# HashEx
BLOCKCHAIN SECURITY

# Macho Lottery

smart contracts
final audit report

February 2024

hashex.org

contact@hashex.org

# Contents

# 1. Disclaimer

This is a limited report on our findings based on our analysis, in accordance with good industry practice at the date of this report, in relation to cybersecurity vulnerabilities and issues in the framework and algorithms based on smart contracts, the details of which are set out in this report. In order to get a full view of our analysis, it is crucial for you to read the full report. While we have done our best in conducting our analysis and producing this report, it is important to note that you should not rely on this report and cannot claim against us on the basis of what it says or doesn't say, or how we produced it, and it is important for you to conduct your own independent investigations before making any decisions. We go into more detail on this in the disclaimer below – please make sure to read it in full.

By reading this report or any part of it, you agree to the terms of this disclaimer. If you do not agree to the terms, then please immediately cease reading this report, and delete and destroy any and all copies of this report downloaded and/or printed by you. This report is provided for information purposes only and on a non-reliance basis and does not constitute investment advice. No one shall have any right to rely on the report or its contents, and HashEx and its affiliates (including holding companies, shareholders, subsidiaries, employees, directors, officers, and other representatives) (HashEx) owe no duty of care towards you or any other person, nor does HashEx make any warranty or representation to any person on the accuracy or completeness of the report. The report is provided "as is", without any conditions, warranties, or other terms of any kind except as set out in this disclaimer, and HashEx hereby excludes all representations, warranties, conditions, and other terms (including, without limitation, the warranties implied by law of satisfactory quality, fitness for purpose and the use of reasonable care and skill) which, but for this clause, might have effect in relation to the report. Except and only to the extent that it is prohibited by law, HashEx hereby excludes all liability and responsibility, and neither you nor any other person shall have any claim against HashEx, for any amount or kind of loss or damage that may result to you or any other person (including without limitation, any direct, indirect, special, punitive, consequential or pure economic loss or damages, or any loss of income, profits, goodwill, data, contracts, use of money, or business interruption, and whether in delict, tort (including without limitation negligence), contract, breach of statutory duty, misrepresentation (whether innocent or negligent) or otherwise under any claim of any nature whatsoever in any jurisdiction) in any way arising from or connected with this report and the use, inability to use or the results of the use of this report, and any reliance on this report. The analysis of the security is purely based on the smart contracts alone. No applications or operations were reviewed for security. No product code has been reviewed. HashEx owns all copyright rights to the text, images, photographs, and other content provided in the following document. When using or sharing partly or in full, third parties must provide a direct link to the original document mentioning the author (hashex.org).

# 2. Overview

HashEx was commissioned by the Macho Lottery team to perform an audit of their smart contracts. The audit was conducted between 22/02/2024 and 27/02/2024.

The purpose of this audit was to achieve the following:

- Identify potential security issues with smart contracts
- Formally check the logic behind given smart contracts.

Information in this report should be used for understanding the risk exposure of smart contracts, and as a guide to improving the security posture of smart contracts by remediating the issues that were identified.

The code is available in @DegenMacho/macho-lottery-sc Github repository and was audited after the commit 23b6273.

**Update.** The Macho Lottery team has responded to this report. The updated code is @DegenMacho/macho-lottery-sc Github repository and was audited after the commit 6a183b6.

## 2.1 Summary

| Project name | Macho Lottery |
|---|---|
| URL | https://www.machosol.com/ |
| Platform | Solana |
| Language | Rust |
| Centralization level | 🔴 High |
| Centralization risk | 🔴 High |

## 2.2 Contracts

| Name | Address |
|---|---|
| MachoLottery | |

## 3. Project centralization risks

1. The block timestamp serves as the randomness source for selecting winning tickets. This design allows the manager, as well as block producers, to potentially manipulate the lottery's end timestamp to influence the outcome.
2. The lottery cap, which dictates the number of tickets that will be declared winners, can be modified by the account assigned the MANAGER role. This ability to change the lottery cap could impact the fairness and predictability of the lottery's outcome.
3. Additionally, the manager account has the authority to alter the address of the Multisig account. This Multisig account is crucial as it receives all the deposited funds from the winning tickets. The capability to change its address introduces a risk of misdirection of funds.

# 4. Found issues

8
Total issues

| | | |
|---|---|---|
| ● | Medium | 3 (38%) |
| ● | Low | 4 (50%) |
| ● | Info | 1 (12%) |

## Ca8. MachoLottery

| ID | Severity | Title | Status |
|---|---|---|---|
| Ca8Ic1 | ● Medium | Bias in ticket winning probability | ✓ Acknowledged |
| Ca8Ibe | ● Medium | Some tickets can not be redeemed | ✓ Resolved |
| Ca8Ibd | ● Medium | Weak source of randomness | ✓ Acknowledged |
| Ca8Ic0 | ● Low | Incorrect error name | ✓ Resolved |
| Ca8Ic3 | ● Low | Unsafe cast from u32 to i32 | ✓ Resolved |
| Ca8Ic4 | ● Low | No need for started field in the Lottery struct | ✓ Acknowledged |
| Ca8Ibf | ● Low | Not used function parameter | ✓ Acknowledged |
| Ca8Ic2 | ● Info | Typographical errors | ✓ Acknowledged |

# 5. Contracts

## Ca8. MachoLottery

### Overview

The project operates as a lottery system wherein participants can purchase tickets by depositing SOL into the program. An account designated with a manager role can initiate the lottery by establishing the lottery cap and the price per ticket. The lottery cap represents the total number of tickets that will be deemed winners. Should the total number of purchased tickets fall below the cap, all tickets are considered winners. The deposits for the winning tickets are then transferred to an account known as the multisig address.

In scenarios where the total number of tickets purchased exceeds the cap, the program will select a number of winning tickets equal to the cap. Participants holding tickets that were not selected as winners have the opportunity to reclaim their deposits.

### Issues

#### Ca8Ic1 Bias in ticket winning probability ● Medium ✓ Acknowledged

The lottery program's method of determining winning tickets inadvertently biases the odds of winning based on ticket position within the specified range. If the number of participants in the lottery is bigger than the lottery cap, the program selects a range of winning tickets with length equal to the cap.

By selecting winning tickets through a range defined by the first and last ticket ID, the program inherently increases the likelihood of winning for tickets located in the middle of this range.

To illustrate this flaw with an example: consider a scenario with 5 participants, and the lottery cap is set to 3. In such a case, a participant holding ticket number 2 is guaranteed to win because the winning range must include ticket number 2 to reach the required cap of 3 winners. However, tickets at the beginning or end of the sequence (such as tickets 1 and 5) do not have this same guaranteed probability of winning.

## Ca8Ibe Some tickets can not be redeemed

● Medium ✓ Resolved

When the account with the manager role ends the lottery, the program calculates winning tickets by selecting the range defined by the first and the last winning ticket numbers.

```
pub fn process_end_lottery(program_id: &Pubkey, accounts: &[AccountInfo]) -> ProgramResult
{
  ...
    if (participations as i32) < (lottery_cap as i32) {
        // Everyone wins
        lottery_winners_starting_at = 0;
        lottery_winners_ending_at = participations;
        winning_funds = ticket_price * participations as u64;
    } else {
        // Using the timestamp as a seed for the random number generator
        let maximum = participations - lottery_cap + 1;

        // Example, we have 5 participants & lottery_cap is 3
        // We have tickets 0 1 2 3 4
        // The maximum value must be 2. If index_a is 2, winners are 2 3 4
        // Minimum value must be 0, so winners are 0 1 2
        // In this case: maximum = 5 - 3 = 2. 2 + 1 = 3 so x % 3 will give us a number
between 0 and 2, both included
        let value = current_timestamp % maximum as u64;

        lottery_winners_starting_at = value as u32;
        lottery_winners_ending_at = value as u32 + lottery_cap;

        // Now, the winners are only the ones in the range, meaning their ticket must be
>= index_a and <= index_b

        // Only up to lottery_cap can win
        winning_funds = ticket_price * lottery_cap as u64;
    }
    // Storing the results
    lottery_info.lottery_result_index_a = lottery_winners_starting_at;
    lottery_info.lottery_result_index_b = lottery_winners_ending_at;
    ...
}
```

However, there is an error in the program's handling of edge cases, particularly when a user's ticket number exactly matches the winning range's end ID. According to the current logic, this ticket number is not classified as a winning ID. Yet, it is also not marked as eligible for a refund. This discrepancy creates a situation where tickets that should be considered non-winning (and thus eligible for refunds) are neither recognized as winners nor processed for refunds.

```
pub fn process_withdraw_lost_lottery_funds(
    program_id: &Pubkey,
    accounts: &[AccountInfo],
) -> ProgramResult {
...
    // Seing how many lost tickets the user have
    let mut lost_tickets: u32 = 0;

    for i in 0..participant_details.bought_tickets_count as usize {
        let ticket_number = participant_details.bought_tickets[i];
        if (ticket_number < lottery_details.lottery_result_index_a)
            || (ticket_number > lottery_details.lottery_result_index_b)
        {
            msg!("Claiming lost ticket number: {}", ticket_number);
            lost_tickets += 1;
        }
    }
...
}
```

### Recommendation

Make the tickets with IDs equal to eligible for refund.

## Ca8Ibd Weak source of randomness ● Medium ✓ Acknowledged

The timestamp is used as the sole source of randomness for determining the winning tickets. Since the timestamp can be predicted, an account with a manager role can manipulate the outcome by adjusting the timing of ticket purchases or lottery closure to ensure certain tickets win.

```
pub fn process_end_lottery(program_id: &Pubkey, accounts: &[AccountInfo]) -> ProgramResult
{
  ...
  let clock = Clock::get()?;
  let current_timestamp = clock.unix_timestamp as u64;
  ...
    if (participations as i32) < (lottery_cap as i32) { //@note why i32 instead of u32?
        // Everyone wins
        lottery_winners_starting_at = 0;
        lottery_winners_ending_at = participations;
        winning_funds = ticket_price * participations as u64;
    } else {
        // Using the timestamp as a seed for the random number generator
        let maximum = participations - lottery_cap + 1;

        // Example, we have 5 participants & lottery_cap is 3
        // We have tickets 0 1 2 3 4
        // The maximum value must be 2. If index_a is 2, winners are 2 3 4
        // Minimum value must be 0, so winners are 0 1 2
        // In this case: maximum = 5 - 3 = 2. 2 + 1 = 3 so x % 3 will give us a number
between 0 and 2, both included
        let value = current_timestamp % maximum as u64;

        lottery_winners_starting_at = value as u32;
        lottery_winners_ending_at = value as u32 + lottery_cap;

        // Now, the winners are only the ones in the range, meaning their ticket must be
>= index_a and <= index_b

        // Only up to lottery_cap can win
        winning_funds = ticket_price * lottery_cap as u64;
    }
  ...
}
```

## Recommendation

Use an external oracle as a source of randomness.

## Ca8Ic0 Incorrect error name ● Low ✓ Resolved

In the `process_end_lottery()` function, on line 50 there is a wrong error name. Instead of `AccountAlreadyInitialized` there should be `UninitializedAccount.`

```
pub fn process_end_lottery(program_id: &Pubkey, accounts: &[AccountInfo]) -> ProgramResult
{
...
    // Check if the lottery account is already initialized
    if !Lottery::check_is_initialized(&lottery_account.data.borrow()) {
        return Err(ProgramError::AccountAlreadyInitialized.into());
    }
...
}
```

## Ca8Ic3 Unsafe cast from u32 to i32 ● Low ✓ Resolved

In the `process_end_lottery()` function, both the `lottery_cap` and `participations` variables are of `u32` type but are cast to `i32` without verifying their upper limits. This casting approach introduces the risk of integer overflow.

```
pub fn process_end_lottery(program_id: &Pubkey, accounts: &[AccountInfo]) -> ProgramResult
{
...
    let lottery_cap = lottery_info.lottery_cap;
    let participations = lottery_info.participations;
    let ticket_price = lottery_info.ticket_price;

    let lottery_winners_starting_at;
    let lottery_winners_ending_at;
    let winning_funds;

    if (participations as i32) < (lottery_cap as i32) {
```

```
    ...
}
```

## Ca8lc4 No need for started field in the Lottery struct ● Low ✓ Acknowledged

During the initialization phase of the lottery, the `started` field is set. Once the lottery has been initialized, this `started` field is consistently true and therefore could be considered redundant. This suggests that the presence of the `started` field may not be necessary for the functioning of the program, as its value does not change post-initialization and does not contribute additional information or control flow. Eliminating this field could simplify the program's logic and data structure without impacting its operation.

## Ca8lbf Not used function parameter ● Low ✓ Acknowledged

The function `process_end_lottery()` expects to receive a system account as a function parameter but does not use it.

```
pub fn process_end_lottery(program_id: &Pubkey, accounts: &[AccountInfo]) -> ProgramResult
{
    let account_info_iter = &mut accounts.iter();

    let manager_account = next_account_info(account_info_iter)?;
    let lottery_account = next_account_info(account_info_iter)?;
    let multisig_account = next_account_info(account_info_iter)?;
    let _system_program_account = next_account_info(account_info_iter)?;
    ...
}
```

## Ca8lc2 Typographical errors ● Info ✓ Acknowledged

The terms "avaialble" and "inicialized" are used in the contract, which is presumably a typographical error. The correct terms should be "available", "initialized". Misnaming variables can lead to confusion for developers, maintainers, and auditors, potentially obscuring the intent and functionality of the code.

# 6. Conclusion

3 medium, 4 low severity issues were found during the audit. 1 medium, 2 low issues were resolved in the update. The reviewed contracts are highly dependent on the owner's account. See the centralization risks chapter.

## Appendix A. Issues' severity classification

- **Critical.** Issues that may cause an unlimited loss of funds or entirely break the contract workflow.  Malicious code (including malicious modification of libraries) is also treated as a critical severity issue. These issues must be fixed before deployments or fixed in already running projects as soon as possible.
- **High.** Issues that may lead to a limited loss of funds, break interaction with users, or other contracts under specific conditions. Also, issues in a smart contract, that allow a privileged account the ability to steal or block other users' funds.
- **Medium.** Issues that do not lead to a loss of funds directly, but break the contract logic. May lead to failures in contracts operation.
- **Low.** Issues that are of a non-optimal code character, for instance, gas optimization tips, unused variables, errors in messages.
- **Informational.** Issues that do not impact the contract operation. Usually, informational severity issues are related to code best practices, e.g. style guide.

## Appendix B. Issue status description

**Resolved.** The issue has been completely fixed.

**Partially fixed.** Parts of the issue have been fixed but the issue is not completely resolved.

**Acknowledged.** The team has been notified of the issue, no action has been taken.

**Open.** The issue remains unresolved.

## Appendix C. List of examined issue types

- Business logic overview
- Functionality checks
- Following best practices
- Access control and authorization
- Reentrancy attacks
- Front-run attacks
- DoS with (unexpected) revert
- DoS with block gas limit
- Transaction-ordering dependence
- ERC/BEP and other standards violation
- Unchecked math
- Implicit visibility levels
- Excessive gas usage
- Timestamp dependence
- Forcibly sending ether to a contract
- Weak sources of randomness
- Shadowing state variables
- Usage of deprecated code

# Appendix D. Centralization risks classification

## Centralization level

- **High.** The project owners can manipulate user's funds, lock user's funds on their will (reversible or irreversible), or maliciously update contracts parameters or bytecode.
- **Medium.** The project owners can modify contract's parameters to break some functions of the project contract or contracts, but user's funds remain withdrawable.
- **Low.** The contract is trustless or its governance functions are safe against a malicious owner.

## Centralization risk

- **High.** Lost ownership over the project contract or contracts may result in user's losses. Contract's ownership belongs to EOA or EOAs, and their security model is unknown or out of scope.
- **Medium.** Contract's ownership is transferred to a contract with not industry-accepted parameters, or to a contract without an audit. Also includes EOA with a documented security model, which is out of scope.
- **Low.** Contract's ownership is transferred to a well-known or audited contract with industry-accepted parameters.

contact@hashex.org

@hashex_manager

blog.hashex.org

linkedin

github

twitter

#HashEx

BLOCKCHAIN SECURITY