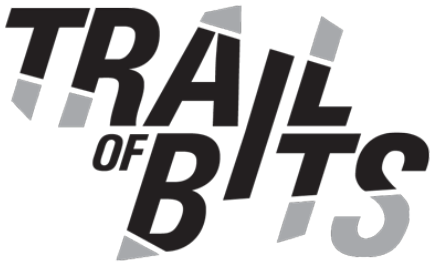

88mph v3

Security Assessment

August 6, 2021

Prepared For:
Guillaume Palayer | *88mph*
mcfly@88mph.app

Zefram Lou | *88mph*
zefram@88mph.app

Prepared By:
Dominik Teiml | *Trail of Bits*
dominik.teiml@trailofbits.com

Maximilian Krüger | *Trail of Bits*
max.kruger@trailofbits.com

Changelog:

August 6, 2021:	Delivered initial report
August 17, 2021:	Added Fix Log (Appendix D)

Executive Summary

From July 19 to August 6, 2021, Trail of Bits conducted a review of the 88mph v3 smart contracts, working from commit `76cd9d1f` of the `88mph-contracts` repository.

During the first week of the audit, we reviewed the codebase to gain an understanding of its intended behaviors, and we began reviewing the `DInterest` contract and the contracts in the `models` and `moneymarkets` directories. In the second week, we continued reviewing the money markets and began reviewing the libraries and the upgradeability mechanism. We also set up a harness for a fuzzing campaign. In the third week, we identified 15 invariants in the `DInterest` contract and used our harness to turn them into Echidna properties.

Our review resulted in 17 findings ranging from high to informational severity, many of which are related to data validation. For example, one high-severity issue involves the protocol's assumption that the money market rate will compound linearly, which could lead to inaccurate exponential moving average (EMA) and interest rate calculations (TOB-88MPH-006). One medium-severity issue concerns the lack of validation of the interest oracle's money market upon `DInterest` initialization (TOB-88MPH-001); another involves the inability of users to specify a minimum desired interest amount for new deposits (TOB-88MPH-004).

The 88mph protocol integrates with several other protocols (Aave, Compound, Harvest, B-Protocol, Cream, and Yearn) and includes many contracts, exposing a large attack surface. While the code is reasonably well structured and includes unit and integration tests, we found many aspects of the code that could be improved, such as its interactions with external contracts, upgradeability mechanisms, and data validation.

Trail of Bits recommends that 88mph take the following steps:

- Address all reported issues.
- Consider the impacts of linearizing the input and output of interest rate calculations.
- Conduct an additional internal review of the protocol's interactions with external contracts, particularly external money markets.
- Expand the unit test suite by adding edge cases for constant values and interactions that are not currently covered and expand our Echidna setup by adding properties to test other contracts.
- Perform a security assessment of protocol components omitted from this review.

Project Dashboard

Application Summary

Name	88mph v3
Version	`76cd9d1f`
Type	Solidity
Platform	Ethereum

Engagement Summary

Dates	July 19–August 6, 2021
Method	Full knowledge
Consultants Engaged	2
Level of Effort	6 person-weeks

Vulnerability Summary

Total High-Severity Issues	3	■ ■ ■
Total Medium-Severity Issues	3	■ ■ ■
Total Low-Severity Issues	2	■ ■
Total Informational-Severity Issues	8	■ ■ ■ ■ ■ ■ ■ ■
Total Undetermined-Severity Issues	1	■
Total	17	

Category Breakdown

Access Controls	1	■
Configuration	1	■
Data Validation	7	■ ■ ■ ■ ■ ■ ■
Optimization	3	■ ■ ■
Timing	1	■
Undefined Behavior	4	■ ■ ■ ■
Total	17	

Code Maturity Evaluation

Category Name	Description
Access Controls	**Satisfactory.** In general, the functions have appropriate access controls.
Arithmetic	**Moderate.** The project uses Solidity 0.8's built-in safe math operations. Because the calculations at the core of the protocol are complex, their documentation and testing should be more thorough. We also found an issue related to the estimation of the money market interest rate (TOB-88MPH-006).
Assembly Use/Low-Level Calls	**Moderate**. The use of assembly is limited to the common purpose of executing `chainid`. However, because of the lack of a contract existence check on `delegatecall` in a dependency, a call to the logic contract could fail silently (TOB-88MPH-008). The codebase also lacks return value checks on several calls to the `transfer` and `transferFrom` functions (TOB-88MPH-002).
Centralization	**Weak.** Privileged users are able to update the majority of the contracts without limits and may be able to leverage the protocol's upgradeability to engage in malicious behavior.
Code Stability	**Satisfactory.** The code was stable during the audit.
Upgradeability	**Moderate.** The project uses OpenZeppelin's `TransparentUpgradeableProxy` to facilitate upgrades for many contracts. We found two issues related to upgradeability (TOB-88MPH-007, TOB-88MPH-008).
Function Composition	**Satisfactory.** The code is reasonably well structured. The functions have clear purposes, and critical functions can be easily extracted for testing.
Front-Running	**Moderate.** We found two issues related to front-running (TOB-88MPH-004, TOB-88MPH-007).
Monitoring	**Satisfactory.** All of the functions emit events where appropriate. The events emitted by the code are capable of effectively monitoring on-chain activity.
Specification	**Moderate.** The 88mph team provided comprehensive documentation. However, many functions are missing code

	comments. We found several pieces of security-critical code that have no comments providing guidance to auditors and developers (TOB-88MPH-012, TOB-88MPH-013, TOB-88MPH-014).
Testing and Verification	**Moderate.** The system utilizes unit and integration tests. However, there are many hard-coded constants; testing more variations of system parameters would be beneficial.

Engagement Goals

The engagement was scoped to provide a security assessment of the full `88mph-contracts` repository at commit hash `76cd9d1f`.

Specifically, we sought to answer the following questions:

- Does the arithmetic produce correct results when given the values that can occur in the protocol?
- Does the system correctly use the APIs of external money markets?
- Are the system's assumptions about external contracts, namely money markets, correct?
- Do the five most important actions exposed by `DInterest`—depositing, topping up, rolling over, funding, and withdrawing—lead to expected outcomes, even when the money markets' interest rates change?
- Are the access controls of each contract sufficient to ensure it behaves as intended in a hostile environment?
- Does the codebase conform to industry best practices?

Coverage

Custom and imported libraries. The project uses OpenZeppelin contracts, custom libraries, and contracts built by the 88mph team. The two libraries—`DecMath` and `SafeERC20`—handle arithmetic and token interactions, respectively. We assessed the arithmetic functions and the safety of the token interactions, given the varying implementations of ERC20 tokens. The `libs` directory also contains token contracts built on top of OpenZeppelin contracts. We checked whether proper access controls are in place, the batch and multi functions are correctly implemented, and the functions are safely overridden.

Models. We assessed the core arithmetic of the protocol. There are three types of models: two for calculating interest and fees and one for computing the EMA based on the current income index and the previous EMA. We focused on the EMA oracle and interest model, in particular `LinearDecayInterestModel`. We checked whether the EMA is computed correctly from the money market income index and whether the interest model sets an appropriate interest rate based on the EMA input.

Money markets. This set of contracts constitutes a functional interface for interest-generating lending and yield-farming protocols for use by `DInterest`. We closely examined the money markets' APIs to ensure that they are used correctly and consistently.

Tokens. This directory contains the deployable implementations of the template contracts in `libs`. We reviewed the contracts to ensure that the newly defined methods have appropriate access controls, that the upgradeability mechanism is correct, and that all other contracts in the system correctly use the API exposed by the token contracts.

Rewards. This directory contains the `mph` reward token and its minter, the vesting contract, and the `xMPH` contract. We reviewed the codebase to ensure that token minting and vesting can occur only when expected. We also checked the access controls and upgradeability mechanism in these modules.

`ZeroCouponBond`. This high-level contract enables the fungibility of deposits into the system. We checked whether the contract behaves as intended and whether the access controls and the upgradeability mechanism are correct.

`DInterest`. We checked the access controls on the functions that modify the protocol parameters and the access controls on the actions that can be taken after a deposit, such as topping up, rolling over, and withdrawing. We manually reviewed the arithmetic, focusing on how the arithmetic behaves when composing various actions. We assessed whether the upgradeability mechanism is vulnerable to common issues. Finally, we fuzzed the contract extensively to verify that the calls to deposit and withdraw functions succeed and fail in accordance with those functions' preconditions.

`Factory`. This contract deploys all new contracts in the system. We checked that it correctly uses OpenZeppelin's `Clones` library, that it correctly emits events, and that it returns the correct values.

We were not able to sufficiently cover the following sections of the codebase with a manual review:

- `DInterestLens.sol`
- `xMPH.sol`
- `/zaps`

Automated Testing and Verification

Trail of Bits used automated testing techniques to enhance coverage of certain areas of the contracts, including the following:

- Slither, a Solidity static analysis framework
- Echidna, a smart contract fuzzer that rapidly tests security properties via malicious, coverage-guided test case generation

Automated testing techniques augment our manual security review but do not replace it. Each method has limitations: Slither may identify security properties that fail to hold when Solidity is compiled to EVM bytecode, and Echidna may not randomly generate an edge case that violates a property.

Automated Testing with Echidna

While reviewing the codebase, we identified properties and invariants that should hold. Using Echidna, we implemented the following property-based tests:

ID	Property	Result
1	`DInterest.deposit(amount, ...)` transfers the correct amount of assets indicated by the `amount` argument.	PASSED
2	`DInterest.deposit(...)` returns a non-zero `depositID`.	PASSED
3	`DInterest.deposit(...)` returns non-zero interest.	PASSED
4	`DInterest.deposit(amount, ...)` reverts if the sender holds less assets than the `amount` argument.	PASSED
5	`DInterest.deposit(amount, ...)` reverts if the sender has not approved the `amount`.	PASSED
6	`DInterest.deposit(amount, ...)` reverts if the `amount` is below the minimum deposit amount.	PASSED
7	`DInterest.deposit(..., maturationTimestamp)` reverts if the time indicated by the `maturationTimestamp` has already passed.	PASSED
8	`DInterest.deposit(...)` reverts if the value of the deposit period is above the value of the maximum deposit period.	PASSED

9	`DInterest.deposit(...)` reverts if the deposit yields no interest.	PASSED
10	`DInterest.deposit(...)` does not revert if all preconditions are met.	PASSED
11	`Dinterest.withdraw(...)` returns the `amount` that it transferred.	PASSED
12	`Dinterest.withdraw(...)` does not transfer more than the deposited amount, plus interest.	PASSED
13	`Dinterest.withdraw(depositID, ...)` reverts if no deposit with the `depositID` exists.	PASSED
14	`Dinterest.withdraw(..., amount, ...)` reverts if the `amount` is zero.	PASSED
15	`Dinterest.withdraw(..., early)` reverts if the value of `early` does not match the deposit state.	PASSED
16	`DInterest.withdraw(...)` does not revert if all preconditions are met.	PASSED

Recommendations Summary

This section aggregates all the recommendations made during the engagement. Short-term recommendations address the immediate causes of issues. Long-term recommendations pertain to the development process and long-term design goals.

Short term

❑ **Add `require(interestOracle.moneyMarket() == moneyMarket, "BAD_ORACLE")` to `DInterest._DInterest_init`.** Alternatively, remove `moneyMarket` from `DInterest` so that the system always fetches `moneyMarket` from `interestOracle.moneyMarket()`. TOB-88MPH-001

❑ **Use `safeTransfer` and `safeTransferFrom` in `xMPH.sol`.** TOB-88MPH-002

❑ **Replace `transferOwnership` with a two-step process, as depicted in figure 3.1.** TOB-88MPH-003

❑ **Add a `minInterestAmount` parameter to `DInterest.deposit` and insert a require statement—`require(interestAmount >= minInterestAmount)`—to ensure that the interest amount meets or exceeds the minimum requirement set by the user.** TOB-88MPH-004

❑ **Use a two-parameter version of `withdraw`.** TOB-88MPH-005

❑ **Evaluate the impact of using an arithmetic average rather than a geometric average for computing the compound rate per second.** If the impact is severe, consider using the geometric average instead. Alternatively, document the possible consequences of using the arithmetic average, particularly the increased likelihood of a system default. TOB-88MPH-006

❑ **Only deploy `TransparentUpgradeableProxy` contracts by passing the calldata of the call to `initialize` as the third constructor argument (figure 7.2).** That way, `initialize` will be called within the same transaction as the proxy contract construction, eliminating the front-running vulnerability entirely. This is the most robust solution to this issue. The `openzeppelin-hardhat-upgrades` plug-in can automatically call `initialize` within the same transaction as the proxy contract construction. TOB-88MPH-007

❑ **Document the fact that `selfdestruct` and `delegatecall` within logic contracts can lead to unexpected behavior, and prevent future upgrades from introducing these functions.** TOB-88MPH-008

❑ **Decide whether the `_rewards` address must be a contract and handle the validation of that address accordingly.** TOB-88MPH-009

❑ **Measure the gas savings from optimizations and carefully weigh them against the possibility of an optimization-related bug.** TOB-88MPH-010

❑ **Remove `vault.balanceOf(address(this))` from `totalValue()` and `totalValue(uint256)`.** TOB-88MPH-011

❑ **In all money markets, add explicit comments explaining the importance of the call to `super._authorizeRescue` and the importance of disallowing the rescue of all tokens used in normal operations.** TOB-88MPH-012

❑ **Remove the modified `libs/SafeERC20.sol` and import `@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol` instead.** Replace `safeApprove` with a custom function with a descriptive name, like `unsafeApproveSupportingNonStandardTokens`, and properly document its security implications and the reasoning behind its use. TOB-88MPH-013

❑ **Carefully investigate all operations that transfer entire balances to ensure that any side effects are intended.** Document the side effects and explain why they are intentional. TOB-88MPH-014

❑ **Expand the 88mph user documentation to include guidance on using Yearn vaults and on the associated risks.** Refer to the comments on Yearn's `withdraw` and `deposit` functions, which contain important guidance on their use. TOB-88MPH-015

❑ **Revise `_paySponsor` to transfer tokens from the sender to the sponsor in a single step.** TOB-88MPH-016

❑ **Replace `msg.sender` with `sender` in the `_transfer` function.** TOB-88MPH-017

Long term

❑ **Avoid the use of duplicated states within the protocol, as they can become inconsistent.** TOB-88MPH-001

❑ **Always use `safeTransfer` and `safeTransferFrom` rather than `transfer` and `transferFrom`.** TOB-88MPH-002

❑ **Use a two-step process for all non-recoverable critical operations to prevent irrevocable mistakes.** TOB-88MPH-003

❑ **For cases in which a transaction gives a sender an output amount in exchange for an input amount—during a trade, for example—ensure that the sender has control over the minimum desired output amount.** Otherwise, state changes outside of the sender's control, possibly caused by front-running, could result in an unexpectedly low output amount and a loss of funds for the user. TOB-88MPH-004

❑ **Examine the APIs of the protocols that the 88mph protocol depends on to see whether simpler and more gas-efficient means of interaction are available.** TOB-88MPH-005

❑ **Document any operations in the code that use a simpler algorithm than the mathematical model requires.** TOB-88MPH-006

❑ **Carefully review the Solidity documentation, especially the "Warnings" section, as well as the pitfalls of using the `delegatecall` proxy pattern.** TOB-88MPH-007, TOB-88MPH-008

❑ **Ensure that data validation is applied consistently across constructors, initializers, and setters.** TOB-88MPH-009

❑ **Monitor the development and adoption of Solidity compiler optimizations to assess their maturity.** TOB-88MPH-010

❑ **Clearly document which assets are held by each contract.** TOB-88MPH-011

❑ **Add documentation and warnings around critical lines of code to prevent accidental modifications from breaking important security invariants.** TOB-88MPH-012

❑ **Read the references cited in this finding to better understand the `approve` front-running attack, and ensure that it is mitigated throughout the codebase.** TOB-88MPH-013

❑ **Whenever possible, ensure that only the specific amount relevant to the context of the transfer is transferred.** For example, when an amount is withdrawn from the Yearn vault, transfer exactly that amount. Many functions of external protocols provide a return value indicating the transferred amount. For situations in which this information is not available, add an operation that measures the balance before and after the call and transfers the difference. TOB-88MPH-014

❑ **Carefully review the documentation of all protocols that the 88mph protocol interacts with and surface the risks and warnings of using these protocols to 88mph users.** TOB-88MPH-015

❑ **Investigate all transfers that are done in sequence and whether they can be combined into a single transfer.** TOB-88MPH-016

❑ **Carefully investigate all ERC20 token implementations in the codebase to ensure that they correctly implement the ERC20 standard.** TOB-88MPH-017

Findings Summary

#	Title	Type	Severity
1	The interest oracle's money market is not validated upon DInterest initialization	Data Validation	Medium
2	Lack of return value check on transfer and transferFrom	Data Validation	Informational
3	Lack of two-step process for contract ownership transfers	Data Validation	Informational
4	Users cannot specify a minimum desired interest	Data Validation	Medium
5	Withdrawing from Yearn to DInterest in a single step can save gas	Optimization	Informational
6	Linearization of exponential compounding could lead to insolvency	Data Validation	High
7	Initialization functions can be front-run	Configuration	High
8	Lack of contract existence check on delegatecall	Data Validation	Informational
9	Inconsistent validation of money markets' rewards address	Data Validation	Low
10	Solidity compiler optimizations can be problematic	Undefined Behavior	Undetermined
11	Redundant addition of zero value in the Harvest money market	Optimization	Informational
12	Lack of documentation concerning Rescuable base contract could result in exploitable modifications	Access Controls	Informational
13	Modifications make the safeApprove function unsafe	Timing	Informational
14	Transferring the entire balance of a contract has unintended consequences	Undefined Behavior	Low
15	Users are not informed of the pitfalls of using Yearn vaults	Undefined Behavior	Medium

16	Sponsor payout uses two transfers when only one is required	Optimization	Informational
17	ERC20Wrapper's transferFrom function ignores the sender argument	Undefined Behavior	High

1. The interest oracle's money market is not validated upon DInterest initialization

Severity: Medium
Type: Data Validation
Target: `DInterest.sol`

Difficulty: High
Finding ID: TOB-88MPH-001

Description
A `DInterest` contract must have the same money market as its interest oracle. The function that sets the interest oracle checks this invariant (figure 1.1).

```
function setInterestOracle(address newValue) external onlyOwner {
    require(newValue.isContract(), "NOT_CONTRACT");
    interestOracle = IInterestOracle(newValue);
    require(interestOracle.moneyMarket() == moneyMarket, "BAD_ORACLE");
    emit ESetParamAddress(msg.sender, "interestOracle", newValue);
}
```

Figure 1.1: `setInterestOracle` in `DInterest.sol#L1483-L1488`

However, `DInterest.__DInterest_init` does not check this invariant.

A mistake during initialization could cause a `DInterest` to have a different money market than its interest oracle. As a result, the system could provide incorrect interest rates, and either the users or the protocol could lose funds.

Exploit Scenario
Alice, a developer, deploys a new `DInterest` for DAI and AAVE and calls `DInterest.initialize`. She passes in the Aave money market address as one parameter and accidentally passes in the Yearn interest oracle address as the other. For some time, Yearn yields significantly lower interest than Aave. As a result, users receive significantly less interest than they would by using Aave directly.

Recommendations
Short term, add `require(interestOracle.moneyMarket() == moneyMarket, "BAD_ORACLE")` to `DInterest.__DInterest_init`. Alternatively, remove `moneyMarket` from `DInterest` so that the system always fetches `moneyMarket` from `interestOracle.moneyMarket()`.

Long term, avoid the use of duplicated states within the protocol, as they can become inconsistent.

2. Lack of return value check on transfer and transferFrom

Severity: Informational
Type: Data Validation
Target: `xMPH.sol`

Difficulty: High
Finding ID: TOB-88MPH-002

Description
The functions `_deposit`, `_withdraw`, and `_distributeReward` in the `xMPH` contract use the ERC20 `transfer` and `transferFrom` functions without checking that they return `true`.

Some tokens, like `BAT`, `HT` (Huobi), and `cUSDC`, do not revert when transfers fail. Instead, they return `false`. The lack of a return value check on such tokens could enable users to steal funds.

The severity of this issue is informational, as the `mph` token used in `xMPH` is likely an instance of `MPHToken.sol`, which reverts when transfers fail.

Exploit Scenario
A new `mph` token used in a new deployment of the 88mph protocol does not revert when transfers fail. As a result, users can deposit funds they do not have and then withdraw them, draining funds from the protocol.

Recommendations
Short term, use `safeTransfer` and `safeTransferFrom` in `xMPH.sol`.

Long term, always use `safeTransfer` and `safeTransferFrom` rather than `transfer` and `transferFrom`.

3. Lack of two-step process for contract ownership transfers

Severity: Informational
Type: Data Validation
Target: `OwnableUpgradeable`

Difficulty: High
Finding ID: TOB-88MPH-003

Description
Many contracts use OpenZeppelin's `OwnableUpgradeable` contract, whose `transferOwnership` function transfers contract ownership in a single step. If the owner of a contract were set to an address not controlled by the 88mph team, the contract would be impossible to recover.

This issue could be prevented by implementing a two-step process in which an owner proposes an ownership transfer and the proposed new owner accepts it.

```
address private _proposedOwner;

function proposeOwnership(address proposedOwner) external onlyOwner {
    require(proposedOwner != address(0));
    _proposedOwner = proposedOwner;
}

function acceptOwnership() external {
    require(msg.sender == _proposedOwner);
    address oldOwner = _owner;
    _owner = _proposedOwner;
    _proposedOwner = address(0);
    emit OwnershipTransferred(oldOwner, _owner);
}
```

Figure 3.1: Proposed code for a two-step ownership transfer

Exploit Scenario
Bob, a developer, changes the owner of a contract that inherits from `OwnableUpgradeable`. By mistake, he passes in the wrong address as the new owner. The development team cannot recover the contract and will have to implement a costly upgrade or redeployment to address the issue.

Recommendations
Short term, replace `transferOwnership` with a two-step process, as depicted in figure 3.1.

Long term, use a two-step process for all non-recoverable critical operations to prevent irrevocable mistakes.

4. Users cannot specify a minimum desired interest

Severity: Medium
Type: Data Validation
Target: `DInterest.sol`

Difficulty: Low
Finding ID: TOB-88MPH-004

Description
`DInterest.deposit(amount, maturationTimestamp)` computes and returns the amount of interest users will receive at `maturationTimestamp`. However, users calling this function cannot specify a minimum desired interest to guarantee that the amount of interest will be profitable.

Exploit Scenario
Charlie, a user of the 88mph protocol, calls `DInterest.deposit`. The amount of interest that is computed and returned is unacceptably low for Charlie. However, he has already deposited his funds into 88mph. To withdraw his funds for use in a more profitable market, Charlie has to pay the early withdrawal fee and the gas costs for another transaction.

Recommendations
Short term, add a `minInterestAmount` parameter to `DInterest.deposit` and insert a require statement—`require(interestAmount >= minInterestAmount)`—to ensure that the interest amount meets or exceeds the minimum requirement set by the user.

Long term, for cases in which a transaction gives a sender an output amount in exchange for an input amount—during a trade, for example—ensure that the sender has control over the minimum desired output amount. Otherwise, state changes outside of the sender's control, possibly caused by front-running, could result in an unexpectedly low output amount and a loss of funds for the user.

5. Withdrawing from Yearn to DInterest in a single step can save gas

Severity: Informational
Type: Optimization
Target: `YVaultMarket.sol`

Difficulty: Not Applicable
Finding ID: TOB-88MPH-005

Description
`YVaultMarket.withdraw` withdraws funds from the `YVaultMarket` itself, measures the `YVaultMarket`'s new balance, and transfers the balance to `msg.sender` (figure 5.1).

```
if (amountInShares > 0) {
    vault.withdraw(amountInShares);
}

// Transfer stablecoin to `msg.sender`
actualAmountWithdrawn = stablecoin.balanceOf(address(this));
if (actualAmountWithdrawn > 0) {
    stablecoin.safeTransfer(msg.sender, actualAmountWithdrawn);
}
```

Figure 5.1: `withdraw` in `YVaultMarket.sol#L65-73`

Yearn v2's `vault.withdraw` takes a second optional parameter, `receiver`, which defaults to `msg.sender` (figure 5.2). This version can also be seen in function #28 of the WBTC yVault on Etherscan.

```
def withdraw(
    maxShares: uint256 = MAX_UINT256,
    recipient: address = msg.sender,
    maxLoss: uint256 = 1,  # 0.01% [BPS]
) -> uint256:
```

Figure 5.2: `withdraw` in `Vault.vy#L1004-1008`

Using this two-parameter version of `withdraw` would be simpler and require less gas (figure 5.3).

```
if (amountInShares > 0) {
    vault.withdraw(amountInShares, msg.sender);
}
```

Figure 5.3: Recommended replacement for the code in figure 5.1

Recommendations
Short term, replace the code in figure 5.1 with the code in figure 5.3.

Long term, examine the APIs of the protocols that the 88mph protocol depends on to see whether simpler and more gas-efficient means of interaction are available.

6. Linearization of exponential compounding could lead to insolvency

Severity: High
Type: Data Validation
Target: `EMAOracle.sol`

Difficulty: Medium
Finding ID: TOB-88MPH-006

Description
In money markets such as Aave and Compound, liquidity is expected to compound exponentially. However, when computing the money market rate per second, the 88mph protocol assumes that the rate will compound linearly:

```
        uint256 incomingValue =
           (newIncomeIndex - _lastIncomeIndex).decdiv(_lastIncomeIndex) /
               timeElapsed;
```

Figure 6.1: `EMAOracle.sol#L82-L84`

To compute the true expansion rate per second, the algorithm should take the *n*th root of the difference of the new income index and the last income index, where *n* is the number of elapsed seconds. Instead, the algorithm divides the difference by *n*. We modeled the difference between these two options and found cases in which the current implementation overestimates the ROI.

Consider a lending pool that starts with 100 capital units and doubles its capital holdings every two seconds. After four seconds, it will own 400 capital units. The correct prediction would be that the money market will own 800 capital units after two more seconds elapse. However, the current implementation of the algorithm will predict instead that the money market will own 1,000.

The consequence of this implementation is that the system could give unreasonably high ROIs to users of fixed-interest yields, potentially increasing the likelihood of system insolvency.

The system also linearizes the interest rate calculation (figure 6.2).

```
    function calculateInterestAmount(
        uint256 depositAmount,
        uint256 depositPeriodInSeconds,
        uint256 moneyMarketInterestRatePerSecond,
        bool, /*surplusIsNegative*/
        uint256 /*surplusAmount*/
    ) external view override returns (uint256 interestAmount) {
        // interestAmount = depositAmount * moneyMarketInterestRatePerSecond * IRMultiplier
* depositPeriodInSeconds
        interestAmount =
            ((depositAmount * PRECISION)
                .decmul(moneyMarketInterestRatePerSecond)
                .decmul(getIRMultiplier(depositPeriodInSeconds)) *
                depositPeriodInSeconds) /
```

```
            PRECISION;
    }
```

Figure 6.2: `LinearDecayInterestModel.sol#L32-L46`

The most accurate calculation would involve raising the result of the input calculation to the power of *n*, where *n* is the number of seconds since the deposit. Instead, the algorithm multiplies the result by *n*. This implementation might offset the possible overestimation of the ROI, but further investigation would be required to confirm whether it does.

Exploit Scenario
In the example above, the protocol's estimated yield to the fixed-interest yield minter is much higher than the actual yield will be. As a result, yield token holders (funders) receive a smaller yield than they expected. Furthermore, the protocol incurs a net loss on the deposit, leading to system insolvency.

Recommendations
Short term, evaluate the impact of using an arithmetic average rather than a geometric average for computing the compound rate per second. If the impact is severe, consider using the geometric average instead. Alternatively, document the possible consequences of using the arithmetic average, particularly the increased likelihood of a system default.

Long term, document any operations in the code that use a simpler algorithm than the mathematical model requires.

7. Initialization functions can be front-run

Severity: High
Type: Configuration
Target: `deploy/*`

Difficulty: High
Finding ID: TOB-88MPH-007

Description
Several implementation contracts have `initialize` functions that can be front-run, allowing an attacker to incorrectly initialize the contracts. If the front-running of one of these functions is not detected immediately, an attacker may be able to steal funds at a later time.

The deployment scripts use two separate transactions for the deployment of the proxy contract and the call to `initialize` (figure 7.1).

```
const deployResult = await deploy(poolConfig.name, {
  from: deployer,
  contract: "DInterest",
  proxy: {
    owner: config.govTimelock,
    proxyContract: "OptimizedTransparentProxy"
  }
});
if (deployResult.newlyDeployed) {
  const DInterest = artifacts.require("DInterest");
  const contract = await DInterest.at(deployResult.address);
  await contract.initialize(
    BigNumber(poolConfig.MaxDepositPeriod).toFixed(),
```

Figure 7.1: `deploy/DInterest.js#L29-L41`

This makes the `initialize` functions vulnerable to front-running by an attacker, who could then initialize the contracts with malicious values. For example, an attacker could set an address that she owns as the `_rewards` parameter of `AaveMarket`'s `initialize` function, which would allow her to earn all rewards that accrue on the Aave money market.

Exploit Scenario
Attacker Eve has studied the next version of the 88mph protocol and identified several parameters of initialization functions that, if set to certain values, will allow her to steal funds from the protocol. She sets up a script to automatically watch the mempool and front-run the `initialize` functions of the next 88mph protocol deployment. Bob, a developer, deploys the next version of the 88mph protocol. Eve's script front-runs the calls to `initialize`. Bob does not notice the front-running attack. Eve can then steal funds deposited into the protocol.

Recommendations
Short term, only deploy `TransparentUpgradeableProxy` contracts by passing the calldata of the call to `initialize` as the third constructor argument (figure 7.2). That way, `initialize` will be called within the same transaction as the proxy contract construction, eliminating the front-running vulnerability entirely. This is the most robust solution to this issue. The `openzeppelin-hardhat-upgrades` plug-in can automatically call `initialize` within the same transaction as the proxy contract construction.

```
    constructor(
        address _logic,
        address admin_,
        bytes memory _data
    ) payable ERC1967Proxy(_logic, _data) {
```

Figure 7.2: openzeppelin-contracts//TransparentUpgradeableProxy.sol#L33-L37*

Long term, carefully review the Solidity documentation, especially the "Warnings" section, as well as the pitfalls of using the `delegatecall` proxy pattern.

8. Lack of contract existence check on delegatecall

Severity: Informational
Type: Data Validation
Target: `openzeppelin-contracts/*/Proxy.sol`

Difficulty: Medium
Finding ID: TOB-88MPH-008

Description
The project uses OpenZeppelin's `TransparentUpgradeableProxy` contract, which executes a `delegatecall` to the logic contract without first performing a contract existence check. This existence check is omitted to save gas. On the other hand, `upgradeTo(newLogic)`, which updates the logic contract, checks that `newLogic` is a contract.

A `call` or `delegatecall` to a non-contract address always succeeds. If `selfdestruct` were executed within the logic contract, future calls to the proxy would always succeed; this could result in unexpected behavior, leading to possible loss of funds. However, there is no `selfdestruct` or `delegatecall` functionality in any of the logic contracts. For this reason, the severity of this issue is informational.

Exploit Scenario
Alice, a developer, introduces a bug through which the logic contract of upgradeable contract A can be destroyed. All calls to the corresponding proxy contract of contract A now succeed instead of reverting. Contract B, which delegates data validation to contract A, calls contract A before releasing funds. Attacker Bob can now drain funds from contract B because contract B's calls to contract A always succeed.

Recommendations
Short term, document the fact that `selfdestruct` and `delegatecall` within logic contracts can lead to unexpected behavior, and prevent future upgrades from introducing these functions.

Long term, carefully review the Solidity documentation, especially the "Warnings" section, as well as the pitfalls of using the `delegatecall` proxy pattern.

9. Inconsistent validation of money markets' rewards address

Severity: Low | Difficulty: Medium
Type: Data Validation | Finding ID: TOB-88MPH-009
Target: `HarvestMarket.sol`, `AaveMarket.sol`, `BProtocolMarket.sol`, `CompoundERC20Market.sol`

Description
The validation of the `_rewards` address in the `HarvestMarket`, `AaveMarket`, `BProtocolMarket`, and `CompoundERC20Market` contracts is inconsistent. For example, in `HarvestMarket`, the initialization function checks whether the `_rewards` address is non-zero (figure 9.1), and the `setRewards` function checks whether it is a contract (figure 9.2). In `AaveMarket`, the `_rewards` address is not validated at all.

```
        require(
            _vault.isContract() &&
                _rewards != address(0) &&
                _stakingPool.isContract() &&
                _stablecoin.isContract(),
```

Figure 9.1: `HarvestMarket.sol#L34-L38`

```
    function setRewards(address newValue) external override onlyOwner {
        require(newValue.isContract(), "HarvestMarket: not contract");
        rewards = newValue;
        emit ESetParamAddress(msg.sender, "rewards", newValue);
    }
```

Figure 9.2: `HarvestMarket.sol#L126-L130`

Recommendations
Short term, decide whether the `_rewards` address must be a contract and handle the validation of that address accordingly.

Long term, ensure that data validation is applied consistently across constructors, initializers, and setters.

10. Solidity compiler optimizations can be problematic

Severity: Undetermined
Type: Undefined Behavior
Target: `hardhat.config.js`

Difficulty: Low
Finding ID: TOB-88MPH-010

Description
88mph has enabled optional compiler optimizations in Solidity.

There have been several optimization bugs with security implications. Moreover, optimizations are actively being developed. Solidity compiler optimizations are disabled by default, and it is unclear how many contracts in the wild actually use them. Therefore, it is unclear how well they are being tested and exercised.

High-severity security issues due to optimization bugs have occurred in the past. A high-severity bug in the `emscripten`-generated `solc-js` compiler used by Truffle and Remix persisted until late 2018. The fix for this bug was not reported in the Solidity CHANGELOG. Another high-severity optimization bug resulting in incorrect bit shift results was patched in Solidity 0.5.6. More recently, another bug due to the incorrect caching of `keccak256` was reported.

A compiler audit of Solidity from November 2018 concluded that the optional optimizations may not be safe.

It is likely that there are latent bugs related to optimization and that new bugs will be introduced due to future optimizations.

Exploit Scenario
A latent or future bug in Solidity compiler optimizations—or in the Emscripten transpilation to `solc-js`—causes a security vulnerability in the 88mph contracts.

Recommendations
Short term, measure the gas savings from optimizations and carefully weigh them against the possibility of an optimization-related bug.

Long term, monitor the development and adoption of Solidity compiler optimizations to assess their maturity.

11. Redundant addition of zero value in the Harvest money market

Severity: Informational
Type: Optimization
Target: `HarvestMarket.sol`

Difficulty: Medium
Finding ID: TOB-88MPH-011

Description
When `HarvestMarket.deposit` is called, funds are deposited into the Harvest vault, a proportional number of liquidity tokens is minted to `HarvestMarket`, and these liquidity tokens are immediately staked (transferred) into the Harvest rewards contract (figure 11.1). `HarvestMarket.withdraw` does the reverse by withdrawing liquidity tokens from the rewards contract and then using these tokens to withdraw assets from the vault. As a result, `HarvestMarket` does not hold any liquidity tokens after a `deposit` or `withdraw` call, except if liquidity tokens were sent to the contract by mistake, in which case they can be rescued.

```
        vault.deposit(amount);

        // Stake vault token balance into staking pool
        uint256 vaultShareBalance = vault.balanceOf(address(this));
        vault.approve(address(stakingPool), vaultShareBalance);
        stakingPool.stake(vaultShareBalance);
```

Figure 11.1: Part of the `deposit` function in `HarvestMarket.sol#L58-63`

Therefore, the return value of `vault.balanceOf(address(this))` in `HarvestMarket` is always zero and can be omitted from `totalValue()` (figure 11.2) and `totalValue(uint256)`.

```
    function totalValue() external view override returns (uint256) {
        uint256 sharePrice = vault.getPricePerFullShare();
        uint256 shareBalance =
            vault.balanceOf(address(this)) +
                stakingPool.balanceOf(address(this));
        return shareBalance.decmul(sharePrice);
    }
```

Figure 11.2: `HarvestMarket.sol#L98-104`

Recommendations
Short term, remove `vault.balanceOf(address(this))` from `totalValue()` and `totalValue(uint256)`.

Long term, clearly document which assets are held by each contract.

12. Lack of documentation concerning Rescuable base contract could result in exploitable modifications

Severity: Informational | Difficulty: High
Type: Access Controls | Finding ID: TOB-88MPH-012
Target: `MoneyMarket.sol`, `YVaultMarket.sol`, `HarvestMarket.sol`, `AaveMarket.sol`, `CompoundERC20Market.sol`, `BProtocolMarket.sol`, `CreamERC20Market.sol`

Description
The protocol uses a `Rescuable` base contract that allows privileged rescuer addresses to recover funds that were accidentally sent to the contracts.

This rescue functionality is a controlled backdoor into the system. It should not allow the transfer of tokens used during the normal operations of the money markets. This fact is not documented properly.

The callback `_authorizeRescue` handles access controls in `Rescuable`. All money markets call `super._authorizeRescue` (figure 12.1), which allows only one specific address to rescue tokens. This line of code is very important, yet it lacks documentation.

```
function _authorizeRescue(address token, address target)
    internal
    view
    override
{
    super._authorizeRescue(token, target);
    require(token != address(stakingPool), "HarvestMarket: no steal");
}
```

Figure 12.1: `HarvestMarket.sol#L135-142`

Exploit Scenario
Developer Charlie adds a new money market. He implements `_authorizeRescue` but forgets to add the call to `super._authorizeRescue`. User Bob accidentally sends tokens to the money market. Attacker Eve exploits the missing authorization to steal Bob's funds.

Recommendations
Short term, in all money markets, add explicit comments explaining the importance of the call to `super._authorizeRescue` and the importance of disallowing the rescue of all tokens used in normal operations.

Long term, add documentation and warnings around critical lines of code to prevent accidental modifications from breaking important security invariants.

13. Modifications make the safeApprove function unsafe

Severity: Informational
Type: Timing
Target: `SafeERC20.sol`

Difficulty: Low
Finding ID: TOB-88MPH-013

Description
The protocol uses a modified version of OpenZeppelin's `SafeERC20.sol`. The `SafeERC20.safeApprove` function provides some protection against the approve front-running attack, as it permits an allowance to be set only to zero or from zero. Certain tokens, like `USDT`, also provide this protection. However, the 88mph protocol does not handle approval operations in this way; rather, allowances need to be increased by a specific value, regardless of the current value. To allow the use of tokens like `USDT`, which are not compatible with this system, the 88mph team modified the `safeApprove` function. The modified `safeApprove` function does not offer protection against the `approve` front-running attack.

The `safeApprove` function has been deprecated by OpenZeppelin.

```
    /**
        @dev Modified from openzeppelin. Instead of reverting when the allowance is non-zero
and value
        is non-zero, we first set the allowance to 0 and then call approve(spender, value).
        This provides support for non-standard tokens such as USDT that revert in this
scenario.
     */
    function safeApprove(
        IERC20 token,
        address spender,
        uint256 value
    ) internal {
        if ((token.allowance(address(this), spender)) > 0) {
            _callOptionalReturn(
                token,
                abi.encodeWithSelector(token.approve.selector, spender, 0)
            );
        }
        _callOptionalReturn(
            token,
            abi.encodeWithSelector(token.approve.selector, spender, value)
        );
    }
```

Figure 13.1: Modified `SafeERC20.sol#L43-63`

All uses of `safeApprove` within the protocol are immediately followed by a call to `transferFrom`, making front-running impossible. Therefore, the severity of this issue is informational.

However, the existence of a function called `safeApprove` that does not match the expected semantics of OpenZeppelin's standard `safeApprove` is problematic.

Exploit Scenario
Developer Alice uses `safeApprove`, assuming it is safe and provides some protection against the approve front-running attack. However, the function is not safe, and users are vulnerable to the `approve` front-running attack.

Recommendations
Short term, remove the modified `libs/SafeERC20.sol` and import `@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol` instead. Replace `safeApprove` with a custom function with a descriptive name, like `unsafeApproveSupportingNonStandardTokens`, and properly document its security implications and the reasoning behind its use.

Long term, read the references below to better understand the `approve` front-running attack, and ensure that it is mitigated throughout the codebase.

References
- ERC20 API: An Attack Vector on Approve/TransferFrom Methods
- https://github.com/ethereum/EIPs/blob/master/EIPS/eip-20.md#approve

14. Transferring the entire balance of a contract has unintended consequences

Severity: Low | Difficulty: Medium
Type: Undefined Behavior | Finding ID: TOB-88MPH-014
Target: `YVaultMarket.sol`, `CompoundERC20Market.sol`, `BProtocolMarket.sol`, `HarvestMarket.sol`

Description
In several areas of the codebase, a contract's entire balance of a specific token is transferred (figures 14.1 and 14.2). Any user can control the balance by sending tokens to the contract. This could be used to manipulate accounting, resulting in a loss of funds.

```
actualAmountWithdrawn = stablecoin.balanceOf(address(this));
if (actualAmountWithdrawn > 0) {
    stablecoin.safeTransfer(msg.sender, actualAmountWithdrawn);
```

Figure 14.1: `YVaultMarket.sol#L70-72`

```
comp.safeTransfer(rewards, comp.balanceOf(address(this)));
```

Figure 14.2: `CompoundERC20Market.sol#L90`

The transfer performed by the `claimRewards` function in `CompoundERC20Market` (figure 14.2) and `BProtocolMarket` has issues. If a user accidentally sends `comp` tokens to these money markets, the tokens can be rescued. However, if `claimRewards` is called, these tokens are sent to the `rewards` address instead. This is inconsistent. Only `comp` tokens that were actually transferred by `claimComp` should be transferred to the `rewards` address. Alternatively, disallow the rescuing of `comp` tokens.

Most of the other transfers have side effects that, while unintended, favor the protocol at the expense of users who have accidentally sent tokens to a contract. This behavior is fine if rescuing the tokens in question is not allowed.

Exploit Scenario
Alice accidentally sends `comp` tokens to the Compound market. The 88mph team calls `claimRewards`, which sends Alice's tokens to the `rewards` address. Alice realizes her mistake and does not understand why the `comp` balance of the Compound market is zero. She is in distress, believing her tokens are gone. After some communication and investigation, the 88mph team is able to refund her tokens from the `rewards` address. While the issue was mitigated, the transfer of the entire balance, including Alice's tokens, has complicated the situation for everyone involved.

Recommendations

Short term, carefully investigate all operations that transfer entire balances to ensure that any side effects are intended. Document the side effects and explain why they are intentional.

Long term, whenever possible, ensure that only the specific amount relevant to the context of the transfer is transferred. For example, when an amount is withdrawn from the Yearn vault, transfer exactly that amount. Many functions of external protocols provide a return value indicating the transferred amount. For situations in which this information is not available, add an operation that measures the balance before and after the call and transfers the difference.

15. Users are not informed of the pitfalls of using Yearn vaults

Severity: Medium
Type: Undefined Behavior
Target: `YVaultMarket.sol`

Difficulty: Low
Finding ID: TOB-88MPH-015

Description
Yearn is inherently more risky than other money markets and requires caution on the user's side. The 88mph protocol uses a uniform abstraction over Yearn, Aave, Compound, and other money markets. Some of Yearn's implementation details can leak through this abstraction.

Depending on the state of a Yearn vault, a user making a withdrawal could realize a loss of funds, or the withdrawal could fail entirely. Before withdrawing from a Yearn vault, users should check the state of the vault and its strategies to prevent a loss or revert.

A Yearn vault has a number of free tokens, which are not invested in strategies. If the withdrawal amount is covered entirely by the free tokens, the withdrawal will succeed. However, if the withdrawal amount exceeds the free token balance, the funds must be withdrawn from the strategies in the withdrawal queue. If a user withdraws from a strategy that is operating at a loss, the user incurs the loss (source). By default, the loss is capped at 0.01% (source). Strategies can also block withdrawals entirely.

For more information on this problem, see finding 10 in the Trail of Bits review of the Yearn protocol.

Exploit Scenario
Charlie, a user, makes a significant deposit into the `crvRenWSBTC` `DInterest` contract. After maturation, he tries to withdraw his deposit, plus interest, to reinvest it elsewhere. The first strategy in the withdrawal queue of the underlying Yearn vault is currently operating at a loss. Charlie's withdrawal is too large to be covered by the free tokens in the vault. The strategy is forced to execute the withdrawal and realize the loss. The loss is less than 0.01% and within the default max loss of Yearn's `withdraw` function. Instead of gaining the promised interest, Charlie incurs a loss of up to 0.01%. Charlie would not have realized the loss if he had been aware of the idiosyncrasies of Yearn, checked the state of the vault, and withdrawn at a better time.

Recommendations
Short term, expand the 88mph user documentation to include guidance on using Yearn vaults and on the associated risks. Refer to the comments on Yearn's `withdraw` and `deposit` functions, which contain important guidance on their use.

Specifically, inform users that they should consider taking the following precautions:

- Check the free token balance in the vault. If the number of free tokens covers the withdrawal, it will succeed.

- Check the strategies in the withdrawal queue. A user could realize a loss if the topmost strategies are operating at a loss and the number of free tokens does not cover the withdrawal.

Consider setting the max loss parameter of Yearn's withdrawal function to zero. However, this could block more withdrawals.

Long term, carefully review the documentation of all protocols that the 88mph protocol interacts with and surface the risks and warnings of using these protocols to 88mph users.

References

- https://github.com/trailofbits/publications/blob/master/reviews/YearnV2Vaults.pdf
- Yearns withdraw function is fairly short and easy to understand

16. Sponsor payout uses two transfers when only one is required

Severity: Informational
Type: Optimization
Target: `Sponsorable.sol`

Difficulty: Low
Finding ID: TOB-88MPH-016

Description
`Sponsorable._paySponsor` transfers tokens from the sender of the meta-transaction to the contract itself and then to the sponsor (figure 16.1). Using two transfers is unnecessary and wastes gas, as a transfer from the sender to the sponsor can be completed in a single step. Only the caller of `transferFrom` needs the sender's approval.

```
        // transfer tokens from sender
        token.safeTransferFrom(sender, address(this), sponsorFeeAmount);

        // transfer tokens to sponsor
        token.safeTransfer(sponsor, sponsorFeeAmount);
```

Figure 16.1: `Sponsorable.sol#L144-148`

Recommendations
Short term, revise `_paySponsor` to transfer tokens from the sender to the sponsor in a single step.

Long term, investigate all transfers that are done in sequence and whether they can be combined into a single transfer.

17. ERC20Wrapper's transferFrom function ignores the sender argument

Severity: High
Type: Undefined Behavior
Target: `ERC20Wrapper.sol`

Difficulty: Low
Finding ID: TOB-88MPH-017

Description
Users calling `ERC20Wrapper.transferFrom(sender, receiver, amount)` can pass another user's address as the `sender` argument; the function should transfer tokens from the `sender`. However, the function ignores the `sender` argument and instead transfers tokens from the function's caller. `ERC20Wrapper` does not correctly implement the ERC20 standard.

This is caused by a bug in the `_transfer` function (figure 17.1). The first argument to `parentMultitoken.wrapperTransfer` is `msg.sender`. It should be `sender`.

```
function _transfer(
    address sender,
    address recipient,
    uint256 amount
) internal virtual {
    parentMultitoken.wrapperTransfer(
        msg.sender,
        recipient,
        tokenID,
        amount
    );
    emit Transfer(sender, recipient, amount);
}
```

Figure 17.1: `_transfer` function in `ERC20Wrapper.sol#L218-230`

Exploit Scenario
Alice, Bob, and Charlie are users of the 88mph protocol. Bob has given Alice an allowance to transfer 1,000 of his `ExampleTokens`. `ExampleToken` is a deployed `ERC20Wrapper`. Alice calls `ExampleToken.transferFrom(bobsAddress, charliesAddress, 500)` to transfer 500 of Bob's tokens to Charlie. Unexpectedly, 500 of Alice's tokens are transferred to Charlie instead.

Recommendation
Short term, replace `msg.sender` with `sender` in the `_transfer` function.

Long term, carefully investigate all ERC20 token implementations in the codebase to ensure that they correctly implement the ERC20 standard.

A. Vulnerability Classifications

Vulnerability Classes	
Class	**Description**
Access Controls	Related to authorization of users and assessment of rights
Auditing and Logging	Related to auditing of actions or logging of problems
Authentication	Related to the identification of users
Configuration	Related to security configurations of servers, devices, or software
Cryptography	Related to protecting the privacy or integrity of data
Data Exposure	Related to unintended exposure of sensitive information
Data Validation	Related to improper reliance on the structure or values of data
Denial of Service	Related to causing a system failure
Error Reporting	Related to the reporting of error conditions in a secure fashion
Patching	Related to keeping software up to date
Session Management	Related to the identification of authenticated users
Timing	Related to race conditions, locking, or the order of operations
Undefined Behavior	Related to undefined behavior triggered by the program
Optimization	Related to unnecessarily expensive operations

Severity Categories	
Severity	**Description**
Informational	The issue does not pose an immediate risk but is relevant to security best practices or Defense in Depth.
Undetermined	The extent of the risk was not determined during this engagement.
Low	The risk is relatively small or is not a risk the customer has indicated is important.

Medium	Individual users' information is at risk; exploitation could pose reputational, legal, or moderate financial risks to the client.
High	The issue could affect numerous users and have serious reputational, legal, or financial implications for the client.

Difficulty Levels	
Difficulty	**Description**
Undetermined	The difficulty of exploitation was not determined during this engagement.
Low	The flaw is commonly exploited; public tools for its exploitation exist or can be scripted.
Medium	An attacker must write an exploit or will need in-depth knowledge of a complex system.
High	An attacker must have privileged insider access to the system, may need to know extremely complex technical details, or must discover other weaknesses to exploit this issue.

B. Code Maturity Classifications

Code Maturity Classes	
Category Name	**Description**
Access Controls	Related to the authentication and authorization of components
Arithmetic	Related to the proper use of mathematical operations and semantics
Assembly Use	Related to the use of inline assembly
Centralization	Related to the existence of a single point of failure
Upgradeability	Related to contract upgradeability
Function Composition	Related to separation of the logic into functions with clear purposes
Front-Running	Related to resilience against front-running
Key Management	Related to the existence of proper procedures for key generation, distribution, and access
Monitoring	Related to the use of events and monitoring procedures
Specification	Related to the expected codebase documentation
Testing & Verification	Related to the use of testing techniques (unit tests, fuzzing, symbolic execution, etc.)

Rating Criteria	
Rating	**Description**
Strong	The component was reviewed, and no concerns were found.
Satisfactory	The component had only minor issues.
Moderate	The component had some issues.
Weak	The component led to multiple issues; more issues might be present.
Missing	The component was missing.

Not Applicable	The component is not applicable.
Not Considered	The component was not reviewed.
Further Investigation Required	The component requires further investigation.

C. Code Quality

The following recommendations are not associated with specific vulnerabilities. However, they enhance code readability and may prevent the introduction of vulnerabilities in the future.

General Recommendations

- **The use of the term `stablecoin` throughout is legacy and now confusing.** Consider renaming it to `asset` or `underlying`.

`YVaultMarket.sol`

- **The `totalValue` functions contain duplicated code.** Consider having the first `totalValue` function, `totalValue()`, call the second, `totalValue(uint256)`, with the parameter `vault.pricePerShare()`.

`AaveMarket.sol`

- **The `totalValue` functions contain duplicated code.** Consider having the second `totalValue` function, `totalValue(uint256)`, call the first, `totalValue()`.

D. Fix Log

On August 17, 2021, Trail of Bits reviewed the fixes and mitigations implemented by the 88mph team to address the issues identified in this report. The 88mph team fixed 11 issues and partially fixed four issues reported in the original assessment. Two issues were not fixed. We reviewed each of the fixes to ensure that the proposed remediation would be effective. For additional information, please refer to the detailed fix log.

#	Title	Severity	
1	The interest oracle's money market is not validated upon DInterest initialization	Medium	Fixed (22901ba6)
2	Lack of return value check on transfer and transferFrom	Informational	Fixed (d0c47909)
3	Lack of two-step process for contract ownership transfers	Informational	Partially fixed (4f4fbc2b)
4	Users cannot specify a minimum desired interest	Medium	Fixed (2e81c399)
5	Withdrawing from Yearn to DInterest in a single step can save gas	Informational	Fixed (07c05247)
6	Linearization of exponential compounding could lead to insolvency	High	Partially fixed (6c6cd151)
7	Initialization functions can be front-run	High	Fixed (66d86f56)
8	Lack of contract existence check on delegatecall	Informational	Not fixed
9	Inconsistent validation of money markets' rewards address	Low	Fixed (ec665342)
10	Solidity compiler optimizations can be problematic	Undetermined	Not fixed
11	Redundant addition of zero value in the Harvest money market	Informational	Fixed (abc5eba1)
12	Lack of documentation concerning Rescuable base contract could result in exploitable modifications	Informational	Fixed (cc8bc566)

13	Modifications make the safeApprove function unsafe	Informational	Fixed (bf435bf8)
14	Transferring the entire balance of a contract has unintended consequences	Low	Partially fixed (54ad4329)
15	Users are not informed of the pitfalls of using Yearn vaults	Medium	Partially fixed (c15b12d2)
16	Sponsor payout uses two transfers when only one is required	Informational	Fixed (41cb7c43)
17	ERC20Wrapper's transferFrom function ignores the sender argument	High	Fixed (2fdfecf9)

Detailed Fix Log

TOB-88MPH-001: The interest oracle's money market is not validated upon DInterest initialization
Fixed. The 88mph team removed the `moneyMarket` state variable from `DInterest`. It is now fetched from the interest oracle when needed and can no longer become inconsistent. (22901ba6)

TOB-88MPH-002: Lack of return value check on transfer and transferFrom
Fixed. The 88mph team replaced all calls to the unsafe `transfer` and `transferFrom` functions in the `xMPH` contract with their safe equivalents. (d0c47909)

TOB-88MPH-003: Lack of return value check on transfer and transferFrom
Partially fixed. The 88mph protocol is now using a modified version of `BoringOwnable` for the `DInterest` and `Vesting02` contracts. We reviewed the original `BoringOwnable` contract and 88mph's modified version. However, the `MPHToken`, `NFT`, and `MoneyMarket` contracts still use OpenZeppelin's `OwnableUpgradeable`. (4f4fbc2b)

TOB-88MPH-004: Users cannot specify a minimum desired interest
Fixed. The 88mph team added versions of `deposit`, `topupDeposit`, and `rolloverDeposit` that take an additional `minimumInterestAmount` parameter and revert if the actual interest is below this parameter's value. The team also modified the sponsored versions of these functions to take an additional `minimumInterestAmount` parameter. (2e81c399)

TOB-88MPH-005: Withdrawing from Yearn to DInterest in a single step can save gas
Fixed. The 88mph protocol now transfers funds directly from the Yearn vault to the caller, thereby skipping an expensive and unnecessary second transfer. (07c05247)

TOB-88MPH-006: Linearization of exponential compounding could lead to insolvency
Partially fixed. The 88mph team updated all files from Solidity 0.8.3 to 0.8.4 and replaced the `DecMath` library with an external library, `prb-math`, wherever `DecMath` is used. Finally, the team updated the calculations for the incoming value (for the calculation of the EMA), the interest amount for new deposits, and the interest to distribute to yield token holders making an early withdrawal. While we did check that all updates were made correctly, we were not able to achieve full certainty that *all* relevant updates have been made. Furthermore, the use of a new, unaudited arithmetic library poses risks; we recommend conducting a security review of `prb-math`. (`6c6cd151`)

TOB-88MPH-007: Initialization functions can be front-run
Fixed. The 88mph protocol now uses the `hardhat-deploy` plug-in's ability to make a `delegatecall` to the logic contract within the proxy's deployment call. (`66d86f56`)

TOB-88MPH-009: Inconsistent validation of money markets' rewards address
Fixed. The 88mph protocol now consistently checks whether the rewards addresses of the money markets are non-zero addresses. (`ec665342`)

TOB-88MPH-011: Redundant addition of zero value in the Harvest money market
Fixed. The 88mph protocol no longer includes the `HarvestMarket`'s liquidity token balance in the total value calculation. The team has properly documented the reasoning behind this change. (`abc5eba1`)

TOB-88MPH-012: Lack of documentation concerning Rescuable base contract could result in exploitable modifications
Fixed. The 88mph team added comments that warn developers of important security implications of `Rescuable::_authorizeRescue()`. (`cc8bc566`)

TOB-88MPH-013: Modifications make the safeApprove function unsafe
Fixed. The 88mph team replaced the unsafe `safeApprove` function with `safeIncreaseAllowance`, which mitigates the `approve` front-running attack and is compatible with tokens like `USDT`. We reviewed the implementation of `safeIncreaseAllowance` and its call sites. (`bf435bf8`)

TOB-88MPH-014: Transferring the entire balance of a contract has unintended consequences
Partially fixed. The 88mph team modified the `claimRewards` function in both the `BProtocolMarket` and the `CompoundERC20Market` contracts. These contracts now transfer only the amount received by `claimComp` to the `rewards` address. The team also modified the `withdraw` function in the `YVaultMarket` contract so that it transfers only the amount withdrawn from the Yearn vault to the caller. However, the `withdraw` function in the `HarvestMarket` contract still transfers the entire balance of the contract without

documentation of the reasoning or intent behind the transfer. Other undocumented uses of `balanceOf(address(this))` remain. We recommend reviewing all uses of `balanceOf(address(this))` to confirm and document their safety during attacker-controlled balance increases. (54ad4329)

TOB-88MPH-015: Users are not informed of the pitfalls of using Yearn vaults
Partially fixed. The 88mph protocol now calls Yearn's `withdraw` function with a maximum allowed loss of zero. The integration with Yearn is still risky, and its risks to 88mph users are still not properly documented. Withdrawals from Yearn can still fail. We recommend deferring the integration with Yearn at the initial deployment of the protocol and carefully investigating whether the benefits of an integration with Yearn outweigh the risks. (c15b12d2)

TOB-88MPH-016: Sponsor payout uses two transfers when only one is required
Fixed. The 88mph team modified the code so that tokens are transferred directly from the sender to the sponsor. (41cb7c43)

TOB-88MPH-017: ERC20Wrapper's transferFrom function ignores the sender argument
Fixed. The 88mph team modified `ERC20Wrapper`'s `_transfer` function to correctly use `sender` rather than `msg.sender`. (2fdfecf9)