Ethena Timelock Audit Report

Prepared by Cyfrin

Version 2.0

Lead Auditors

Dacian

Hans

May 16, 2025

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 operates a crypto-backed synthetic dollar protocol including both on-chain and off-chain components.

The contract under audit, `EthenaTimelockController`, is designed to act as a "timelock" between Ethena's multi-sig and protocol contracts. This ensures that:

- if the multisig should be compromised, there is a minimum time delay until it can call sensitive protocol functions
- protocol users have visibility of future multisig actions and time to scrutinize them, building user confidence and trust

The contract does allow for "whitelisted" executors who can call whitelisted `contract::function` pairs bypassing the timelock mechanism, but the intention is to only use the "whitelist" feature for low-risk frequent actions.

Actors:

- Proposers - can propose new transactions and cancel pending transactions
- Executors - can execute transactions which have not been executed but have served the minimum time delay
- Whitelisted Executors - can call whitelisted `contract::function` pairs bypassing the timelock mechanism

The contract is setup with no owner or admin, and `DEFAULT_ADMIN_ROLE` granted only to itself. Hence after the initial setup the contract's configuration is controlled via the contract itself; Proposers must propose any changes which can then either be cancelled by any Proposer or executed by an Executor once the minimum time delay has been served.

The protocol does control who the initial Proposers and Executors are.

5 Audit Scope

The audit scope was limited to one contract `EthenaTimelockController.sol`.

6 Executive Summary

Over the course of 2 days, the Cyfrin team conducted an audit on the Ethena Timelock smart contracts provided by Ethena. In this period, a total of 9 issues were found.

The contract was very small and had a comprehensive test suite so we only found 4 Lows and the rest were gas optimizations and informational findings. Of the 4 Lows:

- 7.1.1 could result in eth being temporarily stuck in the contract
- 7.1.2 is a way to evade the re-entrancy protection but with no further impact
- 7.1.3 is a bug we discovered and reported in OpenZeppelin's `TimelockController` contract which can result in the incorrect assumption that a timelock function call was successfully executed and in a worst-case scenario lose user eth
- 7.1.4 allows whitelisting to be configured for non-existent contracts

Summary

Project Name	Ethena Timelock
Repository	timelock-contract
Commit	7ef32144f6c3...
Fix Commit	7f02ab09de07...
Audit Timeline	May 12th - May 13th, 2025
Methods	Manual Review

Issues Found

Critical Risk	0
High Risk	0
Medium Risk	0
Low Risk	4
Informational	4
Gas Optimizations	1
Total Issues	9

Summary of Findings

[L-1] Only allow execution if value parameters match `msg.value` to prevent eth remaining in the `EthenaTimelockController` contract	Resolved
[L-2] Re-entrancy protection can be evaded via `TimelockController::executeBatch`	Resolved

[L-3] `TimelockController` won't revert when executing on non-existent contracts	Resolved
[L-4] `EthenaTimelockController::addToWhitelist` and `removeFromWhitelist` don't revert for non-existent `target` address	Resolved
[I-1] Use named imports	Resolved
[I-2] Use named mappings	Resolved
[I-3] Don't allow initially granting `EXECUTOR_ROLE` or `WHITELISTED_EXECUTOR_ROLE` to `address(0)`	Acknowledged
[I-4] Only emit events if state actually changes	Acknowledged
[G-1] Use `ReentrancyGuardTransient` for more efficient `nonReentrant` modifiers	Resolved

7 Findings

7.1 Low Risk

7.1.1 Only allow execution if value parameters match `msg.value` to prevent eth remaining in the `EthenaTimelockController` contract

Description: `EthenaTimelockController::execute` and `executeWhitelistedBatch` allow execution without checking that the `msg.value` is equal to the input `value`/`values` parameters.

This can result in eth being temporarily stuck in the contract, though it can be "rescued" by doing a follow-up execution with zero `msg.value` but non-zero `value` input.

Recommended Mitigation: Enforce an invariant that the `EthenaTimelockController` should never finish a transaction with a positive ETH balance by:

- in `execute` revert if `msg.value != value`
- in `executeWhitelistedBatch` revert if `msg.value != sum(values)`

The idea being that every execution should use all of the input `msg.value` and no eth from any execution should remain in the `EthenaTimelockController` contract.

Ethena: Fixed in commit 89d4190 by enforcing this invariant in `execute`, `executeBatch` and `executeWhitelistedBatch`.

Cyfrin: Verified.

7.1.2 Re-entrancy protection can be evaded via `TimelockController::executeBatch`

Description: `EthenaTimelockController::execute` overrides `TimelockController::execute` and adds a `nonReentrant` modifier to prevent re-entrant calls back into it.

However `TimelockController::executeBatch` is not overridden so re-entrancy can still occur that way. Beyond the re-entrancy evasion this doesn't appear further exploitable.

Proof Of Concept: In `test/EthenaTimelockController.t.sol`, change `MaliciousReentrant::maliciousExecute` to:

```
function maliciousExecute() external {
    if (!reentered) {
        reentered = true;
        // re-enter the timelock through executeBatch
        bytes memory data = abi.encodeWithSignature("maliciousFunction()");
        address[] memory targets = new address[](1);
        targets[0] = address(this);
        uint256[] memory values = new uint256[](1);
        values[0] = 0;
        bytes[] memory payloads = new bytes[](1);
        payloads[0] = data;
        timelock.executeBatch(targets, values, payloads, bytes32(0), bytes32(0));
    }
}
```

Then run the relevant test: `forge test --match-test testExecuteWhitelistedReentrancy -vvv` and see that the test fails because the expected re-entrancy error no longer gets thrown.

Recommended Mitigation: Override `TimelockController::executeBatch` in `EthenaTimelockController` to add `nonReentrant` modifier then call the parent function.

Ethena: Fixed in commit 89d4190.

Cyfrin: Verified.

7.1.3 `TimelockController` won't revert when executing on non-existent contracts

Description: `TimelockController::_execute` does this:

```
function _execute(address target, uint256 value, bytes calldata data) internal virtual {
    (bool success, bytes memory returndata) = target.call{value: value}(data);
    Address.verifyCallResult(success, returndata);
}
```

If `target` is a non-existent contract but `data` contains a valid expected function call with parameters, the `call` will return `true`; `Address.verifyCallResult` fails to catch this case.

Proof Of Concept: Add PoC function to `test/EthenaTimelockController.sol`:

```
function testExecuteNonExistentContract() public {
    bytes memory data = abi.encodeWithSignature("DONTEXIST()");
        _scheduleWaitExecute(address(0x1234), data);
}
```

Run with: `forge test --match-test testExecuteNonExistentContract -vvv`

Recommended Mitigation: We reported this bug to OpenZeppelin but they said they prefer the current implementation as it is more flexible. We disagree with this assessment and believe it is incorrect for `TimelockController::_execute` to not revert when there is valid calldata but the target has no code.

Ethena: Fixed in commit e58c547 by overriding `_execute` to revert if `data.length > 0 && target.code.length == 0`.

Cyfrin: Verified.

7.1.4 `EthenaTimelockController::addToWhitelist` and `removeFromWhitelist` don't revert for non-existent `target` address

Description: `EthenaTimelockController::addToWhitelist` and `removeFromWhitelist` should revert if the `target` address doesn't exist (has no code).

Proof of Concept: Add PoC function to `test/EthenaTimelockController.t.sol`:

```
function testAddNonExistentContractToWhitelist() public {
    bytes memory addToWhitelistData = abi.encodeWithSignature(
        "addToWhitelist(address,bytes4)", address(0x1234), bytes4(keccak256("DONTEXIST()"))
    );
    _scheduleWaitExecute(address(timelock), addToWhitelistData);
}
```

Run with: `forge test --match-test testAddNonExistentContractToWhitelist -vvv`

Recommended Mitigation: Revert if `target.code.length == 0`.

Ethena: Fixed in commit 89d4190 to not allow whitelisting of targets with no code.

Cyfrin: Verified.

7.2 Informational

7.2.1 Use named imports

Description: Use named imports:

```
- import "@openzeppelin/contracts/governance/TimelockController.sol";
- import "@openzeppelin/contracts/utils/ReentrancyGuard.sol";
+ import {TimelockController, Address} from "@openzeppelin/contracts/governance/TimelockController.sol";
+ import {ReentrancyGuard} from "@openzeppelin/contracts/utils/ReentrancyGuard.sol";
```

Ethena: Fixed in commit 89d4190.

Cyfrin: Verified.

7.2.2 Use named mappings

Description: Use named mappings to explicitly indicate the purpose of keys and values:

```
-    mapping(address => mapping(bytes4 => bool)) private _functionWhitelist;
+    mapping(address target => mapping(bytes4 selector => bool allowed)) private _functionWhitelist;
```

Ethena: Fixed in commit 89d4190.

Cyfrin: Verified.

7.2.3 Don't allow initially granting `EXECUTOR_ROLE` or `WHITELISTED_EXECUTOR_ROLE` to `address(0)`

Description: The client has stated that initially they want the `EXECUTOR_ROLE` to be closed and that in the future they may open this up.

Hence `EthenaTimelockController::constructor` should revert if any elements in the `executors` or `whitelistedExecutors` input arrays is `address(0)`.

Ethena: Acknowledged; we prefer to keep the optionality here.

7.2.4 Only emit events if state actually changes

Description: A number of functions in `EthenaTimelockController` will emit events even if the state did not change since they simply write to storage but don't read the current storage value to check if it is changing.

Ideally these functions would revert or at least not emit events if the state did not change:

- `addToWhitelist`
- `removeFromWhitelist`

Ethena: Acknowledged.

7.3 Gas Optimization

7.3.1 Use `ReentrancyGuardTransient` for more efficient `nonReentrant` modifiers

Description: Use `ReentrancyGuardTransient` for more efficient `nonReentrant` modifiers:

```
- import "@openzeppelin/contracts/utils/ReentrancyGuard.sol";
+ import {ReentrancyGuardTransient} from "@openzeppelin/contracts/utils/ReentrancyGuardTransient.sol";

- contract EthenaTimelockController is TimelockController, ReentrancyGuard {
+ contract EthenaTimelockController is TimelockController, ReentrancyGuardTransient {
```

Ethena: Fixed in commit 89d4190.

Cyfrin: Verified.