Ethena Security Review

Pashov Audit Group

Conducted by: 0xunforgiven, SpicyMeatball, btk

May 20th 2024 - May 23th 2024

Contents

1. About Pashov Audit Group

Pashov Audit Group consists of multiple teams of some of the best smart contract security researchers in the space. Having a combined reported security vulnerabilities count of over 1000, the group strives to create the absolute very best audit journey possible - although 100% security can never be guaranteed, we do guarantee the best efforts of our experienced researchers for your blockchain protocol. Check our previous work here or reach out on Twitter @pashovkrum.

2. Disclaimer

A smart contract security review can never verify the complete absence of vulnerabilities. This is a time, resource and expertise bound effort where we try to find as many vulnerabilities as possible. We can not guarantee 100% security after the review or even if the review will find any problems with your smart contracts. Subsequent security reviews, bug bounty programs and on-chain monitoring are strongly recommended.

3. Introduction

A time-boxed security review of the **ethena-labs/ethena-mint-contract-audit** repository was done by **Pashov Audit Group**, with a focus on the security aspects of the application's smart contracts implementation.

4. About Ethena

Ethena Mint V2 introduces delta limits for price divergence, distinct mint/redeem limits, an on-chain identity whitelist, and EIP-1271 signature verification, all configurable via Ethena multi-sig.

5. Risk Classification

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

5.1. Impact

- High - leads to a significant material loss of assets in the protocol or significantly harms a group of users.
- Medium - only a small amount of funds can be lost (such as leakage of value) or a core functionality of the protocol is affected.
- Low - can lead to any kind of unexpected behavior with some of the protocol's functionalities that's not so critical.

5.2. Likelihood

- High - attack path is possible with reasonable assumptions that mimic on-chain conditions, and the cost of the attack is relatively low compared to the amount of funds that can be stolen or lost.
- Medium - only a conditionally incentivized attack vector, but still relatively likely.
- Low - has too many or too unlikely assumptions or requires a significant stake by the attacker with little or no incentive.

5.3. Action required for severity levels

- Critical - Must fix as soon as possible (if already deployed)
- High - Must fix (before deployment if not already deployed)
- Medium - Should fix
- Low - Could fix

6. Security Assessment Summary

review commit hash - b60b7193636d499ce7f89c4f5afe3b99cf31a2b6

fixes review commit hash - 9cd4ad7b46acc35f6b3340c808200279fbe75de0

Scope

The following smart contracts were in scope of the audit:

- `EthenaMinting`
- `SingleAdminAccessControl`

7. Executive Summary

Over the course of the security review, 0xunforgiven, SpicyMeatball, btk engaged with Ethena to review Ethena. In this period of time a total of **3** issues were uncovered.

Protocol Summary

Protocol Name	Ethena
Repository	https://github.com/ethena-labs/ethena-mint-contract-audit
Date	May 20th 2024 - May 23th 2024
Protocol Type	Synthetic Dollar Protocol

Findings Count

Severity	**Amount**
Medium	1
Low	2
Total Findings	**3**

Summary of Findings

ID	Title	Severity	Status
[M-01]	Some orders can be executed multiple times	Medium	Resolved
[L-01]	Missing sanity checks when setting the tokenConfig	Low	Resolved
[L-02]	ETH and WETH redemption limits can be combined	Low	Acknowledged

8. Findings

8.1. Medium Findings

[M-01] Some orders can be executed multiple times

Severity

Impact: High

Likelihood: Low

Description

Functions `_deduplicateOrder()` and `verifyNonce()` are in charge of deduplicating orders and making sure that the same nonce can't be used twice. The issue is that in `verifyNonce()` code converts `invalidatorBit` to `uint128` with unsafe cast and the value of `invalidatorBit` could become 0 while overflow happens and in that case, the `invalidator` won't be set to 1 and that order can be executed multiple times by minter and redeemer role. The issue impact is that users' funds can be manipulated without their consent which could cause them loss.

The issue will happen whenever `uint8(nonce) > 128`:

```
uint128 invalidatorSlot = uint64(nonce) >> 8;
 uint128 invalidatorBit = uint128(1 << uint8(nonce));
 uint128 invalidator = _orderBitmaps[sender][invalidatorSlot];
 if (invalidator & invalidatorBit != 0) revert InvalidNonce();
```

Recommendations

Don't cast `invalidatorBit` to `uint128` or use `uint7(nonce)`

8.2. Low Findings

[L-01] Missing sanity checks when setting the `tokenConfig`

The `addSupportedAsset()` function allows the admin to add new tokens with built-in sanity checks to ensure token validity:

```
if (tokenConfig[asset].isActive || asset == address(0) || asset == address
      (usde)) {
      revert InvalidAssetAddress();
    }
```

However, during deployment, new tokens are added using the internal `_setTokenConfig()` function, which lacks these validations:

```
for (uint128 k = 0; k < _tokenConfig.length;) {
      _setTokenConfig(_assets[k], _tokenConfig[k]);
      unchecked {
        ++k;
      }
    }
```

To prevent errors during deployment, consider adding similar checks:

```
for (uint128 k = 0; k < _tokenConfig.length;) {
      if (_assets[k] == address(0) || _assets[k] == address
        (usde)) revert InvalidAssetAddress();
      _setTokenConfig(_assets[k], _tokenConfig[k]);
      unchecked {
        ++k;
      }
    }
```

[L-02] ETH and WETH redemption limits can be combined

The `EthenaMinting.sol` contract introduced asset based limits for minting and redemption in addition to global limits.

```
modifier belowMaxMintPerBlock(uint128 mintAmount, address asset) {
    TokenConfig memory _config = tokenConfig[asset];
    if (!_config.isActive) revert UnsupportedAsset();
 >> if
   (totalPerBlockPerAsset[block.number][asset].mintedPerBlock + mintAmount > _config.m
      revert MaxMintPerBlockExceeded();
    }
    _;
  }
  modifier belowMaxRedeemPerBlock(uint128 redeemAmount, address asset) {
    TokenConfig memory _config = tokenConfig[asset];
    if (!_config.isActive) revert UnsupportedAsset();
>>  if
  (totalPerBlockPerAsset[block.number][asset].redeemedPerBlock + redeemAmount > _confi
      revert MaxRedeemPerBlockExceeded();
    }
    _;
  }
```

In one case, these limits can be bypassed. When redeeming USDe for ETH and WETH tokens, the different `totalPerBlockPerAsset[block.number]` accumulators are incremented, even though the user basically receives the same ETH asset (WETH can be easily unwrapped). This allows users to redeem USDe beyond their block limits.

```
function redeem(Order calldata order, Signature calldata signature)
    external
    override
    nonReentrant
    onlyRole(REDEEMER_ROLE)
>>  belowMaxRedeemPerBlock(order.usde_amount, order.collateral_asset)
    belowGlobalMaxRedeemPerBlock(order.usde_amount)
  {
    ---SNIP---
>>
    totalPerBlockPerAsset[block.number][order.collateral_asset].redeemedPerBlock += or
```

It is recommended to increment both accumulators if the asset is WETH or ETH.