Ethena Security Review

Pashov Audit Group

Conducted by: Said, 0xCiphky, ZanyBonzy

August 31st 2024 - September 2nd 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** repository was done by **Pashov Audit Group**, with a focus on the security aspects of the application's smart contracts implementation.

4. About StakedENA

The `StakedENA` contract enables users to stake ENA tokens, manage reward distribution with vesting periods, and includes features like cooldown periods and blacklist management.

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 - a4afb516085b2870ff66f77c0cb7c77c0d0cf75f

fixes review commit hash - a8cd4217edb234a30f13cf983458b997bd0fae24

Scope

The following smart contracts were in scope of the audit:

- `StakedENA`
- `DeployStakedENA`

7. Executive Summary

Over the course of the security review, Said, 0xCiphky, ZanyBonzy engaged with Ethena to review StakedENA. In this period of time a total of **7** issues were uncovered.

Protocol Summary

Protocol Name	StakedENA
Repository	https://github.com/ethena-labs/ethena
Date	August 31st 2024 - September 2nd 2024
Protocol Type	Synthetic Dollar Protocol

Findings Count

Severity	**Amount**
Low	7
Total Findings	**7**

Summary of Findings

ID	Title	Severity	Status
[L-01]	Not immediately setting the correct _INITIAL_REWARDER address during the deployment	Low	Resolved
[L-02]	Unnecessary checking if caller is blacklisted inside _withdraw	Low	Resolved
[L-03]	redistributeLockedAmount does not check if the from balance is non-zero	Low	Resolved
[L-04]	initialize does not trigger __ReentrancyGuard_init	Low	Resolved
[L-05]	Blacklisted shares Continue Earning Rewards	Low	Acknowledged
[L-06]	sENA permit signatures cannot be canceled before deadlines	Low	Resolved
[L-07]	No way to cancel/reduce cooldown requests	Low	Acknowledged

8. Findings

8.1. Low Findings

[L-01] Not immediately setting the correct `_INITIAL_REWARDER` address during the deployment

In the `DeployStakedEna` deployment script, the configured `_INITIAL_REWARDER` is not a valid address, requiring an additional step to grant the `REWARDER_ROLE` so that the actual rewarder can add rewards to the deployed `StakedENA`.

```
contract DeployStakedEna is Script, DeploymentUtils {
  struct StakedENADeploymentAddresses {
    address proxyAddress;
    address stakedEnaImplementation;
    address admin;
    address proxyAdminAddress;
  }

      // We intentionally want our protocol multisig to control upgrades, as
      // well as admin functionality on the contract
      // It is well protected with distributed hardware keys and various other
      // security measures
      // Eventually, a timelock on upgrades may be added but this is out of
      // scope for now in the interest of time, and
      // because governance is not yet fully decentralized
      address constant _DESIRED_PROXY_ADMIN_AND_STAKED_ENA_OWNER = address
        (0x3B0AAf6e6fCd4a7cEEf8c92C32DFeA9E64dC1862);
      // @audit - should just put correct rewarder here
>>>   address constant _INITIAL_REWARDER = address
  (0x0000000000000000000000000000000000000002);
      address constant _ENA_ADDRESS = address
        (0x57e114B691Db790C35207b2e685D4A43181e6061);

      ProxyAdmin proxyAdminContract;

      function run() public virtual {
        uint256 deployerPrivateKey = vm.envUint
          ("STAKED_ENA_DEPLOYER_PRIVATE_KEY");
        deployment(deployerPrivateKey);
      }
      // ...
}
```

Consider setting the correct address as `_INITIAL_REWARDER` to eliminate the unnecessary additional step after the deployment process.

[L-02] Unnecessary checking if `caller` is blacklisted inside `_withdraw`

When `_withdraw` is triggered, it checks whether the `caller` is blacklisted.

```
function _withdraw(
    addresscaller,
    addressreceiver,
    address_owner,
    uint256assets,
    uint256shares
  )
    internal
    override
    nonReentrant
    notZero(assets)
    notZero(shares)
  {
    // @audit - caller check is unnecessary
>>> if (hasRole(BLACKLISTED_ROLE, _owner) || hasRole
  (BLACKLISTED_ROLE, caller) || hasRole(BLACKLISTED_ROLE, receiver)) {
      revert OperationNotAllowed();
    }

    super._withdraw(caller, receiver, _owner, assets, shares);
    _checkMinShares();
  }
```

However, `_withdraw` eventually triggers `_burn`, which calls `_beforeTokenTransfer` where the `msg.sender` or `caller` blacklist check is also performed.

```
function _beforeTokenTransfer
    (address from, address to, uint256) internal virtual override {
>>> if (hasRole(BLACKLISTED_ROLE, msg.sender)) {
      revert OperationNotAllowed();
    }
    if (hasRole(BLACKLISTED_ROLE, from) && to != address(0)) {
      revert OperationNotAllowed();
    }
    if (hasRole(BLACKLISTED_ROLE, to)) {
      revert OperationNotAllowed();
    }
  }
```

Consider removing the unnecessary `caller` blacklist check inside `_withdraw`.

[L-03] `redistributeLockedAmount` does not check if the `from` balance is non-zero

Across `StakedENA` functions, especially `_deposit` and `transferInRewards`, there is always a check to ensure the provided amount is non-zero. However, this check is missing in `redistributeLockedAmount`. If `redistributeLockedAmount` is called with a from the balance of 0, it will set `vestingAmount` to 0 and update `lastDistributionTimestamp` when the to address is the zero address. If the `to` address is non-zero, it will mint 0 shares to that address.

```
function redistributeLockedAmount(
    addressfrom,
    addressto
  ) external nonReentrant onlyRole(DEFAULT_ADMIN_ROLE
    if (!hasRole(BLACKLISTED_ROLE, from) || hasRole
      (BLACKLISTED_ROLE, to)) revert OperationNotAllowed();
   // @audit - no check if amountToDistribute is non-zero
    uint256 amountToDistribute = balanceOf(from);
    uint256 enaToVest = previewRedeem(amountToDistribute);
    _burn(from, amountToDistribute);
    // to address of address(0) enables burning
    if (to == address(0)) {
      _updateVestingAmount(enaToVest);
    } else {
      _mint(to, amountToDistribute);
    }

    emit StakedENARedistributed(from, to, amountToDistribute);
  }
```

Consider adding a non-zero check to match the safeguards in other functions

[L-04] `initialize` does not trigger `__ReentrancyGuard_init`

`StakedENA` uses `ReentrancyGuardUpgradeable` but does not trigger `__ReentrancyGuard_init`. While this does not affect reentrancy protection, it is considered best practice to trigger the initializer for upgradable contracts.

[L-05] Blacklisted shares Continue Earning Rewards

The `redistributeLockedAmount` function handles blacklisted users' shares by burning them and potentially initiating a new vesting period via `_updateVestingAmount`. However, because `_updateVestingAmount` can only be called once the most recent vesting period has ended, the shares of blacklisted users continue to earn rewards until the end of the vesting period.

```
function redistributeLockedAmount(
    addressfrom,
    addressto
  ) external nonReentrant onlyRole(DEFAULT_ADMIN_ROLE
    if (!hasRole(BLACKLISTED_ROLE, from) || hasRole
      (BLACKLISTED_ROLE, to)) revert OperationNotAllowed();
    uint256 amountToDistribute = balanceOf(from);
    uint256 enaToVest = previewRedeem(amountToDistribute);
    _burn(from, amountToDistribute);
    // to address of address(0) enables burning
    if (to == address(0)) {
      _updateVestingAmount(enaToVest);
    } else {
      _mint(to, amountToDistribute);
    }

    emit StakedENARedistributed(from, to, amountToDistribute);
  }
```

Although the assets from blacklisted users are eventually redistributed in a future period, the inability to immediately burn the shares and halt their reward accrual upon blacklisting, reduces the potential rewards earned by legitimate users during the current vesting period.

A possible alternative could be to immediately burn and move the shares to a holding contract. This would prevent blacklisted shares from earning rewards and ensure that the reward pool is preserved for legitimate users during the vesting period.

[L-06] sENA permit signatures cannot be canceled before deadlines

sENA permit signature offers the signer the option to create an EIP-712 signature. After signing this signature, a signer might want to cancel it, but will not be able to do so. This is because the contract is based on OpenZeppelin's ERC20PermitUpgradeable.sol in which the function to increase nonce does not exist and the `_useNonce` function is marked internal.

Recommend introducing a public function that signers can directly use to consume their nonce, thereby canceling the signatures.

[L-07] No way to cancel/reduce cooldown requests

Cooldown of assets and shares can only ever increase, both in time and in assets. If a user changes his mind about cooling down, there's no way for him to cancel this request.

```
function cooldownAssets(uint256 assets) external ensureCooldownOn returns
    (uint256 shares) {
//...
    cooldowns[msg.sender].cooldownEnd = uint104
      (block.timestamp) + cooldownDuration;
    cooldowns[msg.sender].underlyingAmount += uint152(assets);
//...
  }

  function cooldownShares(uint256 shares) external ensureCooldownOn returns
    (uint256 assets) {
//...
    cooldowns[msg.sender].underlyingAmount += uint152(assets);
//...
  }
```

Recommend introducing a way to do this, possibly a bool parameter that increases or decreases the `cooldowns.underlyingAmount` depending on which the user selects.