Audit Report

Osmosis Smart Accounts

v1.0

July 3, 2024

Table of Contents

License

THIS WORK IS LICENSED UNDER A CREATIVE COMMONS ATTRIBUTION-NODERIVATIVES 4.0 INTERNATIONAL LICENSE.

Disclaimer

THE CONTENT OF THIS AUDIT REPORT IS PROVIDED “AS IS”, WITHOUT REPRESENTATIONS AND WARRANTIES OF ANY KIND.

THE AUTHOR AND HIS EMPLOYER DISCLAIM ANY LIABILITY FOR DAMAGE ARISING OUT OF, OR IN CONNECTION WITH, THIS AUDIT REPORT.

THIS AUDIT REPORT WAS PREPARED EXCLUSIVELY FOR AND IN THE INTEREST OF THE CLIENT AND SHALL NOT CONSTRUE ANY LEGAL RELATIONSHIP TOWARDS THIRD PARTIES. IN PARTICULAR, THE AUTHOR AND HIS EMPLOYER UNDERTAKE NO LIABILITY OR RESPONSIBILITY TOWARDS THIRD PARTIES AND PROVIDE NO WARRANTIES REGARDING THE FACTUAL ACCURACY OR COMPLETENESS OF THE AUDIT REPORT.

FOR THE AVOIDANCE OF DOUBT, NOTHING CONTAINED IN THIS AUDIT REPORT SHALL BE CONSTRUED TO IMPOSE ADDITIONAL OBLIGATIONS ON COMPANY, INCLUDING WITHOUT LIMITATION WARRANTIES OR LIABILITIES.

COPYRIGHT OF THIS REPORT REMAINS WITH THE AUTHOR.

This audit has been performed by

Oak Security GmbH

https://oaksecurity.io/
info@oaksecurity.io

Introduction

Purpose of This Report

Oak Security has been engaged by OSMOSIS FOUNDATION, LTD to perform a security audit of Osmosis Smart Accounts.

The objectives of the audit are as follows:

1. Determine the correct functioning of the protocol, in accordance with the project specification.
2. Determine possible vulnerabilities, which could be exploited by an attacker.
3. Determine smart contract bugs, which might lead to unexpected behavior.
4. Analyze whether best practices have been applied during development.
5. Make recommendations to improve code safety and readability.

This report represents a summary of the findings.

As with any code audit, there is a limit to which vulnerabilities can be found, and unexpected execution paths may still be possible. The author of this report does not guarantee complete coverage (see disclaimer).

Codebase Submitted for the Audit

The audit has been performed on the following target:

Repository	https://github.com/osmosis-labs/osmosis
Commit	`15566f1f2945a16af9243dd376ee58a0d691b66a`
Scope	Only the `x/smart-account` module was in the scope of this audit.
Fixes verified at commit	`04341160b108bf2f699f34248cd1858bb0050470` Note that only fixes to the issues described in this report have been reviewed at this commit. Any further changes such as additional features have not been reviewed.

Methodology

The audit has been performed in the following steps:

1. Gaining an understanding of the code base's intended purpose by reading the available documentation.
2. Automated source code and dependency analysis.
3. Manual line-by-line analysis of the source code for security vulnerabilities and use of best practice guidelines, including but not limited to:
    a. Race condition analysis
    b. Under-/overflow issues
    c. Key management vulnerabilities
4. Report preparation

Functionality Overview

Osmosis Smart Accounts provides a robust and extensible framework for authenticating transactions. The module allows developers to integrate multiple types of authenticators, each with its own rules and conditions for transaction approval.

How to Read This Report

This report classifies the issues found into the following severity categories:

Severity	Description
Critical	A serious and exploitable vulnerability that can lead to loss of funds, unrecoverable locked funds, or catastrophic denial of service.
Major	A vulnerability or bug that can affect the correct functioning of the system, lead to incorrect states or denial of service.
Minor	A violation of common best practices or incorrect usage of primitives, which may not currently have a major impact on security, but may do so in the future or introduce inefficiencies.
Informational	Comments and recommendations of design decisions or potential optimizations, that are not relevant to security. Their application may improve aspects, such as user experience or readability, but is not strictly necessary. This category may also include opinionated recommendations that the project team might not share.

The status of an issue can be one of the following: **Pending,** **Acknowledged**, or **Resolved**.

Note that audits are an important step to improving the security of smart contracts and can find many issues. However, auditing complex codebases has its limits and a remaining risk is present (see disclaimer).

Users of the system should exercise caution. In order to help with the evaluation of the remaining risk, we provide a measure of the following key indicators: **code complexity**, **code readability**, **level of documentation**, and **test coverage**. We include a table with these criteria below.

Note that high complexity or low test coverage does not necessarily equate to a higher risk, although certain bugs are more easily detected in unit testing than in a security audit and vice versa.

Code Quality Criteria

The auditor team assesses the codebase's code quality criteria as follows:

Criteria	Status	Comment
Code complexity	Medium-High	-
Code readability and clarity	Medium-High	Most functions and state variables are documented with clear and concise comments.
Level of documentation	High	Documentation with architectural diagrams and design is available in `x/smart-account/README.md`.
Test coverage	Medium-High	go cover reports a test coverage of `67.2%` in the `x/smart-account` directory, excluding proto-generated files. For more information on how we compute test coverage please see this section in the appendix.

Summary of Findings

No	Description	Severity	Status
1	Excessive transaction fees are incurred	Major	Resolved
2	Inconsistencies in the `AnyOf` authenticator may lead to errors and incorrect authenticated requests	Minor	Acknowledged
3	Unimplemented `GetAuthenticator` query	Informational	Resolved
4	Composite authenticators can be initialized with a single sub-authenticator	Informational	Resolved
5	Open TODOs	Informational	Partially Resolved
6	Misleading documentation	Informational	Resolved
7	Incorrect error logged	Informational	Resolved
8	Lack of support for `secp256r1` and `ed25519` signature verification	Informational	Acknowledged

Detailed Findings

1. Excessive transaction fees are incurred

Severity: Major

In `x/smart-account/ante/ante.go:175-181`, transaction fees are deducted if the message signer matches the fee payer address. This logic becomes problematic when the transaction contains multiple messages, and the messages' signer is the fee payer. The fee would be incorrectly deducted multiple times, causing the fee payer to incur excessive transaction fees.

For comparison, the `deductFeeDecorator` in `classicSignatureVerificationDecorator` only charges a one-time fee for all the messages in a transaction.

Recommendation

We recommend introducing a `feeDeducted` boolean variable in `x/smart-account/ante/ante.go:175` to ensure the transaction fee is only deducted once per transaction. Specifically, the `if` statement can be implemented like `if account.Equals(feePayer) && !feeDeducted`, where the `feeDeducted` variable will be set to `true` after the fee is deducted. This ensures that subsequent messages with the same signer do not incur additional fee deductions.

Status: Resolved

2. Inconsistencies in the `AnyOf` authenticator may lead to errors and incorrect authenticated requests

Severity: Minor

The `AnyOf` authenticator in `x/smart-account/authenticator/any_of.go:85` iterates over all configured authenticators in the `Authenticate` function until any sub-authenticator succeeds when its `Authenticate` function is called. It does the same in the `ConfirmExecution` function, which calls `ConfirmExecution` on all configured authenticators until there is one without an error.

This means there can be the following situations: `authenticatorA` returns success when `Authenticate` is called but returns an error on the `ConfirmExecution` function. Another authenticator, `authenticatorB`, returns an error on `Authenticate` but succeeds for the `ConfirmExecution` function. Although no authenticator returned success for both `Authenticate` and `ConfirmExecution` functions (which is the requirement for requests to authenticate normally), the `AnyOf` authenticator will authenticate such a request successfully.

Other inconsistencies can also arise because of this logic, especially for stateful authenticators. Consider an example configuration `[A, B]`. If `Authenticate` fails on `A` and succeeds on `B`, whereas `ConfirmExecution` succeeds on `A`, there will never be a call to `ConfirmExecution` on `B`. This can be problematic because authenticators may be written under the assumption that there will always be a `ConfirmExecution` call for every successful `Authenticate` call, leading to unintended states if this is not the case.

Similarly, the logic for `AnyOf` authenticator's `Track` function can also lead to undesired states. This function requires that all configured authenticators do not return an error, even if they were not involved in the authentication at all. This violates the following excerpt from the documentation: "The Call `Track()` on all messages step is executed, notifying the authenticators involved."

Recommendation

We recommend refactoring the `AnyOf` authenticator such that it requires at least one authenticator where `Authenticate`, `ConfirmExecution`, and `Track` all succeed. To implement that, `Authenticate` should be called on all authenticators and the success status should be stored. On any successful authenticators, `ConfirmExecution` should be called, and the ones where this call succeeded should be stored again. Finally, `Track` should be called on these authenticators, and if there is at least one successful call, the request will be authenticated.

Status: Acknowledged

The client mentioned this is intended and documented in https://github.com/osmosis-labs/osmosis/pull/8374.

3. Unimplemented `GetAuthenticator` query

Severity: Informational

The `GetAuthenticator` function, defined in `x/smart-account/keeper/query.go:37-57`, that returns the selected authenticator data is not exposed in queries, as seen in `x/smart-account/client/cli/query.go`.

Recommendation

We recommend exposing the `GetAuthenticator` implementation.

Status: Resolved

4. Composite authenticators can be initialized with a single sub-authenticator

Severity: Informational

Composite authenticators, such as `AnyOf` and `AllOf` can be initialized with a single sub-authenticator in `x/smart-account/authenticator/all_of.go:50` and `x/smart-account/authenticator/any_of.go:60`. Although this does not pose any security threats, it is unnecessary if composite authenticators are initialized with a single sub-authenticator.

Recommendation

We recommend ensuring that composite authenticators can only be initialized with at least two sub-authenticators.

Status: Resolved

5. Open TODOs

Severity: Informational

The codebase contains several open TODOs and further comments which may indicate that the codebase is not ready for release in the following locations:

- `app/ante.go:86`
- `x/smart-account/ante/pubkey.go:49`
- `x/smart-account/ante/pubkey.go:59`
- `x/smart-account/authenticator/authentication_request.go:227`
- `x/smart-account/authenticator/message_filter.go:183`
- `x/smart-account/post/post.go:92`
- `x/smart-account/types/msgs.go:40`

Recommendation

We recommend addressing all open comments and TODO items before a production release.

Status: Partially Resolved

6. Misleading documentation

Severity: Informational

There are instances across the codebase of misleading documentation:

- In `x/smart-account/authenticator/signature_authenticator.go:90-9`

`7`, in contrast to the documentation, the `onAuthenticatorAdded` function does not allow for an empty `config`.
- In `x/smart-account/keeper/keeper.go:111-154`, the `GetInitializedAuthenticatorForAccount` function's documentation is inaccurate.

Inaccurate documentation may mislead developers and decrease the maintainability of the codebase.

Recommendation

We recommend updating the documentation to reflect the current implementation.

Status: Resolved

7. Incorrect error logged

Severity: Informational

In `x/smart-account/ante/ante.go:131-139`, if an error occurs in the `GetInitializedAuthenticatorForAccount` function, the `selectedAuthenticator` variable will become `authenticator.InitializedAuthenticator{}`. This will cause the logging of `selectedAuthenticator.Id` in line `138` to be incorrect.

This issue is also present in `x/smart-account/post/post.go:81-89`.

Recommendation

We recommend not mutating the `selectedAuthenticator` variable if an error occurs.

Status: Resolved

8. Lack of support for `secp256r1` and `ed25519` signature verification

Severity: Informational

The current implementation of the `x/smart-account/authenticator/signature_authenticator.go:47` only supports the `secp256k1` curve. Other widely used elliptic curves like `secp256r1` and `ed25519` are unsupported.

Consequently, accounts using these curves will be unable to use the authenticator flow, limiting the system's flexibility and compatibility. For instance, features like one-click trading with `secp256r1` will fail.

Recommendation

We recommend implementing support for `secp256r1` and `ed25519` signature verifications.

Status: Acknowledged

The client states they aim to support `secp256r1` and `ed25519` signature verifications as CosmWasm authenticators in https://github.com/osmosis-labs/passkey-authenticator.

Security Model

The security model has been carefully crafted to delineate the various assets and actors of Osmosis Smart Accounts. They will then be analyzed in a threat model to outline high-level security risks and proposed mitigations.

The purpose of this security model is to recognize and assess potential threats, as well as derive recommendations for mitigations and counter-measures.

There is a limit to which security risks can be identified by constructing a security/threat model. Some risks may remain undetected and may not be covered in the model described below (see disclaimer).

Assets

The following outlines assets that hold significant value to potential attackers or other stakeholders of the system.

Privileged roles

The `SetActiveState` message can only be executed by the circuit breaker governance and circuit breaker controller accounts. These accounts present an attractive target for attackers because they can enable or disable the Osmosis Smart Accounts feature, which may aid the attacker in future exploitation.

Messages

Messages play a pivotal role in the functioning of Osmosis Smart Accounts, facilitating the exchange of crucial information among its various components. These data packages encompass transaction data, requests, and other relevant information necessary for smart accounts like chosen authenticators for authorization. The security of these messages is of paramount importance, as it safeguards against potential threats like tampering and unauthorized access, ensuring the integrity of smart accounts communications.

Native tokens

Native tokens are digital assets established on the Osmosis blockchain. These tokens embody the concept of fungibility, making them highly versatile and extensively utilized across a broad spectrum of applications, such as decentralized finance (DeFi) platforms, token sales, and cross-chain transfers via IBC. These tokens can be held by smart accounts, which may be an attractive target for attackers as they carry real-world value.

Contract-defined tokens

Contract-defined tokens represent tokens created and controlled by a CosmWasm smart contract. For example, a CW20 token contract manages fungible tokens, while a CW721

contract controls non-fungible tokens. Similar to native tokens, these tokens carry real-world value and can be held by smart accounts, which makes them attractive targets for attackers.

Modules of the underlying blockchain

The `x/smart-account` module is responsible for handling smart accounts in the Osmosis chain. Attackers may find this module an attractive target because unexpected vulnerabilities might allow them to arbitrarily modify the storage state, such as removing an authenticator from a victim account. Other trusted modules could also be an attack vector (e.g., module-to-module hooks that call the relevant keeper functions), as attackers may exploit other modules to compromise the `x/smart-account` module.

Private keys controlling external accounts

Regarding externally owned accounts, an entity retains control over the private key associated with the account. The significance of this private key lies in granting full authority over the account, making it an attractive target for attackers.

Access to the private key bestows the attacker with the ability to impersonate the legitimate account owner and execute actions from that account. The account may also have privileged access to other functionality, such as contract migration admin for smart contracts.

Migration authorization

In CosmWasm smart contracts, if there is a defined contract admin, the contract is not immutable. Gaining unauthorized access to act as the contract admin is valuable to an attacker, as it allows them to modify the contract's source code, potentially introducing backdoors or other malicious code.

In the context of Osmosis Smart Accounts, `CosmwasmAuthenticator` relies on CosmWasm contracts for authorization via sudo entry points. The attacker may find migration authorization on such CosmWasm smart contracts attractive because they can migrate the contract, for example, to disable the authorization by forcing the sudo handlers to return a successful response despite the request being malicious.

Liveness of the underlying blockchain and relayers

Deliberately targeting the liveness of underlying blockchains by attacking validators and relayers can present significant value to an attacker. Through such actions, the attacker may exploit opportunities for profit by shorting the native tokens or coercing stakeholders into halting the attack. Osmosis block times are dependent on the validator set coming to a consensus. Several factors may impact this, such as consensus failures that stem from building a block or issues that stem from the ABCI operations that may slow or halt the chain. Furthermore, IBC transfers will fail as Osmosis is not able to send and receive tokens from other chains.

Essential gas price requirements for protocol functionality

If the gas price gets prohibitively expensive, it may lead to a circumstance where the protocol or its regular operation (e.g., normal operation of smart accounts at regular intervals) becomes

economically non-viable or impractical from a business perspective. On the other hand, cheap gas prices allow attackers to spam transactions easily.

Source code of the protocol

The protocol's source code is valuable to the attacker, as they can analyze it for vulnerabilities and exploit them. The attacker may also review the configuration files (e.g., the genesis file, consensus parameters) and the integrations of modules and the base app to discover potential vulnerabilities within them.

This also includes migration scripts, which can be used to add default authenticators to existing accounts to ensure compatibility.

Blockchain consensus

Blockchain consensus protocols establish agreement among network participants regarding transaction validity and order, ensuring trust and security within the network. If an attacker gains control over the Osmosis network’s consensus (either by compromising validator nodes or coordinating collusion), the protocol’s integrity could be compromised. However, the economic feasibility of such an attack grows as the total value locked in the protocol increases.

Signer data

Signer data, comprising account sequence, account number, and chain ID, is crucial for transaction integrity in Osmosis Smart Accounts. The account sequence ensures transactions are processed in order, preventing replay attacks. The account number uniquely identifies each user, while the chain ID distinguishes transactions across different networks. An attacker manipulating this data could replay transactions, leading to unauthorized fund transfers.

Stakeholders/Potential Threat Actors

The following outlines the various stakeholders or potential threat actors that interact with the system.

Circuit breaker governor

The circuit breaker governor is responsible for enabling the smart accounts feature, which disables the circuit breaker system. This is an essential account of the system, as it is the only role that can reactivate the smart accounts after disabling them (e.g., due to a critical vulnerability). It is therefore essential that the corresponding private keys are secured properly and a multi-sig is used. At the time of writing, the circuit breaker governor is controlled by Osmosis Labs contributors as a multisig.

Circuit breaker controllers

Circuit breaker controllers are responsible for disabling the smart accounts feature, which enables the circuit breaker system. Triggering the circuit breaker is a time-sensitive operation because it must be done immediately in response to an exploit. Hence, any of the addresses in `params.CircuitBreakerControllers` is authorized to trigger the circuit breaker in a timely manner to limit the consequences of a potential exploit.

End users

Users interact with the protocol to add or remove authenticators to their smart accounts and verify selected authenticators. Users may or may not be beneficial to the system, depending on their motivation. For example, if an exploit that allows stealing funds from the protocol is found, users may reproduce the exploit for their own benefit, pushing the protocol to insolvency. On the other hand, users may also be the victims of a potential exploit. Users may also alert the circuit breakers of any incidents so they can disable the smart accounts feature, further limiting the damage.

Attackers may find user accounts valuable as they hold assets (e.g., native tokens and privileged access). Users might unknowingly misconfigure their authenticators, allowing attackers to bypass authorization defenses in case their private keys are compromised.

In the case of Osmosis Smart Accounts, users may misconfigure the authenticator for third-party applications, allowing malicious applications to bypass the defense and elevate their privileges, such as performing an unauthorized fund transfer.

Module developers

Developers are responsible for developing the `x/smart-account` and other modules, which means they can introduce code to manipulate their behavior. They are also in charge of fixing vulnerabilities if they occur. A malicious developer may intentionally introduce a backdoor to the system and exploit it for profit while concealing their identity, commonly known as an insider threat.

CosmWasm authenticator developers

CosmWasm Authenticator represents an authentication method in Osmosis Smart Accounts. This method relies on dispatching a sudo message to the specified contract address for authorization purposes. Similar to module developers, CosmWasm Authenticator developers may introduce backdoors to let a malicious request succeed in authentication intentionally. This is easier for CosmWasm contract developers as they can use the contract migration feature to modify the contract's code arbitrarily.

Validators

Nodes are responsible for verifying and validating transactions and adding them to the blockchain to produce blocks. Validators may get involved in blocking or censoring transactions, disrupting the normal functioning of the system. They can also re-order transactions, potentially running front-/back-running or sandwich attacks.

Additionally, validators might collude and equivocate. They represent valuable targets for attackers since compromised validators can be manipulated to carry out equivocation. If a malicious entity is able to control a majority of the validators (either through a compromise or because of a centralized validator set), it can perform an upgrade to a malicious node implementation. This would, for instance, enable them to change balances of accounts arbitrarily.

Governance participants

Governance participants are involved in the decision-making process related to the blockchain's governance, including creating and executing proposals. Depending on the enabled governance proposals, a successful governance vote could result in the ability to:

- Perform actions on behalf of any accounts, such as disabling a smart account's authenticator or disabling the circuit breaker completely.
- Perform a contract migration to modify the source code of a CosmWasm Authenticator contract, such as accepting a malicious request instead of rejecting it.

Supply chain

The software supply chain encompasses various components like libraries, dependencies, and compilers within Osmosis' codebase.

If any of these components are compromised or contain vulnerabilities, it could lead to the introduction of malicious code or potential exploits within the module or overall system.

Third-party applications

Third-party applications may be granted with authenticators from users to facilitate DeFi activities such as one-click trading. These applications may be malicious and try to circumvent the authentication mechanism implemented by the user. Third-party applications can also perform phishing attacks, tricking users into adding malicious authenticators.

Threat Model

Process Applied

The process performed to analyze the system for potential threats and build a comprehensive model is based on the approach first pioneered by Microsoft in 1999 that has developed into the STRIDE model (https://docs.microsoft.com/en-us/previous-versions/commerce-server/ee823878(v=cs.20).

Whilst STRIDE is aimed at traditional software systems, it is generic enough to provide a threat classification suitable for blockchain applications with little adaptation (see below).

The result of the STRIDE classification has then been applied to a risk management matrix with simple countermeasures and mitigations suitable for blockchain applications.

STRIDE Interpretation in the Blockchain Context

STRIDE was first designed for closed software applications in permissioned environments with limited network capabilities. However, the classification provided can be adapted to blockchain systems with small adaptations. The below table highlights a blockchain-centric interpretation of the STRIDE classification:

Spoofing	In a blockchain context, the authenticity of communications is built into the underlying cryptographic public key infrastructure. However, spoofing attack vectors can occur at the off-chain level and within a social engineering paradigm. An example of the former is a Sybil attack where an actor uses multiple cryptographic entities to manipulate a system (wash-trading, auction smart contract manipulation, etc.). The latter usually consists of attackers imitating well-known actors, for instance, the creation of an impersonation token smart contract with a malicious implementation.
Tampering	Similarly to spoofing, tampering of data is usually not directly relevant to blockchain data itself due to cryptographic integrity. It can still occur though, for example through compromised developers of the protocol that have access to deployment keys or through supply chain attacks that manages to inject malicious code or substitutes trusted software that interacts with the blockchain (node software, wallets, libraries).
Repudiation	Repudiation, i.e. the ability of an actor to deny that they have taken action is usually not relevant at the transaction level of blockchains. However, it makes

	sense to maintain this category, since it may apply to additional software used in blockchain applications, such as user-facing web services. An example is the claim of a loss of a private key and hence assets.
Information Disclosure	Information disclosure has to be treated differently at the blockchain layer and the off-chain layer. Since the blockchain state is inherently public in most systems, information leakage here relates to data that is discoverable on the blockchain, even if it should be protected. Predictable random number generation could be classified as such, in addition to simply storing private data on the blockchain. In some cases, information in the mempool (pending/unconfirmed transactions) can be exploited in front-running or sandwich attacks. At the off-chain layer, the leakage of private keys is a good example of operational threat vectors.
Denial of Service	Denial of service threat vectors translates directly to blockchain systems at the infrastructure level. At the smart contract or protocol layer, there are more subtle DoS threats, such as unbounded iterations over data structures that could be exploited to make certain transactions not executable.
Elevated Privileges	Elevated privilege attack vectors directly translate to blockchain services. Faulty authorization at the smart contract level is an example where users might obtain access to functionality that should not be accessible.

STRIDE Classification

The following threat vectors have been identified using the STRIDE classification, grouped by components of the system.

Blockchain infrastructure	
Spoofing	
Tampering	• Backdoors
Repudiation	
Information Disclosure	• Spyware
Denial of Service	
Elevated Privileges	

Messages relay/execution	
Spoofing	
Tampering	• Modifications of existing authenticators by other parties
Repudiation	
Information Disclosure	• Side channel attack/information disclosure in one-click-trading-authenticator
Denial of Service	• Unfulfillable authenticator configurations • Malicious CosmWasm Authenticators • Recursive calls during pattern-matching
Elevated Privileges	• Insufficient authenticator validation

Governance operations	
Spoofing	• Proposals social engineering
Tampering	• Upgrade of CosmWasm Authenticator contracts with malicious or malfunctioning code • Arbitrary modifications for existing authenticators • Upgrading existing accounts with malicious authenticators
Repudiation	

Information Disclosure	
Denial of Service	• Setting invalid circuit breakers
Elevated Privileges	

Externally owned account	
Spoofing	• Lost account
Tampering	• Pharming/phishing/social engineering
Repudiation	• Denial of actions
Information Disclosure	• Private key leakage • Doxxing/identity disclosure
Denial of Service	• DoS of infrastructure • Continuous activation or deactivation of the authenticator • Ineffective circuit breakers
Elevated Privileges	• Compromised private key and multisig accounts

Resource price requirements	
Spoofing	
Tampering	
Repudiation	
Information Disclosure	
Denial of Service	• Suboptimal gas cost configuration
Elevated Privileges	

Mitigation Matrix

The following mitigation matrix describes each of the threat vectors identified in the STRIDE classification above, assigning an impact and likelihood and suggesting countermeasures and mitigation strategies. Countermeasures can be taken to identify and react to a threat, while mitigation strategies prevent a threat or reduce its impact or likelihood.

Blockchain infrastructure

Threat Vector	Impact	Likelihood	Mitigation	Counter-measures
Backdoors Builder team/deployer may implement back doors or malicious code that modifies the smart accounts implementation.	High	Low	Perform strict KYC on builders. Perform internal code reviews and audits to discover potential backdoors in the early stages. Implement time locks on critical operations.	Monitor upgrade attempts and suspicious transactions. Enforce processes on deployment.
Spyware Builder team/deployer may manipulate code to share private keys or users' private information.	High	Low	Perform strict KYC on builders. Perform internal code reviews and audits to discover potential backdoors in the early stages. Implement time locks on critical operations.	Monitor upgrade attempts and suspicious transactions. Enforce processes on deployment.

Messages relay/execution

Threat Vector	Impact	Likelihood	Mitigation	Countermeasures
Modifications of existing authenticators by other parties A smart account may be used by several parties with authenticators implemented as validations. Other parties may try to remove the authenticators to circumvent defenses. This happens when the private key is shared among multiple parties.	Medium	Low	Discourage private key sharing among multiple parties.	-
Side channel attack/information disclosure in one-click-trading-authenticator Users can configure complex authenticators for one-click trading, which may inadvertently leak information about future trades. For example, setting up spending limits in `MessageFilter` reveals users' plans to place limit orders, exposing their trading strategies and intentions.	Low	Low	Document the possibility of information disclosure from authenticators to increase awareness.	-
Unfulfillable authenticator configurations	Medium	Low	Provide detailed documentation on authenticators and examples of usage.	-

A smart account might misconfigure authenticators such that conditions cannot be satisfied, causing transactions to fail.			Document the possibility of authenticators that cannot be satisfied and methods to avoid such configurations.	
Malicious CosmWasm Authenticators CosmWasm Authenticator uses CosmWasm contracts to determine whether a request should fail or succeed, depending on the response returned from the sudo call. If a malicious CosmWasm contract is used as an authenticator, the attacker can maliciously reject legitimate requests and approve malicious requests. Furthermore, the CosmWasm Authenticator contract can prevent itself from being removed, causing a denial of service attack. Since Osmosis uses permissioned CosmWasm contracts deployment, attackers cannot simply upload a malicious contract to execute the attack, as the proposal must be approved by the governance. However, depending on the admin of the CosmWasm smart contract, an external	High	Medium	Thoroughly review code upload proposals to detect malicious backdoor possibilities. Document the possibility of malicious CosmWasm Authenticators and processes to avoid deployments of malicious CosmWasm Authenticators.	Follow discussions and be active in the community.

party may be able to upgrade an approved smart contract to a malicious version.				
Recursive calls during pattern-matching `MessageFilter` authenticators allow users to specify multi-pattern matching, which may consume excessive computation power when matching the JSON patterns recursively.	Low	Low	Implement a mechanism to charge gas accordingly during pattern matchings.	-
Insufficient authenticator validation Authenticators may be configured incorrectly, allowing a bypass (e.g., a missing field in the `MessageFilter` authenticator or using `AnyOf` authenticator instead of `AllOf` authenticator), which allows third-party applications to circumvent the defense. The possibility of this happening increases when using complex authenticators.	High	Medium	Provide detailed documentation on authenticators and examples of good and bad usage. Document the possibility of insufficient authenticator validation and methods to avoid it.	-

Governance operations

Threat Vector	Impact	Likelihood	Mitigation	Countermeasures
Proposals social engineering This threat vector encompasses attackers' effort to manipulate or deceive participants within the governance system, leading to the proposal and approval of malicious or undesirable changes to the system or any of its components.	High	Medium	Educate users and the community about the project and its scope. Offer communication channels that are actively monitored for malicious posts and can be moderated.	Follow discussions and be active in the community.
Upgrade of CosmWasm Authenticator contracts with malicious or malfunctioning code The underlying blockchain could implement governance modules to execute privileged operations on the CosmWasm Authenticator contracts, such as migrations.	High	Low	Educate users and the community about the project and its scope.	Follow proposal discussions and be active in the community.
Arbitrary modifications for existing authenticators The underlying blockchain could implement governance modules to modify existing authenticators, which may cause issues with smart accounts that rely on it, such as denial of service or ineffective authenticators.	Medium	Low	Educate users and the community about the project and its scope.	Follow proposal discussions and be active in the community.
Upgrading existing	High	Low	Internal code	-

accounts with malicious authenticators During Osmosis blockchain upgrades, there may be malicious authenticators being injected into existing accounts via migration scripts and genesis files.			reviews, unit testing, integration tests, automatic software engineering tools, and audits. Maintain strong social consensus among validators to prevent unauthorized modifications and ensure account security.	
Setting invalid circuit breakers When performing an upgrade, an invalid circuit breaker governor or controllers could be configured, causing circuit breakers to fail to work properly.	High	Low	Verify the upgrade procedure thoroughly.	-

Externally owned account

Threat Vector	Impact	Likelihood	Mitigation	Countermeasures
Lost account The attacker claims that they own an account and that access to the private key has been lost.	Low	Low	Implement a clear policy not to refund lost assets or restore privileges.	Enforce policy and strictness.
Pharming/phishing/social engineering The attacker may manipulate users' or development teams' wallets, lure them to malicious front-ends, manipulate DNS records, or use social engineering to trick users/teams into signing manipulated transactions, modifying the authenticator configuration.	Medium	Medium	Educate users and team, protect DNS records, create awareness, offer blacklists with malicious sites, create activity on social channels to build reputable channels, and deploy front-ends on IPFS or other decentralized infrastructure.	Monitor all systems, monitor communities and impersonations/malicious copies of official channels, communicate attempted pharming/phishing/social engineering, and have processes in place to recover from DNS manipulation, attacks on front-ends quickly.
Denial of actions An attacker may deny a performed action in their account, leading to disputes.	Low	Low	Implement a clear policy not to refund lost assets or restore privileges.	Enforce policy and strictness.
Private key leakage Private keys are accidentally shared or logged.	Medium	Medium	Educate users and team, ensure private keys are properly handled in wallet software, and use	Monitor all systems and have a policy in place to

			hardware wallets/air-gapped devices, security keys, and multi-signatures.	rotate keys.
Doxxing/identity disclosure Private data such as the off-chain identity of users disclosed.	Low	Medium	Educate users and team on no storage of identity/sensible data in databases that link identity to account addresses and follow privacy regulations and guidelines.	-
DoS of infrastructure DoS attack on the front end.	Low	Low	Educate users and team, use firewalls, sentry architecture, load balancers, and VPNs.	Monitor infrastructure and have processes to elastically provision and deploy additional resources.
Continuous activation or deactivation of the authenticator The circuit breaker governor could continuously call `SetActiveState` to deactivate the authenticator. Likewise, a malicious circuit breaker controller could continuously deactivate the authenticator. While the system would fall back to the native authenticator, users who rely on specific authenticators are impacted by the	Medium	Low	Perform strict KYC on the team behind the circuit breaker governor and controllers. Implement a strict checklist of requirements for becoming circuit breakers, including background checks and proven track record.	Establish a contingency plan if circuit breaker governor or controllers act dishonestly. Establish a contingency plan that includes a decision-making process for selecting the circuit breakers while maintaining communication with the community.

continuous activation/deactivation.				
Ineffective circuit breakers If the circuit breaker governor decides never to enable the authenticator again, the module will not be usable. Likewise, the circuit breaker controllers may decide not to disable the authenticator when an exploit occurs. Additionally, circuit breaker controllers are required to disable the authenticator as soon as possible when an exploit occurs. If the circuit breaker controllers are based in the same time zones, there may be a delay in triggering the circuit breaker, causing further damage.	High	Low	Perform strict KYC on the team behind the circuit breaker governor and controllers. Implement a strict checklist of requirements for becoming circuit breakers, including background checks with proven track records. Ensure the circuit breaker controllers are situated in different time zones to achieve a fast response time. Rotate accordingly when there are changes to it (e.g., all circuit breakers representatives are at a conference in the same time zone).	Establish an incident response plan with targeted metrics, such as expected response time.
Compromised private key and multisig accounts Private keys may be compromised, which may lead to the compromise of multisig accounts with privileged access.	High	Medium	Educate users and team.	Monitor all systems and have a policy in place to rotate keys.

Resource price requirements

Threat Vector	Impact	Likelihood	Mitigation	Countermeasures
Suboptimal gas cost configuration If the authenticator's authenticate pricing is not correctly established, attackers can perform a denial of service attack on the system at low costs. Additionally, if the `MaximumUnauthenticatedGas` parameter is misconfigured to an excessive value, attackers can spam the network and perform a denial of service attack.	High	Low	Review the static gas cost parameters to ensure sufficient gas amount is charged. Review the `MaximumUnauthenticatedGas` parameter to ensure it is restrictive enough.	Update static gas parameters to be configurable and updatable by the governance. Monitor the network for spam transactions and adjust the limit accordingly.

Appendix

1. Command to compute test coverage excluding proto-generated files

To compute the test coverage accurately without including code generated by protoc-gen-gogo and protoc-gen-grpc-gateway, file extensions that end with `.pb.go` or `.pb.gw.go` need to be excluded.

The following command is executed in the `x/smart-account` directory to compute the test coverage:

```
go test -coverprofile=coverage.out ./... && grep -vE ".pb.go|.pb.gw.go"
coverage.out | go tool cover -func=/dev/stdin | grep total | awk '{print $3}'
```