Dedaub
Security Technology for Smart Contracts

# Mushrooms Finance - Helvella Field Vault and Strategy

## Smart Contract Security Assessment

Date: Mar. 17, 2021

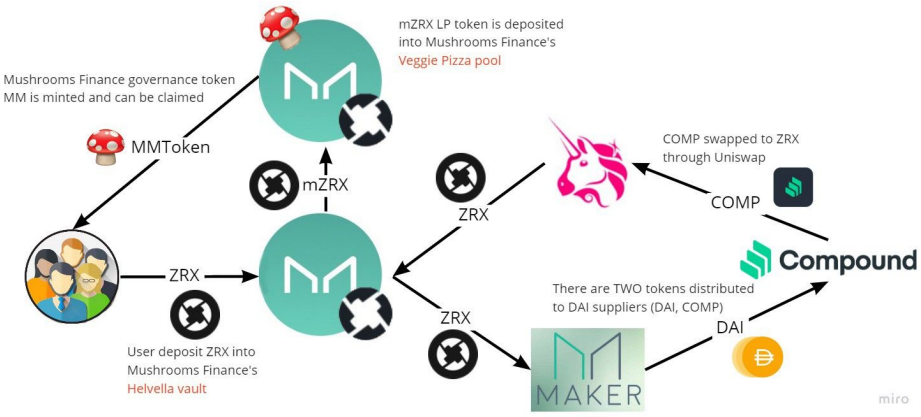
# Preliminaries

Mushrooms finance is a yield farming platform encoding different yield farming strategies. Most of these make use of MakerDAO CDPs, but also interact with many other common DeFi protocols in complex ways. Dedaub was commissioned to perform a security audit on two of the smart contracts of which Mushrooms finance is built upon: A vault, together with its corresponding strategy. Four auditors worked on this task. We reviewed the code in significant depth, assessed the economics of the protocol and processed it through automated tools. We also decompiled the code and analyzed it, using our static analysis (incl. symbolic execution) tools, to detect possible issues.

# Setting and Caveats

Mushrooms finance is built upon several smart contracts, with significant reuse from other known protocols. Dedaub's audit is however limited to two smart contracts, a strategy with corresponding vault referred to as "Helvella Field". It is understood that there is significant engineering similarity to the other vaults and strategies, and that some other contracts have been audited separately by third parties.

The strategy Dedaub has audited is depicted in a recent blog post, and is reproduced below:



A user can deposit ZRX tokens into the “Helvella” vault through the Mushrooms controller contract. The user gets back a corresponding amount of mZRX token representing the user’s share of the vault. The ZRX is then used to open or deposit into a Maker CDP to get DAI. The DAI is then staked on Compound to get cDAI. The Compound position together with the original CDP are managed by keeper bots. These primarily leverage or deleverage the compound position in DAI/cDAI. The COMP tokens that are earned are then “harvested” by the fund manager, swapped to ZRX via Uniswap and redeposited into the original CDP, repeating the cycle. A fraction of the harvest is also converted to MMToken, which is kept by the system.

The audit focused on security, establishing the overall security model and its robustness and also crypto-economic issues. Functional correctness (e.g., that the calculations are correct) was a secondary priority. Functional correctness relative to low-level calculations (e.g., units) is generally most effectively done through thorough testing. The audit is of the smart contract code, not of the underlying strategy and should not be used as an indicator of the strategy’s viability.

## Centralization Aspects

*[This section is included for reference to third party users, as the commissioner of the audit is most probably already aware of this information. ]*

Many of the roles in the system such as `keeper`, `timelock` and `governance` wield power over users. Any one of these has access to the users’ funds either directly or indirectly. For instance, the Mushrooms Deployer contract (an EOA) is used as governance both for the strategy and for the timelock contracts. It can acquire all the users’ funds by, for instance, replacing the controller contract with another implementation.

# Vulnerabilities and Functional Issues

This section details issues that affect the functionality of the contract. Dedaub generally categorizes issues according to the following severities, but may also take other considerations into account such as impact or difficulty in exploitation:

| Category | Description |
|---|---|
| **Critical** | Can be **profitably exploited by any knowledgeable third party attacker** to drain a portion of the system's or user's funds. |
| **High** | Third party attackers may block the system or cause the system or users to lose funds. |
| **Medium** | Examples:<br>1) User or system funds can be lost when third party systems misbehave.<br>2) DoS, under specific conditions. |
| **Low** | Examples:<br>1) Breaking important system invariants, but without apparent consequences.<br>2) Buggy functionality for trusted users where a workaround exists.<br>3) Security issues which may manifest when the system evolves. |

Issue resolution includes "dismissed", by the client, or "addressed", per the auditors.

## Critical Severity

*[No critical severity issues]*

## High Severity

| Description | Status |
| --- | --- |
| *[This issue is classified as "high severity" due to the potential for appropriating funds. However, its impact is limited and its preconditions currently make it seem economically less than likely. Conditions change, however, and we do not recommend dismissing vulnerabilities based on a current pricing analysis.]*<br><br>Function `withdraw` in the vault contract is public but eventually leads to a withdrawal in the strategy contract. The strategy will take its fees (currently 0.2%) and convert them to MM. This conversion is flash-loan attackable. The scenario is:<br><br>- The attacker (who is a customer who has earlier deposited a large sum) gets a flash loan and distorts a pool (probably ZRX-WETH) along the MM-want swap path.<br><br>- The attacker does the withdrawal, the strategy is swapping its 0.2% fees but gets very little MM in return because of the distorted pool: the attacker has made ZRX appear to be a non-desirable coin.<br><br>- The pool is now skewed relative to market rates. The attacker does an inverse swap, restores the pool (makes profit by getting ZRX for nearly nothing), repays flash loan.<br><br>Currently, this attack seems unlikely. First, the fees come from the attacker's funds to begin with. So the attacker has to have been a legitimate customer profiting from the service. Second, the 0.2% fees have to be high enough to warrant flash loan costs, gas costs, Uniswap reverse swap costs, and the attacker's time. Future market conditions, however, may make the attack more feasible. For instance, DyDx may conceivably expand its zero-fee flash loans to include ZRX. | **Dismissed: threat considered and assessed to be low** |

| | |
|---|---|
| *[This issue is a design warning, affecting clients of the system and not the system itself. A number of exploits this year originated from flashloan APIs with this deficiency.]*<br><br>The flash loan API in `MMVault` (not used by the system, but exported to clients of the system) is prone to misuse and, thus, possibly dangerous for clients. The current callback API does not include the originator of the flashloan in its arguments.<br>(Note how the DyDx flash loan API that the strategy contract itself uses *does* provide an `origin` argument.) | **Dismissed: threat considered and assessed to be low** |

## Medium Severity

| Description | Status |
|---|---|
| `StrategyMakerZRXV1` grants infinite approvals to a number of third party contracts, including to maker Dao `join` contracts, compound `cDai`, `UniswapV2 Router`, and a third party `dydx` flashloan wrapper. If any of these services gets compromised in the future (which presently seems unlikely), Mushroom's funds may also be stolen. | **Open** |
| It is unclear how well the CDP would be managed in periods of volatility for the collateral token (ZRX):<br><br>• As the price of ZRX goes up, the collateralization ratio allows for more DAI to be withdrawn, allowing for larger returns. It is understood that this opportunity is not exploited.<br>• On the other hand, the minimum collateralization ratio may actually cause the CDP to be liquidated as `keepMinRatio()` is reverted. | **Open** |
| The check `msg.sender == tx.origin` is not considered a future-proof practice for ensuring that the caller is not a contract. Although the call is currently safe, | **Dismissed: threat** |

| | |
|---|---|
| there are proposals that would violate that property. We emphasize that the threat is theoretical only, currently.<br><br>This check is used in the `onlyBenevolent` modifier in the strategy contract and if it is subverted it will enable flash loan attacks on `harvest()`. If deployed as-is, the programming team should watch closely for Ethereum extensions and have an upgradability strategy in place. | **considered and assessed to be low** |

## Low Severity

| Description | Status |
|---|---|
| `StrategyMakerZRXV1.onFlashLoan` returns a hard-coded constant to indicate success of the flash loan operation. We recommend refactoring this out. Having a constant field that stores the computed hash will result in equally efficient code but with lower chances of mistake. | **Open** |
| Function `withdraw(IERC20 _asset)` (for non-want withdrawals) checks that the asset is not the cToken. But it does not check that the asset is not the want token (ZRX). Seems like a good practice to exclude this case as well, since it is covered by an (easily confused) overloaded `withdraw` function. | **Dismissed: method is only callable by controller. Deemed unlikely to misuse.** |
| Most `withdraw…()` functions in `StrategyMakerZRXV1` either do not return the impact on the balances (which is something that clients typically query separately) or have such balances declared in their API but return 0. Refactoring these APIs will ease the evolution of the system. | **Open** |
| In the `StategyCmpDaiBase` constructor, a call to `IComptroller.enterMarkets` is made but the error codes that are returned are not being checked. While the | **Open** |

| | |
|---|---|
| market being entered is constant and well-known (`CDAI`) it is good practice to check that there were no unexpected errors. | |

## Other/Advisory Issues

This section details issues that are not thought to directly affect the functionality of the project, but we recommend addressing.

| Description | Status |
|---|---|
| Suboptimal software engineering practices, e.g.:<br><br>- No publicly available repo<br>- Limited technical documentation<br>- No test cases (available to us) | **Open** |
| Use of floating pragmas: The floating pragmas `pragma solidity ^0.6.7;` and `pragma solidity ^0.6.12;` are used in `MMVault` and `StrategyMakerZRXV1` respectively, allowing the former to be compiled with any of the 0.6.7 - 0.6.12 versions of the Solidity compiler - the latter can only be compiled with v0.6.12 as this is the last release of v0.6.x. Although the differences between these versions are small, floating pragmas should be avoided and the pragma should be fixed to the version that will be used for the contracts' deployment. | **Open** |
| MMVault was compiled with the Solidity compiler `v0.6.7` which has some known minor issues. We have reviewed the issues but do not believe them to affect the contract. More specifically, at the time of writing, there are 5 known compiler bugs associated with the Solidity compiler `v0.6.7`:<br><br>• Copying an empty `bytes` or `string` array from memory to storage can cause data corruption. (No such copy is ever performed)<br>• Direct assignments of storage arrays with an element size <= 16 bytes (more than one values fit in one 32 byte word) are not correctly cleared if the length of the newly assigned value is smaller than the length of the previous one. (No such array is ever stored, other than the storage initialization of `StrategyCmpdDaiBase:DATA_LEVERAGE` and `StrategyCmpdDaiBase:DATA_DELEVERAGE` which are not affected by the bug.) | **Open** |

| | |
|---|---|
| • Missing escaping in string literals. (Not applicable as the contract does not use `ABIEncoderV2`.)<br>• Accessing elements of array slices whose base types are dynamically encoded can result in invalid data being read (this requires ABIEncoderV2). (Not applicable as contract does not use `ABIEncoderV2`)<br>• ImplicitConstructorCallvalueCheck (Not applicable - an explicit constructor is defined in the contract). | |
| `StrategyMakerZRXV1` was compiled with the Solidity compiler `v0.6.12` which has some known minor issues. We have reviewed the issues but do not believe them to affect the contract. More specifically, at the time of writing, there are 2 known compiler bugs associated with the Solidity compiler `v0.6.12`:<br><br>• Copying an empty `bytes` or `string` array from memory to storage can cause data corruption. (No such copy is ever performed)<br>• Direct assignments of storage arrays with an element size <= 16 bytes (more than one values fit in one 32 byte word) are not correctly cleared if the length of the newly assigned value is smaller than the length of the previous one. (No such assignments are present in the code) | **Open** |
| Gas tokens are used in this system, however we expect that these will be deprecated soon due to EIP-3298. | **Open** |
| Dead code: `MMVault.harvest()` is only callable by the controller, but the current controller never seems to call it. | **Open** |
| Several storage fields could be declared constant, as they are initialized at compile time and are never assigned during contract execution:<br><br>• **`MMVault`**: `DAY_SECONDS`, `lockWindowBuffer`<br>• **`StrategyMakerBase`**: `dssCdpManager`, `daiJoin`, `jug`, `vat`, `debtToken`, `eth_usd`, `ratioBuffMax` | **Open** |

| | |
|---|---|
| • **StrategyCmpdDaiBase**: DATA_LEVERAGE, DATA_DELEVERAGE, dydxFlashloanWrapper, colFactorLeverageBufferMax, colFactorSyncBufferMax<br>• **StrategyMakerZRXV1**: zrx_collateral, zrx_eth, zrx_collateral_decimal, zrx_ilk, zrx_apt, zrx_price_decimal, zrx_price_eth<br><br>Similarly, several storage fields could be declared immutable, as they are initialized during contract creation and are never assigned during contract execution:<br><br>• **MMVault**: token<br>• **StrategyBase**: want<br>• **StrategyMakerBase**: collateral, collateralDecimal, gemJoin, collateralOracle, collateralIlk, priceFeed, collateralPriceDecimal, collateralPriceEth | |

## Disclaimer

The audited contracts have been analyzed using automated techniques and extensive human inspection in accordance with state-of-the-art practices as of the date of this report. The audit makes no statements or warranties on the security of the code. On its own, it cannot be considered a sufficient assessment of the correctness status of the contract. While we have conducted an analysis to the best of our ability, it is our recommendation for high-value contracts to commission several independent audits, as well as a public bug bounty program. Finally, this audit report should not be interpreted as investment advice.

## About Dedaub

Dedaub offers technology and auditing services for smart contract security. The founders, Neville Grech and Yannis Smaragdakis, are top researchers in program analysis. Dedaub's smart contract technology is demonstrated in the contract-library.com service, which decompiles and performs security analyses on the full Ethereum blockchain.