OtterSec

Initia

Security Assessment

November 20th, 2024 — Prepared by OtterSec

James Wang	james.wang@osec.io
Sebastian	sebastian@osec.io
Aleksandre Khokhiashvili	khokho@osec.io
Robert Chen	r@osec.io

Table of Contents

01 — Executive Summary

Overview

Inita engaged OtterSec to assess the program. This assessment was conducted between June 3rd and November 5th, 2024. For more information on our auditing methodology, refer to Appendix B.

Key Findings

We produced 34 findings throughout this audit engagement.

In particular, we identified several critical vulnerabilities, one of them concerning the possibility of delegators escaping slashing by redelegating and then undelegating, as Initia's implementation of slash redelegation fails to account for the redelegate-to-undelegate sequence, enabling them to avoid penalties for prior infractions (OS-INI-ADV-00). Additionally, the bridge system may experience a sequence deadlock if panics occur during hook execution, blocking further transactions (OS-INI-ADV-01). We also highlighted several inconsistencies in the calculation of the spot price in the Cosmos SDk and in the move-based decentralized exchange pool, resulting in the incorrect derivation of the spot price (OS-INI-ADV-03), and in the staking implementation concerning incorrect rounding, slash evasion and the possibility of panicking (OS-INI-ADV-05).

Furthermore, during the finalization of the token deposit, the coins are not reclaimed and burned when a bridging deposit fails, resulting in an infinite mining bug (OS-INI-ADV-04), and the utilization of the pending deposits mechanism, which risks losing funds when deposits fail, as these funds are not included in emitted events (OS-INI-ADV-06).

Additionally, we identified numerous discrepancies concerning gas management, including an error in the movevm that may result in gas undercharging issues (OS-INI-ADV-08), and a denial-of-service due to Cosmos gas meter panic on gas exhaustion in EVM precompiled contracts, allowing callee contracts to forcefully abort caller contracts through unhandled panics (OS-INI-ADV-13).

We also made recommendations to address a few caveats which may be improved for better robustness and ease of utilization of the Opinit bot (OS-INI-SUG-02) and suggested the need to ensure adherence to coding best practices, spcifically concerning feature parity (OS-INI-SUG-06). Moreover, we advised the removal of unutilized and redundant code within the system for increased readability (OS-INI-SUG-03) and streamlining the code to reduce complexity, eliminate redundancy, and enhance readability (OS-INI-SUG-04).

02 — Scope

The source code was delivered to us in a Git repository at https://github.com/initia-labs. This audit was performed against `api/iterator.go`, `crates/natives`, `crates/storage`, `crates/vm` diffs against aptos-vm, `libmovevm`, `precompile`, and `lib.go` files for movevm, opinit, opinit-bots, `x/bank`, `x/distribution`, `x/gov`, `x/ibc`, `x/move`, `x/ibc-hooks`, `x/mstaking`, `x/reward`, `crypto`, and `tx` files for initia and `app`, `indexer`, `jsonrpc`, and `x` files for minievm.

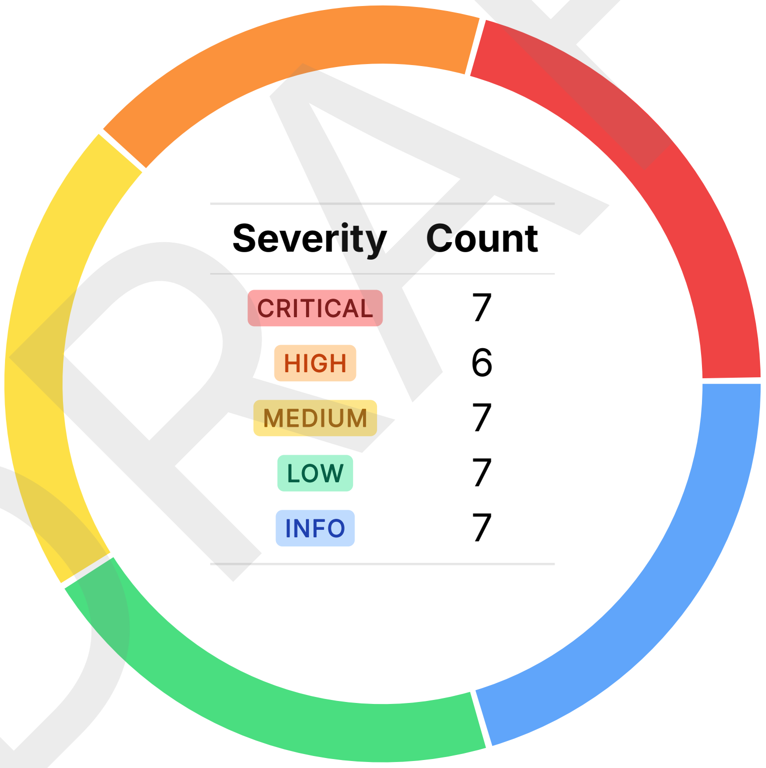
A brief description of the program is as follows:

Name	Description
movevm	Modified aptos-move to work with initia cosmos chain
opinit	Initia optimistic rollup L1 and L2 infrastructure built on cosmos
opinit-bots	Opinit offchain relayer for coordinating interactions between L1 and L2
initia	The main initia chain that integrates movevm into cosmos
minievm	Glue for linking a custom port of geth to cosmos chains

03 — Findings

Overall, we reported 34 findings.

We split the findings into **vulnerabilities** and **general findings**. Vulnerabilities have an immediate impact and should be remediated as soon as possible. General findings do not have an immediate impact but will aid in mitigating future vulnerabilities.



Severity	Count
CRITICAL	7
HIGH	6
MEDIUM	7
LOW	7
INFO	7

04 — Vulnerabilities

Here, we present a technical analysis of the vulnerabilities we identified during our audit. These vulnerabilities have *immediate* security implications, and we recommend remediation as soon as possible.

Rating criteria can be found in Appendix A.

ID	Severity	Status	Description
OS-INI-ADV-00	CRITICAL	RESOLVED ✓	Delegators may escape slashing by redelegating and then undelegating, as Initia's implementation of `SlashRedelegation` fails to account for the redelegate-to-undelegate sequence, enabling them to avoid penalties for prior infractions.
OS-INI-ADV-01	CRITICAL	RESOLVED ✓	The bridge system can experience a sequence deadlock if panics occur during hook execution or if receivers refuse to finalize withdrawal messages, blocking further transactions.
OS-INI-ADV-02	CRITICAL	RESOLVED ✓	The custom transfer function in EVM-cosmos interoperability bypasses EVM's cached balance in `state_object`, resulting in inaccuracies with the `opBalance` opcode.
OS-INI-ADV-03	CRITICAL	RESOLVED ✓	There are multiple inconsistencies in the calculation of the spot price in the Cosmos SDk and in the move-based `dex` pool, resulting in the incorrect derivation of the spot price.
OS-INI-ADV-04	CRITICAL	RESOLVED ✓	`FinalizeTokenDeposit` lacks coin reclamation and burning when a bridging deposit fails, resulting in an infinite mining bug.
OS-INI-ADV-05	CRITICAL	RESOLVED ✓	The staking implementation contains multiple inconsistencies such as rounding, evasion of slashing and the possibility of panicking.

OS-INI-ADV-06	CRITICAL	RESOLVED ✓	Utilization of the `pendingDeposits` mechanism risks losing funds when deposits fail, as these funds are not included in emitted events.
OS-INI-ADV-07	HIGH	RESOLVED ✓	`buildBlockContext` and `buildTxContext` in the EVM context are improperly populated with `nil` values resulting in panics when EVM instructions relying on these values are executed.
OS-INI-ADV-08	HIGH	RESOLVED ✓	An error in the movevm may result in gas undercharging issues due to the consequent setting of `execRes` to a default value, affecting gas metering.
OS-INI-ADV-09	HIGH	RESOLVED ✓	The validator update process in `Opchild` only retrieves the first `MaxValidators`, neglecting the full validator set. This omission may result in unhandled validator changes.
OS-INI-ADV-10	HIGH	RESOLVED ✓	The `ERC20` contracts allows the deployer to burn tokens from any address, creating a centralization risk by enabling token clawbacks.
OS-INI-ADV-11	HIGH	RESOLVED ✓	`AllocateTokens` in the distribution module utilizes `poolSize` as a denominator when calculating validator rewards without checking if `poolSize` is zero.
OS-INI-ADV-12	HIGH	RESOLVED ✓	The seed for `NFT` addresses are derived via a concatenation of the collection name and token ID, separated by a delimiter (`::`). Since both fields may also include `:`, this may result in unintended collisions.
OS-INI-ADV-13	MEDIUM	RESOLVED ✓	The Cosmos gas meter panics on gas exhaustion in EVM precompiled contracts, allowing callee contracts to forcefully abort caller contracts through unhandled panics.

OS-INI-ADV-14	MEDIUM	RESOLVED ✓	`UpdateMetadata` and `UpdateOracleConfig` emit the incorrect event, which may disrupt relayer bots that rely on precise event labels to parse and process on-chain events.
OS-INI-ADV-15	MEDIUM	RESOLVED ✓	Emergency proposals should be handled separately at a later stage in `EndBlocker`, rather than processing them alongside regular active proposals.
OS-INI-ADV-16	MEDIUM	RESOLVED ✓	Collection creators retain the ability to re-enable ungated transfers by accessing the `ConstructorRef`, which undermines transfer restrictions.
OS-INI-ADV-17	MEDIUM	RESOLVED ✓	The `Random` value in `buildBlockContext` is set to zero during `eth_call` execution, rendering the `PREVRANDAO` randomness to be incorrect.
OS-INI-ADV-18	MEDIUM	RESOLVED ✓	The bot fails to listen for and handle `UpdateOracle` events on the child chain, preventing it from relaying oracle data even if the feature is enabled.
OS-INI-ADV-19	MEDIUM	RESOLVED ✓	The voting system has a timing inconsistency which restricts voting and proposal execution at exactly at `voting_end_time` due to a strict comparison check.
OS-INI-ADV-20	LOW	RESOLVED ✓	Certain state variables and mappings for `ERC721` and `ERC20` tokens, are not stored or loaded correctly during the genesis initialization in `nft-transfer` and `minievm` modules.
OS-INI-ADV-21	LOW	RESOLVED ✓	The `is_json` argument in Move within `move_execute_internal` and `move_script_internal` is ignored in Cosmos implementation in `ConvertToSDKMessage`.

OS-INI-ADV-22	LOW	RESOLVED ✓	`AnteHandle` currently rounds up gas prices which results in fee over-distribution and imbalance in yield calculations.
OS-INI-ADV-23	LOW	RESOLVED ✓	The `move -> cosmos` queries in MoveVM bypass caching, which prevents unintended state changes from persisting.
OS-INI-ADV-24	LOW	RESOLVED ✓	`GenerateWithdrawalHash` risks hash collisions due to improper assumptions about the sender field format, as it may contain the `|` separator, especially when utilized with non-Cosmos L2 chains.
OS-INI-ADV-25	LOW	RESOLVED ✓	`slash_unbonding_for_chain` lacks a validation to ensure the fraction parameter does not exceed one.
OS-INI-ADV-26	LOW	RESOLVED ✓	The current implementation of `dex` prevents users from adding liquidity equal to `pool_amount_in`, instead of just disallowing amounts exceeding it.

Slash Evasion due to Untracked Delegation Changes CRITICAL OS-INI-ADV-00

Description

In Initia's implementation of `slash::SlashRedelegation`, the current approach calculates the `slashAmount` based purely on the initial balance of the `redelegation` entry and the given `slashFactor`. However, it fails to account for `redelegation` sequences where a delegator may redelegate their stake from one validator to another and then immediately initiate an `undelegation` (redelegate-to-undelegate sequence).

>_ x/mstaking/keeper/slash.go GO

```go
func (k Keeper) SlashRedelegation(
        ctx context.Context,
        srcValidator types.Validator,
        redelegation types.Redelegation,
        infractionHeight int64,
        slashFactor math.LegacyDec,
) (sdk.Coins, error) {
        [...]
        // perform slashing on all entries within the redelegation
        for _, entry := range redelegation.Entries {
                [...]
                // Calculate slash amount proportional to stake contributing to infraction
                slashAmountDec :=
                    ↪ sdk.NewDecCoinsFromCoins(entry.InitialBalance...).MulDec(slashFactor)
                slashAmount, _ := slashAmountDec.TruncateDecimal()
                totalSlashAmount = totalSlashAmount.Add(slashAmount...)
        [...]
    }
    [...]
}
```

Since the calculation does not consider whether the stake has already been `undelegated`, it fails to slash the correct amount. In the Cosmos SDK's implementation, the slashing process considers both `redelegation` and `unbonding` sequences, effectively reducing the chances for a delegator to escape slashing, ensuring the calculation of slashing is done not just on the balance, but also on the actual shares in the destination validator

Thus, a delegator may move their stake from a validator that is expected to be slashed, to another validator, and immediately undelegate that stake. When the slashing event occurs, the slashing amount calculated may be less than it should be, allowing the delegator to avoid the full impact of the slash.

Remediation

Ensure that Initia's implementation `SlashRedelegation` accounts for unbonding delegations and considers the actual movement of shares, similar to the Cosmos SDK implementation.

Patch

Resolved in 8c31588.

Potential DOS due to Bridge Sequence Deadlock CRITICAL OS-INI-ADV-01

Description

There are multiple vulnerabilities in how the system handles sequencing and finalization of deposit and withdrawal messages. `FinalizeTokenDeposit` enforces a specific message order via `finalizedL1Sequence` to ensure all deposit messages from Layer 1(L!) to Layer 2(L@) are processed sequentially. This strict sequencing is essential for maintaining consistency across the bridge. If the incoming message sequence is older than the current finalized sequence (`req.Sequence < finalizedL1Sequence`), the function ignores it, marking it as a no operation.

>_ *OPinit/x/opchild/keeper/msg_server.go* GO

```go
func (ms MsgServer) FinalizeTokenDeposit(ctx context.Context, req
    ↪ *types.MsgFinalizeTokenDeposit) (*types.MsgFinalizeTokenDepositResponse, error) {
        [...]
        if req.Sequence < finalizedL1Sequence {
                // No op instead of returning an error
                return &types.MsgFinalizeTokenDepositResponse{Result: types.NOOP}, nil
        } else if req.Sequence > finalizedL1Sequence {
                return nil, types.ErrInvalidSequence
        }
    [...]
}
```

However, if the message has a hook execution attached, it may panic and fail to deliver. Thus, the sequence (`L1Sequence`) will get stuck because the sequence number does not advance. This would result in a deadlock, as all subsequent messages will have a sequence number higher than the stuck sequence. As a result, the bridge shuts down effectively because no new deposits or other L1 to L2 messages will proceed.

>_ *opinit/ophost/v1/tx.proto* GO

```go
// MsgFinalizeTokenWithdrawal is a message finalizing funds withdrawal from L2.
message MsgFinalizeTokenWithdrawal {
  option (cosmos.msg.v1.signer) = "receiver";
  option (amino.name)           = "ophost/MsgFinalizeTokenWithdrawal";
  [...]
}
```

Furthermore, `FinalizeTokenWithdrawal` relies on the receiver to finalize the token withdrawal message. Consequently, if the receiver can not sign their withdrawal message, for example, when the receiver is a module, the withdrawal message will become stuck.

Remediation

Handle potential panics in hook execution and advance `L1Sequence` regardless of the result. Also, allow arbitrary actors to deliver the message on behalf of the receiver.

Patch

Resolved in 944722e and PR#106.

EVM Balance Cache Inconsistency CRITICAL OS-INI-ADV-02

Description

Overriding the transfer results in a vulnerability as it breaks the `opBalance` opcode utilized in the EVM. The `opBalance` opcode retrieves account balances by accessing cached values stored within `state_object`. However, the custom transfer function does not update the cached balance within `state_object`. Temporal state-keeping that relies on values cached in `state_object` is broken in the same sense during interactions between EVM and Cosmos.

Remediation

Modify the EVM-to-Cosmos data handling at the `statedb` level ensuring the balance cache reflects all updates and aligns the system's internal state across both platforms.

Patch

Resolved in PR#52.

Discrepancies in Spot Price Calculation CRITICAL OS-INI-ADV-03

Description

There are multiple discrepancies between how the spot price calculation is implemented on the Cosmos side (shown below) and in the `dex` pool. The discrepancy arises because the two implementations handle the multiplication and subsequent division differently. Cosmos operates with decimals from the start ensuring more precision since it allows fractional parts in calculations. It treats the results of `weightQuote * balanceBase` and `weightBase * balanceQuote` as decimals. On the other hand, the `dex` pool in move operates with integers and performs integer multiplication.

>_ initia/x/move/types/connector.go GO

```go
/ GetPoolSpotPrice return quote price in base unit
func GetPoolSpotPrice(
        balanceBase, balanceQuote math.Int,
        weightBase, weightQuote math.LegacyDec,
) math.LegacyDec {
        numerator := weightQuote.MulInt(balanceBase)
        denominator := weightBase.MulInt(balanceQuote)

        return numerator.Quo(denominator)
}
```

Thus, the `dex` pool rounds down to integers (resulting in integer truncation) before converting the result to a decimal format (shown below). This results in loss of precision. Similarly, the cosmos implementation calculates final `Quo` with banker's rounding, while the calculation of spot price in `dex` simply rounds down.

>_ movevm/precompile/modules/initia_stdlib/sources/dex.move RUST

```rust
/// Calculate spot price
public fun get_spot_price(
    pair: Object<Config>,
    base_coin: Object<Metadata>,
): Decimal128 acquires Config, Pool {
   [...]
    decimal128::from_ratio_u64(
        decimal128::mul_u64(&base_weight, quote_pool),
        decimal128::mul_u64(&quote_weight, base_pool),
    )
}
```

Remediation

Align both implementations by utilizing decimal arithmetic for weight and balance multiplications, as done in Cosmos. Additionally, adjust the formula in the Move `DEX` to match the one utilized in the Cosmos implementation.

Patch

Resolved in fea8cc8, f83853e, and PR#110.

Infinite Minting due to Failed Deposits CRITICAL OS-INI-ADV-04

Description

In the current implementation of `msg_server::FinalizeTokenDeposit`, when a deposit transaction fails, the function initiates a refund by triggering a withdrawal event. However, it does not reclaim the previously sent coins from the recipient before this step. In a typical bridging scenario, funds involved in failed transactions should be burned to maintain a balanced supply. This function, however, does not implement such a process, and as a result, the failed deposit is still sent to the recipient account, thus allowing the deposit to exist indefinitely.

>_ *initia/x/move/keeper/staking.go* GO

```go
func (ms MsgServer) FinalizeTokenDeposit(ctx context.Context, req
    ↪ *types.MsgFinalizeTokenDeposit) (*types.MsgFinalizeTokenDepositResponse, error) {
    [...]
    // if the deposit is failed, initate a withdrawal
        if !success {
                l2Sequence, err := ms.IncreaseNextL2Sequence(ctx)
                if err != nil {
                        return nil, err
                }

                err = ms.emitWithdrawEvents(ctx, types.NewMsgInitiateTokenWithdrawal(req.To,
                    ↪ req.From, coin), l2Sequence)
                if err != nil {
                        return nil, err
                }
        }

        return &types.MsgFinalizeTokenDepositResponse{Result: types.SUCCESS}, nil
}
```

This results in an infinite minting bug because the failed bridging message allows the funds to be continuously deposited without corresponding coin destruction (burning), effectively minting new tokens without any underlying asset.

Remediation

Reclaim and burn the coins sent to the recipient in the failed deposit before emitting the refund withdrawal event.

Patch

Resolved in 5856cf8.

Incongruities in Staking Implementation CRITICAL OS-INI-ADV-05

Description

The current `move/staking` implementation contains multiple discrepancies, related to incorrect rounding, evasion of slashing, and the possibility of panicking. For example, in `staking::ShareToAmount` (shown below) the token value is rounded up in the share-to-token conversion instead of rounding down. Furthermore, the current way of deriving the `Unbonding` object in `undelegate` allows users to evade slashing by depositing coins to the `unbonded_store` and inflating the share value. As a result, future `unbonding` operations (where the actual number of shares represented by the `unbonded` tokens) will have zero shares.

>_ *initia/x/move/keeper/staking.go* GO

```go
func (k Keeper) ShareToAmount(ctx context.Context, valAddr sdk.ValAddress, share sdk.DecCoin)
    ↪ (math.Int, error) {
    val, err := k.StakingKeeper.Validator(ctx, valAddr)
    if err != nil {
            return math.ZeroInt(), err
    }
    tokens := val.TokensFromShares(sdk.NewDecCoins(share))
    return tokens.AmountOf(share.Denom).TruncateInt(), nil
}
```

Additionally, in the Cosmos SDk there is a slight possibility that `WithdrawDelegationRewards`, which handles rewards calculation, may result in the chain to halt or panic in certain edge cases. It calls `CalculateDelegationRewards` , which computes the rewards based on the delegator's stake and validator's performance, to ensure that the calculated final stake for the delegator does not exceed their current stake. If this check fails (the calculated stake is larger than the delegator's actual stake), the system panics.

>_ *initia/x/distribution/keeper/delegation.go* GO

```go
func (k Keeper) CalculateDelegationRewards(ctx context.Context, val stakingtypes.ValidatorI, del
    ↪ stakingtypes.DelegationI, endingPeriod uint64) (rewards customtypes.DecPools, err error)
    ↪ {
    [...]
        else {
                panic(fmt.Sprintf("calculated final stake for delegator %s greater than current
                    ↪ stake"+
            [...]       }
    [...]
}
```

Due to the built-in panic recovery mechanisms, panics during transaction processing are generally tolerable. However, `WithdrawDelegationRewards` is called by `WithdrawRewards` which is utilized as part of `BeginBlock` and is not wrapped in a pnic handler. If `WithdrawDelegationRewards` function panics within `BeginBlock`, the panic is not recoverable in the same way as in transaction processing.

>_ *movevm/precompile/modules/initia_stdlib/sources/staking.move* RUST

```
func (k Keeper) WithdrawRewards(ctx context.Context, valAddr sdk.ValAddress) (distrtypes.Pools,
    ↪  error) {
    delModuleAddr := types.GetDelegatorModuleAddress(valAddr)
    if ok, err := k.hasZeroRewards(ctx, valAddr, delModuleAddr); err != nil {
            return nil, err
    } else if ok {
            return nil, nil
    }
    rewardPools, err := k.distrKeeper.WithdrawDelegationRewards(ctx, delModuleAddr, valAddr)
    if err != nil {
            return nil, err
    }
    [...]
}
```

Remediation

Explicitly round down when calling `ShareToAmount` and cap `total_unbonding_amount` to `unbonding_share` to prevent inflation of shares. Additionally, add a panic handler around `WithdrawDelegationRewards` to guard against panics.

Patch

Resolved in 18ed032, daa0f01 and b0ae758.

Incorrect Bridging of Coin Amount CRITICAL OS-INI-ADV-06

Description

`Opinit` holds funds for deposits that failed in `pendingDeposits` and refunds it later on token withdrawal. `msg_server::InitiateTokenWithdrawal` emits an event at the end of a withdrawal operation, logging details like the sender, amount, and token type. However, this event does not include the `pendingDeposits` details. This omission results in `pendingDeposits` becoming orphaned, where the funds are held in a way that the user may not easily track, and as a result, the funds in `pendingDeposits` may go unnoticed and will be lost.

>_ OPinit/x/opchild/keeper/msg_server.go GO

```go
func (ms MsgServer) InitiateTokenWithdrawal(ctx context.Context, req
    ↪ *types.MsgInitiateTokenWithdrawal) (*types.MsgInitiateTokenWithdrawalResponse, error) {
    [...]
        sdkCtx := sdk.UnwrapSDKContext(ctx)
        sdkCtx.EventManager().EmitEvent(sdk.NewEvent(
                types.EventTypeInitiateTokenWithdrawal,
                sdk.NewAttribute(types.AttributeKeyFrom, req.Sender),
                sdk.NewAttribute(types.AttributeKeyTo, req.To),
                sdk.NewAttribute(types.AttributeKeyDenom, coin.Denom),
                sdk.NewAttribute(types.AttributeKeyBaseDenom, baseDenom),
                sdk.NewAttribute(types.AttributeKeyAmount, coin.Amount.String()),
                sdk.NewAttribute(types.AttributeKeyL2Sequence, strconv.FormatUint(l2Sequence,
                    ↪ 10)),
        ))

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

Remediation

Deprecate the `pendingDeposit` logic in favor of immediate refunding when the deposit fails. This process removes any dependency on future transactions (for withdrawals) to access funds.

Patch

Resolved in PR#106.

Lack of EVM Context Structure Population HIGH OS-INI-ADV-07

Description

Due to improper population of the EVM context in `buildBlockContext` and `buildTxContext`, it may result in a nil pointer dereference error when certain EVM instructions are executed. Specifically, the following fields in the context are set to `nil`:

1. `Difficulty`
2. `BaseFee`
3. `BlobBaseFee`
4. `BlobFeeCap`
5. `BlobHashes`
6. `GasPrice`

Remediation

Ensure the values for these fields are set appropriately instead of leaving them as `nil`.

Patch

Partially resolved in e6ad0f9.

Gas Undercharging due to Improper Error Handling HIGH OS-INI-ADV-08

Description

In case of an error in `ExecuteEntryFunction` within the move-vm module, `execRes` will then contain a default value. As a result, the gas consumption will become dysfunctional affecting the gas calculations in `executeEntryFunction`, `executeScript`, and `executeViewFunction` in `handler`. In the above functions gas consumption code (`(gasMeter.ConsumeGas(execRes.Gasutilized, "move runtime"))`) depends on the `Gasutilized` value from `execRes`. However, as `execRes` is a default value, `Gasutilized` will not be properly set.

>_ *movevm/lib.go* GO

```go
// Execute calls a given contract.
// TODO: add params and returns
func (vm *VM) ExecuteEntryFunction(
        [...]
) (types.ExecutionResult, error) {
    [...]
    if err != n {
       return types.ExecutionResult{}, err
    }
    [...]
}
```

Thus, the gas meter will not consume the correct amount of gas. This error in gas consumption calculation may result in undercharging for computational resources, and present a dos opportunity for attackers.

>_ *handler.go* GO

```go
func (k Keeper) executeEntryFunction(
        [...]
) error {
    [...]
    // consume gas first and check error
    gasMeter.ConsumeGas(execRes.Gasutilized, "move runtime")
    if err != nil {
        return err
    }
    [...]
}
```

Remediation

Ensure that even in error scenarios, the consumed amount of gas is properly charged to prevent execution for free.

Patch

Resolved in db5c2db.

Incomplete Validator Set Handling HIGH OS-INI-ADV-09

Description

`Opchild` relies on the host chain to inform it of the set of validators. During validator updates, it iterates through the validators list to process changes, however, the current implementation in `ApplyAndReturnValidatorSetUpdates` only fetches the first `MaxValidators` instead of the entire list. This implies that if there are more than `maxValidators` validators, only the first `maxValidators` will be included in the validator updates processing, while any remaining validators in the set are ignored. Thus, the trailing validators will not get handled properly, and as a result, this inconsistency may result in incorrect consensus power calculations or even halt the chain.

>_ *OPinit/x/opchild/keeper/val_state_change.go* GO

```go
func (k Keeper) ApplyAndReturnValidatorSetUpdates(ctx context.Context) ([]abci.ValidatorUpdate,
    ↪  error) {
    [...]
        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
        }
    [...]
}
```

Remediation

Modify the function call to `GetValidators` to retrieve the entire list of validators, rather than limiting it to `maxValidators`. This ensures that every validator, regardless of position, will be checked for updates.

Patch

Resolved in 308cc3e.

ERC20 clawback by Deployer HIGH OS-INI-ADV-10

Description

There is a potential centralization risk in `burn` within the `minievm` `ERC20` contracts (`erc20::ERC20` and `initia_erc20::InitiaERC20`). `burn` allows deployers to clawback and burn tokens owned by any address. This ability undermines the decentralized and trustless principles of `ERC20` tokens, as users are vulnerable to losing tokens at the owner's discretion.

>_ *minievm/x/evm/contracts/erc20/ERC20.sol* GO

```
function burn(
    address from,
    uint256 amount
) external burnable(from) onlyOwner {
    _burn(from, amount);
}
```

Remediation

Restrict the burn functionality so it only allows burning of tokens that the caller owns.

Patch

Resolved in 5972094.

Unchecked Zero Denominator in Rewards Distribution HIGH OS-INI-ADV-11

Description

The distribution keeper distributes tokens on `BeginBlock` by calling `AllocateTokens`. The vulnerability arises from a lack of validation in `allocation::AllocateTokens` when calculating the fraction of rewards distributed to each validator. Specifically, the calculation of `amountFraction` utilizes `poolSize` as the denominator, but the code does not check if `poolSize` is zero.

>_ x/distribution/keeper/allocation.go GO

```go
func (k Keeper) AllocateTokens(ctx context.Context, totalPreviousPower int64, bondedVotes
    ↪  []abci.VoteInfo) error {
    [...]
    // allocate rewards proportionally to reward power
        for _, rewardWeight := range rewardWeights {
                [...]
                for _, bondedTokens := range bondedTokens[poolDenom] {
                        validator := validators[bondedTokens.ValAddr]

                        amountFraction :=
                            ↪  math.LegacyNewDecFromInt(bondedTokens.Amount).QuoInt(poolSize)
                        reward := poolReward.MulDecTruncate(amountFraction)

                        err = k.AllocateTokensToValidatorPool(ctx, validator, poolDenom, reward)
                        if err != nil {
                                return err
                        }
                        remaining = remaining.Sub(reward)
                }
        }
    [...]
}
```

If `poolSize` is zero, this calculation attempts to divide by zero, which will result in a panic. Since `BeginBlock` is not wrapped in a panic handler, such a panic will not be caught, and as a result, the chain will halt.

Remediation

Add a check to ensure that `poolSize` is greater than zero before calculating `amountFraction`.

Patch

Resolved in e3cb3a4.

NFT Address Collision HIGH OS-INI-ADV-12

Description

The vulnerability is related to the potential for `NFT` object address collisions due to the way `create_nft_seed` and `TokenAddressFromTokenId` generate the seed and address for `NFT` s. `create_nft_seed` generates a unique seed for an `NFT` by concatenating the collection name, a delimiter (`::`), and the `token_id`. `TokenAddressFromTokenId` creates the `NFT` object address utilizing the concatenated string `collectionName + "::" + tokenId`. The delimiter is utilized to separate the collection name from the token ID.

>_ *initia_stdlib/sources/token/nft.move* RUST

```rust
/// Named objects are derived from a seed, the nft's seed is its token_id appended to the
    ↪ collection's name.
public fun create_nft_seed(
    collection: &String,
    token_id: &String
): vector<u8> {
    assert!(
        string::length(token_id) <= MAX_NFT_TOKEN_ID_LENGTH,
        error::out_of_range(ENFT_TOKEN_ID_TOO_LONG)
    );
    let seed = *string::bytes(collection);
    vector::append(&mut seed, b"::");
    vector::append(&mut seed, *string::bytes(token_id));
    seed
}
```

The issue arises because both the collection name and the token ID may include `:` which may be confutilized for a delimiter if utilized in succession `:`. This may result in address collisions if different `NFT` s unintentionally end up having the same address.

Remediation

Prohibit the utilization of the symbol (`:`) in both `CollectionName` and `TokenId`. This will prevent the creation of names or IDs that result in collisions.

Patch

Resolved in 29d9cd7 and 0ddb4cf.

Possibility of Callee Contracts Aborting Caller Contract MEDIUM OS-INI-ADV-13

Description

The `precompiler` EVM contract in the Cosmos SDk utilizes a `gasMeter` to monitor and control gas consumption within transactions. However, since cosmos `gasMeter` panics on gas exhaustion, this provides an opportunity to revert the entire contract call stack regardless of caller contract's attempt to catch and handle callee contract errors. Consequently, this allows a callee contract to perform a denial-of-service attack on the caller contracts by consuming all the gas allocated to it, intentionally triggering a `gasMeter` panic, which may have substantial impacts on certain implementation of contracts.

Remediation

Catch all cosmos side panics and return an explicit error code back to the EVM environment instead of allowing the panic to cascade up the stack.

Patch

Resolved in 9d2a6a3.

Incorrect Event Emission Labels MEDIUM OS-INI-ADV-14

Description

The issue concerns the incorrect event type emitted by the `msg_server::UpdateMetadata` and `msg_server::UpdateOracleConfig`. Both functions currently emit the `EventTypeUpdateBatchInfo` event, which is intended for batch-related updates, and not for updates to metadata or oracle configurations. This will result in confusion regarding the type of update that occurred. Specifically, this may break relayer bots and other monitoring tools that rely on specific event types to parse and handle events correctly.

>_ *OPinit/x/ophost/keeper/msg_server.go* GO

```go
func (ms MsgServer) UpdateMetadata(ctx context.Context, req *types.MsgUpdateMetadata)
    ↪ (*types.MsgUpdateMetadataResponse, error) {
        [...]
        sdk.UnwrapSDKContext(ctx).EventManager().EmitEvent(sdk.NewEvent(
           types.EventTypeUpdateBatchInfo,
           [...]
        ))
        [...]
}

func (ms MsgServer) UpdateOracleConfig(ctx context.Context, req *types.MsgUpdateOracleConfig)
    ↪ (*types.MsgUpdateOracleConfigResponse, error) {
    [...]
        sdk.UnwrapSDKContext(ctx).EventManager().EmitEvent(sdk.NewEvent(
           types.EventTypeUpdateBatchInfo,
           [...]
    ))
        return &types.MsgUpdateOracleConfigResponse{}, nil
}
```

Remediation

Emit accurate, context-specific events that reflect the actual update that is made.

Patch

Resolve in 819e03d and 102f0ad.

Improper Handling Of Emergency Proposals MEDIUM OS-INI-ADV-15

Description

The issue concerns the order in which proposals are handled in `abci::EndBlocker`. Currently, the function iterates over the `ActiveProposalsQueue` first. This queue contains proposals that have moved past the deposit phase and are in the voting phase. For each proposal in this queue, the function retrieves the proposal data and performs tallying to determine if the proposal has passed or failed.

After processing the `ActiveProposalsQueue`, the function handles the `EmergencyProposalsQueue`. These proposals require immediate attention due to their critical nature and are usually processed faster than regular proposals. Similar to the active proposals, the function retrieves each emergency proposal and performs a tally. If the quorum is not reached, the proposal is rescheduled for a re-tally. Handling emergency proposals within the active proposals loop may result in incorrect removal of proposals.

Remediation

Ensure emergency proposals are handled entirely within the separate section of the code that deals with the `EmergencyProposalsQueue`.

Patch

Resolved in 219de26.

Ability to Re-enable Ungated Transfers MEDIUM OS-INI-ADV-16

Description

A collection creator may re-enable ungated transfers of NFTs, even if they were initially disabled, as the creator retains access to the `ConstructorRef`, giving them control over the collection's configuration. The creator may call `enable_ungated_transfer`, effectively re-enabling unrestricted transfer of NFTs at any time. This may be problematic because it allows the creator to bypass any transfer restrictions.

Remediation

Remove `disable_ungated_transfer` and only allow the collection owner to create NFT.

Patch

Resolved in ad551c9.

Faulty PREVRANDAO Randomness MEDIUM OS-INI-ADV-17

Description

In `context::buildBlockContext`, when `eth_call` is called, `Random` is set to zero. As a result, the value returned by `PREVRANDAO` will be zero. `PREVRANDAO` is expected to be a non-zero, pseudo-random value derived from the previous block's header. Setting it to zero breaks contracts that depend on this randomness. It would be more appropriate to return a specific value, as most Ethereum nodes do. This allows contracts that utilize `PREVRANDAO` as a pseudo random number to function properly.

>_ *minievm/x/evm/keeper/context.go* GO

```go
func (k Keeper) buildBlockContext(ctx context.Context) vm.BlockContext {
        sdkCtx := sdk.UnwrapSDKContext(ctx)
        headerHash := sdkCtx.HeaderHash()
        if len(headerHash) == 0 {
                headerHash = make([]byte, 32)
        }
        return vm.BlockContext{
                [...]
                // put header hash to bypass isMerge check in evm
                Random: (*common.Hash)(headerHash),
        }

}
```

Remediation

Utilize the block's header hash as a pseudo-random source and document this for users.

Patch

Resolved in 5c1942a.

Failure to Handle Update Oracle Event MEDIUM OS-INI-ADV-18

Description

In the Opinit chain, there is a parameter to enable oracles on the child chain, which is intended to allow the child chain to receive and process oracle data from the host chain. However, simply enabling this feature does not ensure that the system will start operating correctly without additional steps. The bot responsible for relaying data between the host and child chains is not programmed to listen for `UpdateOracle` events. These events indicate changes in the configuration that should trigger the bot to start or adjust its operation with respect to relaying oracle data.

Remediation

Update the bot to listen for and properly handle the `UpdateOracle` events.

Patch

Resolved in 794ae33.

Timing Logic Inconsistency in Voting MEDIUM OS-INI-ADV-19

Description

There is a timing inconsistency in the voting system in `weight_vote::vote`, which creates a problematic dead moment where neither voting nor proposal execution is possible at the exact `voting_end_time`. This occurs because the logic in the system utilizes strict comparisons when checking the voting end time, which results in a gap where the system does not allow actions at the exact moment the voting period is supposed to end.

>_ *sources/weight_vote.move* GO

```
public entry fun vote(
    account: &signer,
    cycle: u64,
    bridge_ids: vector<u64>,
    weights: vector<BigDecimal>,
) acquires ModuleStore {
    [...]
    let proposal = table::borrow_mut(&mut module_store.proposals, cycle_key);
    assert!(
        time < proposal.voting_end_time,
        error::invalid_state(EVOTING_END),
    );
    [...]
}
```

Remediation

Standardize the comparisons to utilize `<=` in `vote` or `>=` in the proposal execution check to eliminate the timing gap.

Patch

Resolved in a737739.

Unpersisted Genesis States LOW OS-INI-ADV-20

Description

In the current implementation, several important data fields such as `ClassData`, `TokenData`, `DenomPairs`, `ERC721ClassIdsByContractAddr`, `ERC721ContractAddrsByClassId`, `ERC721ClassURIs`, and `ERC20`s which are critical mappings and relationships for tracking NFTs, tokens, and contracts within the system, are not properly persisted during the initialization of the genesis state in `InitGenesis` within `nft-transfer/keeper/genesis` and `minievm/x/evm/keeper/genesis`. `InitGenesis` is responsible for setting up and storing the initial states of modules when the blockchain is first launched. This implies that any operation depending on these states will fail.

Remediation

Ensure to store and load all relevant states to genesis properly.

Patch

Resolved in 263a1f1 and 5ae2b9c.

Inconsistency in Argument Deserialization LOW OS-INI-ADV-21

Description

In Move implementation, `move_execute_internal` and `move_script_internal` accept a `is_json` argument. This boolean flag determines whether the arguments (`args`) provided to these functions should be treated as JSON-encoded. The `is_json` flag is crucial for interpreting the format of the arguments, as it tells the system how to deserialize and handle them. However, On the Cosmos side, in `cosmos_message::ConvertToSDKMessage` , the `is_json` flag is not utilized or passed when converting the Move messages to Cosmos SDK `sdk.Msg` types.

>_ *initia/x/move/types/cosmos_message.go* GO

```go
// ConvertToSDKMessage convert vm CosmosMessage to sdk.Msg
func ConvertToSDKMessage(
        [...]
) (sdk.Msg, error) {
        switch msg := msg.(type) {
        case *vmtypes.CosmosMessage__Move:
                switch msg := msg.Value.(type) {
                case *vmtypes.MoveMessage__Execute:
                        sender, err :=
                            ↪   ac.BytesToString(ConvertVMAddressToSDKAddress(msg.Sender))
                        if err != nil {
                                return nil, err
                        }
                [...]
        }
    [...]
    }
    return nil, ErrNotSupportedCosmosMessage
}
```

For `MoveMessage__Execute` (handling `move_execute_internal`), the arguments are directly passed without checking whether they are JSON or raw byte vectors. Similarly, for `MoveMessage__Script` (handling `move_script_internal`), the `msg.Args` are passed as is, again without considering whether the `is_json` flag was set on the Move side.

Remediation

Ensure `ConvertToSDKMessage` on the Cosmos side takes the `is_json` flag into account when interpreting the arguments.

Patch

Resolved in 2bda6c3.

Incorrect Gas Price Rounding LOW OS-INI-ADV-22

Description

The issue involves the calculation of gas prices in `gas_prices::AnteHandle` (shown below), specifically, utilizing `QuoDec` to divide the total transaction fees by the gas limit. `QuoDec` performs a division and returns a result in `DecCoins`, which are decimal-based coins. However, due to the utilization of bankers rounding, `QuoDec` may produce a gas price that is rounded up. This implies that the calculated gas price is slightly higher than it should be, based on the actual fees paid and gas consumed. The rounding up then has the potential to over-share profits and create deficits on certain parameters.

>_ *x/move/ante/gas_prices.go* GO

```go
// AnteHandle that store gas prices to a context to let the move keeper know tx gas prices.
func (d GasPricesDecorator) AnteHandle(ctx sdk.Context, tx sdk.Tx, simulate bool, next
    ↪ sdk.AnteHandler) (sdk.Context, error) {
    [...]
    if !simulate {
        if gas == 0 {
                return ctx, errors.Wrap(sdkerrors.ErrOutOfGas, "Transaction gas cannot be
                    ↪ zero.")
        }
        // CSR: store a tx gas prices
            ctx = ctx.WithValue(GasPricesContextKey,
                ↪ sdk.NewDecCoinsFromCoins(feeCoins...).QuoDec(math.LegacyNewDec(int64(gas))))
    }
    [...]
    return ctx, nil
}
```

Remediation

Round down the gas price after the division.

Patch

Resolved in c2cdb44.

Uncached Context in MoveVM Queries LOW OS-INI-ADV-23

Description

There is lack of caching in queries from MoveVM to Cosmos. Cosmos utilizes storage wrapping to isolate and control any changes made to the chain state, ensuring that queries do not unintentionally modify state data. However, in the `move -> cosmos` query flow, context caching is not currently in place, implying that data accessed during queries may directly interact with the main chain state. While the current whitelist of allowed queries does not include operations that modify the chain state, there is still a risk if future updates add new types of queries.

Remediation

Implement a cached context for MoveVM queries to maintain state integrity and prevent unintended side effects from future query updates.

Patch

Resolved in aeb6e53.

Risk of Withdrawal Hash Collision LOW

OS-INI-ADV-24

Description

`GenerateWithdrawalHash` generates a unique hash for each withdrawal by concatenating the withdrawal parameters utilizing a specific separator (`|`). However, this approach assumes that the fields, particularly the sender and receiver addresses, will not contain the separator character, which may not always be true in a multi-chain environment. While the current implementation is based on Cosmos, Opinit is designed to be able to work with other L2 chains. Thus it is improper to expect addresses to adhere to any Cosmos specific constraints.

>_ OPinit/x/ophost/types/output.go GO

```go
func GenerateWithdrawalHash(bridgeId uint64, l2Sequence uint64, sender string, receiver string,
    ↪ denom string, amount uint64) [32]byte {
        var withdrawalHash [32]byte
        seed := []byte{}
        seed = binary.BigEndian.AppendUint64(seed, bridgeId)
        seed = binary.BigEndian.AppendUint64(seed, l2Sequence)
        // variable length
        seed = append(seed, sender...) // put utf8 encoded address
        seed = append(seed, Splitter)
        // variable length
        seed = append(seed, receiver...) // put utf8 encoded address
        seed = append(seed, Splitter)
        // variable length
        seed = append(seed, denom...)
        seed = append(seed, Splitter)
        seed = binary.BigEndian.AppendUint64(seed, amount)
        withdrawalHash = sha3.Sum256(seed)
        withdrawalHash = sha3.Sum256(withdrawalHash[:])

        return withdrawalHash
}
```

In `WithdrawalHash`, the digest concatenate fields with `|` as a separator, but since the sender is an L2 address, the assumption where it does not contain `|` is not sound and may result in a hash collision in `GenerateWithdrawalHash`.

Remediation

Ensure to hash individual fields before concatenating them for the final `WithdrawalHash` calculation.

Patch

Resolved in 02f7334.

Missing Fraction Upper Bound Check LOW

OS-INI-ADV-25

Description

`staking::slash_unbonding_for_chain` allows any slashing fraction without any upper limit check. Ideally, the slash fraction should be in the range `[0, 1]` (between 0% and 100%). If the fraction were to exceed one, then the amount calculated for slashing will exceed the total `unbonding_amount`.

Remediation

Add a validation at the beginning of `slash_unbonding_for_chain` to ensure that the fraction does not exceed one.

Patch

Resolved in 7d2dbd5.

Strict Pool Amount Validation LOW OS-INI-ADV-26

Description

In the current implementation of the liquidity provision logic in `dex::calculate_single_asset_provide_liquidity_return_amount`, a validation mechanism is in place to prevent over-depositing and ensures users do not deposit more liquidity than the pool amount (`amount_in` does not exceed `pool_amount_in`). However, the strict comparison check in assertion logic incorrectly prevents users from depositing an amount equal to the current pool balance, even though the design intention was to only prevent amounts exceeding the pool amount.

>_ *precompile/modules/minitia_stdlib/sources/dex.move* MOVE

```move
fun calculate_single_asset_provide_liquidity_return_amount(
    pool: &Pool,
    pair: Object<Config>,
    provide_metadata: Object<Metadata>,
    amount_in: u64
): (u64, u64, bool) acquires Config {
    [...]
    // CONTRACT: cannot provide more than the pool amount to prevent huge price impact
    assert!(
        pool_amount_in > amount_in,
        error::invalid_argument(EPRICE_IMPACT),
    );
    [...]
}
```

Remediation

Modify the comparison operator in the assertion check to utilize `>=` instead of `>` when comparing against `pool_amount_in`, allowing transactions that are exactly equal to the pool amount while still preventing excessive amounts.

Patch

Resolved in 50ed630.

05 — General Findings

Here, we present a discussion of general findings during our audit. While these findings do not present an immediate security impact, they represent anti-patterns and may result in security issues in the future.

ID	Description
OS-INI-SUG-00	There is a discrepancy in how `OnRecvPacket` and `HasPermission` handle the absence of `PermissionedRelayers`, where one allows any relayer while the other allows none.
OS-INI-SUG-01	The current implementation of `LinearTransferRef` allows the original creator to retain control over an object after it is transferred, posing a phishing risk where victims believe they have full ownership, but the creator can reclaim the object at any time.
OS-INI-SUG-02	The `Opinit` bot has a few caveats which may be improved for better robustness and ease of use.
OS-INI-SUG-03	The codebase contains multiple cases of redundancy that should be removed for better maintainability and clarity.
OS-INI-SUG-04	The overall code may be streamlined further to reduce complexity, eliminate redundancy, and enhance readability.
OS-INI-SUG-05	Recommendation for modifying the codebase for improved efficiency.
OS-INI-SUG-06	Suggestions to ensure consistency and proper execution of features in the EVM implementation.

Relayer Permission Inconsistency OS-INI-SUG-00

Description

There is an inconsistency between how permission checks are handled in `ibc_module::OnRecvPacket` and `keeper::HasPermission`. In `OnRecvPacket`, if `PermissionedRelayers.Get` returns an error that is not `collections.ErrNotFound`, it handles it as a generic error (shown below).

>_ x/ibc/perm/ibc_module.go GO

```go
// OnRecvPacket implements the IBCMiddleware interface
func (im IBCMiddleware) OnRecvPacket(
        ctx sdk.Context,
        packet channeltypes.Packet,
        relayer sdk.AccAddress,
) ibcexported.Acknowledgement {
    if permissionedRelayer, err := im.keeper.PermissionedRelayers.Get(ctx,
        ↪ collections.Join(packet.DestinationPort, packet.DestinationChannel)); err != nil &&
        ↪ !errors.Is(err, collections.ErrNotFound) {
            return newEmitErrorAcknowledgement(ctx, err)
    }
    [...]
}
```

However, if the error is `collections.ErrNotFound`, it seems to imply that no specific permissioned relayers are set for that channel. `HasPermission` on the other hand, treats the absence of a `PermissionedRelayers` entry (`collections.ErrNotFound`) as an indication that no relayers are allowed to relay on that channel, returning false. This implies a more restrictive approach where if no permissioned relayers are set, no relayers are allowed to operate.

>_ x/ibc/perm/keeper/keeper.go GO

```go
// HasPermission checks if the relayer has permission to relay packets on the channel.
func (k Keeper) HasPermission(ctx context.Context, portID, channelID string, relayer
    ↪ sdk.AccAddress) (bool, error) {
    permRelayers, err := k.PermissionedRelayers.Get(ctx, collections.Join(portID, channelID))
    if err != nil && errors.Is(err, collections.ErrNotFound) {
            return false, nil
    }
    [...]
}
```

Remediation

Either always allow any relayer or always deny any relayer when no permissions are set consistently across both functions.

Phishing Risk in Ownership OS-INI-SUG-01

Description

The current implementation of `LinearTransferRef` allows original creator to maintain control after object transfer. Objects may be transferred by their creator via `LinearTransferRef` even after ownership transfer. This creates a significant phishing risk where victims may believe they have full ownership of an object after transfer, but the original creator may reclaim it at any time utilizing their stored `TransferRef` .

Remediation

Ensure that once ownership is transferred, no one (including the original creator) may reclaim it without the consent of the new owner.

Opinit Bot Enhancement OS-INI-SUG-02

Description

1. In `process::HandleNewBlock` if a failure occurs such as a transaction getting squeezed out of the `mempool`, the function does not auto-submit failed transactions. Furthermore, an external cronjob is required to help recover from bot failures.

   >_ *opinit-bots/node/broadcaster/process.go* GO

   ```go
   func (b *Broadcaster) HandleNewBlock(block *rpccoretypes.ResultBlock, blockResult
       ↪ *rpccoretypes.ResultBlockResults, latestChainHeight uint64) error {
           [...]
           // check timeout of pending txs
           // @sh-cha: should we rebroadcast pending txs? or rasing monitoring alert?
           if length := b.LenLocalPendingTx(); length > 0 {
                   [...]
           if block.Block.Time.After(pendingTxTime.Add(b.cfg.TxTimeout)) {
                           panic(fmt.Errorf("something wrong, pending txs are not processed
                               ↪ for a long time; current block time: %s, pending tx
                               ↪ processing time: %s", block.Block.Time.UTC().String(),
                               ↪ pendingTxTime.UTC().String()))
                   }
           }
       [...]
   }
   ```

2. `handler::handleChallenge` is not capable of automatically performing challenges against malicious actions, and requires manual monitoring of raised warnings, with admins needing to submit challenges manually.

   >_ *challenger/handler.go* GO

   ```go
   func (c *Challenger) handleChallenge(challenge challengertypes.Challenge) error {
           // TODO: warning log or send to alerting system
           c.logger.Error("challenge", zap.Any("challenge", challenge))

           return nil
   }
   ```

Remediation

Implement an auto-resubmission mechanism for failed transactions within the bot to ensure reliability and reduce dependency on external cronjobs, and automate the challenge response process to mitigate malicious actions or flagged issues without relying on manual intervention.

Code Redundancy OS-INI-SUG-03

Description

1. Replace unnecessary `borrow_global_mut` with `borrow_global` in the following functions in `operator` and `tvl_manager` which are read-only operations:
   (a) `operator::check_operator_permission`
   (b) `operator::get_operator_commission`
   (c) `operator::is_bridge_registered`
   (d) `operator::get_operator_info`
   (e) `tvl_manager::is_snapshot_addable`
2. In `gprc_query::DenomMetadata` fetching the metadata from `k.mk.GetMetadata(ctx, req.Denom)` is redundant as `GetDenomMetaData` already handles the metadata retrieval comprehensively.
3. There is redundant parsing code in `initiateWithdrawalHandler` within `withdraw` which may be removed.

>_ opinit-bots/challenger/child/withdraw.go GO

```go
func (ch *Child) initiateWithdrawalHandler(_ context.Context, args
    ↪ nodetypes.EventHandlerArgs) error {
    [...]
    for _, attr := range args.EventAttributes {
            switch attr.Key {
            case opchildtypes.AttributeKeyL2Sequence:
                    l2Sequence, err = strconv.ParseUint(attr.Value, 10, 64)
                    if err != nil {
                            return err
                    }
        [...]
        }
    }
        return ch.handleInitiateWithdrawal(l2Sequence, from, to, baseDenom, amount)
}
```

4. Remove redundant casting in `remove_vote` (`*tally = *tally - bridge_vp`) and `apply_vote` (`*tally = *tally + bridge_vp`) within `weight_vote`.

Remediation

Remove the redundant code instances highlighted above.

Code Clarity OS-INI-SUG-04

Description

1. Properly register the codecs for `MsgDepositValidatorRewardsPool`, `MsgSetBridgeInfo`, `MsgSpendFeePool` and `MsgUpdateMetadata`.
2. Utilize explicit roundings instead of bankers rounding.
3. In Initias implementation of `depositor`, the `AttributeTypeDepositor` attribute is missing when emitting the `EventTypeProposalDeposit` event.
4. Ensure `MsgInitiateTokenWithdrawalResponse` and `MsgInitiateTokenDepositResponse` populate the sequence field properly in `x/opchild/keeper/msg_server`.
5. Avoid anti-patterns such as updating data while iterating over it, specifically in `merkle` and `batch` in `opinit-bots`.
6. `Accept` in `authz` does not currently support `MsgCancelUnbondingDelegation` as an accepted message type. However, `normalizeAuthzType` does recognize it as a valid `AuthorizationType`. Ensure to resolve this conflict for consistency.

Remediation

Incorporate the suggestion stated above into the codebase.

Code Refactoring OS-INI-SUG-05

Description

1. In Cosmos SDK's `authz` staking module, the design allows an approved actor to choose any validator if the whitelist specified by the approver is empty. However, `authz::Accept` does not support this behavior. If the whitelist is empty, it effectively blocks all validators, which contradicts the intended functionality.

   >_ *initia/x/mstaking/types/authz.go* GO

   ```go
   func (a StakeAuthorization) Accept(ctx context.Context, msg sdk.Msg) (authz.AcceptResponse,
       ↪  error) {
       [...]
       if isValidatorExists {
               return authz.AcceptResponse{}, sdkerrors.ErrUnauthorized.Wrapf("cannot
                   ↪  delegate/undelegate to %s validator", validatorAddress)
           }
       [...]
   }
   ```

2. In `ListenFinalizeBlock`, the block gas limit is not set, resulting in the `GasLimit` to default to `math.MaxUint64`. As a result, incorrect UI is displayed to rpc users.
3. In `minievm` within `jsonrpc/backend/block`, the case where `fullTx` is false is not implemented in `GetBlockByNumber` and `GetBlockByHash`.
4. In `minievm`, native tokens (ETH) are also treated as `ERC20` tokens, creating a unique ETH transfer method that may be unexpected for users familiar with other EVM implementations.
5. `GetStorageRoot` in `statedb` does not actually return the storage root. Since `GetStorageRoot` is currently only utilized to determine whether an account has non-empty storage (and not to fetch an actual Merkle root of the account's storage), its impact is minimal.

Remediation

1. Set `isValidatorExists` to true by default when the `allowedList` is empty, allowing the transaction to proceed with any validator.
2. Set the block gas limit in `ListenFinalizeBlock`.
3. Return the transaction hashes to mirror ETH `rpc` behavior.
4. Unify the implementation or document this difference.
5. Ensure to keep the `stateDB` interface consistent with the expected behavior of returning a storage root in case of future changes in EVM implementation.

Code Maturity

OS-INI-SUG-06

Description

1. `log::ToEthLog` does not set the `blockHash`, `blockNumber`, `transactionHash`, `transactionIndex`, and `logIndex` fields on the `coretypes.Log` object. These fields are significant because they provide context about where the log originated in the blockchain and `ToEthLog` should be modified to accept these additional parameters so that they may be set in `coretypes.Log`.

   >_ *minievm/x/evm/types/log.go* GO

   ```go
   func (l Log) ToEthLog() *coretypes.Log {
           topics := make([]common.Hash, len(l.Topics))
           for i, topic := range l.Topics {
                   topics[i] = common.HexToHash(topic)
           }
           return &coretypes.Log{
                   Address: common.HexToAddress(l.Address),
                   Topics:  topics,
                   Data:    hexutil.MustDecode(l.Data),
           }
   }
   ```

2. The `ParentHash` field in `abci::ListenFinalizeBlock` is set as `common.Hash`, which represents an empty hash. Thus, when fetching the block's parent hash via Ethereum `eth_getBlock*` APIs (such as `eth_getBlockByNumber` and `eth_getBlockByHash`) they will receive this empty `ParentHash` and as a result, will not return the parent block's hash. Ensure to set a valid `ParentHash`.
3. `stateDB::Finalise` does not explicitly burn the balance or remove it from the self destructed EVM accounts. Instead, it only logs this change. Ensure that the balance is fully removed in a self-destructed account.
4. The EVM enforces a call depth limit in `evm::Call` to avoid excessive recursion. Each contract call within a transaction increments the `evm.depth`, which is checked against this limit. If the limit is exceeded, the EVM halts the execution and returns an `ErrDepth` error. Thus, If a contract reaches the call depth limit, any further calls, even simple ones to fetch balances, may trigger this limit, resulting in unexpected failures.

5. `stateDB::GetCodeHash` returns `types.EmptyCodeHash` and not an actual zero hash value when an account does not exist. `GetCodeHash` should return a zero hash value to clearly indicate that an account does not exist.

>_ x/evm/state/statedb.go GO

```go
// GetCodeHash returns the code hash of the account with addr
func (s *StateDB) GetCodeHash(addr common.Address) common.Hash {
        acc := s.getAccount(addr)
        if acc == nil {
                return types.EmptyCodeHash
        }
    [...]
        return common.BytesToHash(cacc.CodeHash)
}
```

6. Ensure the EVM refunded fees are returned to the caller.

Remediation

Implement the above-mentioned suggestions.

A — Vulnerability Rating Scale

We rated our findings according to the following scale. Vulnerabilities have immediate security implications. Informational findings may be found in the General Findings.

CRITICAL

Vulnerabilities that immediately result in a loss of user funds with minimal preconditions.

Examples:

- Misconfigured authority or access control validation.
- Improperly designed economic incentives leading to loss of funds.

HIGH

Vulnerabilities that may result in a loss of user funds but are potentially difficult to exploit.

Examples:

- Loss of funds requiring specific victim interactions.
- Exploitation involving high capital requirement with respect to payout.

MEDIUM

Vulnerabilities that may result in denial of service scenarios or degraded usability.

Examples:

- Computational limit exhaustion through malicious input.
- Forced exceptions in the normal user flow.

LOW

Low probability vulnerabilities, which are still exploitable but require extenuating circumstances or undue risk.

Examples:

- Oracle manipulation with large capital requirements and multiple transactions.

INFO

Best practices to mitigate future security risks. These are classified as general findings.

Examples:

- Explicit assertion of critical internal invariants.
- Improved input validation.

B — Procedure

As part of our standard auditing procedure, we split our analysis into two main sections: design and implementation.

When auditing the design of a program, we aim to ensure that the overall economic architecture is sound in the context of an on-chain program. In other words, there is no way to steal funds or deny service, ignoring any chain-specific quirks. This usually requires a deep understanding of the program's internal interactions, potential game theory implications, and general on-chain execution primitives.

One example of a design vulnerability would be an on-chain oracle that could be manipulated by flash loans or large deposits. Such a design would generally be unsound regardless of which chain the oracle is deployed on.

On the other hand, auditing the program's implementation requires a deep understanding of the chain's execution model. While this varies from chain to chain, some common implementation vulnerabilities include reentrancy, account ownership issues, arithmetic overflows, and rounding bugs.

As a general rule of thumb, implementation vulnerabilities tend to be more "checklist" style. In contrast, design vulnerabilities require a strong understanding of the underlying system and the various interactions: both with the user and cross-program.

As we approach any new target, we strive to comprehensively understand the program first. In our audits, we always approach targets with a team of auditors. This allows us to share thoughts and collaborate, picking up on details that the others may have missed.

While sometimes the line between design and implementation can be blurry, we hope this gives some insight into our auditing procedure and thought process.