Smart Contracts Business Model Canvas

Silviu OJOG, Alina-Andrea MIRON
Bucharest University of Economic Studies, Romania
silviu.ojog@csie.ase.ro, alina.miron@mk.ase.ro

Smart Contracts are the central piece of Ethereum and other compatible blockchains. Their role is to build trusted functionality that unknown parties can interact with. However, their value proposition can be undermined by different security exploits. In many cases, vulnerabilities are overlooked not due to neglect but due to a systematic approach in the review process. This paper aims to appeal to existing frameworks for understanding the business context and provide standardized thinking on auditing smart contracts. The power of a framework lies in the fact that it ensures that auditors do not overlook critical aspects of their vulnerability.

Keywords: *Blockchain, Smart Contract, Business Model Canvas, Audit, Ethereum, Exploit, Vulnerability, Solidity, Security*
DOI: 10.24818/issn14531305/29.1.2025.04

1 Introduction

The first blockchain, Bitcoin, was created by Satoshi Nakamoto in 2009 to store financial transactions [1]. It was built around the idea of a decentralized, distributed, and shared digital ledger. Ethereum, launched in 2015, is considered the second generation of blockchain [2]. Innovation was brought by the creation of a platform based on smart contracts. Its innovation consisted of creating a platform based on smart contracts, which are software with customizable functionalities [3].

Hence, blockchain promises, such as immutability, transparency, decentralization, and fault tolerance, can be applied to any custom logic in smart contracts. Contracts become even more decisive as any error at the implementation level can lead to dangerous consequences, often meaning the loss of available funds. As with any maturing industry, blockchain must create standards around understanding its vulnerabilities, creating best practices, and enforcing them [4].

To be more easily understood, patterns [5] other standards need to be designed by drawing on existing mental models of those involved in the industry, as well as other frameworks with similar impacts from different sectors.

Since smart contracts can hold data and significant amounts of money, not only can they attract many types of attackers, but they could be considered a business of their own. This paper aims to analyze the technical vulnerabilities of smart contracts using business frameworks such as Business Model Canvas [6] or the Business Model Navigator [7].

2. The Smart Contract mindset

Smart contracts do require a shift in security mindset. Unlike traditional pieces of software, smart contracts are immutable, their code is publicly accessible and interactable, and they can be used to deposit money. A subtle difference is that the smart contract code itself is immutable, while the data the smart contract holds can be changed. Updates on the code are not performed in the traditional manner. The only possible way to upgrade a smart contract logic is to deploy a new smart contract and permanently stop access to the initial contract. Holding large amounts of money could potentially attract bad actors as the reward to effort ratio is significantly higher than in other areas. In case an attacker seizes the funds f of a smart contract, he can monetize more easily than in a traditional data breach [8].

Smart contracts were invented with the purpose of extending Blockchain use cases beyond financial transactions. Smart contracts were designed to be Turing complete as opposed to Bitcoin scripting language. However, the Turing completeness in smart

contracts means they can run programming loops within a limited number of steps. The purpose of this limitation is to counter malicious code with blocking or infinite loop. Smart contracts were designed to charge for every computation performed using a concept named gas. A gas limit is imposed on every smart contract transaction. Over the years, gas optimization has become an important subject [9]. Solutions, such as such as second-layer blockchains, have been developed in order to mitigate the spent-on transactions.

A more pragmatic approach is needed when dealing with smart contracts instead of an iterative start-up-driven mindset. Before the actual deployment, it is considered good practice to have two external entities audit the smart contract [10].

3. Types of vulnerabilities

Blockchain and namely Bitcoin, was created, combining different existing computer science theories, algorithms and data structures. Smart contracts are a result of second-generation innovation on Blockchain. They bring the programming logic into Blockchain. They are pieces of software meant to interact with transactions and other smart contracts and save data in a Blockchain state.

As a result, Blockchain vulnerabilities can be categorized into three buckets: general programming vulnerabilities, Blockchain, specific vulnerabilities and platform, specific vulnerabilities.

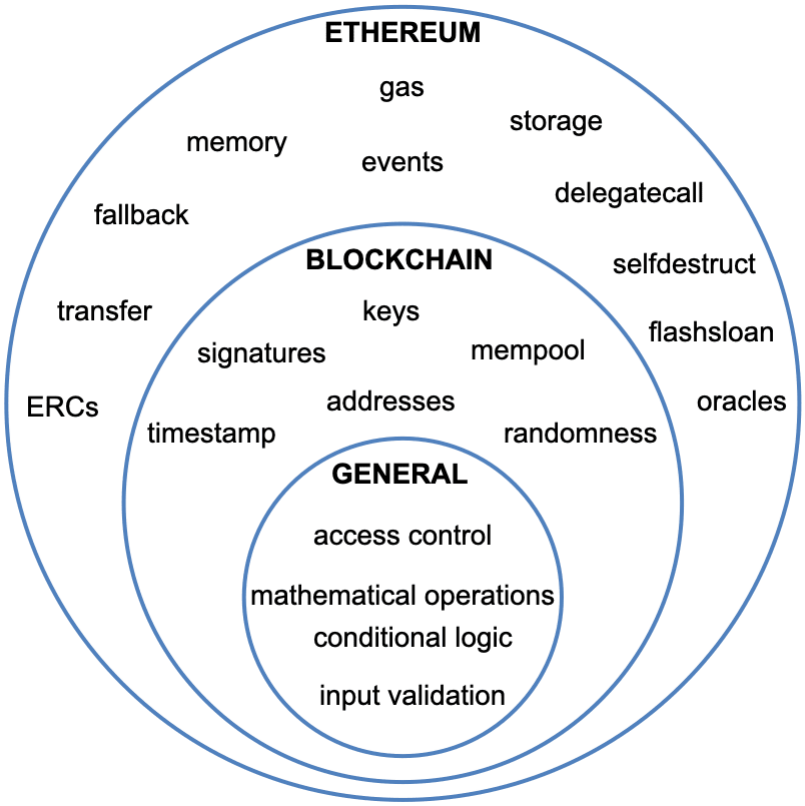

Fig. 1. Vulnerability layers in smart contracts

Figure 1 depicts the three categories of vulnerabilities in small contracts. When integrating Ethereum with other blockchains or L2 solutions, another type of vulnerabilities can emerge.

The concept of access control is not particular to Blockchain or smart contract platforms. It may be used to restrict certain smart contract functionality to particular addresses. Depending on the contract logic, several roles may emerge for those addresses, such as contract owner, deployer, recovery address, oracles, or regular user. Vulnerabilities may arise from assigning incorrect roles, transferring privileges or validating privileges on incorrect parameters.

Another concept highly used throughout programming, but has a particular meaning in blockchain is that of timestamping. Blocks are timestamped in order to create the digital ledger. However, block timestamps are in the control of the miners, namely the block creators. Early randomness is predictable and can be tempered with.

Some of the attacks may be influenced by the three layers, the general, the Blockchain, and the Ethereum one. However, they may be performed differently. For example, a DDOS, attack is in theory not possible on the whole Blockchain network due to its distribution nature. However, an attack on particular transaction can be performed by front running the transaction on the Blockchain level. Gas based denial of service is a Ethereum particular vulnerability which exploits the mechanism of gas. The gas limit will be triggered whenever big complication is needed such as looping through a long array. Gas based attacks may be performed by filling up a certain array with dummy data [11].

4. The business of smart contracts

According to the Cambridge Dictionary, a contract is a legal document that states and explains a formal agreement between two different people or groups [12]. However, smart contracts are not legal agreements. Nor they are not self-enforceable.

Blockchain started from a need of decentralization. Blockchain is the middleman replacer for financial transactions. Smart contracts can be treated like businesses. They are in the business of enabling trust and transparency between multiple parties while adding custom logic.

4.1 Smart Contracts Business Model

The Business Model Canvas, developed by Alexander Osterwalder and Yves Pigneur [6], is a strategic tool that allows for visualization and ideation on different business ideas or concepts. It is a one-page document containing nine quadrants that represent different fundamental elements of a business. Other variations have
been created over the years, such as the lean model canvas or value proposition canvas.

The 9 aspects of Canvas are as follows:

1. Value proposition: The products and services that create value for a specific value segment, solving a problem or satisfying a need.
2. Customer segments: The groups of people or that the company intends to serve and create value for.
3. Channels: The ways in which the company communicates and delivers the value proposition to purchasing segments.
4. Customer relationships: The types of relationships a company establishes with specific customer segments.
5. Revenue streams: The ways in which the business generates revenue from the value proposition it offers to customers.
6. Key resources: The most important assets needed to make the business model work.
7. Key activities: The most important actions the company must take to make its business model work.
8. Key partners: The network of suppliers and partners that make the business model work.
9. Cost structure: All the costs involved in operating the business model.

Smart contracts do not have customers, meaning they do not sell products in the traditional way. Instead, they have user actors who represent particular addresses that can interact. In addition to regular users, there can also be contract owners who have special management rights. They can be those who submitted the contract or different entities. It can also be a single entity or several entities. Certain entities can interact with the contract, but following some actions taken by it, we can include them in the key partner's section.

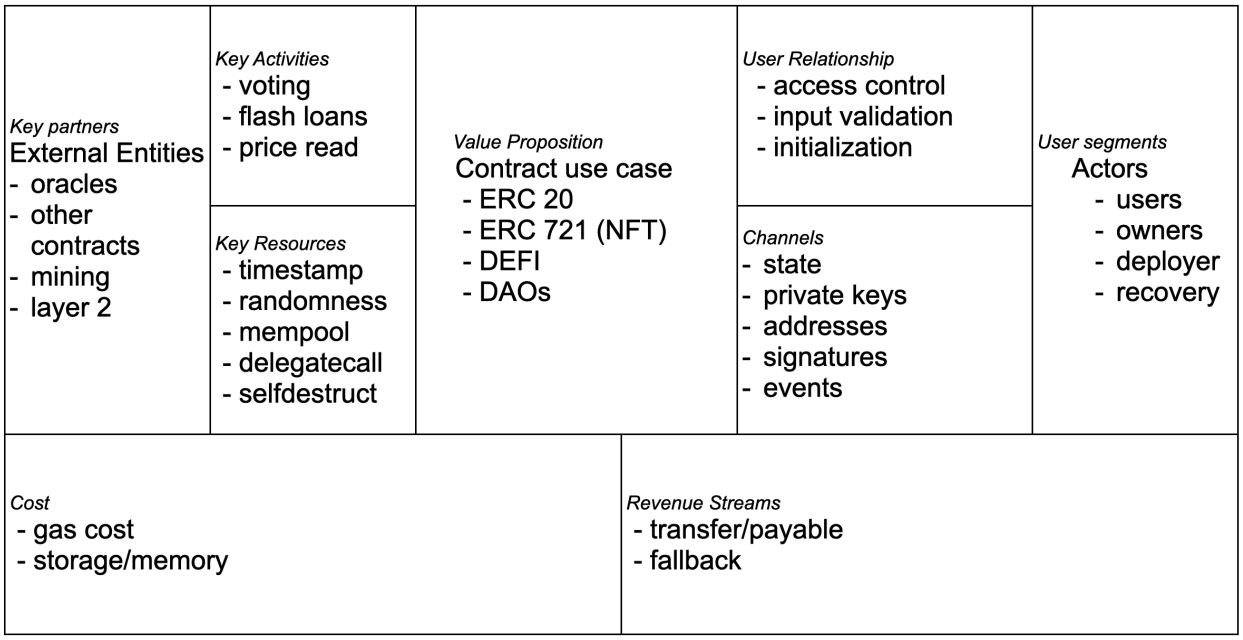

Fig. 2. Rekt Test Business Model Canvas

In Figure 2, one can see the 12 questions of the rekt test proportionally being mapped to six out of the nine quadrants of the business model canvas. Figure 2 represents the Business model Canvas for the specific elements of a Smart contract. Over the years, specific value propositions have emerged, starting with the creation of tokens (ERC20) and continuing with the creation of NFTs (ERC721) and their use in DAO or the DEFI world. These cases can be dissected more depending on their use in specific business verticals, but implementations are necessary from a security point of view.

4.3 The "Rekt" test

The Rekt test was created by a group of blockchain security experts led by Trail of Bits CEO Dan Guido [13]. "Rekt" is a slang term for "wrecked" or "destroyed" in the context of cryptocurrency.

The test is modelled after The Joel Test, which assesses the quality of software development teams.

The Rekt Test is a set of 12 simple yes/no questions designed to evaluate the security practices of the smart contract-based protocols and their development teams.

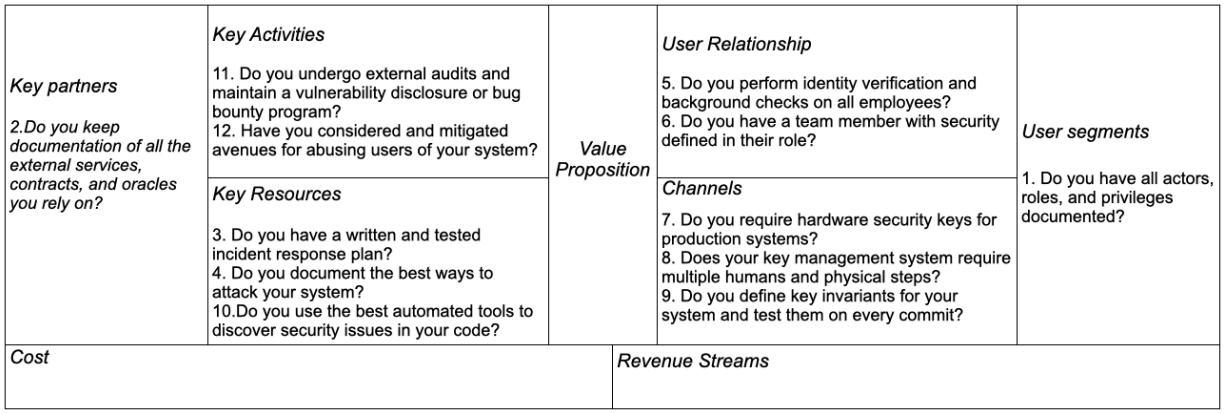

Fig. 3. The "Rekt" Test Business Model Canvas

Figure 3 we can see the 12 questions of the Rekt test proportionally being mapped to six out of the nine quadrants of the business model canvas. The Rekt test does not bring

into question the value of the contract product nor the mechanisms related to funds.

4.2 The who-what-how-why

The 'business model' describes the sum of activities, inputs and outputs surrounding value creation. The "who-what-how-why" framework is used to define business models and dig deep into the intricacies of modelling. According to the Business Model Navigator [7], the "who-what" addresses its external aspects and "how-why" its internal dimensions. When treating smart contracts as a business, one can model its value proposition, access, and internal mechanisms. The aim of the framework is to underline the user and actor segments, calling the contracts, the values passed to and retrieved from smart contracts and the internal mechanism of the smart contracts or of those relied on.

.

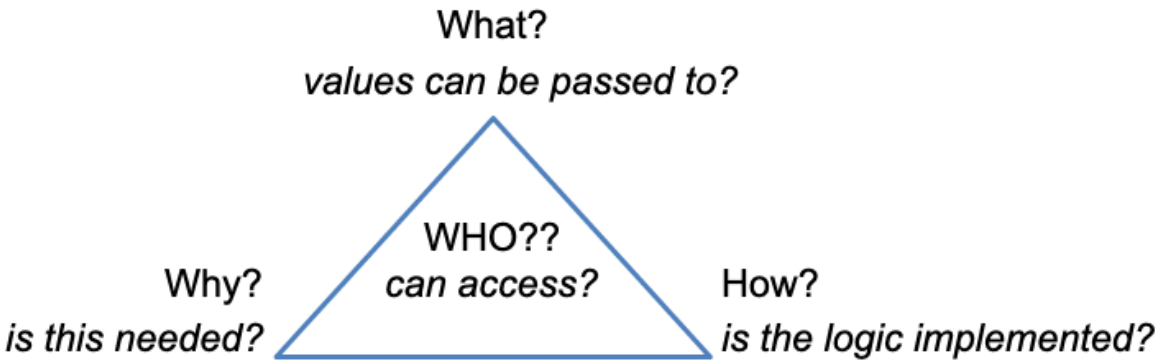

Fig. 4. The who-what-how-why framework

Figure 4 depicts the array of questions one needs to ask in order to map contract activities. Contracts are accessed through transactions, which can receive data natively in ether. Properly handling interactions with external transactions requires implementing robust input validation mechanisms. Rigorous parameter checking, limiting acceptable values, and implementing edge case protections dramatically reduce the risk of exploitation. Using specific authorization and validation modifiers (such as those in the OpenZeppelin library) provides an additional layer of protection.

Securing interactions involving ETH requires careful design of payable functions. Validating received values, implementing minimum and maximum limits, and logically separating funds processing from other contract operations are essential. The pull payment pattern, in which users claim their funds themselves instead of receiving them automatically, can prevent many blocking attacks.

To better understand the overall perspective, we need to map the interaction area of a Smart contract. A smart contract can interact with a transaction coming from an externally owned account, another Smart contract, or it can send transactions to the Ethereum Virtual Machine. Figure 5 maps the interactions that a contract can have. A transaction can only be started from an externally owned account. This is also the tx.origin variable. Before the transaction reaches the destination contract, it can pass through other contracts. The last contract is then "msg.sender" variable. In both cases, the inputs and access rights must be validated whether a transaction comes directly from a contract or from an external account. In figure number [5], the Input Smart contract and External Account represent the potential actors communicating with an audited counterparty. When a smart contract calls another Smart contract, it can use the fallback/receive function mechanism to execute certain codes multiple times.

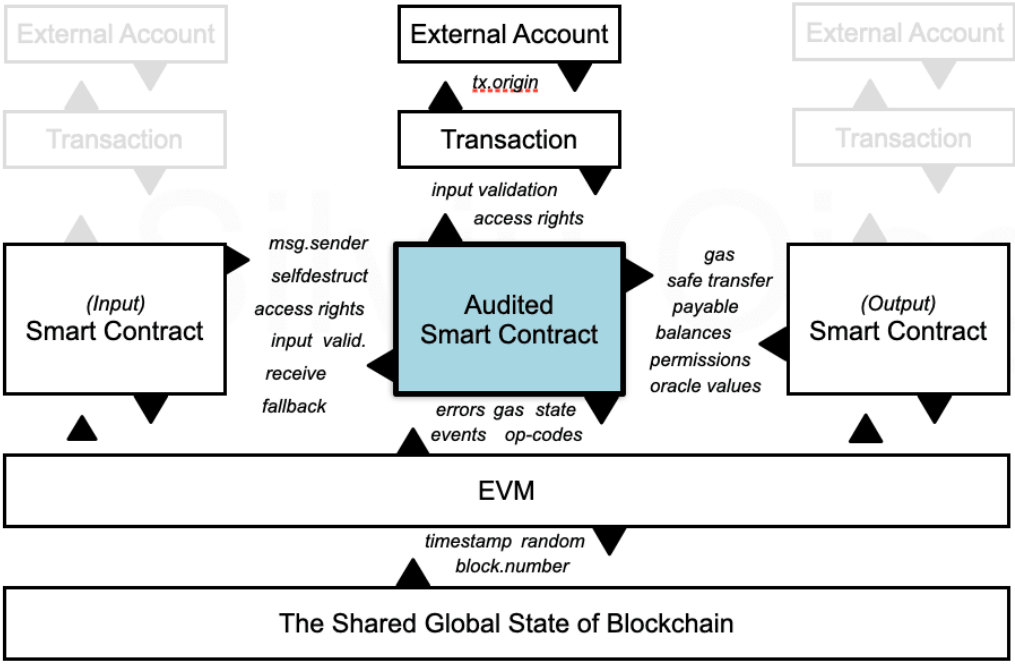

Fig. 5. Smart Contract Interactions

Because of this built-in functionality in Ethereum, there can be attacks such as reentrancy attacks, the most famous being the DAO hack [14]. In figure number [5], the Output Smart Contract represent partner contracts from which values can be read. If these values can be manipulated, the logic will also be faulty. An audited contract can call another contract to read the state of an NFT or the balance of a token. As a result, it can call to send funds. If the contract that receives those funds does not support certain functionalities, these funds can be lost.

4.3 The time component

The smart contract life cycle has three main stages. The first phase is the pre-deployment phare, with activities related to design, development and testing of smart contracts. The design phase establishes the actors, their economic incentives, and the reward mechanism. Design vulnerabilities may create systematic problems, harder to fix, using a simple patch approach. Hence, comprehensive documentation of all use cases and limitations, mathematical modeling and simulation of extreme scenarios and implementation of safety mechanisms such as emergency stop or circuit breakers.

It is considered a good practice, in the development phase, to use highly tested and audited libraries, such as OpenZeppelin [11], implementing a wide range of test, unit, integration, fuzzing, and code review and formal audits from external parties.

The project launch can be vulnerable to assets initial price manipulation, and unfair token distribution. Protecting the launch moment requires implementing anti-bot mechanisms and using whitelist addresses for verified participants. Limiting transactions per address prevents the concentration of resources. Implementing vesting schedules reduces the impact of immediate massive sales. Using dutch auctions or other price discovery mechanisms ensures a more accurate valuation and minimizes the opportunity for manipulation.

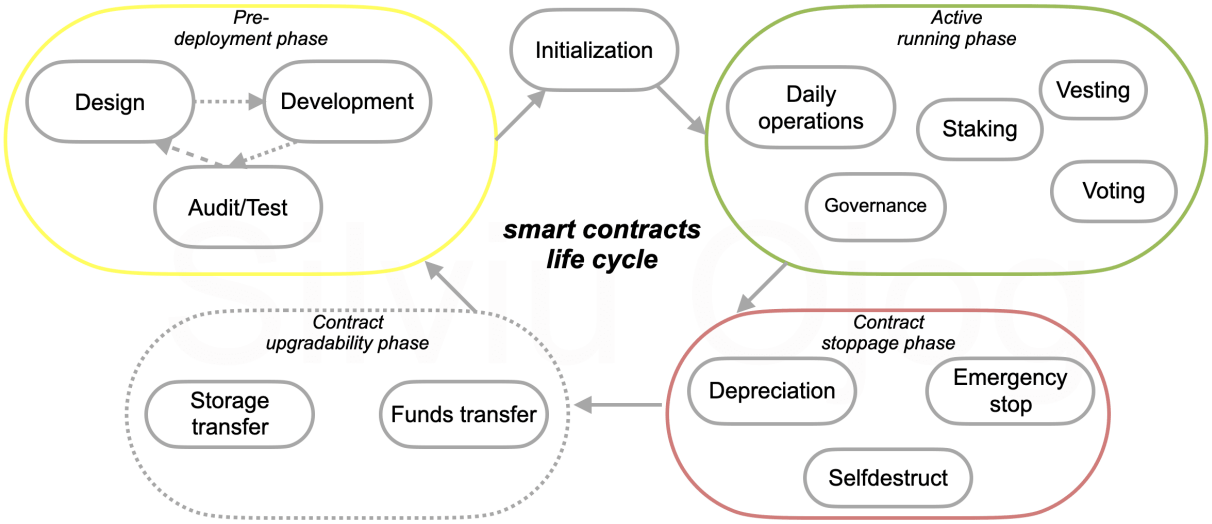

Fig. 6. Second Layer Deployment Process

Figure number 6 depicts the life cycle of Smart contracts. In the first phase, the contracts are developed and audited before being deployed in the blockchain. This phase is vital from the perspective of the smart contracts paradigm: immutable pieces of code. Hence, the time spent on this phase must be more significant than other software products to properly and thoroughly document, test, and audit the contracts. Best practices for language-specific matters must be followed [15]. For example, in the crypto world, all code sources must go through at least two external auditor processes. Depending on the feedback received from these entities, it can be reiterated at any of the phases. Next is the phase where the contract is placed and initialized through a transaction. The importance of this phase is given by the consequences that the wrong treatment of some parameters may cause. The next phase is the active operation, the most extended phase. During this period, in addition to daily ones, such as reading the contract status, there may also be specific operations that, in turn, include other periods. There may be periods of voting, governance, staking or vesting. For example, a voting period requires a voting start time, a period during which voting is allowed, and a deadline. In theory, this active phase of a Smart contract can run indefinitely. If the contract faces an imminent emergency related to a vulnerability or malfunction, or the contract governors want to improve the contract logic, the contract can enter a new shutdown phase. From this phase, it can return to the active running phase, or there can be an upgradeability mechanism in which a new contract will be built to replace the old contract. The update will not be performed instead, in the case of web2, methods are needed to transfer storage and transfer funds.

6 The auditing process

Smart contracts must employ structured evaluation methodologies based on well-defined checklists. These checklists [16] should include key security questions, concise descriptions of potential risks, and corresponding remediation strategies. A structured approach ensures that critical security considerations—such as access control, privilege management, and inheritance handling—are thoroughly examined.

Table 1. The Access Control Audit

No.	Question	Description	Remedy
1	Are all actors and their interactions clearly defined?	Identifying actors and interactions is crucial for security.	List all actors and interactions, then create a diagram.
2	Are there functions without proper access controls?	Missing access controls can lead to unauthorized modifications or withdrawals.	Implement and test access controls like onlyOwner or role-based permissions.
3	Do certain addresses require whitelisting?	Whitelisting restricts interactions to trusted addresses for additional security.	Implement a whitelisting mechanism for sensitive operations.
4	Does the protocol allow privilege transfers?	Privilege transfers should follow a two-step process for added security.	Implement a two-step transfer mechanism requiring confirmation by the new owner.
5	What happens during privilege transfers?	The protocol should function correctly and predictably during transfers.	Verify protocol behavior during privilege transfers.
6	Does the contract inherit other contracts?	Inherited functions may expose unintended access if not properly overridden.	Review the accessibility of inherited external/public functions.
7	Does the contract use tx.origin for validation?	tx.origin can be exploited by malicious contracts; msg.sender is safer.	Use msg.sender instead of tx.origin for validation.

Table 1 shows a series of questions that can be asked to define the permissions required to access smart contracts [16]. The questions are accompanied by a description and remediation if the security implication applies. The checklist-based audit ensures the adoption of an adversarial programming approach and constantly prepares for the worst-case scenario by assuming that elements might malfunction. Two additional columns can be added to the table, one for concrete examples and one for the applicability of the current case. Using checklist approaches, developers can force themselves to anticipate potential issues, vulnerabilities, and challenges before they arise.

7 Conclusions and Future Work

Smart contracts are the security backbone of the Ethereum or compatible blockchain.

The need for auditing and testing encompasses three perspectives: the technical, business and ecosystem. Technically, it is well known that developers often make mistakes when writing code, but errors may spring from language-specific issues or the operating system. Thus, system behavior can be challenging to predict from the behavior of components alone. Business-wise, errors may impact user and customer perspectives and sales. Even if mistakes are not found initially, they may appear due to user interaction. Fixing errors is more expensive in post-production. Lastly, ecosystem-wise, testing is an intrinsic professional activity of blockchain protocols. The credibility of a protocol increases with the existence of a well-established testing suite. There is no complete list of all the vulnerabilities that may arise.

References

[1] Bitcoin: A Peer-to-Peer Electronic Cash System (2008) - Satoshi Nakamoto. Available: https://bitcoin.org/bitcoin.pdf

[2] V. Buterin, Ethereum Whitepaper: A Next-Generation Smart Contract and Decentralized Application Platform, 2014, Available at: https://ethereum.org/en/whitepaper/

[3] N. Szabo, Smart Contracts: Building Blocks for Digital Markets, 1996, Available at: http://www.truevaluemetrics.org/DBpdfs/BlockChain/Nick-Szabo-Smart-Contracts-B ui lding-Blocks-for-Digital-Markets-1996-14591.pdf

[4] Ojog, S.: An Overview of Security Issues in Smart Contracts on the Blockchain. In: Proceedings of 21st International Conference on Informatics in Economy (IE 2022), vol. SIST, no. 321, pp. 51–63. Springer, Heidelberg (2023).

[5] Solidity Security Patterns, Available at: https://github.com/fravoll/solidity-patterns/

[6] Osterwalder, A., & Pigneur, Y., *Business Model Generation: A Handbook for Visionaries, Game Changers, and Challengers*. John Wiley & Sons, 2010.

[7] Gassmann, O., Frankenberger, K., & Csik, M., *The Business Model Navigator: 55 Models That Will Revolutionise Your Business*. Pearson, 2014.

[8] X. (Brian) Wu, Z. Zou, D. Song, "Learn Ethereum: Build your own decentralized applications with Ethereum and smart contracts", Packt Publishing, 2023.

[9] G. Wood, Ethereum: A Secure Decentralized Generalized Transaction Ledger, Available at: https://ethereum.github.io/yellowpaper/paper.pdf

[10] ConsenSys, *Smart Contract Best Practices*, ConsenSys Security Guide. Available at: https://consensys.github.io/smart-contract-best-practices/.

[11] OpenZeppelin, *OpenZeppelin Smart Contract Libraries*, GitHub Repository. Available at: https://github.com/OpenZeppelin/openzeppelin-contracts.

[12] Cambridge Dictionary - https://dictionary.cambridge.org/dictionary/english/contract

[13] The "Rekt Test" – Trail of Bits https://blog.trailofbits.com/2023/08/14/can-you-pass-the-rekt-test/

[14] Understanding a Revolutionary and Flawed Grand Experiment in Blockchain: The DAO Attack - *Journal of Cases on Information Technology 21(1) 19-32.*

[15] Solidity Documentation, *Solidity: Ethereum's Smart Contract Language*, Official Documentation. Available at: https://docs.soliditylang.org/.

[16] Cyfrin, *Audit Checklist for Smart Contracts*, GitHub Repository. Available at: https://github.com/Cyfrin/audit-checklist/blob/main/checklist.json.

Silviu OJOG has graduated the "Gh. Asachi" Technical University, in 2013, in Iasi Romania, BSc in Applied Electronics. He graduated from the University of Bucharest, Romania, with an MSc in Software Engineering in 2016. He is currently enrolled as a PhD Student in Economic Informatics at Bucharest University of Economic Studies. He holds a certification in new venture leadership from the Massachusetts Institute of Technology, USA, following a study program in Brisbane, Australia.

Alina-Andrea MIRON graduated in Public Administration from the National School of Political and Administrative Studies in 2012 in Bucharest, Romania. She holds two master's degrees, one in Public Sector Management at the National School of Political and Administrative Studies in 2014 and the other in Online Marketing at the Academy of Economic Studies in Bucharest, Romania, in 2021. She is currently enrolled as a PhD student in Marketing, and her thesis is related to social responsibility communication. She is also an associate lecturer teaching PR.