Yeld Smart Contract Audit

October, 2020

By CoinFabrik

Introduction

CoinFabrik was asked to audit the contracts for the Yeld.Finance project. First we will provide a summary of our discoveries and then we will show the details of our findings.

Summary

The contracts audited are from the Yeld package. The audit is based on the https://github.com/merlox/yeld-contracts/ repository at the commit f2dd2cce4b2ff60854e5849f88db8aa8ac1b765b.

Contracts

The audited contracts are:

- yDAI.sol: Main contract with the logic to maintain multiple tokens
- RetirementYeldTreasury.sol: contract that holds the retirement yeld funds and distributes them.
- yeldDAI.sol: Token with price update mechanism.

Analyses

The following analyses were performed:

- Misuse of the different call methods
- Integer overflow errors
- Division by zero errors
- Outdated version of Solidity compiler
- Front running attacks
- Reentrancy attacks
- Misuse of block timestamps
- Softlock denial of service attacks
- Functions with excessive gas cost
- Missing or misused function qualifiers

- Needlessly complex code and contract interactions
- Poor or nonexistent error handling
- Failure to use a withdrawal pattern
- Insufficient validation of the input parameters
- Incorrect handling of cryptographic signatures

Detailed findings

Severity Classification

Security risks are classified as follows:

- **Critical:** These are issues that we manage to exploit. They compromise the system seriously. They must be fixed **immediately**.
- **Medium:** These are potentially exploitable issues. Even though we did not manage to exploit them or their impact is not clear, they might represent a security risk in the near future. We suggest fixing them **as soon as possible**.
- **Minor:** These issues represent problems that are relatively small or difficult to take advantage of but can be exploited in combination with other issues. These kinds of issues do not block deployments in production environments. They should be taken into account and be fixed **when possible**.
- **Enhancement:** These kinds of findings do not represent a security risk. They are best practices that we suggest to implement.

This classification is summarized in the following table:

SEVERITY	EXPLOITABLE	ROADBLOCK	TO BE FIXED
Critical	Yes	Yes	Immediately
Medium	In the near future	Yes	As soon as possible
Minor	Unlikely	No	Eventually
Enhancement	No	No	Eventually

Issues Found by Severity

Critical severity

No issues of critical severity found.

Medium severity

No issues of medium severity found.

Minor severity

No issues of minor severity found.

Enhancements

Variables used but not set could be declared constant

Variables used in the code, namely dai, maximumTokensToBurn and weth could be declared as constant. This issue comes from the variables not being changed in the contract. To save gas, they could be declared as in the following example:

```
  address public constant dai = 0x6B175474E89094C44Da98b954EedeAC495271d0F;
  address public constant weth = 0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2;
//...
  uint256 public constant maximumTokensToBurn = 50000 * 1e18;
```

Conclusion

We found the contracts to be straightforward and well done, with care given to many common issues such as reentrancy. The audited code of the contract used contract interfaces properly and there was care given into making them secure. To sum up, the code is well-written and no important issues were found.

Disclaimer: This audit report is not a security warranty, investment advice, or an approval of the Yeld project since CoinFabrik has not reviewed its platform. Moreover, it does not provide a smart contract code faultlessness guarantee.

Appendix 1: Automated tool analysis

Mythril

- No issues found.

Slither

- Experimental features are turned on.

Experimental ABI encoder v2 is enabled in this contract. However, this isn't necessarily an issue and detailed analysis is needed to see if this could lead to problems with the contract.

- The contract sends eth to arbitrary user

These warnings are caused by extractETHIfStuck and buyNBurn by sending Ether to the owner of the contract or to the uniswap router address. Since these addresses are arbitrary, Slither marks them as a warning.

- Many reentrancy reports

In these reports external calls are made before writing storage. The calls are made to known external contracts and these functions have modifiers ensuring the contract isn't called again.

- Some functions use a dangerous strict equality

In these cases the comparison is against 0, checking for failure when minting or withdrawing Compund, or checking, within deposit(), if the pool variable equals 0.

- Local variables never initialized in the context of _withdrawDydx and _supplyDydx

These warnings are from variables that are initialized in the following lines. A more detailed analysis will be needed to discard this error.

- extractTokensIfStuck and redeemYeld ignore return values by transfer

These warnings occur when calling a transfer function and not checking the return value. According to ERC20, the return value must be verified, since it is possible for the function to return false.

- ERC20Detailed constructor has variable shadowing

This warning comes from using user-defined getters to access the variables containing name, symbols and decimals, and a more detailed analysis reveals it not to be an issue in this instance.

- Address.isContract uses assembly

It is recommended to avoid assembly when possible. We have not found any serious issue but it will require extra analysis.

- Low level calls used in sendValue and callOptionalReturn

These calls are made in the SafeERC20 and Address libraries, and a deeper analysis shows them to be harmless.

- Parameters and variables not in mixedCase

After some analysis, it is found that the variables the warnings are about are indeed, in mixed case, with the first character being an underscore.

- Dai, maximumTokensToBurn and weth should be constant

This issue comes from the variables not being changed in the contract. To save gas, they could be made constant.

- Some functions should be declared external

To save gas, some functions could be declared external. Their list is the following:

  - getAccountWei(Structs.Info,uint256)
  - operate(Structs.Info[],Structs.ActionArgs[])
  - setUniswapRouter(address)
  - extractTokensIfStuck(address,uint256
  - extractETHIfStuck()
  - rebalance()
  - getPricePerFullShare()