XChainWatcher: Identifying Anomalies in Cross-Chain Bridges

André Augusto
INESC-ID & IST, University of Lisbon
Lisbon, Portugal

Rafael Belchior
INESC-ID & Blockdaemon
Dublin, Ireland

Jonas Pfannschmidt
Blockdaemon
Dublin, Ireland

André Vasconcelos
INESC-ID & IST, University of Lisbon
Lisbon, Portugal

Miguel Correia
INESC-ID & IST, University of Lisbon
Lisbon, Portugal

Abstract

Cross-chain bridges are a type of middleware for blockchain interoperability that supports the transfer of assets and data across blockchains. However, several of these bridges have vulnerabilities that have caused 3.2 billion dollars in losses since May 2021. Some studies have revealed the existence of these vulnerabilities, but there is little quantitative research available and there are no safeguard mechanisms to protect bridges from such attacks. Furthermore, no studies are available on the practices of cross-chain bridges that can cause financial losses. We propose XChainWatcher (Cross-Chain Watcher), a modular and extensible logic-driven anomaly detector for cross-chain bridges. It operates in three main phases: (1) decoding events and transactions from multiple blockchains, (2) building logic relations from the extracted data, and (3) evaluating these relations against a set of detection rules. Using XChainWatcher, we analyze data from two previously attacked bridges: the Ronin and Nomad bridges. XChainWatcher was able to successfully identify the transactions that led to losses of $611M and $190M (USD) and surpassed the results obtained by a reputable security firm in the latter. We not only uncover successful attacks, but also reveal other anomalies, such as 37 cross-chain transactions (*cctx*) that these bridges should not have accepted, failed attempts to exploit Nomad, over $7.8M worth of tokens locked on one chain but never released on Ethereum, and $200K lost by users due to inadequate interaction with bridges. We provide the first open dataset of 81,000 *cctxs* across three blockchains, capturing more than $4.2B in token transfers.

CCS Concepts

• **Security and privacy** → **Intrusion detection systems**; • **Computer systems organization** → *Dependable and fault-tolerant systems and networks.*

Keywords

Blockchain, Interoperability, Cross-Chain, Anomaly Detection

1 Introduction

In recent years, there has been a remarkable adoption of *blockchain interoperability* through the use of cross-chain mechanisms [8, 16]. The most popular mechanisms are *cross-chain bridges* (or simply *bridges*). Bridges serve as an essential middleware in the blockchain ecosystem, connecting decentralized applications across various blockchains, and facilitating the transfer and exchange of assets.

In the Ethereum ecosystem, numerous bridges connect Ethereum to other blockchains, such as rollups and sidechains. Native bridges support rollups – Layer 2 solutions designed to enhance Ethereum's scalability while inheriting its security (e.g., [10, 30, 42, 49]). In contrast, non-native bridges connect Ethereum to sidechains, which employ an independent consensus mechanism and do not inherit Ethereum's security guarantees. Despite these differences, the primary goal remains the same: enabling decentralized applications on Ethereum to expand to faster and more cost-efficient blockchains.

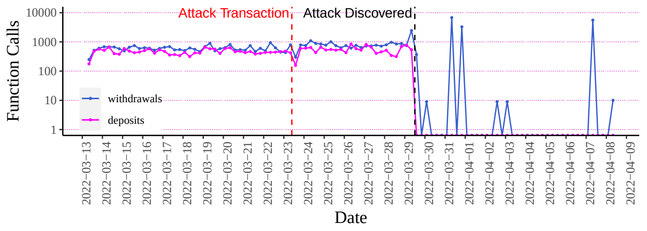

Figure 1: Number of function calls to withdraw and deposit funds into/from the Ronin blockchain through the Ronin bridge. The attack was only discovered six days later, causing deposit calls to drop to zero. Each data point represents the total function calls in periods of 6 hours.

The cross-chain ecosystem is growing. During 2023, cross-chain protocols raised more than $500 million (USD) in investment rounds [18, 22, 24, 44, 51, 65], and processed millions of cross-chain transactions (*cctx*) daily [23]. In November 2024, non-native cross-chain bridges had a total value locked (TVL) of around $11 billion [67] and native bridges $39 billion, highlighting the growing interest in the technology, despite its numerous hacks: from May 2021 to August 2024, hackers stole a staggering amount of $3.2 billion in cross-chain bridges [8], and have indirectly caused losses of several tens of millions in other Decentralized Finance (DeFi) protocols due to on-chain activity and token valuations plummeting [56].

Not even extensively audited bridges are immune to vulnerabilities [23]. In fact, several bridges have been exploited multiple times [7, 12, 43, 54, 55, 59–62, 68]. Moreover, protocols take too long to react to an attack [11, 59, 61], suggesting that teams may not be sufficiently prepared to address integrity breaches, possibly due to **lack of prior awareness, observability, monitoring, or good *SecOps* practices** [8]. In 2022, the Ronin Bridge was attacked, but the team discovered the attack only 6 days later (cf. Figure 1). In the most recent attack, which also targeted the Ronin bridge in August 2024, the team reported that the bridge was paused only about 40 minutes after the first malicious on-chain activity was detected [62]. Even if attacks cannot be reversed, it is possible to work on swift detection and protocol stoppage to avoid further exploitation (in

Section 5.2.5 we show that there were 382 attacking transactions in the Nomad bridge attack). Developing effective incident response frameworks is crucial for efficient attack identification and response to malicious activity, with the great potential of minimizing losses.

Some authors have studied cross-chain security, listing and systematizing vulnerabilities and attacks across the relevant cross-chain layers [8, 25, 35, 69, 72, 73]. However, *quantitative studies* with real-world data are still lacking. Variations in contract implementations, security models [8], bridging models [13] (e.g., *lock-mint*, *burn-mint*, *lock-unlock*), and token types across different chains make it difficult to monitor and safeguard these systems consistently. Additionally, the use of intermediary protocols (e.g., *bridge aggregators* [41, 66]) and the extraction of data from various sources (e.g., transaction data or events emitted by contracts) increase the technical challenges of performing these studies.

To address this gap, we present a middleware monitoring layer that detects anomalies in cross-chain bridges and validates them through an empirical study of real-world exploits. Many academic works suggest that anomaly detectors can be trained automatically from live-captured normal/good behavior. This approach proves impractical for cross-chain bridges because they are inherently complex systems, not formally specified, often misused, and are constantly being attacked due to the large amounts moved. Therefore, there is no hope of live capturing a clean and labeled dataset of cross-chain transactions that can be used to train anomaly detection models automatically – and there are no open-source alternatives at the moment. In this work, to overcome this challenge, we rely on a manual definition of *cross-chain rules* to characterize the expected behavior in a cross-chain bridge. These rules form the basis of our anomaly detection mechanism, enabling us to detect known and undocumented anomalies. This paper, along with the open and labeled dataset provided, establishes the first baseline for future automated approaches to cross-chain anomaly detection.

There is a large variety of bridge solutions in the industry, so designing an anomaly detection tool that fits every scenario is challenging. Therefore, in this paper, we focus on modeling and evaluating our solution for cross-chain bridges that connect Ethereum to its sidechains [29], the most valuable blockchain ecosystem except for Bitcoin (Ethereum alone has a market cap of around \$200 billion). The communication between Ethereum and a sidechain with a cross-chain bridge involves two steps: users first *Deposit* assets transferring tokens from Ethereum to the sidechain, and later *Withdraw* funds, transferring the assets back to Ethereum. While there are some nuances and rules that may need fine-tuning, the rationale followed in this paper can be applied to other interoperability projects (e.g., arbitrary message-passing protocols).

This paper provides the following contributions:

(1) **XChainWatcher.** The first open-source framework for performing anomaly detection in cross-chain bridges, capable of detecting known attacks and other anomalies that harm users and protocol operators. XChainWatcher provides the pipeline for decoding event and transaction data, building logic relations, and evaluating them against the proposed anomaly detection rules.
(2) **Quantitative study on cross-chain security.** We perform an anomaly detection analysis on data extracted from bridge contracts deployed on Ethereum, Gnosis, and Moonbeam – 3 EVM-based blockchains. We release the first open dataset of cross-chain transactions, consisting of over 81,000 *cctxs*, moving more than \$4.2B in token transfers.
(3) **New anomalies.** Through the analysis of the anomaly detection results, we identify past attacks and also new anomalies in cross-chain bridges due to unintended behavior from users and protocols.

The paper is structured as follows. Section 2 provides background information on blockchain, smart contracts, and cross-chain bridges. Section 3 details the design of XChainWatcher. Sections 4 and 5 present the experimental setup and the anomaly detection results. Section 6 outlines the discussion, limitations, and future work. Finally, the related work and conclusions are given in Sections 7 and 8. All monetary values presented in this paper are in US dollars.

2 Background

We provide an overview of the necessary background to understand the remainder of this paper.

2.1 Blockchain and Smart Contracts

Consider a blockchain B a sequence of blocks $\{B_1, B_2, ..., B_n\}$, where n is the nth block such that each block is cryptographically linked to the previous one. Each block contains a root of the current state trie, which holds the state of the blockchain, represented as key-value pairs $S(B_x) = \langle key, value \rangle$. Each key represents an account – either an *Externally Owned Account* (EOA) controlled by a cryptographic key pair or a *Contract Account*. The latter contains code that enforces the so-called smart contracts that execute in the native virtual machine of the blockchain, e.g., the Ethereum Virtual Machine (EVM). The execution of tx in B_x changes the state $S(B_x) \xrightarrow{tx} S'(B_x)$. Examples of state changes include triggering the execution of smart contracts or actions natively supported by the blockchain, such as transferring native currency or triggering internal transactions (which can recursively trigger additional state changes). Smart contracts enable the execution of code, which can define tokens – by following common interfaces, such as the ERC-20 [26] or ERC-721 [27] – or arbitrary logic, such as validating *Merkle proofs* or digital signatures. Code execution may emit events that can be understood as execution logs (also called *topics* in the context of transaction receipts). Events are usually representations of state changes in a certain smart contract.

2.2 Cross-Chain Bridge Model

Contrary to most DeFi dApps, bridges span over two or more blockchains, rather than being confined to one. Figure 2 illustrates the components of a cross-chain bridge, showing a source chain ($\mathcal{S}$) and a target chain ($\mathcal{T}$) with a one-way token deposit flow ($\mathcal{S} \rightarrow \mathcal{T}$).

To perform a cross-chain transfer depositing tokens into another blockchain, a user u_s issues a transaction that is added to the blockchain at timestamp $t1$, tx_{t1}, on a source chain S to escrow tokens. This transaction can directly target a bridge contract or an intermediary protocol that calls internally a bridge contract. The bridge contract subsequently triggers an internal call to the token contract τ_s associated with the token that u_s wishes to bridge.

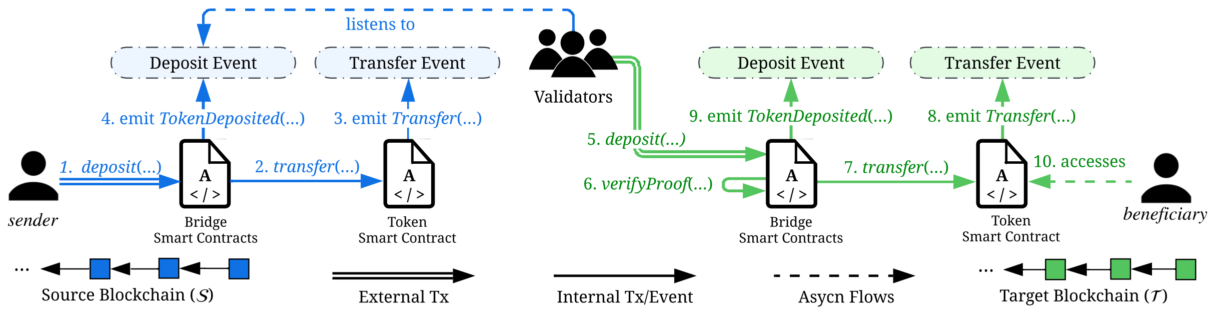

Figure 2: The flow of a token transfer from a source blockchain ($\mathcal{S}$) to a target blockchain ($\mathcal{T}$), using a cross-chain bridge.

This call results in the creation of a commitment $\pi_S(u_s, \tau_s, q)$, indicating that q units (quantity) of token τ_s held by u_s have been escrowed. This commitment reflects either the locking or burning of tokens, leading to a state change in S, which will be a part of the blockchain's new state S_{t2}:

$$S_{t2} \supseteq (S_{t1} \oplus \pi_S(u_s, \tau_s, q))$$

The state change triggers an event emission from the token contract τ_s. In this paper, we focus on the ERC20 token standard, thus, on fungible tokens [6]. Depending on the bridging model, escrowing tokens can be implemented by transferring tokens to a bridge-controlled address (*lock* model) or to the null address (*burn* model). The ability to handle this dichotomy allows this analysis to be agnostic of the bridging model. Therefore, in a *lock-unlock* model, the event can be represented as:

$$\epsilon_{\tau_s,S} = Transfer(u_s, bridge, q)$$

in the form *(from, to, value)*. In a *burn-mint* model, tokens are instead burnt – i.e., transferred to the null address (*0x000*) – even though not as common, this is a more secure approach because it avoids creating a honeypot of locked assets in $\mathcal{S}$. If u is trying to bridge native tokens in $\mathcal{S}$, there is no call to the *Transfer* method of an ERC20 token contract, but the commitment is in the form of a native transfer of tokens in $\mathcal{S}$ – i.e., $tx_{t1}.value = q$. Once the commitment is created in $\mathcal{S}$, the bridge contract emits an event with commitment data and some additional parameters, such as a unique identifier for the deposit, the beneficiary u_t (the user to which the tokens are intended in $\mathcal{T}$), and the token in $\mathcal{T}$ (τ_t) that represents the same token as τ_s:

$$\epsilon_{bridge,S} = TokenDeposited(deposit\_id, \pi_S(u_s, \tau_s, q), u_t, \tau_t)$$

This event is captured by validators (or relayers) – off-chain entities responsible for enabling cross-chain interoperability. Upon detecting an event on chain S, validators initiate a transaction on the target chain T to trigger a state change, such that the commitment is part of the new state T_{t3}:

$$\mathcal{T}_{t3} \supseteq (\mathcal{T}_{t2} \oplus \pi_{\mathcal{T}}(u_t, \tau_t, q))$$

This final commitment $\pi_{\mathcal{T}}$ represents either the minting or unlocking of tokens within τ_t. Similar to the source chain process, the token contract on $\mathcal{T}$ emits a Transfer or Mint event. The Transfer event can be represented as

$$\epsilon_{\tau_t,\mathcal{T}} = Transfer(bridge, u_t, q)$$

where tokens are being unlocked (i.e., transferred from the bridge contract). The bridge contract also emits an event accordingly:

$$\epsilon_{bridge,\mathcal{T}} = TokenDeposited(deposit\_id, \pi_{\mathcal{T}}(u_t, \tau_t, q)).$$

It is crucial that this commitment in $\mathcal{T}$, and subsequent emission of events in both contracts, is only created if **(1)** $\phi(\pi_S(u_s, \tau_s, q))$ holds true, where ϕ is a commitment verification function on $\mathcal{T}$ that verifies the commitment originating from S; and **(2)** π_S was created in a transaction, in a block that is k blocks deep into blockchain $\mathcal{S}$ and the probability of being reverted is negligible. In this paper, we abstract away the specific implementation details of commitments (e.g., zero-knowledge or Merkle proofs) and focus on the observable state changes. By analyzing state transitions, we can detect anomalies in cross-chain bridges independently of their internal logic, enabling a middleware-level system like XChainWatcher to reason about cross-chain activity in a protocol-agnostic way.

The withdrawal flow ($\mathcal{T} \rightarrow \mathcal{S}$) is the inverse but very similar, thus not represented.The key difference is that, usually, the user triggers the final transaction on $\mathcal{S}$, instead of being the validators managing the process (e.g., [32]). This allows the operator to minimize operational costs because validators are not required to issue Ethereum transactions for every withdrawal request.

2.3 Attacks in Cross-Chain Bridges

Since June 2021, attackers have stolen more than 3.2 billion USD from cross-chain bridges [8, 40]. Hackers target smart contracts that have permission to lock, unlock, mint, or burn tokens. If an attacker gains control of a critical contract – through a bug or a compromised private key [8] – they can execute unauthorized token operations. In a lock-unlock model, attackers exploit bridges in two main ways:

(1) Steal funds held by the bridge contract on $\mathcal{S}$. Those funds represent the current total value locked by users.
(2) Steal existing funds (liquidity) on $\mathcal{T}$ that support the unlocking process.

In the burn-mint or lock-mint models, instead of stealing existing funds, attackers mint (create) tokens out of thin air and transfer them to their addresses. These attacks are classified in the literature into two categories based on the direction of the invalid state changes:

(1) **Forged Deposit Attack:** Attackers claim funds – either unlocking existing tokens or minting new ones – on $\mathcal{T}$ without locking or burning tokens on $\mathcal{S}$.
(2) **Forged Withdrawal Attack:** Attackers withdraw funds on $\mathcal{S}$ – similarly, unlocking existing tokens or minting new ones – without previous burn or lock operations on $\mathcal{T}$.

3 XChainWatcher

XChainWatcher[*] is a logic-based *monitoring system* for cross-chain bridges, built as an open-source framework using Souffle [37] – a high-performance Datalog-inspired engine.

The workflow of XChainWatcher is presented in Figure 3. There are three phases: 1) decoding event and transaction data from blockchains, 2) building a set of logic relations based on the data extracted, and 3) evaluating relations using a set of detection rules. We design XChainWatcher to be generic and extensible so that anyone can integrate support for any bridge. Additionally, the logical rules can be fine-tuned for each supported bridge.

3.1 Logical Relations

We model cross-chain operations by defining a comprehensive set of logical relations (i.e., the cross-chain model) that capture events emitted by smart contracts and static configurations common to bridge protocols. These logical relations form the basis for our analysis. We derived them by thoroughly reviewing the open-source code of cross-chain bridge protocols that connect Ethereum to sidechains, and their documentation. We also directly interacted with some protocols and observed the different state changes that occurred – including Polygon [52], Ronin [63], Omnibridge [32], xDAI Bridge [33], and the Nomad Bridge [48]. These bridges connect Ethereum to multiple sidechains, such as Ronin, Gnosis, Polygon, and Moonbeam. The list of relations (Datalog facts) is in Listing 1.

Contract Events. The `native_deposit` relation records deposit events of native currency on $\mathcal{S}$ through the wrapped version of the native currency (e.g., Wrapped Ether contract on Ethereum). The `native_withdrawal` relation logs native token transfers on the target chain when initiating withdrawal of funds ($\mathcal{T} \rightarrow \mathcal{S}$), also using the contract representing the wrapped version of the native currency. For bridge-specific events, we use `sc_token_deposited` and `tc_token_deposited` to capture token deposits in the bridge contract on the source and target chains, respectively. In parallel, the `tc_token_withdrew` and `sc_token_withdrew` relations track token withdrawal events emitted by the bridge contract from the target and source chains. Finally, the `erc20_transfer` relation logs ERC20 token transfers. We also capture mined blockchain transactions through the `transaction` relation.

Static Configurations. The `bridge_controlled_address` relation lists all addresses controlled by the bridge. The `token_mapping` relation links equivalent tokens across chains – a standard practice in the field [38, 46]. We capture each chain's finality time in the `cctx_finality` relation, modeling the necessary confirmation duration in seconds. Finally, the `wrapped_native_token` relation identifies wrapped versions of native currencies on each chain – i.e., the wrapped version of Ether, the native currency of the Ethereum blockchain, is Wrapped Ether (WETH).

[*]https://github.com/AndreAugusto11/XChainWatcher

3.2 Decoders and Logic Relation Builders

The *Static Configuration Loader* imports static facts from the bridge configuration file[†] – this is information that does not depend on event or transaction data: `bridge_controlled_address`, `token_mapping`, `wrapped_native_token`, and `cctx_finality`. On the other hand, the *Event and Transaction Data Decoder* is designed to be bridge-specific, where the remaining relations are extracted from the data decoded from bridge events. This component can be fine-tuned for each bridge allowing the extension of XChainWatcher to support any protocol.

The input for the latter component is a set of transaction receipts. Each receipt contains the events emitted by all contracts with which the transaction interacted. In many cases, the transaction receipt is sufficient to extract all the facts. In other cases, however, it is not enough, namely, when dealing with native token transfers or when the user uses intermediary protocols to interact with a bridge. In the first case, the *sender* transfers funds natively in a transaction (in the *tx.value* field). In the latter, funds are transferred in internal transactions [5]. In both cases, the transferred value is not accessible by the transaction receipt. In this case, we obtain the transaction data by making a request to an RPC node using the RPC methods *eth_getTransaction* or the *debug_traceTransaction* with the parameter *{"tracer":"callTracer"}* [34] that outputs the execution traces and transferred values.

When decoding data and building the logical relations, each transaction is assumed to potentially emit an unlimited number of events, e.g., when batching operations are involved. The extraction of data from relevant events involves extracting the first element in the *topics* list of the transaction receipt, which is equal to the hash of the event signature. For instance, `topic[0]` for any event with signature `Deposit(address,address,uint256)` is calculated with a hashing function `keccak256("Deposit(address,address,uint256)")`.

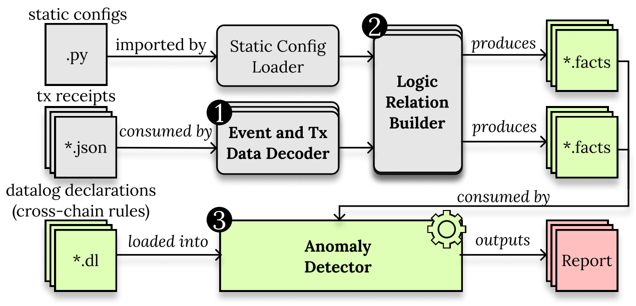

Figure 3: XChainWatcher relies on event and transaction data decoders, logic relation builders, and a Datalog engine to evaluate relations. The *Decoder* and *Logic Relation Builders* are designed to be pluggable and extensible – i.e., anyone can extend XChainWatcher to support other protocols.

3.3 Cross-Chain Rules

Overview. We model the expected behavior of bridges (anomaly-based intrusion detection) instead of modeling specific attacks

[†]An example of configuration file is at https://github.com/AndreAugusto11/XChainWatcher/blob/main/cross-chain-rules-validator/utils/ronin_env.py

```
.decl native_deposit(tx_hash, chain_id, event_index, from, to, amount).
.decl native_withdrawal(tx_hash, chain_id, event_index, from, to, amount).
.decl sc_token_deposited(tx_hash, event_index, deposit_id, beneficiary, dst_token, orig_token, dst_chain_id, amount).
.decl tc_token_deposited(tx_hash, event_index, deposit_id, beneficiary, dst_token, amount).
.decl tc_token_withdrew(tx_hash, event_index, withdrawal_id, beneficiary, orig_token, dst_token, dst_chain_id, amount).
.decl sc_token_withdrew(tx_hash, event_index, withdrawal_id, beneficiary, dst_token, amount).
.decl erc20_transfer(tx_hash, chain_id, event_index, contract, from, to, amount).
.decl transaction(timestamp, chain_id, tx_hash, from, to, value, status, fee).
.decl bridge_controlled_address(chain_id, bridge_address).
.decl token_mapping(source_chain_id, target_chain_id, source_chain_token, target_chain_token).
.decl cctx_finality(chain_id, finality_seconds)
.decl wrapped_native_token(chain_id, token).
```

Listing 1: Definition of the logical relations built by XChainWatcher.

(signature-based intrusion detection) using cross-chain rules [9]. This approach allows us to identify anomalies that have not yet been discovered and that are under the hood of the complexity of analyzing cross-chain data. Each rule enforces a set of validations to determine the validity of events within one or more blockchains. Rules are classified as *isolated* (I) or *dependent* (D). An isolated rule concerns only one blockchain, such as the deposit of tokens in $\mathcal{S}$. In contrast, a dependent rule relies on prior state changes on another blockchain, such as the deposit of tokens in $\mathcal{T}$, which depends on tokens being deposited in $\mathcal{S}$. Each rule is also prefixed with SC, TC, or CCTX to indicate whether it is a check on $\mathcal{S}$, $\mathcal{T}$, or both chains, respectively[‡].

Rule 1 (I). `SC_ValidNativeTokenDeposit` ensures a valid deposit of native tokens by the user in $\mathcal{S}$. This rule specifies a relationship between the transaction issued by the user, the event emitted by the bridge contract, and the event emitted by the contract representing the wrapped version of the native currency. In more detail, the checks are: **(1)** a bridge contract must emit a *Deposit* event; **(2)** there is a non-reverting transaction that transfers the same amount of tokens natively in `tx.value`; **(3)** there is an event emitted by the token contract asserting the creation of a wrapped version of the native currency through a deposit event **(4)** the token contract provided is indeed a version of the native currency of $\mathcal{S}$; **(5)** the validity of the token mappings (i.e., if users are trying to deposit tokens into $\mathcal{T}$ using a different token than what they are using in $\mathcal{S}$); and finally **(6)** the order of the events emitted by each contract (events emitted by token contracts precede events emitted by bridge contracts – cf. Figure 2). In check **(2)** we do not check whether the transaction targets a bridge contract, as it may target an intermediary protocol contract (e.g., a bridge aggregator [66]), which in turn issues an internal transaction to the bridge. We only verify that the deposit event from the token contract must escrow tokens to a valid bridge contract, asserted using `bridge_controlled_address`. This rule ensures that bridge contracts do not emit events asserting the deposit of tokens if the corresponding value was not effectively sent to the bridge – and the other way around. An attack that would have been identified using this rule is [58].

```
SC_ValidNativeTokenDeposit(...args...) :-
  sc_token_deposited(tx_hash, bridge_evt_idx, _, _, dst_token,
                     src_token, dst_chain_id, amount), (1)
  sc_deposit(tx_hash, token_evt_idx, sender, bridge_address, amount), (3)
  transaction(_, src_chain_id, tx_hash, _, sender, _, amount, 1, _), (2)
  token_mapping(src_chain_id, dst_chain_id, src_token, dst_token), (5)
  wrapped_native_token(src_chain_id, src_token), (4)
  bridge_controlled_address(src_chain_id, bridge_address),
  bridge_evt_idx > token_evt_idx. (6)
```

[‡]The complete definition of all rules in the form of Horn Clauses is in https://github.com/AndreAugusto11/XChainWatcher/blob/main/cross-chain-rules-validator/datalog/acceptance-rules.dl

Rule 2 (I). `SC_ValidERC20TokenDeposit` ensures that a valid deposit of ERC20 tokens on the bridge is subject to a series of checks. Specifically, this rule defines a bidirectional relationship $\epsilon_{\tau_s,S} \iff \epsilon_{bridge,S}$ for ERC20 tokens. This means that whenever a state change involves the transfer of ERC20 tokens, the bridge contract must emit an event corresponding to the commitment described by the initial event, and vice versa. The remaining checks presented in *Rule 1* are also enforced.

Failure to comply with Rule 1 or 2 suggests that a user has deposited tokens in the bridge without the bridge recognizing the deposit. Conversely, if a token transfer event occurs, but no corresponding event is emitted by the bridge contract (or value transferred in the transaction), it could signal an attack, where an attacker bypasses the cross-chain logic and steals funds. Examples of attacks that would have been identified using this rule are [1, 2, 68]. Rules 1 and 2 guarantee that the flow 1 – 4 (in blue) in Figure 2 is valid for native and ERC20 tokens, respectively.

```
SC_ValidERC20TokenDeposit(...args...) :-
  sc_token_deposited(tx_hash, bridge_event_index, _, _, dst_token,
                     src_token, dst_chain_id, amount),
  erc20_transfer(tx_hash, src_chain_id, token_event_index, src_token,
                 _, bridge_addr, amount),
  transaction(timestamp, src_chain_id, tx_hash, _, from, _, "0", 1, _),
  token_mapping(src_chain_id, dst_chain_id, src_token, dst_token),
  bridge_controlled_addr(src_chain_id, bridge_addr),
  bridge_event_index > token_event_index.
```

Rule 3 (I). `TC_ValidERC20TokenDeposit` outputs valid token deposits in $\mathcal{T}$. It captures the valid relation between the event emitted by the bridge contract and the respective token contract in which tokens are being unlocked/minted. Similarly to *Rules 1 and 2*, there is a bidirectional relationship $\epsilon_{\tau_t,T} \iff \epsilon_{bridge,T}$. In this instance, tokens are always transferred in the context of a token contract and never natively, thus, we do not need a rule for native token transfers. These events must match variables such as the sender, beneficiary, token, amount being transferred, and order of events. This rule guarantees that flow 5 – 9 (in green) in Figure 2 is valid for any token that is deposited.

```
TC_ValidERC20TokenDeposit(...args...) :-
  tc_token_deposited(tx_hash, bridge_event_index, deposit_id,
                     beneficiary, dst_token, amount),
  erc20_transfer(tx_hash, chain_id, token_event_index, dst_token,
                 bridge_addr_2, beneficiary, amount),
  transaction(_, chain_id, tx_hash, _, _, bridge_addr_1, "0", 1, _),
  bridge_controlled_addr(chain_id, bridge_addr_1),
  bridge_controlled_addr(chain_id, bridge_addr_2),
bridge_event_index > token_event_index.
```

Rule 4 (D). `CCTX_ValidDeposit` correlates events from both $\mathcal{S}$ and $\mathcal{T}$, cross-referencing token deposit events across these chains to generate a list of valid *cctxs*. A valid cross-chain transaction for a deposit requires that all parameters from events on both chains be consistent (e.g., token amounts, sender, beneficiary). Furthermore, the causality between these events must be preserved (e.g., the transaction on $\mathcal{T}$ occurs after the transaction on $\mathcal{S}$). Formally, there is a dependency between the commitment on $\mathcal{T}$ and the commitment on $\mathcal{S}$, as well as the corresponding events: $\epsilon_{bridge,T} \Longleftarrow \epsilon_{bridge,S}$. Since this rule spans multiple blockchains, we must consider their finality times, which we enforce through the `cctx_finality` fact. Failure to comply with this rule indicates, for example, that tokens were moved on only one side of the bridge, such as in the **Forged Deposit Attack**. This rule would have identified cross-chain hacks such as [3, 7, 21, 54, 57]. This rule guarantees that the entire flow of Figure 2 is valid.

```
CCTX_Deposit(...args...) :-
  TC_ValidERC20TokenDeposit(...args...),
  (
      SC_ValidERC20TokenDeposit(...args...) ;
      SC_ValidNativeTokenDeposit(...args...)
  ),
  cctx_finality(src_chain_id, src_chain_finality),
  token_mapping(src_chain_id, dst_chain_id, src_token, dst_token),
  src_chain_ts + src_chain_finality <= dst_chain_ts.
```

We also model the token withdrawal process ($\mathcal{T} \rightarrow \mathcal{S}$). Given its similarity to the deposit of tokens, we do not provide a detailed explanation of the related rules. Instead, we briefly overview their goal and definitions. *Rule 5 (I).* `TC_ValidNativeTokenWithdrawal` ensures that native token withdrawals on the target chain $\mathcal{T}$ are valid. Specifically, a withdrawal must correspond to a *Withdraw* event emitted by the bridge contract and a non-reverting transaction locking or burning funds. This rule is essentially the inverse of Rule 1, applying similar checks but in the withdrawal context. Rule 5 would have identified one attack [12]. *Rule 6 (I).* `TC_ValidERC20TokenWithdrawal` applies to ERC20 token withdrawals on $\mathcal{T}$, ensuring that any withdrawal event emitted by the bridge contract matches a corresponding *Transfer* event for the ERC20 tokens being withdrawn. This rule is analogous to Rule 2, and would have identified one attack [55]. *Rule 7 (I).* `SC_ValidERC20TokenWithdrawal` extends these checks to ERC20 withdrawals on the source chain $\mathcal{S}$, mirroring Rule 3's checks in the reverse direction. Finally, *Rule 8 (D).* `CCTX_ValidWithdrawal` links withdrawal events on $\mathcal{T}$ and $\mathcal{S}$, verifying that all parameters across the chains match and enforcing the correct causal relationship between events, similar to Rule 4 for deposits but in reverse. Rule 8 would have identified multiple attacks such as the **Forged Withdrawal Attack** [11, 31, 59–61].

While it is impossible to design generic rules that allow for every existing bridge, we highlight that these rules can be easily extended/fine-tuned to find anomalies in other bridges.

4 Evaluation Methodology

We evaluate XChainWatcher using the cross-chain rules presented in the last Section and detail the anomaly detection analysis in the Ronin and Nomad bridges.

4.1 Data Sources

We selected two previously exploited bridges to analyze the capabilities of XChainWatcher and the cross-chain rules: the **Nomad bridge** and the **Ronin bridge**. This selection allows us to test XChainWatcher against bridges that have suffered attacks and whose architecture and security assumptions differ (§4.1.2 and §4.1.3). We used Blockdaemon's Universal API [19] to retrieve blockchain data from the Ethereum mainnet. We implemented a fallback to native RPC methods when the API could not provide the necessary data (namely *eth_getLogs* and *eth_getTransactionReceipt*). Additionally, we used these methods to extract data from Moonbeam and Ronin blockchains that are not supported by the API. We gathered addresses of interest, including various versions of deployed contracts through documentation and analysis of the source code of each bridge[§].

4.1.1 Time Frames. Since we adopt an anomaly-based intrusion detection approach (instead of signature-based), which tends to have a high false positive rate [9], we choose to evaluate protocols over smaller time frames. This approach enables us to analyze each flagged anomaly individually, determining whether it results from a modeling error or represents a previously unidentified anomaly in cross-chain bridges. Additionally, we want to study particular attacks and their consequences – involving bigger timeframes would involve significantly more data, without necessarily providing additional relevant insights. Table 1 lists the timestamps used for data extraction. We select an interval of interest for both bridges that includes the attack dates, denoted $[t_1; t_2]$. To avoid missing cross-chain transactions occurring near the start and end of the interval of interest, we incorporate additional intervals before and after that interval ($[t_0, t_1[$ and $]t_2, t_3]$). This is relevant, for example, when there is a deposit of tokens in $\mathcal{S}$ near t_2 and the corresponding transaction in $\mathcal{T}$ falls outside $[t_1; t_2]$ (within $]t_2; t_3]$).

Table 1: Timeframes of Relevance for Data Extraction

	t_0	t_1	t_2	t_3
Nomad Bridge	–	Jan 11, 2022 (1641905876)	Dec 15, 2022 (1671062400)	Jul 31, 2024 (1722441775)
Ronin Bridge	Sep 13, 2021 (1631491200)	Jan 1, 2022 (1640995200)	Apr 28, 2022 (1651156446)	Jul 31, 2024 (1722441775)

Note: The interval of interest is $[t_1, t_2]$. The table presents dates and corresponding Unix timestamps in parentheses. The Nomad and Ronin bridges were attacked on Aug 2, 2022 and Mar 22, 2022, respectively.

To analyze the Nomad bridge, we extracted 20,551 transactions from Ethereum, 16,737 transactions from Moonbeam, and 20,308 transactions from/to other blockchains, which were only used for data analysis. In the additional period, we collected additional 1,774 transactions on Ethereum, from the latest versions of the deployed bridge contracts. On the Ronin bridge, we extracted 72,820 transactions from Ethereum and 75,102 from Ronin. In the additional period, we collected additional 516,657 and 151,325 transactions on Ethereum and Ronin, respectively. The data collection totaled 875,274 transactions across the studied bridges and blockchains.

[§]an example for the Nomad bridge is in https://anonymous.4open.science/r/XChainWatcher-B5F1/cross-chain-rules-validator/utils/nomad_env.py

4.1.2 *Nomad Bridge.*

The Nomad bridge supports six blockchains. We select the most active blockchains in terms of bridge usage: Ethereum ($\mathcal{S}$) and Moonbeam ($\mathcal{T}$). The bridge operates based on fraud proofs [48] – i.e., a set of relayers transfers state proofs between blockchains, and the watchers (which are off-chain parties) have a predefined time window to challenge the relayed data. The data is optimistically accepted if no challenge is received within this window. According to the project documentation, this time window was set to 30 minutes [47] during the selected time frame. The main bridge contract on Moonbeam was deployed on January 11, 2022 (t_1). Since we start our analysis on this date, there is no t_0 ($t_0 = t_1$). The Nomad bridge was exploited on August 2, 2022, causing the bridge contracts to be paused until December 15, 2022. After this date, new transactions depositing tokens into the bridge on Ethereum started being reissued. In $]t_2; t_3]$ we only collect withdrawals in Ethereum to match all the withdrawal requests performed on Moonbeam in $[t_1; t_2]$ that did not complete.

4.1.3 *Ronin Bridge.*

The Ronin bridge connects Ethereum ($\mathcal{S}$) and the Ronin blockchain ($\mathcal{T}$) and operates based on a multi-signature of trusted validators [63] – i.e., deposits and withdrawals are executed when a threshold of validators attests the validity of the action on the origin blockchain (be it a lock or burn of tokens). The Ronin bridge was deployed in early 2021, and the attack occurred on March 22, 2022. The interval of interest spans approximately four months, from the start of 2022 to April 28, 2022, when the main bridge contract on Ronin was paused (0xe806...19fd). To capture incomplete withdrawals on $\mathcal{T}$ before the attack, we analyze additional data on Ethereum between $]t_2; t_3]$. This required scanning for events in the newer version of the main bridge contract (0x6419...af08), which was deployed on Ethereum after the attack on June 22, 2022. Finally, based on the same logic as above, we also captured additional deposits in Ethereum to capture cross-chain transactions initiated in $[t_0; t_1[$, whose deposit in Ronin is at the beginning of $[t_1; t_2]$.

4.2 Experiment Setup

We present the performance analysis of XChainWatcher, using the rules defined in Section 3.3, in finding anomalies in the Nomad and Ronin bridges. We divide the analysis into two main processes: 1) decoding data and building the Datalog facts, and 2) running the cross-chain rules to find anomalies. We computed the results on a MacBook Pro with a 14-core M3 Max processor and 36GB of RAM.

4.2.1 *Decoding and Extracting Data.*

Figure 4 illustrates the cumulative distribution of transaction receipts processing time, differentiating between transfers of native and non-native funds in each bridge. Additional metrics are provided in Table 2. Transactions transferring native tokens take longer because the transaction receipt is not enough to get `tx.value`, thus requiring at least one extra time-consuming RPC call. Furthermore, for native value transfers, some transactions exhibited unusually high latencies (e.g., 6.5% exceeded 10 seconds, with one instance reaching 138.15 seconds). This delay mainly results from the high latency of the *debug_traceTransaction* method when making RPC requests to an Ethereum node [4]. Not only is this a resource-intensive method, but multiple timeouts caused various retries to retrieve the data. A more stable RPC node connection – ideally hosting one alongside XChainWatcher – and extending the timeout period for these resource-intensive methods would significantly reduce the latency, dropping towards the median (0.35 and 0.78 seconds, for Ronin and Nomad, respectively).

Table 2: Facts extraction latency (in seconds) per token type

Bridge	Token type	size	min	max	avg	median	std
Ronin	native	468,997	0.18	138.15	1.82	0.35	4.70
	non-native	347,580	$3.81\text{x}10^{-6}$	3.65	0.28	0.23	0.26
Nomad	native	7,656	0.16	8.78	0.89	0.78	0.46
	non-native	51,702	$3.81\text{x}10^{-6}$	5.83	0.26	0.19	0.28

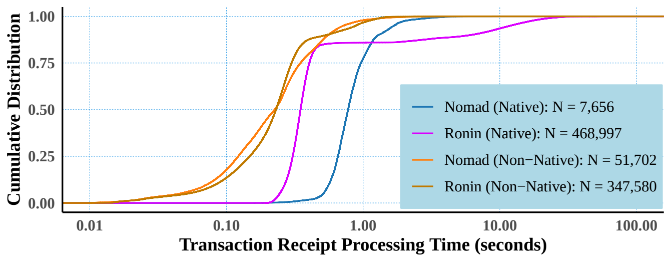

Figure 4: Cumulative distribution of transaction receipt processing time, reflecting the latency of extracting all facts from a transaction receipt for native and non-native token transfers.

4.2.2 *Executing the Cross-Chain Rules.*

Based on the data extracted, we run the detection rules to identify anomalies. In addition to the rules presented in Section 2.2, we implemented additional Datalog rules to compare datasets and perform a more fine-grained analysis – we created 30 logical rules in total, available in the linked repository. The total time consists of decoding the data and building the logic relations plus the execution of the detection rules. For the Ronin bridge, the model processed more than 1,570,000 data tuples, producing results, on average, in 3.58 seconds, while for the Nomad bridge, it analyzed more than 200,000 data tuples and generated results in 0.51 seconds¶.

4.2.3 *Preliminary Findings of Cross-Chain Transactions.*

A byproduct of our work is a dataset of cross-chain transactions captured by rules 4 and 8 – i.e., data from two blockchains that are linkable and represent valid cross-chain token transfers. Figure 5 presents the latency associated with each cross-chain transaction identified on the Nomad bridge (the Ronin bridge data was omitted for the sake of space but provides the same insights).We call out two main insights: 1) the dispersion of the latency of withdrawals is much higher, which is due to the users being the ones responsible for issuing the final transaction on the destination blockchain, contrary to the deposit process (cf. Section 2.2) – the slowest *cctx* took more than 5 months to complete (0x8afe...85bb in $\mathcal{T}$ and 0xdfaa...e3cb in $\mathcal{S}$); 2) all *cctx*s identified by XChainWatcher start at the 30-minute mark, which aligns with our expectations, as the Nomad fraud-proof window is set for this period enforced by `cctx_finality`;

¶detailed results can be found in https://anonymous.4open.science/r/XChainWatcher-B5F1/profiler_html/ronin.html

Table 3: Anomaly detection results, within $[t_1; t_2]$, using the cross-chain rules defined in Section 3.3

	Nomad Bridge		Ronin Bridge	
Logical Rule (cf. Section 3.3)	Captured Records	Anomalies Detected	Captured Records	Anomalies Detected
1. `SC_ValidNativeTokenDeposit`	7,187	0	38,462	0
2. `SC_ValidERC20TokenDeposit`	4,223	39 (14 phishing attempts + 25 transfers to bridge)	5527	83 (3 phishing attempts + 80 transfers to bridge)
Total Value in Transfers to Bridge		**$93.86K**		**$113.00K**
3. `TC_ValidERC20TokenDeposit`	11,417	0	43,990	0
4. `CCTX_ValidDeposit`	11,404	19[*]	43,979	10[*]
5. `TC_ValidNativeTokenWithdrawal`	464	0	0	0
6. `TC_ValidERC20TokenWithdrawal`	4,846	10 (3 unparseable addresses + 7 attack attempts)	35,413	0
7. `SC_ValidERC20TokenWithdrawal`	4,869	2 (2 phishing attempts)	25,470	1 (1 phishing attempt)
8. `CCTX_ValidWithdrawal`	4,482	729[*]	22,830	12,546[*]

Recall that the rules capture expected behavior. Therefore, the anomalies presented are the result of comparing each event emitted by each contract, with being captured or not by the corresponding rule that should have captured it.

[*] Table 4 presents a detailed explanation of these anomalies. Each anomaly is categorized based on the underlying reasons that led to its occurrence.

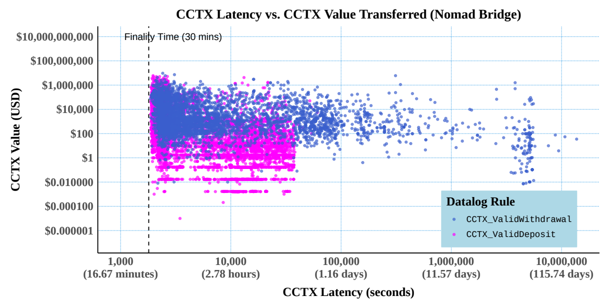

Figure 5: Correlation between the latency and value transferred in each cctx completed before the attack.

5 Anomaly Detection Results

Hereafter, we present the results of the anomaly detection rules. Table 3 shows the number of detected anomalies and the reasons behind each one. Section 5.1 discusses the anomalies found by *isolated* rules (1-3 and 5-7), and Section 5.2 presents and discusses the anomalies found by *dependent* rules (4 and 8).

5.1 Isolated Rules (Rules 1-3 and 5-7)

We start by analyzing the anomalies detected by rules 1-3 and 5-7.

5.1.1 Depositing on $\mathcal{S}$. On the Nomad bridge, we detected 7,187 native value transfers (`sc_deposit`), 4,263 token deposits (`erc20_transfer`), and 11,411 TokenDeposited events emitted by the bridge contract (`sc_token_deposited`), which reveals that 39 value transfers did not have a corresponding bridge event emitted. Further analysis showed that 14 of these transactions are phishing attempts, characterized by numerous events emitted by tokens marked on block explorers as having a bad reputation (e.g., 0x88fc...864a). The remaining 25 transactions were single-event transactions that called the *Transfer* function of multiple reputable ERC20 tokens, with a total of approximately **$93.86K** sent to the bridge without triggering a cross-chain transfer (e.g., 0x7e4e...8d88). On the Ronin bridge, we identified 83 unmatched value transfers, in which 3 were related to phishing attempts and 80 were also random transfers of value to the bridge contract, which accounts for **$113.00K** (e.g., 0xe898...148d).

Finding 1. Attackers use low-value tokens, usually with the name of known tokens, to interact with bridge contracts. These practices are considered phishing attacks, in which users can be misled into using fake tokens to increase their trading value.
Finding 2. Over $206K worth of reputable ERC-20 tokens were sent directly to bridge addresses without using protocol contracts. Despite warnings from DeFi platforms about potentially irreversible losses, this risky behavior appears common.

5.1.2 Depositing on $\mathcal{T}$. We found no anomaly in the process of depositing tokens in $\mathcal{T}$.

5.1.3 Withdrawals on $\mathcal{T}$. In the Nomad bridge, we identified three transactions accepted by the bridge where funds were withdrawn to unintended Ethereum addresses due to being wrongly formatted – they interacted with the Nomad bridge contract using a 32-byte string instead of a 20-byte Ethereum address in the *beneficiary* address field. Therefore, this leads to unparseable data from our tool (the parser is programmed to parse only valid 20-byte addresses). We further discuss this anomaly in Section *5.2.2 Invalid Beneficiary Addresses*. Beyond these three anomalies, we discovered seven transactions from a single address attempting to exploit the bridge using different inputs in the "token" field. The attacker first attempted to provide the address of a malicious smart contract as a token, probably to gain control over the bridge (0x56e6...afe1). In the following 3 transactions, the attacker tried to withdraw funds using a newly created contract that was not mapped to a token in $\mathcal{S}$ (e.g., 0xebd6...bfa9 with token 0x2422...Aefb), in an attempt to have tokens minted on $\mathcal{S}$. Finally, in the latter two, the user attempted to withdraw funds from a (fake) token contract called *Wrapped ETH* (0xcbb4...b91F), to unlock real ETH on Ethereum (e.g., 0x7cd7...03f1). Fortunately, these transactions reverted and all attempts failed. On the Ronin bridge, we identified two events emitted by the bridge contract without a match on `erc20_transfer` or `sc_withdrawal`. These were trying to withdraw unmapped tokens from $\mathcal{T}$ to $\mathcal{S}$, and therefore no tokens were moved, even though the bridge emitted a *Withdraw* event.

Finding 3. Attackers interact with bridge contracts providing fake tokens with symbols or names equal to reputable tokens, in an attempt to deceive the bridge to unlock real funds on the destination blockchain.

5.1.4 *Withdrawals on $\mathcal{S}$.*

The analysis highlights 3 events where funds were transferred from a bridge address without emitting corresponding bridge events: 2 in Nomad and 1 in the Ronin bridge. These instances were linked to phishing attempts and marked accordingly in block explorers (e.g., 0x3587...39ca and 0x78b6...2766).

5.2 Dependent Rules (Rules 4 and 8)

Now, we analyze the results of the *Dependent* rules (4 and 8). Recall from Section 3.3 that Rules 4 and 8 capture linked state changes across blockchains – i.e., for a record to be accepted by these rules, there must be a set of events on both sides of the bridge that are matched. In addition, `cctx_finality` and `token_mapping` must be guaranteed. For example, there may be a valid deposit of tokens in $\mathcal{S}$ captured by `SC_ValidERC20ValidDeposit`. However, no correspondence is found on $\mathcal{T}$, which signals that the protocol is not working as intended (e.g., no availability). In this case, the record of `SC_ValidERC20ValidDeposit` is said to be "unmatched", since it did not match any event on the other blockchain that complies with Rule 4 `CCTX_Deposit`. Table 4 dissects the anomalies detected in Table 3 for `CCTX_ValidDeposit` and `CCTX_ValidWithdrawal`. As an example, Table 3 shows that 19 anomalies have been detected using `CCTX_ValidDeposit`. Table 4 clarifies that 6 of these anomalies are deposits of tokens on $\mathcal{S}$ that did not have a correspondence on $\mathcal{T}$, and 13 are the opposite – deposits of tokens on $\mathcal{T}$ that did not have any prior correspondence on $\mathcal{S}$.

5.2.1 *Cross-Chain Finality Violations.*

One of the most intriguing findings in this paper is the identification of 37 violations of cross-chain rules – 5 on the Nomad bridge and 32 on the Ronin bridge – which were accepted by both bridges at the time, transferring a total value of **$1.3K** and **$667K**, respectively. In the Nomad bridge, 5 instances from `SC_ValidERC20TokenDeposit` and 5 instances from `TC_ValidERC20TokenDeposit` matched each other but were not captured by `CCTX_ValidDeposit` – i.e., even though there were valid commitments on both sides of the bridge, XChainWatcher did not consider this a valid deposit. Similarly, on the Ronin bridge, 10 events were emitted on each side that did not comply with a valid deposit (failed `CCTX_ValidDeposit`), and 22 events on each side that did not comply with a valid withdrawal (`CCTX_ValidWithdrawal`). Figure 6 for the Nomad bridge demonstrates why these events were not captured by `CCTX_ValidDeposit` and `CCTX_ValidWithdrawal`. When depositing tokens using Nomad, in the fastest *cctx*, the time difference between the initial deposit in $\mathcal{S}$ (0xeb06...0fea) and the corresponding deposit on $\mathcal{T}$ (0x2cdc...ef0c) was as short as 87 seconds, approximately 20 times less than the required fraud-proof window. This finding is particularly concerning because it implies that the security mechanisms of the bridge were bypassed. Not only did it fail to comply with the fraud-proof time window, but it was very close to the finality period of the source chain (Ethereum) at the time of the attack – before "The Merge" [28] was around 78 seconds. On the Ronin bridge, the fastest deposit took 66 seconds (0x4688...cdf3 and 0xc299...279d), which was less than Ethereum's finality period. However, the fastest withdrawal took 11 seconds ($11 < 45$, where 45 seconds was Ronin's finality period at the time). These practices pose a considerable risk to *cctx* validation, creating multiple potential attack vectors, particularly for smaller blockchains or those more susceptible to forks.

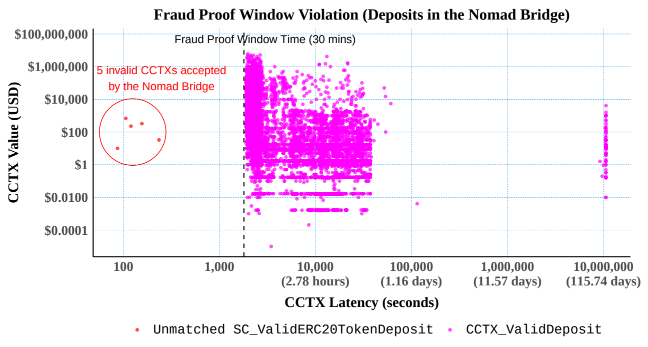

Figure 6: Cross-chain finality violation in the Nomad bridge – we identified 5 unmatched events on both `SC_ValidERC20TokenDeposit` and `TC_ValidERC20TokenDeposit` not captured by `CCTX_ValidDeposit`, due to non-compliance with the fraud-proof time window.

> **Finding 4.** We identified 37 instances where the protocol-defined finality was not satisfied. In Nomad, this was due to smart contract enforcement issues of the fraud-proof window; in Ronin, off-chain validators failed to enforce the source chain's finality period.

5.2.2 *Invalid Beneficiary Addresses.*

In Nomad, users must specify a beneficiary address when transferring funds. To accommodate multiple destination blockchains, Nomad uses a 32-byte field for the beneficiary address instead of a 20-byte address, since some blockchains (e.g., Solana) require 32 bytes. When transferring funds to an EVM-based blockchain, users must left-pad the address with zeros, and the bridge contract extracts the last 20 bytes.

We identified an anomaly when a user submitted a transaction (0x7941...1393) in $\mathcal{S}$ that deposited 10 DAI into a beneficiary address that was right padded instead of left-padded. The contract extracted the last 20 bytes (mainly *0s*) and expected a left-padded address; our tool, which accepted both left and right padding, parsed the address "correctly", i.e., without the padding. The user provided an incorrect input. However, we could not determine whether the error resulted from user misuse or a malfunction in the bridge's UI.

We also detected three anomalies when withdrawing funds in $\mathcal{S}$ (e.g., 0xfcc6...7c5f). These involved events that we could not decode earlier (in Section 5.1.3) because the destination Ethereum address is represented as an unpadded 32-byte string, and therefore represents an invalid Ethereum address. Again, the bridge contract simply extracted the last 20 bytes, whereas our tool throws an error. Interestingly, none of the destination addresses extracted by the bridge contract showed any activity after these transactions, which reveals that the addresses computed by the bridge were not the ones intended by the users – i.e., users mistakenly provided a wrongly formatted address and lost the funds because they do not control these addresses. While these 4 cases can be considered **false positives** from our tool – i.e., not protocol anomalies – they still revealed genuine anomalies in user behavior.

> **Finding 5.** Protocols do not safeguard users against incorrectly formatted inputs, as bridge contracts are often designed to be blockchain-agnostic and may lack strict input validation.

Table 4: Identification of the origin of all anomalies identified by `CCTX_ValidDeposit` and `CCTX_ValidWithdrawal` within $[t_0; t_3]$

	Nomad Bridge			Ronin Bridge		
Logical Rule (cf. Section 3.3)	Captured	Unmatched	Anomaly Explanation	Captured	Unmatched	Anomaly Explanation
1. `SC_ValidNativeTokenDeposit`	7,187	0		38,462	10	10 do not comply with `cctx_finality`
2. `SC_ValidERC20TokenDeposit`	4,223	6	5 do not comply with `cctx_finality` 1 contains an invalid beneficiary address (FP)	5,527	0	
3. `TC_ValidERC20TokenDeposit`	11,417	13	5 do not comply with `cctx_finality` 7 do not comply with `token_mapping` 1 contains an invalid beneficiary address (FP)	43,990	10	10 do not comply with `cctx_finality`
5. `TC_ValidNativeTokenWithdrawal`	464	238	238 events do not have correspondence on $\mathcal{S}$	0	0	
6. `TC_ValidERC20TokenWithdrawal`	4,846	491	491 events do not have correspondence on $\mathcal{S}$	35,411	11,814	22 do not comply with `cctx_finality` 11,792 events do not have correspondence on $\mathcal{S}$
7. `SC_ValidERC20TokenWithdrawal`	4,869	387	3 contains an invalid beneficiary address (FP) 2 do not comply with `token_mapping` 382 events do not have correspondence on $\mathcal{T}$	25,470	732	708 matched events on $\mathcal{T}$ before t_0 (FP)[1] 22 do not comply with `cctx_finality` 2 events do not have correspondence on $\mathcal{T}$

Example: there were 11,417 records captured by `TC_ValidERC20TokenDeposit` (Rule 3), however, only 11,404 were matched by a transaction on $\mathcal{S}$ (counted in `CCTX_ValidDeposit` – cf. Table 3), which indicates there are 13 events emitted by the bridge contract on $\mathcal{T}$ without a corresponding action on $\mathcal{S}$, which is an anomaly.
Note: we mark in red the events that caused loss of funds to the protocol (i.e., attacks identified by XChainWatcher)
[1] false positives (FP) due to the impossibility of extracting data in the Ronin blockchain ($\mathcal{T}$) before t_0, which caused the events to not being matched

5.2.3 *Invalid Token Mappings.*

We identified 9 anomalies in Rules 4 and 8 due to records not complying with the `token_mapping` predicate, 7 when depositing tokens in $\mathcal{T}$ using the Nomad bridge, and 2 when withdrawing tokens in $\mathcal{S}$.

According to the Nomad bridge documentation [46], anyone can deploy a new token on Moonbeam and ask the bridge to link it to the contract that represents the same token on Ethereum. We found 5 transactions that involved the Nomad bridge operator deploying new ERC20 tokens on Moonbeam. One of the transactions 0x7fe7...bf27 deployed a new token contract on Moonbeam mapped to a token contract in $\mathcal{S}$ called WRAPPED GLMR (e.g., 0x92C3...7178). This token is the native token of the Moonbeam blockchain, which already exists in $\mathcal{S}$ and is already mapped by the bridge (0xba8d...A663) – and therefore, this mapping should not have been validated by the bridge operator. Our hypothesis is that these may be users creating fake tokens with the name of real tokens (e.g., WRAPPED GLMR), in an attempt to later on withdraw real funds on Ethereum. This vulnerability was the cause of an attack on the Thorchain bridge in 2022 [71], where attackers created a fake contract called *Wrapped Ether* and tricked the bridge contract into accepting the withdrawal of real Ether. The 4 subsequent transactions tried depositing different amounts of different tokens to multiple addresses in $\mathcal{T}$. Strangely, no activity was found on $\mathcal{S}$ in the mapped contracts – i.e., the tokens were never used by anyone previously (e.g., 0xda3f...5c72).This activity is very unusual, especially since the transactions mapping tokens across blockchains were issued by the Nomad bridge operator, which suggests a lack of contract verification between blockchains by the operator.

Finding 6. The Nomad bridge operator linked fake or duplicate tokens between Moonbeam and Ethereum, including an already existing mapping for WRAPPED GLMR. This highlights a lack of rigorous token contract verification, leaving the protocol vulnerable to spoofing attacks.

5.2.4 *Withdrawals in $\mathcal{T}$ with no Correspondence in $\mathcal{S}$.*

We found 729 (= 238 + 491) withdrawals on $\mathcal{T}$, in which no corresponding transaction was found in $\mathcal{S}$. A first hypothesis to explain the high number of anomalies is whether these values are a consequence of the attack, i.e., multiple users tried (unsuccessfully) to withdraw funds as the bridge was paused. To test this hypothesis, Figure 7 shows the comparison between the matched and unmatched withdrawal events emitted on $\mathcal{T}$ on the Nomad bridge (the results for the Ronin bridge are similar but not shown for the sake of space). As expected, close to when the hack happened in August 2022, there were many unmatched withdrawal events emitted in $\mathcal{T}$ – there were 313 events trying to withdraw \$24.7M worth of tokens in the 24 hours prior to the attack. In Ronin, we identified 468 events withdrawing \$24.3M in the same period. Not surprisingly, in the event of an attack, it is difficult to withdraw tokens, due to the bridge being paused after the attack. However, it is also noticeable that throughout the entire period in which the bridge was functioning, there were always multiple low-value unmatched events, following the same trend as the matched ones. These are funds escrowed in $\mathcal{T}$ in which the corresponding tokens were never unlocked on $\mathcal{S}$ within $[t_1; t_2]$ – and the bridge still holds the escrowed assets in $\mathcal{T}$.

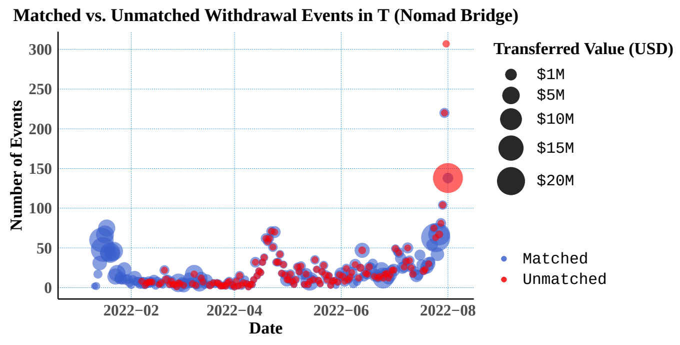

Figure 7: Withdrawal events emitted on $\mathcal{T}$ matched (N = 4,482) or unmatched (N = 828) with another event on $\mathcal{S}$ (through `CCTX_ValidWithdrawal`) in the Nomad bridge.

A manual analysis of these anomalies revealed that many of the destination addresses (beneficiaries on Ethereum) targeted by these events on $\mathcal{T}$ had no funds or had not made any transactions to date. Table 5 illustrates these findings, separating metrics extracted before and after (i.e., as a consequence of) the attack. Our analysis

Table 5: Analysis of the balance of destination addresses on Ethereum targeted by withdrawals on $\mathcal{T}$

	Nomad Bridge			Ronin Bridge		
	Before Attack	After Attack	**Total**	Before Attack	After Attack	**Total**
Unmatched withdrawal events in $\mathcal{T}$	541	188	**729**	11,574	220	**11,794**
Addresses with balance 0 at withdrawal date	95	26	**121**	5,988	66	**6,054**
Addresses with balance 0 at withdrawal date and still today	55	17	**72**	5,212	49	**5,261**
Addresses with balance $<$ 0.0011 at withdrawal date	185	46	**231**	7,381	88	**7,469**
Total Value (in million of USD)	\$0.34M	\$3.27M [1]	**\$3.62M** [1]	\$1.09M	\$0.09M	**\$1.18M**
Addresses that tried withdrawing more than once	34	23	**58**	932	21	**956**
Addresses that tried withdrawing exactly once	460	136	**592**	9,490	176	**9,657**

[1] A single address is responsible for \$3M.

revealed that, spanning both bridges, 6,175 addresses on Ethereum (≈ 49%) had a zero balance at the time of the withdrawal event, in which 5,333 (≈ 43%) are still holding a zero balance at the time of writing. As a result, users cannot withdraw their assets due to not having funds to cover gas fees. According to the Ronin documentation, users should have a minimum of 0.0011 ETH to cover gas fees for issuing a transaction on Ethereum to withdraw funds [64]. 7,700 addresses (≈ 61) did not have sufficient funds to meet this requirement. The total value of unwithdrawn funds amounts to \$4.8M, in which a single transaction attempted to withdraw \$3M. Excluding this outlier, the amount not withdrawn is \$1.8M. Figure 8 shows the distribution of balances of *beneficiary* addresses with non-zero balances when the withdrawal event was triggered in $\mathcal{T}$.

To assess the impact of the attack on these values, we divided the analysis into *pre-attack* and *post-attack* periods. The data shows that the attack does not seem to have any influence. The number of data points before the attack is much higher (≈97%), suggesting that this is a common practice when the bridge is operating normally. Interestingly, even users with many funds, including those with over 10 or 200 ETH in their addresses, were involved in this behavior (cf. Figure 8). Another noteworthy finding is the difference between unique addresses that attempted to withdraw funds once versus those that tried multiple times. Some users repeatedly attempted withdrawals, while others seemingly gave up, likely considering their funds lost. This may also be attributed to user inexperience and inadequate UI/UX [15]. The Pearson correlation coefficient between the number of withdrawal attempts and the amount withdrawn (by each user withdrawal request) is negligible (−0.017) showing no meaningful relationship between both variables.

Finding 7. We found 729 cases where users tried to withdraw funds from the destination blockchain ($\mathcal{T}$), but the bridge never completed the corresponding transaction on the source blockchain ($\mathcal{S}$). This left up to \$4.8M stuck in the bridge. While many of these happened around the time of the attack, the majority occurred during normal use. Moreover, nearly half of the users didn't have enough ETH to pay gas fees on the destination blockchain, preventing them from claiming their funds, and pointing to serious usability issues.

5.2.5 Withdrawals in $\mathcal{S}$ with no Correspondence in $\mathcal{T}$. Both bridges analyzed in this paper suffered a **Forged Withdrawal Attack**, where funds were stolen from $\mathcal{S}$ (Ethereum). As shown in Table 4, 382 unmatched events, under `SC_ValidERC20TokenWithdrawal`, were identified because they were not matched on $\mathcal{T}$ on the Nomad bridge. Analyzing the timestamp of the transactions in which these events were emitted, we conclude that all 382 events were part of the attack, involving 382 **transactions** and 279 **unique addresses**.

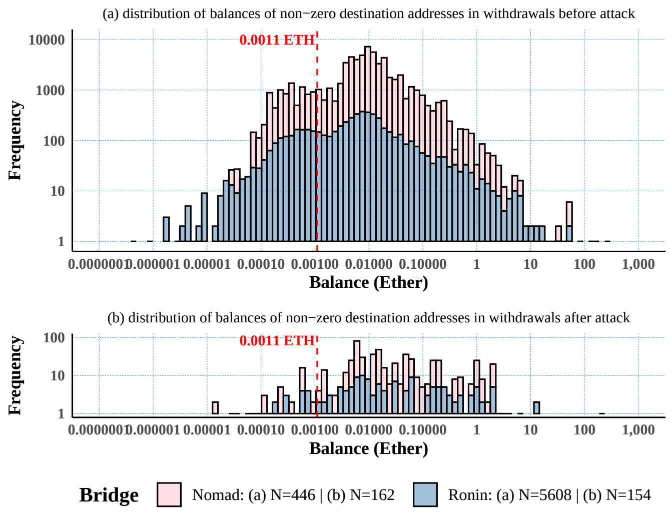

Figure 8: Distribution of the balance of all addresses to which funds are being sent in $\mathcal{S}$ when withdrawing funds from $\mathcal{T}$. A red dotted line marks the value needed to pay for transaction fees to successfully withdraw funds in $\mathcal{S}$.

These totaled \$159, 577, 598 of stolen funds. These events had only 14 unique withdrawal IDs, indicating that attackers copy-pasted data from other transactions, exploiting the bridge's acceptance of any data as valid proof [56]. Our analysis identified 279 addresses that exploited the protocol, the majority of which were contracts deployed in bulk to scatter funds across multiple addresses. We traced the transactions and identified **45 unique EOAs** responsible for deploying these contracts. We cross-referenced our findings with data from *Peckshield*, a reputable security firm, which provided a list of addresses involved in exploiting the bridge at the time of the attack [50]. We identified 9 more EOAs than Peckshield in the same blockchains (36 EOAs). We also found a dataset related to the attack on GitHub [45], which includes 246 transactions, less 136 than ours. To eliminate the possibility of false positives, we manually checked all anomalies not identified by the other datasets.

In the Ronin bridge data, we identified 710 anomalies related to events emitted on $\mathcal{S}$ without correspondence on $\mathcal{T}$. Unfortunately, due to rate limits for extracting data from the Ronin blockchain, we could not decrease t_0 to the date on which the contracts were deployed. This caused our tool to identify anomalies in transactions that would match events emitted well before our period of analysis. We captured over 500k additional transactions in $[t_0; t_1[$, more than 3.5 months taking into account the maximum latency of withdrawals in the Ronin bridge (cf. Figure 5), of around 3 months, and added some margin, but it did not prove to be enough. To exclude withdrawals before t_0, we based ourselves on the `withdrawal_id` – a counter incremented for each withdrawal event emitted in the bridge contract. Of the unmatched 710 events, 708 had a withdrawal ID less than `withdrawal_id` of the first event included in $[t_0; t_1[$, suggesting that they were emitted before our collection data interval. We are left with 2 unmatched withdrawal events in the selected interval. These events were emitted by transactions issued by the same address (0x098b...2f96) on March 23, 2022, 13:29 (0xc28f...d0b7) and 13:31 (0xed2c...9b08), transferring a total of $565.64M. These two transactions are those identified in the industry as pertaining to the Ronin bridge hack. When comparing the results, no false negatives were found.

Finding 8. XChainWatcher successfully identified malicious transactions in the Nomad and Ronin bridges. The Nomad analysis overcame previous analyses by uncovering 9 additional attacker EOAs not reported by security firms and 136 more transactions than the largest existing public dataset of the Nomad hack.

6 Discussion and Future Work

We present the discussion and limitations of our work.

Rule modeling. Modeling cross-chain rules requires exploring the semantics of a bridge protocol, its data model, and associated token contracts. While bridges can be categorized into several classes of models – in this paper, we analyzed bridges that connect Ethereum to sidechains. Specific rule modeling may change slightly depending on the particular instantiation. This paper does not aim to propose a universal cross-chain model applicable to all protocols. Instead, we empirically demonstrate that logic-driven analysis is effective for the detection of unknown anomalies on bridges. Our work establishes the first baseline for future security analyses on cross-chain bridges. Rules are created based on the current behavior of the protocol. If event signatures are changed, XChainWatcher needs to be updated accordingly by the bridge operator to capture the new events. This seems acceptable as XChainWatcher is supposed to run alongside the bridge and controlled by the operator.

Timeframes and Selected Bridges. We focus on short timeframes with confirmed attacks, modeling expected behavior rather than relying on signature-based detection. Since there are no prior anomaly datasets for cross-chain bridges, manually analyzing large volumes of anomalies would be impractical. Thus, we target (1) short periods and (2) timeframes with verified attacks. We selected the Ronin bridge as it is the most profitable cross-chain attack to date, and the Nomad bridge because of the high number of transactions exploiting the protocol in the hack.

Extensibility of XChainWatcher. The framework is extensible and easy to use. To add support for other protocols, users must **(1)** analyze the protocol and incorporate any specific restrictions into the cross-chain rules (i.e., adding any missing protocol-specific cross-chain rules), **(2)** extract transaction receipts to be used in the analysis, **(3)** create an *Event Data Decoder and Extractor* that decodes event data and creates logical relations, and **(4)** populate a configuration file (cf. Figure 3) with RCP connection URLs, bridge contract addresses, and tokens mappings. An important feature of the proposed design of XChainWatcher is that it is agnostic to the state validation logic employed – *Trusted Third Parties* (Ronin), and *Native State Verification* (Nomad) [8] – because it only relies on the events emitted by on-chain contracts.

Event-based Analysis. We chose to perform an event-based analysis for two main reasons. Firstly, protocols typically involve more transactions than those triggering transferring assets, e.g., light client updates. Analyzing all transactions, including those unrelated to actual state changes, would be inefficient and resource-intensive. Moreover, capturing all transactions that target bridge contracts is not enough to extract all the relevant data, as users can issue transactions to intermediary protocols (such as bridge aggregators [66]) that make internal transaction calls to bridge contracts. When contract events are not emitted (e.g., due to a bug in a contract or even a malicious upgrade in a **multi-transaction attack**), our tool detects this behavior as abnormal because a state change will be missing in the cross-chain flow (cf. Section 2.2).

Future Work. Future work is threefold: **(1)** extend analysis periods to identify further anomalies, such as *salami slicing attacks* [17], **(2)** support additional bridges, **(3)** using the clean and labeled dataset to train anomaly detection models to perform large scale analyses of cross-chain data.

7 Related Work

Despite the extensive research corpus on interoperability [15, 16, 70], there is little related work available on monitoring and protecting interoperability solutions. The concept of a cross-chain model was introduced in Hephaestus [14], a theoretical cross-chain model generator, highlighting the importance of defining cross-chain rules to identify misbehavior. XScope [71] uses three static rules to detect three types of attacks (signature-based detection) on cross-chain bridges, specifically targeting smaller chains with limited datasets. XScope's detection capabilities are limited to three specific anomalies, and it focuses exclusively on token deposits (not covering the withdrawal process). Unfortunately, XScope is not open-source, limiting a deeper empirical comparison. In the industry, Hyperlane [36], Range [53], and Layer Zero's Precrime [39] provide analysis tools for bridges. However, these are proprietary and lack technical documentation, systematic evaluation, and datasets, making it challenging to compare directly with our work. Finally, while post-attack analyses typically trace the flow of funds using tools such as Chainalysis [20], our tool enables the retrieval of the same (and more) data by applying cross-chain rules.

8 Conclusion

This paper proposes a monitoring framework for cross-chain bridges powered by a cross-chain model supported by a Datalog engine. We *uncover significant attacks* within cross-chain bridges, such as 1) transactions accepted in one chain *before the finality time of the original one elapsed*, breaking the safety of the bridge protocol; 2) users trying to exploit a protocol through *the creation of fake versions of wrapped Ether* to withdraw real ether on the Ethereum blockchain, breaking safety; 3) bridge contract implementations handling *unexpected inputs differently across chains*, hindering a good UX and leading to the loss of user funds. In addition, we identify every transaction involved in previous hacks on the bridges studied. We show that although only 49 unique externally owned accounts (EOAs) exploited Nomad, there were 380 exploit events, with each address deploying multiple exploit contracts to obscure the flow of funds. Finally, our study highlights a critical user awareness gap – many users struggle to withdraw funds due to the highly manual nature of the process, contrasting with the more streamlined deposit process managed by bridge operators. This user error has led to over $4.8M in unwithdrawn funds due to users mistakenly sending funds to addresses they do not control or that have never been active. We are the first to empirically analyze the security vulnerabilities of cross-chain bridges. Our open-source dataset provides a valuable resource for future research.

References

[1] 2022. Multichain Contract Vulnerability Post Mortem | by Multichain (Previously Anyswap) | Medium. https://medium.com/multichainorg/multichain-contract-vulnerability-post-mortem-d37bfab237c8

[2] 2022. Rekt - Qubit Finance. https://rekt.news/qubit-rekt/

[3] 2022. Rekt - Wormhole. https://rekt.news/wormhole-rekt/

[4] 2024. JSON-RPC API | ethereum.org. https://ethereum.org/en/developers/docs/apis/json-rpc/

[5] Raja Amir. 2023. Etherscan Information Center | Understanding and Ethereum Transaction. https://info.etherscan.com/understanding-an-ethereum-transaction/

[6] Andreas M Antonopoulos and Gavin Wood. 2018. *Mastering Ethereum: building smart contracts and dapps.* O'Reilly Media.

[7] Multichain (Previously Anyswap). 2021. Anyswap Multichain Router V3 Exploit Statement. https://medium.com/multichainorg/anyswap-multichain-router-v3-exploit-statement-6833f1b7e6fb

[8] A. Augusto, R. Belchior, M. Correia, A. Vasconcelos, L. Zhang, and T. Hardjono. 2024. SoK: Security and Privacy of Blockchain Interoperability. In *2024 IEEE Symposium on Security and Privacy (SP).* IEEE Computer Society, Los Alamitos, CA, USA, 234–234. https://doi.org/10.1109/SP54263.2024.00182

[9] Rebecca Gurley Bace, Peter Mell, et al. 2001. Intrusion detection systems. (2001).

[10] Base. 2025. Base. https://base.org

[11] Rob Behnke. 2021. Explained: The pNetwork Hack. https://www.halborn.com/blog/post/explained-the-pnetwork-hack-september-2021

[12] Rob Behnke. 2021. Explained: The THORChain Hack (July 2021). https://www.halborn.com/blog/post/explained-the-thorchain-hack-july-2021

[13] Rafael Belchior, Luke Riley, Thomas Hardjono, André Vasconcelos, and Miguel Correia. 2023. Do You Need a Distributed Ledger Technology Interoperability Solution? *Distrib. Ledger Technol.* 2, 1, Article 1 (March 2023), 37 pages. https://doi.org/10.1145/3564532

[14] Rafael Belchior, Peter Somogyvari, Jonas Pfannschmidt, André Vasconcelos, and Miguel Correia. 2024. Hephaestus: Modeling, Analysis, and Performance Evaluation of Cross-Chain Transactions. *IEEE Transactions on Reliability* 73, 2 (2024), 1132–1146. https://doi.org/10.1109/TR.2023.3336246

[15] Rafael Belchior, Jan Süßenguth, Qi Feng, Thomas Hardjono, André Vasconcelos, and Miguel Correia. 2024. A brief history of blockchain interoperability. *Commun. ACM* 67, 10 (2024), 62–69.

[16] Rafael Belchior, André Vasconcelos, Sérgio Guerreiro, and Miguel Correia. 2021. A Survey on Blockchain Interoperability: Past, Present, and Future Trends. *ACM Comput. Surv.* 54, 8, Article 168 (oct 2021), 41 pages. https://doi.org/10.1145/3471140

[17] Rekha Bhowmik. 2008. Data Mining Techniques in Fraud Detection. *Journal of Digital Forensics, Security and Law* 3, 2 (2008), 35–53. https://doi.org/10.15394/jdfsl.2008.1040

[18] Tom Blackstone. 2023. LayerZero raises $120M to expand cross-chain messaging efforts. https://cointelegraph.com/news/layerzero-raises-120m-to-expand-cross-chain-messaging-efforts

[19] Blockdaemon. 2024. Blockdaemon REST API. https://docs.blockdaemon.com/reference/introduction-txapi

[20] Chainalysis. 2025. Chainalysis. https://www.chainalysis.com/

[21] ChainSwap. 2021. ChainSwap Exploit 11 July 2021 Post-Mortem. https://chain-swap.medium.com/chainswap-exploit-11-july-2021-post-mortem-6e4e346e5a32

[22] Vishal Chawla. 2023. Union Labs raises $4 million to develop cross-chain bridge enabled by ZK proofs. https://www.theblock.co/post/263310/union-labs-raises-4-million-to-develop-cross-chain-bridge-enabled-by-zk-proofs

[23] James Cirrone. 2023. $225 Million Raised in Wormhole Token Sales. https://www.coindesk.com/business/2023/11/29/blockchain-messaging-platform-wormhole-raises-225m-at-25b-valuation

[24] James Cirrone. 2023. Crypto funding: A $72M week for cross-chain oracles, NFT merchandise. https://blockworks.co/news/funding-cross-chain-oracle-nft-merchandise

[25] Li Duan, Yangyang Sun, Wei Ni, Weiping Ding, Jiqiang Liu, and Wei Wang. 2023. Attacks Against Cross-Chain Systems and Defense Approaches: A Contemporary Survey. *IEEE/CAA Journal of Automatica Sinica* 10, 8 (2023), 1643–1663.

[26] Ethereum Foundation. 2024. ERC-20 Token Standard. https://ethereum.org/en/developers/docs/standards/tokens/erc-20/

[27] Ethereum Foundation. 2024. ERC-721 Token Standard. https://ethereum.org/en/developers/docs/standards/tokens/erc-721/

[28] Ethereum Foundation. 2025. The Merge | Ethereum.org. https://ethereum.org/en/roadmap/merge/

[29] Ethereum Foundation. 2025. Sidechains. https://ethereum.org/en/developers/docs/scaling/sidechains/

[30] Arbitrum Foundation. 2025. Arbitrum — The Future of Ethereum. https://arbitrum.io/

[31] Eliza Gkritsi. 2021. $139M BXH Exchange Hack Was the Result of Leaked Admin Key. https://www.coindesk.com/tech/2021/11/01/139m-bxh-exchange-hack-was-the-result-of-leaked-admin-key/

[32] Gnosis. 2025. Deposit Contracts | Gnosis Chain. https://docs.gnosischain.com/about/specs/deposit-contracts

[33] Gnosis. 2025. xDAI Bridge | Gnosis Chain. https://docs.gnosischain.com/bridges/About%20Token%20Bridges/xdai-bridge

[34] The go-ethereum Authors. 2025. Built-in tracers | go-ethereum. https://geth.ethereum.org/docs/developers/evm-tracing/built-in-tracers#call-tracer

[35] Terje Haugum, Bjørnar Hoff, Mohammed Alsadi, and Jingyue Li. 2022. Security and Privacy Challenges in Blockchain Interoperability - A Multivocal Literature Review. In *Proceedings of the International Conference on Evaluation and Assessment in Software Engineering 2022 (EASE '22).* Association for Computing Machinery, New York, NY, USA, 347–356. https://doi.org/10.1145/3530019.3531345

[36] Hyperlane. 2025. Hyperlane. https://hyperlane.xyz/

[37] Herbert Jordan, Bernhard Scholz, and Pavle Subotić. 2016. Soufflé: On synthesis of program analyzers. In *Computer Aided Verification: 28th International Conference, CAV 2016, Toronto, ON, Canada, July 17-23, 2016, Proceedings, Part II 28.* Springer, 422–430.

[38] Polygon Labs. 2025. Mapped tokens – Polygon Knowledge Layer. https://docs.polygon.technology/pos/reference/mapped-tokens/

[39] LayerZero Labs. 2022. LayerZero Security Update – April 2022. https://medium.com/layerzero-official/layerzero-security-update-april-2022-4c27a22380b4

[40] Sung-Shine Lee, Alexandr Murashkin, Martin Derka, and Jan Gorzny. 2023. SoK: Not Quite Water Under the Bridge: Review of Cross-Chain Bridge Hacks. In *2023 IEEE International Conference on Blockchain and Cryptocurrency (ICBC).* 1–14. https://doi.org/10.1109/ICBC56567.2023.10174993

[41] Li.Fi. 2024. LI.FI – Bridge & DEX Aggregation Protocol. https://li.fi/

[42] Mantle. [n. d.]. Mantle | Mass Adoption of Decentralized and Token-Governed Technologies. https://www.mantle.xyz

[43] Poly Network. 2023. The Poly Network Exploit Analysis. https://polynetwork.medium.com/the-poly-network-exploit-analysis-b0a77aff6078

[44] Margaux Nijkerk. 2023. Coinbase, Framework Venture Funds Invest $5M in Socket Protocol, in Bet on Blockchain Interoperability. https://www.coindesk.com/tech/2023/09/06/coinbase-framework-venture-funds-invest-5m-in-socket-protocol-in-bet-on-blockchain-interoperability/

[45] Nomad. 2022. https://github.com/nomad-xyz/hack-data/blob/main/data/hack/transactions.json

[46] Nomad. 2022. FAQ | Nomad Docs. https://docs.nomad.xyz/token-bridge/faq

[47] Nomad. 2022. Glossary | Nomad Docs. https://docs.nomad.xyz/resources/glossary

[48] Nomad. 2022. Introduction | Nomad Docs. https://docs.nomad.xyz/nomad-101/introduction

[49] Optimism. 2025. Optimism. https://optimism.io/

[50] PeckShieldAlert. 2022. #PeckShieldAlert PeckShield has detected 41 addresses grabbed $152M (80%) in the nomadxyz_bridge exploit, including 7 MEV Bots ($7.1M), RariCapital Arbitrum exploiter ($3.4M), and 6 White Hat ($8.2M). https://x.com/PeckShieldAlert/status/1554350737957998592

[51] Eli Phoenix. 2023. Supra Completes Over $24m in Early Stage Funding to Date. https://cointelegraph.com/press-releases/supra-completes-over-24m-in-early-stage-funding-to-date

[52] Polygon. 2024. Polygon Knowledge Center. https://stargateprotocol.gitbook.io/stargate/v2-developer-docs/integrate-with-stargate/how-to-swap

[53] Range. 2024. Range. https://www.range.org/

[54] Rekt. 2021. POLY NETWORK - REKT. https://rekt.news/polynetwork-rekt/

[55] Rekt. 2021. THORChain - REKT 2. https://rekt.news/thorchain-rekt2/

[56] Rekt. 2022. Nomad Bridge - REKT. https://rekt.news/nomad-rekt/

[57] Rekt. 2022. Rekt - BNB Bridge. https://www.rekt.news/bnb-bridge-rekt/

[58] Rekt. 2022. Rekt - Meter. https://rekt.news/meter-rekt/

[59] Rekt. 2022. Ronin Network - REKT. https://rekt.news/ronin-rekt/

[60] Rekt. 2023. Multichain - REKT 2. https://rekt.news/multichain-rekt2/

[61] Rekt. 2023. POLY NETWORK - REKT 2. https://rekt.news/poly-network-rekt2/

[62] Ronin. 2024. Earlier today, we were notified by white-hats about a potential exploit on the Ronin bridge. After verifying the reports, the bridge was paused approximately 40 minutes after the first on-chain action was spotted. https://x.com/ronin_network/status/1820804772917588339

[63] Ronin. 2024. Ronin Bridge | Ronin Docs. https://docs.roninchain.com/apps/ronin-bridge

[64] Ronin. 2025. Withdraw an ERC20 token | Ronin Docs. https://docs.roninchain.com/apps/ronin-bridge/withdraw-token#step-3-confirm-your-withdrawal

[65] Squid. 2023. Squid raises $3.5 million to build next-generation cross-chain swaps powered by Axelar. https://medium.com/@squidrouter/squid-raises-3-5-million-to-build-next-generation-cross-chain-swaps-powered-by-axelar-c3284bf33b02

[66] S. Subramanian, A. Augusto, R. Belchior, A. Vasconcelos, and M. Correia. 2024. Benchmarking Blockchain Bridge Aggregators. In *2024 IEEE International Conference on Blockchain (Blockchain)*. IEEE Computer Society, Los Alamitos, CA, USA, 37–45. https://doi.org/10.1109/Blockchain62396.2024.00015

[67] L2BEAT team. 2025. L2BEAT – The state of the layer two ecosystem. https://l2beat.com/bridges/summary

[68] THORChain. 2021. ETH Parsing Error and Exploit. https://medium.com/thorchain/eth-parsing-error-and-exploit-3b343aa6466f

[69] Ruoyu Yin, Zheng Yan, Xueqin Liang, Haomeng Xie, and Zhiguo Wan. 2023. A survey on privacy preservation techniques for blockchain interoperability. *Journal of Systems Architecture* (Apr 2023), 102892. https://doi.org/10.1016/j.sysarc.2023.102892

[70] Alexei Zamyatin, Dominik Harz, Joshua Lind, Panayiotis Panayiotou, Arthur Gervais, and William Knottenbelt. 2019. XCLAIM: Trustless, Interoperable, Cryptocurrency-Backed Assets. In *2019 IEEE Symposium on Security and Privacy (SP)*. 193–210. https://doi.org/10.1109/SP.2019.00085

[71] Jiashuo Zhang, Jianbo Gao, Yue Li, Ziming Chen, Zhi Guan, and Zhong Chen. 2023. Xscope: Hunting for Cross-Chain Bridge Attacks. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering (ASE '22)*. Association for Computing Machinery, New York, NY, USA, Article 171, 4 pages. https://doi.org/10.1145/3551349.3559520

[72] Mengya Zhang, Xiaokuan Zhang, Yinqian Zhang, and Zhiqiang Lin. 2024. Security of Cross-chain Bridges: Attack Surfaces, Defenses, and Open Problems. In *Proceedings of the 27th International Symposium on Research in Attacks, Intrusions and Defenses* (Padua, Italy) *(RAID '24)*. Association for Computing Machinery, New York, NY, USA, 298–316. https://doi.org/10.1145/3678890.3678894

[73] Qianrui Zhao, Yinan Wang, Bo Yang, Ke Shang, Ming Sun, Haijun Wang, Zijiang Yang, and Xin He. 2023. A Comprehensive Overview of Security Vulnerability Penetration Methods in Blockchain Cross-Chain Bridges. *Authorea (Authorea)* (Oct 2023). https://doi.org/10.22541/au.169760541.13864334/v1