Ethena UStb Complementary Audit Report

Prepared by Cyfrin

Version 1.0

Lead Auditors

Immeas

October 31, 2024

Contents

1 About Cyfrin

Cyfrin is a Web3 security company dedicated to bringing industry-leading protection and education to our partners and their projects. Our goal is to create a safe, reliable, and transparent environment for everyone in Web3 and DeFi. Learn more about us at cyfrin.io.

2 Disclaimer

The Cyfrin team makes every effort to find as many vulnerabilities in the code as possible in the given time but holds no responsibility for the findings in this document. A security audit by the team does not endorse the underlying business or product. The audit was time-boxed and the review of the code was solely on the security aspects of the solidity implementation of the contracts.

3 Risk Classification

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

4 Protocol Summary

Ethena `UStb` is a new stablecoin developed by Ethena, backed by real-world treasury assets (RWAs) such as BlackRock BUIDL, USDC, and USDT. The pricing of these RWAs is determined off-chain by Ethena. The `UStb` token also includes functionality to limit its transferability across different stages: fully disabled, whitelist-only, or fully enabled.

4.1 Actors and Roles

1. **Actors:**
    - **Ethena:** Manages the contracts and provides pricing.
    - **Institutional users:** Lock up RWAs in `UStbMinting` and receive `UStb`.
2. **Roles:**
    - **UStbMinting:**
        - **MINTER_ROLE:** Allows the holder to call `UStbMinting::mint` to mint `UStb` and lock collateral.
        - **REDEEMER_ROLE:** Allows the holder to call `UStbMinting::redeem` to burn `UStb` and unlock collateral.
        - **COLLATERAL_MANAGER_ROLE:** Allows the holder to transfer collateral from the `UStbMinting` contract.
        - **GATEKEEPER_ROLE:** Allows disabling of minting and redeeming, as well as revoking `MINTER_ROLE`, `REDEEMER_ROLE`, and `GATEKEEPER_ROLE`.
        - **DEFAULT_ADMIN_ROLE:** Allows granting and revoking of other roles, setting of max mint and redeem limits, setting stables delta limit and token types, adding and removing supported assets, whitelisted benefactors, and custodian addresses, and changing the `UStb` token address.
    - **UStb:**

- **MINTER_CONTRACT:** The role that can mint `UStb`, initially granted to the `UStbMinting` contract.
- **BLACKLIST_MANAGER_ROLE:** Allows the addition and removal of users from the blacklist.
- **WHITELIST_MANAGER_ROLE:** Allows the addition and removal of users from the whitelist.
- **BLACKLISTED_ROLE:** Users who are not permitted to transfer `UStb`.
- **WHITELISTED_ROLE:** Users who can transfer `UStb` when the transfer state is set to `WHITELIST_ONLY`.
- **DEFAULT_ADMIN_ROLE:** Allows the granting and revoking of other roles, redistribution of `UStb` from blacklisted users, recovery of tokens stuck in the contract and changing the token transfer state.

4.2 Key Components

1. **UStbMinting:** Entry point for minting and redeeming `UStb` by locking or unlocking collateral tokens.
2. **UStb:** Upgradeable stablecoin contract with configurable transfer restrictions.
3. **Off-Chain Services:** Provides an RFQ (Request for Quote) service that supplies prices for collateral tokens, performs mint and redeem calls, and includes safeguards to prevent abuse.

4.3 Centralization Consideration

The protocol includes centralized accounts (`COLLATERAL_MANAGER_ROLE` and `DEFAULT_ADMIN_ROLE`) that manage collateral for `UStb` tokens. Access to these accounts should follow strict operational security (OpSec) best practices, including robust key and access management, to prevent unauthorized access and maintain asset security.

5 Audit Scope

Cyfrin conducted an audit of Ethena UStb based on the code present in the repository commit hash `d82676f`.

The following contracts were included in the scope of the audit:

```
- contracts/interfaces/ISingleAdminAccessControl.sol
- contracts/lib/Upgrades.sol
- contracts/ustb/IUStb.sol
- contracts/ustb/IUStbDefinitions.sol
- contracts/ustb/IUStbMinting.sol
- contracts/ustb/IUStbMintingEvents.sol
- contracts/ustb/UStb.sol
- contracts/ustb/UStbMinting.sol
- contracts/SingleAdminAccessControl.sol
- contracts/SingleAdminAccessControlUpgradeable.sol
```

6 Executive Summary

Over the course of 8 days, the Cyfrin team conducted an complementary audit on the Ethena UStb smart contracts provided by Ethena. In this period, a total of 7 issues were found.

The audit uncovered no critical, high, or medium issues. The low-severity issues identified were related to storage gaps and specific transfer cases when the state is set to whitelist-only. The forge test suite was extensive and demonstrated good coverage.

Summary

Project Name	Ethena UStb
Repository	ethena-ustb-audit
Commit	d82676fa43ce...
Audit Timeline	Oct 23th - Nov 1st
Methods	Manual Review

Issues Found

Critical Risk	0
High Risk	0
Medium Risk	0
Low Risk	3
Informational	4
Gas Optimizations	0
Total Issues	7

Summary of Findings

[L-1] Lack of storage gap in upgradeable base contract	Open
[L-2] `USTb` cannot be burnt when whitelist is enabled	Resolved
[L-3] Non-whitelisted users can transfer `USTb` via whitelisted intermediaries in `WHITELIST_ENABLED` mode	Resolved
[I-1] Unused empty `foundry.toml` file in `contracts/foundry/`	Open
[I-2] Typos and formatting discrepancies	Open
[I-3] Lack of event emitted on state change	Open
[I-4] Unused events and errors	Open

7 Findings

7.1 Low Risk

7.1.1 Lack of storage gap in upgradeable base contract

Description: To manage access control, Ethena uses a modified version of the OpenZeppelin AccessControl library called `SingleAdminAccessControl`. Since `UStb` is upgradeable, this library has been further modified to function as an upgradeable base contract: SingleAdminAccessControlUpgradeable.

However, it lacks a storage gap at the end. Storage gaps are beneficial because they allow the base contract to add storage variables in the future without "shifting down" all state variables in the inheritance chain.

Impact: Upgrading may introduce storage collisions for inheriting contracts.

Recommended Mitigation: Consider adding a storage gap at the end:

```
uint256[48] private __gap
```

7.1.2 `UStb` cannot be burnt when whitelist is enabled

Description: The new Ethena `UStb` token has three transfer states, one of which is `WHITELIST_ENABLED`. In the `WHITELIST_ENABLED` state, only whitelisted users should be able to send and receive `UStb`. This is enforced through a check in the overridden `_beforeTokenTransfer` method to ensure that the `to` address is whitelisted:

```
if (!hasRole(WHITELISTED_ROLE, msg.sender) || !hasRole(WHITELISTED_ROLE, to) ||
↪  hasRole(BLACKLISTED_ROLE, msg.sender) || hasRole(BLACKLISTED_ROLE, to)){
  revert OperationNotAllowed();
}
```

However, when burning tokens, the `to` address will be `address(0)`, which will prevent burning.

Impact: Whitelisted users will be unable to burn their `UStb` while whitelisting is enabled. This limitation would also prevent them from redeeming their collateral from `UStbMinting`, as that account is whitelisted.

Proof of Concept: The following test can be added in `UStb.allTests.t.sol`:

```
function testBurnStateWhitelistEnabledFail() public {
    // transfer state whitelist only, bob is whitelisted
    vm.startPrank(newOwner);
    UStbContract.updateTransferState(IUStbDefinitions.TransferState.WHITELIST_ENABLED);
    UStbContract.grantRole(WHITELISTED_ROLE, bob);
    vm.stopPrank();

    // bob cannot burn his tokens
    vm.prank(bob);
    vm.expectRevert();
    UStbContract.burn(_transferAmount);
}
```

Recommended Mitigation: Consider adding a check for `to != address(0)` in the whitelist verification, or add `address(0)` as a whitelisted address. However, if `address(0)` is added as whitelisted, it would allow a whitelisted operator to burn tokens on behalf of a non-whitelisted user.

Ethena: Fixed in PR#10

Cyfrin: Verified. Whitelisted users can burn during whitelist only, both directly and though redeem.

7.1.3 Non-whitelisted users can transfer `USTb` via whitelisted intermediaries in `WHITELIST_ENABLED` mode

Description: When transfers are limited to the `WHITELIST_ENABLED` state, only whitelisted users should be able to send and receive `USTb`, as detailed in the `AUDIT.md`:

- **WHITELIST_ENABLED**: Only whitelisted addresses can send and receive this token.

This restriction is enforced in `_beforeTokenTransfer` through a check:

```
if (!hasRole(WHITELISTED_ROLE, msg.sender) || !hasRole(WHITELISTED_ROLE, to) ||
↪  hasRole(BLACKLISTED_ROLE, msg.sender) || hasRole(BLACKLISTED_ROLE, to)){
  revert OperationNotAllowed();
}
```

However, a non-whitelisted user can bypass this restriction by approving a whitelisted user to transfer on their behalf. Since only `msg.sender` and `to` are checked, the `from` address can be any non-blacklisted user.

Impact: This behavior violates the requirement stated in `AUDIT.md`. Consequently, a non-whitelisted address can still send `USTb`, albeit only to a whitelisted receiver. Additionally, it enables non-whitelisted users to redeem through `USTbMinting`, as the `USTbMinting` contract is a whitelisted address.

Proof of Concept: The following test can be added to `USTb.allTest.t.sol`:

```
function testWhitelistedOperatorCanTransferNonWhitelistedTokens() public {
    // transfer state whitelist only, bob is whitelisted
    vm.startPrank(newOwner);
    USTbContract.updateTransferState(IUSTbDefinitions.TransferState.WHITELIST_ENABLED);
    USTbContract.grantRole(WHITELISTED_ROLE, bob);
    vm.stopPrank();

    // non-whitelisted user approves whitelisted operator
    vm.prank(greg);
    USTbContract.approve(bob, _transferAmount);

    // whitelisted operator can transfer non-whitelisted user's tokens
    vm.prank(bob);
    USTbContract.transferFrom(greg, bob, _transferAmount);
}
```

Recommended Mitigation: Consider verifying that the `from` address is also whitelisted, similar to how blacklisted addresses are handled in the `FULLY_ENABLED` state.

Ethena: Fixed in PR#10

Cyfrin: Verified. `from` is not required to have role `WHITELISTED_ROLE`

7.2 Informational

7.2.1 Unused empty `foundry.toml` file in `contracts/foundry/`

Description: In the project root, there is a folder named `/foundry` that contains only an empty file, `foundry.toml`. Consider removing this folder if it is unused.

7.2.2 Typos and formatting discrepancies

Description: The following typos where found in the code comments:

- `h2olds` -> `holds`
- `enabeld` -> `enabled`
- `SingleAdminAccessControl` -> `SingleAdminAccessControlUpgradeable`

Also a minor formatting discrepancy:

- Lack of space between closing parenthesis and opening curly bracket here and here.

7.2.3 Lack of event emitted on state change

Description: `UStbMinting` includes a safeguard (`stablesDeltaLimit`) to manage stablecoin value fluctuations. The `DEFAULT_ADMIN_ROLE` can adjust this limit via the `setStablesDeltaLimit` function, shown here:

```
function setStablesDeltaLimit(uint128 _stablesDeltaLimit) external onlyRole(DEFAULT_ADMIN_ROLE) {
  stablesDeltaLimit = _stablesDeltaLimit
}
```

However, this function does not emit an event. Given that recent changes to the deployed EthenaMinting contract added events for `setGlobalMaxMintPerBlock`, `setGlobalMaxRedeemPerBlock`, and `disableMintRedeem`, we recommend adding an event in `setStablesDeltaLimit` as well."

7.2.4 Unused events and errors

Description: The following events and errors in `IUStbDefinitions.sol` are unusused:

- `event MinterAdded`
- `event MinterRemoved`
- `event ToggleTransfers`
- `error CantRenounceOwnership` (only used in test)

Consider using or removing them.