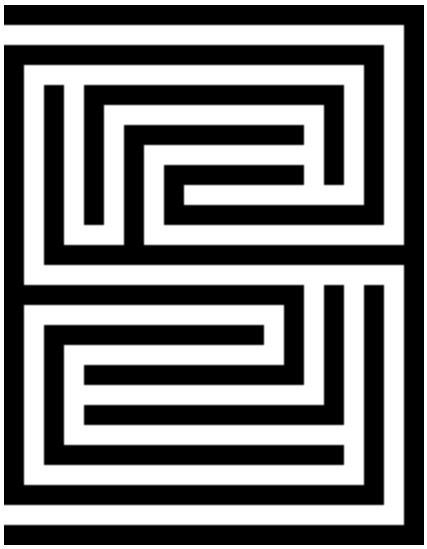

Yeld Finance Public Report

PROJECT: Yeld Finance Review

September 2020

Prepared For:

Merunas Grincalaitis | Yeld Finance

merunas@yeld.finance

Prepared By:

Jonathan Haas | Bramah Systems, LLC.

jonathan@bramah.systems

Table of Contents

Yeld Finance Review

Executive Summary

Scope of Engagement

Bramah Systems, LLC was engaged in September of 2020 to perform a comprehensive security review of multiple Yeld Finance smart contracts (specific contracts denoted below). Our review was conducted over a period of three business days by a member of the Bramah Systems, LLC. executive staff.

Bramah's review pertains to smart contract Solidity code (*.sol) as of commit **0da9e3345f6439c8db7694701ee87a0f0d8fb67c**, specifically within **RetirementYeldTreasury.sol**, **yDAI.sol**, and **yeldDAI.sol**, as per request of Yeld Finance. Bramah Systems completed the assessment using manual, static and dynamic analysis techniques.

Timeline

Review Commencement: September 21, 2020

Report Delivery: September 24, 2020

Engagement Goals

The primary scope of the engagement was to evaluate and establish the overall security of the Yeld Finance system, with a specific focus on trading actions. In specific, the engagement sought to answer the following questions:

- Is it possible for an attacker to steal or freeze tokens?
- Does the Solidity code match the specification as provided?
- Is there a way to interfere with the contract mechanisms?
- Are the arithmetic calculations trustworthy?

Protocol Specification

A basic specification document was compiled by the review team based upon review of the

Yeld finance code and discussion with the team.

The Yeld protocol relies on smart contracts created by the YEARN finance team, which have since received multiple security reviews denoting risk (as of the date of publication, multiple outstanding items exist in the **present list of reviews**).

As prior reviews have noted, documentation of the YEARN codebase is sparse. As this codebase serves as the basis from which Yeld has developed, code comments, where present, are brief. As the Yeld protocol relies upon a number of assumptions made within the YEARN codebase, we advocate Yeld developers take YEARN shortcomings into development consideration.

Yeld Finance did not provide a specification document, and as such our assessment is based upon manual inspection of the code, discussion with the development team, and assessment of relevant code comments. This noted, the code comments created by the Yeld team are incredibly detailed, thoroughly describing the core functionality of each piece. Combining these comments with the usage of modularized functions and clearly named variables, we were able to clearly delineate each step taken by the protocol.

Overall Assessment

Bramah Systems was engaged to evaluate and identify multiple security concerns in the codebase of the Yeld Finance protocol architecture. During the course of our engagement, Bramah Systems denoted numerous instances wherein the protocol deviated from established best practices and procedures of secure software development. With limited exceptions (as described in this report), these instances were most commonly a result of structural limitations of Solidity and not due to inactions on behalf of the development team. It is of particular importance to note that many of the below recommendations apply as a result of usage of the YEARN finance smart contracts. While reviewed by multiple parties, YEARN development lacks many common traits of secure software development. This ran counter to our experience with the Yeld development team, whose modifications were largely up to industry standard with the exception of minor typographical errors. Yeld finance modifications appear to be designed with modularity in mind, streamlining the process of future modifications and potential platform changes. After review of the report, the Yeld finance team accepted denoted risks by Bramah Systems and resolved issues surrounding unclear code comments and typographical errors.

Disclaimer

As of the date of publication, the information provided in this report reflects the presently held, commercially reasonable understanding of Bramah Systems, LLC.'s knowledge of security patterns as they relate to the Yeld Finance Protocol, with the understanding that distributed ledger technologies ("DLT") remain under frequent and continual development, and resultantly carry with them unknown technical risks and flaws. The scope of the review provided herein is limited solely to items denoted within "Scope of Engagement" and contained within "Directory Structure". The report does NOT cover, review, or opine upon security considerations unique to the Solidity compiler, tools used in the development of the protocol, or distributed ledger technologies themselves, or to any other matters not specifically covered in this report.

The contents of this report must NOT be construed as investment advice or advice of any other kind. This report does NOT have any bearing upon the potential economics of the Yeld Finance protocol or any other relevant product, service or asset of Yeld Finance or otherwise. This report is not and should not be relied upon by Yeld Finance or any reader of this report as any form of financial, tax, legal, regulatory, or other advice.

To the full extent permissible by applicable law, Bramah Systems, LLC. disclaims all warranties, express or implied. The information in this report is provided "as is" without warranty, representation, or guarantee of any kind, including the accuracy of the information provided. Bramah Systems, LLC. makes no warranties, representations, or guarantees about the Yeld Finance Protocol. Use of this report and/or any of the information provided herein is at the users sole risk, and Bramah Systems, LLC. hereby disclaims, and each user of this report hereby waives, releases, and holds Bramah Systems, LLC. harmless from, any and all liability, damage, expense, or harm (actual, threatened, or claimed) from such use.

Timeliness of Content

All content within this report is presented only as of the date published or indicated, to the commercially reasonable knowledge of Bramah Systems, LLC. as of such date, and may be superseded by subsequent events or for other reasons. The content contained within this report is subject to change without notice. Bramah Systems, LLC. does not guarantee or warrant the accuracy or timeliness of any of the content contained within this report, whether accessed through digital means or otherwise.

Bramah Systems, LLC. is not responsible for setting individual browser cache settings nor can it ensure any parties beyond those individuals directly listed within this report are receiving the most recent content as reasonably understood by Bramah Systems, LLC. as of the date this report is provided to such individuals.

General Recommendations

Best Practices & Solidity Development Guidelines

Usage of ABIEncoderV2

A majority of the contracts associated with the protocol make usage of an experimental Solidity version (**pragma experimental ABIEncoderV2**) which enables usage of the new ABI encoder. **ABIEncoderV2** allows for the usage of structs and arbitrarily nested arrays (such as **string[]** and **uint256[][]**) in function arguments and return values.

As no present non experimental version for these constructs exists, one must acknowledge the associated risk in utilizing non release-candidate ("RC") software. It is understood that software in the beta phase will generally have more bugs than completed software as well as speed/performance issues and may cause crashes or data loss.

Lack of consistency in Solidity version

Contracts within the repository lack a singular Solidity version. As structural changes may occur between these versions, we suggest consolidating to a singular cohesive version (ideally as recent as possible).

Versions used: **0.5.17** and **^0.5.0**, **ABIEncoderV2**

Usage of block.timestamp

Miners can affect block.timestamp for their benefits. Thus, one should not rely on the exact value of block.timestamp. As a result of such, **block.timestamp** and **now** should traditionally only be used within inequalities.

Time considerations

Multiple variables are set relying upon Solidity's inexact time system. As not every year equals 365 days and not every day has 24 hours because of leap seconds, Solidity's one day/week/year values are inexact. As leap seconds cannot be predicted, an exact calendar match would require updating by an external oracle.

Note, the direct comparison of these variables (e.g. comparing two times) within their respective functions poses additional concern, as discussed in “Usage of block.timestamp” above (namely, a proper comparison may not be set). It is worth noting that this has downstream implications on calculations utilising this passage of time. In particular, usage in the **checkIfPriceNeedsUpdating** and **redeemETH** functions can present concern if the protocol is intended for usage with extreme time sensitivity.

Supply-chain risk

Third party integrations present a significant risk if untrusted parties are involved. While the general security posture of organisations Yeld Finance has integrated with (and resultantly, built protocol integrations for) is quite high, this report (and present security analysis) cannot say for certain these integrations will be without flaw. It is notable that all integrations have seen some form of security scrutiny (be it a bug bounty, security review, or security focused testing via the development team). That said, the scope of this review does not cover the security of these integrations beyond the protocol integrations themselves.

Notably, substantial testing exists for each integration and verification exists for each step of the integration process (primarily through usage of revert) to mitigate the bulk of these concerns.

Functions can be marked external

Functions within a contract that the contract itself does not call should be marked external in order to optimize for gas costs. In our review, most functions within **yeldDAI.sol** should consider this implementation after review of any instances in which the contract may call them.

Slither denotes the following functions as potential areas of concern:

- Ownable.renounceOwnership() (yeldDAI.sol#47-50)
- Ownable.transferOwnership(address) (yeldDAI.sol#51-53)
- ERC20.totalSupply() (yeldDAI.sol#69-71)
- ERC20.balanceOf(address) (yeldDAI.sol#72-74)
- ERC20.transfer(address,uint256) (yeldDAI.sol#75-78)
- ERC20.allowance(address,address) (yeldDAI.sol#79-81)
- ERC20.approve(address,uint256) (yeldDAI.sol#82-85)

- ERC20.transferFrom(address,address,uint256) (yeldDAI.sol#86-90)
- ERC20.increaseAllowance(address,uint256) (yeldDAI.sol#91-94)
- ERC20.decreaseAllowance(address,uint256) (yeldDAI.sol#95-98)
- ERC20Detailed.name() (yeldDAI.sol#144-146)
- ERC20Detailed.symbol() (yeldDAI.sol#147-149)
- ERC20Detailed.decimals() (yeldDAI.sol#150-152)
- yeldDAI.setYDAI(address) (yeldDAI.sol#219-221)
- yeldDAI.mint(address,uint256) (yeldDAI.sol#223-225)
- yeldDAI.burn(address,uint256) (yeldDAI.sol#227-229)
- yeldDAI.changePriceRatio(uint256) (yeldDAI.sol#233-235)
- yeldDAI.updatePrice() (yeldDAI.sol#245-253)
- yeldDAI.extractTokensIfStuck(address,uint256) (yeldDAI.sol#255-257)
- yeldDAI.extractETHIfStuck() (yeldDAI.sol#259-261)

Typographical Errors

Numerous typographical errors in the code exist, however, these typographical errors do not have material consequences, such as the following line:

```
require(yeld.balanceOf(msg.sender) >=
snapshots[msg.sender].yeldBalance, 'Your balance must
be equal or higher the snapshoted balance');
```

While this line could be rewritten more clearly as to state “Your balance must be equal to or higher than the snapshotted balance”, functionally this does not change the behavior of the smart contract.

Specific Recommendations

Unique to the Yeld Finance Protocol

Complex Logic in userPercentage Calculation

The calculation of **userPercentage** within **RetirementYeldTreasury** relies upon a chain of mathematical operations happening within a single line. For debugging purposes and clarity of usage, these statements could be separated.

“Pre-Flattened” Solidity Files

The Solidity files come flattened, containing all contracts needed for deploy of the specific contract. However, by deploying this way, comparing changes between files (especially those considered to be template code) becomes increasingly difficult, which may present confusion.

Code comments may lead to confusion

Comments for the updatePrice function describe that “... this function will be called to calculate how many YELD each staker gets.” While the function does possess controls to ensure it is only called once a day, the function does not appear to be automatically called by any means, requiring manual action.

Comments for **buyNBurn** describe that the function returns “how many YELD tokens have been burned” -- but the timeframe of which period these values are generated from is unclear. The comment should be revised to include whether or not the intent is to include which tokens have been burned within the transaction, or in the totality of all token burnings (such as in **uint256 tokensAlreadyBurned = yeldToken.balanceOf(address(0));**). It is worth noting that in either scenario, buy and burn only applies up to 50k tokens burned due to present hardcoded values in the contract (the **maximumTokensToBurn** value).

Uniswap request timeout period

Presently, the Dai to Ether conversion function makes use of Uniswap in order to execute. The following request is sent to the Uniswap router:

```
uint[] memory amounts = IUniswap(uniswapRouter).swapExactTokensForETH(_amount,
uint(0), path, address(this), now.add(1800));
```

As present network volumes have resulted in significantly delayed transactions, it is suggested that this timeout value be revisited as network congestion continues. While a market exists for all crypto-currencies presently within the contracts, in a future state, such a market may not exist (or changes to Uniswap may materially impact the response returned), presenting additional considerations with respect to time.

Function extractTokensIfStuck provides highly sensitive access

The **extractTokensIfStuck** functionality, while restricted to onlyOwner, does provide sensitive functionality (namely, transfer of tokens). We suggest that this function emit an event or require multi-party verification (preventing a scenario in which a highly permissive ownership is affixed to a singular wallet).

Toolset Warnings

Unique to the Yeld Finance Protocol

Overview

In addition to our manual review, our process involves utilizing concolic analysis and dynamic testing in order to perform additional verification of the presence security vulnerabilities. An additional part of this review phase consists of reviewing any automated unit testing frameworks that exist.
The following sections detail warnings generated by the automated tools and confirmation of false positives where applicable, in addition to findings generated through manual inspection.

Compilation Warnings

While compilation errors exist within the repository at time of review (an invalid checksum address within yUSDC), these compilation warnings do not materially affect the files within the scope of review.

Test Coverage

The contract repository lacks substantial unit test coverage throughout. This testing traditionally would provide a variety of unit tests which encompass the various operational stages of the contract.
Presently, the Yeld Finance protocol (and its relevant components and their respective subcomponents) does not possess tests validating functionality and ensuring that certain behaviors (those relating to erroneous or overflow-prone input) do not see successful execution.

Static Analysis Coverage

The contract repository underwent heavy scrutiny with multiple static analysis agents, including:

- Securify
- MAIAN

- Mythril
- Oyente
- Slither

In each case, the team had either mitigated relevant concerns raised by each of these tools or provided adequate justification for the risk (e.g. inherent risk of using native Ethereum constructs such as **timestamp**).

In particular, static analysis for the contracts denoted the existence of the flattened codebase, which exists as one of the chief barriers to ease of review. As these tools often lack the ability to determine context, flattened codebases create difficulty during assessment (which contract is being properly inherited? Am I using the most recent library?). That said, the modifications made by Yeld **are** specifically denoted, which combined with file diffing allowed for Bramah to determine the scope of relevant modifications.

Certain tools, like Oyenete, do not run on newer versions of Solidity. Where applicable, Bramah manually performed validation checks based upon our understanding of the tool. With the exception of known issues stemming from the usage of YEARN finance code, Bramah did not locate any instances of concern within the modified code.

Directory Structure

At time of review, the directory structure of the Yeld Finance contract (**./contracts**) appeared as it does below. Our review, at request of Yeld, covers the Solidity code (*.sol) as of commit **0da9e3345f6439c8db7694701ee87a0f0d8fb67c**, specifically within **RetirementYeldTreasury.sol**, **yDAI.sol**, and **yeldDAI.sol**.

```
├── LICENSE
├── README.md
├── contracts
│   ├── IUniswap.sol
│   ├── IYeldTokens.sol
│   ├── RetirementYeldTreasury.sol
│   ├── merged
│   │   ├── yDAI_merged.sol
│   │   ├── yTUSD_merged.sol
│   │   ├── yUSDC_merged.sol
│   │   └── yUSDT_merged.sol
│   ├── yDAI.sol
│   ├── yTUSD.sol
│   ├── yUSDC.sol
│   ├── yUSDT.sol
│   ├── yeldDAI.sol
│   ├── yeldTUSD.sol
│   ├── yeldUSDC.sol
│   └── yeldUSDT.sol
├── interfaces
│   ├── Aave.sol
│   ├── Balancer.sol
│   ├── Controller.sol
│   ├── Converter.sol
│   ├── Curve.sol
│   ├── DForce.sol
│   ├── Gauge.sol
│   ├── MStable.sol
│   ├── OneSplitAudit.sol
```

```
│   ├── Oracle.sol
│   ├── Oracle_merged.sol
│   ├── Strategy.sol
│   ├── Uniswap.sol
│   ├── Vault.sol
│   ├── Yfii.sol
│   └── yVault.sol
├── migrations
│   └── 1_initial_migration.js
├── package.json
└── truffle-config.js

6 directories, 72 files
```