



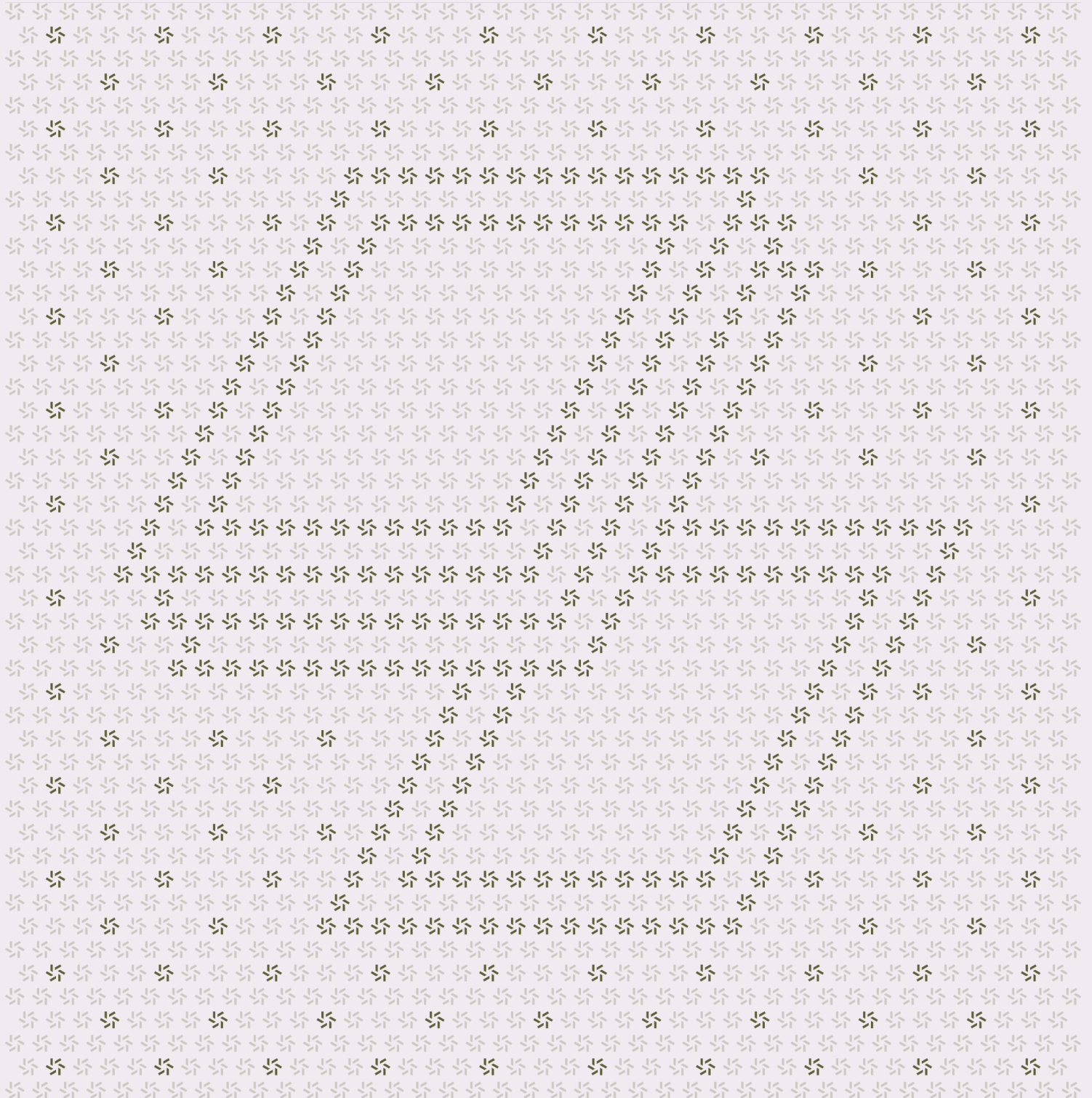
June 18, 2024

Prepared for
Stan L
Zon M
Initia Labs

Prepared by
GunHee Ahn
William Bowling
Filippo Cremonese
Aaron Esau
Junyi Wang
Zellic

Initia

Comprehensive Security Assessment



Contents

About Zellic	7
<hr/>	
1. Overview	7
1.1. Executive Summary	8
1.2. Goals of the Assessment	8
1.3. Non-goals and Limitations	8
1.4. Results	9
<hr/>	
2. Introduction	9
2.1. About Initia	10
2.2. Methodology	10
2.3. Scope	12
2.4. Project Overview	15
2.5. Project Timeline	16
<hr/>	
3. Stdlib/Natives Findings	16
3.1. Unsound native-function declaration leading to critical verifier bypass	17
3.2. Infinite recursion possible with module dependencies	19
3.3. Unchecked UTF-8 decoding enables memory corruption	21
3.4. Missing gas charge on memo in native_nft_transfer	24
3.5. Next zapping ID potentially duplicated	26
3.6. Published module names do not necessarily match binary module	27
3.7. Ability to bypass swap fees	30
3.8. Module can be duplicated in module publish requests	33

3.9.	Hex decode accepting invalid characters	35
3.10.	Stablepools can be created with one or no assets	36
3.11.	Bad decimal-parsing function accepts multiple dots	37
3.12.	Potential out-of-gas reversion when checking object permissions	39
3.13.	User-triggerable invariant violation	41
3.14.	Stablepool swap can be called, repeating the same asset	43
3.15.	Incorrect string-formatting helper	44
3.16.	Incorrect minimum-TVL module-parameter check	46
3.17.	The upgrade policy can be set to a large value	48
3.18.	Call to next on iterator without prepare aborts	49
3.19.	Unnecessarily verbose branching in operator.move	51
4.	Initia Findings	51
4.1.	Frozen module coin store can cause chain halt	52
4.2.	Malicious proposer can skip fee check	54
4.3.	A malicious user can become a permissioned relayer for IBC	56
4.4.	Move coins can be burned twice	59
4.5.	Move coin transfer can bypass blocked accounts	61
4.6.	Error not checked when fetching starting info	63
4.7.	Move coins are case-insensitive in cosmos	65
4.8.	Coin amount not validated when funding and spending community pool	67
5.	MiniEVM Findings	68
5.1.	Query gas limit not enforced through bank module	69

5.2.	Incorrect signer check for shorthand accounts	71
------	---	----

6.	OPinit Findings	72
6.1.	Validator set updates can skip current validators	73
6.2.	Challenger can increase the next output index	76
6.3.	Missing token pair will crash the bridge executor	78
6.4.	Withdrawal hash clash using variable-length fields	80
6.5.	Merkle tree lacks second preimage resistance	82

7.	General Discussion	84
7.1.	Improve AMM tests	85
7.2.	Bridge executor minting power	85
7.3.	Charge before using values	85
7.4.	Empty <code>allowed_publishers</code> array allows every address to publish	86
7.5.	ERC20Keeper with custom contracts	86
7.6.	Suggestions for additional security checks	86
7.7.	Table handle overflow leads to abort	87

8.	Threat Model	87
8.1.	Module: Initia - x/bank	88
8.2.	Module: Initia - x/distribution	88
8.3.	Module: Initia - x/ibc-hooks	89
8.4.	Module: Initia - x/ibc	90
8.5.	Module: Initia - x/move	91

8.6.	Module: Initia - x/mstaking	93
8.7.	Module: Initia - x/reward	94
8.8.	Module: MiniEVM - x/bank	94
8.9.	Module: MiniEVM - x/evm	95
8.10.	Module: OPinit - x/opchild	96
8.11.	Module: OPinit - x/ophost	97
8.12.	Module block.move	100
8.13.	Module table.move	100
8.14.	Module object.move	106
8.15.	Module hex.move	114
8.16.	Module code.move	114
8.17.	Module dex.move	115
8.18.	Module comparator.move	120
8.19.	Module managed_coin.move	121
8.20.	Modules decimal128.move and decimal256.move	122
8.21.	Module string_utils.move	124
8.22.	Module event.move	124
8.23.	Module type_info.move	125
8.24.	Modules any.move and copyable_any.move	126
8.25.	Module address.move	127
8.26.	Module cosmos.move	127
8.27.	Module multisig.move	133
8.28.	Module transaction_context.move	136
8.29.	Module debug.move	136
8.30.	Module table_key.move	137

8.31.	Module account.move	139
8.32.	Module base64.move	143
8.33.	Module from_bcs.move	144
8.34.	Module json.move	148
8.35.	Module oracle.move	149
8.36.	Module zapping.move	150
8.37.	Module reward.move	152

9.	Assessment Results	153
9.1.	Disclaimer	154

10.	Appendix	154
10.1.	Proof of concept: Reference safety verifier bypass	155

About Zellic

Zellic is a vulnerability research firm with deep expertise in blockchain security. We specialize in EVM, Move (Aptos and Sui), and Solana as well as Cairo, NEAR, and Cosmos. We review L1s and L2s, cross-chain protocols, wallets and applied cryptography, zero-knowledge circuits, web applications, and more.

Prior to Zellic, we founded the [#1 CTF \(competitive hacking\) team](#) worldwide in 2020, 2021, and 2023. Our engineers bring a rich set of skills and backgrounds, including cryptography, web security, mobile security, low-level exploitation, and finance. Our background in traditional information security and competitive hacking has enabled us to consistently discover hidden vulnerabilities and develop novel security research, earning us the reputation as the go-to security firm for teams whose rate of innovation outpaces the existing security landscape.

For more on Zellic's ongoing security research initiatives, check out our website zellic.io and follow [@zellic_io](#) on Twitter. If you are interested in partnering with Zellic, contact us at hello@zellic.io.



1. Overview

1.1. Executive Summary

Zellic conducted a security assessment for Initia Labs from April 15th to June 31st, 2024. During this engagement, Zellic reviewed many components of Initia with the goal of identifying security vulnerabilities, design issues, and general weaknesses in security posture.

1.2. Goals of the Assessment

In a security assessment, goals are framed in terms of questions that we wish to answer. These questions are agreed upon through close communication between Zellic and the client. In this assessment, we sought to answer the following questions:

- Is it possible to get nondeterministic code paths in the Initia Move L2?
 - Can an attacker cause the Move VM to crash in any way or cause an infinite loop from the native function implementations?
 - Are there any opportunities for memory corruption, or is there any way to achieve arbitrary code execution outside of the Move VM?
 - Is gas appropriately charged for all native function arguments?
 - Do the new native functions break Move reference safety or security invariants?
 - Are all encoding and decoding functions properly parsing input?
 - What is the impact of rounding errors that occur in funds' balance-related calculations?
 - Is it possible to drain liquidity or steal user funds from the AMMs?
 - Does the optimistic bridge correctly prevent replay attacks?
 - Is the voting power for validators correctly calculated?
-

1.3. Non-goals and Limitations

We did not assess the following areas that were outside the scope of this engagement:

- Front-end components
- Infrastructure relating to the project
- Key custody

Due to the time-boxed nature of security assessments in general, there are limitations in the coverage an assessment can provide. In particular, considering the size of the codebase and the complexity of the system, we cannot guarantee that we have identified all security issues during the assessment. To maximize the likelihood of identifying all security issues, Zellic assigned multiple security engineers to the project. We believe the best way to reduce the risk is to have more eyes on the codebase.

To that end, we always recommend that the project team continue to perform security assessments

either internally or by engaging with additional third-party security firms — especially after a code freeze, prior to a launch. Consider pairing these security-focused reviews with a bug bounty program.






Finally, we recommend adopting a security-focused development workflow, including (but not limited to) augmenting the repository with comprehensive end-to-end tests that achieve 100% branch coverage using any common, maintainable testing framework, thoroughly documenting all function requirements, and training any new developers to have a security mindset while writing code. Please see Discussion section [7.1](#) for our recommendation regarding lack of comprehensive tests in the AMMs, which, in our opinion, is the component that presents the highest risk.

1.4. Results

During our assessment on the scoped Initia components, we discovered 34 findings. Two critical issues were found. Seven were of high impact, five were of medium impact, 12 were of low impact, and the remaining findings were informational in nature.

Additionally, Zellic recorded its notes and observations from the assessment for Initia Labs's benefit in the Discussion section ([7](#)).

Breakdown of Finding Impacts

Impact Level	Count
 Critical	2
 High	7
 Medium	5
 Low	12
 Informational	8



2. Introduction

2.1. About Initia

Initia Labs contributed the following description of Initia:

Initia is a network for 0-to-1 omnichain rollups to create a highly interwoven system of modular networks through architectural ownership of the L1, L2, and the communication layer.

2.2. Methodology

During a security assessment, Zellic works through standard phases of security auditing, including both automated testing and manual review. These processes can vary significantly per engagement, but the majority of the time is spent on a thorough manual review of the entire scope.

Alongside a variety of tools and analyzers used on an as-needed basis, Zellic focuses primarily on the following classes of security and reliability issues:

Basic coding mistakes. Many critical vulnerabilities in the past have been caused by simple, surface-level mistakes that could have easily been caught ahead of time by code review. Depending on the engagement, we may also employ sophisticated analyzers such as model checkers, theorem provers, fuzzers, and so on as necessary. We also perform a cursory review of the code to familiarize ourselves with the components.

Business logic errors. Business logic is the heart of any smart contract application. We examine the specifications and designs for inconsistencies, flaws, and weaknesses that create opportunities for abuse. For example, these include problems like unrealistic tokenomics or dangerous arbitrage opportunities. To the best of our abilities, time permitting, we also review the contract logic to ensure that the code implements the expected functionality as specified in the platform's design documents.

Integration risks. Several well-known exploits have not been the result of any bug within the contract itself; rather, they are an unintended consequence of the contract's interaction with the broader DeFi ecosystem. Time permitting, we review external interactions and summarize the associated risks: for example, flash loan attacks, oracle price manipulation, MEV/sandwich attacks, and so on.

Code maturity. We look for potential improvements in the codebase in general. We look for violations of industry best practices and guidelines and code quality standards. We also provide suggestions for possible optimizations, such as gas optimization, upgradability weaknesses, centralization risks, and so on.

For each finding, Zellic assigns it an impact rating based on its severity and likelihood. There is no hard-and-fast formula for calculating a finding's impact. Instead, we assign it on a case-by-case basis based on our judgment and experience. Both the severity and likelihood of an issue affect its impact. For instance, a highly severe issue's impact may be attenuated by a low likelihood.

We assign the following impact ratings (ordered by importance): Critical, High, Medium, Low, and Informational.

Zellic organizes its reports such that the most important findings come first in the document, rather than being strictly ordered on impact alone. Thus, we may sometimes emphasize an "Informational" finding higher than a "Low" finding. The key distinction is that although certain findings may have the same impact rating, their *importance* may differ. This varies based on various soft factors, like our clients' threat models, their business needs, and so on. We aim to provide useful and actionable advice to our partners considering their long-term goals, rather than a simple list of security issues at present.

Finally, Zellic provides a list of miscellaneous observations that do not have security impact or are not directly related to the scoped components itself. These observations — found in the Discussion ([7. ↗](#)) section of the document — may include suggestions for improving the codebase, or general recommendations, but do not necessarily convey that we suggest a code change.

2.3. Scope

The engagement involved a review of the following targets:

Initia Components

Types	Rust, Move, Solidity, Golang
Platform	Initia
Target	Move Natives
Repository	https://github.com/initia-labs/movevm
Version	455fe586ea89fcf10afdaf0766da5151d163edc8
Programs	api/iterator.go libmovem/** lib.go crates/storage/** crates/vm/** (diff from aptos-core:aptos-node-v1.2.8 only) crates/natives/** (diff from aptos-core:aptos-node-v1.2.8 only)

Target	Initia Stdlib
Repository	https://github.com/initia-labs/movevm ↗
Version	455fe586ea89fcf10afdaf0766da5151d163edc8
Programs	account.move block.move dex.move staking.move transaction_context.move code.move decimal128.move decimal256.move object.move (diff from aptos-core:1c7d250e only) table.move token/initia_nft.move miniswap.move stableswap.move multisig.move json.move vip/reward.move vip/zapping.move

Target	OPinit
Repository	https://github.com/initia-labs/OPinit ↗
Version	3d4dc7e52709e6bbc279ff9cfa17dc5b00b2230e
Programs	x/opchild/**/*.go x/ophost/**/*.go bots/src/**/*.go

Target	Initia
Repository	https://github.com/initia-labs/initia ↗
Version	444d8f47463d2f776a9be3d33f956f378a121b09
Programs	x/bank/**/*.go x/distribution/**/*.go x/ibc/**/*.go x/move/**/*.go x/ibc-hooks/**/*.go x/mstaking/**/*.go x/reward/**/*.go x/gov/**/*.go (diff from cosmos-sdk:v0.13.0 only)
Target	MiniEVM
Repository	https://github.com/initia-labs/minievms ↗
Version	ba17f6a540713740f7abcbce1ce42bc239de8e4c
Programs	x/** app/** types/**

2.4. Project Overview

Zellic was contracted to perform a security assessment with five consultants for a total of 17.4 person-weeks. The assessment was conducted over the course of seven calendar weeks.

Contact Information

The following project manager was associated with the engagement:

Chad McDonald
✈ Engagement Manager
chad@zellic.io ↗

The following consultants were engaged to conduct the assessment:

GunHee Ahn
✈ Engineer
gunhee@zellic.io ↗

William Bowling
✈ Engineer
vakzz@zellic.io ↗

Filippo Cremonese
✈ Engineer
fcremo@zellic.io ↗

Aaron Esau
✈ Engineer
aaron@zellic.io ↗

Junyi Wang
✈ Engineer
junyi@zellic.io ↗

2.5. Project Timeline

The key dates of the engagement are detailed below.

April 15, 2024	Start of primary review period
-----------------------	--------------------------------

April 16, 2024	Initia scope updated
-----------------------	----------------------

April 22, 2024	Kick-off call
-----------------------	---------------

April 25, 2024	Added MiniEVM to scope
-----------------------	------------------------

June 31, 2024	End of primary review period
----------------------	------------------------------

3. Stdlib/Natives Findings 3.1. Unsound native-function declaration leading to critical verifier bypass

Target	table.move		
Category	Coding Mistakes	Severity	Critical
Likelihood	High	Impact	Critical

Description

The table.move module defines functions that can be used to maintain tables containing elements indexed by a key (also known as hashmaps in other contexts).

The module exposes functionality to iterate over the elements of a table, in a mutable or immutable fashion. The intended usage of the mutable version of this feature is as follows:

```
// t is a Table<K, V>
// start and end are Option<bytes> which constrain the start and end of the
// iteration
// order determines whether iteration should occur in ascending or descending
// order (sorted by the BCS serialized key)
let iter = table::iter_mut(&t, start, end, order);
loop {
    if (!table::prepare_mut<K, V>(&mut iter)) {
        break;
    }

    let (key, value) = table::next_mut<K, V>(&mut iter);
}
```

Most of the functionality of the module is supported by native functions, including iterators. In particular, iter and iter_mut use the new_table_iter and new_table_iter_mut native functions, which create an object in the VM memory and return an identifier that refers to the object:

```
native fun new_table_iter<K: copy + drop, V, B>(
    table: &Table<K, V>,
    start: vector<u8>,
    end: vector<u8>,
    order: u8
): u64;

native fun new_table_iter_mut<K: copy + drop, V, B>(
    table: &mut Table<K, V>,
```

```
start: vector<u8>,  
end: vector<u8>,  
order: u8  
): u64;
```

Due to how the native functions are defined, the Move verifier is unable to correctly track the relationship between the table object passed to the `new_table_iter[_mut]` function and the returned iterator. This makes the reference safety verifier ineffective.

Impact

This issue allows to mount a critical bypass for the security invariants enforced by the reference safety verifier, allowing for an instance to obtain two independent mutable references to the same object or to retain a mutable reference to an object even after it is transferred.

The appendix ([10.1](#), ↗) contains a proof-of-concept test that demonstrates the issue.

Recommendations

Change the implementation of the iterator functionality to allow the reference safety verifier to correctly track reference relationships. In practice, this can be achieved by ensuring to receive the type of the table (or table values) as input and returning a reference to the type of the table (or table values) as output.

Remediation

This issue has been acknowledged by Initia Labs, and a fix was implemented in [PR #44](#) ↗.

3.2. Infinite recursion possible with module dependencies

Target	crates/types/module.rs		
Category	Coding Mistakes	Severity	Critical
Likelihood	High	Impact	Critical

Description

Initia supports publishing Move modules from Move code. It is implemented as a publish request, which is posted from the Move code, and is executed at the end of the transaction. The newly published module will not be available until the next transaction. There can only be one pending request per transaction, but it is possible to publish an arbitrary number of modules simultaneously at the same owner address.

The to-be-published modules are taken as an array of binary modules, which are verified at some point before publication. However, before the verification, the following code is run to topologically sort the modules in the dependency graph.

```
pub fn sort_by_deps(
    map: &BTreeMap<ModuleId, (&[u8], CompiledModule)>,
    order: &mut Vec<ModuleId>,
    id: ModuleId,
) {
    if order.contains(&id) {
        return;
    }
    let compiled = &map.get(&id).unwrap().1;
    for dep in compiled.immediate_dependencies() {
        // Only consider deps which are actually in this package. Deps for
        // outside
        // packages are considered fine because of package deployment order.
        Note
        // that because of this detail, we can't use existing topsort from Move
        // utils.
        if map.contains_key(&dep) {
            sort_by_deps(map, order, dep);
        }
    }
    order.push(id)
}
```

Impact

If there is a cyclical dependency, this code recurses infinitely and leads to a chain halt.

Recommendations

Detect circular dependencies in this code before running this code.

Remediation

This issue has been acknowledged by Initia Labs, and a fix was implemented in commit [01d8125d](#).

3.3. Unchecked UTF-8 decoding enables memory corruption

Target	from_bcs.move		
Category	Coding Mistakes	Severity	Critical
Likelihood	Medium	Impact	High

Description

The following function fails to ensure the vector of strings is a well-formed UTF-8 before returning it:

```
public fun to_vector_string(v: vector<u8>): vector<String> {
    from_bytes<vector<String>>(v)
}
```

So, it is possible to obtain a String containing invalid UTF-8 sequences.

Impact

A malformed UTF-8 String may be able to cause an out-of-bounds (OOB) memory read, leading to a segmentation fault or potentially providing an attacker with an arbitrary read primitive — which would allow an attacker to break chain consensus by reading OOB, nondeterministic memory. Arbitrary read primitives may also be used in conjunction with other vulnerabilities to achieve arbitrary code execution.

Additionally, there is a possibility that the String may allow an attacker to overwrite bytes on the heap. Often, heap-corruption vulnerabilities can be directly exploited to achieve code execution; that is, in the worst case scenario, an attacker could create a malicious module that exploits this vulnerability to get remote code execution (RCE) on validators' machines.^[1]

Due to the limited time available for this assessment, we did not attempt to exploit this vulnerability beyond the following simple proof of concept, which demonstrates the ability to obtain a String containing invalid UTF-8 sequences:

¹ Determining whether it is possible to exploit the vulnerability to the extent of RCE is a time-consuming, difficult practice. Additionally, the presence of other bugs may be necessary for exploitation — possibly even those that are not yet discovered. Due to the limited time available for this assessment, we cannot definitively state whether or not it is possible to obtain RCE using the bug at this time (i.e., without further investigation).

```
#[test]
fun zellic_malformed_utf8_oob_read() {
    let invalid_utf8_vec = b"\x01\x01\x80";

    // obtain invalid String
    let res = to_vector_string(invalid_utf8_vec);
    assert!(!vector::is_empty(&res), 0);
    let s = vector::pop_back(&mut res);

    // trigger crash
    let needle = string::utf8(b"");
    string::index_of(&s, &needle);
}
```

The output of the test is the following:

```
thread '<unnamed>' panicked at library/core/src/panicking.rs:156:5:
unsafe precondition(s) violated: hint::unreachable_unchecked must never be
reached
stack backtrace:
0: rust_begin_unwind
   at /rustc/9b00956e56009bab2aa15d7bff10916599e3d6d6/library/std/src/
   panicking.rs:645:5
1: core::panicking::panic_nounwind_fmt::runtime
   at /rustc/9b00956e56009bab2aa15d7bff10916599e3d6d6/library/core/src/
   panicking.rs:110:18
2: core::panicking::panic_nounwind_fmt
   at /rustc/9b00956e56009bab2aa15d7bff10916599e3d6d6/library/core/src/
   panicking.rs:123:9
3: core::panicking::panic_nounwind
   at /rustc/9b00956e56009bab2aa15d7bff10916599e3d6d6/library/core/src/
   panicking.rs:156:5
4: core::hint::unreachable_unchecked::precondition_check
   at /rustc/9b00956e56009bab2aa15d7bff10916599e3d6d6/library/core/src/
   intrinsics.rs:2799:21
5: core::hint::unreachable_unchecked
   at /rustc/9b00956e56009bab2aa15d7bff10916599e3d6d6/library/core/src/
   hint.rs:101:5
6: core::option::Option<T>::unwrap_unchecked
   at /rustc/9b00956e56009bab2aa15d7bff10916599e3d6d6/library/core/src/
   option.rs:1041:30
7: core::str::validations::next_code_point
...

```

Following the stack trace, we can identify the cause of the crash as being the [following code](#):

```
pub const unsafe fn unwrap_unchecked(self) -> T {  
    match self {  
        Some(val) => val,  
        // SAFETY: the safety contract must be upheld by the caller.  
        None => unsafe { hint::unreachable_unchecked() },  
    }  
}
```

Recommendations

Ensure all Strings in the vector are valid before returning the vector:

```
public fun to_vector_string(v: vector<u8>): vector<String> {  
    from_bytes<vector<String>>>(v)  
    let vec_string = from_bytes<vector<String>>>(v);  
    vector::for_each_ref(&vec_string, |s| {  
        assert!(string::internal_check_utf8(string::bytes(s)), EINVAL_UTF8);  
    });  
    vec_string  
}
```

Remediation

This issue has been acknowledged by Initia Labs, and a fix was implemented in [PR #46](#).

3.4. Missing gas charge on memo in native_nft_transfer

Target	cosmos.rs		
Category	Coding Mistakes	Severity	High
Likelihood	High	Impact	High

Description

The native implementation `native_nft_transfer` function in `cosmos.rs` does not charge gas for the `memo` argument.

Impact

In a DOS attack, an attacker can create a transaction with a large `memo` field and spam the network with transactions that do not pay the full, appropriate gas fees.

Recommendations

Add the following line to the `native_nft_transfer` function in `cosmos.rs`:

```
fn native_nft_transfer(
    context: &mut SafeNativeContext,
    ty_args: Vec<Type>,
    mut arguments: VecDeque<Value>,
) -> SafeNativeResult<SmallVec<[Value; 1]>> {
    let gas_params =
        &context.native_gas_params.initia_stdlib.cosmos.nft_transfer;
    context.charge(gas_params.base)?;

    debug_assert!(ty_args.is_empty());
    debug_assert!(arguments.len() == 10);

    let memo = safely_pop_arg!(arguments, Vector).to_vec_u8()?;
    context.charge(gas_params.per_byte * NumBytes::new(memo.len() as u64))?;
    let timeout_timestamp = safely_pop_arg!(arguments, u64);
    let revision_height = safely_pop_arg!(arguments, u64);
    let revision_number = safely_pop_arg!(arguments, u64);
    let source_channel = safely_pop_arg!(arguments, Vector).to_vec_u8()?;
    context.charge(gas_params.per_byte * NumBytes::new(source_channel.len()
    as u64))?;
```



```
// [...]  
}
```

Remediation

This issue has been acknowledged by Initia Labs, and a fix was implemented in [PR #43](#).

3.5. Next zapping ID potentially duplicated

Target	vip/zapping.move		
Category	Coding Mistakes	Severity	High
Likelihood	High	Impact	High

Description

The Initia chain allows vesting rewards from L2s to be released early and put into a liquidity pool. The liquidity token is then locked for the required period. This is known as zapping. When a user uses this feature, a Zapping object is created to record the details of the zapping. Each object has a unique ID, which is used to look it up in a table. The code to determine the ID is as follows.

```
let zid = table::length(&module_store.zappings);

assert!(!table::contains(&module_store.zappings, zid),
    error::already_exists(EZAPPING_ALREADY_EXIST));
```

If deletions have happened in the table, for example if a user has fully released their zapping, the length of the table may no longer match the last ID. At this point, it would no longer be possible to add new zappings until a sufficient number is deleted.

Impact

This is a highly likely scenario causing the zapping feature to be unusable. We suspect in practice the above DOS condition will consistently hold true.

Recommendations

Use a ID-generation scheme uncorrelated to the length of the table.

Remediation

This issue has been acknowledged by Initia Labs, and a fix was implemented in [PR #58](#).

3.6. Published module names do not necessarily match binary module

Target	code.move		
Category	Coding Mistakes	Severity	High
Likelihood	N/A	Impact	Informational

Description

Initia supports publishing Move modules from Move code. It is implemented as a publish request, which is posted from the Move code, and is executed at the end of the transaction. The newly published module will not be available until the next transaction. There can only be one pending request per transaction, but it is possible to publish an arbitrary number of modules simultaneously at the same owner address.

The modules are taken as two parallel arrays of the same length, one for the names of the modules to be published and the other for the actual binary modules to be published. In the following code, the module names are used for checking that the upgrade policy of the module allows the action that is about to be taken, but the binary modules are the actually published modules.

```
public entry fun publish(
    owner: &signer,
    module_ids: vector<String>, // 0x1::coin
    code: vector<vector<u8>>,
    upgrade_policy: u8,
) acquires ModuleStore, MetadataStore {
    // Disallow incompatible upgrade mode. Governance can decide later if this
    // should be reconsidered.
    assert!(vector::length(&code) == vector::length(&module_ids),
        error::invalid_argument(EINVALID_ARGUMENTS));

    // [...]

    // Check upgradability
    let metadata_table = &mut borrow_global_mut<MetadataStore>(addr).metadata;
    vector::for_each_ref(&module_ids,
        |module_id| {
            if (table::contains<String, ModuleMetadata>(metadata_table,
                *module_id)) {
                let metadata = table::borrow_mut<String,
                    ModuleMetadata>(metadata_table, *module_id);
                assert!(metadata.upgrade_policy < UPGRADE_POLICY_IMMUTABLE,
                    error::invalid_argument(EUPGRADE_IMMUTABLE));
```

```

        assert!(can_change_upgrade_policy_to(metadata.upgrade_policy,
upgrade_policy),
            error::invalid_argument(EUPGRADE_WEAKER_POLICY));
// [...]

```

Note that there is no check that the i -th element in `module_ids` matches the name of the i -th element in code.

Impact

On the surface, it looks like it is possible to publish a module while using and updating the upgrade policy of another module, allowing for a bypass of the module-upgrade restrictions. However, due to a coincidence of the structure of the code, the impact is completely mitigated.

The passed-in module names are first put into a set in the following code.

```

let mut expected_modules: BTreeSet<String> = BTreeSet::new();
for name in safely_pop_vec_arg!(arguments, Struct) {
    let str_bytes = get_string(name)?;

    context.charge(gas_params.per_byte * NumBytes::new(str_bytes.len()
as u64))?;
    expected_modules.insert(String::from_utf8(str_bytes).map_err(|_| {
        SafeNativeError::Abort {
            abort_code: EUNABLE_TO_PARSE_STRING,
        }
    }));
}

```

Right before publishing, the following check is performed to ensure that all published modules have been named in the name list.

```

if let Some(mut expected_modules) = expected_modules {
    for m in modules {
        if !expected_modules.remove(m.self_id().short_str_lossless().as_str())
        {
            return Err(metadata_validation_error(&format!(
                "unregistered module: '{}'",
                m.self_id().name()
            )));
        }
    }
}

```

These two checks ensure that while the wrong permissions may be checked for a given module, any published module must be named in the list of modules names. This means any published module's permissions must be checked anyway, even if it is in the incorrect slot in the names array.

Recommendations

Ensure that the names of the modules match the actual name of the binary module.

Remediation

This issue has been acknowledged by Initia Labs, and a fix was implemented in commit [ae04bf3f](#).

3.7. Ability to bypass swap fees

Target	dex.move, stableswap.move, miniswap.move		
Category	Coding Mistakes	Severity	Low
Likelihood	Medium	Impact	Low

Description

The following code calculates the fee amount for a swap:

```
public fun swap_simulation(
    pool_amount_in: u64,
    pool_amount_out: u64,
    weight_in: Decimal128,
    weight_out: Decimal128,
    amount_in: u64,
    swap_fee_rate: Decimal128,
): (u64, u64) {
    // [...]
    let fee_amount = decimal128::mul_u64(&swap_fee_rate, amount_in);
    let adjusted_amount_in = amount_in - fee_amount;
    // [...]
}

public fun mul_u64(decimal: &Decimal128, val: u64): u64 {
    (decimal.val
        * (val as u128)
        / DECIMAL_FRACTIONAL
        as u64)
}
```

It is possible to bypass swap fees by swapping a low amount such that the precision loss from division when calculating fees rounds the fee amount to zero.

Impact

Note that the small swap could be executed repeatedly to equate to one larger swap.^[2]

We provided the following proof of concept (placed in dex.move) to demonstrate exploitability of this issue:

```
#[test]
fun zellic_exploit_rounding_bypass_fees() {
    let pool_amount_in = 100000000;
    let pool_amount_out = 100000000;
    let weight_in = decimal128::from_ratio(1, 1);
    let weight_out = decimal128::from_ratio(1, 1);

    let swap_fee_rate = decimal128::from_ratio(3, 1000);
    // 1 / swap_fee_rate - 1
    let max_avoid_fee_amount = ((1000 / 3 - 1) as u64);
    assert!(max_avoid_fee_amount != 0, 0); // doesn't prove anything if you get
    nothing from it

    let (return_amount, fee_amount) = swap_simulation(
        pool_amount_in,
        pool_amount_out,
        weight_in,
        weight_out,
        max_avoid_fee_amount,
        swap_fee_rate,
    );

    assert!(fee_amount == 0, 1); // we paid no fees
    assert!(return_amount != 0, 2); // we got something back (the amount isn't
    relevant)
}
```

Recommendations

We recommend always rounding the fee calculation in favor of the protocol instead of the user. This way, as swaps get smaller, fees get proportionally higher instead of proportionally smaller.

² There would still be a (relatively small) loss to the user because the amount-out calculation rounds in favor of the protocol, which is good, but fees may still be bypassed.

Remediation

This issue has been acknowledged by Initia Labs, and a fix was implemented in [PR #43](#). The patch was also updated in [PR #59](#) to further prevent exploitation.

3.8. Module can be duplicated in module publish requests

Target	code.move		
Category	Coding Mistakes	Severity	Low
Likelihood	Low	Impact	Low

Description

Initia supports publishing Move modules from Move code. It is implemented as a publish request, which is posted from the Move code, and is executed at the end of the transaction. The newly published module will not be available until the next transaction. There can only be one pending request per transaction, but it is possible to publish an arbitrary number of modules simultaneously at the same owner address.

The following is the signature for the Move function to submit a publication request.

```
public entry fun publish(
    owner: &signer,
    module_ids: vector<String>, // 0x1::coin
    code: vector<vector<u8>>,
    upgrade_policy: u8,
) acquires ModuleStore, MetadataStore {
```

The two arrays, `module_ids` and `code`, can both contain duplicates. The same module name may appear multiple times in `module_ids`, and binary modules with the same name may appear multiple times in `code`.

The `module_ids` are immediately deduplicated in the native code.

```
let mut expected_modules: BTreeSet<String> = BTreeSet::new();
for name in safely_pop_vec_arg!(arguments, Struct) {
    let str_bytes = get_string(name)?;

    context.charge(gas_params.per_byte * NumBytes::new(str_bytes.len()
as u64))?;
    expected_modules.insert(String::from_utf8(str_bytes).map_err(|_| {
        SafeNativeError::Abort {
            abort_code: EUNABLE_TO_PARSE_STRING,
        }
    }));
}
```

Note the use of `BTreeSet`, which is a deduplicating container.

The binary modules in code eventually arrive at the following code, which also deduplicates the modules, in a last-overrides-first manner.

```
let mut map = codes
    .iter()
    .map(|c| {
        let m = CompiledModule::deserialize(c).unwrap();
        (m.self_id(), (c.as_slice(), m))
    })
    .collect::<BTreeMap<_, _>>();
```

Impact

Fortunately, the impact is limited to unintuitive behavior. Only the final set of deduplicated modules are actually used, with some duplicates silently discarded.

Recommendations

We recommend that duplicate checks be added and an intuitive error be returned.

Remediation

This issue has been acknowledged by Initia Labs, and fixes were implemented in the following commits:

- [ae04bf3f](#) ↗
- [83cb42b5](#) ↗

3.9. Hex decode accepting invalid characters

Target	hex.move		
Category	Coding Mistakes	Severity	Low
Likelihood	Low	Impact	Low

Description

The hex.move module provides helper functions that can be used to convert a bytes array to and from a hexadecimal representation.

The decode_string accepts characters that are not valid hexadecimals and decodes characters that are not in the range 0-9a-f. This causes the strings 1f and 0g to be decoded to the value 0x1f.

Impact

This issue is reported as low impact since we could not identify an exploitation path using only the in-scope code. However, the impact could be higher for third-party modules.

Recommendations

Validate the input to ensure it contains only valid hexadecimal characters.

Remediation

This issue has been acknowledged by Initia Labs, and a fix was implemented in [PR #36](#).

3.10. Stablepools can be created with one or no assets

Target	stableswap.move		
Category	Coding Mistakes	Severity	High
Likelihood	Low	Impact	Low

Description

The stableswap.move module implements an AMM based on the Curve StableSwap price function. The create_pair function can be used to create a new AMM pool with an arbitrary number of assets:

```
public fun create_pair(
    creator: &signer,
    name: String,
    symbol: String,
    swap_fee_rate: Decimal128,
    coins: vector<FungibleAsset>,
    ann: u64,
): FungibleAsset acquires Pool, ModuleStore
```

The function does not require the number of assets to be at least two, allowing to create a pool consisting of just one or even no assets at all.

Impact

This issue is reported as low impact since we consider it unlikely to be exploitable to cause damage to third parties. Pools with one or zero assets cause several of the module functions to revert, as they were written assuming pools contain at least two assets. However, considering the potential economic impact, we still classify this issue as high severity.

Recommendations

Require the number of assets to be at least two when creating a new AMM pool. This can be done by adding an assert!(coins.length() >= 2) statement to create_pair.

Remediation

This issue has been acknowledged by Initia Labs, and a fix was implemented in [PR #42](#).

3.11. Bad decimal-parsing function accepts multiple dots

Target	decimal128.move, decimal256.move		
Category	Coding Mistakes	Severity	Low
Likelihood	Low	Impact	Low

Description

The decimal-parsing function shown below is incorrectly implemented. A variant of this function is used in all targets listed.

```
public fun from_string(num: &String): Decimal256 {
    let vec = string::bytes(num);
    let len = vector::length(vec);

    let cursor = 0;
    let dot_index = 0;
    let val: u256 = 0;
    while (cursor < len) {
        let s = *vector::borrow(vec, cursor);
        cursor = cursor + 1;

        // find `.` position
        if (s == 46) continue;

        val = val * 10;
        assert!(s >= 48 && s <= 57,
            error::invalid_argument(EFAILED_TO_DESERIALIZE));

        let n = (s - 48 as u256);
        val = val + n;

        if (cursor == dot_index + 1) {
            // use `<` not `<=` to safely check "out of range"
            // (i.e. to avoid fractional part checking)
            assert!(val < MAX_INTEGER_PART,
                error::invalid_argument(EOUT_OF_RANGE));

            dot_index = dot_index + 1;
        }
    };
};
```

```
// ignore fractional part longer than `FRACTIONAL_LENGTH`  
let val = if (dot_index == len) {  
    val * pow(10, FRACTIONAL_LENGTH)  
} else {  
    let fractional_length = len - dot_index - 1;  
    if (fractional_length > FRACTIONAL_LENGTH) {  
        val / pow(10, fractional_length - FRACTIONAL_LENGTH)  
    } else {  
        val * pow(10, FRACTIONAL_LENGTH - fractional_length)  
    }  
};  
  
new(val)  
}
```

If there is more than one dot, it will still parse the value, although it will return a corrupted result.

Impact

Since output from this function from the user should be checked anyway, we have labelled this as low impact. Any attacker trying to take advantage of this bug could instead simply enter the correct formatting of the number that would be returned by an incorrect parsing of this function.

Recommendations

Adjust the algorithm to handle inputs with multiple dots in a more intuitive manner.

Remediation

This issue has been acknowledged by Initia Labs, and a fix was implemented in [PR #59](#).

3.12. Potential out-of-gas reversion when checking object permissions

Target	object.move		
Category	Coding Mistakes	Severity	Low
Likelihood	Low	Impact	Low

Description

In Initia, an object can belong to another object, recursively. An object is allowed to be transferred without the TransferRef special permission if the path to the owner (any ancestor) that is currently transferring the object has the allow_ungated_transfer flag set. This means all objects on that ownership path must have that flag set and that the path must end in the "currently transferring owner", which must sign the operation.

In order to check this condition, the following loop is used.

```
let count = 0;
while (owner != current_address) {
    let count = count + 1;
    assert!(count < MAXIMUM_OBJECT_NESTING,
        error::out_of_range(EMAXIMUM_NESTING));

    // At this point, the first object exists and so the more likely case is
    // that the
    // object's owner is not an object. So we return a more sensible error.
    assert!(
        exists<ObjectCore>(current_address),
        error::permission_denied(ENOT_OBJECT_OWNER),
    );
    let object = borrow_global<ObjectCore>(current_address);
    assert!(
        object.allow_ungated_transfer,
        error::permission_denied(ENO_UNGATED_TRANSFERS),
    );

    current_address = object.owner;
};
```

Note that the buggy section of the code shown below appears in two places in both copies of object.move, for a total of four occurrences.

```
let count = 0;
while (owner != current_address) {
    let count = count + 1;
    assert!(count < MAXIMUM_OBJECT_NESTING,
error::out_of_range(EMAXIMUM_NESTING));
```

The variable `count` inside the loop shadows the variable outside the loop, which makes it always have the value 1 and always pass the check.

Impact

If this loop continues for too long, either due to a dependency cycle or if the path is simply too long, there may be an out-of-gas condition. Since the correct functioning of the code would result in a revert anyway, this does not have a severe impact.

Recommendations

Make the `count` variable no longer shadowed.

Remediation

This issue has been acknowledged by Initia Labs, and a fix was implemented in [PR #54](#).

3.13. User-triggerable invariant violation

Target	ed25519.move		
Category	Coding Mistakes	Severity	Informational
Likelihood	N/A	Impact	Informational

Description

The ed25519.move module defines functions that can be used to verify ed25519 signatures. In addition to a verify function allowing to verify a single signature, the module also supports a batch_verify function that can be used to perform a more efficient batch verification of the following cases (quoting from the module documentation):

```
/// - Equal number of messages, signatures, and public keys: Standard, generic
    functionality.
/// - One message, and an equal number of signatures and public keys: Multiple
    digital signature
/// (multisig) verification of a single message.
/// - One public key, and an equal number of messages and signatures:
    Verification of multiple
/// messages, all signed with the same private key.
```

The caller is supposed to provide three input arrays with a matching number of elements:

```
public fun batch_verify(
    messages: vector<vector<u8>>,
    public_keys: vector<PublicKey>,
    signatures: vector<Signature>,
)
```

The native function that performs batch signature verification does check that the number of elements in the arrays matches a supported scenario, but in case of a mismatch, it returns an invariant violation error, which is reserved for cases where the Move VM has detected an invariant violation that should never occur, such as an inconsistency due to a verifier bug.

Impact

This issue is reported as informational, as it does not allow to bypass any security invariant. It causes an inappropriate error to be returned when calling batch_verify with incorrectly sized arrays.

Recommendations

Return an appropriate error if the number of elements provided as arguments to `batch_verify` by the user is incorrect.

Remediation

This issue has been acknowledged by Initia Labs, and a fix was implemented in [PR #48](#).

3.14. Stablepool swap can be called, repeating the same asset

Target	stableswap.move		
Category	Coding Mistakes	Severity	High
Likelihood	Low	Impact	Informational

Description

The stableswap.move module implements an AMM based on the Curve StableSwap price function.

The swap function can be used to perform a swap between assets contained in the pool:

```
public fun swap(
    pair: Object<Pool>,
    offer_coin: FungibleAsset,
    return_coin_metadata: Object<Metadata>,
    min_return_amount: Option<u64>
): FungibleAsset acquires Pool
```

The function reverts if offer_coin or return_coin_metadata are not assets contained in the pool. However, it does not require offer_coin and return_coin_metadata to refer to two different assets.

Impact

This issue is reported as informational as we were unable to circumvent an unintentional revert due to an underflow in one of the helper functions called to perform the swap calculations, get_y. However, other third-party modules that rely on the AMM may be impacted to a greater extent, even by a denial-of-service condition. Considering the importance of the AMM modules on the ecosystem and the potential — even if undetermined — impact on third-party modules, we classify this issue as high severity.

Recommendations

Require the input and output assets in a swap to differ from each other.

Remediation

This issue has been acknowledged by Initia Labs, and a fix was implemented in [PR #42](#).

3.15. Incorrect string-formatting helper

Target	string_utils.move		
Category	Coding Mistakes	Severity	Low
Likelihood	Low	Impact	Low

Description

The string_utils.move module implements several helper functions that allow to format an instance of any type to string. The module exposes several functions that differ in how types are serialized.

One of these functions is to_string_with_integer_types, which should serialize integers with a suffix that explicitly specifies their type (e.g., 123u8). However, the function is implemented incorrectly, invoking the native_format internal function with an incorrect argument:

```
public fun to_string_with_integer_types<T>(s: &T): String {
    native_format(s, false, true, true, false)
}

native fun native_format<T>(
    s: &T, type_tag: bool,
    canonicalize: bool,
    single_line: bool,
    include_int_types: bool
): String;
```

The include_int_types argument is hardcoded to false, while it should have been true to implement the intended behavior.

Impact

This issue is reported as low impact as we were unable to identify any concrete impact on the rest of the in-scope code. While third-party modules could be affected to an unpredictable and potentially high extent, we consider this issue unlikely to cause high-severity issues.

Recommendations

Correctly pass the include_int_types parameter as true in to_string_with_integer_types.

Remediation

This issue has been acknowledged by Initia Labs, and a fix was implemented in [PR #69 ↗](#).

3.16. Incorrect minimum-TVL module-parameter check

Target	vip/vip.move		
Category	Coding Mistakes	Severity	Low
Likelihood	Low	Impact	Low

Description

The vip module has two parameters, the `minimum_tv1` and the `maximum_tv1`. These parameters can be updated with the chain signer. An invariant requires that `minimum_tv1` always be less than or equal to `maximum_tv1`.

The function to update the `minimum_tv1` is as follows.

```
public entry fun update_minimum_tv1(
    chain: &signer,
    minimum_tv1: u64,
) acquires ModuleStore {
    check_chain_permission(chain);
    let module_store =
        borrow_global_mut<ModuleStore>(signer::address_of(chain));
    assert!(minimum_tv1 >= 0, error::invalid_argument(EINVALID_MIN_TV1));
    module_store.minimum_tv1 = minimum_tv1;
}
```

Note that the check for the `minimum_tv1` is incorrect and always passes. It should check that the `minimum_tv1` is less than the `maximum_tv1`.

Impact

It is possible to create a condition where the invariant is violated by calling this function. This issue is low impact since only an improperly formed call with chain permissions can trigger it.

Recommendations

Fix the check to correctly ensure the TVL limits are reasonable.

Remediation

This issue has been acknowledged by Initia Labs, and a fix was implemented in [PR #58 ↗](#).

3.17. The upgrade policy can be set to a large value

Target	code.move		
Category	Coding Mistakes	Severity	Informational
Likelihood	N/A	Impact	Informational

Description

Initia supports publishing Move modules from Move code. It is implemented as a publish request, which is posted from the Move code, and is executed at the end of the transaction. The newly published module will not be available until the next transaction. There can only be one pending request per transaction, but it is possible to publish an arbitrary number of modules simultaneously at the same owner address.

The upgrade policy is a number associated with each module, specifying what kind of upgrades can be done to the module. When performing an upgrade, the upgrade policy can be updated to be more restrictive. There is no limit on the maximum value of the upgrade policy in the code, which allows for setting out-of-range upgrade-policy values.

Impact

There is no impact, since the behavior of large policy values is identical to the policy forbidding all upgrades.

Recommendations

Consider restricting the range of upgrade-policy values.

Remediation

This issue has been acknowledged by Initia Labs, and a fix was implemented in commit [01d8125d](#).

3.18. Call to next on iterator without prepare aborts

Target	table.rs		
Category	Coding Mistakes	Severity	Informational
Likelihood	N/A	Impact	Informational

Description

To use a table iterator from Move, the code must first create an iter using the `iter` or `iter_mut` functions. Then, normally, the code should use the `prepare` function, followed by the `next` or `next_mut` functions.

However, if the `prepare` call is skipped, an assertion in the native implementation of `next` (and `next_mut`) at `native_next_box` will fail:

```
fn native_next_box(
    context: &mut SafeNativeContext,
    ty_args: Vec<Type>,
    mut arguments: VecDeque<Value>,
) -> SafeNativeResult<SmallVec<Value; 1>> {
    // [...]
    let iterator_id = get_iterator_id(&safely_pop_arg!(arguments, StructRef))?
    as usize;

    let table_context = context.extensions().get:::<NativeTableContext>();
    let mut iterators = table_context.iterators.borrow_mut();
    let iterator = iterators.get_mut(iterator_id).unwrap();

    assert!(iterator.next.is_some());

    let (key, value) = iterator.next.take().unwrap();
    iterator.next = None;

    Ok(smallvec![key, value])
}
```

The `iterator.next` is usually set in `native_prepare_box`.

Impact

The following test demonstrates the abort:

```
#[test(s = @0x42)]
fun test_zellic_next_abort() {
    let t = T::new<u64, u64>();

    // 1. create an iterator that has "next: None"
    let iter = T::iter(&t, option::none<u64>(), option::none<u64>(), 1);

    // 2. next to break assert
    let (key, value) = T::next<u64, u64>(&mut iter);

    T::drop_unchecked(t)
}
```

```
test tests::compiler_tests::test_move_compile ... ok
failures:
---- tests::compiler_tests::test_move_test stdout ----
thread '<unnamed>' panicked at crates/natives/src/table.rs:687:5:
assertion failed: iterator.next.is_some()
note: run with `RUST_BACKTRACE=1` environment variable to display a backtrace

failures:
    tests::compiler_tests::test_move_test
test result: FAILED. 4 passed; 1 failed; 0 ignored; 0 measured; 0 filtered out; finished in 21.20s
error: test failed, to rerun pass `~p initia-move-compiler --lib`
make: *** [Makefile:57: test-compiler] Error 101
```

Recommendations

Consider properly aborting with an abort code rather than just an assertion failure.

Remediation

This issue has been acknowledged by Initia Labs, and a fix was implemented in [PR #34](#).

3.19. Unnecessarily verbose branching in operator.move

Target	vip/operator.move		
Category	Code Maturity	Severity	Informational
Likelihood	N/A	Impact	Informational

Description

The following code in operator.move can be refactored to only have a single assert, with the branch for computing the value only.

```
if (old_commission_rate > new_commission_rate) {
  let change = old_commission_rate - new_commission_rate;
  assert!(change <= max_commission_change_rate,
    error::invalid_argument(EINVALID_COMMISSION_CHANGE_RATE));
} else {
  let change = new_commission_rate - old_commission_rate;
  assert!(change <= max_commission_change_rate,
    error::invalid_argument(EINVALID_COMMISSION_CHANGE_RATE));
};
```

Impact

This code is unnecessarily verbose and can be refactored to be more concise.

Recommendations

Consider refactoring the code as suggested.

Remediation

This issue has been acknowledged by Initia Labs, and a fix was implemented in [PR #58](#).

4. Initia Findings

4.1. Frozen module coin store can cause chain halt

Target	x/distribution/keeper/allocation.go		
Category	Coding Mistakes	Severity	High
Likelihood	High	Impact	High

Description

The AllocateTokens function is called by the distributions module before each block in the Begin-Blocker handler:

```
func (k Keeper) AllocateTokens(ctx context.Context, totalPreviousPower int64,
bondedVotes []abci.VoteInfo) error {
    if err := k.beforeAllocateTokens(ctx); err != nil {
        return err
    }

    // fetch and clear the collected fees for distribution, since this is
    // called in BeginBlock, collected fees will be from the previous block
    // (and distributed to the previous proposer)
    feeCollector := k.authKeeper.GetModuleAccount(ctx, k.feeCollectorName)
    feesCollectedInt := k.bankKeeper.GetAllBalances(ctx,
    feeCollector.GetAddress())
    feesCollected := sdk.NewDecCoinsFromCoins(feesCollectedInt...)

    // transfer collected fees to the distribution module account
    err := k.bankKeeper.SendCoinsFromModuleToModule(ctx, k.feeCollectorName,
    types.ModuleName, feesCollectedInt)
    if err != nil {
        return err
    }
}
```

This will fetch the balance for any coins sent to the fee collector and send it to the distribution model, which ends up calling `0x1::coin::transfer` in the Move VM.

The issue is that it is possible for someone to initialize a coin, freeze the distribution-module accounts store for that coin using the `FreezeCapability`, and then mint some coins to the fee module, causing the call to `SendCoinsFromModuleToModule` to fail.

Impact

A malicious user can cause the `AllocateTokens` method to return an error, and since it occurs in a `BeginBlocker`, it will cause a consensus failure and halt the chain.

Recommendations

The safest option to resolve this issue could be to have an allowlist of coins that the fee module can distribute instead of fetching all balances.

Remediation

This issue has been acknowledged by Initia Labs, and a fix was implemented in [PR #33](#).

4.2. Malicious proposer can skip fee check

Target	x/move/ante/fee.go		
Category	Coding Mistakes	Severity	High
Likelihood	Medium	Impact	High

Description

When a transaction is submitted to the mempool, the MempoolFeeChecker is run to ensure that the correct fees have been paid to cover the gas:

```
func (fc MempoolFeeChecker) CheckTxFeeWithMinGasPrices(ctx sdk.Context, tx
sdk.Tx) (sdk.Coins, int64, error) {
    feeTx, ok := tx.(sdk.FeeTx)
    if !ok {
        return nil, 0, errors.Wrap(sdkerrors.ErrTxDecode, "Tx must be a FeeTx")
    }

    feeCoins := feeTx.GetFee()
    gas := feeTx.GetGas()

    priority := int64(1)
    if ctx.IsCheckTx() {
        minGasPrices := ctx.MinGasPrices()
        feeValueInBaseUnit := math.ZeroInt()
```

The issue is that the check is only run on IsCheckTx, which means it is not run for transactions proposed in a block. A malicious proposer could include a transaction with an arbitrary amount of gas and no fees.

Impact

A malicious proposer could publish a Move module containing an endless loop, then when it is their turn to propose a block, include a transaction with 0x7fffffffffffffffff gas and no fees that executes the function, essentially halting the chain with all the validators stuck in the loop.

In order for a validator to propose a block they must be bonded and have staked enough to have enough consensus power to be chosen as the proposer.

Recommendations

Consider always running the MempoolFeeChecker; otherwise, ensure that a malicious proposer can be slashed for halting the chain.

Remediation

This issue has been acknowledged by Initia Labs.

Initia Labs provided the following response:

I think we can assume this is not that problematic in these two reason. The Cosmos-SDK have one guard (`block_gas_limit`) to prevent this case. We plan to set this to 200_000_000 for testnet, and then the tx spending higher than this will be ignored. Probably we can also make params `tx_gas_limit` to limit the gas consumption of each contract execution. A malicious proposer can not generate a block commonly due to voting power aspect, and if the malicious proposal could be an active proposer that means this proposer invested on INIT token.

4.3. A malicious user can become a permissioned relay for IBC

Target	app/hook/bridge_hook.go		
Category	Coding Mistakes	Severity	Medium
Likelihood	Medium	Impact	Medium

Description

When a new bridge is created, a hook is run to register the challenger as the permissioned relay for any configured IBC channels:

```
func (h BridgeHook) BridgeCreated(
    ctx context.Context,
    bridgeId uint64,
    bridgeConfig ophosttypes.BridgeConfig,
) error {
    hasPermChannels, metadata := hasPermChannels(bridgeConfig.Metadata)
    if !hasPermChannels {
        return nil
    }

    challenger, err := h.ac.StringToBytes(bridgeConfig.Challenger)
    if err != nil {
        return err
    }

    sdkCtx := sdk.UnwrapSDKContext(ctx)
    for _, permChannel := range metadata.PermChannels {
        portID, channelID := permChannel.PortID, permChannel.ChannelID
        if seq, ok := h.IBCChannelKeeper.GetNextSequenceSend(sdkCtx, portID,
            channelID); !ok {
            return channeltypes.ErrChannelNotFound.Wrap("failed to register
            permissioned relay")
        } else if seq != 1 {
            return channeltypes.ErrChannelExists.Wrap("cannot register
            permissioned relay for the channel in use")
        }

        // register challenger as channel relay
        if err = h.IBCPermKeeper.SetPermissionedRelayer(sdkCtx, portID,
            channelID, challenger); err != nil {
            return err
        }
    }
}
```



```
    }  
  }  
  
  return nil  
}
```

The IBC channel must not be in use, which is checked by ensuring that the sequence number for it is one.

The issue is that a malicious user can watch for newly created IBC channels, then create a new bridge to become the permissioned relayer for the channel.

When the real bridge is created, the correct challenger will become the channel relayer, but the malicious user can update their bridge to take it over again:

```
func (h BridgeHook) BridgeChallengerUpdated(  
    ctx context.Context,  
    bridgeId uint64,  
    bridgeConfig ophosttypes.BridgeConfig,  
) error {  
    hasPermChannels, metadata := hasPermChannels(bridgeConfig.Metadata)  
    if !hasPermChannels {  
        return nil  
    }  
  
    challenger, err := h.ac.StringToBytes(bridgeConfig.Challenger)  
    if err != nil {  
        return err  
    }  
  
    sdkCtx := sdk.UnwrapSDKContext(ctx)  
    for _, permChannel := range metadata.PermChannels {  
        portID, channelID := permChannel.PortID, permChannel.ChannelID  
  
        // update relayer to a new challenger  
        if err = h.IBCPermKeeper.SetPermissionedRelayer(sdkCtx, portID,  
            channelID, challenger); err != nil {  
            return err  
        }  
    }  
  
    return nil  
}
```

Impact

A malicious user can become the permissioned relayer for newly created IBC channels and continue to take over the relayer even after the real bridge is created.

Recommendations

If the permissioned relayer is already set, an error should be raised to indicate that someone else has already claimed the channel.

If possible, the IBC channel and bridge creation should happen within the same transaction to prevent anyone from being able to take over the channel.

Remediation

This issue has been acknowledged by Initia Labs, and a fix was implemented in [PR #191](#).

4.4. Move coins can be burned twice

Target	x/move/keeper/bank.go		
Category	Coding Mistakes	Severity	High
Likelihood	Medium	Impact	Medium

Description

The BurnCoins method has special handling for Move coins not generated by 0x1 as they do not have the burn capability for them. Instead, they are sent to the community pool.

```
// BurnCoins burn coins or send to community pool.
func (k MoveBankKeeper) BurnCoins(
    ctx context.Context,
    accAddr sdk.AccAddress,
    coins sdk.Coins,
) error {
    for _, coin := range coins {
        // if a coin is not generated from 0x1, then send the coin to community
        pool
        // because we don't have burn capability.
        if types.IsMoveCoin(coin) {
            if err := k.communityPoolKeeper.FundCommunityPool(ctx, coins,
                accAddr); err != nil {
                return err
            }

            continue
        }
        // send tokens to 0x1
        err := k.SendCoin(ctx, accAddr, types.StdAddr, coin.Denom,
            coin.Amount)
        if err != nil {
            return err
        }
        // execute burn
        metadata, err := types.MetadataAddressFromDenom(coin.Denom)
        if err != nil {
            return err
        }
    }
}
```

The issue is that the `coins` parameter is passed to `FundCommunityPool`, which contains every coin instead of only the `coin` that matched the condition.

Impact

If one of the coins returns true for `types.IsMoveCoin(coin)`, then all the coins are sent to `FundCommunityPool`, not just the matching one. Then, the next iteration will send the coins again to `0x1` to be burned. If two Move coins are provided, then double of each will be sent to the fund. If one is Move and the other is not, then the other coin will be sent to the fund as well as burned.

Recommendations

Only the matching `coin` should be sent to the community pool.

Remediation

This issue has been acknowledged by Initia Labs, and a fix was implemented in [PR #132](#).

4.5. Move coin transfer can bypass blocked accounts

Target	x/bank/keeper/msg_server.go		
Category	Coding Mistakes	Severity	Medium
Likelihood	Medium	Impact	Low

Description

When sending coins from one address to another using `MsgSend`, the coins are checked to ensure that sending is enabled and that the receiver is not blocked:

```
ctx := sdk.UnwrapSDKContext(goCtx)
if err := k.IsSendEnabledCoins(ctx, msg.Amount...); err != nil {
    return nil, err
}

if k.BlockedAddr(to) {
    return nil, errorsmod.Wrapf(sdkerrors.ErrUnauthorized, "%s is not allowed
to receive funds", msg.ToAddress)
}
```

Once validated, the coins are sent using `SendCoins`, which will end up calling into the Move VM and executing `0x1::coin::transfer`.

The issue is that there is nothing stopping someone from directly calling `0x1::coin::transfer` in the Move VM, bypassing both the `IsSendEnabledCoins` and `BlockedAddr` checks.

Impact

A malicious user can send a coin that has sending disabled or transfer a coin to a blocked address.

Recommendations

If the blocking and sending functionality is required, it could be implemented inside the Move VM instead of relying on the default Cosmos SDK implementation, or the GoAPI could be extended to expose this information.

Remediation

This issue has been acknowledged by Initia Labs, and a fix was implemented in [PR #111](#).

Initia Labs provided the following response:

Those features are not used (no effect even we use it) in our chain. We've changed invariant too in the staking and distribution module to allow funding to blocked accounts.

4.6. Error not checked when fetching starting info

Target	x/distribution/keeper/delegation.go		
Category	Coding Mistakes	Severity	Low
Likelihood	Low	Impact	Low

Description

In `CalculateDelegationRewards`, the starting info for a delegation is fetching using the validator and delegator address pair:

```
valAddr, err :=
    k.stakingKeeper.ValidatorAddressCodec().StringToBytes(del.GetValidatorAddr())
if err != nil {
    return nil, err
}

// fetch starting info for delegation
startingInfo, err := k.DelegatorStartingInfos.Get(ctx,
    collections.Join(valAddr, delAddr))

sdkCtx := sdk.UnwrapSDKContext(ctx)
if startingInfo.Height == uint64(sdkCtx.BlockHeight()) {
    // started this height, no rewards yet
    return
}

startingPeriod := startingInfo.PreviousPeriod
stakes := startingInfo.Stakes
```

The issue is that returned `err` is not checked, resulting in `startingInfo` being `nil` or containing unexpected values.

Impact

The `DelegatorStartingInfos` should always be set if there is a valid delegation, but if an error does occur, then it could cause a `nil` dereference panic or cause the starting info to contain unexpected values.

Recommendations

Any error returned from fetching a value from the store should either be handled correctly or explicitly ignored if that is the intended behavior.

Remediation

This issue has been acknowledged by Initia Labs, and a fix was implemented in [PR #196](#).

4.7. Move coins are case-insensitive in cosmos

Target	x/move/types/denom.go		
Category	Coding Mistakes	Severity	Low
Likelihood	Medium	Impact	Low

Description

When performing operations on Move coins, the metadata address is fetched from the coins denom:

```
// Extract metadata address from a denom
func MetadataAddressFromDenom(denom string) (vmtypes.AccountAddress, error) {
    if strings.HasPrefix(denom, DenomTraceDenomPrefixMove) {
        addrBz, err := hex.DecodeString(strings.TrimPrefix(denom,
            DenomTraceDenomPrefixMove))
        if err != nil {
            return vmtypes.AccountAddress{}, err
        }

        return vmtypes.NewAccountAddressFromBytes(addrBz)
    }

    // non move coins are generated from 0x1.
    return NamedObjectAddress(vmtypes.StdAddress, denom), nil
}
```

The issue is that `hex.DecodeString` is case-insensitive, so a denom of `move/AAAA` and `move/aaaa` will both return the same metadata address. Inside the Move VM, the metadata addresses are always case-insensitive when parsed, but the default cosmos SDK behavior is that they are case-sensitive.

This could cause functions such as `Coins.Validate` to incorrectly validate the coins, even when there are duplicates:

```
// Validate checks that the Coins are sorted, have positive amount, with a
    valid and unique
// denomination (i.e no duplicates). Otherwise, it returns an error.
func (coins Coins) Validate() error {
```

Impact

Any function relying on the fact that a validated coins should contain no duplicates could have this invariant broken.

Recommendations

A check should be added to ensure that the denom values for Move coins are always lowercase.

Remediation

This issue has been acknowledged by Initia Labs, and a fix was implemented in [PR #196](#).

4.8. Coin amount not validated when funding and spending community pool

Target	x/distribution/keeper/msg_server.go		
Category	Coding Mistakes	Severity	Informational
Likelihood	N/A	Impact	Informational

Description

The handler for FundCommunityPool allows users to send coins to the community pool:

```
func (k msgServer) FundCommunityPool(ctx context.Context, msg
    *types.MsgFundCommunityPool) (*types.MsgFundCommunityPoolResponse, error) {
    defer telemetry.MeasureSince(time.Now(), "distribution", "msg",
        "fund-community-pool")

    depositor, err := k.authKeeper.AddressCodec().StringToBytes(msg.Depositor)
    if err != nil {
        return nil, err
    }
    if err := k.Keeper.FundCommunityPool(ctx, msg.Amount, depositor); err
    != nil {
        return nil, err
    }

    return &types.MsgFundCommunityPoolResponse{}, nil
}
```

The issue is that the `msg.Amount` is not validated to ensure that every value is positive and that there are no duplicate coins. The same issue exists in the `CommunityPoolSpend` handler.

Impact

If a negative value is used, the call to `MoveBankKeeper.SendCoin` will end up failing when trying to convert the amount to a `uint64`. If duplicate coins are used, the call to `SendCoin` will succeed, but luckily the call to `sdk.NewDecCoinsFromCoins` when saving the new community pool will end up failing. As code further down the line may expect the coins to already be validated, it would be better to ensure that the coins are validated before being used.

Recommendations

The `msg.Amount` for both `FundCommunityPool` and `CommunityPoolSpend` should be validated, and an appropriate error message should be returned.

Remediation

This issue has been acknowledged by Initia Labs, and a fix was implemented in [PR #190](#).

5. MiniEVM Findings

5.1. Query gas limit not enforced through bank module

Target	x/evm/keeper/context.go		
Category	Coding Mistakes	Severity	High
Likelihood	High	Impact	High

Description

When performing calls to the EVM, a `ContractQueryGasLimit` is used to limit the amount of gas that can be used to prevent users from DOSing a node with an infinite loop in an EVM contract:

```
// Call implements types.QueryServer.
func (qs *queryServerImpl) Call(ctx context.Context, req
    *types.QueryCallRequest) (res *types.QueryCallResponse, err error) {
    defer func() {
        if r := recover(); r != nil {
            err = errorsmod.Wrap(types.ErrEVMCallFailed, fmt.Sprintf("vm
panic: %v", r))
        }
    }()

    sdkCtx := sdk.UnwrapSDKContext(ctx)
    sdkCtx =
    sdkCtx.WithGasMeter(storetypes.NewGasMeter(qs.config.ContractQueryGasLimit))
```

The issue is that this limit is not enforced when `EVMStaticCall` is called directly, such as when the bank module checks the balance of a user:

```
func (k ERC20Keeper) balanceOf(ctx context.Context, addr, contractAddr
    common.Address) (math.Int, error) {
    inputBz, err := k.ERC20ABI.Pack("balanceOf", addr)
    if err != nil {
        return math.ZeroInt(), types.ErrFailedToPackABI.Wrap(err.Error())
    }

    retBz, err := k.EVMStaticCall(ctx, types.NullAddress, contractAddr,
    inputBz)
    if err != nil {
        return math.ZeroInt(), err
    }
}
```

```
res, err := k.ERC20ABI.Unpack("balanceOf", retBz)
if err != nil {
    return math.ZeroInt(), types.ErrFailedToUnpackABI.Wrap(err.Error())
}

balance, ok := res[0].(*big.Int)
if !ok {
    return math.ZeroInt(), types.ErrFailedToDecodeOutput
}

return math.NewIntFromBigInt(balance), nil
}
```

Impact

A malicious user can create an ERC-20 contract with an infinite loop in the `balanceOf` function, mint some coins to themselves, and then trigger a call to `balanceOf` with `minitiad` query bank balances to cause the node to enter into an infinite loop.

Recommendations

The `ContractQueryGasLimit` should always be enforced, or the default Cosmos SDK setting `query-gas-limit` should be used to limit the amount of gas that can be used for queries.

Remediation

This issue has been acknowledged by Initia Labs, and a fix was implemented in [PR #20](#).

5.2. Incorrect signer check for shorthand accounts

Target	x/evm/precompiles/cosmos/contract.go		
Category	Coding Mistakes	Severity	High
Likelihood	Low	Impact	Low

Description

When executing a cosmos message from the Move VM, the signers of the message are checked to ensure that they match the caller or that the shorthand account's original address matches the caller:

```
for _, signer := range signers {
    if bytes.Equal(caller.Address().Bytes(), signer) {
        continue
    }

    // if signer is different from the caller, check if the signer is a
    // shorthand account.
    // and then check shorthand account's original address is the same with the
    // caller.
    if len(signer) != common.AddressLength {
        signerAccount := e.ak.GetAccount(ctx, signer)
        if shorthandCallerAccount,
        ok := signerAccount.(types.ShorthandAccountI); ok {
            addr, err := shorthandCallerAccount.GetOriginalAddress(e.ac)
            if err != nil {
                return nil, ctx.GasMeter().GasConsumedToLimit(),
                types.ErrPrecompileFailed.Wrap(err.Error())
            }

            if bytes.Equal(addr.Bytes(), signer) {
                continue
            }
        }
    }
}
```

The issue is that a shorthand account's original address `addr` is compared against the signer instead of the caller, defeating the purpose of the check.

Luckily this branch cannot be reached, as shorthand accounts will always be stored with an address of length `common.AddressLength`, preventing the if block from being entered.

Impact

If a shorthand account was stored with an address length not equal to `common.AddressLength`, it would allow anyone to send cosmos messages on behalf of the account.

Recommendations

The shorthand account of the caller should be fetched, and its original address should be compared against the signer of the message.

Remediation

This issue has been acknowledged by Initia Labs, and a fix was implemented in [PR #16](#).

6. OPinit Findings

6.1. Validator set updates can skip current validators

Target	x/opchild/keeper/val_state_change.go		
Category	Coding Mistakes	Severity	High
Likelihood	Low	Impact	High

Description

When calculating the validator updates for the current block, the ApplyAndReturnValidatorSetUpdates function iterates through maxValidators to find any validators that have changed voting power.

```
func (k Keeper) ApplyAndReturnValidatorSetUpdates(ctx context.Context)
([]abci.ValidatorUpdate, error) {
    last, err := k.getLastValidatorsByAddr(ctx)
    if err != nil {
        return nil, err
    }

    updates := []abci.ValidatorUpdate{}
    maxValidators, err := k.MaxValidators(ctx)
    if err != nil {
        return nil, err
    }

    validators, err := k.GetValidators(ctx, maxValidators)
    if err != nil {
        return nil, err
    }

    for _, validator := range validators {
        valAddr,
        err := k.validatorAddressCodec.StringToBytes(validator.GetOperator())
        if err != nil {
            return nil, err
        }

        oldPower, found := last[validator.GetOperator()]
        newPower := validator.ConsensusPower()

        // zero power validator removed from validator set
        if newPower <= 0 {
```

```

        continue
    }

    if !found || oldPower != newPower {
        updates = append(updates, validator.ABCValidatorUpdate())

        if err := k.SetLastValidatorPower(ctx, valAddr, newPower); err
        != nil {
            return nil, err
        }
    }

    delete(last, validator.GetOperator())
}

noLongerBonded, err := sortNoLongerBonded(last, k.validatorAddressCodec)
if err != nil {
    return nil, err
}

```

Any validators that remain in the last map are considered to now be unbonded.

The issue is that GetValidators is not sorted, so there is a chance that a currently bonded validator will not be part of the maxValidators returned. This will cause them to be marked as no longer bonded, and they will try to be removed:

```

for _, valAddrBytes := range noLongerBonded {
    validator := k.mustGetValidator(ctx, sdk.ValAddress(valAddrBytes))
    if validator.ConsPower > 0 {
        return nil, errors.New("deleting validator cannot have positive
power")
    }

    valAddr,
    err := k.validatorAddressCodec.StringToBytes(validator.GetOperator())
    if err != nil {
        return nil, err
    }

    if err := k.RemoveValidator(ctx, valAddr); err != nil {
        return nil, err
    }
}

```

In this case, the check for `validator.ConsPower > 0` would fail, and an error would be returned.

Impact

Since `ApplyAndReturnValidatorSetUpdates` is called in the `EndBlocker`, if an error is returned, it will cause a consensus failure and halt the chain.

Recommendations

All the validators should be checked, or `GetValidators` should be sorted by consensus power before iterating through them.

Remediation

This issue has been acknowledged by Initia Labs, and a fix was implemented in [PR #71 ↗](#).

6.2. Challenger can increase the next output index

Target	x/ophost/keeper/msg_server.go		
Category	Coding Mistakes	Severity	Medium
Likelihood	Medium	Impact	Medium

Description

The DeleteOutput handler is used by the challenger to roll back nonfinalized proposals, starting at outputIndex up to the next output index, and then reset the next output index to be outputIndex.

```

nextOutputIndex, err := ms.GetNextOutputIndex(ctx, bridgeId)
if err != nil {
    return nil, err
}

// delete output proposals in [outputIndex, nextOutputIndex) range
for i := outputIndex; i < nextOutputIndex; i++ {
    if err := ms.DeleteOutputProposal(ctx, bridgeId, i); err != nil {
        return nil, err
    }
}

// rollback next output index to the deleted output index
if err := ms.NextOutputIndexes.Set(ctx, bridgeId, outputIndex); err != nil {
    return nil, err
}

```

The issue is that there is no check that the outputIndex is less than the nextOutputIndex. If a challenger submitted such an index, no outputs would be deleted, but the next output index would still be set. This would cause all future calls to ProposeOutput to fail as it tries to fetch the previous output using lastOutputProposal, err := ms.GetOutputProposal(ctx, bridgeId, outputIndex-1), which would not exist.

The challenger could also set the outputIndex to be the max value for a uint64, causing IncreaseNextOutputIndex to overflow back to zero the next time it is called.

Impact

If a challenger submits an outputIndex that is greater than the next output index, the ProposeOutput will always fail, preventing any further outputs from being proposed.

Recommendations

The `outputIndex` should be checked to ensure that it is less than the next output index.

Remediation

This issue has been acknowledged by Initia Labs, and a fix was implemented in [PR #75](#).

6.3. Missing token pair will crash the bridge executor

Target	src/lib/monitor/l2.ts		
Category	Coding Mistakes	Severity	Medium
Likelihood	Medium	Impact	Medium

Description

When the bridge executor is processing an `InitiateTokenWithdrawal` event from an L2, it tries to fetch the corresponding token pair from the L1:

```
export class L2Monitor extends Monitor {
  // [snip]
  private async handleInitiateTokenWithdrawalEvent(
    manager: EntityManager,
    data: { [key: string]: string }
  ): Promise<void> {
    const outputInfo = await this.helper.getLastOutputFromDB(
      manager,
      ExecutorOutputEntity
    )
    if (!outputInfo) return
    const pair = await config.l1lcd.ophost.tokenPairByL2Denom(
      this.bridgeId,
      data['denom']
    )
  }
}
```

The issue is that if the token pair does not exist, the call will fail and end up causing the bridge executor to stop. Since anyone is able to initiate a withdrawal with any denom, there is no guarantee that the token pair will exist.

Impact

A malicious user could crash the bridge executor by submitting a `MsgInitiateTokenWithdrawal` transaction for a denom that did not originate from the L1.

Recommendations

If a token pair does not exist on the L1, then it should be handled gracefully by the bridge executor.

The `MsgInitiateTokenWithdrawal` handler on the L2 could also check to ensure that the coin being withdrawn originated from the L1.

Remediation

This issue has been acknowledged by Initia Labs, and a fix was implemented in [PR #76](#). An additional fix was implemented in [PR #20](#).

6.4. Withdrawal hash clash using variable-length fields

Target	x/ophost/keeper/msg_server.go		
Category	Coding Mistakes	Severity	High
Likelihood	Medium	Impact	Medium

Description

When finalizing a token withdrawal, a hash is generated using the provided values to ensure the same withdrawal is not processed twice and that it can be used to generate the root hash:

```
// verify storage root can be generated from
// withdrawal proofs and withdrawal tx data.
{
    var withdrawalHash [32]byte
    {
        seed := []byte{}
        seed = binary.BigEndian.AppendUint64(seed, bridgeId)
        seed = binary.BigEndian.AppendUint64(seed, req.Sequence)
        seed = append(seed, sender...)
        seed = append(seed, receiver...)
        seed = append(seed, []byte(denom)...)
        seed = binary.BigEndian.AppendUint64(seed, amount.Uint64())

        withdrawalHash = sha3.Sum256(seed)
    }

    if ok, err := ms.HasProvenWithdrawal(ctx, bridgeId, withdrawalHash); err
    != nil {
        return nil, err
    } else if ok {
        return nil, types.ErrWithdrawalAlreadyFinalized
    }

    // should works with sorted merkle tree
    rootSeed := withdrawalHash
    proofs := req.WithdrawalProofs
    for _, proof := range proofs {
        switch bytes.Compare(rootSeed[:], proof) {
            case 0, 1: // equal or greater
                rootSeed = sha3.Sum256(append(proof, rootSeed[:]))
            case -1: // less
```



```
        rootSeed = sha3.Sum256(append(rootSeed[:], proof...))
    }
}
```

```
rootHash := rootSeed
if !bytes.Equal(req.StorageRoot, rootHash[:]) {
    return nil, types.ErrFailedToVerifyWithdrawal.Wrap(
```

The issue is that the receiver and denom are both variable lengths; it is possible to generate the same hash by shifting bytes from the end of one to the start of another or vice versa. This could allow someone to effectively burn coins being held by the bridge under certain circumstances.

Impact

With the way Initia is currently set up, it would require an L1 denom that can be represented in hex, as the only denoms that an attacker can create are in the format `move/[hex_address]`. However, if another chain were to use the OPInit stack or if a new L1 token were added to the L1, it could open this up for attack.

Recommendations

A separator should be added between the variable-length fields to ensure that they cannot be shifted to generate the same hash.

Remediation

This issue has been acknowledged by Initia Labs, and a fix was implemented in [PR #65](#).

6.5. Merkle tree lacks second preimage resistance

Target	x/ophost/keeper/msg_server.go		
Category	Coding Mistakes	Severity	Informational
Likelihood	Low	Impact	Informational

Description

When performing a token withdrawal, a withdrawalHash is first calculated with the following:

```
var withdrawalHash [32]byte
{
    seed := []byte{}
    seed = binary.BigEndian.AppendUint64(seed, bridgeId)
    seed = binary.BigEndian.AppendUint64(seed, req.Sequence)
    seed = append(seed, sender...)
    seed = append(seed, receiver...)
    seed = append(seed, []byte(denom)...)
    seed = binary.BigEndian.AppendUint64(seed, amount.Uint64())

    withdrawalHash = sha3.Sum256(seed)
}
```

The supplied proof is then used to verify it was included in the Merkle tree:

```
// should works with sorted merkle tree
rootSeed := withdrawalHash
proofs := req.WithdrawalProofs
for _, proof := range proofs {
    switch bytes.Compare(rootSeed[:], proof) {
    case 0, 1: // equal or greater
        rootSeed = sha3.Sum256(append(proof, rootSeed[:]))
    case -1: // less
        rootSeed = sha3.Sum256(append(rootSeed[:], proof))
    }
}
```

The issue is that there is no distinction between leaf nodes and intermediate nodes, so it is technically possible to create a withdrawal hash that matches an intermediate node by appending a proof hash to a leaf node and then supplying a proof that matches the intermediate node:

```
func calculateRoot(bridgeId, sequence uint64, sender, receiver []byte, denom
string, amount uint64, withdrawalProofs [][]byte) [32]byte {
    var withdrawalHash [32]byte
    {
        seed := []byte{}
        seed = binary.BigEndian.AppendUint64(seed, bridgeId)
        seed = binary.BigEndian.AppendUint64(seed, sequence)
        seed = append(seed, sender...)
        seed = append(seed, receiver...)
        seed = append(seed, []byte(denom)...)
        seed = binary.BigEndian.AppendUint64(seed, amount)

        withdrawalHash = sha3.Sum256(seed)
    }

    rootSeed := withdrawalHash
    proofs := withdrawalProofs
    for _, proof := range proofs {
        switch bytes.Compare(rootSeed[:], proof) {
            case 0, 1: // equal or greater
                rootSeed = sha3.Sum256(append(proof, rootSeed[:]))
            case -1: // less
                rootSeed = sha3.Sum256(append(rootSeed[:], proof...))
        }
    }

    return rootSeed
}

func Test_FinalizeTokenWithdrawalPreimage4(t *testing.T) {
    storageRoot := decodeHex(t,
        "326ca35f4738f837ad9f335349fc71bdecf4c4ed3485fff1763d3bab55efc88a")
    withdrawalProofs := [][]byte{
        decodeHex(t,
            "32e1a72a7c215563f9426bfe267b6fa22ba49b1fba7162d80094dc2f2b6c5a3a"),
        decodeHex(t,
            "627dc2af9ee001b0e119100599dc3923ccdf2c53f06d89f40400edb1e7907e1"),
        decodeHex(t,
            "bafac86e9ebc05a07701c151846c6de7bca68cd315f7a82fffe05fc4301ac47e"),
    }

    rootHash := calculateRoot(
        uint64(1),
        uint64(4),
        decodeHex(t, "0000000000000000000000000000000000000000000000000000000000000004"),
        decodeHex(t, "0000000000000000000000000000000000000000000000000000000000000001"),
```

```
        "11denom",
        uint64(3_000_000),
        withdrawalProofs[:],
    )
    require.Equal(t, storageRoot, rootHash[:])

    rootHash = calculateRoot(
        uint64(0x32e1a72a7c215563),
        uint64(0xf9426bfe267b6fa2),
        decodeHex(t, "2ba49b1fba7162d80094dc2f2b6c5a3a4402e743"),
        decodeHex(t, "c5c72c0c727da13466d4fcd6dcd88d719f8b00b7"),
        "",
        uint64(0xd13cb6587a1ae64f),
        withdrawalProofs[1:],
    )
    require.Equal(t, storageRoot, rootHash[:])
}
```

Impact

In practice, this issue is unlikely to be exploitable, as the bridge ID will not exist and will not have a balance for the supplied denom and amount. However, it is a good practice to ensure that the Merkle tree has second preimage resistance.

Recommendations

A distinction should be made between leaf nodes and intermediate nodes in the Merkle tree. One common approach is to prepend 0x00 to leaf nodes and 0x01 to intermediate nodes, as mentioned in the [Certificate Transparency specification](#).

Remediation

This issue has been acknowledged by Initia Labs, and a fix was implemented in [PR #83](#). This issue has been acknowledged by Initia Labs, and a fix was implemented in [PR #39](#).

7. General Discussion

The purpose of this section is to document miscellaneous observations that we made during the assessment. These discussion notes are not necessarily security related and do not convey that we are suggesting a code change.

7.1. Improve AMM tests

We strongly recommend creating a more thorough testing suite for `dex.move`, `stableswap.move`, and `miniswap.move` to ensure correctness. Subtle bugs hiding in code are less likely to exist when the test suite is comprehensive. Ensure the test suite is comprehensive — covering edge cases and extreme values — and achieves as close to 100% code coverage as possible. Additionally, consider implementing fuzz testing and formal verification where applicable using the Move prover.

7.2. Bridge executor minting power

The bridge executor account is responsible for finalizing token deposits from the L1 and minting them on the L2 using the `MsgFinalizeTokenDeposit` message. This power allows the bridge executor to mint arbitrary coins on the L2 regardless of what has happened on the L1, so it is vital that the bridge executor account is secure and only accessible to trusted parties.

7.3. Charge before using values

As noted in the code comments, in several places in `table.rs`, consider charging the gas cost before using the values for potentially expensive operations.

```
table.rs: // TODO(Gas): Figure out a way to charge this earlier.
table.rs- context.charge(key_cost + value_cost)?;
--
table.rs: // TODO(Gas): Figure out a way to charge this earlier.
table.rs- context.charge(key_cost + value_cost)?;
--
table.rs: // TODO(Gas): Figure out a way to charge this earlier.
table.rs- context.charge(key_cost + value_cost)?;
--
table.rs: // TODO(Gas): Figure out a way to charge this earlier.
table.rs- context.charge(key_cost + value_cost)?;
```

7.4. Empty `allowed_publishers` array allows every address to publish

The following behavior appears to be intended. However, it is unintuitive behavior and possibly dangerous.

We wanted to note that if `allowed_publishers` is ever an empty array in `code.move`, the following code will allow any address to publish a module:

```
fun assert_allowed(allowed_publishers: &vector<address>, addr: address) {
    assert!(
        vector::is_empty(allowed_publishers)
        || vector::contains(allowed_publishers, &addr),
        error::invalid_argument(EINVALID_ALLOWED_PUBLISHERS),
    )
}
```

7.5. ERC20Keeper with custom contracts

The EVM module currently allows anyone to create and register an ERC20 contract with a custom implementation that will then be used by the Cosmos bank module when interacting with that particular coin. This will require chains to be especially careful of any operations on coins performed in critical paths (such as in a `BeginBlocker` or `EndBlocker`) and the custom ERC-20 logic is able to fail at will or enter into an infinite loop.

7.6. Suggestions for additional security checks

This section contains some suggestions for additional security checks and mitigations that could be considered for implementation.

`MoveValue::convert_option`

The `MoveValue::convert_option` could assert that the vector containing the option value has length zero or one.

`readULEB128` maximum input length

The `readULEB128` function (located in `x/move/types/connector.go`) does not limit the maximum length of the input.

7.7. Table handle overflow leads to abort

Note that it is feasible for the `table_len` to overflow if enough table iterators are created over many transactions, leading to an abort.

```
let table_len = table_data.new_tables.len() as u32; // cast usize to u32 to
ensure same length
Digest::update(&mut digest, UID_PREFIX);
Digest::update(&mut digest, table_context.session_id);
Digest::update(&mut digest, table_len.to_be_bytes());
let bytes = digest.finalize().to_vec();
let handle = AccountAddress::from_bytes(&bytes[0..AccountAddress::LENGTH])
    .map_err(|_| partial_extension_error("Unable to create table handle"))?;
let key_type = context.type_to_type_tag(&ty_args[0])?;
let value_type = context.type_to_type_tag(&ty_args[1])?;
assert!(table_data
    .new_tables
    .insert(TableHandle(handle), TableInfo::new(key_type, value_type))
    .is_none());
```

However, this has no security impact as the abort would be handled properly and simply revert the transaction.

8. Threat Model

This provides a full threat model description for various functions. As time permitted, we analyzed each function in the components and created a written threat model for some critical functions. A threat model documents a given function's externally controllable inputs and how an attacker could leverage each input to cause harm.

Not all functions in the audit scope may have been modeled. The absence of a threat model in this section does not necessarily suggest that a function is safe.

8.1. Module: Initia - x/bank

Message: MsgSend

The MsgSend message allows an account to send coins to another account. The amount is checked to ensure that it is valid, that all values are positive, and that the receiver is not blocked. The Move-BankKeeper is then used to send the coins. This differs from the standard Cosmos SDK in that each coin is iterated over and the Move function `coin::transfer` is called in order to handle the transfer.

Message: MsgSetDenomMetadata

The MsgSetDenomMetadata handler allows an authorized account (defaults to x/gov) to set the denom metadata for a coin. The metadata is checked to ensure that it is valid before being updated in the store.

8.2. Module: Initia - x/distribution

Message: MsgCommunityPoolSpend

This handler allows an authorized account (defaults to x/gov) to distribute coins from the fee pool, updating the community pool. The recipient is checked to ensure it is not blocked, but the amount is not validated to ensure that it is positive (see Finding [4.8.7](#)).

Message: MsgDepositValidatorRewardsPool

This handler is not implemented and will throw an error if called.

Message: MsgFundCommunityPool

This handler allows anyone to contribute coins to the community pool. The Amount is not checked to ensure that it is valid, and although SendCoins will end up failing if a negative value is used, it will not fail if the coins contain duplicate or out-of-order denoms. This allows the CommunityPool to be stored with an invalid DecCoins (see Finding [4.8.7](#)).

Message: MsgSetWithdrawAddress

This handler allows a delegator to set the withdrawal address that will receive the rewards. The delegator and withdraw address are both checked to ensure that they are valid.

Message: MsgWithdrawDelegatorReward

This handler allows a delegator to withdraw rewards from a delegation. This will calculate the reward owed to the delegator, send the reward, and then update the starting info for the delegation.

Message: MsgWithdrawValidatorCommission

This handler allows a validator to withdraw all of their accumulated commission. The commission is calculated, subtracted from the outstanding rewards, and then sent to the validator. Any truncated decimals are left in the accumulated balance for the validator.

8.3. Module: Initia - x/ibc-hooks**Message: MsgUpdateACL**

This handler allows the authority account (the x/gov module) to update the list of addresses that are allowed to use the IBC hooks when transferring tokens.

Message: MsgUpdateParams

This handler allows the authority account (the x/gov module) to update the module parameters, which include the default ACL value for when an address has not been specifically set.

Message: OnAcknowledgementPacketOverride

This handler checks to see if the incoming packet is an ICS-20 or ICS-721 packet and to call the appropriate onAck handler if so. Both packets have their memo field parsed and validated to check whether they specify an async callback Move module which needs to be executed.

For ICS-20 and ICS-721 packets, the module address is checked against the ACLs to see if it is allowed to execute IBC hooks, then the `ibc_ack` function is called on the specified module.

Message: OnRecvPacketOverride

This handler checks to see if the incoming packet is an ICS-20 or ICS-721 packet and to call the appropriate onRecv handler if so. Both packets have their memo field parsed and validated to check whether they have a Move message that needs to be executed.

For ICS-20 and ICS-721 packets, the module address is checked against the ACLs to see if it is allowed to execute IBC hooks, then the receiver is checked to ensure that it matches the function identifier being called. An intermediate account is then generated to hold the tokens, which is then used as the sender of the Move call when executing the hook.

Message: OnTimeoutPacketOverride

This handler checks to see if the incoming packet is an ICS-20 or ICS-721 packet and to call the appropriate handler if so. Both packets have their memo field parsed and validated to check whether they specify an async callback Move module needs to be executed.

For ICS-20 and ICS-721 packets, the module address is checked against the ACLs to see if it is allowed to execute IBC hooks, then the `ibc_timeout` function is called on the specified module.

8.4. Module: Initia - x/ibc

Message: MsgSetPermissionedRelayer

This handler allows the x/gov module to update the permissioned relayer for an IBC channel. The port and channel ID are checked to ensure that they are valid, along with the authority and relayer addresses. This can be used to change the relayer of an in-use IBC channel, unlike creating a bridge, which requires the IBC channel to have a sequence number of one.

Message: MsgTransfer

This handler allows a user to transfer an NFT token to another chain using the ICS-721 protocol. This will first determine if the NFT originated from the current chain by examining the token's class ID, and if so, it will transfer the token into an escrow account before sending the IBC packet. Otherwise, it will burn the token, as the original is located on the receiving chain.

Message: MsgUpdateParams

This handler allows the authority account (the x/gov module) to set the parameters for the module, which include toggling the `SendEnabled` and `ReceiveEnabled` flags.

Message: OnRecvPacket

This handler implements the receiving side of the ICS-721 protocol spec, allowing connected IBC chains to transfer NFTs. The `NonFungibleTokenPacketData` is first validated to ensure that the sender, receiver, and class ID are not blank, that the token ID's array is not empty, and that receiving is enabled. Then, if the receiving chain is the source of the original NFT, the token is unescrowed; otherwise, a new prefix is generated using the IBC port and channel, which is then used to mint new tokens. The mint happens inside the Move VM using the `initia_nft::mint` function.

8.5. Module: Initia - x/move

Struct: MoveBankKeeper

This keeper replaces the default Cosmos SDK bank keeper with a version that delegates all operations to the Move VM. For example, SendCoins will end up calling the Move function `0x1::coin::transfer` to transfer the coins from one account to another. Since anyone can create a coin, care must be taken when using this keeper as the owner of a coin is able to mint/burn coins as well as freeze/unfreeze accounts. See Finding [4.1](#), [↗](#) for more details.

Message: MsgDelist

This handler allows the authority account (the x/gov module) to delist an LP token from the staking bond denoms list, preventing anyone from using the token as a delegation.

Message: MsgExecute

This handler allows the sender to execute an entry function on a Move module. Once the Move VM returns, any events from the call are emitted, any new accounts are created, the contract shared revenue is distributed based on gas usage, any staking deltas are applied, and then finally any queued Cosmos messages from the Move VM are executed.

Message: MsgExecuteJSON

This is the same as the `MsgExecute` handler described above, but the arguments to the Move function are a stringified JSON object.

Message: MsgGovExecute

This handler allows the authority account (the x/gov module) to execute a function on a Move module as the specified sender. Once the authority has been checked, the same flow as the `MsgExecute` handler described above is followed.

Message: MsgGovExecuteJSON

This is the same as the `MsgGovExecute` handler described above, but the arguments to the Move function are a stringified JSON object.

Message: MsgGovPublish

This handler allows the authority account (the x/gov module) to publish a module to the Move VM as a specified sender. The module must be less than 1 MB and have a valid upgrade policy. The module

address must be the same address as the sender.

Message: MsgGovScript

This handler allows the authority account (the x/gov module) to execute a Move script as the specified sender. Once the authority has been checked, the same flow as the MsgScript handler described below is followed.

Message: MsgGovScriptJSON

This is the same as the MsgGovScript handler described above, but the arguments to the Move function are a stringified JSON object.

Message: MsgPublish

This handler allows a user to publish a module to the Move VM. The module must be less than 1 MB and have a valid upgrade policy. The module address must be the same address as the sender.

Message: MsgScript

This handler allows the sender to execute a Move script. Once the Move VM returns, any events from the call are emitted, any new accounts are created, the contract shared revenue is distributed based on gas usage, any staking deltas are applied, and then finally any queued Cosmos messages from the Move VM are executed.

Message: MsgScriptJSON

This is the same as the MsgScript handler described above, but the arguments to the Move function are a stringified JSON object.

Message: MsgUpdateParams

This handler allows the authority account (the x/gov module) to update the module parameters, which include the base denom, the base minimum gas price, the contract shared revenue ratio, and the list of allowed publishers. Each of these fields is first validated, and all must be supplied when updating the parameters.

Message: MsgWhitelist

This handler allows the authority account (the x/gov module) to whitelist an LP token as a bond denom, allowing it to be used for staking. The LP token must have the base denom as one of its pairs,

and the base weight must be larger than the quote weight.

8.6. Module: Initia - x/mstaking

Message: MsgBeginReDelegate

This handler allows a delegator to switch validators, unbonding coins from one validator and bonding them to another. The delegator, source validator, and destination validator addresses are all validated, and the amount of coins is checked to ensure that they are all positive. The redelegation then occurs, first unbonding the coins from the source validator and then delegating them to the destination validator. If the source validator is bonded, a redelegation entry is added into the queue with the corresponding completion time.

Message: MsgCancelUnbondingDelegation

This handler allows a user to cancel an unbonding delegation and redelegate back to the validator. The validator and delegator addresses are validated, the amount is checked to ensure that they are all positive and valid bond denoms, and the creating height is checked to ensure it is greater than zero. The validator is then checked to ensure that it is not jailed and has a valid exchange rate. The appropriate unbonding delegation entry is then fetched and has the appropriate amount of coins removed, before being saved or deleted if it is empty. The coins are delegated back to the validator without subtracting them from the sender's account.

Message: MsgCreateValidator

This handler is used to create a new validator. The address, public key, and description are all checked to ensure that they are not empty and valid, the commission is validated to ensure that the max rate is not negative and not greater than one, and the rate and max change rate are checked to ensure they are not negative and less than the max rate. The rate is also checked to ensure that it is greater than the minimum validator commission rate.

The public key is checked to ensure that it has not already been used by another validator and that the validator address does not already exist. The delegation amount is then checked to ensure they are all valid bond denoms, and finally the validator is created and the coins are delegated to it.

Message: MsgDelegate

This handler allows an account to delegate coins to a validator. The validator and delegator addresses are validated, the amount is checked to ensure that they are all positive and valid bond denoms, and then the coins are delegated to the validator from the delegator's account.

Message: MsgEditValidator

This handler allows a validator to update its description and commission rate. The validator address is validated, the description is checked to ensure that it is not empty, and the commission rate is checked to ensure that the max rate is not negative and not greater than one. The description and commission rate are then updated in the store.

Message: MsgUndelegate

This handler allows a delegator to undelegate their coins from a validator. The delegator and validator addresses are validated, and the amount is checked to ensure that they are positive and that the delegator has delegated enough coins to cover the undelegation. The shares are then undelegated from the validator and an unbonding entry is added to the queue.

Message: MsgUpdateParams

This handler allows the authority account (the x/gov module) to update the module parameters, which include the unbonding time, the maximum number of validators, the maximum number of entries for the unbonding/redelegation query, the number of historical entries, the bond denoms that can be used for staking, and the minimum voting power.

8.7. Module: Initia - x/reward**Message: MsgUpdateParams**

This handler allows the authority account (the x/gov module) to update the module parameters, which include the denom of the rewards, the dilution period, the release rate, and the release-enabled flag.

8.8. Module: MiniEVM - x/bank**Message: EVMSendKeeper**

This keeper replaces the default Cosmos SDK bank keeper with a version that delegates all operations to the EVM. For example, SendCoins will end up calling transfer on the ERC20 contract to transfer the coins from one account to another. Since anyone can create a token, care must be taken when using this keeper as the ERC20 can contain a custom implementation that could unexpectedly error or enter an infinite loop.

Message: MsgMultiSend

This handler is not implemented and will throw an error.

Message: MsgSend

This handler is used to transfer coins from one account to another. It validates that the `from` and `to` addresses are valid and that the coins are positive, valid, and have the `SendEnabled` flag set. The `to` account is also checked to ensure that it is not blocked. The `SendCoins` function is then called, which will end up performing an EVM call using the `transfer` function of the ERC-20 token representing the coin.

Message: MsgSetDenomMetadata

The `MsgSetDenomMetadata` handler allows an authorized account (defaults to `x/gov`) to set the `denom` metadata for a coin. The metadata is checked to ensure that it is valid before being updated in the store.

Message: MsgSetSendEnabled

The `MsgSetSendEnabled` handler allows an authorized account (defaults to `x/gov`) to toggle whether a coin has the `SendEnabled` flag set or not or if it should use the global default.

8.9. Module: MiniEVM - x/evm**Message: MsgCall**

The `MsgCall` handler allows a user to call a method on an EVM contract. The specified contract and sender accounts are checked, and the input is checked to ensure that it is a valid hex string. If the caller's account is longer than 20 bytes, a shorthand account is created and used as the caller of the EVM function. Once the call to the EVM returns, any queued-up Cosmos messages sent via the EVM precompile are dispatched.

Message: MsgCreate

This handler allows a user to create a new contract in the EVM. The sender is checked to ensure that it is valid, and the code is checked to ensure that it is a hex string. If the `AllowedPublishers` parameter has been set, it is checked to ensure that the sender is in the list. The contract is then deployed and the address returned.

Message: MsgCreate2

This handler is the same as `MsgCreate`, described above, except that a salt can be provided and the contract created using the `CREATE2` opcode, allowing the contract to be created deterministically at a specific address.

Message: MsgUpdateParams

This handler allows the authority account (the x/gov module) to update the module parameters, including any additional EIPs to be used by the EVM and the list of allowed publishers.

8.10. Module: OPinit - x/opchild**Message: MsgAddValidator**

This handler allows the authority account (defaults to the x/opchild module) to add a validator to the designated validator set. The validator address and public key are checked to ensure that they do not already exist, then the validator is created with a consensus power of 1.

Message: MsgExecuteMessages

This handler allows the admin account (set via the module parameters) to execute MsgProposal messages as the authority account (defaults to the x/opchild module). Each message has its ValidateBasic method called if it exists, before passing the message to the appropriate handler using a cached context. If all the messages are handled correctly, the cached context is written to the store.

Message: MsgFinalizeTokenDeposit

The MsgFinalizeTokenDeposit handler allows the bridge executor to mint coins when a user sends a InitiateTokenDeposit message on the L1. The sequence number is checked to ensure that it has not been processed before, then the coins are minted and the sequence number is recorded.

Message: MsgInitiateTokenWithdrawal

The MsgInitiateTokenWithdrawal handler allows a user to transfer coins back to the L1 originated from there. The amount is checked to ensure that it is positive and valid, the next L2 sequence is increased, and the coins are burned. The initiate_token_withdrawal event is then emitted, which will be processed by the bridge executor.

Message: MsgRemoveValidator

This handler allows the authority account (defaults to the x/opchild module) to remove a validator from the designated validator set. The validator address is checked to ensure that it is valid and exists, then the validator's consensus power is set to 0 so that it is removed when the next EndBlocker is run.

Message: MsgSetBridgeInfo

The `MsgSetBridgeInfo` handler allows the bridge executor to update the current bridge config. The bridge ID, address, L1 chain ID, and L1 client ID cannot be changed, but the `BridgeConfig` can be updated.

Message: MsgSpendFeePool

This handler allows the authority account (defaults to the `x/opchild` module) to send any collected fees from the fee module to a recipient address. The recipient and authority addresses are checked to ensure that they are valid, and the amount is checked to ensure that it is positive and contains no duplicates. The coins are then transferred from the fee module to the recipient address.

Message: MsgUpdateOracle

The `MsgUpdateOracle` handler allows the bridge executor to apply an update to the L2 oracle handler for a specific height. The sender must be a valid address, and the height cannot be zero. The message data is decoded into a `slinkyaggregator.Vote` before being aggregated and written using `l2slinky.WritePrices`.

Message: MsgUpdateParams

This handler allows the authority account (defaults to the `x/opchild` module) to update the module params, which include the maximum number of validators, the number of historical entries to persist, the minimum gas prices, the bridge executor account, the admin account, and a list of addresses that do not need to pay fees.

8.11. Module: OPinit - x/ophost**Message: MsgCreateBridge**

The `MsgCreateBridge` handler allows anyone to register a new bridge with the L1 to allow an L2 to communicate. The bridge config must contain a valid challenger and proposer address, a valid batch-info chain ID and submitter, and have a finalization period, submission interval, and submission start time that are not zero. If a registration fee has been set by the L1, then that must be paid by the bridge creator. A new bridge config is then saved to the store and a bridge account is created. Finally, the `BridgeCreated` hook is called.

Message: MsgDeleteOutput

The `MsgDeleteOutput` handler allows the challenger account for a bridge to delete unfinalized L2 output proposals when it detects that they are invalid. The bridge ID and output index are checked to ensure that they are not zero, and the signer is checked to ensure that it is the challenger for

the bridge ID. Then, all the proposals from the specified `outputIndex` to the next output index are deleted, unless they are already finalized, in which case the transaction will fail. The next output index is then updated to the `outputIndex`. There is no check that the `outputIndex` supplied is less than the next output index; see Finding [6.2](#), [↗](#) for more information.

Message: `MsgFinalizeTokenWithdrawal`

The `MsgFinalizeTokenWithdrawal` handler allows users to withdraw coins that have been returned to the L1 using the `InitiateTokenWithdrawal` message on the L2, once the proposed output index has been finalized. The sender and receiver must be valid accounts; the amount must be valid and positive; the sequence, bridge ID, and output index cannot be zero; and the withdrawal proofs, version, state root, storage root, and last block hash must all be 32 bytes.

The output index for the bridge ID is checked to ensure that it is finalized, and the request's version, state root, storage root, and last block hash are used to generate the output root, which must match the finalized output proposal root.

Then, the withdrawal hash is generated using the bridge ID, sequencer number, sender, receiver, coin denom, and coin amount and checked to ensure the hash has not been claimed before. The provided withdrawal proofs are then used to ensure that the withdrawal hash was part of the Merkle tree by checking the calculated root hash matches the requested storage root.

If all the checks pass, the withdrawal hash is recorded as claimed and the coins are sent from the bridge escrow account to the receiver account.

Message: `MsgInitiateTokenDeposit`

The `MsgInitiateTokenDeposit` handler allows a user to transfer coins from the L2 to the L1, by moving the coins into the bridge escrow account and emitting an `initiate_token_deposit` event, which will be picked up by the bridge executor. The sender and receiver accounts are validated, the amount is checked to ensure it is valid and positive, and the bridge ID is checked to ensure it is not zero.

The L1 sequence for the bridge is increased by one, the coins are sent to the bridge escrow account, a token pair is created if it does not exist to map the L1 and L2 coins, and then the `initiate_token_deposit` event is emitted.

Message: `MsgProposeOutput`

The `MsgProposeOutput` handler is used by the bridge proposer to submit a new L2 block proposal. The proposer address is checked to ensure it is valid, the bridge ID is checked to ensure it is not zero, and the output root is checked to ensure that it is 32 bytes. The signer is then checked to ensure that it is the proposer for the bridge ID. If the proposal is not the first, the previous proposal is fetched to check that the new L2 block number is greater than the previous one. The new proposal is then saved to the store for the bridge ID.

Message: MsgRecordBatch

The `MsgRecordBatch` handler is a no-op that is only used for indexing the transaction, which can later be used to fetch the transaction data for performing a rollback of an L2.

Message: MsgUpdateBatchInfo

The `MsgUpdateBatchInfo` is used to update the batch info for a bridge. The authority address must be valid, and the bridge ID, chain, and submitter cannot be empty. Only the x/gov module account or the current bridge proposer can update the batch info. The `BridgeBatchInfoUpdated` hook is called, then the batch info for the bridge is updated in the store.

Message: MsgUpdateChallenger

The `MsgUpdateChallenger` handler is used to update the challenger for a bridge. The authority and new challenger address must be valid, and the bridge ID cannot be zero. Only the x/gov module account or the current bridge challenger can update the challenger. The `BridgeChallengerUpdated` hook is called, then the challenger for the bridge is updated in the store.

Message: MsgUpdateMetadata

The `MsgUpdateMetadata` handler is used to update the metadata for a bridge. The authority must be valid, the metadata must be less than `MaxMetadataLength`, and the bridge ID cannot be zero. Only the x/gov module account or the current bridge proposer can update the metadata. The `BridgeMetadataUpdated` hook is called, then the metadata for the bridge is updated in the store.

Message: MsgUpdateParams

This handler allows the authority account (defaults to the x/gov module) to update the module params, which include the registration fee required to be paid when creating a bridge. The fee is validated to ensure that they are valid denoms and are positive.

Message: MsgUpdateProposer

The `MsgUpdateProposer` handler is used to update the proposer for a bridge. The authority and new proposer address must be valid, and the bridge ID cannot be zero. Only the x/gov module account or the current bridge proposer can update the proposer. The `BridgeProposerUpdated` hook is called, then the proposer for the bridge is updated in the store.

8.12. Module block.move

Function: get_block_info

This unpermissioned function can be used to get the current block height and time stamp.

8.13. Module table.move

Function: new

This function can be used to create a new table. Note that this is the only function that allows to create a table object. Therefore, other generic functions in this module do not normally require to constrain the type of the table key/values, as table objects are guaranteed to have been created using types that have, respectively, copy + drop and store.

Inputs

- Type $K_{\langle \text{copy} + \text{drop} \rangle}$
 - **Validation:** No additional validation required besides ability constraints.
 - **Impact:** Type of the keys used to address elements of the table.
- Type $V_{\langle \text{store} \rangle}$
 - **Validation:** No additional validation required besides ability constraints.
 - **Impact:** Type of the values stored in the table.

Function: destroy_empty

This function can be used to destroy an empty table.

Inputs

- Type $K_{\langle \text{copy} + \text{drop} \rangle}$
 - **Validation:** None required.
 - **Impact:** Type of the table keys.
- Type V
 - **Validation:** None required.
 - **Impact:** Type of the table values.
- table: $\text{Table}\langle K, V \rangle$
 - **Validation:** The table is checked to be empty.
 - **Impact:** Table to be deleted.

Function: add

This function can be used to add an element to a table. The function reverts if an element is already associated to the given key.

Inputs

- Type `K<copy + drop>`
 - **Validation:** None required.
 - **Impact:** Type of the table keys.
- Type `V`
 - **Validation:** None required.
 - **Impact:** Type of the table values.
- `table: &Table<K, V>`
 - **Validation:** None required.
 - **Impact:** Table to which an element is to be added.
- `key: K`
 - **Validation:** Table must not contain an entry at the same key.
 - **Impact:** Key at which the new element is associated.
- `val: V`
 - **Validation:** None required.
 - **Impact:** Element to add to the table.

Functions: borrow, borrow_mut

These functions can be used to get a reference (immutable or mutable) to an element of the table. The function reverts if there is no element associated with the given key.

Inputs

- Type `K<copy + drop>`
 - **Validation:** None required.
 - **Impact:** Type of the table keys.
- Type `V`
 - **Validation:** None required.
 - **Impact:** Type of the table values.
- `table: &Table<K, V>`
 - **Validation:** None required.
 - **Impact:** Table to borrow from.
- `key: K`
 - **Validation:** An element associated with this key must exist.
 - **Impact:** Key identifying which element to borrow.

Functions: `borrow_with_default`, `borrow_mut_with_default`

These functions can be used to get a reference (immutable or mutable) to an element of the given table. Unlike their `borrow`/`borrow_mut` counterpart, they allow to specify a default value to be used if the table does not contain an element associated with the given key.

Inputs

- Type `K<copy + drop>`
 - **Validation:** None required.
 - **Impact:** Type of the table keys.
- Type `V`
 - **Validation:** None required.
 - **Impact:** Type of the table values.
- `table: &Table<K, V>`
 - **Validation:** None required.
 - **Impact:** Table to borrow from.
- `key: K`
 - **Validation:** An element associated with this key must exist.
 - **Impact:** Key identifying which element to borrow.
- `default: &V`
 - **Validation:** None required.
 - **Impact:** Default value used if no element is associated with the given key.

Function: `length`

This function can be used to get the length of a table.

Inputs

- Type `K<copy + drop>`
 - **Validation:** None required.
 - **Impact:** Type of the table keys.
- Type `V`
 - **Validation:** None required.
 - **Impact:** Type of the table values.
- `table: &Table<K, V>`
 - **Validation:** None required.
 - **Impact:** Table to operate on.

Function: empty

This function can be used to determine whether a table is empty or not.

Inputs

- Type `K<copy + drop>`
 - **Validation:** None required.
 - **Impact:** Type of the table keys.
- Type `V`
 - **Validation:** None required.
 - **Impact:** Type of the table values.
- `table: &Table<K, V>`
 - **Validation:** None required.
 - **Impact:** Table to operate on.

Function: upsert

This function can be used to perform an upsert operation on a table, inserting an element into the table and discarding the existing value associated with the given key if such a value already exists.

Inputs

- Type `K<copy + drop>`
 - **Validation:** None required.
 - **Impact:** Type of the table keys.
- Type `V<drop>`
 - **Validation:** Importantly, this function ensures that the value has drop, preventing an element without the ability from being silently discarded.
 - **Impact:** Type of the table values.
- `table: &Table<K, V>`
 - **Validation:** None required.
 - **Impact:** Table to operate on.
- `key: K`
 - **Validation:** None required.
 - **Impact:** Key that identifies the table position where the value is inserted (and the old value removed when applicable).
- `value: V`
 - **Validation:** None required.
 - **Impact:** Value to be inserted into the table.

Function: remove

This function can be used to remove an element from a table.

Inputs

- Type K <copy + drop>
 - **Validation:** None required.
 - **Impact:** Type of the table keys.
- Type V
 - **Validation:** None required.
 - **Impact:** Type of the table values.
- table: $\&Table<K, V>$
 - **Validation:** None required.
 - **Impact:** Table to operate on.
- key: K
 - **Validation:** None required.
 - **Impact:** Identifies the element to be removed.

Function: contains

This function can be used to check whether a table contains an element associated with a given key.

Inputs

- Type K <copy + drop>
 - **Validation:** None required.
 - **Impact:** Type of the table keys.
- Type V
 - **Validation:** None required.
 - **Impact:** Type of the table values.
- table: $\&Table<K, V>$
 - **Validation:** None required.
 - **Impact:** Table to operate on.
- key: K
 - **Validation:** None required.
 - **Impact:** Key value to test.

Functions: `iter`, `iter_mut`

These functions can be used to create an object that allows to iterate on the keys and values contained in the table, in a mutable and immutable fashion. Note that these functions were involved in Finding [3.1.7](#), which allowed to bypass critical security invariants, including the reference safety verifier. They were changed to remediate the vulnerability.

Inputs

- Type `K<copy + drop>`
 - **Validation:** None required.
 - **Impact:** Type of the table keys.
- Type `V`
 - **Validation:** None required.
 - **Impact:** Type of the table values.
- `table: &Table<K, V>`
 - **Validation:** None required.
 - **Impact:** Table to operate on.
- `start: Option<K>`
 - **Validation:** None required.
 - **Impact:** Start point for the iterator. The start is exclusive.
- `end: Option<K>`
 - **Validation:** None required.
 - **Impact:** End point for the iterator. The end point is inclusive.
- `order: u8`
 - **Validation:** Must be either 1 or 2.
 - **Impact:** Determines whether the iteration has to occur in ascending or descending order.

Functions: `prepare`, `prepare_mut`

This function must be used to prepare an iterator before obtaining the next element. It returns a boolean that tells if the iterator can yield another element.

Inputs

- Type `K<copy + drop>`
 - **Validation:** None required.
 - **Impact:** Type of the table keys.
- Type `V`
 - **Validation:** None required.
 - **Impact:** Type of the table values.

- `table_iter: &TableIter<K, V>`
 - **Validation:** None required.
 - **Impact:** Iterator to prepare.

Functions: `next`, `next_mut`

This function can be used to obtain the next element from an iterator. It returns the key and a reference (mutable or immutable) to the table value.

Inputs

- Type `K<copy + drop>`
 - **Validation:** None required.
 - **Impact:** Type of the table keys.
- Type `V`
 - **Validation:** None required.
 - **Impact:** Type of the table values.
- `table_iter: &TableIter<K, V>/&mut TableIter<K, V>`
 - **Validation:** None required.
 - **Impact:** Table iterator to act on.

8.14. Module `object.move`

Function: `address_to_object`

This function can be used to get an instance of `Object` for a given type at a given address.

Inputs

- Type `T<key>`
 - **Validation:** An object of type `T` must exist at the given address.
 - **Impact:** Type of the returned `Object` instance.
- `object: address`
 - **Validation:** An object of type `T` must exist at the given address.
 - **Impact:** Address of the returned `Object`.

Function: `is_object`

This function can be used to test whether an address owns an `Object`.

Inputs

- object: address
 - **Validation:** None required.
 - **Impact:** Address to test.

Function: create_object_address

This function can be used to derive the address of an object given the address of the creator and a bytes seed. Note that this function does not create the object, it only generates the address for an object generated from the given address and seed.

Inputs

- source: address
 - **Validation:** None required.
 - **Impact:** Address of the creator of the object.
- seed: vector<u8>
 - **Validation:** None required.
 - **Impact:** Seed used to derive the object address.

Function: create_user_derived_object_address

This function can be used to derive an object address obtained from the address of the creator and another address. Note that the derived address differs from the address returned by create_object_address, even if the latter is given and encoded address as a seed.

Inputs

- source: address
 - **Validation:** None required.
 - **Impact:** Address of the creator of the object.
- derive_from: address
 - **Validation:** None required.
 - **Impact:** Seed used to derive the object address.

Function: create_guid_object_address

This function can be used to derive an object address from the address of its creator and a sequence number. Note that this function uses a different suffix than create_object_address and create_user_derived_object_address, and therefore it is not possible to derive the same object

address by encoding, for example, a particular byte array as a u64. We also note that this function is currently not used by the Initia standard library.

Inputs

- source: address
 - **Validation:** None required.
 - **Impact:** Address of the creator of the object.
- creation_num: u64
 - **Validation:** None required.
 - **Impact:** Sequence number used as seed to derive the object address.

Function: `object_address`

This function can be used to get the address of an Object.

Inputs

- Type `T<key>`
 - **Validation:** None required.
 - **Impact:** Type associated with the object.
- object: `Object<T>`
 - **Validation:** None required.
 - **Impact:** Object which address is returned.

Function: `convert`

This function can be used to convert an Object of a type into an Object of a different type, owned by the same address. Note that an object of the destination type must exist at the given address.

Inputs

- Type `X<key>`
 - **Validation:** None required.
 - **Impact:** Source type.
- Type `Y<key>`
 - **Validation:** An object of this type must exist at the address of the source object.
 - **Impact:** Destination type.
- object: `Object<X>`
 - **Validation:** An object of the destination type must exist at the address of this source object.

- **Impact:** Object to convert.

Function: `create_named_object`

This function can be used to create an object on behalf of a signer, with its address derived from a bytes seed (ref. `create_object_address`).

We note that the function can be used by any contract, and that although the documentation suggests that "named objects cannot be deleted", the function allows to specify `can_delete = true`. We did not identify an exploitable issue due to this behavior, which we attribute to an error in the documentation.

Inputs

- `creator: &signer`
 - **Validation:** None required.
 - **Impact:** Owner of the object.
- `seed: vector<u8>`
 - **Validation:** None required.
 - **Impact:** Seed used to derive the object address.
- `can_delete: bool`
 - **Validation:** None required.
 - **Impact:** Determines whether the object is marked as deletable.

Function: `create_user_derived_object`

This function can be used to create an object on behalf of an address, with its address derived from the address of another object (ref. `create_user_derived_object_address`). Note that the function can only be called by friend modules, and that the `DeriveRef` required to invoke it can only be obtained from the `ConstructorRef` for the same object.

Inputs

- `creator_address: address`
 - **Validation:** None required.
 - **Impact:** Owner of the object.
- `derive_ref: &DeriveRef`
 - **Validation:** None required.
 - **Impact:** Seed used to derive the object address.
- `can_delete: bool`
 - **Validation:** None required.
 - **Impact:** Determines whether the object is marked as deletable.

Function: `create_object`

This public function can be used to create an object owned by a given address, with a unique address (not controlled by the caller in any meaningful way).

Inputs

- `owner_address`: `address`
 - **Validation**: None required.
 - **Impact**: Owner of the object.
- `can_delete`: `bool`
 - **Validation**: None required.
 - **Impact**: Determines whether the object is marked as deletable.

Function: `generate_delete_ref`

This function can be used to create a `DeleteRef`, a capability that allows to remove an existing object.

Inputs

- `ref`: `&ConstructorRef`
 - **Validation**: The object must have been created as deletable.
 - **Impact**: `ConstructorRef` identifying the object for which the `DeleteRef` capability will be created.

Function: `generate_extend_ref`

This function can be used to create an `ExtendRef`, a capability that allows to extend an existing object. The capability allows to generate a `signer` for the object, and thus to create more child objects on behalf of the object to be extended.

Inputs

- `ref`: `&ConstructorRef`
 - **Validation**: None required.
 - **Impact**: `ConstructorRef` identifying the object for which the capability will be created.

Function: `generate_transfer_ref`

This function can be used to create a `TransferRef`, a capability that allows to transfer ownership of an object as well as to control whether the object can be transferred by its owner without using a `TransferRef` capability (controlling the `allow_ungated_access` flag).

Inputs

- `ref: &ConstructorRef`
 - **Validation:** None required.
 - **Impact:** `ConstructorRef` identifying the object for which the capability will be created.

Function: `generate_derive_ref`

This function can be used to create a `DeriveRef`, a capability that allows to create an object at a deterministic address derived from the address of the creator and the address of the object referred to by the `ConstructorRef`. Note that there is no nonce, and thus for any given caller and `ConstructorRef`, only a single object address can be derived.

Inputs

- `ref: &ConstructorRef`
 - **Validation:** None required.
 - **Impact:** `ConstructorRef` identifying the object for which the capability will be created.

Function: `generate_signer`

This special function can be used to create a signer instance for performing actions on behalf of the address referenced by the `ConstructorRef`.

Inputs

- `ref: &ConstructorRef`
 - **Validation:** None required.
 - **Impact:** `ConstructorRef` identifying the object for which the signer will be created.

Function: delete

This function can be used to delete an object from global storage.

Inputs

- ref: DeleteRef
 - **Validation:** Version of the DeleteRef must match the version of the global ObjectCore object.
 - **Impact:** Identifies the object to be deleted.

Function: generate_signer_for_extending

This function can be used to generate a signer from an ExtendRef.

Inputs

- ref: &ExtendRef
 - **Validation:** Version of the ExtendRef must match the version of the global ObjectCore object.
 - **Impact:** Identifies the object for which the signer will be created.

Functions: enable_ungated_transfer, disable_ungated_transfer

This function can be used to enable or disable ungated transfer. If ungated transfer is enabled, an object can be transferred freely by its owner. When ungated transfer is disabled, transferring ownership of an object requires a corresponding TransferRef (or LinearTransferRef).

Inputs

- ref: &TransferRef
 - **Validation:** Version of the TransferRef must match the version of the global ObjectCore object.
 - **Impact:** Identifies the object for which ungated transfer should be enabled/disabled.

Function: generate_linear_transfer_ref

This function can be used to generate a LinearTransferRef for an object, a one-time-use capability that can be used by the object owner to transfer the object.

Inputs

- `ref: &TransferRef`
 - **Validation:** Version of the `TransferRef` must match the version of the global `ObjectCore` object
 - **Impact:** Identifies the object for which the `LinearTransferRef` will be created.

Function: `transfer_with_ref`

This function can be used by the owner of an object to transfer it to a different address, given a `LinearTransferRef` for the object.

Inputs

- `ref: LinearTransferRef`
 - **Validation:** Version of the `LinearTransferRef` must match the version of the global `ObjectCore` object.
 - **Impact:** Identifies the object to be transferred.
- `to: address`
 - **Validation:** None required.
 - **Impact:** Address of the new owner for the object.

Function: `transfer_raw`

This function can be used by the owner of an object to transfer it. Ungated transfer must be enabled for the object being transferred.

Inputs

- `owner: &signer`
 - **Validation:** Must be the (possibly indirect) owner of the object.
 - **Impact:** Authorizes the transfer.
- `object: address`
 - **Validation:** Must be owned by `owner` and have ungated transfer enabled.
 - **Impact:** Address of the object to be transferred.
- `to: address`
 - **Validation:** None required.
 - **Impact:** Address of the new owner of the object.

8.15. Module hex.move

This module exposes functions `encode_to_string` and `decode_string` that can be used to encode and decode a vector of bytes to/from hexadecimal. An issue with `decode_string` accepting invalid hexadecimal characters is discussed in Finding [3.9.7](#).

8.16. Module code.move

This module exposes functions that allow to publish and upgrade other modules.

Function: `set_allowed_publishers`

This function can only be called by the special `@initia_std` address. It can be used to set the allowlist of addresses that are allowed to publish new modules.

Inputs

- `chain: &signer`
 - **Validation:** Must be `@initia_std`.
 - **Impact:** Used to verify the call is authorized.
- `allowed_publishers: vector<address>`
 - **Validation:** If empty, publishing is not restricted; otherwise, the list must contain `@initia_std`.
 - **Impact:** List of allowed publishers.

Function: `publish`

This function can be used by an allowed publisher to publish or upgrade a module.

Inputs

- `owner: &signer`
 - **Validation:** Must be an authorized publisher and the owner of the module.
 - **Impact:** Used to authorize the operation.
- `module_ids: vector<String>`
 - **Validation:** Must match the IDs of the modules provided in the `code` argument.
 - **Impact:** Used to efficiently check the modules upgrade policy.
- `code: vector<vector<u8>>`
 - **Validation:** Must be a valid module, and the IDs must match with `module_ids`.
 - **Impact:** The code of the modules to be published or upgraded.
- `upgrade_policy: u8`

- **Validation:** Must be greater than zero. Could be further restricted to be at most two (not an exploitable issue). Must be compatible with the existing upgrade policy if upgrading a package.
- **Impact:** New upgrade policy for the published packages.

8.17. Module dex.move

This module implements a DEX between an arbitrary token pair, using the Balancer price curve specialized for pools containing two assets. The module also maintains a centralized registry for all pools, enabling easier discovery of trading pools.

The module implements several convenience functions that simplify its usage. Some functions, identified by the `_script` suffix, are intended as convenience functions that implement some common usage patterns, particularly useful to simplify scripts. These and other trivial functions are not explicitly included in this threat model description as they do not perform any security checks that affect the underlying DEX operation. This does not mean that the helper functions do not perform any security check at all; for instance, `swap_script` includes a check on the output amount, which needs to be performed manually when using the lower-level swap function. However, this check can indeed be implemented by the caller of swap and is not fundamentally required for the correct operation of the DEX. We note that not all convenience functions are suffixed with `_script` (as is the case for `provide_liquidity_from_coin_store`).

Function: `get_spot_price`

This function can be used to compute the spot price for a given pair, in either trade direction. Note that (by design) the spot price is *not* the effective price of a trade, as it only represents the correct price for an infinitesimally small trade.

Inputs

- `pair`: `Object<Config>`
 - **Validation:** Must refer to an existing pool.
 - **Impact:** Identifies the pool.
- `base_coin`: `Object<Metadata>`
 - **Validation:** Must refer to one of the two assets contained in the pool.
 - **Impact:** Determines the base asset for the spot price.

Function: `update_swap_fee_rate`

This function can only be invoked by the special `@initia_std` address. It can be used to set the swap fee rate for a given trading pair. Note that this is the only function that allows to change the swap fee rate for a pool. The initial swap fee rate is set by the creator of the pair at the time of its creation.

Inputs

- chain: &signer
 - **Validation:** Must be @initia_std.
 - **Impact:** Used to authorize the operation.
- pair: Object<Config>
 - **Validation:** No validation required.
 - **Impact:** Identifies the pool to modify.
- swap_fee_rate: Decimal128
 - **Validation:** Must be at most MAX_FEE_RATE (5%).
 - **Impact:** New swap fee rate to set.

Function: withdraw_liquidity

This function can be used to redeem pool tokens for the corresponding amount of the pool assets. The function can only be invoked on pools that are not in a bootstrapping phase.

Inputs

- lp_token: FungibleAsset
 - **Validation:** Must (implicitly) correspond to the address of a pool.
 - **Impact:** Input LP tokens to redeem.
- min_coin_a_amount: Option<u64>
 - **Validation:** None required.
 - **Impact:** If specified, the amount of A asset returned by redeeming the LP tokens must be greater than this amount.
- min_coin_b_amount: Option<u64>
 - **Validation:** None required.
 - **Impact:** If specified, the amount of B asset returned by redeeming the LP tokens must be greater than this amount.

Function: single_asset_provide_liquidity

This function can be used to add liquidity to the pool by supplying only one of the two pool assets. This is effectively implemented as a balanced swap of some of the single-sided liquidity for a corresponding amount of the other asset, followed by a normal balanced liquidity injection. Note that the swap does incur fees, which prevents using this function as a workaround to not be charged for swapping assets.

The function can only be called on pools that are not in the liquidity bootstrapping period.

Inputs

- `pair: Object<Config>`
 - **Validation:** None explicitly required.
 - **Impact:** Identifies the pool to which liquidity is to be added.
- `provide_coin: FungibleAsset`
 - **Validation:** Must be one of the two pool assets.
 - **Impact:** Fungible asset to add to the pool liquidity.
- `min_liquidity_amount: Option<u64>`
 - **Validation:** None required.
 - **Impact:** If provided, the amount of pool LP tokens returned is required to be greater than this.

Function: swap

This function can be used to perform a swap. This is the lowest-level publicly accessible function, used by several other convenience functions.

The function can only be called on pools that are not in the liquidity bootstrapping period.

Inputs

- `pair: Object<Config>`
 - **Validation:** None explicitly required.
 - **Impact:** Identifies the pool where the swap occurs.
- `offer_coin: FungibleAsset`
 - **Validation:** Must be one of the two pool assets.
 - **Impact:** Input asset for the swap.

Function: create_pair

This function can be used to create a new pool.

Inputs

- `creator: &signer`
 - **Validation:** None required.
 - **Impact:** Creator of the pool.
- `name: String`
 - **Validation:** None (could be unrelated to the pool assets and thus misleading).
 - **Impact:** Name of the pool.

- symbol: String
 - **Validation:** None (could be unrelated to the pool assets and thus misleading).
 - **Impact:** Symbol for the pool.
- swap_fee_rate: Decimal128
 - **Validation:** Must be less than 5%.
 - **Impact:** Swap fee rate.
- coin_a: FungibleAsset
 - **Validation:** Must be different from coin_b.
 - **Impact:** First asset of the pool pair.
- coin_b: FungibleAsset
 - **Validation:** Must be different from coin_a.
 - **Impact:** Second asset of the pool pair.
- weights: Weights
 - **Validation:** None.
 - **Impact:** Weights for the pool assets.

Function: provide_liquidity

This function can be used to provide liquidity to the pool. The function is intended to provide balanced liquidity, and any imbalanced amount will be effectively donated to the pool.

Inputs

- pair: Object<Config>
 - **Validation:** None required.
 - **Impact:** Determines the pool where liquidity is to be added.
- coin_a: FungibleAsset
 - **Validation:** Must be the first asset in the pool pair (only implicitly checked).
 - **Impact:** First asset to add to the pool liquidity.
- coin_b: FungibleAsset
 - **Validation:** Must be the second asset in the pool pair (only implicitly checked).
 - **Impact:** Second asset to add to the pool liquidity.
- min_liquidity_amount: Option<u64>
 - **Validation:** None required.
 - **Impact:** If provided, the program asserts the liquidity returned is at least this amount.

Function: swap_simulation

This function can be used to simulate a swap, given an input amount, reserve amounts, and pool parameters (asset weights and swap fee rate). Note that while this function is read only on its own, it is also used by the actual swap function to compute the result of a swap (output amount received

and fees paid).

Inputs

- pool_amount_in: u64
 - **Validation:** N/A.
 - **Impact:** Amount of the input-asset pool reserves.
- pool_amount_out: u64
 - **Validation:** N/A.
 - **Impact:** Amount of the output-asset pool reserves.
- weight_in: Decimal128
 - **Validation:** N/A.
 - **Impact:** Weight of the input asset.
- weight_out: Decimal128
 - **Validation:** N/A.
 - **Impact:** Weight of the output asset.
- amount_in: u64
 - **Validation:** N/A.
 - **Impact:** Amount of input asset provided for the swap.
- swap_fee_rate: Decimal128
 - **Validation:** N/A.
 - **Impact:** Swap fee rate.

Function: swap_simulation_given_out

This function can be used to simulate a swap. Unlike swap_simulation, this function receives the desired output amount and computes the required input amount and the fees to be paid.

Inputs

- pool_amount_in: u64
 - **Validation:** N/A.
 - **Impact:** Amount of the input-asset pool reserves.
- pool_amount_out: u64
 - **Validation:** N/A.
 - **Impact:** Amount of the output-asset pool reserves.
- weight_in: Decimal128
 - **Validation:** N/A.
 - **Impact:** Weight of the input asset.
- weight_out: Decimal128
 - **Validation:** N/A.

- **Impact:** Weight of the output asset.
- amount_in: u64
 - **Validation:** N/A.
 - **Impact:** Desired amount of output asset.
- swap_fee_rate: Decimal128
 - **Validation:** N/A.
 - **Impact:** Swap fee rate.

8.18. Module comparator.move

This module exposes functions that can be used to compare the BCS serialization of various datatypes.

Function: compare

This function can be used to compare (lexicographically) the BCS serialization of two objects of the same type. The returned Result object can be read using the is_equal, is_smaller_than, and is_greater_than functions.

Inputs

- Type T
 - **Validation:** None required.
 - **Impact:** Type of the objects to compare.
- left: &T
 - **Validation:** None required.
 - **Impact:** First object to compare.
- right: &T
 - **Validation:** None required.
 - **Impact:** Second object to compare.

Function: compare_u8_vector

This function can be used to compare two byte vectors directly. The returned Result object can be read using the is_equal, is_smaller_than, and is_greater_than functions.

Inputs

- left: vector<u8>
 - **Validation:** None required.
 - **Impact:** First vector to compare.

- `right: vector<u8>`
 - **Validation:** None required.
 - **Impact:** Second vector to compare.

8.19. Module `managed_coin.move`

This module is a more convenient wrapper for the more general coins module. It allows to more easily define fungible tokens without explicitly managing capabilities.

Function: `initialize`

This function can be used to create a new fungible token.

Inputs

- `account: &signer`
 - **Validation:** None.
 - **Impact:** Creator of the asset — will have the ability to mint and burn.
- `maximum_supply: Option<u128>`
 - **Validation:** None.
 - **Impact:** If provided, sets a maximum supply cap.
- `name: String`
 - **Validation:** None.
 - **Impact:** Name of the new asset.
- `symbol: String`
 - **Validation:** None.
 - **Impact:** Symbol of the new asset.
- `decimals: u8`
 - **Validation:** Must be at most 32 (indirectly).
 - **Impact:** Number of decimal positions for the asset.
- `icon_uri: String`
 - **Validation:** None.
 - **Impact:** URI for the icon of the created asset.
- `project_uri: String`
 - **Validation:** None.
 - **Impact:** URI for the created asset.

Function: `burn`

This function can be used by the creator of a fungible asset to burn a quantity of the asset. The asset is withdrawn from the store associated with the creator address, meaning the creator cannot burn

assets owned by a third party.

Inputs

- `account: &signer`
 - **Validation:** Must be the owner of the fungible-asset metadata.
 - **Impact:** Authorizes the operation.
- `metadata: Object<Metadata>`
 - **Validation:** None required.
 - **Impact:** Identifies the asset to be burned.
- `amount: u64`
 - **Validation:** None explicitly required (account must own a sufficient amount of assets).
 - **Impact:** Amount of assets to burn.

Function: `mint`

This function can be used by the creator of a fungible asset to mint a quantity of the asset.

Inputs

- `account: &signer`
 - **Validation:** Must be the owner of the fungible-asset metadata.
 - **Impact:** Authorizes the operation.
- `dst_addr: address`
 - **Validation:** None.
 - **Impact:** Owner of the newly minted assets.
- `metadata: Object<Metadata>`
 - **Validation:** None required.
 - **Impact:** Identifies the asset to be minted.
- `amount: u64`
 - **Validation:** Must not cause minting of more than the maximum supply.
 - **Impact:** Amount of assets to mint.

8.20. Modules `decimal128.move` and `decimal256.move`

These modules implement functions that can be used to work with fixed precision (18 decimal digits) unsigned numbers that can be represented in, respectively, 128 and 256 bits.

The modules expose functions to create a `Decimal128/Decimal256` number from a raw value (already premultiplied by `1e18`) as well as parsing a number from a string. Note that an issue with num-

ber parsing is discussed in issue [3.11](#), [7](#).

These functions are implemented in `decimal128`:

- `new`: Creates a `Decimal128` from a (premultiplied) `u128` value.
- `new_u64`: Creates a `Decimal128` from a (premultiplied) `u64` value.
- `one`: Creates a `Decimal128` with value 1.
- `zero`: Creates a `Decimal128` with zero value.
- `from_ratio_u64`: Creates a `Decimal128` that represents the ratio between two `u64` integers.
- `from_ratio`: Creates a `Decimal128` that represents the ratio between two `u128` integers.
- `add`: Adds two `Decimal128`, returning a new `Decimal128` as a result.
- `sub`: Adds two `Decimal128`, returning a new `Decimal128` as a result.
- `mul_u64`: Multiplies a `Decimal128` by a `u64` integer value, returning an unscaled `u64` integer result.
- `mul_u128`: Multiplies a `Decimal128` by a `u128` integer value, returning an unscaled `u128` integer result.
- `mul`: Multiplies two `Decimal128` numbers, returning a `Decimal128` result.
- `div_u64`: Divides a `Decimal128` by a `u64` integer value, returning a `Decimal128` result.
- `div`: Divides a `Decimal128` by a `u128` integer value, returning a `Decimal128` result.
- `val`: Retrieves the inner value of the `Decimal128` (which is scaled by `1e18`).
- `is_same`: Compares two `Decimal128`s.
- `from_string`: Parses a number into a `Decimal128`.

And these functions are implemented in `decimal256`:

- `new`: Creates a `Decimal256` from a (premultiplied) `u256` value.
- `new_u64`: Creates a `Decimal256` from a (premultiplied) `u64` value.
- `new_u128`: Creates a `Decimal256` from a (premultiplied) `u128` value.
- `one`: Creates a `Decimal256` with value 1.
- `zero`: Creates a `Decimal256` with zero value.
- `from_ratio_u64`: Creates a `Decimal256` that represents the ratio between two `u64` integers.
- `from_ratio_u128`: Creates a `Decimal256` that represents the ratio between two `u128` integers.
- `from_ratio`: Creates a `Decimal256` that represents the ratio between two `u256` integers.
- `add`: Adds two `Decimal256`, returning a new `Decimal256` as a result.
- `sub`: Adds two `Decimal256`, returning a new `Decimal256` as a result.
- `mul_u64`: Multiplies a `Decimal256` by a `u64` integer value, returning an unscaled `u64` integer result.
- `mul_u128`: Multiplies a `Decimal256` by a `u128` integer value, returning an unscaled `u128` integer result.
- `mul_u256`: Multiplies a `Decimal256` by a `u256` integer value, returning an unscaled `u256` integer result.

- `mul`: Multiplies two `Decimal256` numbers, returning a `Decimal256` result.
- `div_u64`: Divides a `Decimal256` by a `u64` integer value, returning a `Decimal256` result.
- `div_u128`: Divides a `Decimal256` by a `u128` integer value, returning a `Decimal256` result.
- `div`: Divides a `Decimal256` by a `u256` integer value, returning a `Decimal256` result.
- `val`: Retrieves the inner value of the `Decimal256` (which is scaled by `1e18`).
- `is_same`: Compares two `Decimal256s`.
- `from_string`: Parses a number into a `Decimal256`.

8.21. Module `string_utils.move`

This module implements utility functions that help formatting `Move` values as strings.

The following functions return a string representation of any object, differing in minor formatting settings:

- `public fun to_string<T>(s: &T): String`
- `public fun to_string_with_canonical_addresses<T>(s: &T): String` (uses canonical representation for addresses)
- `public fun to_string_with_integer_types<T>(s: &T): String` (qualifies integers with a suffix identifying their type, e.g., `8u128`)
- `public fun debug_string<T>(s: &T): String` (pretty prints the inputs, using indentation and newlines)

Despite `Move` lacking support for `varargs`, the module also provides some limited format string support via these functions:

- `public fun format1<T0: drop>(fmt: &vector<u8>, a: T0): String`
- `public fun format2<T0: drop, T1: drop>(fmt: &vector<u8>, a: T0, b: T1): String`
- `public fun format3<T0: drop, T1: drop, T2: drop>(fmt: &vector<u8>, a: T0, b: T1, c: T2): String`
- `public fun format4<T0: drop, T1: drop, T2: drop, T3: drop>(fmt: &vector<u8>, a: T0, b: T1, c: T2, d: T3): String`

8.22. Module `event.move`

This module implements functions that can be used to emit events. A module can emit an event by invoking the `emit` function with an instance of any structure defined by the same module. This invariant is checked by the `event_validation` verifier.

Function: emit

This function can be used by a module by to emit an event. It cannot be used by scripts (this invariant is enforced by the event_validation verifier).

Inputs

- Type `T<store + drop>`
 - **Validation:** No additional validation required (verifier guarantees this is a struct / struct instantiation defined by the caller).
 - **Impact:** The type of the emitted event.
- `msg: T`
 - **Validation:** No additional validation required.
 - **Impact:** The event to emit.

8.23. Module type_info.move

This module exposes functions that can be used to obtain metadata about a type.

Function: type_of

This function can be used to introspect a given type. It returns a `TypeInfo` object that can be read using trivial helper functions:

```
struct TypeInfo has copy, drop, store {
  account_address: address,
  module_name: vector<u8>,
  struct_name: vector<u8>,
}
```

Inputs

- Type `T`
 - **Validation:** None required.
 - **Impact:** The type to introspect.

Function: type_name

This function can be used to obtain a string that represents the fully qualified name of a type (e.g., `0x1::coin::Coin`).

Inputs

- Type T
 - **Validation:** None required.
 - **Impact:** The type to introspect.

8.24. Modules `any.move` and `copyable_any.move`

These modules implement functions that can be used to wrap/unwrap any object into generic Any objects, which can be used to manipulate the value opaquely.

The two modules differ in that the `copyable` version defines an Any wrapper that has the `copy` ability (and in turn requires the wrapped value to have `copy`), while the `noncopyable` version cannot be copied, but relaxes the `copy` requirement for the wrapped value.

Internally, the object being wrapped is BCS serialized, and only the serialized representation is stored. The original instance of the object being wrapped is destroyed.

Function: `pack`

This function can be used to wrap an instance of an object. Importantly, the object to be wrapped is required to have `store` and `drop`. This prevents the caller from being able to — for example — dispose of an undroppable object and bypass the hot-potato pattern.

Inputs

- Type `T<drop + store>/T<drop + store + copy>`
 - **Validation:** None required.
 - **Impact:** Type of the object to wrap.
- `x: T`
 - **Validation:** None required.
 - **Impact:** Instance of the object to wrap.

Function: `unpack`

This function can be used to unwrap an Any instance to recover the original wrapped object.

Inputs

- Type T
 - **Validation:** Must match the type of the wrapped object contained in the Any.
 - **Impact:** Type of the wrapped object.

- x: Any
 - **Validation:** Must contain an object of type T.
 - **Impact:** Wrapped object to unwrap.

8.25. Module address.move

This module implements functions that can be used to convert an address to and from a string representation.

- public fun from_sdk(sdk_addr: String): address
- public fun to_sdk(vm_addr: address): String

8.26. Module cosmos.move

This module exposes functions that can be used to interact with Cosmos modules by sending messages.

Function: stargate_vote

This function can be used to cast votes on Stargate.

Inputs

- sender: &signer
 - **Validation:** None required.
 - **Impact:** Identifies the sender of the Cosmos message.
- proposal_id: u64
 - **Validation:** None.
 - **Impact:** Identifies the proposal being voted.
- voter: String
 - **Validation:** None (from the Move side).
 - **Impact:** The account address to be stored in the JSON.
- option: u64
 - **Validation:** None.
 - **Impact:** Identifies the option being voted.
- metadata: String
 - **Validation:** None.
 - **Impact:** Metadata associated with the vote message.

Function: stargate

This function can be used to send arbitrary Stargate messages.

Inputs

- sender: `&signer`
 - **Validation:** None required.
 - **Impact:** Identifies the sender of the Cosmos message.
- data: `String`
 - **Validation:** None.
 - **Impact:** JSON-encoded Cosmos message.

Function: move_execute

This function can be used to queue up a message that requests execution of a Move function. Note that this function allows for dynamic dispatching, which is not normally possible in Move. However, this function also has some additional constraints; arguments for the invoked function can only be primitive types or instances of allowlisted structs (such as `String` or `Option`). This ensures the caller cannot instantiate arbitrary Move objects.

Note that validation of the target function and of the arguments is not performed by `move_execute` or by the native helper function it invokes. It is possible to queue up Cosmos messages requesting the call of an invalid function or using invalid parameters. Such invalid calls will cause a revert when the call is processed.

Inputs

- sender: `&signer`
 - **Validation:** None.
 - **Impact:** Sender of the message (caller of the Move function).
- module_address: `address`
 - **Validation:** None.
 - **Impact:** Address of the module to invoke.
- module_name: `String`
 - **Validation:** None.
 - **Impact:** Name of the module to invoke.
- function_name: `String`
 - **Validation:** None.
 - **Impact:** Name of the function to invoke.
- type_args: `vector<String>`
 - **Validation:** None.
 - **Impact:** Type arguments to the invoked function (serialized as string).

- args: `vector<vector<u8>>`
 - **Validation:** None.
 - **Impact:** BCS-serialized arguments.

Function: `move_execute_with_json`

This function can be used to queue up a message that requests execution of a Move function. The function is identical to `move_execute`, with the exception that the function arguments are expected to be encoded as JSON.

Function: `move_script`

This function can be used to queue up a message requesting to run a Move script. The considerations for `move_execute` also apply for this function; it allows dynamic dispatching, has the same constraints limiting the types of the arguments to primitive and allowlisted ones, and no strict checks are performed at the time the message is queued up, resulting in a revert only afterwards.

Inputs

- sender: `&signer`
 - **Validation:** None.
 - **Impact:** Sender of the message (caller of the Move script).
- code_bytes: `vector<u8>`
 - **Validation:** None immediately — must be a valid script.
 - **Impact:** Script to be executed.
- type_args: `vector<String>`
 - **Validation:** None.
 - **Impact:** Type arguments.
- args: `vector<vector<u8>>`
 - **Validation:** None immediately — will be validated before running the script.
 - **Impact:** Arguments supplied to the script — BCS serialized.

Function: `move_script_with_json`

This function can be used to queue up a message requesting to run a Move script. It is identical to `move_script`, with the exception that the arguments are expected to be JSON encoded.

Function: `delegate`

This function can be used to delegate funds to a delegator.

Inputs

- delegator: &signer
 - **Validation:** None.
 - **Impact:** Identifies the delegating party and authorizes the operation.
- validator: String
 - **Validation:** None.
 - **Impact:** Identifies the validator receiving a delegation.
- metadata: Object<Metadata>
 - **Validation:** None.
 - **Impact:** Identifies the asset being delegated.
- amount: u64
 - **Validation:** None.
 - **Impact:** Amount to be delegated.

Function: fund_community_pool

This function can be used to transfer funds to the community pool. The assets are taken from the storage associated with the provided signer.

Inputs

- sender: &signer
 - **Validation:** None.
 - **Impact:** Account funding the community pool.
- metadata: Object<Metadata>
 - **Validation:** None.
 - **Impact:** Identifies the asset to be transferred.
- amount: u64
 - **Validation:** None.
 - **Impact:** Amount to be transferred.

Function: transfer

This function can be used to initiate an IBC transfer of a given amount of fungible tokens.

Inputs

- sender: &signer
 - **Validation:** None.
 - **Impact:** Sender initiating the transfer.

- receiver: String
 - **Validation:** None.
 - **Impact:** Recipient of the transfer.
- metadata: Object<Metadata>
 - **Validation:** None.
 - **Impact:** Identifies the asset to transfer.
- token_amount: u64
 - **Validation:** None.
 - **Impact:** Amount to transfer.
- source_port: String
 - **Validation:** None.
 - **Impact:** Source port for the IBC transfer.
- source_channel: String
 - **Validation:** None.
 - **Impact:** Source channel for the IBC transfer.
- revision_number: u64
 - **Validation:** None.
 - **Impact:** Specifies the height-based time-out for the transfer.
- revision_height: u64
 - **Validation:** None.
 - **Impact:** Specifies the height-based time-out for the transfer.
- timeout_timestamp: u64
 - **Validation:** None.
 - **Impact:** Time stamp specifying the time-out for the transfer.
- memo: String
 - **Validation:** None.
 - **Impact:** Arbitrary data attached to the IBC-transfer message.

Function: nft_transfer

This function can be used to initiate an IBC transfer of a nonfungible asset.

Inputs

- sender: &signer
 - **Validation:** None.
 - **Impact:** Sender initiating the transfer.
- receiver: String
 - **Validation:** None.
 - **Impact:** Recipient of the transfer.
- collection: Object<Collection>
 - **Validation:** None.

- **Impact:** Identifies the collection of which the asset being transferred is a part of.
- token_ids: vector<String>
 - **Validation:** None.
 - **Impact:** Identifies the specific asset to be transferred.
- source_port: String
 - **Validation:** None.
 - **Impact:** Source port for the IBC transfer.
- source_channel: String
 - **Validation:** None.
 - **Impact:** Source channel for the IBC transfer.
- revision_number: u64
 - **Validation:** None.
 - **Impact:** Specifies the height-based time-out for the transfer.
- revision_height: u64
 - **Validation:** None.
 - **Impact:** Specifies the height-based time-out for the transfer.
- timeout_timestamp: u64
 - **Validation:** None.
 - **Impact:** Time stamp specifying the time-out for the transfer.
- memo: String
 - **Validation:** None.
 - **Impact:** Arbitrary data attached to the IBC-transfer message.

Function: pay_fee

This function can be used to pay for IBC-transfer fees.

Inputs

- sender: &signer
 - **Validation:** None.
 - **Impact:** Sender paying the fees.
- source_port: String
 - **Validation:** None.
 - **Impact:** Source port for the IBC transfer.
- source_channel: String
 - **Validation:** None.
 - **Impact:** Source channel for the IBC transfer.
- recv_fee_metadata: Object<Metadata>
 - **Validation:** None.
 - **Impact:** Asset to be used to pay for receive fees.

- `recv_fee_amount`: `u64`
 - **Validation**: None.
 - **Impact**: Amount to be used as a receive fee.
- `ack_fee_metadata`: `Object<Metadata>`
 - **Validation**: None.
 - **Impact**: Asset to be used to pay for fees in case of message acknowledgment.
- `ack_fee_amount`: `u64`
 - **Validation**: None.
 - **Impact**: Amount to be used as an acknowledgment fee.
- `timeout_fee_metadata`: `Object<Metadata>`
 - **Validation**: None.
 - **Impact**: Asset to be used for time-out fees.
- `timeout_fee_amount`: `u64`
 - **Validation**: None.
 - **Impact**: Amount to be used to pay fees in case of message time-out.

8.27. Module `multisig.move`

This module allows to create and manage multi-sig accounts with a set of N members. Any of the members can create a proposal for executing a transaction from the account, and a minimum number of members K ($0 \leq K \leq N$) must authorize the proposal to execute it.

Function: `create_multisig_account`

This function can be used to create a new multi-sig account.

Inputs

- `account`: `&signer`
 - **Validation**: Must be a member of the multi-sig.
 - **Impact**: Creator of the multi-sig account.
- `name`: `String`
 - **Validation**: None required.
 - **Impact**: Used to derive the address of the multi-sig.
- `members`: `vector<address>`
 - **Validation**: Must contain account.
 - **Impact**: Members of the multi-sig.
- `threshold`: `u64`
 - **Validation**: Must be at most the number of members.
 - **Impact**: Number of signers required to perform an action.
- `max_voting_period_height`: `Option<u64>`

- **Validation:** None.
- **Impact:** Optional maximum number of blocks limiting the voting period of a proposal.
- `max_voting_period_timestamp`: `Option<u64>`
 - **Validation:** None.
 - **Impact:** Optional maximum time window limiting the voting period of a proposal.

Function: `create_proposal`

This function can be used by a member of the multi-sig to create a new proposal.

Inputs

- `account`: `&signer`
 - **Validation:** Must be a member of the multi-sig.
 - **Impact:** Authorizes proposal creation.
- `multisig_addr`: `address`
 - **Validation:** Must be the address of a multi-sig.
 - **Impact:** Address of the multi-sig for which the proposal will be created.
- `module_address`: `address`
 - **Validation:** None.
 - **Impact:** Address of the module to call.
- `module_name`: `String`
 - **Validation:** None.
 - **Impact:** Name of the module to call.
- `function_name`: `String`
 - **Validation:** None.
 - **Impact:** Name of the function to call.
- `type_args`: `vector<String>`
 - **Validation:** None.
 - **Impact:** Type arguments provided to the function to call.
- `args`: `vector<vector<u8>>`
 - **Validation:** None.
 - **Impact:** BCS-serialized arguments provided to the function to call.

Function: `vote_proposal`

This function can be used by a member of a multi-sig to endorse or reject a proposal.

Inputs

- `account: &signer`
 - **Validation:** Must be a member of the multi-sig.
 - **Impact:** Authorizes the vote.
- `multisig_addr: address`
 - **Validation:** Must be the address of a multi-sig.
 - **Impact:** Address of the multi-sig.
- `proposal_id: u64`
 - **Validation:** Must be the ID of a nonexpired pending proposal for multi-sig_address.
 - **Impact:** Identifies the proposal to be voted on.
- `vote_yes: bool`
 - **Validation:** None.
 - **Impact:** Determines whether the vote endorses or rejects a proposal.

Function: `execute_proposal`

This function can be used to execute a proposal that has accrued enough votes from its members.

Inputs

- `account: &signer`
 - **Validation:** Must be a member of the multi-sig.
 - **Impact:** Authorizes execution of the proposal.
- `multisig_addr: address`
 - **Validation:** Must be a multi-sig.
 - **Impact:** Address of the multi-sig.
- `proposal_id: u64`
 - **Validation:** Must be the ID of a pending, nonexpired proposal that has accrued enough positive votes.
 - **Impact:** Identifies the proposal to be executed.

Function: `update_config`

This function can be used to update the configuration of a multi-sig: its members, the required threshold, and the maximum voting period windows. The function can only be invoked by the multi-sig address on itself, by executing a proposal voted by enough members.

Inputs

- `account: &signer`
 - **Validation:** Address of the multi-sig to update.
 - **Impact:** Multi-sig to update.
- `new_members: vector<address>`
 - **Validation:** Must contain at least `new_threshold` members and contain unique members.
 - **Impact:** Set of new members of the multi-sig.
- `new_threshold: u64`
 - **Validation:** Must be at most the number of `new_members`.
 - **Impact:** New required threshold.
- `new_max_voting_period_height: Option<u64>`
 - **Validation:** None.
 - **Impact:** Optional new maximum number of blocks for which the proposal voting period can last.
- `new_max_voting_period_timestamp: Option<u64>`
 - **Validation:** None.
 - **Impact:** Optional new maximum time window for which the proposal voting period can last.

8.28. Module `transaction_context.move`

This module exposes utility functions related to the transaction context.

Function: `get_transaction_hash`

This function can be used to get the hash of the executing transaction.

Function: `generate_unique_address`

This function can be used to generate a new unique address. The address is derived from the hash of the current transaction and a sequence number.

8.29. Module `debug.move`

The native functions when not in testing mode will act as a no-op (no operation) for all of the following.

Function: print

This function can be used by any caller to print a value to the console during testing.

Inputs

Full prototype: `public fun print<T>(x: &T)`

- Type `T`
 - **Validation:** None.
 - **Impact:** Determines the type of input `x` to be printed.
- `x: &T`
 - **Validation:** None.
 - **Impact:** The value to be printed.

Function: print_stack_trace

This function can be used by any caller to print a stack trace to the console during testing.

Inputs

Full prototype: `public fun print_stack_trace()`

8.30. Module `table_key.move`**Function: encode_u64**

This function returns big-endian encoding of a `u64` value.

Inputs

Full prototype: `public fun encode_u64(key: u64): vector<u8>`

- `key: u64`
 - **Validation:** None.
 - **Impact:** The key to encode to a vector of bytes and reverse.

Function: decode_u64

This function returns the `u64` value of a big-endian-encoded vector of bytes.

Inputs

Full prototype: `public fun decode_u64(key_bytes: vector<u8>): u64`

- `key_bytes: vector<u8>`
 - **Validation:** None.
 - **Impact:** The vector of bytes to reverse and decode to a u64 value.

Function: `encode_u128`

This function returns big-endian encoding of a u128 value.

Inputs

Full prototype: `public fun encode_u128(key: u128): vector<u8>`

- `key: u128`
 - **Validation:** None.
 - **Impact:** The key to encode to a vector of bytes and reverse.

Function: `decode_u128`

This function returns the u128 value of a big-endian-encoded vector of bytes.

Inputs

Full prototype: `public fun decode_u128(key_bytes: vector<u8>): u128`

- `key_bytes: vector<u8>`
 - **Validation:** None.
 - **Impact:** The vector of bytes to reverse and decode to a u128 value.

Function: `encode_u256`

This function returns big-endian encoding of a u256 value.

Inputs

Full prototype: `public fun encode_u256(key: u256): vector<u8>`

- `key: u256`
 - **Validation:** None.

- **Impact:** The key to encode to a vector of bytes and reverse.

Function: decode_u256

This function returns the u256 value of a big-endian-encoded vector of bytes.

Inputs

Full prototype: `public fun decode_u256(key_bytes: vector<u8>): u256`

- `key_bytes: vector<u8>`
 - **Validation:** None.
 - **Impact:** The vector of bytes to reverse and decode to a u256 value.

8.31. Module account.move

Function: create_account_script

This function is used by chain-external calls to create accounts of the base type.

Inputs

Full prototype: `public entry fun create_account_script(addr: address)`

- `addr: address`
 - **Validation:** The account does not already exist.
 - **Impact:** The address of the new account that is created.

Function: create_account

This function is used by on-chain code to create new accounts.

Inputs

Full prototype: `public fun create_account(addr: address): u64`

- `addr: address`
 - **Validation:** The account does not already exist.
 - **Impact:** The address of the new account that is created.

Function: create_table_account

This function is used internally by the standard library to create accounts for storing tables.

Inputs

Full prototype: `public(friend) fun create_table_account(addr: address): u64`

- `addr: address`
 - **Validation:** The account does not already exist.
 - **Impact:** The address of the new account that is created.

Function: create_object_account

Internal function used by the standard library to create an account for holding the object.

Inputs

Full prototype: `public(friend) fun create_object_account(addr: address): u64`

- `addr: address`
 - **Validation:** The account does not already exist, or if it exists, it is an object account.
 - **Impact:** The address of the new account that is created.

Function: exists_at

Checks if an account exists at an address.

Inputs

Full prototype: `#[view] public fun exists_at(addr: address): bool`

- `addr: address`
 - **Validation:** None necessary.
 - **Impact:** The address is checked for the existence of an account.

Function: get_account_number

Gets the account number of an account.

Inputs

Full prototype: `#[view] public fun get_account_number(addr: address): u64`

- `addr: address`
 - **Validation:** Checks that the account exists.
 - **Impact:** The address whose account is queried.

Function: `get_sequence_number`

Gets the sequence number of an account.

Inputs

Full prototype: `#[view] public fun get_sequence_number(addr: address): u64`

- `addr: address`
 - **Validation:** Checks that the account exists.
 - **Impact:** The address whose account is queried.

Function: `is_base_account`

Checks if the address is a base account type. That is, not a table, object, or other special type of account.

Inputs

Full prototype: `#[view] public fun is_base_account(addr: address): bool`

- `addr: address`
 - **Validation:** Checks that the account exists.
 - **Impact:** The address whose account is queried.

Function: `is_object_account`

Checks if the address is an object account type.

Inputs

Full prototype: `#[view] public fun is_object_account(addr: address): bool`

- `addr: address`

- **Validation:** Checks that the account exists.
- **Impact:** The address whose account is queried.

Function: `is_table_account`

Checks if the address is an table account type.

Inputs

Full prototype: `#[view] public fun is_table_account(addr: address): bool`

- `addr: address`
 - **Validation:** Checks that the account exists.
 - **Impact:** The address whose account is queried.

Function: `is_module_account`

Checks if the address is a module account type.

Inputs

Full prototype: `#[view] public fun is_module_account(addr: address): bool`

- `addr: address`
 - **Validation:** Checks that the account exists.
 - **Impact:** The address whose account is queried.

Function: `get_account_info`

Calls the Cosmos `get_account_info` interface.

Inputs

Full prototype: `native public fun get_account_info(addr: address): (bool /* found */, u64 /* account_number */, u64 /* sequence_number */, u8 /* account_type */)`

- `addr: address`
 - **Validation:** None necessary — error returned if not found.
 - **Impact:** The address whose data is queried.

Function: create_address

Parses an address from a u8 vector into an address type.

Inputs

Full prototype: `native public(friend) fun create_address(bytes: vector<u8>): address`

- `bytes: vector<u8>`
 - **Validation:** The vector is of the correct length.
 - **Impact:** The bytes that are parsed into the address.

Function: create_signer

This internal function converts an address into a signer for non-base accounts.

Inputs

Full prototype: `native public(friend) fun create_signer(addr: address): signer`

- `addr: address`
 - **Validation:** None.
 - **Impact:** The passed-in address is converted into a signer.

8.32. Module base64.move

Function: to_string

Converts binary data to a Base64-encoded string.

Inputs

Full prototype: `public fun to_string(bytes: vector<u8>): String`

- `bytes: vector<u8>`
 - **Validation:** None necessary.
 - **Impact:** The data is converted into a Base64 string.

Function: from_string

Converts a Base64 string into a vector of u8.

Inputs

Full prototype: `public fun from_string(str: String): vector<u8>`

- `str: String`
 - **Validation:** An error is returned if the underlying library reports an error.
 - **Impact:** The string is parsed into binary data.

8.33. Module `from_bcs.move`

Function: `to_bool`

This function converts a vector of bytes to a boolean value.

Inputs

Full prototype: `public fun to_bool(v: vector<u8>): bool`

- `v: vector<u8>`
 - **Validation:** Must pass validation checks done by `Value::simple_deserialize` — must be a BCS-encoded boolean.
 - **Impact:** Converts a BCS-encoded vector of bytes to the desired target type.

Function: `to_u8`

This function converts a vector of bytes to a u8 value.

Inputs

Full prototype: `public fun to_u8(v: vector<u8>): u8`

- `v: vector<u8>`
 - **Validation:** Must pass validation checks done by `Value::simple_deserialize` — must be a BCS-encoded u8.
 - **Impact:** Converts a BCS-encoded vector of bytes to the desired target type.

Function: `to_u16`

This function converts a vector of bytes to a u16 value.

Inputs

Full prototype: `public fun to_u16(v: vector<u8>): u16`

- `v: vector<u8>`
 - **Validation:** Must pass validation checks done by `Value::simple_deserialize` — must be a BCS-encoded u16.
 - **Impact:** Converts a BCS-encoded vector of bytes to the desired target type.

Function: to_u32

This function converts a vector of bytes to a u32 value.

Inputs

Full prototype: `public fun to_u32(v: vector<u8>): u32`

- `v: vector<u8>`
 - **Validation:** Must pass validation checks done by `Value::simple_deserialize` — must be a BCS-encoded u32.
 - **Impact:** Converts a BCS-encoded vector of bytes to the desired target type.

Function: to_u64

This function converts a vector of bytes to a u64 value.

Inputs

Full prototype: `public fun to_u64(v: vector<u8>): u64`

- `v: vector<u8>`
 - **Validation:** Must pass validation checks done by `Value::simple_deserialize` — must be a BCS-encoded u64.
 - **Impact:** Converts a BCS-encoded vector of bytes to the desired target type.

Function: to_u128

This function converts a vector of bytes to a u128 value.

Inputs

Full prototype: `public fun to_u128(v: vector<u8>): u128`

- `v: vector<u8>`
 - **Validation:** Must pass validation checks done by `Value::simple_deserialize` — must be a BCS-encoded `u128`.
 - **Impact:** Converts a BCS-encoded vector of bytes to the desired target type.

Function: `to_u256`

This function converts a vector of bytes to a `u256` value.

Inputs

Full prototype: `public fun to_u256(v: vector<u8>): u256`

- `v: vector<u8>`
 - **Validation:** Must pass validation checks done by `Value::simple_deserialize` — must be a BCS-encoded `u256`.
 - **Impact:** Converts a BCS-encoded vector of bytes to the desired target type.

Function: `to_address`

This function converts a vector of bytes to an address value.

Inputs

Full prototype: `public fun to_address(v: vector<u8>): address`

- `v: vector<u8>`
 - **Validation:** Must pass validation checks done by `Value::simple_deserialize` — must be a BCS-encoded address.
 - **Impact:** Converts a BCS-encoded vector of bytes to the desired target type.

Function: `to_bytes`

This function converts a vector of bytes to a vector of bytes.

Inputs

Full prototype: `public fun to_bytes(v: vector<u8>): vector<u8>`

- `v: vector<u8>`
 - **Validation:** Must pass validation checks done by `Value::simple_deserialize` — must be a BCS-encoded vector of bytes.

- **Impact:** Converts a BCS-encoded vector of bytes to the desired target type.

Function: to_vector_bytes

This function converts a vector of bytes to a vector of vectors of bytes.

Inputs

Full prototype: `public fun to_vector_bytes(v: vector<u8>): vector<vector<u8>>`

- `v: vector<u8>`
 - **Validation:** Must pass validation checks done by `Value::simple_deserialize` — must be a BCS-encoded vector of vectors of bytes.
 - **Impact:** Converts a BCS-encoded vector of bytes to the desired target type.

Function: to_vector_string

This function can be used by the user to convert a vector of bytes to a vector of strings.

Inputs

Full prototype: `public fun to_vector_string(v: vector<u8>): vector<String>`

- `v: vector<u8>`
 - **Validation:** Must pass validation checks done by `Value::simple_deserialize` — must be a BCS-encoded vector of strings. Note that we documented a finding for the lack of UTF-8 validation in this function in Finding [3.3.7](#).
 - **Impact:** Converts a BCS-encoded vector of bytes to the desired target type.

Function: to_string

This function converts a vector of bytes to a string.

Inputs

Full prototype: `public fun to_string(v: vector<u8>): String`

- `v: vector<u8>`
 - **Validation:** Must pass validation checks done by `Value::simple_deserialize` — must be a BCS-encoded string. Must

be a well-formed UTF-8 string.

- **Impact:** Converts a BCS-encoded vector of bytes to the desired target type.

8.34. Module json.move

Function: empty

Creates a new empty JSON object.

Inputs

Full prototype: `public fun empty(): JsonObject`

Function: data

Gets a reference to the inner SimpleMap.

Inputs

Full prototype: `public fun data(json_obj: &JsonObject): &SimpleMap<JsonIndex, JsonElem>`

- `json_obj: &JsonObject`
 - **Validation:** None, but it must be of the correct type.
 - **Impact:** The JSON object whose inner SimpleMap is returned as a reference.

Function: stringify

Computes the string representation of the passed-in JSON object.

Inputs

Full prototype: `public fun stringify(json_obj: &JsonObject): String`

- `json_obj: &JsonObject`
 - **Validation:** None, but the object must be created through this module.
 - **Impact:** The object is converted into a string.

Function: parse

Turns a JSON-string representation into a JSON object.

Inputs

Full prototype: `public fun parse(json_string: String): JsonObject`

- `json_string: String`
 - **Validation:** Validated while parsing.
 - **Impact:** The string that is converted into a JSON object.

Function: find

Finds the `JsonIndex` matching a string key in a `JsonObject` given a parent `JsonIndex`.

Inputs

Full prototype: `public fun find(obj: &JsonObject, index: &JsonIndex, key: &String): JsonIndex`

- `obj: &JsonObject`
 - **Validation:** None, but the object must be created through this module.
 - **Impact:** The object on which the lookup is done.
- `index: &JsonIndex`
 - **Validation:** None, but the lookup of this index must succeed, or it will result in a revert.
 - **Impact:** This is the parent key where the lookup is performed.
- `key: &String`
 - **Validation:** None necessary.
 - **Impact:** The string key being looked up.

8.35. Module oracle.move

Function: get_price

This function can be used by any caller to extract the price of a given asset pair. The price is returned as a tuple of three values: `(u256, u64, u64)`. The first value is the price of the asset pair, the second value is the time stamp of the price, and the third value is the price's number of decimals.

Inputs

Full prototype: `#[view] public fun get_price(pair_id: String): (u256, u64, u64)`

- `pair_id: String`
 - **Validation:** Must be a oracle stored in `NativeOracleContract`'s prices map.
 - **Impact:** Specifies which asset pair's price to get.

8.36. Module zapping.move

Function: batch_claim_zapping_script

This function is used by external users to claim expired zapping stakes in a batch.

Inputs

Full prototype: `public entry fun batch_claim_zapping_script(account: &signer, zids: vector<u64>,)` acquires ModuleStore, LSStore

- `account: &signer`
 - **Validation:** Must be a signer.
 - **Impact:** This is the account from which the zappings are unlocked.
- `zids: vector<u64>`
 - **Validation:** Checks that the zids are present under the account.
 - **Impact:** These zids are unlocked.

Function: batch_claim_reward_script

This function is used by external users to claim the rewards on the staked zappings in a batch.

Inputs

Full prototype: `public entry fun batch_claim_reward_script(account: &signer, zids: vector<u64>,)` acquires ModuleStore, LSStore

- `account: &signer`
 - **Validation:** Must be a signer.
 - **Impact:** This is the account from which the zapping rewards are claimed.
- `zids: vector<u64>`
 - **Validation:** Checks that the zids are present under the account.
 - **Impact:** These zids will have their staking reward claimed.

Function: claim_zapping_script

This function is used by external users to claim expired zapping stakes.

Inputs

Full prototype: `public entry fun claim_zapping_script(account: &signer, zid: u64,)` acquires ModuleStore, LSStore

- `account: &signer`
 - **Validation:** Must be a signer.
 - **Impact:** This is the account from which the zapping is unlocked.
- `zid: u64`
 - **Validation:** Checks that the `zid` is present under the account.
 - **Impact:** This `zid` is unlocked.

Function: `claim_reward_script`

This function is used by external users to claim the rewards on the staked zapping.

Inputs

Full prototype: `public entry fun claim_reward_script(account: &signer, zid: u64) acquires ModuleStore, LSStore`

- `account: &signer`
 - **Validation:** Must be a signer.
 - **Impact:** This is the account from which the zapping reward is claimed.
- `zid: u64`
 - **Validation:** Checks that the `zid` is present under the account.
 - **Impact:** This `zid` will have its staking reward claimed.

Function: `update_lock_period_script`

This function is used by the chain special signer to update the lock period on zappings. It does not affect existing zappings.

Inputs

Full prototype: `public entry fun update_lock_period_script(chain: &signer, lock_period: u64,) acquires ModuleStore`

- `chain: &signer`
 - **Validation:** Must be chain.
 - **Impact:** None, used for validation only.
- `lock_period: u64`
 - **Validation:** None necessary as this function will only be called by trusted users.
 - **Impact:** The new lock period is this value.

Function: claim

Claims an unlocked zapping and returns the inner Delegation and accrued rewards.

Inputs

Full prototype: `public fun claim(zapping: Zapping, zid: u64): (Delegation, Fungible-Asset)`

- `zapping: Zapping`
 - **Validation:** None, but this type can only be created in this module.
 - **Impact:** This zapping is claimed.
- `zid: u64`
 - **Validation:** None, but this function is only callable by callers with a Zapping, all of which are trusted.
 - **Impact:** The id is emitted in an event.

8.37. Module reward.move**Function: register_reward_store**

This is an internal function used to create the reward store.

Inputs

Full prototype: `public(friend) fun register_reward_store<Vesting: copy + drop + store>(chain: &signer, bridge_id: u64,)`

- Type `Vesting<copy + drop + store>`
 - **Validation:** Not validated but only called internally with valid arguments.
 - **Impact:** The type of the vesting (either user or operator). It is only used for determining the hash seed for the object.
- `chain: &signer`
 - **Validation:** None, but it must be a signer.
 - **Impact:** Used as the source address for creating the reward-store address.
- `bridge_id: u64`
 - **Validation:** None, but the creation will fail if the reward store already exists.
 - **Impact:** The bridge ID for which the reward store is created.

Function: add_reward_per_stage

This is an internal function used to update the internally accounted amount of reward for a given stage in the reward store. The actual reward coins are directly deposited.

Inputs

Full prototype: `public(friend) fun add_reward_per_stage(reward_store_addr: address, stage: u64, reward: u64)` acquires RewardStore

- `reward_store_addr: address`
 - **Validation:** None, but must be able to acquire the reward store there.
 - **Impact:** The reward store whose stage reward is updated.
- `stage: u64`
 - **Validation:** None, but this function only has trusted callers.
 - **Impact:** The stage for which the rewards are updated.
- `reward: u64`
 - **Validation:** None, but this function only has trusted callers.
 - **Impact:** This amount is added to the stage reward at the reward store.

Function: `withdraw`

Internal function that withdraws reward coins from this reward store.

Inputs

Full prototype: `public(friend) fun withdraw(reward_store_addr: address, amount: u64,): FungibleAsset` acquires RewardStore

- `reward_store_addr: address`
 - **Validation:** None, but it must be able to acquire the reward store there.
 - **Impact:** The reward store that is withdrawn from.
- `amount: u64`
 - **Validation:** None, but this function only has trusted callers.
 - **Impact:** The amount to withdraw.

9. Assessment Results

During our assessment on the scoped Initia components, we discovered 34 findings. Two critical issues were found. Seven were of high impact, five were of medium impact, 12 were of low impact, and the remaining findings were informational in nature.

9.1. Disclaimer

This assessment does not provide any warranties about finding all possible issues within its scope; in other words, the evaluation results do not guarantee the absence of any subsequent issues. Zellic, of course, also cannot make guarantees about any code added to the project after the version reviewed during our assessment. Furthermore, because a single assessment can never be considered comprehensive, we always recommend multiple independent assessments paired with a bug bounty program.

For each finding, Zellic provides a recommended solution. All code samples in these recommendations are intended to convey how an issue may be resolved (i.e., the idea), but they may not be tested or functional code. These recommendations are not exhaustive, and we encourage our partners to consider them as a starting point for further discussion. We are happy to provide additional guidance and advice as needed.

Finally, the contents of this assessment report are for informational purposes only; do not construe any information in this report as legal, tax, investment, or financial advice. Nothing contained in this report constitutes a solicitation or endorsement of a project by Zellic.

10. Appendix

10.1. Proof of concept: Reference safety verifier bypass

The following test shows how the iterator feature of the table module can be abused to retain two mutable references, which appear to be independent to the Move verifier but in fact refer to the same underlying object.

```
#[test_only]
struct TestItem has key, store, drop {
    v: u64,
}

#[test_only]
fun test_print_refs<T, S>(t: &mut T, s: &mut S) {
    print(&utf8(b"Got references to these two objects:"));
    print(t);
    print(s);
}

#[test_only]
fun test_modify_ref1(t: &mut TestItem, _s: &mut TestItem, v: u64) {
    t.v = v;
}

#[test_only]
fun test_modify_ref2(_t: &mut TestItem, s: &mut TestItem, v: u64) {
    s.v = v;
}

#[test(account = @0x1)]
fun test_table_double_mut_ref_poc(account: signer) {
    let t = new<u64, TestItem>();

    add(&mut t, 1, TestItem { v: 1 });
    let iter1 = iter_mut(&mut t, option::none(), option::none(), 1);
    let iter2 = iter_mut(&mut t, option::none(), option::none(), 1);

    assert!(prepare_mut(&mut iter1), 0);
    let (_, ref1) = next_mut(&mut iter1);

    assert!(prepare_mut(&mut iter2), 0);
    let (_, ref2) = next_mut(&mut iter2);

    test_print_refs(ref1, ref2);

    test_modify_ref1(ref1, ref2, 0x42);
    assert!(ref1.v == 0x42, 1);
}
```

```
assert!(ref2.v == 0x42, 2);

test_modify_ref2(ref1, ref2, 0x43);
assert!(ref1.v == 0x43, 3);
assert!(ref2.v == 0x43, 4);

move_to(&account, TableHolder { t });
}
```