instance logic [3, 5-7]. The pattern might need to be supplemented by access control modifiers in case, when business logic forbids calls of the factory by any user. It should be used for predefined contracts deployment, for instance, creation of the one-time used ticket.
- ***Factory method*** – is similar to the factory, but it allows to provide variety to created instances thus adapting to the different tasks at the cost of logic complexity increasing [5, 7]. This pattern as well may need to be supplemented by access control modifiers. It should be used for the smart contracts those have several possible ways of business processes handling.
- ***Abstract factory*** – is basically factory of other factories [5, 7]. As other factories it might need access control enhancement via modifiers. Due to its logic complexity and blockchain's block gas limitations (i.e. logic complexity and storage capacity cap) this design pattern might be less useful than above-mentioned ones. However it is feasible to be implemented and the pattern is essential in cases, when business restrictions draw need in creation of a several smart contracts instances set and furthermore the ability to create different sets for different cases is required. For instance, the pattern would be useful for multiple blockchain interaction instances.
- ***Builder*** – creates custom instances on the basis of user-defined input data [5, 7]. Thus a builder allows to provide more adaptability to a DApp in comparison with other creation design patterns. The major security issue for the pattern along with a need of access control lies in the testing area. Instead of the common code implemented the builder testing the created instances should be tested as well, However due to vast variety of instances those are possible to create using a builder exhaustive security testing of each created instance might be very resource consuming comparatively to the previously mentioned patterns.
- ***Prototype*** – clones existing instances [5, 7]. Usually the pattern is good for making custom set instances form the preset instances. However cloning is resource consuming in Solidity due to storage constraints, making this pattern impractical. Therefore it is better avoid its usage at the current stage of the blockchain technology development, particularly smart contracts' way of execution by the blockchain nodes.
- ***Singleton*** – ensures that only one instance of a particular class is ever created [5-7]. The pattern is useful for creation of unique objects, those find many implementation as a non-fungible tokens. More over the pattern is implemented as a standard smart contracts ERC-721, ERC-1155 [5, 6]. Thus one can safely use a singleton by the means of these best practices.

The analyses of these creation design patterns show the common need of them to provide access control in order to avoid their misuse by the third parties. Another security issue that needs to be addressed is related to the extensive quality assurance and security testing covering beside the applied pattern's code base, but the created code base by this pattern. The latter causes additional complexity for the development. Thus the testing should be performed by either by another smart contract developer, who is capable in security testing techniques or by the testing specialist, who is capable in smart contract development.

Discussion

The application of the best practices of software development such as design patterns induce higher development standards. Those include cybersecurity enhancement. However distributed applications, smart contracts in particular, are performed differently from desktop, embedded or web applications. Consequently both quality and security parameters should be reviewed for them [1]. Therefore the best practices are to be reviewed as well.

This research considers creation design patterns application for Solidity programming language which is used in the most blockchains for the smart contracts development purpose. The known works review shown lack of relevant research in the field. The general scarcity of works about design patterns security analyses impact the case of the smart contracts development focusing mostly on the use-cases rather than on generic cases and metrics [5, 6]. In order to fill the gap this research contains the generalization of the creation design patterns analyses, which allowed to show the common drawback for the most of the creation patterns application for DApps development purpose – a need of access control injection for these patterns.

Consequently the perspective of further development for the research is the studying of ability of access control integration to the creation design patterns as well as comparative analyses of implementation ways. Moreover security of the language specific creation design patterns should be performed in order to define possible similarities between these two kinds of patterns. Therefore the outcome of the latter research may aid generic approach of design patterns security enhancement.

REFERENCES

1. K. Hermann et al. A Taxonomy of Functional Security Features and How They Can Be Located. *Preprint submitted on 8 Jan 2025*. 41 p. URL: https://arxiv.org/pdf/2501.04454 (accessed 12.01.2025).

2. S. M. Peldszus. State of the Art in Secure Software Systems Development in Security Compliance *in Model-driven Development of Software Systems in Presence of Long-Term Evolution and Variants.* 2022, Pp. 37–63. doi: 10.1007/978-3-658-37665-9_3.

3. E. Gamma, R. Helm, R. Johnson, J. Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software. Toronto: Addison-Wesley, 1995, 397 p.

4. Solidity: Documentation. URL: https://docs.soliditylang.org (accessed 12.01.2025).

5. Y. Baryshev. Design Patterns Security Analysis for Blockchain-based Applications Development with JavaScript and Solidity. *Матеріали XLVIII науково-технічної конференції підрозділів Вінницького національного технічного університету.* Вінниця, ВНТУ, 2019. С. 865-868. URL: https://conferences.vntu.edu.ua/public/files/1/fitki_2019_netpub.pdf (accessed 12.01.2025).

6. M. Wöhrer, U. Zdun. Design Patterns for Smart Contracts in the Ethereum Ecosystem. *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, Halifax, NS, Canada, 2018, pp. 1513-1520, doi: 10.1109/Cybermatics_2018.2018.00255

7. A. Kamran et al. Design Patterns for Humans! URL: https://github.com/kamranahmedse/design-patterns-for-humans/blob/master/readme.md (accessed 12.01.2025).

Баришев Юрій Володимирович — канд. техн. наук, доцент кафедри захисту інформації, факультет інформаційних технологій та комп’ютерної інженерії, Вінницький національний технічний університет, Вінниця, e-mail: yuriy.baryshev@vntu.edu.ua

Зарезенко Дмитро Павлович — аспірант кафедри обчислювальної техніки, факультет інформаційних технологій та комп’ютерної інженерії, Вінницький національний технічний університет, Вінниця, e-mail: dmytro.zarezenko@gmail.com

Yurii Baryshev — PhD. (Eng), Associate Professor of Information Protection Department, Faculty of Information Technologies and Computer Engineering, Vinnytsia National Technical University, Vinnytsia, email: yuriy.baryshev@vntu.edu.ua

Dmytro Zarezenko — Postgraduate Student of Computer Engineering Department, Faculty of Information Technologies and Computer Engineering, Vinnytsia National Technical University, Vinnytsia, e-mail: dmytro.zarezenko@gmail.com