Published: 22 November 2025

Citation in BibTeX format

CCS '25: ACM SIGSAC Conference on Computer and Communications Security
October 13 - 17, 2025
Taipei, Taiwan

Conference Sponsors:
SIGSAC

RESEARCH-ARTICLE

Denial of Sequencing Attacks in Ethereum Layer 2 Rollups

ZIHAO LI, The Hong Kong Polytechnic University, Hong Kong, Hong Kong, Hong Kong

ZHIYUAN SUN, The Hong Kong Polytechnic University, Hong Kong, Hong Kong, Hong Kong

ZHEYUAN HE, University of Electronic Science and Technology of China, Chengdu, Sichuan, China

JINZHAO CHU, The Hong Kong Polytechnic University, Hong Kong, Hong Kong, Hong Kong

HAO ZHOU, The Hong Kong Polytechnic University, Hong Kong, Hong Kong, Hong Kong

XIAPU LUO, The Hong Kong Polytechnic University, Hong Kong, Hong Kong, Hong Kong

View all

Open Access Support provided by:

University of Electronic Science and Technology of China

The Hong Kong Polytechnic University

Southern University of Science and Technology

Denial of Sequencing Attacks in Ethereum Layer 2 Rollups

Zihao Li[*]
The Hong Kong Polytechnic University
Hong Kong, China

Zhiyuan Sun[*]
The Hong Kong Polytechnic University
Hong Kong, China
Southern University of Science and Technology
Shen Zhen, China

Zheyuan He
University of Electronic Science and Technology of China
Chengdu, China

Jinzhao Chu
The Hong Kong Polytechnic University
Hong Kong, China

Hao Zhou
The Hong Kong Polytechnic University
Hong Kong, China

Xiapu Luo[†]
The Hong Kong Polytechnic University
Hong Kong, China

Ting Chen
University of Electronic Science and Technology of China
Chengdu, China

Yinqian Zhang[†]
Southern University of Science and Technology
Shen Zhen, China

Abstract

Layer 2 rollups offer promising solutions to address Ethereum's scalability issues. However, the centralized nature of the sequencer in these rollups makes them vulnerable to denial of service attacks, in which adversaries overwhelm the sequencer with invalid transactions that cannot be included in blocks, thereby exhausting its computational resources for transaction processing. To mitigate such threat, layer 2 rollups implement the legality check mechanism to filter out invalid transactions before they reach the sequencer.

In this work, we unveil a novel denial of sequencing attack that disrupts the liveness of layer 2 rollups at zero cost by bypassing the legality check. Specifically, our attack enables an adversary to craft malicious invalid transactions that bypass the legality check but are ultimately discarded by the sequencer after execution. As a result, the adversary can exhaust the sequencer's computational resources without incurring any fees. To construct such malicious transactions, we propose two approaches: a side-channel based approach and an incomplete check based approach, both of which rely on underlying vulnerabilities in rollups. Additionally, we investigate two widely used rollups, i.e., Arbitrum and Polygon zkEVM, and uncover four unknown vulnerabilities within them, which can be exploited to launch our attack using the two proposed approaches. Through extensive experiments conducted in a local environment, we demonstrate that all our attack variants, each exploiting distinct vulnerabilities, lead to severe attack effects at zero cost. Moreover, we discuss three feasible mitigations against our attack. At the time of writing, both the vulnerabilities and our attack have been acknowledged by the respective official teams, who have awarded us bug bounties to highlight the severity of our findings.

[*]Co-first authors
[†]Corresponding authors

CCS Concepts

• **Security and privacy → Distributed systems security**.

Keywords

Layer 2 Rollups, Denial of Sequencing Attacks

ACM Reference Format:
Zihao Li, Zhiyuan Sun, Zheyuan He, Jinzhao Chu, Hao Zhou, Xiapu Luo, Ting Chen, and Yinqian Zhang. 2025. Denial of Sequencing Attacks in Ethereum Layer 2 Rollups. In *Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security (CCS '25), October 13–17, 2025, Taipei.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3719027.3765100

1 Introduction

The rapid rise of blockchain technology have attracted a growing number of users seeking to utilize its transparent and freely accessible decentralized services, e.g., decentralized finance [46], non-fungible token [9], and decentralized autonomous organization [41]. However, this surge in user engagement has further exacerbated the scalability issues of blockchains [23, 42, 43]. This limitation arises because blocks are constrained in size and frequency by burdensome security mechanisms, such as consensus protocols and peer-to-peer networks [32]. For example, Ethereum, one of the most widely used blockchains, faces critical throughput challenges, processing only tens of transactions per second [5, 32]. Moreover, due to these throughput limitations, users are forced to compete for transaction inclusion by increasing their transaction fees. Notably, as highlighted by community [14], high transaction fees have become a major obstacle to blockchain usability.

In Ethereum ecosystem, several solutions are proposed to address the scalability issues, which can be categorized into two types:

internal and external solutions. Internal solutions introduce new functionalities to Ethereum underlying mechanisms to improve its transaction processing efficiency, e.g., transaction parallel execution [33], speculative execution [7], and sharding networks [8]. External solutions rely on external components to efficiently process transactions off Ethereum, thereby improving transaction throughput, e.g., layer 2 rollups [13] and state channels [12]. Among these solutions, layer 2 rollups have emerged as the most widely adopted solution in practice due to their low transaction fees, high throughput, and seamless integration with Ethereum [13]. Currently, the total value locked in main layer 2 rollups has surpassed 38B USD [25].

Ethereum layer 2 rollups process transactions on layer 2 blockchains through their sequencer. In addition to off-chain transaction processing, layer 2 rollups typically handle transactions in a centralized manner, forgoing the burdensome mechanisms to further reduce the time overhead in transaction processing [10]. During handling transactions, the sequencer periodically assembles processed transactions into batches and submits them, along with the resulting state root of layer 2 blockchains, to Ethereum as the payload of Ethereum transactions. This procedure ensures that layer 2 transactions inherit the same security guarantees as Ethereum transactions, which are safeguarded by Ethereum security mechanisms [44].

The sequencer plays a critical role in layer 2 rollups, as their liveness relies on the sequencer's capability to process transactions. However, due to the sequencer's centralized nature, it is vulnerable to denial of service attack. In such an attack, an adversary can flood the sequencer with invalid transactions that cannot be included in layer 2 blocks. These malicious transactions can exhaust the sequencer's computational resources, thereby compromising layer 2 rollups' liveness by hindering transaction processing.

To mitigate this threat and ensure that only valid transactions are forwarded to the sequencer, layer 2 rollups employ a legality check mechanism. Specifically, this mechanism pre-executes layer 2 transactions in parallel on a forked blockchain state and filters out invalid transactions. As a result, the legality check prevents the sequencer from wasting resources on processing invalid transactions, thereby mitigating such denial of service attacks. Additionally, even if an adversary attempts to overwhelm layer 2 rollups with valid transactions, the transaction charging mechanism [4] employed by layer 2 rollups will impose prohibitive financial costs on the adversary, thereby effectively deterring such attacks [45].

In our work, we unveil a novel *denial of sequencing* attack aimed at disrupting the liveness of layer 2 rollups by exhausting the sequencer's computational resources at zero cost. We illustrate the workflow of our attack in Fig. 1. Specifically, an adversary continuously crafts and submits malicious invalid transactions that can bypass the legality check, which serves to preemptively discard invalid transactions that cannot be included in layer 2 blocks. These malicious transactions are executed by the sequencer, consuming its computational resources and hindering subsequent transaction processing. Eventually, these malicious transactions are discarded by the sequencer without incurring any fees for the adversary.

The core challenge in launching our attack lies in constructing malicious invalid transactions that can deterministically bypass the legality check while still being discarded by the sequencer. To tackle this challenge, we propose two approaches for constructing such malicious transactions, i.e, a side-channel based approach and an incomplete check based approach. In the first approach, malicious transactions exploit side-channels in layer 2 rollups to determine whether they are being executed in the legality check or the sequencer. Based on this awareness, they dynamically adjust their execution behaviors, i.e., appearing benign during the legality check while becoming invalid in the sequencer, to bypass the legality check, and ultimately be discarded by the sequencer. In the second approach, malicious transactions trigger invalid execution behaviors that are not recognized by the legality check, but are later identified by the sequencer, causing them to be discarded.

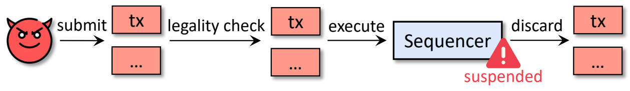


Figure 1: The workflow of our denial of sequencing attack.

To demonstrate the feasibility and generalizability of our denial of sequencing attack, we analyze two widely used layer 2 rollups, i.e., Arbitrum [2] and Polygon zkEVM [40], i.e., which represent the two major types of layer 2 rollups: optimistic layer 2 rollups and zero-knowledge layer 2 rollups [24], respectively. Through our investigation, we have identified four unique vulnerabilities in the two layer 2 rollups, which enable an adversary to successfully launch our denial of sequencing attack using our two proposed approaches for constructing malicious invalid transactions. Specifically, two of these vulnerabilities are related to the side-channel based approach, with each corresponding to an exploitable side-channel. The remaining two vulnerabilities pertain to the incomplete-check based approach, with each corresponding to an unchecked invalid execution behavior. By exploiting each of these four vulnerabilities using the corresponding approaches we propose, an adversary can construct malicious invalid transactions, which bypass the legality check and exhaust the sequencer's computational resources, thereby disrupting the liveness of target layer 2 rollups.

Attack scope. Arbitrum and Polygon zkEVM, as representative layer 2 rollups, are vulnerable to the specific attack variants that exploit the four vulnerabilities identified by us to construct and submit malicious invalid transactions to the corresponding layer 2 rollups. Furthermore, both optimistic and zero-knowledge layer 2 rollups are susceptible to the general attack methodology of our attack, as they rely on the legality check and the sequencer (i.e., the attack targets in our denial of sequencing attack) for transaction processing and block generation on their layer 2 blockchain. However, attacking other layer 2 rollups necessitates further investigation to identify exploitable vulnerabilities on them for constructing corresponding malicious invalid transactions.

Attack evaluation. We conduct extensive experiments to evaluate the feasibility and impact of our attack. Our results show that all four attack variants, which exploit distinct vulnerabilities identified by us, are feasible. The corresponding malicious transactions successfully bypass the legality check and are discarded at zero cost after execution in the sequencer. Additionally, we assess the potential for enhancing the attack effect by extending the execution time of malicious transactions. These transactions are generated by intensively executing inefficient opcodes, which typically consume significant time due to numerous disk I/O operations. Furthermore, we demonstrate that our attack can disable or partially disable the processing of benign transactions in layer 2 rollups end-to-end until the attack ceases. Moreover, we discuss three potential mitigations, and explore their respective strengths and weaknesses.

Vulnerability disclosure. We have disclosed all four identified vulnerabilities to the Arbitrum and Polygon zkEVM teams via the Immunefi platform [19], along with our attack that exploits these vulnerabilities to disrupt the liveness of their layer 2 rollups. At the time of writing, both the teams have acknowledged and patched the vulnerabilities, awarding us corresponding bug bounties to highlight their severity, and recognized the validity of our attack.

Contributions. We summarize our contributions as follows:

- *Attack.* We conduct the first in-depth study on the weaknesses of the legality check in Ethereum layer 2 rollups, introducing a novel denial of sequencing attack that disrupts the liveness of these rollups by constructing malicious invalid transactions capable of bypassing the legality check and exhausting the sequencer's computational resources at zero cost.
- *Vulnerability.* We uncover four vulnerabilities in two widely used layer 2 rollups, Arbitrum and Polygon zkEVM, which can be exploited to construct the malicious invalid transactions through two approaches, i.e., a side-channel based approach and an incomplete check based approach.
- *Evaluation.* We conduct extensive experiments to demonstrate the feasibility of our attack, and examine its impact and cost. Our results show that all attack variants can cause severe malicious effects, such as disabling transaction processing in layer 2 rollups end-to-end, while incurring zero cost for the adversary.
- *Mitigation.* We further explore three potential mitigations against our attack and discuss their respective strengths and weaknesses.

We refer readers to [1] for our full paper version with the appendix.

2 Background of Ethereum layer 2 rollups

In this work, we focus on the layer 2 rollups of Ethereum, which are designed to enhance Ethereum's scalability by increasing transaction throughput, reducing transaction congestion, and alleviating the workload on Ethereum (i.e., the layer 1 blockchain) [31]. These layer 2 rollups achieve this by aggregating and processing transactions on layer 2 (L2) blockchains off the layer 1 blockchain (i.e., Ethereum) [22, 32]. In addition to the off-chain transaction processing, layer 2 rollups typically handle transactions in a centralized manner, forgoing the burdensome mechanisms, such as consensus protocols and peer-to-peer networks, to further reduce the time overhead associated with transaction processing [10]. During handling layer 2 transactions, the layer 2 rollups periodically assemble the L2 transactions processed on the layer 2 blockchain into batches, and submit them, along with the resulting state root of the layer 2 blockchain, to Ethereum as the payload of Ethereum transactions. This procedure ensures that the layer 2 transactions inherit the same security guarantees as Ethereum transactions, which are safeguarded by the complicated security mechanisms of Ethereum [44].

There are two major types of layer 2 rollups for Ethereum, i.e., optimistic layer 2 rollups and zero-knowledge layer 2 rollups [32, 44]. Both types of layer 2 rollups can be divided into three critical components: the sequencer, the legality check, and the validators [44]. The primary differences between the two types of layer 2 rollups lie in their validators [32, 44], which we will detail later. In Fig. 2, we

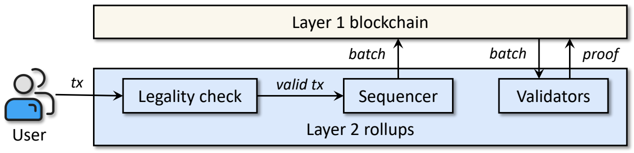


Figure 2: Architecture of Ethereum layer 2 rollups.

illustrate the architecture of the two types of L2 rollups, including the interaction between the three components, to facilitate a better understanding of the layer 2 rollups.

The sequencer plays a critical role in layer 2 rollups, and it is responsible for (i) ordering layer 2 transactions submitted from users, (ii) executing these transactions on layer 2 blockchains, (iii) assembling the processed transactions into batches, and (iv) submitting these batches to the layer 1 blockchain [32]. Layer 2 rollups, such as Arbitrum [2] and Polygon zkEVM [40], typically employ the sequencer in a permissioned and centralized manner to ensure its accountability for the layer 2 transactions it processes and to improve transaction processing efficiency [32].

The legality check ensures that only valid transactions are forwarded to the sequencer, preventing it from processing invalid transactions. Specifically, since the sequencer operates in a centralized manner, it is vulnerable to exhaustion of computational resources by being overwhelmed with invalid transactions (which are not eligible for inclusion in the layer 2 blockchain), potentially impeding services of layer 2 rollups, such as transaction processing. Furthermore, with the legality check in place, even if an adversary attempts to overwhelm the layer 2 rollups with valid transactions, the transaction charging mechanism [4] employed by layer 2 rollups will impose prohibitive financial costs on the adversary, thereby effectively deterring such attacks [45].

The validators are responsible for ensuring the finality of L2 transactions by (i) verifying that the L2 transactions are executed correctly as specified in the batches, and (ii) confirming that the updated L2 blockchain state, after the execution of these L2 transactions, is correct [32]. Specifically, optimistic and zero-knowledge layer 2 rollups employ different mechanisms in their validators to ensure transaction finality, which we detail below:

Optimistic layer 2 rollups. Optimistic L2 rollups like Arbitrum [2] employ interactive fraud proof schemes in their validators to ensure transaction finality [11, 22, 28]. Specifically, the validators in optimistic L2 rollups can initiate challenges to dispute the correctness of the L2 transactions and their execution results submitted on the L1 blockchain. If a challenge is accepted, the incorrect L2 transactions and their associated state transitions are rolled back and redressed. Once the challenge period has elapsed, or if the challenges fail to establish, the corresponding L2 transactions and their execution results are proved to be finalized and valid.

Zero-knowledge layer 2 rollups. Zero-knowledge L2 rollups, such as Polygon zkEVM [40], employ zero-knowledge proof based validity proof schemes in their validators to ensure transaction finality [6, 11, 28]. Specifically, the validators in zero-knowledge L2 rollups are responsible for generating validity proofs for L2 transactions and their execution results submitted on L1 blockchain. Only upon the validity proofs are generated are the transactions and their execution results are proved to be finalized and valid.

3 Preliminary

3.1 System model

We utilize the real-world deployment environment of representative L2 rollups like Arbitrum [2] and Polygon zkEVM [40] as our system model to ensure the practicality and reliability of our study. Specifically, in our model, L2 rollups consist of three major components: the sequencer, the legality check, and the validators, which collectively form the L2 rollup framework. Additionally, We employ Ethereum as the underlying L1 blockchain for L2 rollups. Notably, our study does not rely on specific features of Ethereum but instead focuses on the security of the generic framework of L2 rollups.

Moreover, L2 components are managed in a centralized manner and controlled by a permissioned entity. Users of L2 rollups can only submit their transactions to the sequencer, which is responsible for processing these transactions and periodically submitting them to the L1 blockchain. Additionally, the sequencer employs a load-balancing mechanism [29] to receive incoming user transactions, ensuring that it remains operational even under high transaction volume. For validators, they can operate under both optimistic and zero-knowledge L2 rollup mechanisms.

Furthermore, we assume that interactions between L2 components and the L1 blockchain are reliable and stable. Specifically, all three L2 components and the L1 blockchain are assumed to be unaffected by external factors such as power outages and network congestion. Additionally, external attack vectors originating from the L1 blockchain and L2 rollups, such as governance-layer attacks (e.g., governance issues [41]), consensus-layer attacks (e.g., Byzantine attacks [37]), and network-layer attacks (e.g., network intrusions [27]), are beyond the scope of our study.

Adversary. We consider an adversary $\mathcal{A}$ who interacts with L2 rollups by only submitting transactions, without the capability to interfere with the network or conduct social engineering attacks against L2 rollups. The adversary $\mathcal{A}$ has limited cryptocurrency funds in victim L2 rollups to configure the fee parameters for processing their transactions on L2 rollups. Notably, we will later demonstrate that launching our attack ultimately incurs no cost for the adversary, as the crafted attack transactions are not included in the L2 blockchain but instead exhaust the computational resources of the sequencer. Additionally, the adversary $\mathcal{A}$ can control multiple externally owned accounts (EOAs) to sign and submit transactions at a rate similar to that of legitimate users. As a result, L2 rollups cannot trivially distinguish attack transactions from legitimate ones based solely on network load analysis.

Attack target. The adversary $\mathcal{A}$ aims to disrupt the liveness of L2 rollups by submitting malicious transactions. These malicious transactions will be processed by the sequencer but ultimately discarded without being included in L2 blockchain. In this scenario, the sequencer is overwhelmed by the influx of malicious transactions, significantly degrading its ability to process benign transactions. Consequently, the liveness of victim L2 rollups is severely compromised, leading to a situation where either no transactions or only a minimal number of benign transactions can be processed.

3.2 Notations

– **L2 users.** L2 users can manage multiple EOAs and use them to sign L2 transactions, which are then submitted to L2 rollups. Additionally, when signing their transactions, L2 users need to configure fee parameters (similar to gas prices in Ethereum [47]) to cover the processing costs on L2 rollups.

– **Transaction pool.** The transaction pool maintained by the sequencer stores pending transactions that have not been processed and assembled into L2 blocks. Transactions in the pool are ordered based on their nonce values and the fee parameters specified by L2 users. Specifically, for transactions signed by the same L2 user, those with smaller nonces are processed first. For transactions from different L2 users, those with higher fee parameters are given priority, ensuring that users willing to pay higher transaction fees have their transactions prioritized for processing.

– **L2 transaction processing.** L2 transactions either invoke smart contracts by providing the necessary execution parameters, or serve to only transfer funds between EOAs. When contracts are involved, the sequencer executes the L2 transactions opcode by opcode on the L2 blockchain. Transaction execution concludes upon encountering termination opcodes, such as `RETURN` or `STOP`, or when triggering execution errors lead to premature termination. After execution, the sequencer includes the L2 transactions in L2 blocks, and assembles them into L2 batches to submit to the L1 blockchain.

– **Transaction execution errors.** Errors that cause transaction execution to halt can be categorized into two types: *legal errors* and *illegal errors*. Legal errors refer to issues that do not compromise the validity of transaction execution. For example, out-of-gas errors [32] and errors triggered by error-handling opcodes like `REVERT` fall under this category. Illegal errors refer to issues that render transaction execution invalid and prevent transaction inclusion in L2 blocks, typically due to errors that the sequencer cannot handle. For example, proof overflow errors in zero-knowledge L2 rollups [32] arise when the proof cost of L2 transactions exceeds the maximum limit of a block, preventing validators from generating corresponding validity proofs. In such cases, the transaction execution becomes invalid, and the sequencer needs to discard corresponding L2 transactions from the transaction pool.

– **Reloading invalid transactions.** For several transactions that raise *partially illegal errors*, i.e., errors that the sequencer can handle in specific scenarios, they will be returned to the transaction pool for inclusion in future L2 blocks. For example, consider zero-knowledge L2 rollups where three transactions, $\mathcal{T}_1$, $\mathcal{T}_2$, and $\mathcal{T}_3$, are processed by the sequencer. If, during block assembly, (i) the combined proof cost of $\mathcal{T}_1$ and $\mathcal{T}_2$ exceeds the block's maximum limit, and (ii) the proof cost of $\mathcal{T}_2$ alone does not exceed the limit, then $\mathcal{T}_2$ will be returned to the transaction pool. The sequencer will then process $\mathcal{T}_3$, and attempt to assemble $\mathcal{T}_1$ and $\mathcal{T}_3$ into a L2 block.

– **Transaction fees.** For transactions that either execute successfully or encounter only legal errors, the sequencer proceeds to include them in L2 blocks and charges transaction fees to the L2 users who signed them. These fees are determined based on the fee parameters set by L2 users during transaction signing, as well as the actual execution costs incurred by the sequencer during processing, similar to gas prices and costs in Ethereum [47].

– **Non-packable transactions.** Non-packable transactions refer to L2 transactions that either encounter illegal errors or consistently raise partially illegal errors during execution in the sequencer. Since these transactions can not be included in L2 blocks, the sequencer will not charge any fees to the L2 users who signed them.

Intended Functionality of Transaction Processing with Legality Check

Input: There exists a set of L2 users, denoted as $\mathbb{U}$, and a sequence of L2 transactions, denoted as $\mathbb{T}$, which are submitted to the sequencer. Each transaction in $\mathbb{T}$ is signed by one of the L2 users in $\mathbb{U}$.

Initial check: Invalid transactions are filtered out during the initial checks in the legality check procedure without involving transaction execution. This ensures that invalid transactions are efficiently discarded. Specifically, for an L2 transaction $\mathcal{T}$ in $\mathbb{T}$ signed by an EOA $\mathcal{U}$ in $\mathbb{U}$, the initial check verifies: (i) whether $\mathcal{U}$ has sufficient funds if $\mathcal{T}$ involves a fund transfer, (ii) whether $\mathcal{T}$ is signed with a valid nonce, i.e., the nonce must not decrease for each EOA, and (iii) whether $\mathcal{T}$ satisfies the gas bumping requirements, i.e., its transaction fee must be higher than that of a previous transaction (not yet included in a block) signed by $\mathcal{U}$ with the same nonce. If any of the three conditions are not met, the transaction $\mathcal{T}$ is deemed invalid and will be discarded.

Pre-execution check: Layer 2 rollups fork the current state of the L2 blockchain, denoted as $\mathcal{S}'$, and execute the L2 transactions in $\mathbb{T}$ that passed the initial check on the forked blockchain state $\mathcal{S}'$. The legality check determines whether any illegal errors or partially illegal errors occur during the transaction execution. If any illegal errors or partially illegal errors are encountered, the corresponding L2 transactions are deemed invalid and will be discarded. By executing on the forked state $\mathcal{S}'$, layer 2 rollups can conduct the pre-execution check in parallel, improving the efficiency of the legality check without affecting the actual state of the L2 blockchain. For the L2 transactions that pass the pre-execution check, they will be stored in the transaction pool, awaiting further processing by the sequence. The transactions in the pool will be ordered in a sequence, denoted as $\mathbb{T}'$, based on their nonce values and the fee parameters specified by L2 users.

Execution: The sequencer picks and executes L2 transactions in $\mathbb{T}'$ sequentially based on their priority. These transactions are executed on the latest state of the L2 blockchain, denoted as $\mathcal{S}$, and perform state transitions transaction by transaction. For example, when executing two transactions, $\mathcal{T}_1$ and $\mathcal{T}_2$, the blockchain state transitions from $\mathcal{S}$ to $\mathcal{S}_1$ and then to $\mathcal{S}_2$. Notably, at the initial part of transaction execution, the sequencer will perform several validity checks, such as verifying the gas limit and gas price. Transactions that violate these constraints are directly discarded. However, such transactions will still remain in transaction pool until they are picked and discarded by the sequencer.

Post-execution processing: The sequencer collects errors raised during each transaction execution. If a transaction encounters illegal errors, the sequencer will remove it from the execution list, and the next transaction will execute on the state prior to the discarded transaction. For example, consider two transactions, $\mathcal{T}_1$ and $\mathcal{T}_2$, executing on state $\mathcal{S}$. If $\mathcal{T}_1$ alters the state $\mathcal{S}$ to $\mathcal{S}_1$ but encounters illegal errors, the state $\mathcal{S}_1$ will be rolled back to $\mathcal{S}$, and $\mathcal{T}_2$ will re-execute on $\mathcal{S}$, transitioning the state to a new state $\mathcal{S}_1'$. Transactions that encounter illegal errors will be discarded by the sequencer from the transaction pool, and the L2 rollups will not charge any transaction fees to the corresponding L2 users. In contrast, transactions that encounter partially illegal errors will be returned to the transaction pool for inclusion in future L2 blocks.

Finalization: For L2 transactions included in L2 blocks, the sequencer assembles the processed transactions into batches. These batches are then submitted to the L1 blockchain as the payload of L1 transactions. Once the L1 transactions containing the L2 batches are finalized on the L1 blockchain, the L2 transactions are considered finalized and valid. Their transaction orders and execution results cannot be altered, except in the case of block reorganization or network fork on the L1 blockchain.

Figure 3: Intended functionality of transaction processing procedure with the legality check in layer 2 rollups.

3.3 Legality check

The legality check is designed to filter out invalid transactions before they reach the sequencer. In Fig. 3, we illustrate the intended functionality of the transaction processing with legality check in L2 rollups to provide a more comprehensive understanding of our attack. Specifically, the legality check consists of two stages: the initial check and the pre-execution check. The initial check stage aims to filter out invalid transactions without executing them, ensuring their efficient removal. The pre-execution check stage pre-executes transactions on a forked L2 blockchain state to identify invalid transactions by checking if any (partially) illegal errors encounter during execution. Invalid transactions identified in both the stages are discarded, and will not be processed in the sequencer.

After passing the legality check, transactions are forwarded to the sequencer for further processing. The sequencer collects errors encountered during execution, and filters out transactions that cannot be included in L2 blocks during the post-execution processing stage. This processing stage ensures that only valid transactions are ultimately included in L2 blockchain. Transactions excluded in this stage are either discarded or returned to the transaction pool for potential inclusion in future L2 blocks.

4 Attack overview

In our denial of sequencing attack, the adversary exploits the non-inclusion property of non-packable transactions in L2 blocks, enabling them to exhaust the sequencer in layer 2 rollups by forcing the sequencer to process these malicious transactions without incurring any cost. Since layer 2 rollups typically operate in a centralized manner, the sequencer is solely responsible for transaction processing and block production. As a result, when the sequencer is overwhelmed by malicious transactions, block production procedure slows down or halts entirely, disrupting the liveness of victim layer 2 rollups. In Fig. 4, we illustrate the workflow of our denial of sequencing attack, which consists of the following five steps:

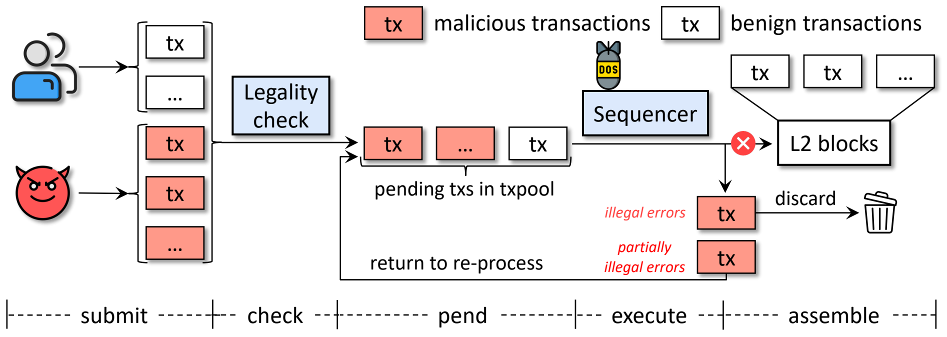


Figure 4: The workflow of our denial of sequencing attack.

(1) The adversary repeatedly crafts a series of non-packable transactions, which are designed to trigger illegal or partially illegal execution errors in the sequencer, with high transaction fee parameters, and submits them to the target layer 2 rollup.
(2) These malicious transactions pass both the initial check and pre-execution stages, and are forwarded to the sequencer. Before execution on L2 blockchain, these transactions are temporarily stored in the transaction pool, where they are ordered based on their configured transaction fee parameters.
(3) Since these malicious transactions are configured with high transaction fee parameters, the sequencer prioritizes them for execution. During execution on the L2 blockchain, these malicious transactions trigger illegal or partially illegal errors, preventing their inclusion in L2 blocks. Transactions raising illegal errors are discarded by the sequencer, while those raising partially illegal errors are returned to the transaction pool by the sequencer. Since these malicious transactions are not included in L2 blocks, the adversary incurs no cost, even though these transactions are configured with high fee parameters.
(4) After being returned to the transaction pool, these malicious transactions, configured with high transaction fee parameters, are prioritized for re-selection by the sequencer for re-execution on new blockchain state. Since these malicious transactions consistently trigger partially illegal errors, they repeatedly enter a loop of execution and return to the transaction pool.
(5) The sequencer becomes flooded by processing both malicious transactions submitted directly by the adversary, and those re-selected from the transaction pool. This exhaustion of computational resources leads to minimal or halted L2 transaction processing, ultimately disrupting victim L2 rollups' liveness.

Challenge. The core challenge in launching our attack lies in _deterministically_ crafting *non-packable* transactions. Specifically, both legality check and the sequencer execute L2 transactions to determine whether illegal or partially illegal errors are raised during transaction execution. In this case, non-packable transactions *must* bypass legality check without triggering illegal or partially illegal errors, yet encounter illegal or partially illegal errors in the sequencer, thereby preventing their inclusion in L2 blocks. If the crafted transactions prematurely encounter errors and fail to bypass legality check, they will be discarded in legality check before reaching the sequencer, resulting in no attack impact on the target L2 rollup while only consuming the adversary's computational resources. Additionally, if the crafted transactions do not encounter errors both in legality check and the sequencer, they will be included in L2 blocks, incurring significant costs for the adversary while failing to disrupt the liveness of the victim L2 rollup.

Approaches for crafting non-packable transactions. To successfully launch our attack, we design two approaches for crafting non-packable transactions, which are inspired by two observations derived from real-world examples.

Observation 1 (Side-channel): After being submitted to L2 rollups, L2 transactions can identify whether they are executed in legality check or in the sequencer through side-channels.

Approach 1: With the side-channels, the adversary can (i) deploy a malicious contract containing an `if` branch statement that checks the side-channel information, and (ii) launch the attack by crafting malicious transactions that invoke this contract. In this `if` statement, the contract behaves differently depending on whether the side-channel reports that the malicious transactions are executed in legality check or in the sequencer. When executed in legality check, the contract behaves benignly, allowing the malicious transactions to bypass legality check and reach the sequencer. However, when executed in the sequencer, the contract behaves maliciously by triggering illegal or partially illegal errors, preventing the malicious transactions from being included in L2 blocks.

Example 1: Li et al. [32] identify a potential state discrepancy between (i) the forked L2 blockchain state for executing L2 transactions in the pre-execution check stage, and (ii) the current L2 blockchain state in the sequencer within L2 rollups. Specifically, they find that the forked L2 blockchain state used in pre-execution check stage may lag behind the current L2 blockchain state. Leveraging this finding, they propose an approach to uncover bugs in L2 rollups by concealing the bug-triggering logic of a test transaction during the pre-execution check stage. This approach employs a pair of transactions, where, the first transaction, $\mathcal{T}_1$, modifies a specified blockchain state, while the second transaction, $\mathcal{T}_2$, acting as the test transaction, checks whether the state modification has taken effect. To ensure execution order, they assign $\mathcal{T}_2$ a larger nonce than $\mathcal{T}_1$, enforcing $\mathcal{T}_2$ to execute after $\mathcal{T}_1$. If the corresponding state remains unmodified (i.e., $\mathcal{T}_1$ has not been executed), the test transaction $\mathcal{T}_2$ infers that it is being executed on the stale forked L2 blockchain state during the pre-execution stage. Consequently, it will behave benignly to bypass the pre-execution check. Once forwarded to the sequencer, the test transaction shifts its execution logic to ultimately trigger bugs in zero-knowledge L2 rollups.

Notably, we present the above example to facilitate the understanding of side-channels in crafting non-packable transactions. However, the above finding does not fully satisfy the requirements of our side-channels for two reasons. First, based on this finding, the adversary cannot *deterministically* craft non-packable transactions, because, in some cases, the second transaction, $\mathcal{T}_2$ cannot distinguish between the pre-execution check stage and execution in the sequencer. Second, launching our attack based on this finding will incur financial costs for the adversary. When submitting a large number of malicious transactions, the adversary will find the attack cost prohibitive. Specifically, during the pre-execution check stage, the forked L2 blockchain state can align with the current blockchain state. For example, if the first transaction, $\mathcal{T}_1$, has already been included in L2 blocks before the pre-execution check stage forks the L2 blockchain state, the two states will be synchronized. In this case, the second transaction, $\mathcal{T}_2$, will mistakenly assume it is executed in the sequencer, even though it is actually executed on the forked L2 blockchain state during the pre-execution check stage. As a result, the second transaction, $\mathcal{T}_2$, will trigger errors and be discarded before reaching the sequencer. Furthermore, regardless of whether the second transaction, $\mathcal{T}_2$, is included in L2 blocks, the first transaction, $\mathcal{T}_1$, will always be included in L2 blocks, imposing financial costs on the adversary.

Observation 2 (Incomplete check): The legality check is not comprehensive. Certain illegal errors or partially illegal errors are not recognized as errors during the pre-execution procedure of legality check, but are considered errors during execution in the sequencer.

Approach 2: Given the incompleteness of legality check in recognizing illegal and partially illegal errors, the adversary can launch the attack by crafting malicious transactions that trigger these errors. In this scenario, even though the malicious transactions trigger illegal and partially illegal errors, they will still be forwarded to the sequencer. Once they encounter these errors in the sequencer, they will be either discarded or returned to the transaction pool, preventing their inclusion in L2 blocks.

Example 2: A disclosed vulnerability in Polygon zkEVM allows an adversary to bypass legality check, and force the sequencer to execute invalid transactions for cross-chain asset withdrawals [50]. Specifically, as a layer 2 rollup for Ethereum, Polygon zkEVM enables users to transfer assets from Ethereum to Polygon zkEVM via cross-chain bridges. During this process, users first lock their assets on Ethereum. Once Polygon zkEVM confirms the event of asset lock, users can send the *claim* transactions on Polygon zkEVM to withdraw their assets. Notably, claim transactions are sequenced in transaction pools based on their fee parameters, and Polygon zkEVM processes them similarly to normal L2 transactions, with additional logic for cross-chain asset transfers. In addition, claim transactions are permitted to be fee-free (i.e., with transaction fee parameters set to zero), since new users on Polygon zkEVM may not have assets to cover transaction fees of their claim transactions.

When processing claim transactions for cross-chain asset withdrawals, the pre-execution check stage first determines whether the transactions are valid. For example, transactions are invalid if no corresponding assets have been locked on Ethereum. However, the pre-execution check stage contains an incomplete check for filtering out invalid transactions. Specifically, based on whether a claim transaction's fee parameters are zero, and whether the transaction itself is valid, the pre-execution check stage needs to handle four possible cases: (i) invalid claim transactions with zero fee parameters, (ii) valid claim transactions with zero fee parameters, (iii) valid claim transactions with non-zero fee parameters, and (iv) invalid claim transactions with non-zero fee parameters. For the first three cases, the pre-execution check stage correctly identifies valid transactions, ensuring that only valid transactions are forwarded to the sequencer. However, in the fourth case, due to missing validation logic, the pre-execution check stage mistakenly recognizes invalid claim transactions with non-zero fee parameters as valid, and forwards them to the sequencer. Since these claim transactions are inherently invalid, the sequencer will ultimately recognize and discard them.

4.1 Attack design consideration

In this subsection, we address three key design questions to clarify the considerations behind our denial of sequencing attack.

Q1: How can our attack disrupt the liveness of L2 rollups by consuming the sequencer's computational resources?

In layer 2 rollups, the sequencer operates in a centralized manner, and is solely responsible for transaction processing and block production. By continuously submitting malicious transactions that can eventually reach the sequencer for execution, our attack can deplete the sequencer's computational resources. Notably, since the sequencer processes L2 transactions sequentially, our attack can exhaust its resources in two ways. First, since these malicious transactions do not incur transaction fees for the adversary, the adversary can generate and submit a large number of malicious transactions via multiple EOAs, thereby flooding the transaction pool. As a result, when selecting transactions for execution, the sequencer is more likely to process the malicious ones. Second, transaction execution can be resource-intensive and time-consuming [18, 38]. The adversary can craft transactions that involve significant CPU computation and disk I/O [29, 48], further straining the sequencer's computational capacity. By leveraging these two ways to craft malicious transactions, the sequencer is forced to allocate most of its computational resources in processing malicious transactions, leading to minimal or halted L2 block production and ultimately disrupting the liveness of the targeted L2 rollup.

Q2: Can an adversary disrupt the liveness of L2 rollups by directly overloading the legality check?

Although the legality check is a critical component in L2 rollups, and the sequencer needs to wait for transactions forwarded from the legality check before proceeding with further transaction execution and block production, it is impractical for an adversary to disrupt the liveness of L2 rollups by directly exhausting the legality check's computational resources. This is primarily due to differences in transaction handling between the legality check and the sequencer. Specifically, unlike the sequencer, which processes transactions sequentially, the legality check is designed to process transactions and determine their validity in parallel on a load-balanced cluster. As a result, the legality check can handle multiple L2 transactions simultaneously, making it difficult for an adversary to monopolize its computational resources and block other L2 users by submitting a large number of malicious transactions.

Q3. Why must malicious transactions be configured with high transaction fee parameters?
After passing the legality check, both malicious and benign transactions are temporarily stored in the transaction pool before being selected by the sequencer for further execution and block generation. Within the transaction pool, transactions are prioritized based on their fee parameters, with the sequencer selecting those offering higher fees for execution first. To maximize the likelihood of malicious transactions being selected and exhausting the sequencer's computational resources, the adversary must configure malicious transactions with high fee parameters. Otherwise, if benign transactions offer higher fees, they will be prioritized, allowing the sequencer to continue processing them and generating L2 blocks. This reduces the impact of our denial of sequencing attack on the liveness of the targeted L2 rollup. Notably, since these malicious transactions trigger illegal or partially illegal execution errors, they cannot be included in L2 blocks. Consequently, even if they are configured with exceptionally high fees to maximize their selection by the sequencer, they will ultimately be discarded without incurring any transaction costs for the adversary. Therefore, it is both reasonable and cost-effective for the adversary to conduct the attack by assigning high fee parameters to malicious transactions.

5 Attack design

In §4, we propose two approaches, i.e., side-channel and incomplete check, for deterministically crafting non-packable transactions. In this section, we will detail the vulnerabilities identified by us in popular layer 2 rollups, which enable an adversary to successfully launch our attack via using these two approaches.

L2 rollup selection. As detailed in §2, layer 2 rollups can be categorized into two types, i.e., optimistic and zero-knowledge layer 2 rollups. To demonstrate the generalizability of our denial of sequencing attack, we have identified vulnerabilities in two widely used layer 2 rollups representing each category, i.e., Arbitrum [2] for the former and Polygon zkEVM [40] for the latter. Notably, Polygon zkEVM introduces a new cost metric to quantify the proof cost of generating a zero-knowledge proof for transaction execution [40]. Additionally, Polygon zkEVM introduces a new virtual machine architecture for processing transactions, i.e., zkExecutor [40], to facilitate the generation of zk proofs for transaction execution. In contrast, Arbitrum retains the traditional gas cost model for transaction execution, similar to Ethereum, and continues to use the Ethereum Virtual Machine (EVM) for executing transactions.

To uncover vulnerabilities that an adversary can exploit to launch our denial of sequencing attack, we investigate the documentation and projects of Arbitrum [3] and Polygon zkEVM [39], focusing on their transaction processing procedures, particularly the logic of their legality check and post-execution processing in the sequencer. By analyzing discrepancies between these two stages, we identify four specific vulnerabilities: two related to side channels, and the other two related to incomplete check. We discuss these four vulnerabilities in detail in the following subsections.

5.1 Vulnerabilities related to side-channels

We have identified two vulnerabilities in Polygon zkEVM, which are related to side-channels. By exploiting the two vulnerabilities

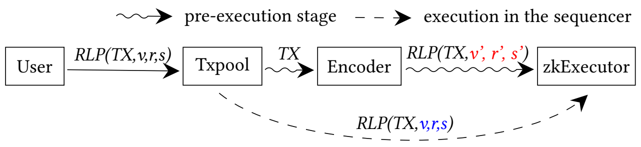


Figure 5: Transaction encoding and decoding during the pre-execution stage and the execution in the sequencer.

to launch our denial of sequencing attack, an adversary can craft malicious transactions that behave benignly during legality check, but trigger illegal or partially illegal errors in the sequencer.

Vulnerability #1: Discrepancy in proof cost. This side-channel arises from a discrepancy in transaction encoding and decoding between the pre-execution and the execution in the sequencer.

As shown in Fig. 5, after receiving a raw transaction encoded in RLP format [20] from a user, i.e., *RLP(TX, v, r, s)*, the transaction pool activates the legality check to determine whether the transaction can be stored in the pool for future processing and block generation. Notably, the values *v, r, s* represent the ECDSA signature values [21] of the raw transaction, which ensure the authenticity and integrity of the transaction sender [47]. During the legality check, the raw transaction is first decoded into an *unsigned transaction* object, i.e., *TX*, which contains the transaction metadata, such as the sender, recipient, and nonce, for the initial check. Following the initial check, the unsigned transaction is re-encoded into RLP format, i.e., *RLP(TX, v', r', s')*, where *v', r', s'* are fixed values specified by Polygon zkEVM. The encoded transaction is then forwarded to the zkExecutor for the pre-execution check. Once the legality check is passed, the transaction in raw format, i.e., *RLP(TX, v, r, s)* will be directly forwarded to the sequencer for further processing, where the zkExecutor is also employed for transaction execution.

The key discrepancy lies on the different encoded transactions sent to zkExecutor between the pre-execution stage and the execution in the sequencer, specifically the differing signature values of *v, r, s* and *v', r', s'*. During execution on the zkExecutor, these signature values are used in the process of recovering the transaction signer [40], where the proof cost varies depending on the specific signature values. This variation arises because the zkExecutor measures proof cost in a fine-grained manner, i.e., based on the executed code operations [40]. For example, recovering the transaction signer involves elliptic curve scalar multiplication. If certain bits of the elliptic curve points are zero, the corresponding scalar multiplication operations are bypassed, incurring no proof cost. As a result, different signature values can lead to different proof costs for recovering the transaction signer.

Vulnerability exploitation: Given the proof cost limit, denoted as $\mathcal{L}_b$, for transactions within a block on Polygon zkEVM, an adversary can exploit the first vulnerability to craft a malicious transaction whose proof cost remains within $\mathcal{L}_b$ during the legality check. However, upon execution in the sequencer, the transaction's proof cost exceeds $\mathcal{L}_b$, triggering a proof overflow error, ultimately resulting in the malicious transaction being discarded.

Crafting the malicious transaction involves two steps, corresponding to the two parts of a transaction's proof cost. Specifically, the proof cost of a transaction, denoted as C, is the sum of the proof cost for transaction preprocessing, denoted as C_p, and the proof

```
1 Contract Vulnerability#2 {
2   function DoS(uint256 signedGasPrice) public {
3     if (tx.gasprice == signedGasPrice) {
4       .../* benign operations */
5     } else {
6       .../* intentionally trigger errors */}}}
```

Figure 6: Code snippet of the contract demonstrating the exploitation of the second vulnerability.

cost for transaction execution [40], denoted as C_e. Notably, transaction preprocessing includes recovering the transaction signer from the signature values, such as *v', r', s'*. Since these signature values are fixed in the encoded transaction during pre-execution as *v', r', s'*, we denote the proof cost of transaction preprocessing during pre-execution as C'_p. In the first step, the adversary crafts a contract such that invoking it results in a proof cost of transaction execution C_e slightly below $\mathcal{L}_b$, while ensuring that the sum of C'_p and C_e does not exceed $\mathcal{L}_b$. In the second step, the adversary randomly generates transaction signatures to identify signature values that cause the sum of C_p and C_e to exceed $\mathcal{L}_b$. As a result, malicious transactions can be generated by (i) invoking the crafted contract and (ii) signing the transaction with the desired signature values.

Vulnerability #2: Discrepancy in effective gas price. This side-channel arises from the discrepancy in transactions' *effective gas price* between the pre-execution and the execution in the sequencer.

The *effective gas price* is introduced as part of a new transaction fee mechanism introduced by Polygon zkEVM, which is designed to enhance the fairness of transaction charges, and further reduce transaction fees. In this mechanism, users still specify a gas price when signing transactions, representing the amount they are willing to pay per unit of gas. However, the actual gas price applied, i.e., the effective gas price, is a reduced version of the originally specified gas price. This reduction factor is determined by two factors: (i) the cost of data availability associated with submitting the layer 1 transaction that carries the user's transaction as its payload (cf. details in §2), and (ii) the extent to which the originally signed gas price exceeds the minimum gas price suggested by Polygon zkEVM. During transaction execution, the *effective gas price* replaces the originally signed gas price. Users can retrieve its value during transaction execution using opcodes such as `GASPRICE` [40].

The key discrepancy lies on the distinct values of *effective gas price* between the pre-execution stage and the execution in the sequencer. Specifically, during pre-execution, as discussed in the first vulnerability, transactions in the unsigned transaction object are encoded into RLP format, and sent to the zkExecutor for pre-execution check. However, since the effective gas price is determined based on the information derived after pre-execution, such as the cost of data availability, Polygon zkEVM assigns the maximum possible value, i.e., the originally signed gas price, to the effective gas price in the encoded transaction. In contrast, during execution in the sequencer, the effective gas price is properly computed, and the updated value will be used for subsequent transaction processing.

Vulnerability exploitation: An adversary can exploit the effective gas price as a side-channel to determine whether the current execution is in the pre-execution stage or the sequencer.

In Fig. 6, we present a code snippet of a contract that exploits the second vulnerability. By generating transactions that invoke this contract, an adversary can launch our denial-of-sequencing attack. Specifically, the `DoS` function in the contract takes the originally signed gas price as an argument, i.e., `signedGasPrice` (Line 2). During execution, the function compares the originally signed gas price with the current effective gas price, i.e., `tx.gasprice`, to determine whether the current execution is in the pre-execution stage or the sequencer (Line 3). Notably, in Polygon zkEVM, when inquiring about the current gas price using `tx.gasprice`, the effective gas price will be returned, which is the actual gas price applied during transaction execution. When the originally signed gas price matches the effective gas price, the contract recognizes that it is executed in the pre-execution stage, and will perform benign operations (Line 4). Otherwise, the contract will intentionally trigger illegal or partially illegal errors (Line 6), such as proof overflow errors, leading to the transaction discarded.

(Partially) illegal errors: In exploiting Vulnerability #2, the adversary triggers illegal or partially illegal errors during execution in the sequencer, causing the transaction to be discarded.

In our example, we use proof overflow errors to illustrate the vulnerability exploitation. Notably, various illegal and partially illegal errors exist in practice, which can be exploited by attackers to carry out our attack. For example, attackers can force a contract to access an invalid memory region, triggering another illegal error, `ZKR_SM_MAIN_ADDRESS` [32], which results in the transaction being discarded. Moreover, since the limit for L2 block assembly is slightly lower than the threshold for triggering proof overflow errors due to certain preprocessing steps in the front of L2 blocks [40], attackers can craft malicious transactions whose proof cost falls between the two thresholds. In such cases, since the proof cost of the malicious transactions exceeds the limit for L2 block assembly but does not trigger proof overflow errors, the sequencer treats them as invalid, raises a partially illegal error, and returns these malicious transactions to the transaction pool for further processing.

5.2 Vulnerabilities related to incomplete check

We have identified two other vulnerabilities in Arbitrum and Polygon zkEVM, which are related to incomplete check. By exploiting the two vulnerabilities to launch our attack, an adversary can craft malicious transactions that bypass the legality check, despite triggering illegal or partially illegal errors during this process.

Vulnerability #3: Incomplete check in Arbitrum. This vulnerability arises from an incomplete check on the comparison between the maximum fee per gas and the maximum priority fee per gas in the legality check of Arbitrum.

Following EIP-1559 [34], which modifies the transaction pricing mechanism, the single gas price parameter specified by users in Arbitrum has been replaced by two components: the base fee and the tip. The base fee is dynamically adjusted based on network congestion and is burned after the transaction is completed, while the tip serves as an additional priority fee paid to validators and miners to prioritize transaction inclusion in blocks [34].

Additionally, users can specify the maximum gas fee per unit of gas, denoted as *GasFeeCap*, and the maximum priority fee per unit of gas, denoted as `GasTipCap`, when signing transactions [34]. During transaction execution in the sequencer, the sequencer verifies whether *GasFeeCap* exceeds *GasTipCap*. If this condition is violated, the sequencer raises an `ErrTipAboveFeeCap` error, and discards the transaction. This restriction exists because if *GasTipCap* exceeds *GasFeeCap*, the user will always be unable to pay the total priority fee, compromising the transaction pricing mechanism [34]. However, the legality check in Arbitrum does not enforce this validation, allowing transactions where *GasTipCap* exceeds *GasFeeCap* to pass.

Vulnerability exploitation: An adversary can craft malicious transactions by setting *GasTipCap* higher than *GasFeeCap*. These transactions will bypass the legality check but trigger the `ErrTipAboveFeeCap` error in the sequencer, ultimately causing them to be discarded.

Vulnerability #4: Incomplete check in Polygon zkEVM. This vulnerability originates from an incomplete check on the comparison between the gas limit of layer 2 transactions and the gas limit of layer 2 blocks in the legality check of Polygon zkEVM.

When signing layer 2 transactions, users can specify a gas limit for each transaction. The gas limit denotes the maximum amount of gas a user is willing to pay for a transaction, with each operation, such as EVM opcodes, consuming a certain amount of gas [40]. Generally, more complex operations consume more gas.

Similar to Ethereum, each layer 2 block in Polygon zkEVM has a maximum gas limit, which constrains the total gas consumption of all transactions within the block [40]. Additionally, during transaction execution in the sequencer, Polygon zkEVM verifies whether the gas limit specified by a layer 2 transaction is lower than the gas limit of a layer 2 block, and discards the transaction if this condition is violated. However, the legality check in Polygon zkEVM does not enforce this validation, allowing layer 2 transactions with gas limits exceeding the gas limit of a layer 2 block to pass.

Vulnerability exploitation: An adversary can craft malicious transactions with gas limits exceeding the gas limit of a layer 2 block. These transactions will bypass the legality check but are later identified as invalid during execution in the sequencer due to their illegal gas limits, ultimately causing them to be discarded.

5.3 Optimize the attack effect

By exploiting the four identified vulnerabilities, an adversary can craft malicious transactions that bypass the legality check but encounter illegal or partially illegal errors in the sequencer. In such cases, the adversary can enforce the sequencer to process malicious transactions that are ultimately discarded without incurring any cost. By exhausting the computational resources of the sequencer, the adversary can disrupt the liveness of layer 2 rollups.

To further optimize the attack effect, the adversary can incorporate time-consuming operations into the malicious transactions [38]. This optimization enhances the attack effect in two ways:

– First, the sequencer is forced to spend more time processing each malicious transaction. Given that the time required for signing transactions remains constant [48], the adversary can prolong the time for the sequencer to be unusable even with a limited number of malicious transactions, thereby amplifying the attack effect.

– Second, by increasing the processing time of malicious transactions, the sequencer has less capacity to handle benign transactions. Hence, even if benign transactions are still processed, the overall obstruction to transaction processing is exacerbated.

Constructing time-consuming malicious transactions: The key to constructing time-consuming malicious transactions is to craft contracts that frequently execute inefficient opcodes. Inefficient opcodes are those that require longer execution times relative to their gas costs [38, 48]. For example, prior studies have identified several inefficient opcodes, e.g., `SLOAD` [29, 38] and `EXTCODEHASH` [38, 48]. These opcodes typically retrieve state data from blockchain state storage on disk, necessitating intensive disk I/O operations and thereby increasing their execution time.

After crafting contracts with inefficient opcodes, the adversary can invoke them within malicious transactions to extend processing time before triggering illegal or partially illegal errors. For example, when exploiting the second vulnerability and invoking the malicious contract in Fig. 6, if a malicious transaction recognizes that execution is occurring in the sequencer (Line 3), it can first invoke contracts containing inefficient opcodes to prolong processing time before ultimately triggering illegal or partially illegal errors at Line 6, causing the malicious transaction to be discarded.

Attack effect's extent. The incomplete check arises from inconsistencies between the checks performed by the sequencer and those conducted by legality check. Since the sequencer enforces checks both at the initial part of transaction execution and after execution (§3.3), attack effect's extent (i.e., transaction execution time) varies depending on when malicious transactions are discarded. Specifically, if transactions are discarded at the initial part, the attack effect cannot benefit from the optimization, because contract execution is not involved. Conversely, if transactions are discarded after execution, the attack effect can be maximized through the optimization. Notably, in both cases, layer 2 rollups' liveness is disrupted, as malicious transactions are prioritized over benign transactions due to the configured high transaction fee parameters.

6 Evaluation

We evaluate the feasibility, cost, and impact of our attack on Arbitrum and Polygon zkEVM by answering the following four research questions. **RQ1:** *Is our attack feasible by crafting malicious invalid transactions that exploit the four identified vulnerabilities in Arbitrum and Polygon zkEVM?* **RQ2:** *What are the costs and impacts of our denial of sequencing attack on Arbitrum and Polygon zkEVM?* **RQ3:** *How effective is the proposed optimization in enhancing the effect of our attack?* **RQ4:** *Can existing tools identify the four vulnerabilities exploited by our attack to preemptively prevent it?*

Experimental Setup. Our experiments are conducted on a 64-bit machine equipped with 10 CPU cores and 128 GB memory. We deploy vulnerable versions of Arbitrum and Polygon zkEVM in our local environment to conduct our experiments. Notably, in the latest releases of Arbitrum and Polygon zkEVM, the official teams have confirmed our findings and resolved the identified vulnerabilities. Additionally, we set up a private Ethereum testnet to serve as the layer 1 blockchain for the two rollups. In the vulnerable versions of the two layer 2 rollups, we instrument the legality check and sequencer components to monitor our test transactions, determine whether they are executed in the legality check or sequencer, and record the execution results.

6.1 RQ1: Feasibility of our attack

As detailed in §5, we have identified four vulnerabilities in Arbitrum and Polygon zkEVM, which can be exploited to construct malicious

Table 1: Four variants of our attack against Arbitrum and Polygon zkEVM, each exploiting a distinct vulnerability.

Variants	Vulnerabilities	Attack approach	Vulnerable rollups
$Attack_1$	Vulnerability #1	side-channel based	Polygon zkEVM
$Attack_2$	Vulnerability #2	side-channel based	Polygon zkEVM
$Attack_3$	Vulnerability #3	incomplete check based	Arbitrum
$Attack_4$	Vulnerability #4	incomplete check based	Polygon zkEVM

invalid transactions for conducting our denial of sequencing attack on on these vulnerable layer 2 rollups.

In Table 1, we categorize the four corresponding attack variants against these vulnerable layer 2 rollups, each exploiting a different vulnerability. Specifically, the first two attack variants, DoS_1 and DoS_2, leverage the side-channel based approach to conduct attacks on Polygon zkEVM by exploiting the first two vulnerabilities, respectively. The third attack variant, DoS_3, employs the incomplete check based approach to attack Arbitrum by exploiting the third vulnerability. The fourth attack variant, DoS_4, also utilizes the incomplete-check based approach to attack Polygon zkEVM by exploiting the fourth vulnerability.

For each attack variant, the transaction construction process for generating malicious invalid transactions consists of three steps, as detailed below. This process is designed based on the vulnerability exploitation described in §5.

– *Step 1: Malicious contract deployment.* For each attack variant, we deploy the corresponding malicious contract if required. Specifically, for the first attack variant, we use the Yul language to develop the malicious contract, as it provides fine-grained control over contract execution. By leveraging Yul, we can precisely adjust the proof cost for invoking the malicious contract to remain slightly below the proof cost limit for a block, thereby constructing malicious transactions by invoking the malicious contract to exploit the first vulnerability. For the second attack variant, we deploy the corresponding malicious contract based on the template in Fig. 6. Additionally, within the malicious contract, we choose to trigger the proof overflow error by adopting the contract deployed for the first vulnerability exploitation. For the last two attack variants, since these variants do not require contract invocation to exploit the vulnerabilities, we do not deploy any malicious contract.

– *Step 2: Initial allocation of cryptocurrency.* We randomly generate a series of malicious EOAs, and allocate a small amount of cryptocurrency to each of them. Notably, conducting our attack does not incur any cost to the adversary. However, the adversary must hold several funds while submitting the transactions, as the initial check stage in the legality check verifies whether the transaction sender has sufficient funds to cover the transaction fees.

– *Step 3: Signing malicious transactions.* While signing the malicious transactions, the malicious EOAs invoke the deployed malicious contracts from the first step, and configure the corresponding parameters to trigger the identified vulnerabilities. For example, setting *GasTipCap* higher than *GasFeeCap* exploits the third vulnerability, while specifying gas limits exceeding the gas limit of a layer 2 block triggers the fourth vulnerability. Each EOA signs n malicious transactions. In our experiments, we set n to 5 to bypass checks that prematurely discard transactions due to an excessive number of pending transactions from a single EOA [30]. These malicious transactions are signed with incrementing nonce values, and submitted sequentially in their nonce order.

```
1 Contract AttackOptimizationTemplate {
2   function AttackOpt(uint32 i) public {
3     assembly {
4       for {} gt(i, 0) {i := sub(i, 1)}{
5         // OP: SLOAD or EXTCODESIZE
6         pop(OP(xor(blockhash(number()), gas()))}}}}
```

Figure 7: Contract template for generating malicious transactions that intensively execute inefficient opcodes.

To evaluate the feasibility of our attack, we follow the transaction construction process detailed above to generate a total of 1,000 malicious transactions for each attack variant. We then submit the malicious transactions for each variant individually to the corresponding vulnerable layer 2 rollups, and monitor their execution process and results. Finally, our experimental results demonstrate that all malicious transactions across the four attack variants pass the legality check, and are forwarded to the sequencer. However, upon execution, the sequencer discards all malicious transactions. Eventually, all these transactions are not included in layer 2 blocks.

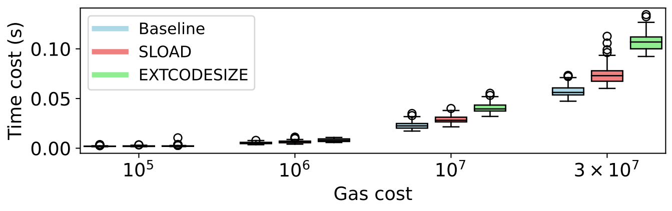
Answer to RQ1: *All attack variants are feasible, as malicious transactions bypass the legality check, and are discarded by the sequencer.*

6.2 RQ2: Effect of attack optimization

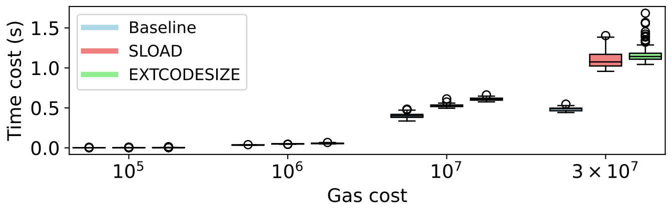
As discussed in §5.3, our attack effect can be enhanced by intensively executing inefficient opcodes, which typically consume significant time and resources due to numerous disk I/O operations. In such cases, the sequencer is forced to spend more time in processing each malicious transaction, leading to a more severe attack impact. Prior studies have identified `SLOAD` and `EXTCODESIZE` as representative inefficient opcodes [29, 35, 38, 48]. In this subsection, we evaluate the extent to which our attack's effect can be enhanced by intensively executing the two inefficient opcodes.

We generate the malicious transactions that intensively execute the inefficient opcodes by adopting the contract templates in prior studies [48]. During the attack, malicious transactions invoke the `AttackOpt` function, leading to intensive execution of inefficient opcodes. Specifically, as shown in Fig. 7, the contract template utilizes Yul to implement a loop-based execution of inefficient opcodes (Lines 4 - 6). Within the loop, `OP` can be replaced with either `SLOAD` or `EXTCODESIZE`, depending on the inefficient opcode attackers intend to execute. The loop iterates i times, where i is specified as a parameter of `AttackOpt` function. Additionally, the contract template employs `BLOCKHASH` and `XOR` to introduce randomness in the operands of the inefficient opcodes. Notably, repeatedly accessing the same state data results in caching the data in memory. In such cases, executing inefficient opcodes does not involve intensive disk I/O operations [16, 17]. To ensure sustained execution of inefficient opcodes with intensive disk I/O operations, randomness in operand selection is required, thereby persistently enhancing attack impact.

To obtain a more comprehensive characterization of attack effect optimization with different levels of inefficient opcode execution, we evaluate the execution time of optimized malicious transactions under different gas costs (i.e., 10^5, 10^6, 10^7, and 3×10^7), and different types of inefficient opcodes (i.e., `SLOAD` and `EXTCODESIZE`).



(a) Execution time of malicious transactions on Arbitrum



(b) Execution time of malicious transactions on Polygon zkEVM

Figure 8: Execution time of malicious transaction under different gas costs and inefficient opcodes.

Additionally, we employ a baseline strategy to construct malicious transactions that involve only stack operations.

Notably, in some cases, gas cost cannot be set arbitrarily. For example, when exploiting the first vulnerability, malicious transactions must adjust their proof cost to be slightly lower than the block's proof cost limit. Hence, the gas cost also approaches the block's gas cost limit, as it is related to the proof cost.

In our experiments, we execute the malicious transactions 100 times for each combination of gas cost, opcode type, and layer 2 rollup, and record the execution time of each transaction. We display our results in Fig. 8. Specifically, while consuming 3×10^7 gas, the optimization effect of malicious transactions reaches its peak. The median execution time of malicious transactions under the baseline strategy is 0.49 seconds on Polygon zkEVM and 0.056 seconds on Arbitrum. Compared to the baseline, transactions involving `EXTCODESIZE` consumes 1.14 seconds on Polygon zkEVM and 0.107 seconds on Arbitrum, increasing execution time by factors of 2.33 and 1.91, respectively. Similarly, transactions involving `SLOAD` take 1.07 seconds on Polygon zkEVM and 0.073 seconds on Arbitrum, extending execution time by factors of 2.18 and 1.30, respectively. Our results indicate that, for both inefficient opcodes across the two rollups, malicious transactions involving `EXTCODESIZE` consistently exhibit longer execution times than those involving `SLOAD`. Compared to the baseline strategy, the execution time of malicious transactions involving inefficient opcodes increases significantly as the gas cost rises. Additionally, the optimization effect becomes more pronounced with higher gas costs.

Answer to RQ2: *The execution time of malicious transactions can be extended by intensively executing inefficient opcodes.*

6.3 RQ3: Attack impact and cost

In this subsection, we evaluate whether our attack can end-to-end disable the processing of layer 2 transactions on the sequencer, and examine the impact and cost of our attack in comparison with other denial of service attacks in the Ethereum ecosystem.

End-to-end analysis. We collect historical benign transactions from Arbitrum and Polygon zkEVM, and submit them to the corresponding testing rollups while conducting our attack. During this process, we monitor the benign transactions' processing, and assess when they are executed in the sequencer. According to our results, while our attack is ongoing, benign transactions successfully pass the legality check and are temporarily stored in transaction pools.

For Polygon zkEVM, the processing of benign transactions is completely disabled, as the sequencer continues executing malicious transactions. This occurs because the configured fee parameters of malicious transactions are higher than those of benign transactions. Specifically, we scrape historical maximum fee parameters and set the malicious transaction fees to be ten times higher than these maximum values. As a result, even when both benign and malicious transactions co-exist in the transaction pool, the sequencer prioritizes executing malicious transactions over benign ones. Ultimately, our attack fully disables the sequencer, preventing benign transactions from being processed until the attack ceases.

For Arbitrum, the processing of benign transactions is delayed but not completely disabled. Specifically, the delay occurs as the sequencer interleaves the execution of both benign and malicious transactions. This happens because Arbitrum orders transactions on a "first-come, first-serve" (FCFS) basis, allowing benign transactions to be prioritized if they arrive before malicious ones, even when they have lower fee parameters. However, during our attack, Arbitrum's liveness is still disrupted, as benign transactions experience delays due to previously submitted malicious transactions, ultimately slowing down the transaction processing.

Comparison with related attacks. In Table 2, we compare our denial of sequencing attack with other related denial of service attacks in the Ethereum ecosystem in terms of their impact and cost, facilitating a better understanding of our attack.

In general, these attacks target different vulnerable components, either reducing transaction throughput or disabling specific vulnerable components. Compared to the attacks that reduce transaction throughput, our attack differs them in two key aspects: First, it can completely disable the transaction processing procedure because the sequencer in victim layer 2 rollups operates in a centralized manner. Second, our attack incurs zero cost for the adversary, as the identified vulnerabilities allow the adversary to deterministically craft non-packable transactions that bypass the legality check. In addition, compared to the attacks with zero cost, our attack can disrupt the liveness of the entire victim system (i.e., layer 2 rollups), ultimately disabling the processing of layer 2 transactions.

Answer to RQ3: *All attack variants disable or partially disable the processing of benign transactions in L2 rollups end-to-end at zero cost.*

6.4 RQ4: Vulnerability awareness

In this subsection, we explore whether existing tools can be utilized to potentially prevent our denial of sequencing attack by detecting the vulnerabilities exploited in our attack.

Tool selection. We select two open-source tools, Fluffy [49] and LOKI [36], to assess whether they can detect the vulnerabilities exploited in our attack. To our best knowledge, they are among the most relevant for identifying vulnerabilities in blockchain clients, particularly those related to the vulnerabilities we have identified. We set up both tools with the necessary adaptations to conduct

Table 2: Impact and cost comparison of denial of service attacks in Ethereum ecosystem

Attack	Vulnerable components	Attack impact	Financial cost
Perez et al. [38]	Gas pricing model	Reduce transaction throughput	Transaction fees
Li et al. [29]	RPC services	Latency in RPC services	Zero cost
Li et al. [30]	Transaction pool	Disable target transaction pools	Transaction fees
Yaish et al. [48]	Censorship mechanism	Reduce transaction throughput	Transaction fees
He et al. [18]	Gas pricing model	Degrade blockchain performance	Transaction fees
Tsuchiya et al. [45]	Modified transaction validation	Amplify network traffic	Zero cost
Denial of Sequencing	**Legality check in L2 rollups**	**Disable processing of L2 transactions**	**Zero cost**

testing on the vulnerable L2 rollups, including adjusting test transactions' format to ensure compatibility with target L2 rollups. Additionally, to ensure reliable results, we refrain from modifying their core methodologies for generating test transactions. Notably, there are other related tools [32] that have not released their code. We choose to compare and discuss them qualitatively in Appendix D.

To determine whether the vulnerabilities we identified, or other similar vulnerabilities that can be exploited to conduct our attack, are triggered by the tools, we instrument the legality check and sequencer components in the two vulnerable layer 2 rollups. During testing, we consider a desired vulnerability to be triggered if the tools can continuously generate m transactions that pass the legality check but are ultimately discarded by the sequencer. We choose this metric for identifying whether the desired vulnerabilities are triggered rather than others, such as transaction throughput, because the tools may either generate invalid transactions that are discarded by the legality check or trigger unexpected errors that cause the sequencer to crash. While in these scenarios, the transaction throughput may be reduced to zero, the root cause is not related to the triggering of our identified vulnerabilities or similar vulnerabilities. In such cases, the tools are unable to prevent our attack by preemptively identifying the underlying vulnerabilities.

In our experiments, we run the two tools for 12 hours on each of the vulnerable layer 2 rollups, i.e., Arbitrum and Polygon zkEVM. Additionally, we set m to 10 to determine whether the vulnerabilities exploitable for conducting our attack are triggered. Finally, our experimental results show that both tools are unable to detect the four vulnerabilities identified by us, as well as other similar vulnerabilities. The main reason for their incapability is that the vulnerabilities exploited in our attack are more challenging to be detected. Specifically, to disrupt the liveness of layer 2 rollups by triggering the vulnerabilities identified by us (or similar ones), the tools must continuously generate test transactions that can trigger these vulnerabilities. Unfortunately, these tools mainly focus on vulnerabilities that can be triggered by a single transaction, such as those that cause the sequencer to crash. As a result, the tools fail to continuously generate the desired test transactions because the execution feedback does not match their expectations.

Answer to RQ4: *Existing tools fail to identify vulnerabilities that can be exploited to construct malicious transactions for our attack, rendering them incapable of preemptively preventing it.*

7 Possible mitigations

Even if the four identified vulnerabilities are fixed, our attack remains a threat to layer 2 rollups due to its generalizability (as discussed in §8). For example, if attackers identify similar vulnerabilities in a layer 2 rollup's legality check, they can still disrupt the victim layer 2 rollup's liveness by crafting malicious invalid transactions to exhaust the sequencer's computational resources. In this section, we explore three potential mitigations against our attack, and discuss their respective strengths and weaknesses.

Reputation based transaction ordering. In our evaluation of the attack effect on Arbitrum (RQ3 in §6.3), we observe that, while Arbitrum's liveness is disrupted, the attack effect is mitigated to some extent (i.e., the processing of benign transactions is not entirely disabled). This is because Arbitrum employs a FCFS transaction ordering mechanism. However, this mechanism has a critical limitation, i.e., benign transactions submitted after malicious transactions will experience delays until the attack subsides.

In contrast to FCFS mechanism, a reputation based transaction ordering mechanism can further mitigate the attack effect by prioritizing the processing of benign transactions over malicious ones. Specifically, to mitigate the attack effect, such a mechanism requires (i) rewarding L2 users who actively submit benign transactions, and (ii) penalizing those who frequently submit invalid transactions (i.e., transactions resembling malicious transactions that cannot be included in blocks). Additionally, new L2 users who have not yet submitted transactions should have default reputation scores higher than those of users who frequently submit invalid transactions.

A reputation based mechanism that satisfies these requirements offers advantages over FCFS mechanism in mitigating the attack effect. Specifically, even if attackers initially prepare multiple EOAs with high reputation scores (by submitting benign transactions), their reputation scores will progressively decrease during the attack, because attackers must continuously submit malicious transactions to sustain the attack (§6.3). Meanwhile, the reputation scores of benign users will increase, enabling their transactions to be prioritized over those of attackers. Notably, even in the worst-case scenario for a reputation based mechanism, where all benign users are new users, its mitigating effect will be no worse than that of FCFS mechanism. This is because, under a reputation based mechanism, benign transactions of new users sent after malicious ones can still be prioritized, as attackers' reputation scores can be penalized below the default score of new users. In contrast, FCFS mechanism does not offer such mitigation, making it inherently less effective in countering the attack when benign transactions are sent after malicious ones. However, implementing a reputation based mechanism incurs extra computational and storage overhead, which can impact the performance of L2 rollups. Notably, our above discussion is primarily qualitative. For layer 2 protocols that consider adopting it, we recommend conducting a comprehensive quantitative analysis to assess its trade-offs in practice.

Penalty rationality. We acknowledge that the penalty strategy may appear restrictive, potentially leading to honest users being wrongfully penalized. For honest users who occasionally submit invalid transactions, the punishment remains reasonable, as it only delays their transaction processing rather than blocking it entirely, thereby minimizing the negative impact. For honest users who frequently submit invalid transactions triggering partially illegal errors, distinguishing them from malicious users exhibiting similar behaviors is inherently challenging. In such cases, the penalty strategy can serve as a trade-off, for example, by selectively whitelisting certain errors like reaching the block limit, to balance mitigating harm to benign users with protecting against attackers. Nevertheless, even with such adjustments, the strategy may still inadvertently disincentivize honest users from actively participating in L2 rollups.

Formal verification methods. Formal verification methods are known to used for rigorously proving the behavioral consistency and equivalence between different programs [26].

Formal verification methods are feasible solutions for mitigating our attack by identifying vulnerabilities that can be exploited. This is because the root cause of malicious transactions bypassing the legality check but being discarded by the sequencer lies in the discrepancies between how the legality check and the sequencer handle transactions. Specifically, malicious transactions, which are crafted using the side-channel based approach, exploit vulnerabilities arising from differences in transaction execution environments between the legality check and the sequencer. Besides, malicious transactions, which are crafted using the incomplete check based approach, exploit vulnerabilities caused by inconsistencies in the checks performed by the legality check and the sequencer. Hence, formal verification is viable by eliminating discrepancies in transaction handling between the legality check and the sequencer, thereby removing vulnerabilities that can be exploited in our attack.

However, transaction execution involve complex execution logic, particularly in zkExecutor, which incorporates extra logic for generating zero-knowledge proofs. Hence, formal verification may struggle to comprehensively explore the entire state space. If certain portions of the state space remain unexplored, formal verification may fail to detect vulnerabilities that can be exploited in our attack.

Differential testing. Differential testing is an automated software testing technique that compares the behavior and outputs of two programs implementing the same functionality on identical inputs to identify discrepancies indicative of vulnerabilities [15].

Differential testing is a feasible solution to mitigating our attack for reasons similar to formal verification methods, i.e., their ability to identify vulnerabilities exploitable by our attack by uncovering inconsistencies or incompleteness between the legality check and the sequencer. However, differential testing is subject to both unsoundness and incompleteness, and its findings must be manually validated. To effectively uncover the specific vulnerabilities exploited by our attack, conventional differential testing must be appropriately tailored in three aspects. First, the execution behaviors and outputs of the sequencer and the legality check must be aligned to avoid false positives, as these two components do not implement exactly the same functionality. For example, the sequencer includes additional logic for assembling transactions into blocks and submitting block batches to layer 1 blockchain. Second, test transactions need to be mutated to explore diverse execution behaviors in both sequencer and legality check, thereby increasing the likelihood of uncovering discrepancies and potential vulnerabilities. Third, execution feedback should drive the mutation process, guiding differential testing to continuously trigger the vulnerabilities necessary for launching our attack (RQ4 in §6.4).

8 Discussion

Due to page limits, we focus here on our attack's generality, differences between resource exhaustion attacks in L1 and L2, and ethical consideration, with the rest part provided in Appendix B.

Our attack's generality. Our attack pattern is generalizable, and poses threats to other L2 rollups for three reasons. First, it targets L2 rollups whose transaction processing includes the legality check, which is widely adopted in existing L2 rollups to filter out invalid transactions before they reach the sequencer. Second, the legality check inherently lacks completeness in filtering out all invalid transactions, making the sequencer vulnerable to processing non-packable transactions. For example, certain errors can only be determined at runtime within sequencer. Notably, the implementation vulnerabilities exploited by our attack stem from sources of incompleteness in the legality check that allow an adversary to deterministically construct invalid transactions that bypass the legality check but are ultimately discarded by the sequencer. Third, due to the tight coupling between L2 components, the implementation vulnerabilities exploitable by our attack are likely to be introduced and remain unnoticed [32], even when using testing tools (RQ4).

Moreover, our attack variant that specifically exploits the inherent incompleteness of the legality check is feasible. For example, the variant can exploit the side-channel based approach proposed by Li et al. [32], where certain errors are triggered under runtime conditions. However, this variant suffers from two limitations compared to our main attack: (1) the adversary cannot deterministically craft non-packable transactions, and (2) launching such an attack would incur financial costs for the adversary. Nonetheless, this variant still poses threats to layer 2 protocols, as it can exhaust computational resources, albeit at a financial cost to the adversary. Notably, the cost can be further decreased by carefully crafting contract logic to terminate execution early when the runtime conditions for error triggering are not met, thereby reducing overall gas consumption.

Differences between resource exhaustion attacks in L1 and L2. The attack pattern that exploits inconsistencies or incompleteness in pre-validation to consume resources before final rejection exists in both layer 1 and layer 2 environments. However, due to differences in their system models, particularly the centralization of transaction execution and block production in layer 2, the consequences of such attacks vary significantly, motivating attackers to pursue different targets and outcomes. For example, in layer 1, attacks such as the DETER attack [30] and the blockchain amplification attack [45] allow adversaries to disrupt the liveness of a single node or a subset of the network. In contrast, our attack exploits the centralized sequencer model in layer 2 environment to disrupt the entire layer 2 system, causing no transactions or only a minimal number of benign transactions to be processed.

Ethical consideration. We confined the attack evaluation to our local environment, ensuring that our experiments did not disrupt external entities. To prevent potential exploitation, we exclusively disclosed the vulnerabilities to Arbitrum and Polygon zkEVM teams through Immunefi platform [19]. Moreover, the technical details of vulnerability exploitation were not made publicly available to ensure attackers cannot exploit them before patches were in place.

9 Related work

Related studies can be categorized into three classes, i.e., attacks against layer 2 rollups, security enhancement for layer 2 rollups, and attacks against Ethereum. We detail them in Appendix C.

10 Conclusion

Our study brings to light the weaknesses of the legality check in mitigating security threats to Ethereum layer 2 rollups arising from their centralized nature. By uncovering four critical vulnerabilities in the legality check of two widely used layer 2 rollups, i.e., Arbitrum and Polygon zkEVM, we demonstrate that an adversary can exploit these vulnerabilities to launch the denial of sequencing attack to disrupt the liveness of the targeted layer 2 rollups at zero cost.

Acknowledgements

The authors thank the anonymous reviewers for their constructive comments. This work is partly supported by Hong Kong RGC Projects (PolyU15224121, PolyU15231223), HK RGC Grant for Theme-based Research Scheme Project T43-513/23-N, National Natural Science Foundation of China under Grant (No.62172301, No.62332004), and Sichuan Provincial Natural Science Foundation for Distinguished Young Scholars under Grant No.2023NSFSC1963.

References

[1] 2024. Our full paper with the appendix. https://zzzihao-li.github.io/.
[2] Arbitrum. 2025. Arbitrum layer 2 protocol. https://arbitrum.io/.
[3] Arbitrum. 2025. Arbitrum Nitro. https://github.com/OffchainLabs/nitro/.
[4] Arbitrum. 2025. Arbitrum transaction charging mechanism. https://docs.arbitrum.io/how-arbitrum-works/gas-fees.
[5] Mirko Bez, Giacomo Fornari, and Tullio Vardanega. 2019. The scalability challenge of ethereum: An initial quantitative analysis. In *IEEE International Conference on Service-Oriented System Engineering*.
[6] Stefanos Chaliasos, Itamar Reif, Adrià Torralba-Agell, Jens Ernstberger, Assimakis Kattis, and Benjamin Livshits. 2024. Analyzing and Benchmarking ZK-Rollups. *AFT* (2024).
[7] Yang Chen, Zhongxin Guo, Runhuai Li, Shuo Chen, Lidong Zhou, Yajin Zhou, and Xian Zhang. 2021. Forerunner: Constraint-based speculative transaction execution for ethereum. In *SOSP*.
[8] Hung Dang, Tien Tuan Anh Dinh, Dumitrel Loghin, Ee-Chien Chang, Qian Lin, and Beng Chin Ooi. 2019. Towards scaling blockchain systems via sharding. In *SIGMOD*.
[9] Dipanjan Das, Priyanka Bose, Nicola Ruaro, Christopher Kruegel, and Giovanni Vigna. 2022. Understanding security issues in the NFT ecosystem. In *CCS*.
[10] Martin Derka, Jan Gorzny, Diego Siqueira, Donato Pellegrino, Marius Guggenmos, and Zhiyang Chen. 2024. Sequencer Level Security. *arXiv* (2024).
[11] Luca Donno. 2022. Optimistic and validity rollups: Analysis and comparison between optimism and starknet. *arXiv* (2022).
[12] Stefan Dziembowski, Lisa Eckey, Sebastian Faust, Julia Hesse, and Kristina Hostáková. 2019. Multi-party virtual state channels. In *EUROCRYPT*.
[13] Christof Ferreira Torres, Albin Mamuti, Ben Weintraub, Cristina Nita-Rotaru, and Shweta Shinde. 2024. Rolling in the shadows: Analyzing the extraction of mev across layer-2 rollups. In *CCS*.
[14] Finbold. 2022. Vitalik Buterin admits transaction fees are huge problem. https://finbold.com/vitalik-buterin-admits-fees-are-a-huge-problem-for-ethereums-usability/.
[15] Muhammad Ali Gulzar, Yongkang Zhu, and Xiaofeng Han. 2019. Perception and practices of differential testing. In *ICSE-SEIP*.
[16] Zheyuan He, Zihao Li, Jiahao Luo, Feng Luo, Junhan Duan, Jingwei Li, Shuwei Song, Xiapu Luo, Ting Chen, and Xiaosong Zhang. 2025. Auspex: Unveiling Inconsistency Bugs of Transaction Fee Mechanism in Blockchain. In *USENIX Security*.
[17] Zheyuan He, Zihao Li, Ao Qiao, Jingwei Li, Feng Luo, Sen Yang, Gelei Deng, Shuwei Song, Xiaosong Zhang, Ting Chen, and Xiapu Luo. 2025. Maat: Analyzing and Optimizing Overcharge on Blockchain Storage. In *USENIX FAST*.
[18] Zheyuan He, Zihao Li, Ao Qiao, Xiapu Luo, Xiaosong Zhang, Ting Chen, Shuwei Song, Dijun Liu, and Weina Niu. 2024. NURGLE: Exacerbating Resource Consumption in Blockchain State Storage via MPT Manipulation. *arXiv* (2024).
[19] Immunefi. 2025. Bug bounty program of Immunefi platform. https://immunefi.com/bug-bounty-program/
[20] Kamil Jezek. 2021. Ethereum data structures. *arXiv* (2021).
[21] Don Johnson, Alfred Menezes, and Scott Vanstone. 2001. The elliptic curve digital signature algorithm. *International journal of information security* (2001).
[22] Harry Kalodner, Steven Goldfeder, Xiaoqi Chen, S Matthew Weinberg, and Edward W Felten. 2018. Arbitrum: Scalable, private smart contracts. In *USENIX Security*.
[23] Dodo Khan, Low Tang Jung, and Manzoor Ahmed Hashmani. 2021. Systematic literature review of challenges in blockchain scalability. *Applied Sciences* (2021).
[24] Adrian Koegl, Zeeshan Meghji, Donato Pellegrino, Jan Gorzny, and Martin Derka. 2023. Attacks on rollups. In *DICG*.
[25] l2beat. 2025. Statistics on the total value locked in popular layer 2 rollups. https://l2beat.com/scaling/tvs.
[26] David Lacey, Neil D Jones, Eric Van Wyk, and Carl Christian Frederiksen. 2002. Proving correctness of compiler optimizations by temporal logic. *ACM SIGPLAN Notices* (2002).
[27] S Latha and Sinthu Janita Prakash. 2017. A survey on network attacks and Intrusion detection systems. In *ICACCS*.
[28] Jiasun Li. 2023. On the security of optimistic blockchain mechanisms. *ssrn* (2023).
[29] Kai Li, Jiaqi Chen, Xianghong Liu, Yuzhe Richard Tang, XiaoFeng Wang, and Xiapu Luo. 2021. As Strong As Its Weakest Link: How to Break Blockchain DApps at RPC Service.. In *NDSS*.
[30] Kai Li, Yibo Wang, and Yuzhe Tang. 2021. Deter: Denial of ethereum txpool services. In *CCS*.
[31] Zihao Li, Zheyuan He, Xiapu Luo, Ting Chen, and Xiaosong Zhang. 2025. Unveiling Financially Risky Behaviors in Ethereum ERC20 Token Contracts. *CJE* (2025).
[32] Zihao Li, Xinghao Peng, Zheyuan He, Xiapu Luo, and Ting Chen. 2024. fAmulet: Finding Finalization Failure Bugs in Polygon zkRollup. In *CCS*.
[33] Haoran Lin, Hang Feng, Yajin Zhou, and Lei Wu. 2025. ParallelEVM: Operation-Level Concurrent Transaction Execution for EVM-Compatible Blockchains. In *EuroSys*.
[34] Yulin Liu, Yuxuan Lu, Kartik Nayak, Fan Zhang, Luyao Zhang, and Yinhong Zhao. 2022. Empirical analysis of eip-1559: Transaction fees, waiting times, and consensus security. In *CCS*.
[35] Feng Luo, Huangkun Lin, Zihao Li, Xiapu Luo, Ruijie Luo, Zheyuan He, Shuwei Song, Ting Chen, and Wenxuan Luo. 2024. Towards Automatic Discovery of Denial of Service Weaknesses in Blockchain Resource Models. In *CCS*.
[36] Fuchen Ma, Yuanliang Chen, Meng Ren, Yuanhang Zhou, Yu Jiang, Ting Chen, Huizhong Li, and Jiaguang Sun. 2023. LOKI: State-Aware Fuzzing Framework for the Implementation of Blockchain Consensus Protocols.. In *NDSS*.
[37] Ulysse Pavloff, Yackolley Amoussou-Guenou, and Sara Tucci-Piergiovanni. 2024. Byzantine attacks exploiting penalties in Ethereum PoS. In *DSN*.
[38] Daniel Perez and Benjamin Livshits. 2020. Broken metre: Attacking resource metering in EVM. *NDSS* (2020).
[39] Polygon. 2025. Polygon zkEVM. https://github.com/0xpolygonhermez.
[40] Polygon. 2025. Polygon zkEVM layer 2 protocol. https://polygon.technology/polygon-zkevm.
[41] Tanusree Sharma, Yujin Potter, Kornrapat Pongmala, Henry Wang, Andrew Miller, Dawn Song, and Yang Wang. 2024. Unpacking how decentralized autonomous organizations (daos) work in practice. In *ICBC*.
[42] Dongxian Shi, Xiaoqing Wang, Ming Xu, Liang Kou, and Hongbing Cheng. 2023. RESS: A reliable and effcient storage scheme for bitcoin blockchain based on raptor code. *CJE* (2023).
[43] Jian Su and Mengnan Jiang. 2023. A hybrid entropy and blockchain approach for network security defense in SDN-based IIoT. *CJE* (2023).
[44] Zhiyuan Sun, Zihao Li, Xinghao Peng, Xiapu Luo, Muhui Jiang, Hao Zhou, and Yinqian Zhang. 2024. DoubleUp Roll: Double-spending in Arbitrum by Rolling It Back. In *CCS*.
[45] Taro Tsuchiya, Liyi Zhou, Kaihua Qin, Arthur Gervais, and Nicolas Christin. 2024. Blockchain Amplification Attack. *arXiv* (2024).
[46] Sam Werner, Daniel Perez, Lewis Gudgeon, Ariah Klages-Mundt, Dominik Harz, and William Knottenbelt. 2022. Sok: Decentralized finance (defi). In *ACM AFT*.
[47] Gavin Wood. 2014. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum yellow paper* (2014).
[48] Aviv Yaish, Kaihua Qin, Liyi Zhou, Aviv Zohar, and Arthur Gervais. 2024. Speculative Denial-of-Service Attacks In Ethereum. In *USENIX Security*.
[49] Youngseok Yang, Taesoo Kim, and Byung-Gon Chun. 2021. Finding consensus bugs in ethereum via multi-transaction differential fuzzing. In *OSDI*.
[50] Polygon zkEVM. 2023. A disclosed vulnerability in Polygon zkEVM. https://x.com/0xiczc/status/1662090451493740545.