Quantstamp

Ethena Labs

Executive Summary

This audit report was prepared by Quantstamp, the leader in blockchain security.

Type	Decentralized Stablecoin
Timeline	2023-09-20 through 2023-09-28
Language	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	None
Source Code	• ethena-labs/ethena #5f0b385
Auditors	• Michael Boyle Auditing Engineer • Jonathan Mevs Auditing Engineer • Jeffrey Kam Auditing Engineer

Documentation quality	Medium
Test quality	Medium
Total Findings	13 Fixed: 9 Acknowledged: 4
High severity findings	0
Medium severity findings	4 Fixed: 2 Acknowledged: 2
Low severity findings	3 Fixed: 3
Undetermined severity findings	0
Informational findings	6 Fixed: 4 Acknowledged: 2

Summary of Findings

This report presents the results of an audit conducted by the Quantstamp team on the USDe token by Ethena Labs. The USDe token aims to be a decentralized stablecoin backed by interest-bearing tokens and shorts with the equivalent value in a delta-neutral strategy. USDe is a standard ERC-20 token with the OpenZeppelin burnable and permit extensions. While USDe is truly permissionless on-chain, minting and redeeming USDe is handled by Ethena Labs through a centralized smart contract. Owners of the USDe token can choose to stake their USDe in return for stUSDe, which will reward them with some of the profits of the underlying assets.

Users should be aware that trust is required in the Ethena Labs team to manage the underlying financial positions that support the value of USDe. This also requires users to trust the exchanges in which Ethena Labs is creating their positions and that the perpetual shorts can be exercised, even in extreme market turbulence.

The audit uncovered 13 findings, including four issues of medium severity. Our primary concern revolves around the high amount of trust users must have in Ethena Labs to maintain the value of USDe (ETHN-1). Overall, the code is well-written and contains sufficient documentation. It relies heavily on OpenZeppelin Contracts to handle various components of the project such as access control and tokens along with their extensions.

We recommend reviewing this document in detail and fixing all the issues before deploying the code in production.

Update

We would like to highlight the commitment of the team to address all issues. Of the 13 issues raised, nine were fixed and the remaining four were acknowledged with sufficient reasoning. Additionally, updates were made to the documentation to clearly state the trust assumptions of the protocol.

ID	DESCRIPTION	SEVERITY	STATUS
ETHN-1	**Privileged Roles and Ownership**	● Medium	Acknowledged
ETHN-2	**Malicious Users Can Perform Dos Attack by Setting Delegated Signer to Self**	● Medium	Fixed
ETHN-3	**System Supports Withdrawal but Does Not Support Deposit of Native Token**	● Medium	Fixed

ID	DESCRIPTION	SEVERITY	STATUS
ETHN-4	The `SOFT_RESTRICTED_STAKER_ROLE` Can Be Bypassed	● Medium	Acknowledged
ETHN-5	Vesting Rate Can Be Slowed by the Rewarder	● Low	Fixed
ETHN-6	Input Validation	● Low	Fixed
ETHN-7	Ownership Can Be Renounced	● Low	Fixed
ETHN-8	The Return Value of `ecrecover()` Should Be Validated For the Zero Address	● Informational	Fixed
ETHN-9	Potential Index Out of Bounds on Mint Orders	● Informational	Fixed
ETHN-10	Manual Liquidity Control Could Delay USDe Redemptions	● Informational	Acknowledged
ETHN-11	Potential Collateral Sent to Non-Team Controlled Addresses	● Informational	Fixed
ETHN-12	Defining Custom Cryptographic Operations Introduces Unnecessary Risks	● Informational	Fixed
ETHN-13	System Only Support 2^{64} Nonces	● Informational	Acknowledged

Assessment Breakdown

Quantstamp's objective was to evaluate the repository for security-related issues, code quality, and adherence to specification and best practices.

Disclaimer

Only features that are contained within the repositories at the commit hashes specified on the front page of the report are within the scope of the audit and fix review. All features added in future revisions of the code are excluded from consideration in this report.

Possible issues we looked for included (but are not limited to):

- Transaction-ordering dependence
- Timestamp dependence
- Mishandled exceptions and call stack limits
- Unsafe external calls
- Integer overflow / underflow
- Number rounding errors
- Reentrancy and cross-function vulnerabilities
- Denial of service / logical oversights
- Access control
- Centralization of power
- Business logic contradicting the specification
- Code clones, functionality duplication
- Gas usage
- Arbitrary token minting

Methodology

1. Code review that includes the following
   1. Review of the specifications, sources, and instructions provided to Quantstamp to make sure we understand the size, scope, and functionality of the smart contract.
   2. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
   3. Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to Quantstamp describe.
2. Testing and automated analysis that includes the following:
   1. Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run those test cases.
   2. Symbolic execution, which is analyzing a program to determine what inputs cause each part of a program to execute.
3. Best practices review, which is a review of the smart contracts to improve efficiency, effectiveness, clarity, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.

4. Specific, itemized, and actionable recommendations to help you take steps to secure your smart contracts.

Scope

Files Included

Repo: https://github.com/ethena-labs/SmartContracts(56c2494d1efa00e08e63374f667174d519ad33e0) Files: EthenaMinting.sol USDe.sol StakedUSDe.sol

Files Excluded

Repo: https://github.com/ethena-labs/SmartContracts(56c2494d1efa00e08e63374f667174d519ad33e0) Files: Everything else

Findings

ETHN-1 Privileged Roles and Ownership

● Medium Acknowledged

Update

Marked as "Acknowledged" by the client. The client provided the following explanation:

> By design, Ethena has admin privileges. However more detailed documentation will be

File(s) affected: `EthenaMinting.sol`, `StakedUSDeV2.sol`

Description: The proper functioning of the protocol heavily depends on the protocol's hedging strategy. In case of market turmoil, it is possible that the protocol cannot liquidate funds from centralized exchanges, so users might not be able to redeem their full balance. Furthermore, to facilitate users redeeming tokens for their USDe, the protocol admin must first transfer enough funds from the custody back into the Ethena Minting contract. There is no on-chain guarantee that USDe will be redeemable as the team can simply choose not to return the funds. Below we list all the privileged roles in the system.

1. `EthenaMinting`
   1. `DEFAULT_ADMIN_ROLE`
      1. Can set the `maxMintPerBlock`.
      2. Can set the `maxRedeemPerBlock`.
      3. Can add and remove addresses from other roles (excluding `owner`).
   2. `owner`
      1. Can set the USDe token address.
      2. Can add and remove supported assets.
   3. `MINTER_ROLE`
      1. Can mint stablecoins from assets.
      2. **Can transfer any asset in the contract to any address.**
   4. `REDEEMER_ROLE`
      1. Can redeem stablecoins for assets.
   5. `GATEKEEPER_ROLE`
      1. Can disable minting and redeeming.
      2. Can remove the `MINTER_ROLE` from an address.
2. `StakedUSDeV2.sol`
   1. `owner/DEFAULT_ADMIN_ROLE`
      1. **Can add and remove addresses from other roles. This should be the role of the Gatekeeper according to documentation.**
      2. **Can redistribute stUSDe from wallets with the `FULL_RESTRICTED_STAKER_ROLE` to any unrestricted address.**
   2. `REWARDER_ROLE`
      1. Can add vested USDe tokens to the contract via `transferInRewards()`.

In the current state of the system, privileged roles can perform various actions that would be detrimental to the health of USDe. Most notably, the following capabilities could be problematic:

1. The price of an order is not considered on-chain, which means that Ethena Labs could mint any amount of USDe without providing an equal amount of underlying tokens or redeem a small amount of USDe for all of the underlying assets.
2. The underlying assets backing USDe can be withdrawn from the minting contract to any address.
3. stUSDe tokens can be seized at any time by the admin role of the staking contract. There should be a separation of roles between assigning the fully restricted role and redistributing their tokens.
4. Profits from the underlying assets intended for stUSDe holders need to be manually deposited.

Recommendation: Consider taking steps to make each part of the process more transparent and traceable for users. This would include extensive user-facing documentation, dashboards showing the value of the underlying assets along with their associated centralized positions, greater separation of roles, and whitelisted addresses that are approved to custody the underlying assets.

ETHN-2 Malicious Users Can Perform Dos Attack by Setting Delegated Signer to Self

● Medium ⓘ Fixed

Update
Marked as "Fixed" by the client. Addressed in: `e4282dcf1471841f71dd5cb6efe303b104481f92`. The client provided the following explanation:

> Nested mapping structure for delegatedSigner, plus added function to undelegate

File(s) affected: `EthenaMinting.sol`

Description: When `order.benefector` is a smart contract, it can delegate an EOA to be the signer for the contract, by calling `setDelegatedSigner()`. When `mint()` and `redeem()` are called, they first check that the order is valid by calling `validateOrder()`, which in turn checks that the signer of the order is either `msg.sender` or a delegated signer for `msg.sender` (i.e. `delegatedSigner[signer] == order.benefactor`). Suppose contract A delegates Alice to be the signer by calling `delegateSigner(Alice)`, this would result in `delegatedSigner[Alice] = A`. However, since `setDelegatedSigner()` can be called by anyone, this allows a malicious actor, Bob, to block any `mint()` or `redeem()` calls for contract A (signed by Alice) by front-running it with a call `delegateSigner(Alice)`, which would set `delegatedSigner[Alice] = Bob`, causing `verifyOrder()` to fail.

Recommendation: Consider reversing the order of the mapping so that the order benefactor sets the signer. If this goes against the design philosophy, consider creating a nested mapping from the delegated signer to the order benefactor to a boolean. This would prevent a DoS as a single user could have multiple valid delegated signers.

ETHN-3 System Supports Withdrawal but Does Not Support Deposit of Native Token

● Medium ⓘ Fixed

Update
Marked as "Fixed" by the client. Addressed in: `40fcdb40ba46889c6bc4ddcd554637e98bde2f8b`. The client provided the following explanation:

> Added receive()

File(s) affected: `EthenaMinting.sol`

Description: The system does not support using native ETH as collateral to mint new USDe, as shown in `_transferCollateral()`. However, it seems possible for users to redeem native tokens since `_transferToBeneficiary()` supports it. Furthermore, there is no direct way (other than `selfdestruct()`) for the contract to receive native tokens as `fallback()` and `receive()` are not implemented and there is no payable function.

Recommendation: Consider implementing the `receive()` function as a means to deposit the native token into the contract.

ETHN-4 The `SOFT_RESTRICTED_STAKER_ROLE` Can Be Bypassed

● Medium ⓘ Acknowledged

Update
Marked as "Acknowledged" by the client. The client provided the following explanation:

> It is by design that the SOFT_RESTRICTED_STAKER_ROLE can interact with the contract and earn yield by acquiring and selling stUSDe on the open market

File(s) affected: `StakedUSDe.sol`, `StakedUSDeV2.sol`

Description: The `SOFT_RESTRICTED_STAKER_ROLE` is given to addresses that are partially blacklisted from the contract. They are not permited to deposit funds but can still perform all other operations. Therefore, an address with the `SOFT_RESTRICTED_STAKER_ROLE` could transfer their USDe tokens to another address they own and deposit them.

Recommendation: Sybil attacks are difficult to avoid. Consider removing the `SOFT_RESTRICTED_STAKER_ROLE` or further restricting the role.

ETHN-5 Vesting Rate Can Be Slowed by the Rewarder

● Low ⓘ Fixed

Update

Marked as "Fixed" by the client. Addressed in: `4db61cc26152af219e97811c89224c6407bdc96e`. The client provided the following explanation:

> Previous vesting must complete before new rewards can be transferred in

File(s) affected: `StakingUSDe.sol`

Description: The rewarder can artificially reduce the vesting rate by transferring in rewards with `transferInRewards()` while previous rewards are still vesting. For example, let's assume `VESTING PERIOD = 1 hour` and the initial reward is `600`, so the current rate of vesting is `10` tokens per minute. Then, after 30 minutes pass, the rewarder transfers in another `60` token as a reward. This means the unvested amount is now `360`, which leads to a vesting rate of `6` tokens per minute. However, users should be entitled to the `10` tokens per minute rate for the remaining `300` tokens reward from the initial transfer.

Recommendation: Consider modifing the logic to account for the case even when the rewarder transfers rewards in between vesting periods. At the very least, document this behavior to the users.

ETHN-6 Input Validation

• Low ⓘ Fixed

Update

Marked as "Fixed" by the client. Addressed in: `e1e57d7a42cafd3add8b98560da668dab44e7f44`. The client provided the following explanation:

> 1. setUSDe removed, unnecessary. 2. deterministically set in StakedUSDeV2 constructor, 3a. fixed, 3b. will fail due to address(0) check in OZ ERC20 implementation on transfer

File(s) affected: `EthenaMinting.sol`, `USDeSilo.sol`, `StakeUSDeV2.sol`

Description: Consider the following input validation checks:

1. `EthenaMinting.sol`
   1. `setUSDe()`
      1. Check that `_usde` is not the same as `usde`.
2. `StakeUSDeV2.sol`
   1. `setCooldownDuration()`
      1. `duration` should not exceed the specified 90-day maximum cooldown.
   2. `unstake()`
      1. `receiver` should not be the zero address.

Recommendation: Consider adding the recommended checks.

ETHN-7 Ownership Can Be Renounced

• Low ⓘ Fixed

Update

Marked as "Fixed" by the client. Addressed in: `eb4baeec5159d4a174c03f9f617a7ddb40a102ae`. The client provided the following explanation:

> Remove ownership from EthenaMinting and StakedUSDe, and Access Control from USDe, such that each contract only uses one or the other. Extend upon AccessControl and Ownable to prevent renouncing and make ownership transfers 2 step

Description: If the owner renounces their ownership, all ownable contracts will be left without an owner. Consequently, any function guarded by the `onlyOwner` modifier will no longer be able to be executed.

Additionally, the DEFAULT_ADMIN_ROLE can be renounced/revoked from `EthenaMinting` and `StakeUSDe.sol`.

Recommendation: Confirm that this is the intended behavior. If not, override and disable the `renounceOwnership()` and `revokeRole()` functions in the affected contracts. For extra security, consider using a two-step process when transferring the ownership of the contract (e.g. `Ownable2Step` from OpenZeppelin).

ETHN-8 The Return Value of `ecrecover()` Should Be Validated For the Zero Address

• Informational ⓘ Fixed

Update

Marked as "Fixed" by the client. Addressed in: `f6517e6f906e3c8ebda26b308c8bad8524a9748c`. The client provided the following explanation:

> use OZ ECDSA module

File(s) affected: `EthenaMinting.sol`

Description: `ecrecover()` is used to recover the signer of a message. If the recovery fails, `ecrecover()` will return the zero address. This can lead to unintended consequences and therefore, the call should revert if the zero address is returned.

Recommendation: Consider adding a check after the call to `ecrecover()` that reverts if `signer == address(0)`.

ETHN-9 Potential Index Out of Bounds on Mint Orders

● Informational ⓘ Fixed

Update

Marked as "Fixed" by the client. Addressed in: `f6517e6f906e3c8ebda26b308c8bad8524a9748c`. The client provided the following explanation:

> check order type

File(s) affected: `EthenaMinting.sol`

Description: `EthenaMinting.mint()` does not confirm that the order type being processed is indeed of `OrderType.MINT`. In the case where an Order with `OrderType.REDEEM` is passed to this function, `verifyRoute()` will not validate matching lengths of the `route.addresses` and `route.ratios`, which will result in index out of bounds when transferring collateral. This will lead to a failing transaction without adequate failure logging.

Recommendation: In `mint()`, ensure `order.order_type == OrderType.MINT`.

ETHN-10 Manual Liquidity Control Could Delay USDe Redemptions

● Informational ⓘ Acknowledged

Update

Marked as "Acknowledged" by the client. The client provided the following explanation:

> Internally team will maintain ~$200k worth of collateral available for hot redemptions. However team and our custodian does have liquidity control for redemptions

File(s) affected: `EthenaMinting.sol`

Description: Documentation and the code suggest that the `EthenaMinting` contract doesn't hold any underlying LST assets, but rather custodial wallets do. This is evident in the documentation and the `_transferCollateral` function where collateral is transferred elsewhere. As funds are not stored in the contract itself, the team controls the liquidity that is managed in this contract. As a result, there is the potential that there could be friction for USDe redemptions if there is insufficient liquidity in the Ethena Minting contract.

Recommendation: Make clear to the user the schedule for funding the minting smart contract from custodian wallets.

ETHN-11 Potential Collateral Sent to Non-Team Controlled Addresses

● Informational ⓘ Fixed

Update

Marked as "Fixed" by the client. Addressed in: `9f9527a537cd0b0e80f9c5a88cbeae5892d2d570`. The client provided the following explanation:

> Minters can only transfer assets to whitelisted custodian addresses.

Description: There is the potential for collateral to be transferred to addresses that are not controlled by the team as there is no storage in the contract state that contains a set of these addresses that are valid. As a result, there is the potential for USDe-Buy-Orders to be processed that do not transfer the collateral to a team-controlled address. While it is understood that Ethena servers perform validation before sending transactions to the smart contract, we would still like to highlight this possibility.

Recommendation: Consider adding a role specifically for custody and only allowing transfers out to go to addresses with this role. This will both improve security and give users peace-of-mind when funds are moved out of the contract.

ETHN-12 Defining Custom Cryptographic Operations Introduces Unnecessary Risks

● Informational ⓘ Fixed

Update
Marked as "Fixed" by the client. Addressed in: `f6517e6f906e3c8ebda26b308c8bad8524a9748c`. The client provided the following explanation:

> use OZ ECDSA module

File(s) affected: `EthenaMinting.sol`

Description: `getRsv()` is defined in `EthenaMinting.sol` to help unpack `r,s,v` from the signature bytes. It is recommended to use existing battle-tested libraries (e.g. OpenZepplin's ECDSA library) for cryptographic operations due to the high risk of introducing unintended vulnerabilities.

Recommendation: Consider using OpenZepplin's ECDSA library instead of having a custom implementation.

ETHN-13 System Only Support 2^{64} Nonces

● Informational ⓘ Acknowledged

Update
Marked as "Acknowledged" by the client. The client provided the following explanation:

> Won't pose a problem

File(s) affected: `EthenaMinting.sol`

Description: Currently, the nonce implementation in `_deduplicateOrder()` implies that we can only have 2^{64} different slots for a user, where each slot allows 256 bits. However, in the `_orderBitmaps` mapping, the key and value types are of `uint256`, which means different nonces might collide even when they shouldn't (e.g. $2^{64}+1$ and 1).

Recommendation: While it is unlikely for nonces to be this large, we recommend the team consider this issue, document the expected behavior in this scenario, and make appropriate fixes if necessary.

Definitions

- **High severity** – High-severity issues usually put a large number of users' sensitive information at risk, or are reasonably likely to lead to catastrophic impact for client's reputation or serious financial implications for client and users.
- **Medium severity** – Medium-severity issues tend to put a subset of users' sensitive information at risk, would be detrimental for the client's reputation if exploited, or are reasonably likely to lead to moderate financial impact.
- **Low severity** – The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low impact in view of the client's business circumstances.
- **Informational** – The issue does not post an immediate risk, but is relevant to security best practices or Defence in Depth.
- **Undetermined** – The impact of the issue is uncertain.
- **Fixed** – Adjusted program implementation, requirements or constraints to eliminate the risk.
- **Mitigated** – Implemented actions to minimize the impact or likelihood of the risk.
- **Acknowledged** – The issue remains in the code but is a result of an intentional business or design decision. As such, it is supposed to be addressed outside the programmatic means, such as: 1) comments, documentation, README, FAQ; 2) business processes; 3) analyses showing that the issue shall have no negative consequences in practice (e.g., gas analysis, deployment settings).

Code Documentation

- Missing a word in the 4th bullet point of "Please provide a brief summary of the purpose and function of the system" in `AUDIT_MINT.md`.

Adherence to Best Practices

- Stay consistent in the use of custom errors. The following functions use require statements when there should be custom errors for additional gas savings and for consistency with the rest of the code base:
  - `EthenaMinting.redeem()`
  - `EthenaMinting._deduplicateOrder()`
  - `StakedUSDe.constructor()`
  - `StakedUSDeV2.cooldownAssets()`
  - `StakedUSDeV2.cooldownShares()`
  - `USDe.constructor()`
- `EthenaMinting.verifyOrder()` should revert with the error `InvalidAddress()` instead of `InvalidAmount()` when `order.beneficiary == address(0)`.
- Consider declaring `10_000` as a constant in the contract to improve readability.
- Since `packRSV()` is only used in tests, consider moving it out of the main `EthenaMinting.sol` contract.
- Consider checking that `EthenaMinting` has enough of `asset` before transferring in `_transferToBeneficiary()`.
- Emitting an event before changing the state variable will void the need to store the old value in memory, which will save gas.
- The function `ecr()` in `EthenaMinting.sol` is not used. Consider removing it.

Toolset

The notes below outline the setup and steps performed in the process of this audit.

Setup

Tool Setup:

- Slither v0.9.6

Steps taken to run the tools:

1. Install the Slither tool: `pip3 install slither-analyzer`
2. Run Slither from the project directory: `slither .`

Automated Analysis

Slither

Slither output is included in the report's findings or discarded as false positives.

Test Suite Results

Ethena Labs wrote a robust test suite with a relatively large number of tests for the size of the codebase. There are 165 passing tests, some of which use fuzzing.

Update

After the fix review, the test suite now contains 241 passing tests.

```
Running 11 tests for test/foundry/staking/StakedUSDe.ACL.t.sol:StakedUSDeACL
[PASS] testAdminCanCancelTransfer() (gas: 41005)
[PASS] testCanTransferOwnership() (gas: 61491)
[PASS] testCancelTransferAdmin() (gas: 39602)
[PASS] testCorrectSetup() (gas: 12244)
[PASS] testNewOwnerCanPerformOwnerActions() (gas: 108218)
[PASS] testNonAdminCantRenounceRoles() (gas: 44230)
[PASS] testOldOwnerCantPerformOwnerActions() (gas: 92378)
[PASS] testOldOwnerCantTransferOwnership() (gas: 92466)
[PASS] testOwnershipCannotBeRenounced() (gas: 24007)
[PASS] testOwnershipTransferRequiresTwoSteps() (gas: 50082)
[PASS] test_admin_cannot_transfer_self() (gas: 18812)
Test result: ok. 11 passed; 0 failed; 0 skipped; finished in 134.27ms
```

```
Running 28 tests for test/foundry/staking/StakedUSDe.blacklist.t.sol:StakedUSDeBlacklistTest
[PASS] testAdminCannotRenounce() (gas: 23994)
[PASS] testBlackListManagerCannotAddOthers() (gas: 80438)
[PASS] testBlacklistManagerCanBlacklist() (gas: 112974)
[PASS] testBlacklistManagerCanNotBlacklistAdmin() (gas: 68580)
[PASS] testBlacklistManagerCanUnblacklist() (gas: 95304)
[PASS] testBlacklistManagerCannotRedistribute() (gas: 252472)
[PASS] testCanBurnOnRedistribute() (gas: 184288)
[PASS] testOwnerCanRemoveBlacklistManager() (gas: 39556)
[PASS] testStakeFlowCommonUser() (gas: 191773)
[PASS] test_fullBlacklist_can_not_be_transfer_recipient() (gas: 260440)
[PASS] test_fullBlacklist_deposit_reverts() (gas: 138316)
[PASS] test_fullBlacklist_transferFrom_pass() (gas: 201540)
[PASS] test_fullBlacklist_transfer_pass() (gas: 185995)
[PASS] test_fullBlacklist_user_can_not_burn_and_donate_to_vault() (gas: 184165)
[PASS] test_fullBlacklist_withdraw_pass() (gas: 193242)
[PASS] test_grant_role() (gas: 73240)
[PASS] test_redistributeLockedAmount() (gas: 226300)
[PASS] test_renounce_reverts() (gas: 73162)
[PASS] test_revoke_role() (gas: 63230)
[PASS] test_revoke_role_by_myself_reverts() (gas: 136154)
[PASS] test_revoke_role_by_other_reverts() (gas: 138122)
[PASS] test_softBlacklist_deposit_reverts() (gas: 146374)
[PASS] test_softBlacklist_transferFrom_pass() (gas: 222650)
[PASS] test_softBlacklist_transfer_pass() (gas: 199776)
[PASS] test_softBlacklist_withdraw_pass() (gas: 210324)
[PASS] test_softFullBlacklist_deposit_reverts() (gas: 203491)
[PASS] test_softFullBlacklist_transfer_pass() (gas: 241681)
[PASS] test_softFullBlacklist_withdraw_pass() (gas: 261269)
Test result: ok. 28 passed; 0 failed; 0 skipped; finished in 175.83ms

Running 28 tests for test/foundry/staking/StakedUSDeV2.blacklist.t.sol:StakedUSDeV2CooldownBlacklistTest
[PASS] testAdminCannotRenounce() (gas: 24062)
[PASS] testBlackListManagerCannotAddOthers() (gas: 80528)
[PASS] testBlacklistManagerCanBlacklist() (gas: 113154)
[PASS] testBlacklistManagerCanNotBlacklistAdmin() (gas: 68760)
[PASS] testBlacklistManagerCanUnblacklist() (gas: 95449)
[PASS] testBlacklistManagerCannotRedistribute() (gas: 252541)
[PASS] testCanBurnOnRedistribute() (gas: 184342)
[PASS] testOwnerCanRemoveBlacklistManager() (gas: 39646)
[PASS] testStakeFlowCommonUser() (gas: 255916)
[PASS] test_fullBlacklist_can_not_be_transfer_recipient() (gas: 260370)
[PASS] test_fullBlacklist_deposit_reverts() (gas: 138250)
[PASS] test_fullBlacklist_transferFrom_pass() (gas: 201488)
[PASS] test_fullBlacklist_transfer_pass() (gas: 185974)
[PASS] test_fullBlacklist_user_can_not_burn_and_donate_to_vault() (gas: 184144)
[PASS] test_fullBlacklist_withdraw_pass() (gas: 260431)
[PASS] test_grant_role() (gas: 73330)
[PASS] test_redistributeLockedAmount() (gas: 226337)
[PASS] test_renounce_reverts() (gas: 73162)
[PASS] test_revoke_role() (gas: 63339)
[PASS] test_revoke_role_by_myself_reverts() (gas: 136200)
[PASS] test_revoke_role_by_other_reverts() (gas: 138168)
[PASS] test_softBlacklist_deposit_reverts() (gas: 146308)
[PASS] test_softBlacklist_transferFrom_pass() (gas: 222597)
[PASS] test_softBlacklist_transfer_pass() (gas: 199759)
[PASS] test_softBlacklist_withdraw_pass() (gas: 274497)
[PASS] test_softFullBlacklist_deposit_reverts() (gas: 203359)
[PASS] test_softFullBlacklist_transfer_pass() (gas: 241705)
[PASS] test_softFullBlacklist_withdraw_pass() (gas: 345373)
Test result: ok. 28 passed; 0 failed; 0 skipped; finished in 48.12ms

Running 21 tests for test/foundry/staking/StakedUSDe.t.sol:StakedUSDeTest
[PASS] testCanTransferRewardsAfterVesting() (gas: 197189)
[PASS] testCannotStakeWithoutApproval() (gas: 95998)
[PASS] testCannotTransferRewardsWhileVesting() (gas: 148903)
[PASS] testCantWithdrawBelowMinShares() (gas: 207636)
[PASS] testDecimalsIs18() (gas: 5522)
[PASS] testFairStakeAndUnstakePrices() (gas: 355940)
[PASS] testFuzzFairStakeAndUnstakePrices(uint256,uint256,uint256,uint256,uint256) (runs: 256, μ: 516711,
~: 524776)
```

```
[PASS] testFuzzNoJumpInVestedBalance(uint256) (runs: 256, μ: 142969, ~: 142975)
[PASS] testInitialStake() (gas: 149947)
[PASS] testInitialStakeBelowMin() (gas: 155123)
[PASS] testMintToDiffRecipient() (gas: 152042)
[PASS] testMintWithSlippageCheck(uint256) (runs: 256, μ: 190046, ~: 190179)
[PASS] testOnlyRewarderCanReward() (gas: 215958)
[PASS] testOwnerCanChangeRewarder() (gas: 259440)
[PASS] testOwnerCanRescuestUSDe() (gas: 199413)
[PASS] testOwnerCannotRescueUSDe() (gas: 153026)
[PASS] testStakeUnstake() (gas: 185808)
[PASS] testStakingAndUnstakingBeforeAfterReward() (gas: 274037)
[PASS] testTransferRewardsFailsInsufficientBalance() (gas: 147623)
[PASS] testTransferRewardsFailsZeroAmount() (gas: 97756)
[PASS] testUSDeValuePerStUSDe() (gas: 410370)
Test result: ok. 21 passed; 0 failed; 0 skipped; finished in 3.39s

Running 5 tests for test/foundry/minting/tests/EthenaMinting.Delegate.t.sol:EthenaMintingDelegateTest
[PASS] testCanUndelegate() (gas: 114754)
[PASS] testDelegateFailureMint() (gas: 111333)
[PASS] testDelegateFailureRedeem() (gas: 284529)
[PASS] testDelegateSuccessfulMint() (gas: 239828)
[PASS] testDelegateSuccessfulRedeem() (gas: 322700)
Test result: ok. 5 passed; 0 failed; 0 skipped; finished in 54.24ms

Running 19 tests for test/foundry/minting/tests/EthenaMinting.core.t.sol:EthenaMintingCoreTest
[PASS] test_add_and_remove_supported_asset() (gas: 56251)
[PASS] test_cannotAdd_USDe_revert() (gas: 19879)
[PASS] test_cannotAdd_addressZero_revert() (gas: 15679)
[PASS] test_cannot_add_asset_already_supported_revert() (gas: 73300)
[PASS] test_cannot_removeAsset_not_supported_revert() (gas: 19111)
[PASS] test_expired_orders_revert() (gas: 94769)
[PASS] test_fuzz_mint_noSlippage(uint256) (runs: 256, μ: 209502, ~: 209502)
[PASS] test_fuzz_multipleInvalid_custodyRatios_revert(uint256) (runs: 256, μ: 112086, ~: 112109)
[PASS] test_fuzz_singleInvalid_custodyRatio_revert(uint256) (runs: 256, μ: 99673, ~: 99703)
[PASS] test_mint() (gas: 197714)
[PASS] test_multipleValid_custodyRatios_addresses() (gas: 344031)
[PASS] test_nativeEth_withdraw() (gas: 290028)
[PASS] test_receive_eth() (gas: 19191)
[PASS] test_redeem() (gas: 293815)
[PASS] test_redeem_invalidNonce_revert() (gas: 305848)
[PASS] test_sending_mint_order_to_redeem_revert() (gas: 82378)
[PASS] test_sending_redeem_order_to_mint_revert() (gas: 253224)
[PASS] test_unsupported_assets_ERC20_revert() (gas: 166022)
[PASS] test_unsupported_assets_ETH_revert() (gas: 136231)
Test result: ok. 19 passed; 0 failed; 0 skipped; finished in 3.80s

Running 15 tests for test/foundry/minting/tests/USDe.t.sol:MintingBaseSetup
[PASS] testCanCancelOwnershipChange() (gas: 32428)
[PASS] testCanTransferOwnership() (gas: 33318)
[PASS] testCantInitWithNoOwner() (gas: 112039)
[PASS] testCorrectInitialConfig() (gas: 14893)
[PASS] testMinterCanMint() (gas: 63512)
[PASS] testMinterCantMintToZeroAddress() (gas: 13236)
[PASS] testNewMinterCanMint() (gas: 71735)
[PASS] testNewOwnerCanPerformOwnerActions() (gas: 43073)
[PASS] testOldMinterCantMint() (gas: 32248)
[PASS] testOldOwnerCantSetMinter() (gas: 40711)
[PASS] testOldOwnerCantTransferOwnership() (gas: 37440)
[PASS] testOnlyOwnerCanSetMinter() (gas: 20377)
[PASS] testOwnerCantMint() (gas: 17558)
[PASS] testOwnershipCannotBeRenounced() (gas: 19384)
[PASS] testOwnershipTransferRequiresTwoSteps() (gas: 40765)
Test result: ok. 15 passed; 0 failed; 0 skipped; finished in 14.74ms

Running 23 tests for
test/foundry/staking/StakedUSDeV2.cooldownDisabled.t.sol:StakedUSDeV2CooldownDisabledTest
[PASS] testCanTransferRewardsAfterVesting() (gas: 197191)
[PASS] testCannotStakeWithoutApproval() (gas: 95954)
[PASS] testCannotTransferRewardsWhileVesting() (gas: 148885)
[PASS] testCantWithdrawBelowMinShares() (gas: 209858)
[PASS] testDecimalsIs18() (gas: 5611)
[PASS] testFairStakeAndUnstakePrices() (gas: 357709)
```

```
[PASS] testFuzzFairStakeAndUnstakePrices(uint256,uint256,uint256,uint256,uint256) (runs: 256, μ: 519812,
~: 526989)
[PASS] testFuzzNoJumpInVestedBalance(uint256) (runs: 256, μ: 142954, ~: 142959)
[PASS] testInitialStake() (gas: 149925)
[PASS] testInitialStakeBelowMin() (gas: 155079)
[PASS] testMintToDiffRecipient() (gas: 151998)
[PASS] testMintWithSlippageCheck(uint256) (runs: 256, μ: 189901, ~: 190056)
[PASS] testOnlyRewarderCanReward() (gas: 215914)
[PASS] testOwnerCanChangeRewarder() (gas: 259489)
[PASS] testOwnerCanRescuestUSDe() (gas: 199486)
[PASS] testOwnerCannotRescueUSDe() (gas: 153049)
[PASS] testStakeUnstake() (gas: 187524)
[PASS] testStakingAndUnstakingBeforeAfterReward() (gas: 275914)
[PASS] testTransferRewardsFailsInsufficientBalance() (gas: 147646)
[PASS] testTransferRewardsFailsZeroAmount() (gas: 97779)
[PASS] testUSDeValuePerStUSDe() (gas: 412326)
[PASS] test_cooldownAssets_fails_cooldownDuration_zero() (gas: 10723)
[PASS] test_cooldownShares_fails_cooldownDuration_zero() (gas: 10626)
Test result: ok. 23 passed; 0 failed; 0 skipped; finished in 4.15s

Running 54 tests for test/foundry/minting/tests/EthenaMinting.ACL.t.sol:EthenaMintingACLTest
[PASS] testAdminCanCancelTransfer() (gas: 40981)
[PASS] testCanRepeatedlyTransferAdmin() (gas: 46373)
[PASS] testCanTransferOwnership() (gas: 61484)
[PASS] testCancelTransferAdmin() (gas: 39661)
[PASS] testCorrectInitConfig() (gas: 3888356)
[PASS] testNewOwnerCanPerformOwnerActions() (gas: 87705)
[PASS] testNonAdminCanRenounceRoles() (gas: 35040)
[PASS] testOldOwnerCantPerformOwnerActions() (gas: 94590)
[PASS] testOldOwnerCantTransferOwnership() (gas: 92449)
[PASS] testOwnershipCannotBeRenounced() (gas: 24105)
[PASS] testOwnershipTransferRequiresTwoSteps() (gas: 50269)
[PASS] test_admin_can_add_gatekeeper() (gas: 43690)
[PASS] test_admin_can_add_minter() (gas: 42719)
[PASS] test_admin_can_disable_mint(bool) (runs: 256, μ: 37232, ~: 15756)
[PASS] test_admin_can_disable_redeem(bool) (runs: 256, μ: 102358, ~: 15663)
[PASS] test_admin_can_enable_mint() (gas: 212407)
[PASS] test_admin_can_enable_redeem() (gas: 301361)
[PASS] test_admin_can_remove_gatekeeper() (gas: 35942)
[PASS] test_admin_can_remove_minter() (gas: 36028)
[PASS] test_admin_cannot_transfer_self() (gas: 21641)
[PASS] test_base_transferAdmin() (gas: 64000)
[PASS] test_fuzz_nonAdmin_cannot_enable_mint_revert(address) (runs: 256, μ: 145730, ~: 145730)
[PASS] test_fuzz_nonAdmin_cannot_enable_redeem_revert(address) (runs: 256, μ: 319930, ~: 319930)
[PASS] test_fuzz_nonMinter_cannot_transferCustody_revert(address) (runs: 256, μ: 97583, ~: 97583)
[PASS] test_fuzz_nonOwner_cannot_add_supportedAsset_revert(address) (runs: 256, μ: 44328, ~: 44328)
[PASS] test_fuzz_nonOwner_cannot_remove_supportedAsset_revert(address) (runs: 256, μ: 101991, ~: 101991)
[PASS] test_fuzz_notAdmin_cannot_add_gatekeeper(address) (runs: 256, μ: 63799, ~: 63799)
[PASS] test_fuzz_notAdmin_cannot_add_minter(address) (runs: 256, μ: 63781, ~: 63781)
[PASS] test_fuzz_notAdmin_cannot_remove_gatekeeper(address) (runs: 256, μ: 92196, ~: 92196)
[PASS] test_fuzz_notAdmin_cannot_remove_minter(address) (runs: 256, μ: 93221, ~: 93221)
[PASS] test_fuzz_notMinter_cannot_mint(address) (runs: 256, μ: 125063, ~: 125063)
[PASS] test_fuzz_not_gatekeeper_cannot_disable_mintRedeem_revert(address) (runs: 256, μ: 61227, ~: 61227)
[PASS] test_fuzz_not_gatekeeper_cannot_remove_minter_revert(address) (runs: 256, μ: 61726, ~: 61726)
[PASS] test_fuzz_not_gatekeeper_cannot_remove_redeemer_revert(address) (runs: 256, μ: 61748, ~: 61748)
[PASS] test_gatekeeper_can_disable_mintRedeem() (gas: 112844)
[PASS] test_gatekeeper_can_remove_minter() (gas: 19650)
[PASS] test_gatekeeper_can_remove_redeemer() (gas: 19715)
[PASS] test_gatekeeper_cannot_add_minters_revert() (gas: 63012)
[PASS] test_gatekeeper_cannot_enable_mint_revert() (gas: 147769)
[PASS] test_gatekeeper_cannot_enable_redeem_revert() (gas: 321927)
[PASS] test_grantRole_AdminRoleExternally() (gas: 43171)
[PASS] test_grantRole_nonAdminRole() (gas: 43239)
[PASS] test_minter_canTransfer_custody() (gas: 125106)
[PASS] test_redeem_notRedeemer_revert() (gas: 295785)
[PASS] test_renounceRole_AdminRole() (gas: 15620)
[PASS] test_renounceRole_forDifferentAccount() (gas: 15411)
[PASS] test_renounceRole_nonAdminRole() (gas: 33788)
[PASS] test_renounceRole_notAdmin() (gas: 17608)
[PASS] test_revokeRole_AdminRole() (gas: 15560)
[PASS] test_revokeRole_nonAdminRole() (gas: 33894)
[PASS] test_revokeRole_notAdmin() (gas: 44885)
```

```
[PASS] test_revoke_AdminRole() (gas: 17948)
[PASS] test_role_authorization() (gas: 86098)
[PASS] test_transferAdmin_notAdmin() (gas: 40630)
Test result: ok. 54 passed; 0 failed; 0 skipped; finished in 6.08s

Running 28 tests for test/foundry/staking/StakedUSDeV2.cooldownEnabled.t.sol:StakedUSDeV2CooldownTest
[PASS] testCannotStakeWithoutApproval() (gas: 95955)
[PASS] testCantCooldownBelowMinShares() (gas: 258374)
[PASS] testDecimalsIs18() (gas: 5589)
[PASS] testFairStakeAndUnstakePrices() (gas: 420260)
[PASS] testFuzzCooldownAssets(uint256) (runs: 256, μ: 228987, ~: 229203)
[PASS] testFuzzCooldownAssetsUnstake(uint256) (runs: 256, μ: 253160, ~: 253364)
[PASS] testFuzzCooldownShares(uint256) (runs: 256, μ: 226847, ~: 227010)
[PASS] testFuzzFairStakeAndUnstakePrices(uint256,uint256,uint256,uint256,uint256) (runs: 256, μ: 694700,
~: 704136)
[PASS] testFuzzNoJumpInVestedBalance(uint256) (runs: 256, μ: 143031, ~: 143037)
[PASS] testInitialStake() (gas: 149946)
[PASS] testInitialStakeBelowMin() (gas: 155124)
[PASS] testMintToDiffRecipient() (gas: 152018)
[PASS] testMintWithSlippageCheck(uint256) (runs: 256, μ: 189895, ~: 190038)
[PASS] testOnlyRewarderCanReward() (gas: 215954)
[PASS] testOwnerCanChangeRewarder() (gas: 259512)
[PASS] testOwnerCanRescuestUSDe() (gas: 199485)
[PASS] testOwnerCannotRescueUSDe() (gas: 153049)
[PASS] testSetCooldown_error_gt_max() (gas: 16301)
[PASS] testSetCooldown_fuzz(uint24) (runs: 256, μ: 22696, ~: 22696)
[PASS] testSetCooldown_zero() (gas: 22102)
[PASS] testStakeUnstake() (gas: 250001)
[PASS] testStakingAndUnstakingBeforeAfterReward() (gas: 338386)
[PASS] testTransferRewardsFailsInsufficientBalance() (gas: 147646)
[PASS] testTransferRewardsFailsZeroAmount() (gas: 97823)
[PASS] testUSDeValuePerStUSDe() (gas: 532063)
[PASS] test_constructor() (gas: 3874758)
[PASS] test_fails_v1_exit_functions_cooldownDuration_gt_0() (gas: 24808)
[PASS] test_fails_v2_if_set_duration_zero() (gas: 23152)
Test result: ok. 28 passed; 0 failed; 0 skipped; finished in 4.76s

Running 9 tests for
test/foundry/minting/tests/EthenaMinting.blockLimits.t.sol:EthenaMintingBlockLimitsTest
[PASS] test_fuzz_maxMint_perBlock_exceeded_revert(uint256) (runs: 256, μ: 91312, ~: 91312)
[PASS] test_fuzz_maxMint_perBlock_setter(uint256) (runs: 256, μ: 23532, ~: 23532)
[PASS] test_fuzz_maxRedeem_perBlock_exceeded_revert(uint256) (runs: 256, μ: 269515, ~: 269515)
[PASS] test_fuzz_maxRedeem_perBlock_setter(uint256) (runs: 256, μ: 25528, ~: 25528)
[PASS] test_fuzz_mint_maxMint_perBlock_exceeded_revert(uint256) (runs: 256, μ: 91337, ~: 91337)
[PASS] test_fuzz_nextBlock_mint_is_zero(uint256) (runs: 256, μ: 202582, ~: 202582)
[PASS] test_fuzz_nextBlock_redeem_is_zero(uint256) (runs: 256, μ: 293039, ~: 293041)
[PASS] test_multiple_mints() (gas: 249437)
[PASS] test_multiple_redeem() (gas: 449370)
Test result: ok. 9 passed; 0 failed; 0 skipped; finished in 8.13s

Ran 11 test suites: 241 tests passed, 0 failed, 0 skipped (241 total tests)
```

Code Coverage

Overall, the testing coverage is moderate. The line and statement coverage is sufficient, but it is preferred that the function coverage and branch coverage are over 90%.

Update

The addition of new tests increased the test coverage.

File	% Lines	% Statements	% Branches	% Funcs
contracts/EthenaMinting.sol	90.00% (**99**/110)	84.57% (**148**/175)	62.90% (**39**/62)	73.33% (**22**/30)
contracts/SingleAdminAccessControl.sol	100.00% (**15**/15)	100.00% (**17**/17)	100.00% (**6**/6)	100.00% (**7**/7)

File	% Lines	% Statements	% Branches	% Funcs
contracts/StakedUSDe.sol	97.44% (**38/**39)	98.55% (**68/**69)	100.00% (**20/**20)	100.00% (**13/**13)
contracts/StakedUSDeV2.sol	92.31% (**24/**26)	90.00% (**27/**30)	62.50% (**5/**8)	100.00% (**6/**6)
contracts/USDe.sol	100.00% (**5/**5)	100.00% (**6/**6)	100.00% (**2/**2)	100.00% (**3/**3)

Changelog

- 2023-09-28 - Initial report

About Quantstamp

Quantstamp