MAVEN
by MSafe

Audit

Presented by:

OtterSec contact@osec.io

Akash Gurugunti sud0u53r.ak@osec.io

Marco Ilardi goten@osec.io

Robert Chen r@osec.io

Contents

01 | Executive Summary

Overview

MSafe engaged OtterSec to perform an assessment of the `mavencore` program for Maven. This assessment was conducted between April 9th and April 23rd, 2023. For more information on our auditing methodology, see Appendix B.

Key Findings

Over the course of this audit engagement, we produced 11 findings total.

In particular, we discovered issues around arbitrary delete of orders (OS-MSF-ADV-00), and a lack of checks for forbidden IDs (OS-MSF-ADV-01).

We also made recommendations around gas optimizations (OS-MSF-SUG-06,OS-MSF-SUG-02), absence of checks (OS-MSF-SUG-05,OS-MSF-SUG-07), and the improper use of constants (OS-MSF-SUG-03).

02 | Scope

The source code was delivered to us in a git repository at github.com/Momentum-Safe/MavenCore. This audit was performed against commit 233da9e.

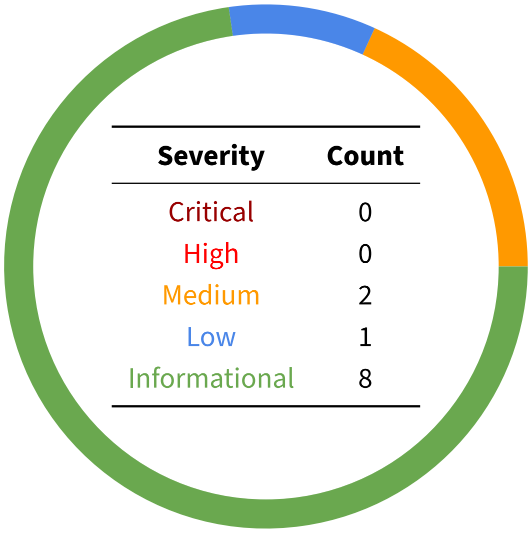
A brief description of the programs is as follows.

Name	Description
mavencore	Maven Core is a platform for managing multiple permissioned assets with various levels of operations such as `Admin`, `Coin`, `Object`, and `Timelock` operations, as well as a `Recovery` mechanism. It features a vault that gives allowance to users with timely spending limits and an emergency pause feature that stops all operations in the event of protocol issues.

03 | Findings

Overall, we reported 11 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 don't have an immediate impact but will help mitigate future vulnerabilities.



Severity	Count
Critical	0
High	0
Medium	2
Low	1
Informational	8

04 | Vulnerabilities

ID	Severity	Status	Description
OS-MSF-ADV-00	Medium	Resolved	`execute_object_operation` does not check the correct `ASSET`, causing an arbitrary delete of orders.
OS-MSF-ADV-01	Medium	Resolved	`update_admin_policy` allows forbidden IDs.
OS-MSF-ADV-02	Low	Resolved	Slicing a vector should abort if end is less than `start` to avoid unintended behavior.

OS-MSF-ADV-00 [med]| Arbitrary Delete Of Orders

Description

In `maven.move`, any user can call `execute_object_operation` with the incorrect ASSET generic and pop the order from `execution` without properly executing it, as the `precheck` fails and returns `false`. This allows anyone to stop others from executing the object operations.

maven.move RUST

```rust
public entry fun execute_object_operation<ASSET: key + store>(
    global_pause: &Pause,
    maven: &mut Maven,
) {
    pause::ensure_not_paused(global_pause);
    let (order, op_sn) = order::pop_min_order(&mut maven.order_book);
    [...]
    while (i < op_count) {
        [...]
        let (success, error_code) =
 ↪  vault::execute_object_precheck<ASSET>(&maven.vault, &op);
        [...]
    };

    if (successes) {
        [...]
    };
}
```

Remediation

Implement the same checks as in `execute_coin_precheck`, where if the generic T does not match the `asset_key`, the function aborts instead of continuing with the operation.

Patch

Fixed in cac0a8a.

OS-MSF-ADV-01 [med] | Lack Of Check For Forbidden IDs

Description

`update_admin_policy` in `permission.move` does not verify whether the permission ID provided is non-forbidden before allowing the setting of the admin policy. Therefore, it is possible to set the forbidden permission ID as the admin policy, resulting in a denial of service scenario whereby all admin operations fail. It is impossible to change the admin policy again.

permission.move RUST

```rust
fun update_admin_policy(
    book: &mut PermissionBook,
    policy: String,
    permission_id: PermissionID,
) {
    validate_admin_policy_name(policy);
    assert!(exists_permission(book, permission_id),
 ↪  E_PERMISSION_NOT_EXIST);
    if (table::contains(&book.admin_policy, policy)) {
        let old_permission_id = table::borrow(&book.admin_policy,
 ↪  policy);
        decrease_permission_ref(book, *old_permission_id);
    };
    utils::update_table(&mut book.admin_policy, policy,
 ↪  permission_id);
    increase_permission_ref(book, permission_id);
    event::emit(UpdateAdminPolicyEvent { policy, permission_id });
}
```

Remediation

Update `update_admin_policy` to include a check for forbidden permission IDs.

Patch

Fixed in 908d6c1.

OS-MSF-ADV-02 [low] | Insert Range Check In Utils

Description

In `utils.move`, `vector_slice` should return the subslice of the vector, starting at the `start` index and ending at the end index. However, there is no check to ensure that end is higher than `start`; in this case, the function returns an empty vector.

utils.move RUST

```
public fun vector_slice<T: copy>(vec: &vector<T>, start: u64, end:
  ↪  u64): vector<T> {
     let vec_slice = vector::empty<T>();
     while (start < end) {
         vector::push_back(&mut vec_slice, *vector::borrow(vec,
  ↪  start));
         start = start + 1;
     };
     vec_slice
 }
```

Remediation

Ensure that the function aborts if `end < start` rather than returning an empty array.

Patch

Fixed in 9e952dd.

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 antipatterns and could lead to security issues in the future.

ID	Description
OS-MSF-SUG-00	Two error codes in `maven.move` have the same value.
OS-MSF-SUG-01	`get_struct_name` is not used.
OS-MSF-SUG-02	Utilize functions that require less gas for revoking roles.
OS-MSF-SUG-03	Constant usage in `permission.move` may lead to issues.
OS-MSF-SUG-04	Lack of checks for `expiration_ms`.
OS-MSF-SUG-05	Misspelled variable present in `maven.move`.
OS-MSF-SUG-06	Two loops that iterate over the same vector may be combined.
OS-MSF-SUG-07	Absence of a check for zero weight value.

OS-MSF-SUG-00 | Error Code Conflict

Description

In `maven.move`, `E_EXCEED_COIN_OPERATION_LIMIT` and `E_EXCEED_RECOVERY_OPERATION_LIMIT` have the same value.

maven.move RUST

```rust
    const E_COIN_OPEARTION_SIZE_MUST_ONE: u64 = 1009;
    const E_EXCEED_COIN_OPERATION_LIMIT: u64 = 1010;
    const E_EXCEED_RECOVERY_OPERATION_LIMIT: u64 = 1010;
    const E_NOT_META_OPERATION: u64 = 1011;
```

Remediation

Renumber the value of error codes.

Patch

Fixed in 025c786.

OS-MSF-SUG-01 | Remove Unused Function

Description

`get_struct_name` in `utils.move` will abort if the type T is a ground type. It is also not used in the codebase.

utils.move RUST

```rust
    public fun get_struct_name<T>(): String {
        let type = type_name::get<T>();
        let type_string = type_name::borrow_string(&type);

        // Starts after address and a double colon: `<addr as HEX>::`
        let i = address::length() * 2 + 2;
        let str_bytes = ascii::as_bytes(type_string);

        // skip module name
        loop {
            let char = vector::borrow(str_bytes, i);
            if (char != &ASCII_COLON) {
                i = i + 1;
            } else {
                break
            }
        };

        // skip '::'
        let struct_name = vector_slice(str_bytes, i + 2,
     ↪  vector::length(str_bytes));
        ascii::string(struct_name)
    }
```

Remediation

Remove the unused function.

Patch

Fixed in d469db1.

OS-MSF-SUG-02 | Gas Optimization

Description

In `role.move`'s `revoke_role_from_signer`, it may be more efficient to use `vector::swap_remove` instead of `vector::remove` when removing role IDs from `signer.roles`, as the order of the IDs in the vector is not essential. This may help reduce gas consumption.

role.move RUST

```
fun revoke_role_from_signer(role_book: &mut RoleBook, signer_id:
↪   address, role_id: RoleID) {
    assert!((!is_void_role(role_id)) && (!is_void_signer(signer_id)),
↪   E_EDIT_RESERVED);
    let signer = get_signer_mut(role_book, signer_id);
    assert!(has_role(signer, role_id), E_SIGNER_NOT_HAS_ROLE);
    let (_, index) = vector::index_of(&signer.roles, &role_id);
    vector::remove(&mut signer.roles, index);
    event::emit(RevokeRoleEvent { address: signer_id, role_id });

    let role = get_role_mut(role_book, role_id);
    reference::decrease(&mut role.signer_ref);
    if (reference::is_zero(&role.signer_ref)) {
        assert!(reference::is_zero(&role.permission_ref),
↪   E_ROLE_STILL_BE_USED);
    };
}
```

Remediation

Replace the reference of `vector::remove` with `vector::swap_remove`.

Patch

Fixed in d469db1.

OS-MSF-SUG-03 | Mismatched Constant Usage

Description

In `permission.move`, `get_admin_permission` and `get_coin_permission` uses a default policy name `RESERVED_DEFAULT_PERMISSION_NAME` while `init_default_policy` uses the constant `DEFAULT_POLICY_NAME`. This will cause issues if the values of those constants are changed in the future.

Remediation

Utilize the constant `DEFAULT_POLICY_NAME` in `get_admin_permission` and `get_coin_permission`.

Patch

Fixed in d469db1.

OS-MSF-SUG-04 | Allowed Execution Of Expired Orders

Description

In `maven.move`, `execute_permission_recovery` allows the users to execute orders, even if they have expired.

maven.move RUST

```rust
public entry fun execute_permission_recovery(
    global_pause: &Pause,
    maven: &mut Maven,
    op_id: u64,
    sys_clock: &Clock
) {
    pause::ensure_not_paused(global_pause);
    let op = order_timelock::pop_executable_order(&mut
 ↪  maven.order_book_timelock, op_id, sys_clock);
    let permission_book = &mut maven.permission_book;
    permission::execute_recovery(permission_book, &op);
    event::emit(ExecutePermissionRecoveryEvent {
        maven: object::id(maven),
        op_id,
    });
}
```

Remediation

Verify that the current timestamp is lower than the `expiration_ms` in `time_lock::destroy` to prevent the execution of expired orders.

Patch

Fixed in cb70ba6.

OS-MSF-SUG-05 | Misspelled Variable In Maven.move

Description

A misspelled variable `weight_apporve` is present in `maven.move`. This variable is used in the following functions:

1. `start_permission_recovery`.
2. `weight_of_permission`.
3. `result_of_permission`.

Remediation

Correct the variable name.

Patch

Fixed in b95b733.

OS-MSF-SUG-06 | Presence Of Useless Loop

Description

The current implementation of `execute_admin_operation` in `maven.move` contains a redundancy issue. The first `while` loop stores approved operations in the `operations` vector, while the second loop iterates over the same vector again. This results in an unnecessary iteration and may be optimized to improve efficiency and reduce gas usage.

maven.move RUST

```rust
public entry fun execute_admin_operation(
    global_pause: &Pause,
    maven: &mut Maven,
    sys_clock: &Clock,
    ctx: &mut TxContext
) {
    [...]
    // check all permission
    let approved_set = vec_set::empty<PermissionID>();
    let i = 0;
    while (i < op_count) {
        [...]
    };
    // execute all operations
    let i = 0;
    while (i < op_count) {
        [...]
    };
}
```

Remediation

Remove the second loop and execute the approved operations directly in the first loop.

Patch

Fixed in 06f947c.

OS-MSF-SUG-07 | Missing Signer Count Check In Permission

Description

In `permission.move`, `has_enough_weight` does not ensure that the value of `signer_count` is greater than zero.

maven.move RUST

```
public fun has_enough_weight(role_book: &RoleBook, permission:
 ↪  &Permission, weight: u64): bool {
    let signers_count = role::get_signer_amount_by_role(role_book,
 ↪  permission.approver);
    let threshold = math::min(permission.threshold, signers_count);
    weight >= threshold
}
```

Remediation

Insert an `assert` statement to ensure that `signers_count` is always greater than zero.

Patch

Fixed in 4204eb4.

A | Vulnerability Rating Scale

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

Critical	Vulnerabilities that immediately lead to 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 could lead to 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 could lead to denial of service scenarios or degraded usability. Examples: • Malicious input that causes computational limit exhaustion • Forced exceptions in normal user flow
Low	Low probability vulnerabilities which could still be exploitable but require extenuating circumstances or undue risk. Examples: • Oracle manipulation with large capital requirements and multiple transactions
Informational	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 implementation of the program 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 sum, 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 get a comprehensive understanding of 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 other 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.