Audit Report

Abstract Module Bases, Proxy and Manager Contracts

v1.0

May 9, 2023

Table of Contents

License

Disclaimer

This audit has been performed by

Oak Security

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

Introduction

Purpose of This Report

Oak Security has been engaged by Abstract Money Inc to perform a security audit of parts of the Abstract framework implementing account abstraction.

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/AbstractSDK/contracts
Commit	`a93a6c2241a5d6060ca7f4f477277972281d1e09`
Scope	The scope of the audit was limited to the following directories: - `contracts/core/proxy` - `contracts/core/manager` - `packages/abstract-api` - `packages/abstract-app`

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

Abstract is a framework for creating modular account-abstraction-based smart contracts. The core contracts of an account consist of a manager and a proxy contract. The manager contract manages permissions and modules, while the proxy contract holds funds and supports asset value calculations.

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	The architecture is complex due to custom integrations with other contracts.
Code readability and clarity	Medium	Some of the function names are misleading and not descriptive, which is reported as an individual issue.
Level of documentation	Medium-High	Documentation was available at https://docs.abstract.money, along with a recorded code walkthrough video.
Test coverage	Medium-High	A custom testing environment called cw-orchestrator (previously BOOT) is available at https://github.com/AbstractSDK/cw-orchestrator.

Summary of Findings

No	Description	Severity	Status
1	Liquidity token price source does not work with Osmosis pools	Major	Acknowledged
2	API contracts cannot use reply handlers	Major	Resolved
3	Installed modules can execute arbitrary Cosmos messages	Minor	Acknowledged
4	Root users updating module addresses might cause inconsistencies	Minor	Acknowledged
5	Migration to an older version is not prevented and may lead to state inconsistencies	Minor	Resolved
6	Governance type lacking validation	Minor	Resolved
7	`subscriptor_address` is not updatable	Minor	Resolved
8	Queries might fail due to unbounded iterations	Minor	Resolved
9	Add action prevailing over remove action when updating oracle's assets	Minor	Resolved
10	Lack of input validations when upgrading modules	Minor	Resolved
11	Querying configuration does not return subscriptor's address	Informational	Resolved
12	Remove duplicate permission checking	Informational	Resolved
13	Contracts should implement a two-step ownership transfer	Informational	Resolved
14	Incorrect comments	Informational	Resolved
15	Insufficient validation to hinder off-chain attacks	Informational	Resolved
16	Overflow checks not enabled for release profile	Informational	Resolved
17	Misleading message names	Informational	Resolved
18	Sudo messages are not exported	Informational	Resolved
19	Governance type implementation diverges from documentation	Informational	Resolved

Detailed Findings

1. Liquidity token price source does not work with Osmosis pools

Severity: Major

In `packages/abstract-os/src/objects/price_source.rs:222-226`, the liquidity pool token asset must be a `CW20` token, or else an error will occur. This is problematic because Osmosis uses native tokens as liquidity pool tokens (e.g., `gamm/pool/1`), preventing root users from configuring `PriceSource::LiquidityToken` asset value calculation for Osmosis-based pools.

Recommendation

We recommend adding support for native liquidity pool tokens.

Status: Acknowledged

The client states that they acknowledge that the LP tokens will not work on Osmosis at this point. They should be able to add that feature in the future without major breaking changes.

2. API contracts cannot use reply handlers

Severity: Major

In `packages/abstract-api/src/endpoints.rs`, the `reply` entry point is not registered for Abstract APIs. This means the API contract's reply entry point will not be executed even if defined, causing the transaction to revert.

We classify this issue as major because it affects the correct functioning of the system.

Recommendation

We recommend supporting the `reply` entry point for API contracts.

Status: Resolved

3. Installed modules can execute arbitrary Cosmos messages

Severity: Minor

The `proxy` contract allows whitelisted modules to execute arbitrary messages through the `ExecuteMsg::ModuleAction` entry-point handled by the `execute_module_action` function in `contracts/core/proxy/src/commands.rs:19-33`.

As a result, any ill-intended module could perform high-impact actions on the proxy, such as stealing all funds.

Similarly, the `execute_ibc_action` function in lines `37-63` allows arbitrary execution of Cosmos messages over IBC with a similar impact.

Recommendation

We recommend restricting allowed messages to the `ExecuteMsg` type or even further if possible.

Status: Acknowledged

The client states that they acknowledge that the permissions granted to the modules are wide, but the complexity and gas overhead of granular permission control is out of the scope of their V1 launch. Because of this, they opted to make the registration of modules a permissioned action, which allows them to ensure that modules added to their platform are not malicious.

4. Root users updating module addresses might cause inconsistencies

Severity: Minor

The `update_module_addresses` function in `contracts/core/manager/src/commands.rs:46` allows the caller to update the `OS_MODULES` map directly. This privileged function is intended for contract-to-contract interaction in which the caller likely performs lookups and validations on the message parameter. For example, the manager contract calls this function internally through the `register_module` and `set_migrate_msgs_and_context` functions.

In contrast, the root user should not call this function to update the module addresses as it could introduce unintended consequences to their OS, for example, if:

- The new address is not whitelisted in the proxy contract using the `whitelist_dapp_on_proxy` function.
- The old address is not removed from the proxy contract using the `remove_dapp_from_proxy_msg` function.
- New modules that require dependencies to work are directly set instead of using the intended install procedure via the `install_module` function.
- Modules are removed without checking their dependents from the `DEPENDENTS` storage state.
- The root user bypasses the `install_module` function's validation in `contracts/core/manager/src/commands.rs:87-89` by directly calling the module factory contract. This can be achieved by updating any dummy name to the module factory contract address via the `update_module_addresses` function and

using the `ExecOnModule` message to forward the message to the module factory contract to register the module.
- Duplicate addresses are provided within the `to_add` vector which would only cause the last one to be stored.

While most of these examples may not cause significant issues, not automatically removing an old module from the proxy contract would allow it to execute commands via `execute_module_action` even though the `OS_MODULES` storage state is modified.

Recommendation

We recommend preventing the root user from calling the `UpdateModuleAddresses` entry point.

Status: Acknowledged

The client states that while they agree that a user should not call this function light-heartedly, they do not agree to disallow it. Instead, they decided to augment the endpoint to take a `Binary` type that hides the interface from the user, making it difficult to perform the action.

The reason they want the endpoint to remain available is for scenarios where the user installed a badly configured module or where a bug in their dependency resolution locks their ability to add/remove a module. In that case, the client can support the user by providing them with the necessary messages to remove the module and its dependencies from the account manually.

5. Migration to an older version is not prevented and may lead to state inconsistencies

Severity: Minor

The `migrate` entry point for multiple contracts in scope does not prevent a contract migration to an older version. Currently, the `set_contract_version` and `migrate_module_data` functions are only executed when the contract is being migrated to a newer version. In the case that a migration is performed to an older version, no error would be raised, causing the new version of the contract to not be set, and the module data would be silently not migrated to the next version.

- `contracts/core/manager/src/contract.rs:34`
- `contracts/core/proxy/src/contract.rs:65`
- `packages/abstract-app/src/endpoints/migrate.rs:41`

In addition, the `migrate` function does not check if the contract name is indeed the same. In case of a mistake when selecting the code ID, the contract instance could be migrated to a different contract.

We classify this issue as minor since only the root user can cause it.

Recommendation

We recommend enforcing the condition that the contract can only be migrated to a newer version with the same name.

Status: Resolved

6. Governance type lacking validation

Severity: Minor

When the `os-factory` contract instantiates the manager contract, the governance type is converted to a string. Then it is passed to the `os_info` as a string. This conversion is made, but the governance type should only support `Monarchy` and `External`, which are the two variants supported by the `GovernanceDetails` enum in `packages/abstract-os/src/objects/gov_type.rs:9`.

The `set_root_and_gov_type` function in `contracts/core/manager/src/commands.rs:239` allows for this value to be updated directly by the root user as a string. Hence, its value can diverge from the enum variants.

Similarly, the instantiate function also takes this piece of data as a `string` instead of an enum without further validation.

Recommendation

We recommend preserving the `governance_type` as an enum rather than casting it to a string so its variants are properly preserved.

Status: Resolved

7. `subscriptor_address` is not updatable

Severity: Minor

The `manager` contract implements the `subscriptor_address` role to suspend most of the contract's functionality. However, the `update_info` function in `contracts/core/manager/src/commands.rs:407-427` does not allow for this address to be updated.

It is best practice to allow updates of privileged addresses, which is useful in case of operational changes or if a privileged account is compromised.

Recommendation

We recommend adding `subscriptor_address` to the list of updatable configuration details.

Status: Resolved

8. Queries might fail due to unbounded iterations

Severity: Minor

In `packages/abstract-api/src/endpoints/query.rs:48` and `60-64`, all traders and dependencies are returned without pagination. If too many traders or dependencies are registered/installed, the `BaseQueryMsg::Traders` and `BaseQueryMsg::Config` queries might fail due to an out-of-gas error.

Recommendation

We recommend implementing a pagination mechanism for both queries.

Status: Resolved

9. Add action prevailing over remove action when updating oracle's assets

Severity: Minor

The `oracle` package implements the `update_assets` function in `packages/abstract-os/src/objects/oracle.rs:47-72`, which accepts two vectors: One of the assets to be added to the oracle and one of the assets to be removed. The function first removes any assets in the `to_remove` vector and then adds the ones in `to_add` vector.

This approach is not recommended since if the same asset is found in both vectors, it will neither be removed nor added, and no error is raised. This implies that in case of a mistake where the owner accidentally sends an asset in both vectors, they will be unaware of that mistake.

Recommendation

We recommend one of the following approaches:

- Perform the add action before the remove action so in case of a mistake the asset is removed rather than added.
- Check for overlapping assets between `to_add` and `to_remove` and raise an error if any are found.

Status: Resolved

10. Lack of input validations when upgrading modules

Severity: Minor

The `manager` contract's `ExecuteMsg::Upgrade` entry-point is handled by the `upgrade_modules` function in `contracts/core/manager/src/commands.rs:255-277`. This function does not thoroughly validate the `modules` argument vector, which can lead to the following problems:

Firstly, in the case that duplicated modules are provided, two identical migration messages will be forwarded, causing the second migration attempt to fail if the "migrate only if newer" pattern is in place (as recommended by best practices), causing the whole call to fail.

Secondly, the function does not check that the provided vector is empty, allowing for an inefficient execution that will forward the `Callback` message without actual changes.

Thirdly, the whole execution would be short-circuited to the `upgrade_self` function if one of the modules to upgrade is "`abstract:manager`". This is problematic if other modules need to be migrated along because they will be ignored and potentially cause a partial state mutation. For example, if the last module to be migrated is the manager contract, the `MIGRATE_CONTEXT` storage state will store the previous `module_id` and old dependencies while the `update_module_addresses` function will overwrite the old module address to the new one in line `300`. Since the migration is never executed for other modules, the root user can only call the `ExecOnModule` message to the invalid new app address (which should be the old one), and the `handle_callback` and `assert_migrate_requirements` functions will execute the same module ID more than once, consuming extra gas.

Finally, as the provided modules are not checked to have a migrate function, the forwarded migrate message will fail when a non-migratable module is provided, causing the whole call to revert.

We classify this issue as minor because it can only be caused by the root user.

Recommendation

We recommend performing the following validations:

- The vector does not contain duplicate modules.
- The vector must not be empty.
- The vector length must be one when the "`abstract:manager`" module needs to be migrated.

Status: Resolved

11. Querying configuration does not return the subscriptor's address

Severity: Informational

The `manager` contract implements an entry point to query its configuration. This call is managed by the `handle_config_query` function in `contracts/core/manager/src/queries.rs:36-49`.

However, this function does not retrieve the full configuration details: `subscriptor_address` is not retrieved, even though it plays a central role as it is in charge of the contract's suspension.

Recommendation

We recommend adding the `subscriptor_address` to the data returned by the configuration query endpoint.

Status: Resolved

12. Remove duplicate permission check

Severity: Informational

In the `assert_admin` function in `contracts/core/manager/src/commands.rs:233`, there is a validation to assert that the caller is the root user. This is unnecessary as the `execute_update_admin` function performs this check already and is called later in the function.

Recommendation

We recommend removing the admin check in line `233`.

Status: Resolved

13. Contracts should implement a two-step ownership transfer

Severity: Informational

The contracts within the scope of this audit allow the current owner to execute a one-step ownership transfer. While this is common practice, it presents a risk for the ownership of the contract to become lost if the owner transfers ownership to the incorrect address. A two-step ownership transfer will allow the current owner to propose a new owner, and then the account that is proposed as the new owner may call a function that will allow them to claim ownership and actually execute the config update.

Recommendation

We recommend implementing a two-step ownership transfer. The flow can be as follows:

1. The current owner proposes a new owner address that is validated.
2. The new owner account claims ownership, which applies the configuration changes.

Status: Resolved

14. Incorrect comments

Severity: Informational

The `exec_on_module` function contains an incorrect comment in `contracts/core/manager/src/commands.rs:167` that seems to be copied from another function.

A similar situation also occurs in the `set_module_data` function in `packages/abstract-os/src/objects/module_version.rs:48-50`.

Recommendation

We recommend correcting these comments.

Status: Resolved

15. Insufficient validation to hinder off-chain attacks

Severity: Informational

The `manager` contract includes three pieces of information that could potentially be used as part of traditional web exploits such as script injection or phishing schemes: `name`, `description`, and `link`. These could:

1. Contain whitespace intended to trick users
2. Contain code that may be executed in a user's browser
3. Contain links to external resources that may be malicious

4. Contain insecure links

Recommendation

We recommend performing a best-effort validation on the affected fields, for instance disallowing leading or trailing whitespace and non-printable characters, disallowing typical code syntax such as ", ', =, >, <, and only allowing whitelisted links to using HTTPS. However, we acknowledge that proper validation may not be possible on the smart contract side.

Status: Resolved

16. Overflow checks not enabled for release profile

Severity: Informational

None of the packages and contracts enabled `overflow-checks` for the release profile. While enabled implicitly through the workspace manifest, future refactoring might break this assumption.

Recommendation

We recommend enabling overflow checks in all packages, including those that do not currently perform calculations, to prevent unintended consequences if changes are added in future releases or during refactoring. Note that enabling overflow checks in packages other than the workspace manifest will lead to compiler warnings.

Status: Resolved

17. Misleading message names

Severity: Informational

The `SuspendOs` and `EnableIBC` functions in the manager contract are both potentially misleading function names. Based on their names, the functions seem declarative of one specific action, but each of these functions actually supports enabling and disabling functionality. For example, a call to `EnableIBC` with a status of false actually disables the IBC functionality.

Recommendation

We recommend updating the names of these functions to reflect their functionality better.

Status: Resolved

18. Sudo messages are not exported

Severity: Informational

In `packages/abstract-api/src/endpoints.rs` and `packages/abstract-app/src/endpoints.rs`, sudo messages are not exported as valid entry points. If governance decides to vote and execute arbitrary messages on the API or app contracts, they will fail.

Recommendation

We recommend exporting the sudo message for API and app contracts. Alternatively, we recommend documenting the reason why sudo entry points are not exposed.

Status: Resolved

19. Governance type implementation diverges from documentation

Severity: Informational

The governance documentation page lists three governance types: monarchy, multisig, and token. However, the `GovernanceDetails` enum in `packages/abstract-os/src/objects/gov_type.rs:9-22` only implements two governance types: monarchy and multisig.

Recommendation

We recommend adding the token governance type to the `GovernanceDetails` enum or removing the token governance type from the documentation.

Status: Resolved