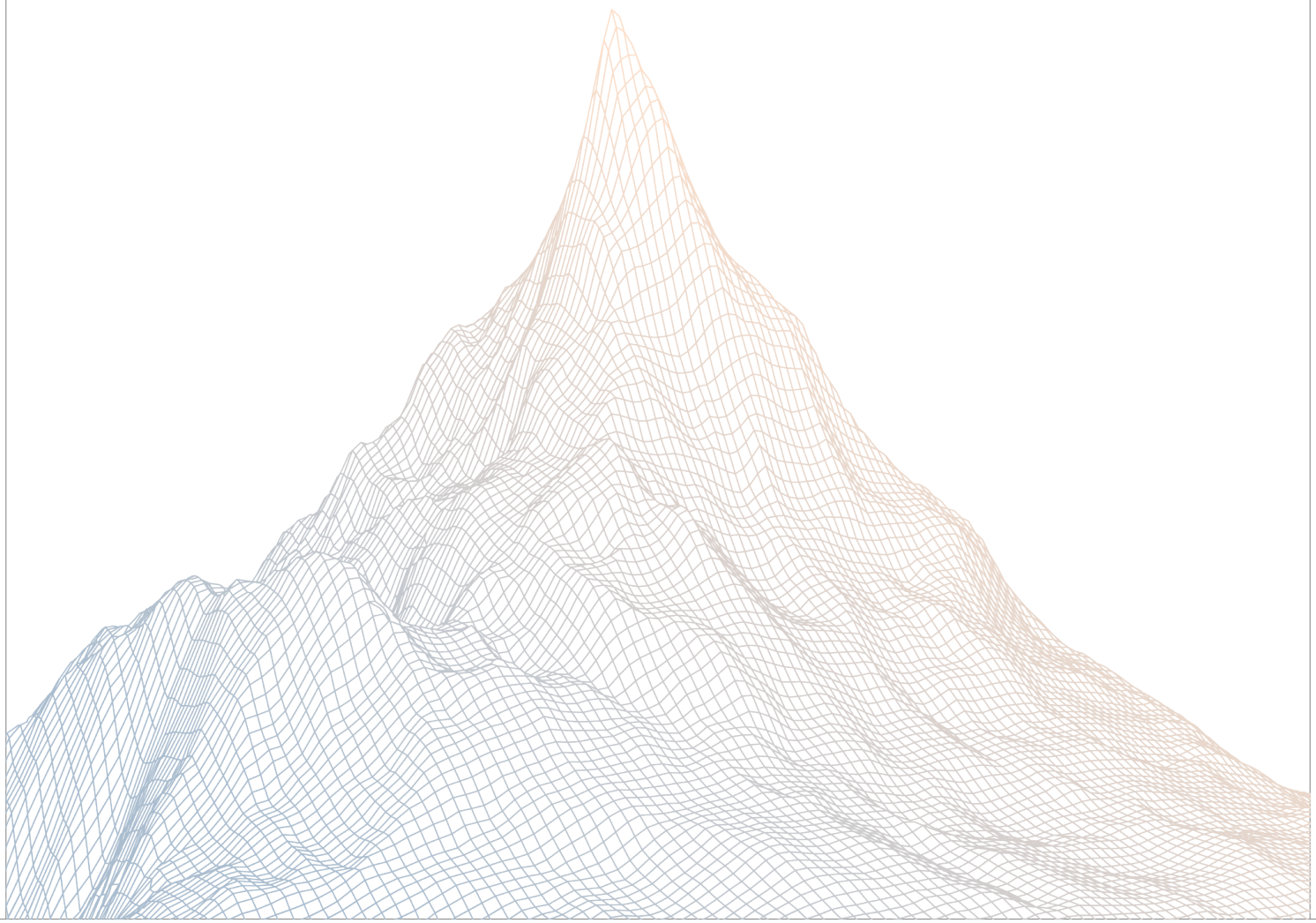
Zenith

Mighty Bear

Smart Contract Security Assessment

VERSION 1.1



AUDIT DATES: March 17th to March 19th, 2025

AUDITED BY: arsen
n4nika

Contents

1 Introduction

1.1 About Zenith

Zenith is an offering by Code4rena that provides consultative audits from the very best security researchers in the space. We focus on crafting a tailored security team specifically for the needs of your codebase.

Learn more about us at https://code4rena.com/zenith.

1.2 Disclaimer

This report reflects an analysis conducted within a defined scope and time frame, based on provided materials and documentation. It does not encompass all possible vulnerabilities and should not be considered exhaustive.

The review and accompanying report are presented on an "as-is" and "as-available" basis, without any express or implied warranties.

Furthermore, this report neither endorses any specific project or team nor assures the complete security of the project.

1.3 Risk Classification

SEVERITY LEVEL	IMPACT: HIGH	IMPACT: MEDIUM	IMPACT: LOW
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

2

Executive Summary

2.1 About Mighty Bear

AI-Infrastructure for the Next Era of Gaming and Entertainment.

2.2 Scope

The engagement involved a review of the following targets:

Target	goat-gaming-smart-contracts-ton
Repository	https://github.com/MightyBear/goat-gaming-smart-contracts-ton
Commit Hash	`7d6c1099f2a3842121c88644b6782a9c8e715a65`
Files	`ton/contracts/*`

2.3 Audit Timeline

March 17, 2025	Audit start
March 18, 2025	Audit end
March 24, 2025	Report published

2.4 Issues Found

SEVERITY	COUNT
Critical Risk	0
High Risk	1
Medium Risk	0
Low Risk	3
Informational	2
Total Issues	**6**

3

Findings Summary

ID	Description	Status
H-1	User funds can be permanently stuck in the protocol controlled jetton_wallet	Resolved
L-1	Incorrect logic in ensure_initialized and ensure_uninitialized	Resolved
L-2	Locking contract doesn't check for the bouncable msg	Resolved
L-3	Bounced op::internal_transfer msg from jetton-minter-discoverable is not processed	Acknowledged
I-1	Users can accidentally hardlock their tokens by not providing forward_ton_amount when sending jettons to either of the system's wallets	Acknowledged
I-2	mode 1 must be used during the send_jetton	Resolved

4

Findings

4.1 High Risk

A total of 1 high risk findings were identified.

[H-1] User funds can be permanently stuck in the protocol controlled jetton_wallet

SEVERITY: High	IMPACT: High
STATUS: Resolved	LIKELIHOOD: Medium

Target

- locking.fc#L130-L151

Description:

1. User has sent the locked funds from his wallet to the protocol controlled `locked_jetton_wallet`.
2. This protocol controlled `locked_jetton_wallet` calls `recv_internal` on the `locking.fc`
3. `locking.fc` calls the `unlock` function. The main purpose of the `unlock` function is to execute 2 actions:

   (a) call protocol controlled `locked_jetton_wallet` and `burn` the user's locked tokens from there

```
var burn_msg_body = begin_cell() ;;to burn locked tokens
        .store_uint(OP_JETTON_BURN, 32)
        .store_uint(query_id, 64) ;; query_id
        .store_coins(amount)
        .store_slice(from_address) ;; response_destination
        .store_maybe_ref(null()) ;; custom_payload
        .end_cell();

    ;;
    var burn_msg = begin_cell()
        .store_uint(0x18, 6)
        .store_slice(locked_jetton_wallet)
        .store_coins(GAS_CONSUMPTION_JETTON_BURN)
        .store_uint(0, 1 + 4 + 4 + 64 + 32 + 1)
```

```
        .store_uint(1, 1)
        .store_ref(burn_msg_body)
        .end_cell();

    send_raw_message(burn_msg, 1);
```

(b) call the protocol controlled `jetton_wallet` and send the normal tokens from there to the user (*via the protocol `jetton_wallet`)

```
    var transfer_msg_body = begin_cell()
        .store_uint(OP_JETTON_TRANSFER, 32)
        .store_uint(query_id, 64) ;; query_id
        .store_coins(amount)
        .store_slice(from_address)
        .store_slice(from_address)
        .store_maybe_ref(null())

    .store_coins(GAS_CONSUMPTION_JETTON_TRANSFER_NOTIFICATION_NO_PAYLOAD)

        .store_maybe_ref(null())
        .end_cell();

    var transfer_msg = begin_cell()
        .store_uint(0x18, 6)
        .store_slice(jetton_wallet)
        .store_coins(0)
        .store_uint(0, 1 + 4 + 4 + 64 + 32 + 1)
        .store_uint(1, 1)
        .store_ref(transfer_msg_body)
        .end_cell();

    send_raw_message(transfer_msg, 64);
}
```

However, the following can happens:

1. `burn` function is called on the protocol controlled `locked_jetton_wallet` and decreases the locked balance respectively, but if the consequent `transfer` of the jettons to the user (*actual unlocking) fails and bounces back to the `locking.fc`, the problem can happens
2. Since `locking.fc` doesn't handle the bounce mechanism, the bounce function logic fails.

So, what state we have?

- The locked funds are `burned` from the protocol controlled `locked_jetton_wallet`

- But Actual funds aren't sent to the user

Why it is a problem? Because, after such failure, it is not possible to send the normal funds which resides in the protocol controlled `jetton_wallet` to the user.

Why? Because if we take a look on protocol controlled `jetton_wallet` we can see in the `op::transfer() -> send_tokens` that it asks the msg.sender to be the owner address, who is actual `locking.fc`

And because the `locking.fc` doesn't have the option of "simple transfer" from the protocol controlled jetton_wallet, the user's funds will be permanently stuck.

```
() recv_internal(int my_balance, int msg_value, cell in_msg_full,
    slice in_msg_body) impure {
    if (in_msg_body.slice_empty?()) {
        return ();
    }

    int op = in_msg_body~load_uint(32); ;; Read operation code

    var cs = in_msg_full.begin_parse(); ;;start parsing

    cs~load_uint(4); ;; Skip flags

    slice sender_address = cs~load_msg_addr(); ;; Get sender address

    if (op == OP_INIT) {
        init(in_msg_body, sender_address, msg_value); ;;@audit callable only
    by admin

        return ();
    } elseif (op == OP_JETTON_MINTER_DISCOVERABLE_TAKE_WALLET_ADDRESS) {
        init_jetton_wallets(in_msg_body, sender_address);

        return ();
    }

    if (op == OP_JETTON_TRANSFER_NOTIFICATION) {
        var (_, _, locked_jetton_master, jetton_wallet,
    locked_jetton_wallet) = load_data();


        ensure_initialized(jetton_wallet, locked_jetton_wallet);

        if (equal_slices_bits(sender_address, jetton_wallet)) {
            lock(in_msg_body, sender_address, msg_value,
    locked_jetton_master);
```

```
        } elseif (equal_slices_bits(sender_address, locked_jetton_wallet)) {
            unlock(in_msg_body, sender_address, msg_value, jetton_wallet,
    locked_jetton_wallet);
        }

        return ();
    }

    throw(38); ;; Unknown operation
}
```

Recommendations:

The most straightforward way to solve this problem is to implement the `op::sendDirectTransfer` from the locking.fc, which would allow to execute the direct transfer from the protocol controlled `jetton_wallet` to any address.

Mighty Bear: Resolved by adding transfer methods in @98de0fd47e6...

Zenith: Verified

4.2 Low Risk

A total of 3 low risk findings were identified.

[L-1] Incorrect logic in `ensure_initialized` and `ensure_uninitialized`

SEVERITY: Low	IMPACT: Low
STATUS: Resolved	LIKELIHOOD: Low

Target

- locking.fc

Description:

The logic to ensure the `locking` contract is initialized/uninitialized is flawed making it possible for the contract to deadlock itself.

The current checks are:

- `ensure_uninitialized`: `is_address_none(jetton_wallet) & is_address_none(locked_jetton_wallet)`
- `ensure_initialized`: `~(is_address_none(jetton_wallet) & is_address_none(locked_jetton_wallet))`

Put into words, the contract is uninitialized when both `locked_jetton_wallet` and `jetton_wallet` are `none` and is initialized when at least one of the two is not `none`.

This causes an issue in the following scenario:

- `init` gets called, requesting the addresses for the `jetton_wallet` and `locked_jetton_wallet`
- if now execution of one of the two messages fails in the `jetton-minter` contract, only one of the two wallets gets set
- this now blocks `init` from being called again because `ensure_uninitialized` returns `false`
- and also prevents meaningful interaction with the contract since both wallets are required to be set for the contract to work

Recommendations:

Consider changing the logic to the following:

- The contract is uninitialized when at least one of the two wallets it `none`
- The contract is initialized when both of the wallets are `~none`

Diff:

```
diff --git a/ton/contracts/locking/locking.fc
    b/ton/contracts/locking/locking.fc
index 1dd69f8..89a3ba6 100644
-- a/ton/contracts/locking/locking.fc
++ b/ton/contracts/locking/locking.fc
@@ -53,11 +53,11 @@ const int GAS_CONSUMPTION_JETTON_BURN = 35000000;
 }

 () ensure_uninitialized(slice jetton_wallet, slice locked_jetton_wallet)
    impure inline {
   throw_unless
   (41, is_address_none(jetton_wallet) &
  is_address_none(locked_jetton_wallet));
   throw_unless
  (41, is_address_none(jetton_wallet) | is_address_none(locked_jetton_
      wallet));
 }

 () ensure_initialized(slice jetton_wallet, slice locked_jetton_wallet)
    impure inline {
   throw_unless
  (42, ~(is_address_none(jetton_wallet) & is_address_none(locked_jetton_
      wallet)));
   throw_unless
(42, ~(is_address_none(jetton_wallet) | is_address_none(locked_jetton_
    wallet)));
 }

 ;; Handle lock request
```

Mighty Bear: Resolved with @98de0fd47e65...

Zenith: Verified.

[L-2] Locking contract doesn't check for the bouncable msg

SEVERITY: Low	IMPACT: Low
STATUS: Resolved	LIKELIHOOD: Low

Target

- locking.fc#L228fc#L228)

Description:

Main locking contract can't handle bounced messages, because it lacks proper flag checks.

In the locking contract we can see that most of the messages are sent with `0x18`, which means that the msg is bouncable.

```
() recv_internal(int my_balance, int msg_value, cell in_msg_full, slice
    in_msg_body) impure {
    if (in_msg_body.slice_empty?()) {
        return (); ;; Ignore empty messages
    }

    int op = in_msg_body~load_uint(32); ;; Read operation code
    var cs = in_msg_full.begin_parse(); ;;start parsing

    cs~load_uint(4); ;; Skip flags
```

If some of the tx's initiated from the locking contract revert, the bouncable logic will not be handled properly, which can potentially result in a inconsistent state

Recommendations:

```
if (flags & 1) {
.........respective bouncable logic.......
    return ();
  }
```

Mighty Bear: Resolved. Dedicated bounced message handling was implemented in

@6643018b2738..., returning early when encountering bounced messages

Zenith: Verified

[L-3] Bounced op::internal_transfer msg from jetton-minter-discoverable is not processed

SEVERITY: Low	IMPACT: Low
STATUS: Acknowledged	LIKELIHOOD: Low

Target

- jetton-minter-discoverable.fc#L55-L56

Description:

When an `op::internal_transfer` message bounces between jetton wallets, the balance is correctly reverted.

```
() on_bounce (slice in_msg_body) impure {

  in_msg_body~skip_bits(32); ;; 0xFFFFFFFF
  (int balance, slice owner_address, slice jetton_master_address,
    cell jetton_wallet_code) = load_data();

  int op = in_msg_body~load_uint(32);
    ;;check the operation comed from bouncable msg
  throw_unless(709, (op == op::internal_transfer())
    | (op == op::burn_notification()));
    ;;it must be only 2 of these commands

  int query_id = in_msg_body~load_uint(64); ;;we retrieve the query id
  int jetton_amount = in_msg_body~load_coins();
    ;;check amount of jetton returned

  balance += jetton_amount; ;;increase the balance back to the correct value

  save_data(balance, owner_address, jetton_master_address,
    jetton_wallet_code);
}
```

However, if the message bounces from a `jetton_wallet` to the `jetton-minter-discoverable`, the `total_supply` is not properly updated.

```
() recv_internal(int msg_value, cell in_msg_full, slice in_msg_body)
    impure {
......
......
    if (flags & 1) { ;; ignore all bounced messages
        return ();
    }
```

As a result, the circulating supply may end up lower than the recorded `total_supply`.

It can happens in the following situation.

1. locking.fc calls `jetton-minter-discoverable` to mint the locked tokens.
   The `total_supply` is updated with `respective jetton_amount` and the function is done

```
mint_tokens(to_address, jetton_wallet_code, amount, master_msg);
    ;;mint tokens
save_data(total_supply + jetton_amount, admin_address, content,
    jetton_wallet_code);
return ();
```

2. However, if the msg's bounces (**we can clearly see that during the `mint_tokens` the msg is sent with 0x18 bouncable mode*), the `jetton-minter-discoverable` doesn't roll back the `total_supply` and the contract end up with the incorrect `total_supply`

```
() recv_internal(int msg_value, cell in_msg_full, slice in_msg_body)
    impure {
......
......

    if (flags & 1) { ;; ignore all bounced messages
        return ();
    }
```

Recommendations:

The `recv_internal` must check what function operation is bounced, and if it is equal to the `op::internal_transfer`, roll back the balance

```
() on_bounce (slice in_msg_body) impure {

  in_msg_body~skip_bits(32);

  int op = in_msg_body~load_uint(32);
```

```
  throw_unless(709, (op == op::mint());

  total_supply -= jetton_amount;

  save_data(total_supply, admin_address, content, jetton_wallet_code);
}
```

Mighty Bear: Acknowledged. Won't Fix as we might be switching jetton-minter-discoverable with another contract.

4.3 Informational

A total of 2 informational findings were identified.

[I-1] Users can accidentally hardlock their tokens by not providing `forward_ton_amount` when sending jettons to either of the system's wallets

`SEVERITY:` Informational	`IMPACT:` Informational
`STATUS:` Acknowledged	`LIKELIHOOD:` Low

Target

- jetton-wallet.fc#L125-L135

Description:

In the `jetton_wallet`, a `transfer_notification` message is only sent if a `forward_ton_amount` is specified. If now a users sends tokens to either of the jetton wallets and does not provide this, their tokens will be stuck on the respective wallet without them getting any tokens in return. Since the `locking` contract is never notified of the transfer.

Recommendations:

This is a user error and cannot be easily fixed but users should be made aware of this in order to prevent confusion.

Mighty Bear: Acknowledged.

[I-2] mode 1 must be used during the send_jetton

SEVERITY: Informational	IMPACT: Informational
STATUS: Resolved	LIKELIHOOD: Low

Target

- locking.fc#L300

Description:

During the update, the `GAS_CONSUMPTION_JETTON_TRANSFER` was added into the `unlock` function, in the `transfer_msg`. Additionally the msg is sent now with mode 1. Basically using fix msg_value

However, such logic is not preserved in the `send_jetton` in the `transfer_msg`. The fixed `GAS_CONSUMPTION_JETTON_TRANSFER` is used as msg.value, but the actual msg is sent with mode64, which carry all the remaining value in addition to main msg.value

Recommendations:

1. Set the msg mode to 1

Mighty Bear: Acknowledged. We're ok with using mode64 for this method because only the admin will use it.

For the other methods, which the users will use, we wanna make sure that the users pay for any fees e.g. storage fees that the transactions will incur and we keep the excess fee.