our shielding · Your smart contracts, our shielding · Your smart c

shieldify

Rewardy

SECURITY REVIEW

Date: 11 November 2025

CONTENTS

1. About Shieldify

Positioned as the first hybrid Web3 Security company, Shieldify shakes things up with a unique subscription-based auditing model that entitles the customer to unlimited audits within its duration, as well as top-notch service quality thanks to a disruptive 6-layered security approach. The company works with very well-established researchers in the space and have secured multiple millions in TVL across protocols, also can audit codebases written in Solidity, Vyper, Rust, Cairo, Move and Go.

Learn more about us at shieldify.org.

2. Disclaimer

This security review does not guarantee bulletproof protection against a hack or exploit. Smart contracts are a novel technological feat with many known and unknown risks. The protocol, which this report is intended for, indemnifies Shieldify Security against any responsibility for any misbehavior, bugs, or exploits affecting the audited code during any part of the project's life cycle. It is also pivotal to acknowledge that modifications made to the audited code, including fixes for the issues described in this report, may introduce new problems and necessitate additional auditing.

3. About Rewardy

Rewardy Wallet is a blockchain-based WEB3 entertainment wallet that allows users to earn points through community activities. Users may manage their assets easily and securely with no gas fees, and earn points for being active in 20+ global communities.

More docs - Vision & Features

Summary

- **Token Symbol:** `RWD`
- **Token Name:** `Rewardy`
- **Decimals:** `8`
- **Initial Supply:** `3_000_000_000_000_000_000 (u128)`
- **Metadata:** Includes icon URL and website URL
- **Owner:** Only the **root object owner (deployer address)** can perform sensitive actions such as mint, burn, pause, and ownership transfer

Architecture Overview

```
flowchart TD
    A[Module Deployer (rewardy_coin_factory_address)] -->|init_module| B[
       Create Named Object (ASSET_SYMBOL)]
    B --> C[primary_fungible_store::
       create_primary_store_enabled_fungible_asset]
    C --> D[Metadata Object (icon/url/decimals)]
    C --> E[MintRef / BurnRef / TransferRef / ExtendRef generation]
    E --> F[Store RewardyCoin resource (key)]
    F --> G[Initialize paused=false]
    B --> H[dispatchable_fungible_asset::register_dispatch_functions (
       deposit/withdraw)]
    subgraph Runtime
      I[User Primary Store] <--> |deposit/withdraw| J[FungibleAsset]
    end
```

- The **RewardyCoin resource** holds the refs `(`mint_ref`, `burn_ref`, `transfer_ref`, `extend_ref`)` and the `paused` flag.
- **Ownership verification** ensures only the root object owner (module deployer) can perform admin actions.
- When `paused == true`, all mint/burn/deposit/withdraw paths are restricted.

Key Features

- Standard **Fungible Asset** creation compliant with Aptos FA framework
- Owner-only **minting** and **burning**
- Dispatchable **deposit/withdraw** via registered entry functions
- **Pause/Unpause** functionality for risk control
- **Ownership transfer** support
- Rich **metadata** (token icon & project website)

Security & Permission Model

- Only the **root object owner** can perform:
  - `init_module()`, `set_puased()`, `mint()`, `burn()`, `change_owner()`
- When `paused == true`, the following are **restricted**:
  - `mint()`, `burn()`, `deposit()`, `withdraw()`
- `authorized_borrow_refs` ensures the signer matches the root owner via `object::root_owner(asset)`.
- Verify URLs, symbols, and decimals before deployment as they are hardcoded.

4. Risk Classification

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

4.1 Impact

- **High** - results in a significant risk for the protocol's overall well-being. Affects all or most users
- **Medium** - results in a non-critical risk for the protocol affects all or only a subset of users, but is still unacceptable
- **Low** - losses will be limited but bearable - and covers vectors similar to griefing attacks that can be easily repaired

4.2 Likelihood

- **High** - almost certain to happen and highly lucrative for execution by malicious actors
- **Medium** - still relatively likely, although only conditionally possible
- **Low** - requires a unique set of circumstances and poses non-lucrative cost-of-execution to rewards ratio for the actor

5. Security Review Summary

The security review lasted 9 days, with a total of 144 hours dedicated to the audit by two researchers from the Shieldify team.

Overall, the code is well-written. The audit report contributed by identifying three Medium and five Low severity issues. They're mainly related to blocklisted recipient credits forcibly redirected to the admin, hardcoded admin address and the lack of an emergency recovery mechanism.

The Rewardy team has done a great job with their test suite and provided exceptional support, and promptly implemented all of the suggested recommendations from the Shieldify researchers.

5.1 Protocol Summary

Project Name	**Rewardy - RWD_BRIDGE, RWD_APTOS**
Repository #1	RWD_BRIDGE
Repository #2	RWD_APTOS
Type of Project	OFT, Bridge, Ownership
Security Review Timeline	9 days
Review Commit Hash #1	c68e1d39d98b2e819021384771a0bf4722ec5947
Review Commit Hash #2	76412ed7232407d54c3e2b130eb890188347a010
Fixes Review Commit Hash #1	6fee4995a716b863616af94ee53bcf72244d4fd2
Fixes Review Commit Hash #2	76412ed7232407d54c3e2b130eb890188347a010

5.2 Scope

The following smart contracts were in the scope of the security review:

File	nLOC
RWD_APTOS/rwd.coin.factory.move	146
RWD_BRIDGE/contracts/MyOFT.sol	11
RWD_BRIDGE/sources/oft_implementation/oft_adapter_fa.move	209
RWD_BRIDGE/sources/shared_oapp/oapp_core.move	306
RWD_BRIDGE/sources/shared_oapp/oapp_receive.move	69
RWD_BRIDGE/sources/shared_oapp/oapp_store.move	69
RWD_BRIDGE/sources/shared_oft/oft.move	330
RWD_BRIDGE/sources/shared_oft/oft_core.move	179
RWD_BRIDGE/sources/shared_oft/oft_impl_config.move	311
RWD_BRIDGE/sources/shared_oft/oft_store.move	32
Total	**1662**

6. Findings Summary

The following number of issues have been identified, sorted by their severity:

- **Medium** issues: 3
- **Low** issues: 5
- **Info** issues: 2

ID	Title	Severity	Status
[M-01]	Blocklisted Recipient Credits Are Forcibly Redirected to the Admin, Enabling Custodial Capture of Inbound Transfers	Medium	Acknowledged
[M-02]	Native Token Always Deposited to Hardcoded Admin Address	Medium	Fixed
[M-03]	No Emergency Recovery Mechanism for Escrowed Funds When Remote Pathway Fails	Medium	Acknowledged
[L-01]	Admin Can Renounce with Active Blocklist	Low	Acknowledged
[L-02]	Maximum Fee Cap Allows 100% Fees	Low	Acknowledged
[L-03]	Delegate Remains Powerful After Admin Renounces	Low	Acknowledged
[L-04]	Rewardy Burn Fails if the `from` Address Lacks a Primary Store	Low	Acknowledged
[L-05]	Removing a Peer Does Not Deregister the Receive Pathway	Low	Acknowledged
[I-01]	Application Functions Can Be Called Before Initialization Completes	Info	Acknowledged
[I-02]	Duplicate Error Codes Reduce Debugging Clarity	Info	Acknowledged

7. Findings

[M-01] Blocklisted Recipient Credits Are Forcibly Redirected to the Admin, Enabling Custodial Capture of Inbound Transfers

Severity

Medium Risk

Description

The receiveside flow in the Aptos adapter uses `credit(to, amount_ld, src_eid, receive_value)` to unlock funds from the escrow object and move them into the recipient's primary store. Immediately before the `primary_fungible_store::transfer()` call, the recipient address is obtained by invoking `redirect_to_admin_if_blocklisted()`.

That helper first evaluates `is_blocklisted(wallet)`, which returns `true` only if `Config.blocklist_enabled` is `true` and the address appears in `Config.blocklist`. If the address is blocklisted, `redirect_to_admin_if_blocklisted` emits a `BlockedAmountRedirected` event and returns `get_admin()`.

Otherwise, it returns the original recipient. Because credit uses that returned address as the transfer target and does not revert on a blocklisted recipient, any inbound credit for a blocklisted recipient is deterministically diverted to the current admin account. Outbound sends are separately guarded by `assert_not_blocklisted` in `debit_fungible_asset`, so a blocklisted sender cannot initiate a debit, but a nonblocklisted sender can still send to a blocklisted recipient and have the funds delivered to the admin.

There is no additional slippage or dust check at this stage that would prevent delivery - the diversion happens unconditionally once the blocklist predicate is `true`.

This behavior is, therefore, active and exploitable by whoever controls `set_blocklist()`, and it materially alters the received funds without reverting the transfer.

Location of Affected Code

File: sources/oft_implementation/oft_adapter_fa.move#L46

```
public(friend) fun credit(
    to: address,
    amount_ld: u64,
    src_eid: u32,
    lz_receive_value: Option<FungibleAsset>,
): u64 acquires OftImpl {
    // Default implementation does not make special use of LZ Receive
       Value sent; just deposit to the OFT address
    option::for_each(lz_receive_value, |fa| primary_fungible_store::
       deposit(@oft_admin, fa));

    // Release rate limit capacity for the pathway (net inflow)
    release_rate_limit_capacity(src_eid, amount_ld);

    // unlock the amount from escrow
    let escrow_signer = &object::generate_signer_for_extending(&store().
       escrow_extend_ref);

    // Deposit the extracted amount to the recipient, or redirect to the
       admin if the recipient is blocklisted
    primary_fungible_store::transfer(
        escrow_signer,
        metadata(),
        redirect_to_admin_if_blocklisted(to, amount_ld),
        amount_ld
    );

    amount_ld
}
```

Impact

The admin can blocklist any wallet, and then all inbound bridge receipts to that wallet get credited to the admin address instead.

Recommendation

If this redirection is intentional for compliance, make it explicit in the docs and UI and require an optin from users. If not desired, replace with a hard revert on credits to blocklisted addresses.

Team Response

Acknowledged.

[M-02] Native Token Always Deposited to Hardcoded Admin Address

Severity

Medium Risk

Description

In `oft_adapter_fa.move`, the credit function deposits all native tokens received via `lz_receive_value` to the hardcoded `@oft_admin` address rather than the current admin returned by `get_admin()`, causing fund leakage after admin transfer.

Location of Affected Code

In the `oft_adapter_fa::credit()` function, any native tokens sent alongside LayerZero message deliveries are unconditionally deposited to the hardcoded `@oft_admin` address:

File: sources/oft_implementation/oft_adapter_fa.move#L46

```
public(friend) fun credit(
    to: address,
    amount_ld: u64,
    src_eid: u32,
    lz_receive_value: Option<FungibleAsset>,
): u64 acquires OftImpl {
    // Default implementation does not make special use of LZ Receive
       Value sent; just deposit to the OFT address
    option::for_each(lz_receive_value, |fa| primary_fungible_store::
       deposit(@oft_admin, fa));
    // ...
}
```

When the OApp admin changes via `oapp_core::transfer_admin()`, this deposit destination remains unchanged. Any native tokens sent by executors continue accruing to the original admin address rather than the current admin. As a side note, that address is not changeable at all.

Additionally, the `oft_impl_config` module initializes with a hardcoded fee deposit address of `@oft_admin`. If that address is to be changeable, a decision should be made if it should also change `fee_deposit_address` if it was set to a default value.

Impact

If that address is to be changed, there is no possibility of doing so. If access is lost or should be revoked, then it could possibly results in loss of deposited funds.

Recommendation

Replace the hardcoded `@oft_admin` with a call to `oapp_core::get_admin()` to deposit to the current admin.

Team Response

Fixed.

[M-03] No Emergency Recovery Mechanism for Escrowed Funds When Remote Pathway Fails

Severity

Medium Risk

Description

In `oft_adapter_fa.move`, the adapter pattern locks tokens in escrow until a matching receive credits them, but provides no recovery mechanism if the remote path becomes permanently inaccessible, e.g. due to removal or misconfiguration.

However, the likelihood of this is low, as either the remote app would need to be broken or the removal of a peer would need to happen on the destination chain after sending, but before receipt, which leaves a small window for this.

While unlikely, the application already prepares for an event of delivery failure by implementing functions such as `nilify()` or `burn()`, but they do not return associated escrowed funds.

Location of Affected Code

The oft_adapter_fa module locks fungible assets in escrow during the debit_fungible_asset operation:

File: sources/oft_implementation/oft_adapter_fa.move#L75

```
public(friend) fun debit_fungible_asset(
    sender: address,
    fa: &mut FungibleAsset,
    min_amount_ld: u64,
    dst_eid: u32,
): (u64, u64) acquires OftImpl {
    // ... fee calculation ...

    // Lock the amount in escrow
    let escrow_address = escrow_address();
    primary_fungible_store::deposit(escrow_address, extracted_fa);

    (amount_sent_ld, amount_received_ld)
}
```

These escrowed tokens are only released when a corresponding credit operation occurs from a successful cross-chain message in `oft_adapter_fa.move`:

File: sources/oft_implementation/oft_adapter_fa.move#L46

```
public(friend) fun credit(
    to: address,
    amount_ld: u64,
    src_eid: u32,
    lz_receive_value: Option<FungibleAsset>,
): u64 acquires OftImpl {
    // ...

    // unlock the amount from escrow
    let escrow_signer = &object::generate_signer_for_extending(&store().
       escrow_extend_ref);

    // Deposit the extracted amount to the recipient
    primary_fungible_store::transfer(
        escrow_signer,
        metadata(),
        redirect_to_admin_if_blocklisted(to, amount_ld),
        amount_ld
    );

    amount_ld
}
```

If the destination chain's OAPP is misconfigured, the escrowed funds become permanently locked. LayerZero provides admin tools to clear, skip, burn, or nilify messages, but none of these allow access to the escrow accounting.

Impact

User funds can be permanently locked in escrow if the remote path becomes inaccessible.

Recommendation

Implement a recovery mechanism that allows fund release. This, however, is a centralized approach, and may require a Timelock or a Multisig as a risk reduction logic.

Team Response

Acknowledged.

[L-01] Admin Can Renounce with Active Blocklist

Severity

Low Risk

Description

The admin can call `renounce_admin()` while the blocklist remains enabled, setting the admin address to `@0x0`. Subsequently, inbound transfers to blocklisted addresses will be transferred to the zero address.

Location of Affected Code

When `renounce_admin()` is called in `oapp_core.move`, the admin address is set to `@0x0`:

File: sources/shared_oapp/oapp_core.move#L239

```
public entry fun renounce_admin(account: &signer) {
    let admin = address_of(move account);
    assert_admin(admin);
    oapp_store::set_admin(@0x0);
    emit(AdminTransferred { admin: @0x0 });
}
```

The blocklist functionality in `oft_impl_config.move` continues to operate normally, but the `redirect_to_admin_if_blocklisted()` function now returns `@0x0` for blocked recipients. Additionally, it is no longer possible to unblock those addresses.

File: sources/shared_oft/oft_impl_config.move#L146

```
public(friend) fun redirect_to_admin_if_blocklisted(recipient: address,
   amount_ld: u64): address acquires Config {
    if (!is_blocklisted(recipient)) {
        recipient
    } else {
        emit(BlockedAmountRedirected {
            amount_ld,
            blocked_address: recipient,
            redirected_to: get_admin(),  // Returns @0x0
        });
        get_admin()  // Returns @0x0
    }
}
```

Impact

All transfers to blocklisted addresses will permanently be sent to `0x0`. Additionally, there is no way to unblock those addresses.

Recommendation

Either disallow renounce when the blocklist is active, or automatically call `irrevocably_disable_blocklist()` internally on admin renounce.

Team Response

Acknowledged.

[L-02] Maximum Fee Cap Allows 100% Fees

Severity

Low Risk

Description

In `sources/shared_oft/oft_impl_config.move`, the `MAX_FEE_BPS` constant is set to `10,000` basis points (100%), allowing the admin to configure fees that would completely confiscate user transfers, which is an unreasonable upper bound for a transfer fee.

Location of Affected Code

File: sources/shared_oft/oft_impl_config.move#L47-L52

```
const MAX_FEE_BPS: u64 = 10_000;

public(friend) fun set_fee_bps(fee_bps: u64) acquires Config {
    assert!(fee_bps <= MAX_FEE_BPS, EINVALID_FEE);
    assert!(fee_bps != store().fee_bps, ESETTING_UNCHANGED);
    store_mut().fee_bps = fee_bps;
    emit(FeeSet { fee_bps });
}
```

Impact

While technically the admin could set any fee below 100%, such a cap provides no protection against misconfiguration or malicious admin behavior. Standard practice in DeFi protocols is to cap fees at reasonable levels (typically 5-20%).

Recommendation

Reduce `MAX_FEE_BPS` to a reasonable maximum, such as `2,000` (20%) or `500` (5%).

Team Response

Acknowledged.

[L-03] Delegate Remains Powerful After Admin Renounces

Severity

Low Risk

Description

The OApp role model separates the "admin" from the "delegate". Authorization for most endpointlevel actions is enforced by `assert_authorized`, which compares the caller to `get_delegate()` and does not involve the admin field. Functions such as `set_config`, `set_send_library()`, `set_receive_library()`, `register_receive_pathway()`, `clear()`, `skip()`, `burn()`, and `nilify()` all gate on `assert_authorized`, so a configured delegate retains the ability to change messaging libraries, manipulate receive pathways, and administratively act on queued messages.

Conversely, adminonly operations use `assert_admin` and affect state kept within the OApp itself, for example, `set_peer()`, `remove_peer()`, `set_enforced_options()`, and `transfer_admin()`

. The `renounce_admin()` path only executes `set_admin(@0x0)` and emits `AdminTransferred`, leaving `get_delegate()` unchanged.

As a result, after an admin renounces, the system remains controllable by the delegate with respect to all actions guarded by `assert_authorized`. This means renouncing admin does not remove a significant class of control over endpoint interactions and operational message handling - any expectations that renouncing produces a fully permissionless state would be incorrect unless the delegate is first cleared by `set_delegate(@0x0)`.

Location of Affected Code

File: sources/shared_oapp/oapp_core.move#L239-L244

```
/// Permanently renounce OApp admin rights. Once this is called, the
   admin cannot be reinstated
public entry fun renounce_admin(account: &signer) {
    let admin = address_of(move account);
    assert_admin(admin);
    oapp_store::set_admin(@0x0);
    emit(AdminTransferred { admin: @0x0 });
}
```

Impact

After the admin renounces, the delegate (if set) still has authority over endpoint configurations and actions. In a decentralized scenario, this can mislead users, as significant control persists.

Recommendation

We recommend documenting clearly that renouncing does not remove delegate powers.

Team Response

Acknowledged.

[L-04] Rewardy Burn Fails if the `from` Address Lacks a Primary Store

Severity

Low Risk

Description

In the `rwd.coin.factory.move`, the `burn()` uses `primary_fungible_store::primary_store()` on `from` address. If `from` never created a primary store, the call reverts even for a zero balance, which is a surprising UX during admin cleanups or forced burns.

It is worth mentioning that the `mint()` path is using `ensure_primary_store_exists()` correctly.

Location of Affected Code

File: RWD_APTOS/rwd.coin.factory.move#L136-L142

```
public entry fun burn(owner: &signer, from: address, amount: u64)
    acquires RewardyCoin {
     not_paused();
     let asset = get_metadata();
     let burn_ref = &authorized_borrow_refs(owner, asset).burn_ref;
     let from_wallet = primary_fungible_store::primary_store(from, asset);
     fungible_asset::burn_from(burn_ref, from_wallet, amount);
}
```

Impact

Admin attempting to burn from a fresh account will see an aborted transaction.

Recommendation

We recommend using `ensure_primary_store_exists()` or a guarded path that treats "no store" as zero.

Team Response

Acknowledged.

[L-05] Removing a Peer Does Not Deregister the Receive Pathway

Severity

Low Risk

Description

In `oapp_corer.move` , peer addition and removal are asymmetric. When a peer is added, the `set_peer` both registers the receive pathway at the endpoint and stores the peer in `oapp_store` .

When a peer is removed, the `remove_peer` only deletes it from `oapp_store` and emits the event - it does not instruct the endpoint to deregister the receive pathway.

As a result, the endpoint may continue to accept deliveries to this OApp for the previously registered source. Those deliveries will then hit `lz_receive_with_value` in `oapp_receive.move` , pass endpoint clearing, and finally revert on `ENOT_PEER` because `get_peer_bytes32` no longer matches.

This produces stuck inbound messages that require manual admin actions at the endpoint layer, and can be used to create extra traffic or force operational workloads.

Location of Affected Code

File: sources/shared_oapp/oapp_core.move#L280-L285

```
public entry fun remove_peer(account: &signer, eid: u32) {
    assert_admin(address_of(move account));
    assert!(oapp_store::has_peer(eid), EUNCONFIGURED_PEER);
    oapp_store::remove_peer(eid);
    emit(PeerSet { eid, peer: ZEROS_32_BYTES() });
}
```

Impact

Operational confusion as received will result in reverts.

Recommendation

In `remove_peer()`, also deregister the corresponding receive pathway at the endpoint.

Team Response

Acknowledged.

[I-01] Application Functions Can Be Called Before Initialization Completes

Severity

Informational Risk

Description

The OFT contract allows user-facing functions to execute before `initialize()` completes, causing aborts when accessing uninitialized state rather than providing clear error messages. The initialization process requires calling multiple initialization functions (`oft_core::initialize()`, `oft_store::initialize()`, `oft_adapter_fa::initialize()`), which a different from the default `init_module()` - should be set before the OFT is fully operational.

However, user entry points like `send_withdraw()`, `quote`, and balance views can be called before initialization completes.

Impact

If those initializations are not called, users may encounter unexpected aborts.

Recommendation

Ensure initialization is always completed before allowing users to interact with the modules. Consider moving initialization to the default `init_module()` and exposing important fields for amendment via setters, instead of relying on a second initialization routine.

Team Response

Acknowledged.

[I-02] Duplicate Error Codes Reduce Debugging Clarity

Severity

Informational Risk

Description

Error codes are reused across different modules with different meanings, potentially causing confusion when interpreting abort codes in dashboards and monitoring tools.

The same numeric error codes have different meanings in different modules. For example, error code `1` represents:

`EUNAUTHORIZED` in `oapp_core` `EINVALID_LOCAL_DECIMALS` in `oft_core` `EINVALID_METADATA_ADDRESS` in `oft_adapter_fa` `EADDRESS_BLOCKED` in `oft_impl_config`

While Move's type system scopes error codes to their module (making aborts technically unambiguous), it reduces maintainability and debugging efficiency.

Recommendation

Use non-overlapping error code ranges for different modules (e.g., `oapp_core` uses `1-99` , `oft_core` uses `100-199` , etc.) or implement a consistent error code namespace.

Team Response

Acknowledged.

our shielding · Your smart contracts, our shielding · Your smart c

shieldify

Thank you!