# Ethena USDe

## Security Review

Cantina Managed review by:
**Kurt Barry**, Lead Security Researcher
**0x4non**, Associate Security Researcher

October 31, 2023

# Contents

# 1 Introduction

## 1.1 About Cantina

Cantina is a security services marketplace that connects top security researchers and solutions with clients. Learn more at cantina.xyz

## 1.2 Disclaimer

Cantina Managed provides a detailed evaluation of the security posture of the code at a particular moment based on the information available at the time of the review. While Cantina Managed endeavors to identify and disclose all potential security issues, it cannot guarantee that every vulnerability will be detected or that the code will be entirely secure against all possible attacks. The assessment is conducted based on the specific commit and version of the code provided. Any subsequent modifications to the code may introduce new vulnerabilities that were absent during the initial review. Therefore, any changes made to the code require a new security review to ensure that the code remains secure. Please be advised that the Cantina Managed security review is not a replacement for continuous security measures such as penetration testing, vulnerability scanning, and regular code reviews.

## 1.3 Risk assessment

| Severity | Description |
|---|---|
| **Critical** | *Directly* exploitable security vulnerabilities that need to be fixed. |
| **High** | Security vulnerabilities that may not be directly exploitable or may require certain conditions in order to be exploited. All high issues should be addressed. |
| **Medium** | Objective in nature but are not security vulnerabilities. Should be addressed unless there is a clear reason not to. |
| **Low** | Subjective in nature. They are typically suggestions around best practices or readability. Code maintainers should use their own judgment as to whether to address such issues. |
| **Gas Optimization** | Suggestions around gas saving practices. |
| **Informational** | Suggestions around best practices or readability. |

### 1.3.1 Severity Classification

The severity of security issues found during the security review is categorized based on the above table. When determining the severity one first needs to determine whether the finding is subjective or objective. All subjective findings are considered of low severity.

Next it is determined whether the finding can be regarded as a security vulnerability. Some findings might be objective improvements that need to be fixed, but do not impact the project's security overall (Medium).

Finally, objective findings of security vulnerabilities are classified as either critical or high. Critical findings should be directly vulnerable and have a high likelihood of being exploited. High findings on the other hand may require specific conditions that need to be met before the vulnerability becomes exploitable.

## 2 Security Review Summary

Ethena provides derivative infrastructure in order to transform Ethereum into the first crypto-native yield bearing synthetic dollar which is not reliant on the banking system: USDe.

From Oct 16th to Oct 20th the Cantina team conducted a review of ethena on commit hash beb03623. The team identified a total of **24** issues in the following risk categories:

- Critical Risk: 0
- High Risk: 0
- Medium Risk: 0
- Low Risk: 2
- Gas Optimizations: 14
- Informational: 8

# 3 Findings

## 3.1 Low Risk

### 3.1.1 Socialized Frozen Balance Redistributions Are Vulnerable to Sandwiching

**Severity:** Low Risk

**Context:** StakedUSDe.sol#L153

**Description:** One option for redistributing frozen funds is to gift them proportionally to all other stakers by burning shares from the blacklisted address and not minting any new, compensating shares. This bypasses the usual vesting period for new rewards and can thus be sandwiched via depositing prior to and then withdrawing after a socialized redistribution, allowing the sandwicher to obtain a disproportionate share of the redistributed funds. If the cooldown period is non-zero, the attack is still possible, although it is impossible to use flash loans, reducing the severity.

**Recommendation:** Consider using the vesting functionality to make it harder to profit from such attacks. At a minimum use a private RPC endpoint for such transactions and do not announce them in advance if they will be significant in size.

**Ethena:** Vesting added in commit b34aa97a.

**Cantina:** Fix verified.

### 3.1.2 Re-using `MINTER_ROLE` for `transferToCustody` Authorization Violates the Principle of Least Privilege

**Severity:** Low Risk

**Context:** EthenaMinting.sol#L247

**Description:** The `transferToCustody` function moves arbitrary assets from the EthenaMinting contract into a target custodian address. Authorization to call this function is based on whether the caller possesses the `MINTER_ROLE` role. It's not necessary for minters to call this function as part of normal minting, and hence is an extra privilege granted to minters that they do not strictly need, violating the principle of least privilege. This increases the scope of actions a malicious or compromised minter address can engage in.

**Recommendation:** Consider using a distinct role to determine access to this function, or even making it admin-only.

**Ethena:** Fixed in commit 1d8dff15 by adding a distinct role, `COLLATERAL_MANAGER_ROLE`.

**Cantina:** Fix verified.

## 3.2 Gas Optimization

### 3.2.1 Unnecessary Non-Owner Check When Removing an Address From the Blacklist

**Severity:** Gas Optimization

**Context:** StakedUSDe.sol#L123

**Description:** The `notOwner` modifier prevents the `owner` address from being removed from the blacklist. However, since the `owner` address can't be blacklisted, it should never be on the blacklist in the first place, and even if it were, it would be desirable to be able to remove it.

**Recommendation:** Remove the `notOwner` modifier from `removeFromBlacklist`.

**Ethena:** Fixed in commit ba05410f.

**Cantina:** Fix verified.

### 3.2.2 Unnecessary Invocation of `getUnvestedAmount()`

**Severity:** Gas Optimization

**Context:** StakeUSDe.sol#L91

**Description:** The indicated line adds the return value of `getUnvestedAmount()` to the `amount` argument. However, the line above enforces a revert unless `getUnvestedAmount()` returns zero. Thus this modification to `amount` is unnecessary.

**Recommendation:** Use `amount` directly, removing this line, to reduce gas costs.

**Ethena:** Fixed in commit b34aa97a.

**Cantina:** Fix verified.

### 3.2.3 Compact `UserCooldown` struct used in `StakedUSDeV2`

**Severity:** Gas Optimization

**Context:** IStakedUSDeCooldown.sol#L7-L10

**Description:** The `UserCooldown` struct currently utilizes a `uint256` for the `underlyingAmount`. This can be optimized to use a `uint152` type, potentially saving gas by reducing the storage footprint. This optimization assumes that the `underlyingAmount` will never exceed `type(uint152).max`.

**Recommendation:** Refactor the `UserCooldown` struct to:

```
struct UserCooldown {
  uint104 cooldownEnd;
  uint152 underlyingAmount;
}
```

Ensure that the `underlyingAmount` will never surpass the maximum value of `uint152` before implementing this change.

**Ethena:** Fixed in commit ddb2ed94.

**Cantina:** Fix verified.

### 3.2.4 Use `unchecked` for subtraction in `getUnvestedAmount`

**Severity:** Gas Optimization

**Context:** StakedUSDe.sol#L180

**Description:** In the `getUnvestedAmount` function, the subtraction operation `VESTING_PERIOD - timeSinceLastDistribution` can be wrapped in an `unchecked` block. This is safe due to the preceding `if` statement which ensures that `timeSinceLastDistribution` is always less than `VESTING_PERIOD`.

**Recommendation:**

```
- return ((VESTING_PERIOD - timeSinceLastDistribution) * vestingAmount) / VESTING_PERIOD;
+ uint256 deltaT;
+ unchecked {
+   deltaT = (VESTING_PERIOD - timeSinceLastDistribution);
+ }
+ return (deltaT * vestingAmount) / VESTING_PERIOD;
```

**Ethena:** Fixed in commit 0cda7bfb.

**Cantina:** Fix verified.

### 3.2.5 Simplify `cooldownAssets` and `cooldownShares` using named returns in `StakedUSDeV2`

**Severity:** Gas Optimization

**Context:** StakedUSDeV2.sol#L95, StakedUSDeV2.sol#L111

**Description:** The functions `cooldownAssets` and `cooldownShares` in the `StakedUSDeV2` contract currently declare a return variable within the function body and then explicitly return it at the end. This can be simplified and made more concise by using named return values in the function signature, removing the need for an explicit return statement.

**Recommendation:** Consider refactoring the `cooldownAssets` and `cooldownShares` functions to use named return values. This can make the code shorter and more readable. Below is a diff showcasing the recommended changes:

```
- function cooldownAssets(uint256 assets, address owner) external ensureCooldownOn returns (uint256) {
+ function cooldownAssets(uint256 assets, address owner) external ensureCooldownOn returns (uint256 shares) {
    if (assets > maxWithdraw(owner)) revert ExcessiveWithdrawAmount();

-   uint256 shares = previewWithdraw(assets);
+   shares = previewWithdraw(assets);

     // ...........
-   return shares;
  }

- function cooldownShares(uint256 shares, address owner) external ensureCooldownOn returns (uint256) {
+ function cooldownShares(uint256 shares, address owner) external ensureCooldownOn returns (uint256 assets) {
    if (shares > maxRedeem(owner)) revert ExcessiveRedeemAmount();

-   uint256 assets = previewRedeem(shares);
+   assets = previewRedeem(shares);

     // ...........
-   return assets;
  }
```

**Ethena:** Fixed in commit 3f57fdf7.

**Cantina:** Fix verified.

### 3.2.6 Replace `_msgSender()` with `msg.sender` in `StakedUSDeV2`

**Severity:** Gas Optimization

**Context:** StakedUSDeV2.sol#L103, StakedUSDeV2.sol#L119

**Description:** The contract `StakedUSDeV2` uses the `_msgSender()` function to retrieve the sender's address. However, there's no indication that metatransactions are being utilized in this contract. Using `_msgSender()` without metatransactions can introduce unnecessary complexity and potential overhead.

**Recommendation:** If metatransactions are not intended to be used, consider replacing `_msgSender()` with the native `msg.sender` for simplicity and clarity.

**Ethena:** Fixed in commit f46ec3f7.

**Cantina:** Fix verified.

### 3.2.7 `verifyNonce` Always Returns `true`

**Severity:** Gas Optimization

**Context:** EthenaMinting.sol#L385

**Description:** The `verifyNonce` function returns a boolean value, but this value is always `true` (the function reverts in any case that returning `false` would be appropriate). Since this function is called on-chain, having this unneeded return value uses unnecessary gas.

**Recommendation:** Remove the unused return value to save gas.

**Ethena:** Fixed in commit 790673ed.

**Cantina:** Fix verified.

### 3.2.8 Remove extra variable declaration in `verifyNonce` and `_deduplicateOrder`

**Severity:** Gas Optimization

**Context:** EthenaMinting.sol#L381-L382, EthenaMinting.sol#L393-L394

**Description:** The functions `verifyNonce` and `_deduplicateOrder` in `EthenaMinting` use an extra variable declaration for `invalidatorStorage` which seems unnecessary. The code can be optimized by directly accessing the `_orderBitmaps` mapping without the need for this extra variable. This not only reduces the lines of code but potentially makes the code more readable and efficient.

**Recommendation:** Consider refactoring the code as suggested in the provided diff.

```
diff --git a/protocols/USDe/contracts/EthenaMinting.sol b/protocols/USDe/contracts/EthenaMinting.sol
index 32da3a5..dacf1c4 100644
--- a/protocols/USDe/contracts/EthenaMinting.sol
+++ b/protocols/USDe/contracts/EthenaMinting.sol
@@ -378,8 +378,7 @@ contract EthenaMinting is IEthenaMinting, SingleAdminAccessControl, ReentrancyGu
     if (nonce == 0) revert InvalidNonce();
     uint256 invalidatorSlot = uint64(nonce) >> 8;
     uint256 invalidatorBit = 1 << uint8(nonce);
-    mapping(uint256 => uint256) storage invalidatorStorage = _orderBitmaps[sender];
-    uint256 invalidator = invalidatorStorage[invalidatorSlot];
+    uint256 invalidator = _orderBitmaps[sender][invalidatorSlot];
     if (invalidator & invalidatorBit != 0) revert InvalidNonce();

     return (true, invalidatorSlot, invalidator, invalidatorBit);
@@ -390,8 +389,7 @@ contract EthenaMinting is IEthenaMinting, SingleAdminAccessControl, ReentrancyGu
   /// @notice deduplication of taker order
   function _deduplicateOrder(address sender, uint256 nonce) private returns (bool) {
     (bool valid, uint256 invalidatorSlot, uint256 invalidator, uint256 invalidatorBit) = verifyNonce(sender,
↪  nonce);
-    mapping(uint256 => uint256) storage invalidatorStorage = _orderBitmaps[sender];
-    invalidatorStorage[invalidatorSlot] = invalidator | invalidatorBit;
+    _orderBitmaps[sender][invalidatorSlot] = invalidator | invalidatorBit;
     return valid;
   }
```

**Ethena:** Fixed in commit 74f21b8a.

**Cantina:** Fix verified.

### 3.2.9 Use direct comparison in `verifyRoute` return statement

**Severity:** Gas Optimization

**Context:** EthenaMinting.sol#L370-L373

**Description:** The function `verifyRoute` checks if `totalRatio` is equal to `10_000` and returns a boolean value based on the comparison. The current implementation uses an if-else structure to determine the return value.

**Recommendation:** To make the code more concise and potentially more gas efficient, consider replacing the `if-else` statement with a single return statement: `return (totalRatio == 10_000);`.

```
--- a/protocols/USDe/contracts/EthenaMinting.sol
+++ b/protocols/USDe/contracts/EthenaMinting.sol
@@ -367,10 +367,7 @@ contract EthenaMinting is IEthenaMinting, SingleAdminAccessControl, ReentrancyGuard
       }
       totalRatio += route.ratios[i];
     }
-    if (totalRatio != 10_000) {
-      return false;
-    }
-    return true;
+    return (totalRatio == 10_000);
   }
```

**Ethena:** Fixed in commit 36bed285.

**Cantina:** Fix verified.

### 3.2.10 Use `unchecked` and pre-increment on `for` loops

**Severity:** Gas Optimization

**Context:** EthenaMinting.sol#L126, EthenaMinting.sol#L130, EthenaMinting.sol#L363, EthenaMinting.sol#L424

**Description:** Using pre-increment `++i` instead of post-increment `i++` can saves 5 gas per loop, also using `unchecked` for incrementing `i` can saves 30-40 gas per loop.

**Recommendation:** Use this pattern;

```
for (uint256 i; i < len;) {
  // do some work
  unchecked {
    ++i;
  }
}
```

**Ethena:** Fixed in commit bb12f434.

**Cantina:** Fix verified.

### 3.2.11 Unnecessary Order Type Check In `verifyRoute`

**Severity:** Gas Optimization

**Context:** File.sol#L353-L355

**Description:** The `verifyRoute` function short circuits if the order type is a redemption, as redemptions do not utilize routes. However, `verifyRoute` is only called in `mint`, which confirms the order type is a mint, so this short circuit check is never triggered, making it a waste of gas.

**Recommendation:** Remove this unnecessary check, leaving a comment for documentation if desired.

**Ethena:** Fixed in commit c56a0cd1.

**Cantina:** Fix verified.

#### 3.2.12 `verifyOrder` Always Returns `true`

**Severity:** Gas Optimization

**Context:** EthenMinting.sol#L347

**Description:** The `verifyOrder` function returns a boolean value, but this value is always `true` (the function reverts in any case that returning `false` would be appropriate). Since this function is called on-chain, having this unneeded return value uses unnecessary gas.

**Recommendation:** Remove the unused return value to save gas.

**Ethena:** This function is also used to validate before submitting orders on chain in our market maker api to save gas.

**Spearbit:** The off-chain logic must still be checking whether the function reverts in order to be correct; it should not be hard to adapt it to a version of the function without the boolean return value, although since the gas savings will be minor, it isn't too big of a deal.

**Ethena:** Return value removed in commit 116af39a.

**Cantina:** Fix verified.

#### 3.2.13 Unreachable Revert Statements In EthenaMinting

**Severity:** Gas Optimization

**Context:** EthenaMinting.sol#L172, EthenaMinting.sol#L203

**Description:** Because the `_deduplicateOrder` function always returns `true` (it reverts in any case where it would be appropriate for it to return `false`), the `revert Duplicate();` statements on these lines are unreachable. Thus doing the return value check wastes gas.

**Recommendation:** Invoke `_deduplicateOrder` without checking its return value to reduce gas costs. Alternatively, modify it to return `false` in at least some cases in which it currently reverts (this will not save gas but makes the code logically consistent).

**Ethena:** Fixed in commit 790673ed.

**Cantina:** Fix verified.

#### 3.2.14 Some Storage Variables Could Be `immutable` or `constant` to Reduce Gas Costs

**Severity:** Gas Optimization

**Context:** EthenaMinting.sol#L63, StakedUSDeV2.sol#L20, StakedUSDeV2.sol#L22

**Description:** The indicated contract fields (`usde` in EthenaMinting.sol#L63, `silo` in StakedUSDeV2.sol#L20) are only set in the constructor of the relevant contract, and could thus be designated as `immutable` to reduce both deployment and runtime gas costs. Meanwhile, `MAX_COOLDOWN_DURATION` in StakedUSDeV2.sol#L22 is a know value that can be designated as `constant`.

**Recommendation:** Make these fields `immutable`, or add the ability to change them if such methods were omitted. Make `MAX_COOLDOWN_DURATION` a constant variable.

**Ethena:** Fixed in commit fcfd5e82.

**Cantina:** Fix verified.

## 3.3 Informational

### 3.3.1 Unusual Choice of Indexing For Some Event Parameters

**Severity:** Informational

**Context:** IEthenaMintingEvents.sol#L9-L16, IEthenaMintingEvents.sol#L18-L26, IStakedUSDe.sol#L8, IEthenaMintingEvents.sol#L52-L56

**Description:** Generally, it makes sense to mark event parameters as `indexed` when there is an expectation that consumers of the event will likely only care about the event when that parameter has a certain value. For example, it is logical to make token recipient or source addresses `indexed` in events, as owners of those addresses will want to know when their address gains or loses tokens. It makes less sense to index the amount of tokens transferred, as it is rare to care about all transfers of a specific value.

Some events make unusual choices about indexing. For example, the `Mint`and `Redeem` events in IEthenaMintingEvents.sol index the collateral asset, the collateral amount, and the USDe amount; they leave the minter, benefactor and beneficiary un-indexed. It would be more typical to index the addresses (so involved parties can monitor transactions relevant to them), and to not index the token amounts (which could be confusing if unrelated transactions are being done with identical amounts between different parties; this could even provide a vector for griefing event listeners by intentionally causing them to examine irrelevant events).

Other non-standard choices include indexing the `amount` parameter of the `RewardsReceived` event in IStakedUSDe.sol and the parameters to `MaxMintPerBlockChanged` and `MaxRedeemPerBlockChanged` in IEthenaMintingEvents.sol#L52-L56.

**Recommendation:** Critically examine the decision to mark each event parameter as `indexed` or not with a focus on the intended use-case.

**Ethena:** Fixed in commit 99f33341.

**Cantina:** Fix verified.

### 3.3.2 Shadowing of `owner` variable in `StakedUSDeV2`

**Severity:** Informational

**Context:** StakedUSDeV2.sol#L52, StakedUSDeV2.sol#L65, StakedUSDeV2.sol#L95, StakedUSDeV2.sol#L111

**Description:** The usage of `owner` variable as `calldata` in multiple functions on `StakedUSDeV2` contract shadows the `owner` method from `SingleAdminAccessControl.sol#L65`. This can lead to confusion and potential errors during code maintenance or future development.

**Recommendation:** Rename the `owner` variable declarations in `StakedUSDeV2.sol` to `_owner` to avoid shadowing and potential confusion.

**Ethena:** Fixed in commit bce786de.

**Cantina:** Fix verified.

### 3.3.3 Unused Imports

**Severity:** Informational

**Context:** interfaces/IERC4626Minimal.sol#L4, USDeSilo.sol#L5

**Description:** The indicated imports are unused in their respective files. If they are being used in files that import the files containing them, it makes for more self-documenting code to explicitly import them where they are used. In the case of USDeSilo.sol#L5, `SafeERC20` is not needed anyway as the USDE is ERC20-compliant.

**Recommendation:** Remove unused imports and add them explicitly to any files that are currently importing them transitively.

**Ethena:** Fixed in commit e7d85ecc.

**Cantina:** Fix verified.

### 3.3.4 Unused Errors

**Severity:** Informational

**Context:** interfaces/IEthenaMinting.sol#L47, interfaces/IStakedUSDe.sol#L18

**Description:** The indicated errors (`InvalidAffirmedAmount` in IEthenaMinting and `SlippageExceeded` in IStakedUSDe) are unused.

**Recommendation:** Remove unneeded error definitions to reduce code clutter and the chance of confusion.

**Ethena:** Fixed in commit b910d2d6.

**Cantina:** Fix verified.

### 3.3.5 Unused Events

**Severity:** Informational

**Context:** interfaces/IEthenaMintingEvents.sol#L29, interfaces/IEthenaMintingEvents.sol#L32, interfaces/IStakedUSDeCooldown.sol#L17

**Description:** The `CustodyWalletAdded` and `CustodyWalletRemoved` events in IEthenaMintingEvents.sol are unused and seem to be superseded by the `CustodianAddress*` events. The `SiloUpdated` event in IStakedUSDeCooldown.sol is unused and since there is no functionality for upgrading the silo, it seems that it is unneeded.

**Recommendation:** Remove unused and unneeded events from the interfaces to reduce clutter in the code and the potential for confusion by integrators.

**Ethena:** Fixed in commit 395e1ab7.

**Cantina:** Fix verified.

### 3.3.6 Update references to deprecated `draft-ERC20Permit.sol` in favor of `ERC20Permit.sol`

**Severity:** Informational

**Context:** USDe.sol#L6, StakedUSDe.sol#L11, IUSDe.sol#L6

**Description:** The file `draft-ERC20Permit.sol` is deprecated since `EIP-2612` became final on 2022-11-01.

**Recommendation:** Update references from `..../draft-ERC20Permit.sol` to `..../ERC20Permit.sol` and from `..../draft-IERC20Permit.sol` to `..../IERC20Permit.sol`

**Ethena:** Fixed in commit f175a26e and commit 00e5a2ca.

**Cantina:** Fix verified.

### 3.3.7 Use a constant for magic number `10_000` in `EthenaMinting`

**Severity:** Informational

**Context:** EthenaMinting.sol#L370, EthenaMinting.sol#L425

**Description:** The magic number `10_000` is directly referenced in `EthenaMinting`. Direct usage of such numbers can lead to potential errors if the value needs modification in the future and can also reduce the clarity of the code.

**Recommendation:** Use a constant to represent the value `10_000`.

**Ethena:** Fixed in commit 7a7cd656 and commit 00e5a2ca.

**Cantina:** Fix verified.

### 3.3.8 Misleading Error When Beneficiary Is The Zero Address

**Severity:** Informational

**Context:** EthenaMinting.sol#L343

**Description:** When the `order.beneficiary` field is equal to `address(0)`, the `verifyOrder` function reverts with an `InvalidAmount()` error. This is misleading as the problem is not with a quantity of tokens, but rather an address. This could cause confusion for anyone trying to diagnose why a transaction reverted.

**Recommendation:** Use the `InvalidAddress()` error, which already exists, instead of `InvalidAmount()`.

**Ethena:** Fixed in commit e87f0710.

**Cantina:** Fix verified.