Survey | Received 15 October 2025; Revised 12 December 2025; Accepted 4 January 2026; Published 20 January 2026
https://doi.org/10.55092/blockchain20260001

Security analysis of blockchain-based cryptocurrency

Zekai Liu, Xiaoqi Li*, Dongyang Lyu, Chunyi Zhang and Zhongwen Li

School of Cyberspace Security, Hainan University, Haikou, China

* Correspondence author; E-mail: csxqli@hainanu.edu.cn.

Highlights:

- Proposes a five-layer taxonomy of cryptocurrency security threats and attacks.
- Analyzes 165 real-world attack incidents and identifies 15 typical attack patterns.
- Provides a comparative evaluation of existing detection and defense mechanisms.
- Summarizes research gaps and future trends for strengthening blockchain security.

Abstract: Cryptocurrency is a novel exploration of a form of currency that proposes a decentralized electronic payment scheme based on blockchain technology and cryptographic theory. While blockchain has the security characteristics of being distributed and tamper-proof, increasing market demand has led to a rise in malicious transactions and attacks, thereby exposing cryptocurrency to vulnerabilities, privacy issues, and security threats. Particularly concerning are the emerging types of attacks and threats, which have made securing cryptocurrency increasingly urgent. This paper classifies existing cryptocurrency security threats and attacks into five fundamental categories based on the blockchain infrastructure, and it analyzes in detail the vulnerability principles exploited by each type of threat and attack. Furthermore, the paper examines the attackers' logic and methods and provides insights that enable easy reproduction of the vulnerabilities. We also summarize and evaluate existing detection and defense solutions, offering important references for ensuring cryptocurrency security. Finally, the paper discusses the future development trends of cryptocurrency.

Keywords: cryptocurrency; blockchain; security threats; smart cntract; attack detection and defense; security taxonomy; vulnerability classification

1. Introduction

In recent years, cryptocurrency flourish at an unprecedented rate [1]. Their decentralized and anonymous characteristics draw widespread global attention. However, with the rapid expansion of the cryptocurrency market, security threats and attacks also present an increasingly severe situation [2]. Although blockchain technology, as the underlying support of cryptocurrency, provides a certain degree of security, its inherent technical complexity and emerging characteristics make security vulnerabilities difficult to

completely avoid. Moreover, potential defects in the anonymity and traceability of cryptocurrency provide opportunities for malicious attackers.

Since the concept of cryptocurrency emerged, a series of major security incidents continuously occur, which not only cause severe damage to the blockchain ecosystem but also bring significant economic losses to individuals and institutions. For example, in 2016 The DAO attack exposes the enormous risks of smart contract vulnerabilities, resulting in the theft of over $50 million in cryptocurrency [3]; the 2017 Parity wallet smart contract vulnerability freezes over $30 million in Ether [4]; the 2018 Coincheck trading platform theft results in losses of over $500 million in NEM [5]; and the 2021 Poly Network cross-chain attack allows hackers to steal approximately $610 million in cryptocurrency [6]. These incidents fully demonstrate that cryptocurrency security issues become a critical factor restricting its healthy development.

In response to the ever-increasing number of cryptocurrency attacks, researchers conduct extensive and in-depth studies. For example, Chen *et al.* comprehensively review attack types, defense mechanisms, and privacy protection technologies for blockchain systems in their research, focusing on the balance between security and performance in blockchain-based application scenarios [7]. Guru *et al.* systematically analyze potential security threats in consensus mechanisms such as PoW and PoS and propose targeted defense strategies to enhance system resilience [8]. Platt and McBurney delve into Sybil attacks in blockchain consensus mechanisms, comparing the attack resistance capabilities of consensus protocols applying different algorithms and offering insights for more robust consensus mechanism design [9]. Akbar *et al.*, based on a comprehensive study of traditional consensus mechanisms, innovatively propose a distributed hybrid mechanism combining the advantages of PoW and PoS to resist double-spending attacks and fully demonstrate its feasibility for future applications [10]. Chaganti *et al.* systematically review the DoS attack patterns existing in the blockchain ecosystem, clarifying future research challenges by analyzing the limitations of existing defense strategies [11]. Furthermore, Madhushanie *et al.* systematically analyze the execution mechanisms and defense strategies for selfish mining attacks in blockchain systems, and discuss in detail their potential adverse effects [12].

These studies not only reveal the security threats and technical bottlenecks faced by blockchain technology in its development and application but also provide a solid theoretical foundation and research directions for building more secure and efficient blockchain systems, laying an important foundation for this paper's research.

Based on the aforementioned research work, we conduct a more comprehensive and in-depth study on cryptocurrency security issues. First, by collecting real-world attack events that have occurred in blockchain networks in recent years, we summarize the attack methods or security threats applied by attackers in each attack event and further extract the 11 most common and representative patterns. Next, based on the generally accepted blockchain architecture layers, we propose a classification scheme that divides these typical patterns into five categories, and we analyze their principles, execution mechanisms, and reproduction methods in detail. Subsequently, we systematically organize, evaluate, and comparatively analyze the existing detection and defense strategies for these typical patterns, analyzing their advantages and disadvantages. Finally, we discuss possible future research directions to address the complex evolution based on typical patterns, thereby strengthening the security of cryptocurrency.

The main contributions of this study are as follows:

- Comprehensive data collection: We have collected 165 real-world cryptocurrency attack cases and published them at https://doi.org/10.6084/m9.figshare.30857972.
- Complete classification system: Based on the generally accepted blockchain architecture layers, we construct an attack pattern classification system. This allows for a more systematic analysis of the principles, and execution mechanisms of different attack patterns.
- In-depth comparative analysis: We comprehensively summarize and compare the advantages and disadvantages of existing attack detection and defense strategies.

2. Background

2.1. Related content

2.1.1. Blockchain

Blockchain technology is a decentralized data storage and management model. It achieves data immutability, disintermediation, and distributed management by constructing a distributed ledger [13]. Its core advantage lies in the decentralization theory, which ensures data reliability and security through communication and data sharing among multiple nodes in the network, where all nodes have equal status, thus avoiding the risk of single points of failure [13]. To ensure data security, blockchain employs technologies such as hash algorithms and asymmetric encryption [14].

Bitcoin, the first cryptocurrency based on blockchain, is proposed by Satoshi Nakamoto in 2009 [15]. Its success promotes the development of the global cryptocurrency market. Bitcoin's blockchain technology verifies and records transactions through a distributed ledger, protects data and value, and maintains market stability. Each transaction is encrypted and linked into an immutable chain [16].

Figure 1 shows the six layers of blockchain architecture: including the data layer, network layer, and consensus layer as the foundation, and the incentive layer, contract layer, and application layer as extensions.

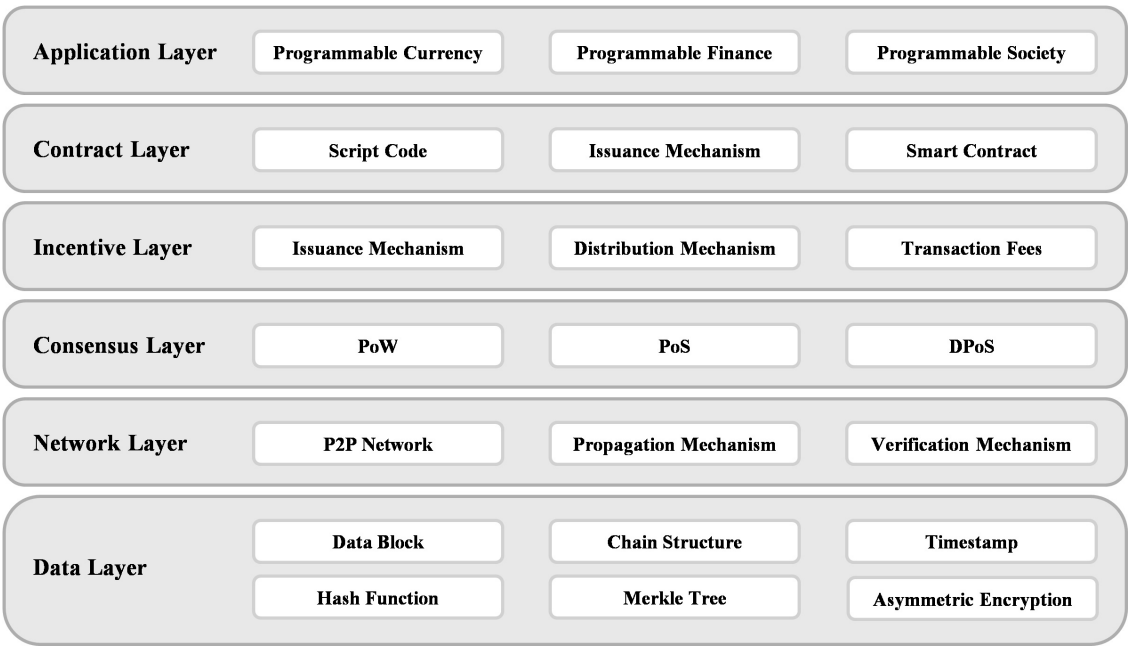

Figure 1. The six layers of blockchain network architecture.

2.1.2. Smart contracts

A smart contract is an automatically executing contract protocol based on blockchain technology [17]. It primarily runs on blockchain systems such as Ethereum. Its purpose is to ensure the security, fairness, and transparency of transactions between parties without the need for intermediaries. Essentially, a smart contract is a piece of computer code that consists of multiple code modules and can implement complex business logic. It has characteristics such as self-execution, immutability, and decentralization. It can be used to create cryptocurrency, manage digital assets, determine voting results, and so on.

2.1.3. Lightning Network

Lightning Network is a two-layer solution to Bitcoin's scalability problem, and has been developing since then [18]. It allows the network to conduct millions of transactions per second, thereby significantly increasing scalability.

For example, if Alice and Bob need to conduct multiple token transactions within the network, recording each transaction on the mainnet creates a significant waiting time until the previous transaction record completes before the next transaction can take place. The Lightning Network, however, proposes a bilateral payment channel that transfers transactions to a side network. If one party wants to stop the transactions, they simply upload the final transaction result to the mainnet.

Another advantage of the Lightning Network is that if Alice and Eric are strangers, Alice has an open payment channel with Bob, Bob has an open payment channel with Dale, and Dale has an open payment channel with Eric. When Alice tries to transact with Eric, she only needs to ensure that Bob and Dale have sufficient transaction funds in their accounts. Alice can then achieve the transaction with Eric through her friend Bob, and Bob's friend Dale.

2.1.4. Mining pool

As the total computing power of blockchain networks continues to increase, a single node can hardly compete for block-production rights alone. Therefore, mining pools emerge to aggregate the computing power of multiple nodes [19].

In a mining pool, nodes are divided into one administrator node and multiple miner nodes. Miner nodes perform mining using the pool administrator's public key (address), while the administrator distributes rewards according to each miner's contributed shares. This mechanism benefits both parties. Miner nodes gain higher and more stable income through cooperation and receive proportional rewards regardless of whether their individual shares solve a full block. Conversely, the shares submitted by miner nodes are useless without the administrator's private key; only the administrator can sign and publish new blocks to claim the block reward [20].

2.2. Preliminary preparation

In the preparation stage, we systematically collect and curate 165 real-world cryptocurrency attack incidents that occur up to September 2025 through Ethereum community forums, blockchain security reports, and other publicly verifiable sources, and construct a dataset based on these events.

As shown in Table 1, we categorize all incidents based on the layered architecture of blockchain systems. Using this classification, we select twelve representative attack patterns for in-depth analysis, chosen according to their occurrence frequency, economic impact, and influence across different protocols. These patterns encompass transaction malleability, eclipse attacks, transaction delay, Sybil attacks, double-spending, 51% attacks, selfish mining, bribery attacks, reentrancy, integer overflow/underflow, flash-loan attacks, and sandwich attacks. Additionally, we incorporate three supplementary attack patterns: collision attacks, block-withholding attacks, and resource-exhaustion vulnerabilities. Although these have rarely been observed in practice, they remain theoretically feasible and offer valuable insights into the overall security and robustness of blockchain systems.

Table 1. Summary of attack incidents at all levels.

Layer	# Incidents	Representative Threats
Data	8	Transaction Malleability Attack, Replay Attack.
Network	28	DDoS, DNS hijack, Domain Hijack.
Consensus	26	51% attack, Double Spend Attack.
Incentive	9	Selfish Mining Attack, Bribery Attack.
Contract	56	Reentrancy Attack, Overflow/Underflow, EVM Bugs.
Application	38	Flash loan Attack, Price manipulation Attack, Sandwich Attack.

3. Classification methods

Leveraging the characteristics of each blockchain layer, we propose a classification method for the security threats and attacks confronting cryptocurrency. This approach categorizes security threats and attacks, analyzing the characteristics, logic, methods, and principles of vulnerabilities exploited by various attack types. This classification is informed by research on vulnerability analysis, such as the work on Android applications by Li *et al.* [21].

3.1. Data layer

Although the data layer features immutability and decentralization, it still faces security threats such as data privacy theft and malicious data attacks [22]. Among these, data privacy theft may lead to the leakage of users' personal information, causing losses and risks to users. Meanwhile, malicious data attacks may result in property losses for multiple nodes within the blockchain network, leading to the paralysis of the entire blockchain system and security risks such as data tampering.

3.1.1. Collision attack

A collision attack primarily targets the message digest algorithms in the data layer [23]. Due to the security guarantees of message digest algorithms, they are hard-coded into the design of blockchain platforms like Ethereum and are regarded as immutable and indestructible building blocks. However, with advancements in cryptanalysis and the epochal increase in computational power, the effective security guarantees provided by message digest algorithms weaken over time.

For message digest algorithms, collisions inevitably exist. Protecting message digest algorithms from collision attacks proves crucial for enhancing the security of the data layer. For example, a victim holds a message m and computes its digest $h(m)$ using a specific message digest algorithm $h(x)$. An attacker only needs to try every possible message m_i, compute its digest $h(m_i)$ using the same message digest algorithm, and will certainly find an m_i such that $h(m_i) = h(m)$. From a purely mathematical perspective, in a uniform distribution, the probability of randomly selecting n integers within the numerical range $[1,d]$ and ensuring that at least two of them are identical is given by Equation (1).

$$P(d,n) = \begin{cases} 1 - \prod_{i=1}^{n-1} \left(1 - \frac{i}{d}\right), & n \leq d \\ 1, & n > d \end{cases} \tag{1}$$

Thus, in practice, such an attack requires enormous computational power, making collision attacks a type of "future" attack. Relying solely on computational power makes it difficult to execute a complete attack. Before launching a collision attack, attackers often conduct prolonged cryptanalysis, targeting the most likely datasets for collisions.

3.1.2. Transaction malleability attack

A transaction malleability attack, also known as a transaction ductility attack or plasticity attack, can be regarded as a variant of the typical double-spend attack in the data layer [24]. It should be noted that this article classifies the typical double-spending attack under the blockchain consensus layer.

Double Spend Attack: When multiple transaction messages point to the same content, we can determine that these transactions conflict, as only one of them may be valid. An attacker deliberately initiates two conflicting transactions to deceive a third party and gain sufficient benefits. Since blockchain nodes always regard the longest chain as the correct chain, when an attacker issues two transactions for the same funds simultaneously, the blockchain forks, and the blockchain defaults to considering the branch containing the first received transaction request as correct, building upon it. However, it also retains the other branch to prevent it from becoming the longest chain.

The attacker exploits this feature to launch a double-spend attack, with the attack process as Figure 2:

(1) The attacker issues a transaction request s, transferring a portion of their funds to a third party that accepts the transaction.
(2) The attacker issues another transaction request t, transferring the same funds to another account under their control, causing the blockchain to fork into branch S and branch T.
(3) The third party believes they have received the funds from the attacker and confirms transaction request s.
(4) The attacker immediately sells these funds, while the transaction request t remains unconfirmed.
(5) The attacker mines on branch T.
(6) The length of branch T surpasses that of branch S, making it the new main chain. Transactions on branch S revert to their previous state, causing the third party to return the funds claimed by transaction s to the attacker, achieving the goal of double-spending.

However, executing such a double-spend attack proves difficult, as it requires the attacker to control at least 51% of the computational power to ensure dominance in coin mining, allowing branch T to exceed branch S in length.

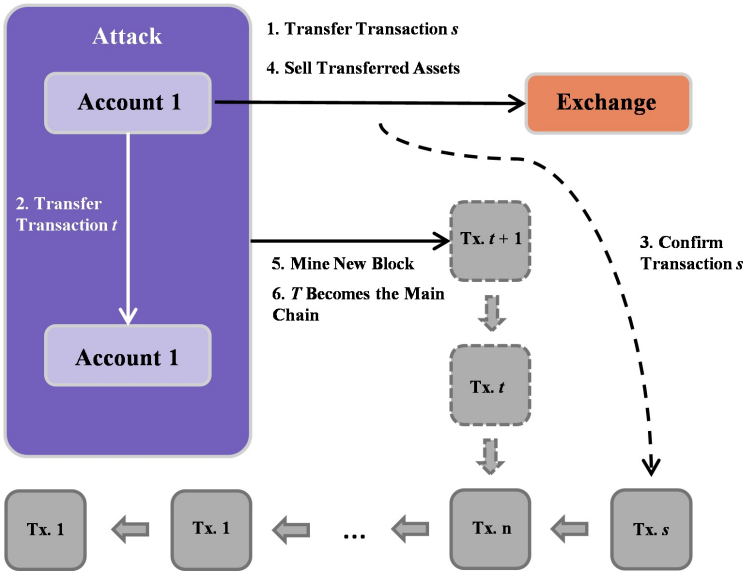

Figure 2. Double spend attack.

Transaction Malleability Attack: A transaction malleability attack originates from a vulnerability in the source code of the blockchain system, which allows attackers to alter a transaction's signature without changing its output or content. Unlike the typical double-spend attack, in a transaction malleability attack, the attacker initiates a transaction not as the sender of funds but as the recipient. The transaction malleability attack process unfolds as Figure 3:

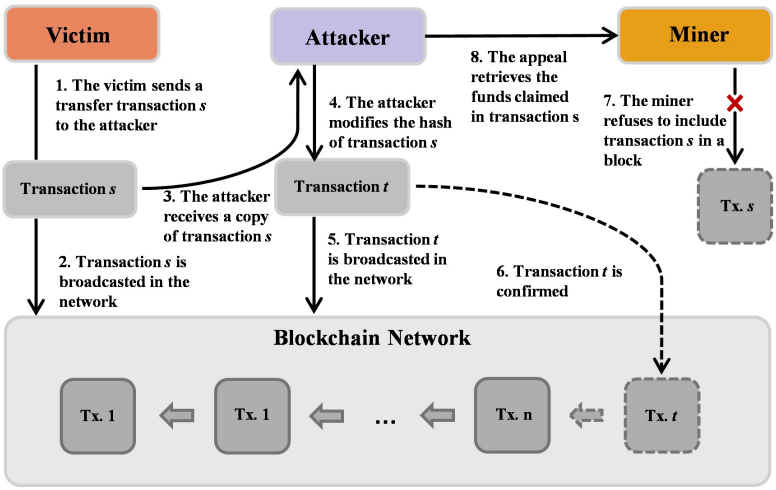

Figure 3. Transaction malleability attack.

(1) The attacker sends a withdrawal transaction request to the victim, prompting the victim to create a transaction s that transfers a portion of funds to the attacker's address. At this point, the transaction remains secure and trustworthy.

(2) The attacker waits for the transaction to broadcast across the blockchain network.

(3) The attacker receives a copy of the transaction.

(4) The attacker modifies the transaction information, altering its unique identifier to create transaction t.

(5) The attacker waits for transaction t to broadcast across the blockchain network.

(6) Both transaction s and transaction t can be confirmed. If transaction t gets confirmed, the transaction malleability attack succeeds.

(7) If transaction t is confirmed, miners consider transaction s a double spend and refuse to include it in a block.
(8) At this stage, the attack has not yet caused harm to the victim. The attacker needs to file a claim to obtain the funds claimed by transaction s. The victim only sees that transaction s remains unconfirmed, while the funds appear credited to the attacker's account.

3.2. Network layer

In the network layer, attackers typically launch malicious attacks targeting the P2P network. The forms of malicious attacks may include delay attacks, node forgery, and more [25]. The network layer also needs to handle issues such as routing selection, congestion control, and internetworking to ensure security and stability. Attackers may exploit these aspects to initiate attacks at the network layer, thereby undermining the security of the blockchain system.

3.2.1. Eclipse attack

An eclipse attack represents a relatively common attack method in the network layer of a blockchain [26]. Due to the distributed and decentralized nature of blockchain networks, all network nodes can only communicate with each other through adjacent nodes. An attacker modifies the target node's routing table and adds a sufficient number of malicious nodes around the target node to monopolize all connections with the target node. This isolates the target node from the blockchain network, preventing it from obtaining accurate information. Subsequently, the attacker can exploit the isolated node to perform actions such as forging transactions or executing double-spending attacks, ultimately achieving control over and disruption of the entire blockchain network.

The primary process of an eclipse attack unfolds as follows:

(1) The attacker uses P2P network attacks, such as flooding, to force the victim node to disconnect from its adjacent nodes.
(2) After realizing it cannot communicate with adjacent nodes, the victim may choose to restart, or the attacker may forcibly cause the victim node to restart.
(3) The attacker tampers with the victim node's routing table, ensuring that, upon restarting, the victim can only connect to malicious nodes controlled by the attacker.

3.2.2. Transaction delay attack

A transaction delay attack should be distinguished from a transaction malleability attack. A transaction malleability attack represents a variant of a double-spending attack that exploits vulnerabilities in the data layer, whereas a transaction delay attack involves a malicious occupation of network resources.

An attacker launching a transaction delay attack broadcasts a large number of transactions to the network in a short period but does not immediately record these transactions in a block. This causes the confirmation time of these transactions to be relatively delayed, consuming resources from the blockchain network.

The Lightning Network typically employs hash-time-lock technology to ensure the security of atomic asset swaps, with its security primarily relying on time locks and fund locks. A time lock stipulates that

each fund swap transaction must be completed within a specific time. However, an attacker can create a large number of fake transactions in a short time and deliberately delay their transmission, leading to network congestion and reduced network performance, thereby affecting the normal operation of the Lightning Network. Therefore, a transaction delay attack is regarded as a congestion attack in the blockchain environment. Although a transaction delay attack alone rarely brings substantial profits to the attacker, it is frequently used as a precursor to other attacks.

3.3. Consensus layer

Consensus layer attacks are divided into attacks targeting authorized consensus mechanisms and attacks targeting unauthorized consensus mechanisms [8]. Both types of attacks essentially involve the attacker using certain methods to prevent the entire network of nodes from reaching a correct consensus.

3.3.1. Sybil attack

Douceur first introduces the concept of a Sybil attack in P2P networks [27]. In an authorized consensus mechanism, each node exerts the same influence on the consensus process. Therefore, an attacker can influence the entire network of nodes in reaching a correct consensus by controlling multiple fake accounts or blockchain nodes, thereby achieving the intended attack behavior. The attack process unfolds as follows:

(1) A blockchain network operates in a decentralized and distributed manner, with nodes controlled by different identities distributed almost uniformly around the world.
(2) The attacker creates t fake accounts. Due to the authorized consensus mechanism, each fake account holds the same influence in the consensus process as a legitimate account.
(3) In the blockchain network, every decision is made through a collective vote by all accounts. Assuming the number of legitimate accounts in the network is s, the attacker only needs to ensure $t \geq s$ to force the entire network of nodes to reach an incorrect consensus.

By launching a Sybil attack, the attacker can control the voting or ranking process, thereby gaining an advantage over legitimate accounts in the competition and undermining the fairness of the blockchain network.

3.3.2. 51% attack

A 51% attack constitutes an attack targeting unauthorized consensus mechanisms. This attack method involves an attacker who, by controlling more than 50% of the entire network's computational power or "stake," firmly holds the bookkeeping rights in the blockchain network. This allows the attacker to control the entire network and prevent the storage and verification of other blocks. The attacker can then leverage this method to execute other attacks, such as double-spending, selfish mining, and more.

51% Attack in PoW Systems: In a Proof of Work (PoW) system, miners expend computational power to gain bookkeeping rights [28]. If an attacker or their group controls more than half of the computational power in the blockchain network, they can initiate a 51% attack. At this point, the attacker possesses the ability to arbitrarily modify transactions and can even use their computational advantage to generate a new branch chain, replacing the current main chain.

After successfully executing a 51% attack, the attacker can arbitrarily manipulate and modify information on the blockchain. Specifically, the attacker can exclude or alter the order of transaction actions, preventing some or all transactions from being confirmed, and obstructing the normal mining operations of some or all miners. The attacker can also use a 51% attack as a sub-attack to facilitate the following types of attacks:

- Double-Spending Attack: The attacker deliberately initiates two conflicting transactions and uses a 51% attack to reverse transactions, deceiving a third party to gain sufficient benefits.
- Transaction Malleability Attack: Through a 51% attack, the attacker can effectively ensure that a modified transaction T gets confirmed, thereby guaranteeing the success rate of a transaction malleability attack. However, a 51% attack is not necessary for a transaction malleability attack.
- Selfish Mining Attack: The attacker can leverage their computational advantage to prioritize gaining bookkeeping rights but does not immediately publish the new block. Instead, they continue mining on this block to ensure a sustained advantage in obtaining bookkeeping rewards.
- History-Revision Attack: A history-revision attack often occurs after a 51% attack due to the attacker's inability to continuously secure bookkeeping rights [29]. Honest miner nodes, originally victims, may turn into attackers to minimize their losses by launching a 51% attack on the new main chain, making their mined branch chain the main chain again. A history-revision attack often becomes a repetitive process, during which the roles of the victim and attacker continuously shift.

51% Attack in PoS Systems: In a Proof of Stake (PoS) system, the proportion of assets a node invests in each round of competition for bookkeeping rights is recorded as its "stake" [30]. The more assets invested, the greater the "stake", and the higher the advantage in competing for bookkeeping rights. Once the attacker's invested assets exceed half of the total assets competing for bookkeeping rights in a given round, they gain a significant advantage in securing the bookkeeping rights for that round, thereby achieving control. Unlike in a PoW system, the risks an attacker faces when executing a 51% attack in a PoS system far exceed the expected gains.

In a PoW system, an attacker launching a 51% attack incurs no equipment loss, whether the attack succeeds or fails. However, in a PoS system, even if the attack succeeds, the cryptocurrency based on PoS likely depreciates, and the attacker, holding the largest stake, suffers the greatest loss.

In a study by Lee and Kim, a short-selling attack applicable to PoS systems is proposed. Short-selling represents an operation mode in the financial asset domain [31]. Suppose Alice borrows an asset p from Bob and sells it at market price to obtain funds f. Under stable market conditions, Alice should repurchase asset p with funds f after a period and return it to Bob. However, if the market price of the asset changes during this period, making funds f significantly greater than the current market price of asset p, Alice profits from the difference.

In a PoS system, cryptocurrency exchanges exist, making "short-selling" theoretically possible:

(1) The attacker holds a quantity A of tokens, ensuring these tokens are sufficient to launch a 51% attack, *i.e.*, exceeding half of the total tokens invested in the competition for that round.
(2) The attacker borrows the entire quantity of tokens from an exchange, denoted as B.
(3) The attacker expends the full purchasing power of the borrowed tokens to transfer assets, which can be used to purchase tokens in a PoW system for the next "short-selling attack." Let the

purchasing power of the tokens at this time be i.

(4) The attacker uses malicious methods such as double-spending attacks or block-dropping attacks to degrade the performance of the current blockchain network, causing the tokens to depreciate.

(5) The attacker purchases a quantity B of tokens to return to the exchange, while the purchasing power of the tokens has now decreased to l.

(6) The total assets the attacker possesses after completing the "short-selling attack" can be calculated as $(i-l)B+lA$, with a final profit of $(i-l)(B-A)$.

3.4. Incentive layer

The incentive layer relies on an economic model to drive the system's stable operation. Incentive layer attacks and security threats are implemented by exploiting reward mechanisms, which can seriously jeopardize the consensus mechanism. We systematically analyze selfish mining attacks, bribery attacks, and block-withholding attacks, and discuss their similarities and differences.

3.4.1. Selfish mining attack

A selfish mining attack occurs when an attacker discovers a block but does not immediately publish it, instead continuing to mine on that block in secret [12]. Once the chain becomes sufficiently long, the attacker publishes all the mined blocks at once, replacing the existing main chain and establishing a new main chain. The previous main chain reverts to its prior state, rendering the blocks mined by honest miners—despite their significant computational resource expenditure—invalid. The selfish mining attack process unfolds as Figure 4:

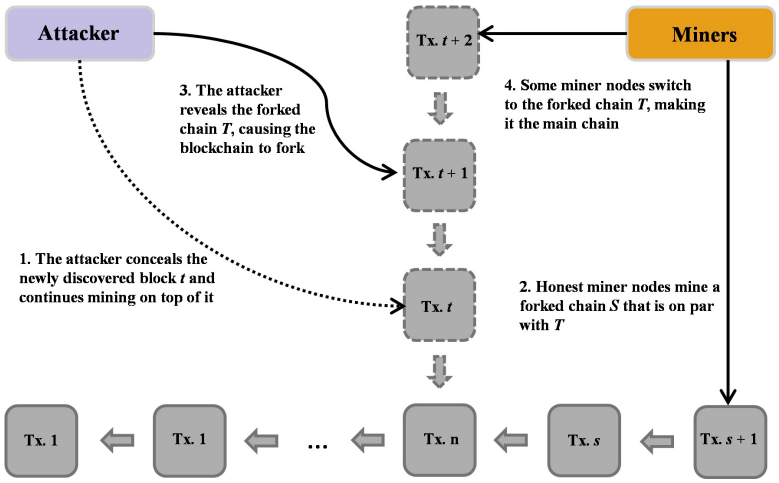

Figure 4. Selfish mining attack.

Assumption 1: All miner nodes, except the attacker, operate honestly.

Before the selfish mining attack begins, the attacker behaves identically to honest miner nodes. At this point, the blockchain network contains the longest main chain, and all nodes on the chain compete for the next bookkeeping right to earn rewards.

(1) The attacker discovers a new block first but chooses not to publish it immediately, instead concealing the block or sharing it within an internal network. The attacker continues mining on this block, effectively leading other nodes by one block.

Assumption 2: We consider the worst-case scenario for the attacker, where the blockchain network contains at most two competing branch chains: branch chain S, mined by honest nodes, and branch chain T, selfishly mined by the attacker. In this scenario, the attacker must compete against the combined computational power of all other nodes.

(2) Launching a selfish mining attack requires careful timing for publishing the branch chain. Since honest miner nodes mine simultaneously while the attacker mines new blocks, the attacker does not always have a 100% guarantee of maintaining an advantage. Two unfavorable situations may arise: a. the length of branch chain S, mined by honest nodes, equals the length of branch chain T, selfishly mined by the attacker; b. some blocks concealed by the attacker become invalid.

(3) If one of the aforementioned unfavorable situations occurs, the attacker immediately publishes the remaining blocks they hold, releasing branch chain T and causing a blockchain fork.

(4) At this point, the attacker hopes that honest miner nodes switch to mining on chain T, allowing chain T to gain a length advantage over chain S in the competition.

When honest miner nodes adopt chain T as the new main chain, the selfish mining attack is considered complete, and branch chain S becomes an invalid branch.

We can also consider an alternative scenario: when branch chain T becomes the main chain, some honest miner nodes may choose to continue mining along branch chain S (especially miners who have published blocks on branch chain S). In this case, these honest miner nodes transform into new attackers.

It's worth noting that selfish mining attacks are almost impossible to gain substantial profits in mining pools with low hashrate share. As the size of the mining pool decreases, its chances of winning in the competition drop significantly, and the expected return of selfish mining also decreases sharply. Therefore, small mining pools typically lack the hashrate foundation to carry out selfish mining attacks.

3.4.2. Bribery attack

The core idea of a bribery attack is that attackers induce some miners to deviate from honest strategies and perform specific behaviors that violate consensus rules by paying external rewards (such as censorship, double-signing, or participating in alternative branch mining designed by the attacker). When the computing power of the group of miners participating in the violation reaches a certain threshold, the security of the blockchain system can be compromised.

Based on the above ideas, Sarenche et al. proposed a bribery attack model similar to selfish mining but with zero risk to the attacker [32]. This attack uses external economic incentives to guide some mining pools to create orphan blocks, thus wasting effective computing power in the network and reducing mining difficulty in future cycles through difficulty adjustment mechanisms. Unlike selfish mining, this type of attack does not require the attacker to maintain the lead of their private chain, nor does it incur losses from failing to compete for block production; the attacker only needs to guide other miners to complete the offensive chain fork construction through economic incentives. The attack process can be summarized as follows:

(1) The attacker continuously monitors newly generated blocks on the blockchain and selects a block B mined by a small-hashrate mining pool as the target. The attacker's hashrate p_{attacker} and the target pool's hashrate p_{target} must satisfy $p_{\text{attacker}} > p_{\text{target}} + 2\delta$, where δ is a small bias ensuring that the attack incentive strictly exceeds the target pool's expected reward.

(2) The attacker deploys a bribery smart contract and embeds the hashes of the target block B and its parent. The contract specifies two types of rewards:
   - $L_{\text{main}} = p_{\text{target}} + \delta$ to incentivize miners to produce a competing block B'.
   - $L_{\text{bonus}} = \delta$ to reward the miner who first extends B' with a supporting block.

   A time-lock mechanism ensures that if no competing block is mined within the valid period, all rewards return automatically to the attacker, making the attack risk-free.

(3) For non-target mining pools, extending block B yields only the regular block reward, whereas mining the competing block B' yields both the standard block reward and the additional incentive L_{main}. Under rational economic behavior, miners prefer to mine B'.

(4) Once a mining pool successfully mines the competing block B', it claims L_{main}. The attacker then concentrates its own hashrate on mining the descendants of B', creating two parallel branches:
   - branch S, rooted at block B.
   - branch T, rooted at block B'.

(5) The additional reward L_{bonus} further incentivizes other mining pools to support branch T. During the branch competition, regardless of whether S or T eventually wins, at least one block becomes an orphan. Orphan blocks represent wasted non-attacker hashrate, which reduces the global difficulty through the adjustment mechanism and increases the attacker's block-winning probability in future mining epochs.

(6) The attacker's expected reward is R. Considering the incentive cost, the attack remains profitable as long as:

$$R = p_{\text{attacker}} \times \text{BlockReward} \tag{2}$$

$$R - (L_{\text{main}} + L_{\text{bonus}}) > 0 \tag{3}$$

   Because incentives are paid only when competing or supporting blocks are successfully mined, the attacker incurs no potential loss throughout the entire attack process.

3.4.3. Block-withholding attack

A block-withholding attack, also known as a block-hiding attack, shares strategic similarities with a selfish mining attack. Proposed by Rosenfeld in 2011, this attack rarely appears in practice [33]. A block withholding attack primarily manifests as an attacker never publishing the blocks they discover, thereby reducing the overall revenue of the mining pool.

Early research categorizes withholding attacks into two types: destroy and wait. In the former, the attacker does not profit from malicious behavior and is limited to sabotaging the mining pool through malicious means, which incurs high costs. The latter represents a more complex block-hiding attack similar to selfish mining. Nicolas *et al.* criticize this attack as impractical.

Nicolas *et al.* extend the destroy type attack and propose a scenario where the attacker can profit: the attackers split into two groups, where one group infiltrates other mining pools, claiming rewards while wasting the pool's computational resources, and the other group mines normally in a private pool, earning bookkeeping rewards. A block withholding attack significantly intensifies malicious competition between mining pools, disrupting the normal mining order. The block-withholding attack process unfolds as Figure 5:

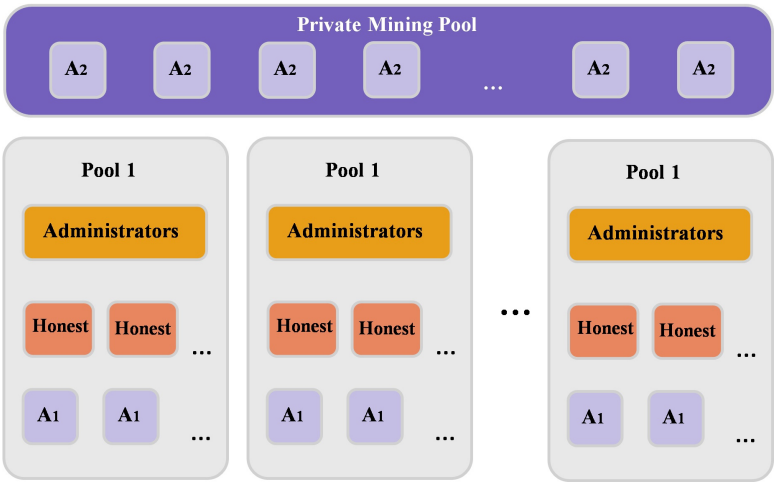

Figure 5. Block-withholding attack.

Assumption 1: All miners use the pool administrator's public key to mine in the same pool, the pool administrator distributes profits based on miners' contributions, and the administrator operates honestly.

Assumption 2: Miners do not frequently switch mining pools.

Assumption 3: The computational power of miners in the blockchain network is evenly distributed.

Assumption 4: The attacker's computational power accounts for $\alpha = 20\%$ of the total computational power of the blockchain network.

(1) The attackers split into two groups, $A1$ and $A2$, each with computational power of $\frac{\alpha}{2} = 10\%$. Group $A1$ infiltrates the mining pools of the blockchain network in a randomly distributed manner, constantly changing identities to reduce the likelihood of exposure. Including group $A1$, the mining pool collectively controls $1 - \frac{\alpha}{2} = 90\%$ of the total computational power of the blockchain network.

(2) Group $A2$ conducts normal mining activities in a private pool they deploy, which excludes honest miner nodes from joining.

(3) Since the pool administrator distributes rewards based on miners' contributions, if an honest miner node mines a hash share sufficient to generate a new block, it is immediately sent to the pool administrator to claim a reward. The pool administrator can then use this hash share and their private key to publish a new block, earning a substantial bookkeeping reward.

(4) However, if the hash share is mined by an attacker infiltrating the pool, the attacker conceals it, preventing the pool administrator from earning the bookkeeping reward. Similarly, since the attacker lacks the pool administrator's private key, they cannot publish the new block either.

(5) The pool administrator cannot determine whether miners engage in block withholding or identify which miner nodes are malicious. Over time, the pool administrator only notices that the pool's revenue falls far below expectations.

(6) The attackers profit in the following ways: (a) Group $A1$ mimics honest miner nodes, sharing rewards while wasting the computational power of the infiltrated pool; (b) Group $A2$ mines normally and earns bookkeeping rewards after publishing new blocks.

(7) Since the computational power of the pool infiltrated by group $A1$ accounts for $1 - \frac{\alpha}{2} = 90\%$ of the total computational power of the blockchain network, but group $A1$ conceals the shares

they mine, the average revenue of miner nodes in the infiltrated pool decreases by $\frac{1-\frac{\alpha}{2}}{1-\alpha} - 1 = \frac{90}{80} - 1 \approx 0.13$.

(8) Meanwhile, group $A2$, mining in the private pool, remains unaffected and gains higher profits: $\frac{1-\frac{\alpha}{2}}{1-\alpha} + 1 - 1 = \frac{\alpha}{4(1-\alpha)} = \frac{20}{4\times 80} \approx 0.0625$.

(9) Since only group $A2$ can earn bookkeeping rewards, the average profit of the attackers exceeds that of honest miner nodes by approximately 6% to 7%, which can be expressed with the Equation (4).

$$2 \times \frac{1-\frac{\alpha}{2}}{1-\alpha} + 2 - 1 = \frac{\alpha}{4(1-\alpha)} \tag{4}$$

(10) Through a block-withholding attack, the attackers achieve profits that exceed their investment, demonstrating that block-withholding attacks are theoretically profitable.

3.5. Contract layer

Based on the programming languages and runtime environments of smart contracts, security threats and attacks at the contract layer can be categorized into those targeting smart contracts and those targeting the contract virtual machine [34]. The former primarily stem from Solidity language characteristics and developers' non-standard programming practices, such as reentrancy and integer overflow vulnerabilities. The latter arise from unreasonable code applications and designs, including resource exhaustion vulnerabilities and similar issues.

3.5.1. Reentrancy attack

Reentrancy attacks are a prevalent threat in the Ethereum blockchain, tied to vulnerabilities in Solidity, its smart contract language. These attacks exploit flaws in programming logic, leading to substantial losses. They are classified into single-function attacks, where an attacker repeatedly invokes a vulnerable function via a callback loop, and cross-function attacks, where shared data between functions is exploited. Reentrancy vulnerabilities (Listing 1) typically arise in the `withdraw` function due to transferring funds before updating the balance, enabling attackers to repeatedly call it through an external callback [35].

```
1  contract Bank {
2      mapping(address => uint) public balances;
3
4      function withdraw(uint256 amount) public payable{
5          require(balances[msg.sender] >= amount);
6          (bool success, ) = msg.sender.call{value: amount}("");
7          require(success);
8          balances[msg.sender] -= amount;
9      }
10 }
```

Listing 1. Reentrancy vulnerability.

Below, we simulate the process of a single-function reentrancy attack in the Ethereum blockchain:

(1) The attacker identifies a Bank contract with a reentrancy vulnerability through program analysis.

(2) The attacker deposits a certain amount of tokens into the Bank, recorded as balance. These tokens suffice to call the Bank contract and initiate the attack.

(3) The attacker calls the Bank contract, attempting to withdraw an amount of tokens, denoted as amount, from the Bank.

(4) In the withdraw function of the Bank contract, the contract first checks if the attacker's balance in the Bank exceeds amount. If it does, the contract sends amount tokens to the attacker's account and updates the attacker's balance to balance-amount.

(5) The attacker calls the withdraw function through an attack function (Listing 2). When the withdraw function sends amount tokens to the attacker's account, it triggers the fallback function written by the attacker.

```
1      function attack(uint amount) public {
2          emit Withdraw(address(bank), amount);
3          bank.withdraw(amount ether);
4      }
5  }
```

Listing 2. Attack function in the attack contract.

(6) In the fallback function (Listing 3), the attacker also calls the withdraw function. Since the previous call to the withdraw function has not yet been completed, the attacker's recorded balance in the bank contract does not decrease. As a result, the Bank contract continues to send amount tokens to the attacker's account, triggering the fallback function again.

```
1      fallback() external payable {
2          // Check if the Bank contract has any balance available
       for withdrawal
3          if (address(bank).balance >= amount) {
4              bank.withdraw(amount);
5          }
6      }
```

Listing 3. Fallback function in the attack contract.

(7) Consequently, the Bank contract continuously sends tokens to the attacker's account until the Bank's balance is depleted or the gas limit is exhausted.

3.5.2. Integer overflow vulnerability

The principle of an integer overflow and underflow vulnerability operates as follows: In the Ethereum Virtual Machine (EVM), integer types consist of signed integers (int) and unsigned integers (uint). Generally, the uint type sees more frequent use. In the uint type, when an integer exceeds the upper

or lower limit of its designated bit size, the EVM defaults to a modulo operation. This causes the maximum value to wrap around to the minimum value when incremented by one, and the minimum value to wrap around to the maximum value when decremented by one. Integer overflow vulnerabilities can be categorized into integer overflow and integer underflow [36].

Integer Overflow (Listing 4): For example, with a `uint256` type variable, Solidity can only handle values less than or equal to the bit size limit. If we add 1 to the maximum value, it exceeds the storage range, causing the data to become 0. In real-world attacks, an attacker can exploit an integer overflow to reduce the cost of purchasing tokens:

```
contract Bank {
    mapping(address => uint256) public balances;
    uint256 public num;

    function buyTokens(uint256 amount) public payable {
        num = num + amount; // num is uint256 and can overflow
        balances[msg.sender] += amount;
    }
}
```

Listing 4. Integer overflow vulnerability.

(1) The attacker identifies a vulnerable `Bank` contract through program analysis.
(2) Since `num` is declared as a 256-bit unsigned integer variable, the attacker can construct a value that causes an integer overflow.
(3) The attacker spends a minimal fee to acquire a large number of tokens, which they can then sell for profit.

Integer Underflow (Listing 5): When using the `uint256` type, subtracting 1 from the minimum value of 0 causes an underflow, making the data wrap around to the maximum value of the current type. An attacker can exploit an integer underflow to drain the victim's balance at a low cost:

```
contract Bank {
    mapping(address => uint256) public balances;

    function withdraw(uint256 amount) public {
        balances[msg.sender] = balances[msg.sender] - amount;
    // Can underflow
        payable(msg.sender).transfer(amount);
    }
}
```

Listing 5. Integer underflow vulnerability.

(1) The attacker identifies a vulnerable `Bank` contract through program analysis.
(2) The attacker analyzes the contract vulnerability. Specifically, the vulnerability arises from the fact that `balances[msg.sender]-amount` is calculated as a `uint`, forcing the result to be greater than or equal to zero.
(3) The attacker constructs a very large `amount`, causing an integer underflow that drains the `Bank` balance.

3.5.3. Resource-exhaustion vulnerability

A resource-exhaustion vulnerability differs from reentrancy attacks and integer overflow vulnerabilities, as it represents a typical attack targeting the contract virtual machine. An attacker deploys malicious code on the EVM to maliciously consume system storage and computational resources. Therefore, the virtual machine must implement corresponding restriction mechanisms to prevent the abuse of system resources. For instance, Ethereum introduces the concept of Gas to address this issue [37].

- Gas: Serves as the basic unit to measure the computational resources consumed by a transaction.
- Gas Price: Represents the fee (in Ether) required for one unit of Gas.
- Gas Limit: Indicates the maximum amount of Gas a transaction sender is willing to pay for the execution of the transaction.

The introduction of Gas ensures that an attacker who intends to perform malicious operations on the Ethereum Virtual Machine must incur a high cost, which most attackers prefer to avoid. Consequently, resource-exhaustion vulnerability attacks gradually fade from attention and, in most cases, are only discussed as a typical example of attacks targeting the contract virtual machine.

3.6. Application layer

The application layer, as the carrier of blockchain technology, provides solutions for a wide range of business scenarios. Its functions can be broadly divided into two areas: mining and blockchain transactions. This article primarily focuses on security threats and attack methods in transaction scenarios, systematically analyzing flash loan attacks and sandwich attacks.

3.6.1. Flash loans attack

Flash loans provide traders with an instant, collateral-free borrowing mechanism whose security relies on the atomicity of blockchain transactions. The loan is valid strictly within a single transaction: as long as the user repays the principal L and the associated loan fee F_{loan} before the transaction ends, the loan is considered successful; otherwise, the entire transaction is reverted. While this mechanism improves system-wide liquidity, it also enables attackers to mobilize large amounts of capital within a short time window at virtually zero risk.

In a typical attack scenario, the attacker deploys a Flash Loan Receiver contract that implements the complete execution path required for the exploit, including the flash-loan request, callback handling, AMM manipulation, cross-protocol interactions, arbitrage extraction, and final repayment. The main attack process unfolds as Figure 6:

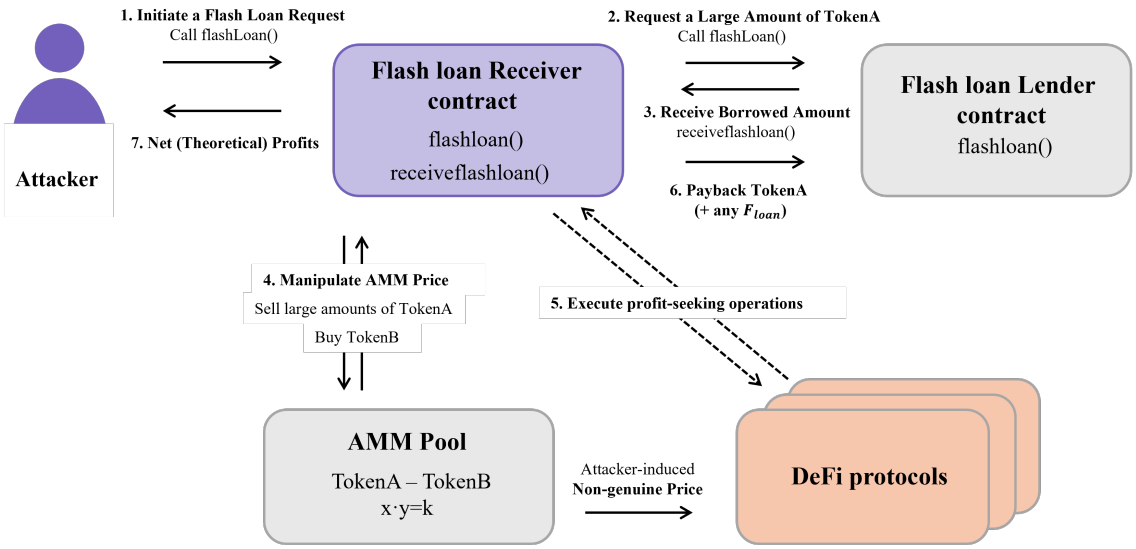

Figure 6. Flash loan attack.

(1) The attacker invokes the entry function of the Receiver contract, which sends a loan request to the flash-loan provider on behalf of the attacker.

(2) The Receiver contract calls the lender's `flashLoan()` function to borrow a large amount of TokenA. The total amount borrowed constitutes the flash-loan principal L.

(3) After transferring TokenA to the Receiver contract, the flash-loan provider immediately calls the predefined callback function `receiveFlashLoan()`. At this moment, the attacker controls a large amount of TokenA within a single atomic transaction and proceeds to execute subsequent manipulation and arbitrage steps.

(4) In flash-loan attacks centered on price manipulation, the attacker interacts with AMM-based decentralized exchanges (e.g., a TokenA–TokenB pool), which follow the constant-product market-making mechanism:

$$x \cdot y = k \tag{5}$$

$$\text{price}_{A/B} = \frac{y}{x} \tag{6}$$

where x and y denote the reserves of TokenA and TokenB, and k is a constant. Using the borrowed TokenA, the attacker initiates one or multiple large swaps to distort the price:

- selling a large amount of TokenA to acquire TokenB, sharply increasing x and reducing y;
- causing the price $\text{price}_{A/B} = y/x$ to decrease significantly (TokenA becomes undervalued, TokenB becomes overvalued), or to shift in the opposite direction depending on the target protocol's price-reading logic.

These swap operations generate AMM fees F_{swap}. Through this process, the attacker accumulates a substantial amount of TokenB while temporarily distorting the price environment.

(5) Since many DeFi protocols rely on AMM reserves or oracle-reported prices for asset valuation, they read the manipulated "false price" during the attack window. The attacker exploits this distorted state to obtain revenue R, including but not limited to:

- using the inflated TokenB as collateral in lending protocols to borrow excessive amounts of TokenA or other assets;

- triggering liquidation events with undervalued TokenA, allowing the attacker to liquidate positions at extremely low cost and claim liquidation bonuses;
- minting or redeeming tokens under extreme TokenA/TokenB ratios to extract value differences.

These cross-protocol interactions incur additional on-chain gas costs F_{gas}.

(6) Before the transaction ends, the Receiver contract repays the flash-loan principal L along with the loan fee F_{loan} to the lender contract.

(7) The attack yields positive profit if:

$$R - (L + F_{\text{loan}} + F_{\text{gas}} + F_{\text{swap}}) > 0 \tag{7}$$

Owing to the atomicity of flash loans, if any step fails to produce the expected revenue, the attacker can revert the entire transaction, leaving no on-chain trace and ensuring low operational cost and controlled risk.

3.6.2. Sandwich attack

A sandwich attack is a typical form of MEV arbitrage in which the attacker inserts manipulative transactions before and after the victim's transaction. By forcing the victim to trade within a squeezed price interval, the attacker captures price differences as profit. The attack process usually includes the following steps.

(1) The attacker uses an automated bot to monitor the pending transaction pool and filters AMM transactions with slippage tolerance, especially those specifying only a minimum output amount. Such transactions allow the execution price to fluctuate within a certain range, making them vulnerable to manipulation.

(2) Suppose the locked victim transaction $\text{Tx}_{\text{victim}}$ inputs x units of TokenA and expects to receive at least y units of TokenB, with a slippage tolerance ε. The attacker simulates the execution of the victim's transaction under the current pool state and computes the maximal extent to which the price can be pushed without triggering the victim's slippage protection.

(3) The attacker then broadcasts a high-gas front-running transaction Tx_{pre} that performs a large TokenA$\rightarrow$TokenB swap, reducing the TokenB reserve in the pool and increasing the price. Since miners order transactions by gas price, Tx_{pre} executes before the victim's transaction. As long as the resulting price increase remains within the victim's slippage tolerance, the victim's trade still succeeds but executes at a worse price.

(4) After the victim's transaction is processed, the pool price rises further. The attacker immediately submits a back-running transaction Tx_{post}, selling all previously purchased TokenB back into the pool. By arbitraging against the victim-induced price increase, the attacker secures a stable and predictable profit.

4. Resolution strategies

4.1. Data layer

Most attacks targeting the data layer aim to compromise blockchain data. Therefore, such attacks can be thwarted at their root by adopting advanced cryptographic tools and more secure cryptographic algorithms. Table 2 presents the defense and detection mechanisms for data layer attacks.

Table 2. Detection and defense strategies for data layer attacks.

Attacks	**Detection**			**Defense**		
	Method	Difficulty	Description	Method	Difficulty	Description
Collision Attack	N/A (Preventive focus)	–	–	Increase Message Digest Length	Medium	Significantly raises attack difficulty
				Introduce Timestamp Mechanism	Low	Limits attacker time window
Transaction Malleability Attack	Verify Transaction ID	Low	Simple, no extra tools required	Segregated Witness	High	Fundamentally eliminates signature-based malleability
	Third-party Monitoring	Medium	Real-time suspicious activity detection	Transaction ID Confirmation + Multi-signature	Medium	Additional protection without full protocol upgrade

4.1.1. Collision attack

Although collision attacks in blockchain networks may still require a long time to emerge on a large scale, preventive measures remain necessary to address potential future attack scenarios.

- Increase Message Digest Length: Since message digest algorithms output ciphertexts of the same length regardless of the plaintext's length, significantly enhancing security becomes possible by increasing the length of the output ciphertext, making attacks more difficult. However, this also increases the computational power required to execute the algorithm, posing another challenge for honest nodes.
- Introduce Timestamp Mechanism: Security is achieved by limiting the time available for an attacker to perform a collision attack.

4.1.2. Transaction malleability attack

Detection: To mitigate transaction malleability attacks, the risk of occurrence can be reduced by verifying the transaction ID before and after broadcasting or confirmation. Furthermore, third-party tools like blockchain.info and TradeBlock can be utilized to monitor the blockchain network for suspicious transaction activities, enabling the timely detection and prevention of transaction malleability attacks by identifying discrepancies in transaction IDs.

Defense: Core developer Wuille and others propose the Segregated Witness (SegWit) solution to address transaction malleability attacks [38]. The core idea of this solution involves separating the signature information of a block from the transaction data and storing it independently. The hash value generated for the block header depends entirely on the transaction content and the private key held by the

node initiating the transaction [39]. As a result, an attacker can no longer alter the transaction hash by modifying the signature information, rendering transaction malleability attacks infeasible.

Furthermore, methods such as transaction ID confirmation and multi-signature techniques can also prevent transaction malleability attacks.

4.2. Network layer

The blockchain network layer is primarily susceptible to attacks targeting its P2P network. Attackers frequently exploit the decentralized structure to compromise honest nodes, employing tactics such as routing hijacking, which can intercept or redirect network traffic, and malicious resource occupation, which can overwhelm nodes and hinder their ability to participate. These methods disrupt the blockchain network's operation by impairing communication and consensus processes. Unlike data layer attacks, network layer attacks tend to be more frequent, inflict greater immediate harm by potentially isolating nodes or segments of the network, and have a broader scope of impact on the network's overall availability and stability. Detection and defense mechanisms for these attacks are detailed in Table 3.

Table 3. Defense and detection strategies for network layer attacks.

Attacks	**Detection**			**Defense**		
	Method	Difficulty	Description	Method	Difficulty	Description
Eclipse Attack	Adjacency relationship + block info verification	Medium	High accuracy, low overhead, decentralized verification	Enhance Node Security (firewalls, IDS)	Low	Fundamental protection, timely block malicious traffic
	Transaction + connectivity analysis with ML	High	Good scalability and flexibility	Increase Number of Nodes	Medium	Reduces influence of single malicious node
				Connect Only with Trusted Nodes	High	Significantly reduces risk, but needs verification mechanism
Transaction Delay Attack	N/A (Preventive focus)	–	–	Compress Attack Manipulation Time	Medium	Reduces attacker time window
				Optimizing Gas + Imposing limits on ransaction amounts	Medium	Makes mass transaction creation uneconomical
				Optimize Transaction Queue	High	Prevents predictable delay of transactions

4.2.1. Eclipse attack

Detection: Alangot *et al.* propose a method that leverages the adjacency relationships between nodes and the block information of adjacent nodes to detect potential attackers. This method uses the decentralized mechanism of the blockchain to verify detection results, thereby improving the accuracy and reliability of detection [40].

Xu *et al.* design a detector that analyzes transaction data and node connectivity in the Ethereum network. This detector feeds the collected information into a machine-learning model for training and optimization, enabling the identification of potential eclipse attack behaviors [41].

These two studies provide different approaches and methods for detecting eclipse attacks. From the perspective of experimental data, Alangot *et al.*'s detection method performs better in terms of efficiency and accuracy, with lower computational and communication overhead compared to other similar methods.

Meanwhile, the detector designed by Xu *et al.* also achieves promising results in experiments and offers scalability and flexibility. However, its experimental sample size remains relatively small, potentially requiring further validation and refinement.

Defense: To defend against eclipse attacks, strategies such as increasing the number of nodes in the network, enhancing node security, and connecting only with trusted nodes prove effective.

- Enhance Node Security: Providing security support for nodes serves as the most fundamental measure to prevent eclipse attacks. Installing security devices such as firewalls and intrusion detection systems on nodes enables timely detection and blocking of malicious traffic.
- Increase the Number of Network Nodes: When malicious nodes launch an eclipse attack, they typically control one or more nodes in the network to manipulate message traffic. Increasing the number of nodes reduces the influence of a single node, making it more difficult for attackers to control the network.
- Connect Only with Trusted Nodes: Each node can significantly reduce the risk of connecting to malicious nodes by ensuring that its adjacent nodes are trustworthy, thereby enhancing the overall security of the network. However, this method poses challenges for newly joined nodes. Therefore, a verification mechanism must also be introduced to identify node characteristics, allowing only verified nodes to join the network.

4.2.2. Transaction delay attack

Currently, several solutions exist to address transaction delay attacks:

- Compress Attack Manipulation Time: Reducing the time threshold for transactions in a pending state and optimizing the transaction processing performance of the blockchain prevent attackers from gaining sufficient time to execute the attack.
- Increase the Cost of Launching an Attack: Increasing the cost of launching an attack can be achieved by optimizing transaction fees (Gas) and imposing limits on transaction amounts, making it difficult for attackers to create a large number of transactions at a low cost in a short period.
- Optimize Transaction Processing Queue: Randomizing the processing of pending transactions prevents attackers from determining the next transaction to be processed. Alternatively, prioritizing the earliest submitted transactions can prevent attackers from delaying the processing of specific transactions.

However, these solutions are not effective and may even negatively impact the blockchain network. Therefore, when selecting a defense strategy for transaction delay attacks, it becomes essential to carefully consider various factors, including the current network conditions and specific application scenarios.

4.3. Consensus layer

Attacks targeting the consensus layer can be mitigated by optimizing the consensus mechanism. However, since the behavior of attackers before an attack shows almost no difference from that of honest nodes, detecting such attacks using conventional methods proves challenging. To address consensus layer attacks, a combination of techniques from multiple dimensions—such as data monitoring, behavior analysis, and network traffic monitoring—must be employed to maximize the prevention of their occurrence. Table 4 presents the detection and defense mechanisms for consensus layer attacks.

Table 4. Defense strategies for consensus layer attacks.

Attacks	Defense		
	Method	Difficulty	Description
Sybil Attack	Increase the Number of Network Nodes	Medium	Reduces proportion of fake nodes; needs to combine with other strategies
	Introduce Identity Verification Mechanisms	Medium	Significantly hinders creation of fake accounts; may increase centralization risk
	Increase Attack Cost	High	Makes mass fake node creation economically infeasible
51% Attack	Enhance Advantage of Honest Nodes	High	Disperses computational power/stake, strengthens decentralization
	Introduce Multiple Consensus Mechanisms	High	Reduces risk of single mechanism dominance, enhances overall security
	Node Management + Monitoring	Medium	Early detection of anomalies, additional security layer

4.3.1. Sybil attack

The primary strategy for defending against a Sybil attack involves improving the security mechanisms of the blockchain network and enhancing the security and accuracy of the consensus mechanism. Several methods exist to mitigate Sybil attacks, but relying solely on a single method often fails to provide effective defense and may even introduce risks to the overall network [9].

- Increase the Number of Network Nodes: Attackers executing a Sybil attack typically need to control a large number of fake nodes. Increasing the total number of nodes in the blockchain network reduces the likelihood of a Sybil attack. However, adding a large number of nodes in a short period may impact the blockchain network, so this approach should be combined with other strategies.
- Introduce Identity Verification Mechanisms: Implementing multi-factor identity verification for each newly joined node makes it difficult for attackers to create numerous accounts. However, identity verification mechanisms may increase the centralization of the blockchain.
- Increase the Cost of Launching an Attack: Adopting a hybrid consensus mechanism can deter Sybil attacks. For example, in a PoW mechanism, requiring nodes to expend real computational power to compete for bookkeeping rights makes the attack costly. Additionally, gradually increasing the cost of creating accounts can meet the needs of some honest nodes for multiple accounts while preventing attackers from controlling a large number of accounts.

4.3.2. 51% attack

A 51% attack occurs when an attacker controls more than 50% of the computational power or stake in the entire blockchain network, allowing them to tamper with and control transaction information [30]. Defending against a 51% attack often requires a combination of technical and managerial measures to ensure the network's decentralization, security, and reliability. The following two strategies prove effective:

- Enhance the Advantage of Honest Nodes: In a PoW system, increasing the computational power of honest miner nodes by adopting more advanced equipment can boost the overall computational power of the blockchain network. Encouraging more miner nodes to join the blockchain network also disperses computational power, enhancing the network's decentralization and reducing the likelihood of an attacker dominating the network's computational power. In a PoS system, encouraging honest nodes to participate in the competition for bookkeeping rights can reduce the stake advantage of malicious nodes.
- Introduce Multiple Consensus Mechanisms: A 51% attack primarily occurs in blockchain networks that adopt a single consensus mechanism. Introducing different consensus mechanisms such as PoS or DPoS, and using them in combination with PoW, reduces the likelihood of an attacker controlling the computational power or stake, thereby ensuring the network's decentralization and security.

Additionally, strengthening the management and monitoring of network nodes proves essential to detect abnormal attacker behavior as early as possible. Employing techniques such as multi-signature for nodes and sharding can also enhance the network's resistance to attacks.

4.4. Incentive layer

Incentive layer attacks typically target these economic mechanisms or protocol rules, attempting to gain undue benefits or disrupt consensus. Table 5 describes the detection and defense mechanisms for incentive layer attacks.

Table 5. Defense and detection strategies for incentive layer attacks.

Attacks	**Detection**			**Defense**		
	Method	Difficulty	Description	Method	Difficulty	Description
Selfish Mining Attack	Dataset-based model	Medium	Accurate identification from historical cases	Optimize transaction fees + adjust incentives	Medium	Alters reward structure, deters attacks economically
	Miner behavior + network pattern analysis	High	Systematic and comprehensive	Select honest nodes + incentivize honesty	Medium	Strengthens network honesty
	Game-theoretic revenue analysis	High	Infers attacks via revenue deviation	Hybrid consensus	High	Reduces reliance on computational power
Bribery Attack	Real-time on-chain + game-theoretic analysis	High	Identifies deviation from Nash equilibrium, dynamic response	BribeGuard protocol	High	Eliminates incentives at game-theoretic level
				Customized validator election + confirmation	High	Prevents reversal by bribed validators
Block-Withholding Attack	Network traffic + block time monitoring	Medium	Leverages correlation of production/difficulty times	Forfeiture mechanism	Low	Simple, direct penalty, minimal network changes
	Hybrid statistical + cross-check	Medium	High accuracy via multi-node verification	Robust consensus in pools	High	Advanced protection, optimizes reward distribution
				Distribute hash power + join fair pools	Low	Reduces risk without protocol changes

4.4.1. Selfish mining attack

Detection: Detecting a selfish mining attack requires considering the actual conditions of the blockchain network. The analysis must encompass multiple aspects, including network topology, node transaction behavior, and mining information, to accurately identify attack behavior.

Wang *et al.* propose a detection scheme based on datasets: by collecting cases of past selfish mining attacks and using analytical techniques to build a detection model, they achieve accurate identification of selfish mining behavior in blockchain networks [42]. Madhushanie *et al.*, in their study, provide a

systematic review of detection approaches, highlighting methods that analyze miner behavior and network patterns to infer selfish mining activities [12]. Additionally, establishing a game-theoretic model between attackers and honest miner nodes allows for the detection of each miner node's revenue, thereby inferring the presence of attack behavior.

Defense: Optimizing transaction fees can reduce the occurrence of selfish mining attacks. Research indicates that adjusting economic incentives may deter selfish mining by altering the reward structure, potentially stabilizing blockchain networks [12]. Other researchers also affirm this approach in their articles.

Madhushanie *et al.* propose a more systematic scheme to defend against selfish mining attacks, detailed as follows [43]:

- Select Appropriate Honest Nodes: By randomly selecting a sufficient number of active and honest nodes and incentivizing their honest behavior in the blockchain network, these nodes help defend against selfish mining attacks.
- Introduce Multiple Consensus Mechanisms: Adopting a hybrid consensus mechanism effectively mitigates selfish mining attacks. For example, the Ethereum blockchain introduces the PoS consensus mechanism to reduce the PoW mechanism's excessive reliance on computational power.

Additionally, schemes such as decentralization, multi-signature mechanisms, and identity verification techniques can also reduce the likelihood of selfish mining attacks.

4.4.2. Bribery attack

Detection: Sarenche *et al.* proposed a comprehensive detection and mitigation mechanism that combines real-time analysis of on-chain activity with game-theoretic evaluation of validator behavior to combat bribery attacks [32]. By monitoring and analyzing validators' voting and block proposal patterns, the mechanism identifies whether their behavior deviates from a rational (honest) Nash equilibrium. Once a risk is detected, the system dynamically adjusts the block reward and penalty ratios to increase the cost of attacks in real time, thereby mitigating the attack.

Defense: Karakostas *et al.* designed a protocol called BribeGuard to defend against bribery attacks in PoS blockchains by introducing novel reward distribution and penalty mechanisms [44]. This protocol eliminates the incentive for attacks at a game-theoretic level by adjusting incentive parameters to make the on-chain cost and risk of punishment for bribing validators economically infeasible.

Awathare *et al.* designed a decentralized exchange (DEX) model that mitigates bribery attacks by integrating a customized validator election and transaction confirmation mechanism [45]. This mechanism aims to ensure that the final confirmation of transactions cannot be easily reversed by validators bribed by a small number of attackers with substantial stake or computing power.

4.4.3. Block-withholding attack

Detection: A block-withholding attack shares certain behavioral similarities with a selfish mining attack, and both can be detected by analyzing network traffic. Furthermore, since block production time and Bitcoin difficulty adjustment time are typically correlated under normal conditions, monitoring these times can also help determine the presence of malicious behavior in the network. Li *et al.* propose a hybrid statistical test and cross-check method to detect block-withholding attacks, leveraging statistical

analysis and multi-node verification for high accuracy [46].

Defense: Chen *et al.* propose a method called the "forfeiture mechanism", which penalizes miners in a pool who maliciously withhold blocks [33]. If a miner is found engaging in malicious behavior, all miners in that pool face penalties. This method fosters closer cooperation between honest miner nodes and the pool administrator, as both aim to identify the attacker as quickly as possible.

Mihaljević *et al.* propose another defense strategy: deploying a robust consensus protocol into mining pools to counter block-withholding attacks. This approach enhances pool security by optimizing reward distribution and monitoring miner behavior, ensuring the pool's revenue remains unaffected [20].

Both defense methods exhibit a degree of innovation but differ in their approaches. Chen *et al.*'s scheme proves simpler and more direct, causing minimal changes to the blockchain network, while Mihaljević et al.'s scheme appears more advanced and complex.

Additionally, miner nodes can reduce the risk of block-withholding attacks by reasonably distributing computational power and joining fair mining pools.

4.5. Contract layer

Attacks on the contract layer typically exploit vulnerabilities in smart contract code or the virtual machine, bypassing the normal logic of the contract to execute an attack. This places higher demands on the security of both the contract code and the contract virtual machine.

Techniques such as code auditing and secure coding practices enable the timely discovery and remediation of vulnerabilities in contracts. Currently, various smart contract detection solutions exist, including static analysis tools and dynamic testing frameworks, which effectively enhance the security and robustness of contracts.

Static Analysis: This involves analyzing the smart contract code or bytecode line by line to identify potential security vulnerabilities.

Dynamic Analysis: Dynamic analysis primarily involves detecting smart contracts by simulating their execution, and analyzing potential anomalies that may arise during runtime. Commonly used tools like Echidna and Manticore simulate the execution of smart contracts to identify vulnerabilities within them.

Table 6 presents the detection and defense mechanisms for contract layer attacks.

Table 6. Defense and detection strategies for contract layer attacks.

Attacks	**Detection**			**Defense**		
	Method	Difficulty	Description	Method	Difficulty	Description
Reentrancy Attack	State change + contract info	Medium	Low false positives/negatives, uses security patterns	Use stricter calls	Low	Simple code-level fix, prevents recursive calls effectively
	Static AST + data flow analysis	Medium	Fully static, no execution needed	Limit same-address transaction frequency	Medium	Stops rapid repeated attacks
	Data flow analysis for state variables	Medium	Accurate tracking of state values			
Integer Overflow Vulnerability	Static analysis + vulnerability dataset	Low	Fast matching, dataset continuously updated	Use higher-bit-width integers	Low	Simple type change, no extra libs
	Dynamic analysis + Z3 solver	High	Precise detection via simulation and constraint solving	Use SafeMath library	Low	Proven secure arithmetic operations
	Enable built-in overflow checks (Solidity 0.8.0)	Low	Native compiler protection, no additional code needed			

4.5.1. Reentrancy attack

Detection: In the detector designed by Alkhalifah *et al.*, changes in the state before and after transaction execution, combined with information such as contract addresses and function names, help determine the presence of a reentrancy attack. Predefined security patterns prevent false negatives, while two independent detection mechanisms avoid false positives [35].

Chinen *et al.* propose a fully static analysis-based detection scheme that evaluates the structure and behavior of the code by fitting an Abstract Syntax Tree (AST) and employing data flow analysis techniques. This approach effectively detects reentrancy attacks in blockchain networks [47].

Similar to Chinen *et al.*'s scheme, Yu *et al.* develop a detection model that uses data flow analysis to identify user-defined attributes and accurately maintain the values of state variables, thereby uncovering reentrancy vulnerabilities in smart contracts [48].

Existing detection schemes primarily rely on data flow analysis, static analysis, and dynamic analysis. However, these methods exhibit limitations with external calls and fail to achieve the desired effectiveness in certain application scenarios, making this a key focus for future contract detection schemes.

Defense: In addition to performing security audits on contract code and employing secure coding techniques, the following methods can be adopted during smart contract deployment to prevent reentrancy attacks:

- Use Stricter Programming Statements: In the Solidity language, contract function calls primarily include `call()`, `transfer()`, and `send()`. Only the `call()` function triggers the `fallback` function of an external contract. Therefore, avoiding the use of the `call()` function can prevent reentrancy attacks. Adding the "non-reentrant" keyword to a function ensures that the function completes execution without being repeatedly called. Additionally, setting `lock(true)` and `lock(false)` can also prevent a function from being called multiple times.
- Limit the Number of Transactions with the Same Address: When the number of transactions with the same account reaches a threshold within a short period, transactions with that account are immediately halted. This functionality, which can be implemented by checking the account address, effectively prevents attackers from performing reentrancy attacks through rapid transactions.

4.5.2. Integer overflow vulnerability

Detection: Ayoade *et al.* employ static analysis techniques to detect vulnerabilities in contract code and create a vulnerability dataset that includes details such as the location of vulnerable statements and operand types [49]. By matching arithmetic operations in parts of the contract code with those in the dataset, vulnerabilities are flagged, and the dataset is updated upon detection.

Additionally, Wang *et al.* design a dynamic analysis-based detection model for unsigned integer overflow vulnerabilities. This model simulates the execution of contract code on the contract virtual machine, checks and constrains the results of unsigned integer operations (such as binary operations, relational operations, and shift operations), and uses the Z3 solver to perform constraint solving, identifying contracts with vulnerabilities [50].

Defense: Using higher-bit-width integer types, secure mathematical libraries such as SafeMath, and enabling overflow checks in Solidity 0.8.0 or later can effectively mitigate integer overflow vulnerabilities in smart contracts, significantly reducing the risk of exploitation by attackers.

4.6. Application layer

Table 7 presents the detection and defense mechanisms for application layer attacks.

Table 7. Defense and detection strategies for application layer attacks.

Attacks	Detection			Defense		
	Method	Difficulty	Description	Method	Difficulty	Description
Flash Loans Attack	Static taint analysis + expanded taint sources + call graph	Medium	High accuracy for price manipulation detection	Runtime mitigation by disrupting atomicity in mempool	High	Actively breaks attack atomicity in real-time
	Runtime analysis of function signatures	High	Real-time detection of non-price-based attacks	Simulation framework for countermeasures	Medium	Quantifies effectiveness in various market conditions
Sandwich Attack	Real-time transaction stream analysis	Medium	Real-time identification within blocks	Detection + sanctioning of malicious accounts	Medium	Directly penalizes attackers
	Gas features + SVM/GAT	High	Accurate modeling of Gas behavior and account patterns			

4.6.1. Flash loans attack

Detection: Flash Loan Attack Detection focuses on analyzing the complex data flow, function calls, and anomalous operation patterns within individual atomic transactions.

Wu *et al.* introduced the FlashDeFier framework, a static taint analysis method for flash loan vulnerabilities [51]. This framework significantly improves the detection accuracy of price manipulation vulnerabilities by expanding the scope of taint sources and taint slots and constructing a detailed inter-contract call graph.

Alhaidari *et al.* proposed FlashGuard, a runtime detection method for non-price-based flash loan attacks [52]. This method identifies attack patterns in real-time by analyzing smart contract function signatures.

Defense: Defense strategies primarily focus on breaking the atomicity of attacking transactions or eliminating vulnerabilities exploited by attackers.

Alhaidari *et al.*'s FlashGuard not only detects attacks but also mitigates them [52]. It counterattacks by exploiting a brief window of time when transactions are visible but not yet confirmed in the mempool, thus disrupting the atomicity of attacking transactions.

Werapun *et al.* proposed the Flash Loan Attack Analysis (FAA) framework to quantify and simulate the effectiveness of existing preventative measures (such as Fair Reserves) in different market environments, particularly high-volatility markets, thereby assisting security professionals in adopting countermeasures more effectively [53].

4.6.2. Sandwich attack

Sandwich Attacks are preemptive attacks that exploit transaction transparency and order manipulation. Currently, the primary method for preventing sandwich attacks is by detecting and sanctioning malicious accounts that conduct them.

Li *et al.* proposed a real-time sandwich attack detection system based on the Geth client [54]. This system analyzes transaction data streams to identify anomalous patterns and potential attack behaviors within blocks in real time, and integrates this information into the Go-Ethereum client for real-time analysis.

Liu *et al.* proposed the cascaded classification framework GasTrace for detecting sandwich attack malicious accounts in Ethereum [37]. This method places particular emphasis on the analysis and modeling of Gas features. GasTrace uses Support Vector Machines (SVMs) in the initial stage to generate predicted probabilities for accounts, and then utilizes Graph Attention Networks (GATs) in the second stage to capture behavioral features for cascaded classification.

5. Conclusion

This paper preliminarily explores the security threats and attacks faced by blockchain-based cryptocurrencies. Based on the characteristics of blockchain layers, these threats and attacks are categorized into five types: data layer attacks, network layer attacks, consensus layer attacks, contract layer attacks, and application layer attacks. This classification facilitates a systematic study of attack characteristics, logic, and processes. Comparative analysis reveals that even within the same layer, different types of attacks exhibit significant differences in principles and implementation methods. For instance, both collision attacks and transaction malleability attacks belong to the data layer, but the former focuses on stealing private data, while the latter targets the blockchain network through malicious data. Furthermore, the study identifies similarities and close relationships between attacks at different layers. For example, a transaction malleability attack represents a variant of a double-spending attack, while a selfish mining attack often requires a 51% attack as a prerequisite. This paper also conducts a systematic analysis and comparison of existing attack detection and defense methods, evaluating them from multiple perspectives, including security, implementation complexity, and applicability to large-scale networks.

However, the limitations of this study must be acknowledged, particularly in terms of the scope of attack types covered and the depth of the research, both of which require further expansion. Therefore, future research should focus on the following areas to enhance the breadth, depth, and rigor of the study:

Expand the Scope of Attack Types: Security threats and attacks targeting cryptocurrency exhibit complexity and diversity. This paper only examines relatively typical attack cases, yet attack methods in blockchain networks often lack universality. Future research will aim to expand the dataset of collected attack cases, enabling a broader assessment of potential attack risks.

Investigate Composite Attacks and Future Threats: As blockchain networks continue to evolve, simple attack methods increasingly struggle to pose significant threats to the system. Attackers now focus more on discovering new vulnerabilities or combining multiple attack techniques. This paper currently lacks sufficient research in this area. Future studies will delve deeper into the characteristics of composite attacks, building on existing cases to explore potential security threats and anticipate future attack methods.

Develop More Robust Security Detection Models: Current security detection models mainly target single attack types and rely on static or dynamic analysis. Although they perform well in simple scenarios, they often fail to provide accurate feedback in complex, interactive environments, allowing attackers to bypass defenses and execute attacks successfully. Future research therefore, focuses on extending and optimizing detection models to improve their adaptability and robustness in complex settings.

Through these efforts, this study provides more comprehensive theoretical support and practical guidance for strengthening blockchain network security.

Acknowledgments

We acknowledge the use of generative AI tools to assist with the translation and linguistic refinement of the manuscript.

Authors' contribution

Zekai Liu: conceptualization, methodology, visualization and writing—original draft. Xiaoqi Li: conceptualization, supervision and writing—review & editing. Dongyang Lyu: data curation, investigation and formal analysis. Chunyi Zhang: data curation, validation and formal analysis; Zhongwen Li: data curation, validation and formal analysis. All authors have read and agreed to the published version of the manuscript.

Conflicts of interests

The authors declare no conflict of interest.

References

[1] Catherine S, Rani MN, Suresh N. The metaverse economy: transforming money with digital currency. In *Creator's Economy in Metaverse Platforms: Empowering Stakeholders Through Omnichannel Approach*, 1st ed. Hershey: IGI Global Scientific Publishing, 2024. pp. 202–209.

[2] Weichbroth P, Wereszko K, Anacka H, Kowal J. Security of cryptocurrencies: a view on the state-of-the-art research and current developments. *Sensors* 2023, 23(6):3155.

[3] Scharfman J. Decentralized autonomous organization (DAO) fraud, hacks, and controversies. In *The Cryptocurrency and Digital Asset Fraud Casebook, Volume II: DeFi, NFTs, DAOs, Meme Coins, and Other Digital Asset Hacks*, 1st ed. Cham: Springer 2024. pp. 65–106.

[4] Bukhari ST, Janjua MU, Qadir J. Secure storage of crypto wallet seed phrase using ECC and splitting technique. *IEEE Open J. Comput. Soc.* 2024, 5:278–289.

[5] Krause D. The $1.4 billion Bybit Hack: cybersecurity failures and the risks of cryptocurrency deregulation. *SSRN* 2025, SSRN 5150171.

[6] Wu J, Lin K, Lin D, Zhang B, Wu Z, *et al.* Safeguarding blockchain ecosystem: understanding and detecting attack transactions on cross-chain bridges. *arXiv* 2024, arXiv:2410.14493.

[7] Chen Y, Chen H, Zhang Y, Han M, Siddula M, *et al.* A survey on blockchain systems: attacks, defenses, and privacy preservation. *High-Confid. Comput.* 2022, 2(2):100048.

[8] Guru A, Mohanta BK, Mohapatra H, Al-Turjman F, Altrjman C, *et al.* A survey on consensus protocols and attacks on blockchain technology. *Appl. Sci.* 2023, 13(4):2604.

[9] Platt M, McBurney P. Sybil in the haystack: a comprehensive review of blockchain consensus mechanisms in search of strong Sybil attack resistance. *Algorithms* 2023, 16(1):34.

[10] Akbar NA, Muneer A, ElHakim N, Fati SM. Distributed hybrid double-spending attack prevention mechanism for proof-of-work and proof-of-stake blockchain consensuses. *Future Internet* 2021, 13(11):285.

[11] Chaganti R, Boppana RV, Ravi V, Munir K, Almutairi M, *et al.* A comprehensive review of denial of service attacks in blockchain ecosystem and open challenges. *IEEE Access* 2022, 10:96538–96555.

[12] Madhushanie N, Vidanagamachchi S, Arachchilage N. Selfish mining attack in blockchain: a systematic literature review. *Int. J. Inf. Secur.* 2024, 23(3):2333–2351.

[13] Bennet D, Maria L, Sanjaya YPA, Zahra ARA. Blockchain technology: revolutionizing transactions in the digital age. *ADI J. Recent Innovation* 2024, 5(2):192–199.

[14] Wang X. Blockchain security and applications: a comprehensive analysis from hash functions to consensus algorithms. *Theor. Nat. Sci.* 2024, 31:292–298.

[15] Depoortère C. Examining the writings of Satoshi Nakamoto: a monetary analysis of the Bitcoin protocol. *Rev. Polit. Econ.* 2024, 37(2):392–411.

[16] Nasir NM, Hassan S, Zaini KM. Securing permissioned blockchain-based systems: an analysis on the significance of consensus mechanisms. *IEEE Access* 2024, 12:138211–138238.

[17] Li Z, Li W, Li X, Zhang Y. StateGuard: detecting state derailment defects in decentralized exchange smart contract. In *Companion Proceedings of the ACM Web Conference 2024*, Singapore, May 13–17, 2024, pp. 810–813.

[18] Lisi A, Maesa DDF, Mori P, Ricci L. Lightnings over rose bouquets: an analysis of the topology of the Bitcoin Lightning Network. In *2021 IEEE 45th Annual Computers, Software, and Applications Conference (COMPSAC)*, Madrid, Spain, July 12–16, 2021, pp. 324–331.

[19] Wen Y, Huang C. Exploration of mined block temporarily holding and enforce fork attacks by selfish mining pool in proof-of-work blockchain systems. *IEEE Access* 2022, 10:61159–61174.

[20] Mihaljević MJ, Wang L, Xu S, Todorović M. An approach for blockchain pool mining employing the consensus protocol robust against block withholding and selfish mining attacks. *Symmetry* 2022, 14(8):1711.

[21] Li X, Yu L, Luo X. On discovering vulnerabilities in Android applications. In *Mobile Security and Privacy*, 1st ed. Cambridge: Elsevier, 2017. pp. 155–166.

[22] Yu C, Yang W, Xie F, He J. Technology and security analysis of cryptocurrency based on blockchain. *Complexity* 2022, 2022(1):5835457.

[23] Kuznetsov O, Rusnak A, Yezhov A, Kuznetsova K, Kanonik D, *et al.* Merkle trees in blockchain: a study of collision probability and security implications. *Internet Things* 2024, 26:101193.

[24] Zhang C, Liao W, Liu X, Wu H, Alenazi MJ. A multi-signature scheme for defending malleability attack on DeFi. *IEEE Access* 2025, 13:17683–17694.

[25] Dai Q, Zhang B, Dong S. A DDoS-attack detection method oriented to the blockchain network layer. *Secur. Commun. Netw.* 2022, 2022(1):5692820.

[26] Dai Q, Zhang B, Dong S. Eclipse attack detection for blockchain network layer based on deep feature extraction. *Wireless Commun. Mobile Comput.* 2022, 2022(1):1451813.

[27] Platt M, Platt D, McBurney P. Sybil attack vulnerability trilemma. *Int. J. Parallel Emergent Distrib. Syst.* 2024, 39(4):446–460.

[28] Hao Y. Research of the 51% attack based on blockchain. In *2022 3rd International conference on computer vision, image and deep learning & international conference on computer engineering and applications (CVIDL & ICCEA)*, Changchun, China, May 20–22, 2022, pp. 278–283.

[29] Rouzbahani MA, Garakani HG. Blockchain security threats: survey. In *2024 11th International Symposium on Telecommunications (IST)*, Tehran, Iran, October 9–10, 2024, pp. 171–175.

[30] Aponte-Novoa FA, Orozco ALS, Villanueva-Polanco R, Wightman P. The 51% attack on blockchains: a mining behavior study. *IEEE Access* 2021, 9:140549–140564.

[31] Lee S, Kim S. Proof-of-stake at stake: predatory, destructive attack on PoS cryptocurrencies. In *Proceedings of the 3rd Workshop on Cryptocurrencies and Blockchains for Distributed Systems*, London, United Kingdom, September 25, 2020, pp. 7–11.

[32] Sarenche R, Nikova S, Preneel B. Mining power destruction attacks in the presence of petty-compliant mining pools. *arXiv* 2025, arXiv:2502.07410.

[33] Chen H, Chen Y, Xiong Z, Han M, He Z, *et al.* Prevention method of block withholding attack based on miners' mining behavior in blockchain. *Appl. Intell.* 2023, 53(9):9878–9896.

[34] Ferreira Torres C, Iannillo AK, Gervais A, State R. The eye of horus: spotting and analyzing attacks on ethereum smart contracts. In *International Conference on Financial Cryptography and Data Security*, Virtual, March 1–5, 2021, pp. 33–52.

[35] Alkhalifah A, Ng A, Watters PA, Kayes A. A mechanism to detect and prevent ethereum blockchain smart contract reentrancy attacks. *Front. Comput. Sci.* 2021, 3:598780.

[36] Sun J, Huang S, Zheng C, Wang T, Zong C, *et al.* Mutation testing for integer overflow in Ethereum smart contracts. *Tsinghua Sci. Technol.* 2021, 27(1):27–40.

[37] Liu Z, Li X, Peng H, Li W. Gastrace: detecting sandwich attack malicious accounts in Ethereum. In *2024 IEEE International Conference on Web Services (ICWS)*, Shenzhen, China, July 7–13, 2024, pp. 1409–1411.

[38] Basile M, Nardini G, Perazzo P, Dini G. SegWit extension and improvement of the BlockSim Bitcoin simulator. In *2022 IEEE International Conference on Blockchain (Blockchain)*, Espoo, Finland, August 22–25, 2022, pp. 115–123.

[39] Kedziora M, Pieprzka D, Jozwiak I, Liu Y, Song H. Analysis of segregated witness implementation for increasing efficiency and security of the Bitcoin cryptocurrency. *J. Inf. Telecommun.* 2023, 7(1):44–55.

[40] Alangot B, Reijsbergen D, Venugopalan S, Szalachowski P, Yeo KS. Decentralized and lightweight approach to detect eclipse attacks on proof of work blockchains. *IEEE Trans. Netw. Serv. Manage.* 2021, 18(2):1659–1672.

[41] Xu G, Guo B, Su C, Zheng X, Liang K, *et al.* Am I eclipsed? A smart detector of eclipse attacks for Ethereum. *Comput. Secur.* 2020, 88:101604.

[42] Wang Z, Lv Q, Lu Z, Wang Y, Yue S. ForkDec: accurate detection for selfish mining attacks. *Secur. Commun. Netw.* 2021, 2021(1):5959698.

[43] Madhushanie N, Vidanagamachchi S, Arachchilage N. BA-flag: a self-prevention mechanism of selfish mining attacks in blockchain technology. *Int. J. Inf. Secur.* 2024, 23(4):2783–2792.

[44] Karakostas D, Kiayias A, Zacharias T. Blockchain bribing attacks and the efficacy of counterincentives. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, Salt Lake City, USA, October 14–18, 2024, pp. 1031–1045.

[45] Awathare N. Decentralized exchange that mitigate a bribery attack. *arXiv* 2025, arXiv:2510.20645.

[46] Li Z, Li C, Wei X, Sun J. A hybrid statistical test and cross-check method for detecting block withholding attack of blockchain. In *2024 6th International Conference on Electronic Engineering and Informatics (EEI)*, Chongqing, China, June 28–30, 2024, pp. 418–423.

[47] Chinen Y, Yanai N, Cruz JP, Okamura S. Ra: a static analysis tool for analyzing re-entrancy attacks in Ethereum smart contracts. *J. Inf. Process. Syst.* 2021, 29:537–547.

[48] Yu R, Shu J, Yan D, Jia X. Redetect: reentrancy vulnerability detection in smart contracts with high accuracy. In *2021 17th International Conference on Mobility, Sensing and Networking (MSN)*, Exeter, UK, December 13–15, 2021, pp. 412–419.

[49] Ayoade G, Bauman E, Khan L, Hamlen K. Smart contract defense through bytecode rewriting . In *2019 IEEE International Conference on Blockchain (Blockchain)*, Los Alamitos, USA, July 14–17, 2019, pp. 384–389.

[50] Wang W, Song J, Xu G, Li Y, Wang H, *et al.* ContractWard: automated vulnerability detection models for Ethereum smart contracts. *IEEE Trans. Network Sci. Eng.* 2021, 8(2):1133–1144.

[51] Wu KW. Strengthening DeFi security: a static analysis approach to Flash Loan vulnerabilities. *arXiv* 2024, arXiv:2411.01230.

[52] Alhaidari A, Palanisamy B, Krishnamurthy P. Protecting DeFi platforms against non-price flash loan attacks. In *Proceedings of the Fifteenth ACM Conference on Data and Application Security and Privacy*, Pittsburgh, USA, June 4–6, 2024, pp. 281–292.

[53] Werapun W, Karode T, Arporntthip T, Suaboot J, Sangiamkul E, *et al.* The flash loan attack analysis (FAA) framework—a case study of the warp finance exploitation. *Informatics* 2022, 10(1):3.

[54] Li D, Zhang K, Wang L, Du G. A Geth-based real-time detection system for sandwich attacks in Ethereum. *Discover Comput.* 2024, 27(1):11.