BAILSEC.IO

OFFICE@BAILSEC.IO

X: @BAILSECURITY

TG: @HELLOATBAILSEC

FINAL REPORT:

DEFI.MONEY LeverageZap

July 2024

Disclaimer:

Security assessment projects are time-boxed and often reliant on information that may be provided by a client, its affiliates, or its partners. As a result, the findings documented in this report should not be considered a comprehensive list of security issues, flaws, or defects in the target system or codebase.

The content of this assessment is not an investment. The information provided in this report is for general informational purposes only and is not intended as investment, legal, financial, regulatory, or tax advice. The report is based on a limited review of the materials and documentation provided at the time of the audit, and the audit results may not be complete or identify all possible vulnerabilities or issues. The audit is provided on an "as-is," "where-is," and "as-available" basis, and the use of blockchain technology is subject to unknown risks and flaws.

The audit does not constitute an endorsement of any particular project or team, and we make no warranties, expressed or implied, regarding the accuracy, reliability, completeness, or availability of the report, its content, or any associated services or products. We disclaim all warranties, including the implied warranties of merchantability, fitness for a particular purpose, and non-infringement.

We assume no responsibility for any product or service advertised or offered by a third party through the report, any open-source or third-party software, code, libraries, materials, or information linked to, called by, referenced by, or accessible through the report, its content, and the related services and products. We will not be liable for any loss or damages incurred as a result of the use or reliance on the audit report or the smart contract.

The contract owner is responsible for making their own decisions based on the audit report and should seek additional professional advice if needed. The audit firm or individual assumes no liability for any loss or damages incurred as a result of the use or reliance on the audit report or the smart contract. The contract owner agrees to indemnify and hold harmless the audit firm or individual from any and all claims, damages, expenses, or liabilities arising from the use or reliance on the audit report or the smart contract.

By engaging in a smart contract audit, the contract owner acknowledges and agrees to the terms of this disclaimer.

1. Project Details

Important:

Please ensure that the deployed contract matches the source-code of the last commit hash.

Project	defi.money - LeverageZap
Website	defi.money
Language	Solidity
Methods	Manual Analysis
Github repository	https://github.com/defidotmoney/dfm-contracts/blob/07a030783bc5d92e37eec54f3b3d3378750aad49/contracts/periphery/zaps/LeverageZap.sol
Resolution 1	https://github.com/defidotmoney/dfm-contracts/blob/73b86b77de652eabcb1feb474d75f758408cb816/contracts/periphery/zaps/LeverageZap.sol

2. Detection Overview

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
High				
Medium	1	1		
Low	2	1		1
Informational	2			2
Governance				
Total	5	2		3

2.1 Detection Definitions

Severity	Description
High	The problem poses a significant threat to the confidentiality of a considerable number of users' sensitive data. It also has the potential to cause severe damage to the client's reputation or result in substantial financial losses for both the client and the affected users.
Medium	While medium level vulnerabilities may not be easy to exploit, they can still have a major impact on the execution of a smart contract. For instance, they may allow public access to critical functions, which could lead to serious consequences.
Low	Poses a very low-level risk to the project or users. Nevertheless the issue should be fixed immediately
Informational	Effects are small and do not post an immediate danger to the project or users
Governance	Governance privileges which can directly result in a loss of funds or other potential undesired behavior

3. Detection

LeverageZap

LeverageZap is a contract that facilitates recursive borrowing in a gas-efficient manner by using flash loans and swaps. It allows users to perform creation, adjustment and closing of loans via CDP, with the use of flash loans provided by the stablecoin for leverage. The odosV2 swap aggregator helps to convert the flash-loaned stablecoin to the collateral token and vice versa for leveraging up/down.

The most important part of this security review is to ensure that this contract does not allow any abusive behavior on the main CPD architecture. The reason for this is trivial: As soon as users allow delegation for their position in the main contract, the LeverageZap contract can alter this position. Therefore it must be prevented under all circumstances that a malicious user can modify the position from other users via the LeverageZap contract. Any position altering should only be possible by the original position owner.

Privileged Functions:

- None

Issue_01	SafeApprove will revert if two markets have the same collateral token
Severity	**Medium**
Description	The function _getCollateralOrRevert approves the maximum amount of collateral token to the Controller contract: https://github.com/defidotmoney/dfm-contracts/blob/07a030783bc5d92e37eec54f3b3d3378750aad49/contracts/periphery/zaps/LeverageZap.sol#L231 However, the use of safeApprove will fail when the previous allowance to the Controller address is not zero. When more than one market shares the same collateral token within the protocol, the LeverageZap contract will have to make the same approval to the same contract twice, which will make the transaction revert. The impact will be that the LeverageZap contract won't be able to interact with some of the markets that have the same collateral token, causing a DoS which effectively renders the leverage possibility unusable.
Recommendations	Change the safeApprovecall to an approve call.
Comments / Resolution	Resolved by using forceApprove().

Issue_02	MEV bot can sandwich a transaction to make it revert
Severity	**Low**
Description	When a user wants to open a loan or increase an existing one, there's a subtraction done to calculate the debt amount to borrow from the protocol. This operation happens twice, at lines 179 and 195: https://github.com/defidotmoney/dfm-contracts/blob/07a030783bc5d92e37eec54f3b3d3378750aad49/contracts/periphery/zaps/LeverageZap.sol#L179 https://github.com/defidotmoney/dfm-contracts/blob/07a030783bc5d92e37eec54f3b3d3378750aad49/contracts/periphery/zaps/LeverageZap.sol#L195 A MEV bot can exploit this by sandwiching the transaction of a user by sending a huge amount of stablecoin directly to the LeverageZap contract. This will cause the subtraction to underflow, reverting the whole transaction and preventing the user from interacting with the protocol. In short, it is a griefing attack. Moreover, the bot can recover the stablecoin in the backrun transaction thanks to the _transferTokensToCaller() function This issue isn't exploitable as of now because the protocol only intends to deploy on Layer 2 like OP, where MEV is still not possible. However, it is possible that in the future, the OP Sequencer will become decentralized and MEV will become feasible in some layer 2s, making this issue exploitable.
Recommendations	Implement an operation to floor the subtraction result at zero, making

	it impossible to underflow when the stablecoin balance is higher than the flash loaned amount.
Comments / Resolution	Resolved with _calculateDebtAmount() that prevents underflow.

Issue_03	Lack of fee check in onFlashLoan() could cause unexpected fee payment
Severity	**Low**
Description	Based on EIP3186, it is allowed for stableCoin contract (lender) to charge a flash loan fee. That is done after the onFlashLoan() callback, where the lender will transfer back the flash loan amount with a fee from LeverageZap (borrower). In the case of LeverageZap, it is assumed that the fee is zero, as the stableCoin contract (BridgeToken.sol), does not implement the flashFee(). However, in onFlashLoan(), there is no check to ensure that the fee is zero. Furthermore, an infinite allowance is approved for stableCoin, which allows it to transfer any amount back to it. https://github.com/defidotmoney/dfm-contracts/blob/07a030783bc5d92e37eec54f3b3d3378750aad49/contracts/periphery/zaps/LeverageZap.sol#L151 The implication of this issue is that there is no mechanism in LeverageZap that will prevent an unexpected fee payment by stableCoin, that could cause the users to incur a loss.
Recommendations	Implement a check on fee to ensure it is zero or below a certain cap.

Comments / Resolution	Acknowledged and added natspec to warn any potential forks about the risk.

Issue_04	Slippage loss due to Odos interaction
Severity	**Informational**
Description	Even though the main interaction will revert if any action results in an undercollateralized position, a swap can still result in some slippage loss. If the slippage loss is not too high to prevent the main interaction, users will completely bear this loss.
Recommendations	Consider being very careful with the frontend implementation of the routingData
Comments / Resolution	Acknowledged.

Issue_05	Missing validation of flashLoan() return value
Severity	**Informational**
Description	EIP-3156 states that flashLoan() will return true on successful completion. That means it's possible for a EIP-3156 compliant lender to return false on failure, instead of reverting. However, functions like createLoan() does not check the return value of flashLoan(): https://github.com/defidotmoney/dfm-contracts/blob/07a030783bc5d92e37eec54f3b3d3378750aad49/contracts/periphery/zaps/LeverageZap.sol#L74 This could cause loan creation and other loan operations to proceed even when there is an error with the flash loan.
Recommendations	Implement a check on flashLoan() return value. Alternatively, consider acknowledging this and ensure the stablecoin (lender) always returns true on flashLoan(), e.g. using OZ implementation.
Comments / Resolution	Acknowledged as the target lender contract will be using OZ implementation.