Cyberscope

Audit Report

Ton Arena

April 2025

Repository : https://gitlab.com/bet_ton/core/-/tree/develop

Commit : 2ae9ebe1cae4f814e291f704cabd6859509a05b6

Audited by © cyberscope

Table of Contents

Risk Classification

The criticality of findings in Cyberscope's smart contract audits is determined by evaluating multiple variables. The two primary variables are:

1. **Likelihood of Exploitation**: This considers how easily an attack can be executed, including the economic feasibility for an attacker.
2. **Impact of Exploitation**: This assesses the potential consequences of an attack, particularly in terms of the loss of funds or disruption to the contract's functionality.

Based on these variables, findings are categorized into the following severity levels:

1. **Critical**: Indicates a vulnerability that is both highly likely to be exploited and can result in significant fund loss or severe disruption. Immediate action is required to address these issues.
2. **Medium**: Refers to vulnerabilities that are either less likely to be exploited or would have a moderate impact if exploited. These issues should be addressed in due course to ensure overall contract security.
3. **Minor**: Involves vulnerabilities that are unlikely to be exploited and would have a minor impact. These findings should still be considered for resolution to maintain best practices in security.
4. **Informative**: Points out potential improvements or informational notes that do not pose an immediate risk. Addressing these can enhance the overall quality and robustness of the contract.

Severity	Likelihood / Impact of Exploitation
Critical	Highly Likely / High Impact
Medium	Less Likely / High Impact or Highly Likely/ Lower Impact
Minor / Informative	Unlikely / Low to no Impact

Review

Repository	https://gitlab.com/bet_ton/core/-/tree/develop/
Commit	2ae9ebe1cae4f814e291f704cabd6859509a05b6
Test Deploys	https://testnet.tonscan.org/address/EQDTwOJgScFhkxqvzdZbQNf3yNzE0_-pZFYrLXSaJIorEJif https://testnet.tonscan.org/address/EQCwB_pAzsTt4t2DUTH6JHBCeG-FjyRDWJk9QY7JcfuMbY8k

Audit Updates

Initial Audit	18 Mar 2025

Source Files

Filename	SHA256
creator.fc	a00eb3c73aa6a0c642566a9a19c6dea3a260ede53090d339d479af09e61e363b
bet.fc	0406a7527653a01acd2b30702e1a2145d33416260b281882f47b4165c44eb012
jetton_bet.fc	e6f1127d0fc5b2fec33768eab098835747ef99426e823097eefaefe65a703339
bet_utils.fc	3b30613284515d73fc51bf1681ae31a9388794ee90a7d1b362a26776051526c5
jetton_proxy.fc	8bc70811f795b3993bf66c61a2ff8d46815737fca2c47a1aaca5f99558b2bf14
liquidity_pool.fc	d9610d053c07f0744b52df2be9025ea61529eb6c5c86ae5bc2bbc536275daeac

opcodes.fc	6bd1ee33412479955727c675efe9e4b8bafa7e83a39dc6d9c156891cdc54c718
stages.fc	ebcd262f89896f5e7ade679953f8cc0560ff2ba3dfc62d4c1816514fb8ac188f
stdlib.fc	752f345de290e9594f3802ff51c5780c5eca2bcc955b161adaffb4be0845e31c
ticket.fc	6fad8f1dd40ff8b294e69d5edefa0a8cbaca2d0d94cceecb60a2b7008f47d915

Overview

The Tonarena project has undergone an audit of its core smart contracts: `creator.fc`, `bet.fc`, `jetton_bet.fc`, `bet_utils.fc`, `jetton_proxy.fc`, `liquidity_pool.fc`, `ticket.fc`, `opcodes.fc`, `stages.fc`, `stdlib.fc`. In the following, an overview of the protocol's main functionalities is provided.

creator.fc

Local State:

`owner`, `commission_percentage`, `bet_code`, `jetton_bet_code`, `lp_address`

op :: create

When the contract receives an internal message with the `create` opcode in the `msg_body`, it extracts several pieces of information from the message. This includes a `uint256` value representing the `game_hash`, the addresses of `player1` and the `arbitrator`, a `uint64` value for the `valid_period`, relevant `bet_info`, a `uint256` for the `bet_size`, the relevant `token_Address` and `jetton_wallet`, and a boolean indicating whether an LP (liquidity pool) is used. The contract then distinguishes between the use of native tokens and a jetton. If no jetton address is specified, it calculates a bet address based on the specified `game_hash`. It sends a message to this address containing the `init` opcode, a `query_id`, a validity period, the `player1` address, the `arbitrator` address, and the `commission_percentage`. The entire message value is forwarded when sending this message. The message's value must be greater than the specified `bet_size`. On the other hand, if a jetton is specified, the contract derives the destination address using the `jetton_bet_code` and the incoming `game_hash`. It then forwards a similar message that includes the `init` opcode. In this case, the value carried by the message is transferred without any restriction on its size.

bet.fc

Local State:

`creator_address` , `game_hash` , `player1` , `player2` , `arbitrator_address` , `commission_percentage` , `valid_till` , `bet_size` , `stage` , `use_lp`

op :: init

Once the contract receives the `init` opcode, it checks that the sender of the message is the `creator_address` . If this check succeeds, it also confirms that the state variable `valid_till` is set to 0 and that the stage is set to `waiting_player2` . It then proceeds to load the validity period from the message as a `unit64` while setting the `valid_till` variable to a future instance. Information about the `player1` , the `arbitrator` address, the `commission_percentage` , the bet size, and the usage of `lp` is loaded from the message. If the latter is enabled, `player2` is set as the `lp` address. The contract then finalizes by updating its state.

op :: join

If the contract receives the `join` opcode, it requires that its state is set to `waiting_player2` to proceed and that it is within its validity period. If `lp` is enabled, it also confirms that the sender of the address is the stored address as `player2` in the global state. The contract aims to restrict access to the `lp` address with this check. If the option for `lp` is not enabled, the contract loads `player2` from the message body and requires the message to carry a value greater than `bet_size` . Finally, the contract sets the stage to `started` and updates its state.

op :: resolve

The `resolve` opcode may be received only by the `arbitrator_address` . Once this opcode is received, the contract loads `player1` 's address and a `uint` to represent `player1` 's choice from the arbitrator's message. Similarly, it loads `player2` 's address and choice from the same message. In addition, the `winner_choice` is loaded from the message. If the winner choice coincides with one of the two addresses, that address is set as the `winner_address` . The winner may receive the contract's balance.

op :: cancel

The `cancel` opcode can only be received by the arbitrator. If the contract is in the `waiting_player2` stage, it will send all its balance, excluding a commission, to `player1` and get destroyed. If it is in the `started` stage, it will refund `player1` and `player2` (excluding possible commissions) and get destroyed.

jetton_bet.fc

Local State: `creator_address` , `game_hash` , `player1` , `player2` , `arbitrator_address` , `commission_percentage` , `valid_till` , `bet_size` , `stage` , `token_address` , `jetton_wallet`

op :: init

Once the contract receives the `init` opcode, it checks that the sender of the message is the `creator_address` . If this check succeeds, it also confirms that the state variable `valid_till` is set to 0 and that the stage is set to `waiting_player1` . It then proceeds to load the validity period from the message as a `uint64` while setting the `valid_till` variable to a future instance. Information about `player1` , the `arbitrator_address` , the `commission_percentage` , the `bet_size` , the `token_address` , and the `jetton_wallet` is loaded from the message. The contract then finalizes by updating its state.

op :: join

Once the contract receives the `init` opcode, it checks that the sender of the message is the `creator_address` . If this check succeeds, it also confirms that the state variable `valid_till` is set to 0 and that the stage is set to `waiting_player1` . It then proceeds to load the validity period from the message as a `uint64` while setting the `valid_till` variable to a future instance. Information about `player1` , the `arbitrator_address` , the `commission_percentage` , the `bet_size` , the `token_address` , and the `jetton_wallet` is loaded from the message. The contract then finalizes by updating its state.

op :: resolve

The `resolve` opcode may be received only by the `arbitrator_address`. Once this opcode is received, the contract loads `player1`'s address and a `uint` to represent `player1`'s choice from the arbitrator's message. Similarly, it loads `player2`'s address and choice from the same message. In addition, the `winner_choice` is loaded from the message. If the `winner_choice` coincides with one of the two addresses, that address is set as the `winner_address`. The winner may receive the contract's balance.

op :: cancel

The `cancel` opcode can only be received by the arbitrator. If the contract is in the `waiting_player1` stage, it will send the remaining balance to the `creator_address`. If it is in the `waiting_player2` stage, it will send all its balance, excluding a commission, to `player1` and get destroyed. If it is in the `started` stage, it will refund `player1` and `player2` (excluding possible commissions) and get destroyed.

liquidity_pool.fc

Local State:

`owner`, `ticket_code`, `interest`

op :: connect

Once the contract receives the `connect` opcode, it will fail unless the message was received by the owner address. If this condition is satisfied, the contract extracts from the message's body an address as the `bet_address` and a `uint256` as the `bet_size`. It then forwards to the `bet_address` a message with opcode `join`, the `query_id`, and a TON balance equal to `bet_size`.

op :: deposit

In the case the message body includes the `deposit` opcode, the contract calculates the `ticket_address` as a function of the `ticket_code` stored in the state variables and the address of the sender of the message. The contract then forwards to that address a message including the `create_ticket` opcode, the `query_id`, the `interest`, and the value of native tokens carried with the internal message.

op :: redeem

Lastly, if the contract receives the `redeem` opcode, it loads from the message body a destination address and a `uint256` as value. Then it calculates an `expected_ticket_address` as a function of the loaded destination address and returns unless the sender of the message is that address. If all checks are confirmed, the contract sends to the destination address TON tokens equal to the loaded value with no other information.

ticket.fc

Local State: `lp` , `owner` , `value` , `start_ts`

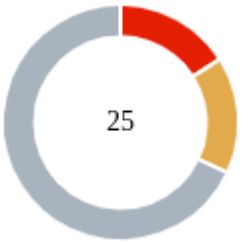
op :: create_ticket

Once the contract receives a message containing the `create_ticket` opcode, it verifies that the sender is the `lp` address. If that condition is verified, the contract loads from the incoming message a `uint7` as the interest and sets the `new_start_ts` to the current timestamp. If this is the first time this opcode is received, `start_ts` is set to the current timestamp, the storage variable named `value` is incremented by the `uint256` loaded to represent incoming coins in the `lp` from the message, and the state is saved. If this is not the first time this message is received, an `elapsed_time` is calculated from the last call, and an interest is calculated using the stored `value` , a fixed `interest_rate` , and the elapsed time. The state variable of `value` is then incremented by interest and the incoming amount, while the `start_ts` is reset to the current timestamp.

op :: redeem

This opcode may be only received by the owner of the contract. Once processed, the state variable for `value` is set to zero, and the contract sends to the `lp` address known from its state a message containing the `redeem` opcode, a `query_id` , the owner, and the zero value. The contract resets the `value` variable to 0 and self-destroys if there is no balance left.

Findings Breakdown



Critical	4
Medium	4
Minor / Informative	17

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	4	0	0	0
Medium	4	0	0	0
Minor / Informative	17	0	0	0

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
Critical	EAA	External Arbitrator Address	Unresolved
Critical	IPD	Inaccessible Pool Deposit	Unresolved
Critical	PPO	Potential Player Overwrite	Unresolved
Critical	UTPD	Unverified Third Party Dependencies	Unresolved
Medium	FJWA	Fixed Jetton Wallet Address	Unresolved
Medium	IBI	Inconsistent Bet Initiation	Unresolved
Medium	LBF	Locked Bounced Funds	Unresolved
Medium	MSF	Missing Storage Fee	Unresolved
Minor / Informative	CCR	Contract Centralization Risk	Unresolved
Minor / Informative	ISI	Inconsistent State Initialization	Unresolved
Minor / Informative	MCM	Misleading Comment Messages	Unresolved
Minor / Informative	MVN	Misleading Variables Naming	Unresolved
Minor / Informative	MAC	Missing Access Control	Unresolved
Minor / Informative	MBC	Missing Balance Check	Unresolved

●	MC	Missing Check	Unresolved
●	MLVC	Missing LP Value Check	Unresolved
●	MPAC	Missing Player Address Check	Unresolved
●	MRO	Missing Refund Operation	Unresolved
●	PLT	Potential Locked TON	Unresolved
●	PUTA	Potentially Uninitialized Ticket Address	Unresolved
●	RSU	Redundant State Update	Unresolved
●	TUU	Time Units Usage	Unresolved
●	TSI	Tokens Sufficiency Insurance	Unresolved
●	UBD	Unchecked Bet Duration	Unresolved
●	UAS	Unverified Argument Structure	Unresolved

EAA - External Arbitrator Address

Criticality	Critical
Location	creator#L62
Status	Unresolved

Description

The contract permits users to deploy new bet instances using externally provided parameters. Specifically, the contract accepts parameters like the arbitrator address directly from the user. An arbitrator is responsible for resolving pending bets. If the arbitrator is controlled by the creator of the bet, it may compromise the fairness of the outcome.

```
if (op == op::create) {
...
int game_hash = in_msg_body~load_uint(256);
cell player1 = in_msg_body~load_ref();
slice arbitrator_address = in_msg_body~load_msg_addr();
...
}
```

Recommendation

It is advisable that crucial parameters for the execution of the contract are provided in an immutable manner by the contract itself, assuming trusted parties. This ensures the integrity and security of the contract's operations.

IPD - Inaccessible Pool Deposit

Criticality	Critical
Location	ticket.fc#L58 liquidity_pool.fc#L102
Status	Unresolved

Description

The ticket contract implements the `redeem` opcode to notify the `liquidity_pool` contract for the withdrawal of a user's funds. However, the forwarded message carries a fixed value set to `0`, which effectively prevents users from accessing their funds.

```
if (op == op::redeem) {
throw_unless(403, equal_slices_bits(sender_address, owner));
value = 0;
...
}
```

Recommendation

Access to users' funds should be unrestricted. This can be achieved by transferring information about the respective value and appropriately handling the users' funds within the pool contract.

PPO - Potential Player Overwrite

Criticality	Critical
Location	jetton_bet.fc#L90
Status	Unresolved

Description

The contract performs state updates that could lead to loss of funds. Specifically, the contract implements the `join` method to allow users to join an active bet. It expects both `player1` and `player2` to send the join opcode from their jetton wallets. If a player other than `player1` is the first to send the `join` opcode, that player is stored as `player2`. `player1` may then send the `join` opcode potentially overwriting `player2`. This can lead to loss of funds for that player.

```
if (op == op::join) {
throw_unless(ERROR::NOT_FROM_JETTON_WALLET, equal_slices_bits(sender_address,
jetton_wallet));
throw_if(ERROR::NOT_ALLOWED_AT_STAGE, stage == stage::started);
throw_unless(ERROR::DEADLINE_PASSED, valid_till >= now());
int jetton_amount = in_msg_body~load_coins();
throw_unless(ERROR::NOT_ENOUGH_VALUE, jetton_amount >= bet_size);
player2 = in_msg_body~load_ref();
if (stage == stage::waiting_player1) {
stage = stage::waiting_player2;
} else {
stage = stage::started;
}
save_data();
return ();
}
```

Recommendation

Access to state variables should be limited to relevant entities. Participants in a bet should not have access to state variables that could potentially lead to system manipulation.

UTPD - Unverified Third Party Dependencies

Criticality	Critical
Location	creator#L62
Status	Unresolved

Description

The contract uses an external contract in order to determine the transaction's flow. The external contract is untrusted. As a result, it may produce security issues and harm the transactions.

```
if (op == op::create) {
int game_hash = in_msg_body~load_uint(256);
cell player1 = in_msg_body~load_ref();
slice arbitrator_address = in_msg_body~load_msg_addr();
int valid_period = in_msg_body~load_uint(64);
slice bet_info = (in_msg_body~load_ref()).begin_parse();
int bet_size = bet_info~load_coins();
slice token_address = bet_info~load_msg_addr();
slice jetton_wallet = bet_info~load_msg_addr();
...
}
```

Specifically, the contract relies on external information provided by the users for the token address and the jetton wallet to use. These contracts are unverified and may cause significant inconsistencies to the current execution flow.

Recommendation

The contract should use a trusted external source. A trusted source could be either a commonly recognized or an audited contract. The pointing addresses should not be able to change after the initialization.

FJWA - Fixed Jetton Wallet Address

Criticality	Medium
Location	jetton_bet.fc#L90
Status	Unresolved

Description

The contract is initialized with a `token_address` and a `jetton_wallet`, both provided by the creator of the bet as specified by the finding `UTPD`. The contract uses these variables to restrict access to the `join` opcode. However, this control limits access only to the specific `jetton_wallet` address. Consequently, only one of the players, either `player1` or `player2`, would be able to join the bet, assuming the wallet is controlled by a single entity.

```
if (op == op::join) {
throw_unless(ERROR::NOT_FROM_JETTON_WALLET, equal_slices_bits(sender_address,
jetton_wallet));
...
}
```

Recommendation

The contract should control access to the `join` function using user-dependent wallet addresses. This can be achieved by calculating the expected wallet address for a given user, using the known jetton address and wallet code. Access to the `join` opcode should be restricted to trusted wallet addresses only.

IBI - Inconsistent Bet Initiation

Criticality	Medium
Location	jetton_bet.fc#L90
Status	Unresolved

Description

The contract, through the `join` opcode, allows users to register for a bet. In the current implementation, the `join` opcode may be sent twice by players who are not `player1`. This could result in the initialization of a bet with significant inconsistencies between the players who deposited tokens and the actual participants of the bet.

```
if (op == op::join) {
throw_unless(ERROR::NOT_FROM_JETTON_WALLET, equal_slices_bits(sender_address,
jetton_wallet));
throw_if(ERROR::NOT_ALLOWED_AT_STAGE, stage == stage::started);
throw_unless(ERROR::DEADLINE_PASSED, valid_till >= now());
int jetton_amount = in_msg_body~load_coins();
throw_unless(ERROR::NOT_ENOUGH_VALUE, jetton_amount >= bet_size);
player2 = in_msg_body~load_ref();
if (stage == stage::waiting_player1) {
stage = stage::waiting_player2;
} else {
stage = stage::started;
}
save_data();
return ();
}
```

Recommendation

It is advisable to ensure that the join operation can be executed only by the expected entities. Specifically, the first player should always be `player1`, and subsequently, `player2` should be set only once to a single entity.

LBF - Locked Bounced Funds

Criticality	Medium
Location	creator.fc#L62
Status	Unresolved

Description

The contract sends a bounceable message along with the associated balance. If the message bounces during the recipient's action phase, the forwarded funds return in the `creator` contract and are not returned to the external caller.

```
if (op == op::create) {
...
cell msg = begin_cell()
.store_msg_flags_and_address_none(BOUNCEABLE)
.store_slice(destination_address)
.store_coins(0)
.store_statinit_ref_and_body_ref(bet_state_init, master_msg.end_cell())
.end_cell();
send_raw_message(msg, SEND_MODE_BOUNCE_ON_ACTION_FAIL |
SEND_MODE_CARRY_ALL_REMAINING_MESSAGE_VALUE);
return ();
...
}
```

Recommendation

The contract should gracefully handle bounced messages to ensure operational consistency. The value of bounced messages should be returned to the original sender.

MSF - Missing Storage Fee

Criticality	Medium
Location	liquidity_pool.fc
Status	Unresolved

Description

TON validators are responsible for collecting storage fees from smart contracts. These fees are deducted from the smart contract's balance during the Storage phase of any transaction, covering the cost of maintaining the account state up to the current time. If the smart contract lacks sufficient balance to cover these fees, it may become frozen. TON validators are responsible for collecting storage fees from smart contracts. These fees are deducted from the smart contract's balance during the Storage phase of any transaction, covering the cost of maintaining the account state up to the current time. If the smart contract lacks sufficient balance to cover these fees, it may become frozen.

In this context, the `liquidity_pool` contract processes transactions without maintaining the required balance for storage fees and as a result may be frozen.

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

Recommendation

It is advisable to ensure that the smart contract maintains a minimum balance needed to pay storage fees. This can be achieved by retaining a portion of the incoming message value to cover the necessary fees.

CCR - Contract Centralization Risk

Criticality	Minor / Informative
Location	creator#L120,127,134 liquidity_pool.fc#L55,75 ticket#L58 bet.fc#L111,149 jetton_bet.fc#L106,135
Status	Unresolved

Description

The contract's functionality and behavior are heavily dependent on external parameters or configurations. While external configuration can offer flexibility, it also poses several centralization risks that warrant attention. Centralization risks arising from the dependence on external configuration include Single Point of Control, Vulnerability to Attacks, Operational Delays, Trust Dependencies, and Decentralization Erosion.

```
if (op == op::connect) {
...
}
```

```
if (op == op::redeem) {
...
}
```

```
if (op == op::resolve) {
...
}
if (op == op::cancel) {
...
}
```

Recommendation

To address this finding and mitigate centralization risks, it is recommended to evaluate the feasibility of migrating critical configurations and functionality into the contract's codebase itself. This approach would reduce external dependencies and enhance the contract's self-sufficiency. It is essential to carefully weigh the trade-offs between external configuration flexibility and the risks associated with centralization.

ISI - Inconsistent State Initialization

Criticality	Minor / Informative
Location	bet.fc#L70 jetton_bet.fc#L76
Status	Unresolved

Description

The contract is using controls in its initialization process to ensure optimal behavior. Specifically, the contract can only execute the init opcode if its `stage` is set as `waiting_player2` . However the state `stage` variable is not properly initialized in the code. Unless the stage is initialized by populating the respective cells in the initialization code, the contract's main functionalities will fail. Similarly, the stage is not initialized for the `jetton_bet` contract.

```
if (op == op::init) {
...
throw_unless(ERROR::NOT_ALLOWED_AT_STAGE, stage ==
stage::waiting_player2);
...
}
```

Recommendation

It is advisable to initialize the state to a proper configuration within the contract. This approach would improve the overall readability and maintainability of the codebase.

MCM - Misleading Comment Messages

Criticality	Minor / Informative
Location	bet.fc#L133 jetton_bet.fc#L128
Status	Unresolved

Description

The contract is using misleading comment messages. Specifically, the contract declares as draws all instances where the `winner_choice` provided by the arbitrator does not match either of the player choices. These comment messages do not accurately reflect the actual implementation, making it difficult to understand the source code. As a result, the users will not comprehend the source code's actual implementation.

```
if (op == op::resolve) {
...
} else {
;; draw
...
}
```

Recommendation

The team is advised to carefully review the comment in order to reflect the actual implementation. To improve code readability, the team should use more specific and descriptive comment messages.

MVN - Misleading Variables Naming

Criticality	Minor / Informative
Location	ticket.fc#L40,58
Status	Unresolved

Description

Variables can have misleading names if their names do not accurately reflect the value they contain or the purpose they serve. The contract uses some variable names that are too generic or do not clearly convey the information stored in the variable. Misleading variable names can lead to confusion, making the code more difficult to read and understand. In particular, the contract implements the `value` variable which increments according to the contents of incoming `create_ticket` messages and the interest accrued. Nevertheless, the contract does not utilize this `value` and does not necessarily possess the implied monetary value.

```
if (op == op::create_ticket) {
...
value += in_msg_body~load_coins();
...
}
```

```
if (op == op::deposit) {
...
.store_coins(msg_value);
...
send_raw_message(msg, SEND_MODE_PAY_FEES_SEPARATELY |
SEND_MODE_BOUNCE_ON_ACTION_FAIL);
...
}
```

Recommendation

It's always a good practice for the contract to contain variable names that are specific and descriptive. The team is advised to keep in mind the readability of the code.

MAC - Missing Access Control

Criticality	Minor / Informative
Location	bet.fc#L101
Status	Unresolved

Description

The contract implements conditional segments to ensure proper code execution based on the message sender. However, even if `use_lp` is set to `false`, the contract still allows access from an LP address.

```
if (use_lp?) {
...
} else {
throw_unless(ERROR::NOT_ENOUGH_VALUE, msg_value > bet_size);
player2 = in_msg_body~load_ref();
}
```

Recommendation

The contract should ensure consistency of operations by implementing proper conditional checks. These checks would prevent the LP address from calling the contract when `use_lp` is set to `false`.

MBC - Missing Balance Check

Criticality	Minor / Informative
Location	liquidity_pool.fc#L55
Status	Unresolved

Description

Once the `connect` opcode is received by the `liquidity_pool` contract from the owner, the contract forwards a message to an active bet address, transferring tokens equal to the `bet_size` value. However, the contract lacks the necessary checks to verify that this value is available in its balance or that the value was sent by the owner.

```
if (op == op::connect) {
throw_unless(403, equal_slices_bits(sender_address, owner));
slice bet_address = in_msg_body~load_msg_addr();
int bet_size = in_msg_body~load_coins();
cell msg = begin_cell()
.store_msg_flags_and_address_none(BOUNCEABLE)
.store_slice(bet_address);
.store_coins(bet_size)
.store_prefix_only_body()
.store_op(op::join)
.store_query_id(query_id)
.end_cell();

send_raw_message(msg, 3);

return ();
}
```

Recommendation

It is advisable to maintain operational consistency by closely monitoring the contract's available balance at all times. This approach helps prevent potential issues and interruptions in operations.

MC - Missing Check

Criticality	Minor / Informative
Location	creator.fc#L56
Status	Unresolved

Description

The contract is processing variables that have not been properly sanitized and checked that they form the proper shape. These variables may produce vulnerability issues. Specifically, the contract does not ensure that the `msg_value` is greater than the required `MIN_TON_FOR_STORAGE`.

```
raw_reserve(MIN_TON_FOR_STORAGE, RESERVE_REGULAR |
RESERVE_BOUNCE_ON_ACTION_FAIL);
msg_value -= MIN_TON_FOR_STORAGE;
```

Recommendation

The team is advised to properly check the variables according to the required specifications.

MLVC - Missing LP Value Check

Criticality	Minor / Informative
Location	bet.fc#L96
Status	Unresolved

Description

The contract implements the `join` method to allow a second player to join the bet. Additionally, it supports using an `lp_address` as a counterpart for the bet. However, in this case, the contract does not verify that the incoming message from the `lp_address` carries the necessary `msg_value` to fulfill the `bet_size`.

```
if (use_lp?) {
slice p2_parsed = player2.begin_parse();
slice lp_address = p2_parsed~load_msg_addr();
throw_unless(ERROR::NOT_FROM_LP, equal_slices_bits(sender_address,
lp_address));
} else {
...
}
```

Recommendation

The contract should verify that all incoming messages carry the necessary `msg_value` to ensure the submission of the required funds, thereby maintaining consistency in the code logic.

MPAC - Missing Player Address Check

Criticality	Minor / Informative
Location	bet.fc#L92 jetton_bet.fc#L90
Status	Unresolved

Description

The contract implements the `join` method to allow players to access the initialized bet. However, it does not include checks to ensure that `player1` and `player2` are not the same entity.

```
if (op == op::join) {
throw_unless(ERROR::NOT_FROM_JETTON_WALLET, equal_slices_bits(sender_address,
jetton_wallet));
throw_if(ERROR::NOT_ALLOWED_AT_STAGE, stage == stage::started);
throw_unless(ERROR::DEADLINE_PASSED, valid_till >= now());
int jetton_amount = in_msg_body~load_coins();
throw_unless(ERROR::NOT_ENOUGH_VALUE, jetton_amount >= bet_size);
player2 = in_msg_body~load_ref();
if (stage == stage::waiting_player1) {
stage = stage::waiting_player2;
} else {
stage = stage::started;
}
save_data();
return ();
}
```

Recommendation

The contract should implement all necessary checks to ensure operational consistency. Such measures include verifying that `player1` and `player2` are different entities.

MRO - Missing Refund Operation

Criticality	Minor / Informative
Location	creator.fc#L62
Status	Unresolved

Description

The contract initiates a new bet if the incoming message carries the necessary `bet_size` funds. If more than `bet_size` tokens are transferred, the entire message value is sent to the bet contract. However, only the `bet_size` amount can be returned.

```
if (op == op::create) {
...
if (is_address_none(token_address)) {
;; game using TON
throw_unless(ERROR::NOT_ENOUGH_VALUE, msg_value > bet_size);
...
send_raw_message(msg, SEND_MODE_BOUNCE_ON_ACTION_FAIL |
SEND_MODE_CARRY_ALL_REMAINING_MESSAGE_VALUE);
return ();
}
```

Recommendation

The contract should manage all scenarios involving transferred funds. Specifically, it should handle cases where an excess amount is sent, ensuring that any surplus is refunded to the external user.

PLT - Potential Locked TON

Criticality	Minor / Informative
Location	creator.fc#L114
Status	Unresolved

Description

The contract contains TON that are unable to be transferred. Thus, it is impossible to access the locked TON. This may produce a financial loss for the users that have sent TON to the contract. Specifically, the contract allows the creation of a bet using jetton while transferring TON carried by the message to the bet address. These tokens may be inaccessible.

```
cell msg = begin_cell()
.store_msg_flags_and_address_none(BOUNCEABLE)
.store_slice(destination_address)
.store_coins(0)
.store_statinit_ref_and_body_ref(bet_state_init, master_msg.end_cell())
.end_cell();
send_raw_message(msg, SEND_MODE_BOUNCE_ON_ACTION_FAIL |
SEND_MODE_CARRY_ALL_REMAINING_MESSAGE_VALUE);
```

Recommendation

The team is advised to either remove the transfer of TON or add a withdrawal functionality. It is important to carefully consider the risks and potential issues associated with locked TON.

PUTA - Potentially Uninitialized Ticket Address

Criticality	Minor / Informative
Location	liquidity_pool.fc#L81
Status	Unresolved

Description

The contract forwards a message to an address calculated using the `ticket_Code` and a user's address. If the ticket is not initialized, the execution of the `deposit` opcode will fail.

```
if (op == op::deposit) {
cell state_init = calculate_ticket_state_init(ticket_code, sender_address);
slice ticket_address = calculate_ticket_address(state_init);
cell msg = begin_cell()
.store_msg_flags_and_address_none(NON_BOUNCEABLE)
.store_slice(ticket_address)
.store_coins(0)
.store_prefix_only_body()
.store_op(op::create_ticket)
.store_query_id(query_id)
.store_uint(interest, 7)
.store_coins(msg_value)
.end_cell();

send_raw_message(msg, SEND_MODE_PAY_FEES_SEPARATELY |
SEND_MODE_BOUNCE_ON_ACTION_FAIL);

save_data();
return ();
}
```

Recommendation

It is advisable to ensure that the recipient contract is correctly initialized to maintain operational consistency and prevent failed transactions. This can be achieved by implementing the necessary conditions or initializing the relevant contracts when necessary.

RSU - Redundant State Update

Criticality	Minor / Informative
Location	jetton_bet.fc#L90
Status	Unresolved

Description

The contract performs redundant state updates that do not accurately reflect its actual state. Specifically, the contract implements the `join` method to allow users to join an active bet. It expects both `player1` and `player2` to send the `join` opcode from their jetton wallets. If `player1` is the first to send the `join` opcode, `player2` can also be set from the incoming message body. This operation is redundant, as it does not reflect the actual state and can potentially be overwritten in a future call to the actual `player2`.

```
if (op == op::join) {
...
player2 = in_msg_body~load_ref();
if (stage == stage::waiting_player1) {
stage = stage::waiting_player2;
} else {
stage = stage::started;
}
save_data();
return ();
}
```

Recommendation

It is advised to consistently update the state of the contract. Specifically, it is advisable to ensure that the `join` opcode can only be accepted by `player1` when no active player is set. Additionally, each player should only have access to their respective state variables.

TUU - Time Units Usage

Criticality	Minor / Informative
Location	ticket.fc#L49
Status	Unresolved

Description

The contract is using arbitrary numbers to form time-related values. As a result, it decreases the readability of the codebase and prevents the compiler from optimizing the source code.

```
int interest = (value * interest_rate * elapsed_time) / (365 * 24 * 60 * 60 * 100);
```

Recommendation

It is good practice to use the time units reserved keywords like `seconds`, `minutes`, `hours`, `days` and `weeks` to process time-related calculations.

It's important to note that these time units are simply a shorthand notation for representing time in seconds, and do not have any effect on the actual passage of time or the execution of the contract. The time units are simply a convenience for expressing time in a more human-readable form.

TSI - Tokens Sufficiency Insurance

Criticality	Minor / Informative
Location	ticket.fc#L40
Status	Unresolved

Description

The tokens are not held within the contract itself. Instead, the contract is designed to provide the tokens from an external administrator. While external administration can provide flexibility, it introduces a dependency on the administrator's actions, which can lead to various issues and centralization risks. In particular, the contract increments its current state by an amount denoted as interest. This is a value that is not included in the contract.

```
if (op == op::create_ticket) {
...
int interest = (value * interest_rate * elapsed_time) / (365 * 24 * 60 * 60 *
100);
value += interest;
...
}
```

Recommendation

It is recommended to consider implementing a more decentralized and automated approach for handling the contract tokens. One possible solution is to hold the tokens within the contract itself. If the contract guarantees the process it can enhance its reliability, security, and participant trust, ultimately leading to a more successful and efficient process.

UBD - Unchecked Bet Duration

Criticality	Minor / Informative
Location	bet.fc#L74 jetton_bet.fc#L77
Status	Unresolved

Description

The `bet` and `jetton_bet` are initialized using user-provided parameters. These parameters include the duration of a bet, which can be set to an unrealistically large value. If such a large value is set, players may not be able to withdraw their funds unless the arbitrator invokes the cancel opcode.

```
int valid_period = in_msg_body~load_uint(64);
valid_till = valid_period + now();
```

Recommendation

It is advisable to ensure that the duration of a bet does not exceed a reasonable limit. This helps maintain consistency and ensures that users can always access their funds.

UAS - Unverified Argument Structure

Criticality	Minor / Informative
Location	creator.fc#L62
Status	Unresolved

Description

The contract accepts the `player1` cell through user-provided information. `player1` is expected to be properly formatted, using a consistent TON address and a hash for the player's choice. However, the contract does not enforce these restrictions, which could lead to inconsistencies during code execution.

```
cell player1 = in_msg_body~load_ref();
```

Recommendation

The contract should ensure that user-provided arguments conform to the proper format. This will maintain operational consistency and enhance trust in the system.

Summary

Ton Arena contracts implement a betting mechanism. This audit investigates security issues, business logic concerns and potential improvements. Ton Arena is an interesting project that has a friendly and growing community. The Smart Contract analysis reported a number of critical issues that could affect the normal operation of the project. The team is advised to take these recommendations into consideration to improve the overall security and consistency of the platform.

Disclaimer

The information provided in this report does not constitute investment, financial or trading advice and you should not treat any of the document's content as such. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes nor may copies be delivered to any other person other than the Company without Cyberscope's prior written consent. This report is not nor should be considered an "endorsement" or "disapproval" of any particular project or team. This report is not nor should be regarded as an indication of the economics or value of any "product" or "asset" created by any team or project that contracts Cyberscope to perform a security assessment. This document does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors' business, business model or legal compliance. This report should not be used in any way to make decisions around investment or involvement with any particular project. This report represents an extensive assessment process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk Cyberscope's position is that each company and individual are responsible for their own due diligence and continuous security Cyberscope's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies and in no way claims any guarantee of security or functionality of the technology we agree to analyze. The assessment services provided by Cyberscope are subject to dependencies and are under continuing development. You agree that your access and/or use including but not limited to any services reports and materials will be at your sole risk on an as-is where-is and as-available basis Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives false negatives and other unpredictable results. The services may access and depend upon multiple layers of third parties.

About Cyberscope

Cyberscope is a blockchain cybersecurity company that was founded with the vision to make web3.0 a safer place for investors and developers. Since its launch, it has worked with thousands of projects and is estimated to have secured tens of millions of investors' funds.

Cyberscope is one of the leading smart contract audit firms in the crypto space and has built a high-profile network of clients and partners.

The Cyberscope team

cyberscope.io