Mythos Studios

Audit

Presented by:

OtterSec contact@osec.io

Akash Gurugunti Sud0u53r.ak@osec.io

Robert Chen r@osec.io

Contents

Appendices

01 | Executive Summary

Overview

Mythos Studios engaged OtterSec to perform an assessment of the `mythos-contracts` program. This assessment was conducted between March 23rd and March 25th, 2023. For more information on our auditing methodology, see Appendix B.

Key Findings

Over the course of this audit engagement, we produced 4 findings total.

In particular, we found an issue with the `unchecked` block that may potentially lead to a loss of funds (OS-MYT-ADV-00) and with the ordering of token minting that may cause a denial of service for early bidders (OS-MYT-ADV-01).

Additionally, we provided a suggestion to disallow auctions from beginning when in the `live` and `preBuy` phase simultaneously (OS-MYT-SUG-00) and to ensure that bidders cannot bid once an auction has ended (OS-MYT-SUG-01).

02 | Scope

The source code was delivered to us in a git repository at github.com/westcoastnft/mythos-contracts. This audit was performed against commit 4dabf00.

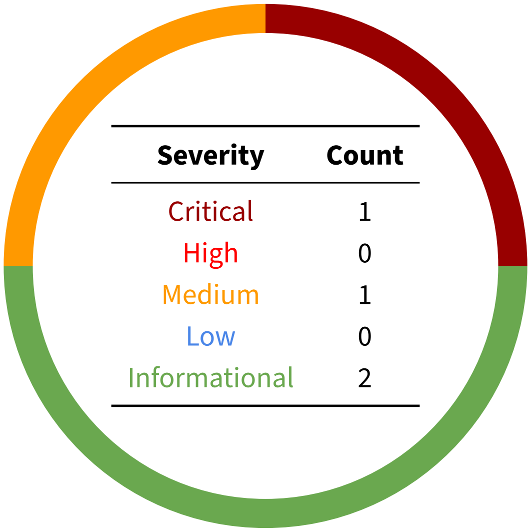
A brief description of the programs is as follows.

Name	Description
Mythos	An auction contract implementing a stepped dutch auction that allows users to place bids and mint Mythos NFTs.

03 | Findings

Overall, we reported 4 findings.

We split the findings into **vulnerabilities** and **general findings**. Vulnerabilities have an immediate impact and should be remediated as soon as possible. General findings don't have an immediate impact but will help mitigate future vulnerabilities.



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

04 | Vulnerabilities

Here, we present a technical analysis of the vulnerabilities we identified during our audit. These vulnerabilities have *immediate* security implications, and we recommend remediation as soon as possible.

Rating criteria can be found in Appendix A.

ID	Severity	Status	Description
OS-MYT-ADV-00	Critical	Resolved	Potential underflow in `refundValue` calculation leads to loss of funds.
OS-MYT-ADV-01	Medium	Resolved	The order for token distribution is not guaranteed to be the same as the bidding order.

OS-MYT-ADV-00 [crit] | Underflow In Refund Value Calculation

Description

The _sendTokensAndRefund value in Mythos.sol is to be refunded and calculated by multiplying the number of tokens minted with the price while subtracting from the user's contribution. This is executed in an unchecked block to save gas on extra checks that are added by the solidity compiler by default.

An account with the REFUND_ROLE may exploit this vulnerability by providing their address as the recipient's to address which is not registered as a bidder.

This will cause the values of userContribution and maxNumberOfTokens to be zero, and as a result, the numberOfTokens value will not be properly validated. An attacker can then pass an arbitrary value to numberOfTokens, causing an underflow during the calculation of refundValue.

Since the amount of refundValue ETH is sent to the receiver, the account with REFUND_ROLE can pass in a value for numberOfTokens such that the refund amount is less than the balance of the contract and steal the funds.

Proof of Concept

Consider the following scenario:

1. An auction is created with predefined parameters.
2. Users place bids on the auction.
3. Once the auction ends and the price is set, if sendTokensAndRefund is called with an address that did not bid, the value of userContribution and maxNumberOfTokens will be zero.
4. Since the value is zero, the value of numberOfTokens is not validated to be less than or equal to the value of maxNumberOfTokens. And by passing in an arbitrary value, an account with REFUND_ROLE may underflow the refundValue and steal ETH from the contract.

Remediation

Remove the unchecked block for the refundValue calculation.

Patch

Fixed in dcdcb3f by removing the unchecked block and moving the numberOfTokens check outside the if block.

OS-MYT-ADV-01 [med] | Undefined Order For Token Distribution

Description

`sendTokensAndRefund` in `Mythos.sol` is responsible for distributing tokens to the bidders based on the price fixed after an auction has ended. This function is called by a user with `REFUND_ROLE`.

As there is a limit on the number of tokens that can be minted, the order in which token receivers are specified becomes significant. It is important that the tokens are distributed in the same order as the bidding process, taking into account each user's last bidding time for sorting.

However, this order is not enforced on-chain, which means that a user who placed a bid early may not receive their tokens if the limit is reached before their tokens are minted.

Proof of Concept

Consider the following scenario:

1. An auction is created with predefined parameters.
2. The auction starts, and a user (user1) bids a certain amount.
3. Another user (user2) bids an amount after user1.
4. Once the auction ends, if user2's tokens are distributed first, then it is possible that user1 is denied minting new tokens due to the token mint limit.

Remediation

Implement the ordering of the token minting on-chain or by ensuring that the ordering of token receivers is properly executed off-chain.

Patch

Fixed in 1fc5784 by transferring the refund functionality to the owner using the `onlyOwner` modifier. The owner should ensure off-chain that the bids are in proper order.

05 | General Findings

Here, we present a discussion of general findings during our audit. While these findings do not present an immediate security impact, they represent antipatterns and could lead to security issues in the future.

ID	Description
OS-MYT-SUG-00	An auction can be in a `live` and `preBuy` phase simultaneously.
OS-MYT-SUG-01	Bidders should not be able to bid after the duration of the auction is complete.

OS-MYT-SUG-00 | Improper Auction Phase

Description

`setPreBuyActive` in `Mythos.sol` is used by the supporter to determine whether the auction is in a `prebuy` phase. The auction cannot begin when in the `prebuy` phase.

However, a supporter can deactivate the `prebuy` phase before starting an auction and can activate it once the auction has started since there are no checks in `setPreBuyActive` to ensure that `preBuyActive` cannot be set to true if the auction is live.

This allows for cases where an auction is in `live` and `prebuy` phase simultaneously.

Remediation

Check that the value of `auctionActive` is false in `setPreBuyActive`.

Patch

Fixed in dcdcb3f by checking `auctionActive` and `price` in `setPreBuyActive`.

OS-MYT-SUG-01 | Disallow Bidding After Auction Duration Ends

Description

`bid` in `Mythos.sol` is used by bidders to bid for an auction. This checks whether an auction is live and allows bidders to bid based on the value of `auctionActive`.

However, an auction ends when the duration of the auction is complete. A bidder should not be able to bid once an auction has ended.

Remediation

Check that the value of `block.timestamp` is less than `startTime + duration`.

A Vulnerability Rating Scale

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

Critical

Vulnerabilities that immediately lead to loss of user funds with minimal preconditions

Examples:

- Misconfigured authority or access control validation
- Improperly designed economic incentives leading to loss of funds

High

Vulnerabilities that could lead to loss of user funds but are potentially difficult to exploit.

Examples:

- Loss of funds requiring specific victim interactions
- Exploitation involving high capital requirement with respect to payout

Medium

Vulnerabilities that could lead to denial of service scenarios or degraded usability.

Examples:

- Malicious input that causes computational limit exhaustion
- Forced exceptions in normal user flow

Low

Low probability vulnerabilities which could still be exploitable but require extenuating circumstances or undue risk.

Examples:

- Oracle manipulation with large capital requirements and multiple transactions

Informational

Best practices to mitigate future security risks. These are classified as general findings.

Examples:

- Explicit assertion of critical internal invariants
- Improved input validation

B | Procedure

As part of our standard auditing procedure, we split our analysis into two main sections: design and implementation.

When auditing the design of a program, we aim to ensure that the overall economic architecture is sound in the context of an on-chain program. In other words, there is no way to steal funds or deny service, ignoring any chain-specific quirks. This usually requires a deep understanding of the program's internal interactions, potential game theory implications, and general on-chain execution primitives.

One example of a design vulnerability would be an on-chain oracle that could be manipulated by flash loans or large deposits. Such a design would generally be unsound regardless of which chain the oracle is deployed on.

On the other hand, auditing the implementation of the program requires a deep understanding of the chain's execution model. While this varies from chain to chain, some common implementation vulnerabilities include reentrancy, account ownership issues, arithmetic overflows, and rounding bugs.

As a general rule of sum, implementation vulnerabilities tend to be more "checklist" style. In contrast, design vulnerabilities require a strong understanding of the underlying system and the various interactions: both with the user and cross-program.

As we approach any new target, we strive to get a comprehensive understanding of the program first. In our audits, we always approach targets with a team of auditors. This allows us to share thoughts and collaborate, picking up on details that the other missed.

While sometimes the line between design and implementation can be blurry, we hope this gives some insight into our auditing procedure and thought process.