RISK AUDIT

for

YPP Foundation

on

Aug 14, 2025

FIDESIUM

Executive Summary

Report

37
/100

TOTAL

Low risk

Aug 14, 2025

Abstract

Fidesium's automated risk assessment service was requested to perform a risk posture audit on Eczodex **contracts**

Repository Link: https://github.com/Eczodex/ypp-protocol

Initial Commit Hash:

`759c9611ed224cb6b5ede4c168c86e15cca83c65`

Included Contracts:

- YppBootstrapLiquidity.sol
- YppCouncilToken.sol
- YppGoverningCouncil.sol
- YppMintContract.sol
- mintUscyRewards.sol
- teamRewards.sol
- uscyLiquidityRewards.sol

Issue Summary

Critical	High	Medium	Low	Info
1 Issues	2 Issues	8 Issues	1 Issues	2 Issues

Caveats

Eczodex's codebase is generally well written, but does incur a handful of flaws.

Test Approach

Fidesium performed both Whitebox and Blackbox testing, as per the scope of the engagement, and relied on automated security testing.

Methodology

The assessment methodology covered a range of phases and employed various tools, including but not limited to the following:

- Mapping Content and Functionality of API
- Application Logic Flaws
- Access Handling
- Authentication/Authorization Flaws
- Brute Force Attempt
- Input Handling
- Source Code Review
- Fuzzing of all input parameter
- Dependency Analysis

Severity Definitions

Critical	The issue can cause large economic losses, large-scale data disorder or loss of control of authority management.
High	The issue puts users' sensitive information at risk or is likely to lead to catastrophic financial implications.
Medium	The issue puts a subset of users' sensitive information at risk, reputation damage or moderate financial impact.
Low	The risk is relatively small and could not be exploited on a recurring basis, or is low-impact to the client's business.
Informational	The issue does not pose an immediate risk but is relevant to security best practices or defence in Depth.

Risk Issues

Vulnerability	Description	Risk	Probability	Status
Whitelist bypass: Unrestricted `_update`	`YppCouncilToken` allows unrestricted transfer despite whitelist	Critical	Highly Probable	Active
Quorum Manipulation	`YppGoverningCouncil` hardcodes a manipulable quorum	High	High	Active
Merkle Root Overwrites	`uscyLiquidityRewardsContract` and `teamRewards` contracts allow overwriting of Merkle Roots	High	Medium	Active
Precision Loss: Integer Division	Precision Loss: Integer Division	Medium	High	Active
Missing Zero Address Check	Multiple contracts are missing zero address checks	Medium	Medium	Active
Missing Contract Validation	`YppGoverningCouncil` presumes contracts are set correctly	Medium	Medium	Active
Missing Contract Validation	`uscyMintRewardsContract` presumes contracts are set correctly	Medium	Medium	Active
Missing Contract Validation	`YppBootstrapLiquidity` presumes contracts are set correctly	Medium	Medium	Active
Centralization	Priviliged roles have significant modification rights over the contracts and their state.	Medium	Low	Active
Unbounded Array Growth	`YppMintContract.rewardPoolAddresses` grows without bounds or removal mechanisms	Medium	Low	Active
Missing Bounds Validation	`YppMintContract` does not sufficiently validate bounds	Medium	Low	Active
Gas Vulnerability: Unbounded external calls	`YppBootstrapLiquidity` makes contract calls without gas bounds	Low	Low	Active
Gas Inefficiency: Missing Immutable	`YppBootstrapLiquidity.yppMintImplementation` is set once in constructor but is not immutable	Info	Info	Active
Gas Inefficiency: Runtime computation of constant	`YppBootstrapLiquidity` computes a known constant at runtime	Info	Info	Active

Risk Overview

Team Risk

Low risk: 1

No issues found in founding team

Doxxing Status	Team Experience	Risk Summary
Public	Highly relevant	Low

Smart Contract Risks

Risk summary: 43

The contracts are well written, and secure with only a few minor issues..

Vulnerabilities Critical

Whitelist bypass: Unrestricted `_update`

Vulnerability severity: **Critical**

Vulnerability probability: **Highly Probable**

`YppCouncilToken` allows unrestricted transfer despite whitelist

`_update` is caled in the ERC-20 framework on every movement of tokens and is the only place where these transfers can be blocked.

Recommendations:

Implement whitelist checks on transfer in `_update` to avoid voting rights acrruing to non whitelisted entities

```
function _update(address from, address to, uint256 value) internal override {
    if (from != address(0) && to != address(0)) { // Not mint/burn
        require(isWhitelisted(to), "Recipient not whitelisted");
    }
    super._update(from, to, value);
}
```

Vulnerabilities High

Quorum Manipulation

Vulnerability severity: **High**

Vulnerability probability: **Medium**

`YppGoverningCouncil` hardcodes a manipulable quorum

Quorum is hardcoded at a 1 vote, which can be trivially manipulated by a single rogue or compromised council member

Recommendations:

- Hardcode a higher quorum setting

Merkle Root Overwrites

Vulnerability severity: **High**

Vulnerability probability: **Medium**

`uscyLiquidityRewardsContract` and `teamRewards` contracts allow overwriting of Merkle Roots

This could allow authorized parties, through malice, accident, or compromise, to render rewards unclaimable.

Recommendations:

Once claims on a root begin, make that root immutable:

```
require(!hasClaimsStarted[hashedUUID], "Claims already started for this UUID");
```

Vulnerabilities Medium

Precision Loss: Integer Division

Vulnerability severity: **Medium**

Vulnerability probability: **High**

Precision Loss: Integer Division

The `teamRewards` contract uses integer division to calculate team vesting. This can lead to lost assets due to precision loss to rounding

Recommendations:

Precalculate unlocks based on number of expected vestings, and allow any remainder to be carried over to the final vest

```
uint256 baseAmount = allocation / unlockPeriod;
uint256 remainder = allocation % unlockPeriod;
```

Centralization

Vulnerability severity: **Medium**

Vulnerability probability: **Low**

Priviliged roles have significant modification rights over the contracts and their state.

- `YppBootstrapLiquidity.DEFAULT_ADMIN_ROLE`
- `YppCouncilToken.DEFAULT_ADMIN_ROLE`
- `teamRewards.DEFAULT_ADMIN_ROLE`
- `uscyMintRewardsContract.DEFAULT_ADMIN_ROLE`
- `uscyLiquidityRewardsContract.DEFAULT_ADMIN_ROLE`
- `YppBootstrapLiquidity.deployer`

Recommendations:

- Ensure wallets that hold these roles are controlled by well managed multisigs

Missing Bounds Validation

Vulnerability severity: **Medium**

Vulnerability probability: **Low**

`YppMintContract` does not sufficiently validate bounds

`updateAndAdjust` should validate bounds on `newMA`

Recommendations:

`require(newMA <= MAX_REASONABLE_MA)`

Vulnerabilities Medium

Unbounded Array Growth

Vulnerability severity: **Medium**

Vulnerability probability: **Low**

`YppMintContract.rewardPoolAddresses` grows without bounds or removal mechanisms

This can lead to governance attacks, economic manipulation, operational paralysis, state bloat, and in extreme cases DoS and total protocol failure

Recommendations:

- Implement pool recycling mechanisms
- Implement pagination
- Implement emergency controls

```
mapping(address => uint256) public poolIndex;
uint256 public activePoolCount
...
function whitelistRewardContract(
    ...
    bool recycledSlot = false;
    ...
    for (uint256 i = 0; i < rewardPoolAddresses.length; i++) {
        address existingAddress = rewardPoolAddresses[i];
        if (existingAddress != address(0) &&
            !rewardContracts[existingAddress].active &&
            rewardContracts[existingAddress].unclaimedRewards == 0 &&
            rewardContracts[existingAddress].claimedRewards == 0) {

            delete rewardContracts[existingAddress];
            delete poolArrayIndex[existingAddress];

            arrayIndex = i;
            recycledSlot = true;
            emit PoolRecycled(existingAddress, contractAddress, arrayIndex);
            break;
        }
    }

    if (!recycledSlot) {
        arrayIndex = rewardPoolAddresses.length;
        rewardPoolAddresses.push(contractAddress);
    } else {
        rewardPoolAddresses[arrayIndex] = contractAddress;
    }
    ...
    function removeWhitelistRewardContract(
    ...
    pool.active = false;
    activePoolCount--;

    if (pool.unclaimedRewards > 0) {
        totalUnclaimedRewards -= pool.unclaimedRewards;
        pool.unclaimedRewards = 0;
    }
```

Vulnerabilities Medium

Missing Zero Address Check

Vulnerability severity: **Medium**

Vulnerability probability: **Low**

Multiple contracts are missing zero address checks

- `YppGoverningCouncil.initialize(_token)`
- `YppGoverningCouncil.initialize(_timelock)`

Recommendations:

Check contracts against `address(0)`

Vulnerabilities Medium

Missing Contract Validation

Vulnerability severity: **Medium**

Vulnerability probability: **Medium**

`uscyMintRewardsContract` presumes contracts are set correctly

These could be misconfigured, either through malice or accidental misconfiguration to point to incorrect contracts, ranging in severity from rendering the protocol non functional, to actively and maliciously draining customers

- `_mintingContract`
- `_timelockController`

Recommendations:

Validate contract abis and codesize in constructor

```
uint256 mintingContractCodeSize;
uint256 timelockCodeSize;
assembly {
    mintingContractCodeSize := extcodesize(_mintingContract)
}
require(mintingContractCodeSize > 0, "_mintingContract is not a contract");
assembly {
    timelockCodeSize := extcodesize( timelockController)
}
require(_timelockController > 0, "Timelock is not a contract");
try IYppMintContract(_token).getRewardPoolCap() returns (uint256) {
} catch {
    revert InvalidTokenAddress();
}
```

Vulnerabilities Medium

Missing Contract Validation

Vulnerability severity: **Medium**

Vulnerability probability: **Medium**

`YppGoverningCouncil` presumes contracts are set correctly

These could be misconfigured, either through malice or accidental misconfiguration to point to incorrect contracts, ranging in severity from rendering the protocol non functional, to actively and maliciously draining customers

- `_token`
- `_timelock`

Recommendations:

Validate contract abis and codesize in constructor

```
uint256 tokenCodeSize;
uint256 timelockCodeSize;
assembly {
    tokenCodeSize := extcodesize(_token)
}
require(mintCodeSize > 0, "_yppMintAddress is not a contract");
assembly {
    timelockCodeSize := extcodesize( timelock)
}
assembly {
    timelockCodeSize := extcodesize(_timelock)
}
require(timeLockCodeSize > 0, "Timelock is not a contract");
try IERC20(_token).totalSupply() returns (uint256) {
} catch {
    revert InvalidTokenAddress();
}
```

Vulnerabilities Medium

Missing Contract Validation

Vulnerability severity: **Medium**

Vulnerability probability: **Medium**

`YppBootstrapLiquidity` presumes contracts are set correctly

These could be misconfigured, either through malice or accidental misconfiguration to point to incorrect contracts, ranging in severity from rendering the protocol non functional, to actively and maliciously draining customers

- `_yppMintAddress`
- `_timelockController`

Recommendations:

Validate contract abis and codesize in constructor

```
uint256 mintCodeSize;
uint256 timeLockCodeSize;
assembly {
    mintCodeSize := extcodesize(_yppMintAddress)
}
require(mintCodeSize > 0, "_yppMintAddress is not a contract");
assembly {
    timeLockCodeSize := extcodesize( timelock)
}
require(timeLockCodeSize > 0, "Timelock is not a contract");
IYppMint testContract = IYppMint(_yppMintAddress);
try testContract.mint(address(0), address(0), 0) {
    // Unexpected success - might be problematic
} catch Error(string memory reason) {
    // Expected revert with reason - function exists
} catch (bytes memory) {
    revert("Invalid YppMint implementation - mint function issue");
}
```

Vulnerabilities Low

Gas Vulnerability: Unbounded external calls

Vulnerability severity: **Low**

Vulnerability probability: **Low**

`YppBootstrapLiquidity` makes contract calls without gas bounds

```
        yppMintImplementation.updateUnclaimedRewards(address(this), amount);

        yppMintImplementation.mint(address(this), deployer, amount);
```

This can cause out of gas exceptions due to reentrancy, inefficiency, or injection attacks.

Recommendations:

Limit gas for external contract interactions:

```
(bool success1,) = address(yppMintImplementation).call{gas: GAS_LIMIT_UPDATE_REWARDS}(
    abi.encodeCall(IYppMint.updateUnclaimedRewards, (address(this), amount))
);
if (!success1) revert ExternalCallFailed("updateUnclaimedRewards");

(bool success2,) = address(yppMintImplementation).call{gas: GAS_LIMIT_MINT}(
    abi.encodeCall(IYppMint.mint, (address(this), deployer, amount))
);
if (!success2) revert ExternalCallFailed("mint");
```

Vulnerabilities Info

Gas Inefficiency: Missing Immutable

Vulnerability severity: **Info**

Vulnerability probability: **Info**

`YppBootstrapLiquidity.yppMintImplementation` is set once in constructor but is not immutable

Recommendations:

Mark this variable as `immutable`. This can save upto 2000 gas per read.

Gas Inefficiency: Runtime computation of constant

Vulnerability severity: **Info**

Vulnerability probability: **Info**

`YppBootstrapLiquidity` computes a known constant at runtime

```
uint256 amount = 400_000 ether;
```

Recommendations:

Convert to a constant `uint256 private constant BOOTSTRAP_AMOUNT = 400000000000000000000000;`

Disclaimer

Disclaimer

This report is governed by the Fidesium terms and conditions.

This report does not constitute an endorsement or disapproval of any project or team, nor does it reflect the economic value or potential of any related product or asset. It is not investment advice and should not be used as the basis for investment decisions. Instead, this report provides an assessment intended to improve code quality and mitigate risks inherent in cryptographic tokens and blockchain technology.

Fidesium does not guarantee the absence of bugs or vulnerabilities in the technology assessed, nor does it comment on the business practices, models, or regulatory compliance of its creators. All services, reports, and materials are provided "as is" and "as available," without warranties of any kind, including but not limited to merchantability, fitness for a particular purpose, or non-infringement.

Cryptographic assets and blockchain technologies are novel and carry inherent technical risks, uncertainties, and the possibility of unpredictable outcomes. Assessment results may contain inaccuracies or depend on third-party systems, and reliance on them is solely at the Customer’s risk.

Fidesium assumes no liability for content inaccuracies, personal injuries, property damages, or losses related to the use of its services, reports, or materials. Third-party components are provided "as is," and any warranties are strictly between the Customer and the third-party provider.

These services and materials are intended solely for the Customer's use and benefit. No third party or their representatives may claim rights to or rely on these services, reports, or materials under any circumstances.