OpenZeppelin

Holdefi Audit

OPENZEPPELIN | MARCH 29, 2021

Security Audits

Update: *OpenZeppelin completed the audit of the Holdefi platform and delivered the findings in April 2020, but the publishing of the report has been delayed until March 2021. As the DeFi ecosystem has evolved over the last year, OpenZeppelin strongly recommends another audit be conducted on the codebase prior to a mainnet launch.*

The fixes to the issues identified in this report have not been reviewed by OpenZeppelin.

Introduction

Holdefi is a lending platform where users can hold their assets and earn interest or borrow tokens and repay them after a specific period of time. Anyone can supply assets to Holdefi's liquidity pool and immediately begin earning interest.

The Holdefi team asked us to review and audit the smart contracts for their DeFi protocol. We examined the code, and here we publish our findings.

The audited commit is `f4df394d7cf6df347b1f1e9af8e2676c5933c2c9` and the files included in the scope were Holdefi, HoldefiPauser, HoldefiPrices, HoldefiSettings, CollateralsWallet, and Ownable. The SafeMath, SampleToken and SimpleMedianizer were not included in the scope.

All external code and dependencies were assumed to work as intended.

Privileged roles

The Holdefi team currently administers all aspects of the protocol to decide which assets can be used and how price feeds and rates are set. They also control various economic parameters, such as the size of the incentive used to encourage third parties to liquidate under-collateralized loans or which markets will have a promoted rate. All sensitive actions that owners and privileged roles can carry out (the most sensitive ones listed below) are instant and forced, with no opt-in nor opt-out mechanism for users of the protocol.

The owner of the `Holdefi` contract can:

- Withdraw the liquidation and promotion reserves.
- Set the promotion rate.
- Set and fix the address of the `HoldefiPrices` contract.

The owner of the contracts that inherit from the `HoldefiPauser` contract can:

- Set the pauser's address.
- Pause and unpause functionalities.

The owner of the `HoldefiPrices` contract can:

- Add stable coins.
- Set the price of an asset.

The owner of the `HoldefiSettings` contract can:

- Set the address of the `HoldefiContract`.
- Set the rates for borrow, suppliers share, value to loan, penalty, and bonus.
- Add markets.
- Deactivate markets.
- Add collaterals.

is unclear whether a single externally owned account or a multisig account will represent these roles at the time of audit. As for now, it requires users to fully trust the Holdefi team with these privileged roles.

Update*: While we were auditing this project, the Holdefi team found the following issue:*

The Holdefi contract makes use of the HoldefiPrices and the HoldefiSettings contracts to whitelist the assets allowed in the project. In those contracts, the intrinsic characteristics of the assets such as the price or the borrow power for a particular collateral asset are stored.

When someone has submitted some collateral asset using the `collateralize` functions, the value is added to the account's balance by using the ERC20 `transferFrom` function while using ERC20 tokens, and with the `msg.value` value specified in the transaction for ETH.

A problem might happen if the owner of the `HoldefiPrices` contract, which is currently the one in charge of updating the prices of the project in a centralized manner, uses the same number of digits to express the price of every asset. In that case, because different ERC20 tokens can have different number of decimals, when the `Holdefi` contract needs to perform a borrow to a collateralized user, the outcome could be wrong.

In the borrow scenario, the current borrow power is given by:

`borrowPowerScaled = A*(balance(collateral) * price(collateral)) - ∑ balanceBorrows(market, collateral) * price(market)`

Where `A = ratesDecimal/valueToLoanRate(collateral)`. In the same way, a new borrow would be calculated as:

`assetToBorrowValueScaled = amount(market) * price(market)`

Which only succeeds if `borrowPowerScaled &gt; assetToBorrowValueScaled`.

This implies that a higher balance for one of the collaterals, due to a higher `decimals` value for that ERC20 token, could allow a greater borrow in real value.

`fixedPrice(market) = price(market) * 10**(18 - tokenDecimals) * priceDigits`

Where `tokenDecimals` is the decimals used for the ERC20 token and `priceDigits` is the number of digits used to scale the price.

Furthermore, because there might be ERC20 stable tokens that do not have 18 decimals, the `addStableCoin` function from the `HoldefiPrices` contract sets up wrongly a stable asset with a hardcoded 18 decimals value into the contract.

Because the oracle is controlled by the Holdefi's owner, consider shifting the price of the assets to compensate the difference in ERC20 token decimals, and taking into account that the EIP20 Standard marks the `decimals` function as optional, allowing that some mainnet ERC20 tokens may not have a `decimals` function to call.

Additionally, consider asking as parameter the number of decimals for non 18 decimals ERC20 stable tokens when setting up the asset into the contract.

Moreover, consider refactoring the code to include the decimals as part of the calculation, which could allow the usage of an external oracle too.

In addition, consider explicitly documenting how the difference in decimals is handled in the project, and also consider implementing a bug bounty program.

JTNDZGl2JTIwY2xhc3MlM0QlMjJidG4tY29udGFpbmVyJTIyJTNFJTBBJTBBJTNDYnV0dG9uJTIw
b25jbGljayUzRCUyMmN1c3RvbXNjcm9sbCUyOCUyOSUyMiUzRSUzQ2ElMjBocmVmJTNEJTIyJ
TIzY3JpdGljYWwlMjIlMjBjbGFzcyUzRCUyMmN1c3RvbS1saW5rJTIyJTNFcmVhZCUyMGFsbCUy
MHRoZSUyMGlzc3VlcyUzQyUyRmElM0UlM0MlMkZidXR0b24lM0UlMEElMEElM0MlMkZkaXYlM0
U=

Critical Severity

TIzaW50cm9kdWN0aW9uJTIyJTIwY2xhc3MlM0QlMjJjdXN0b20tbGluayUyMiUzRSUzQyUyMFBy
ZXZpb3VzJTNDJTJGYSUzRSUzQyUyRmJ1dHRvbiUzRSUwQSUwQSUzQ2J1dHRvbiUyMG9uY2
xpY2slM0QlMjJjdXN0b21zY3JvbGwlMjglMjklMjIlM0UlM0NhJTIwaHJlZiUzRCUyMiUyM2hpZ2glMjIl
MjBjbGFzcyUzRCUyMmN1c3RvbS1saW5rJTIyJTNFbmV4dCUyMCUzRSUzQyUyRmElM0UlM0Ml
MkZidXR0b24lM0UlMEElMEElM0MlMkZkaXYlM0U=

High Severity

[H01] Owner can bypass time checks in markets and collateral assets

The `HoldefiSettings` contract is in charge of managing some of the most important actions that the admin of the system can perform, such as add and remove markets, add and remove collateral assets, set a borrow rate and set a suppliers share rate for a given market, and finally set the value to loan rate, set the penalty rate and set a bonus rate for a particular collateral asset.

Because these actions are critical in the financial behavior of the project, this contract inherits functionalities from the `Ownable` contract, and all these mentioned functions are guarded by the `onlyOwner` modifier so the owner is the only one that can modify the system's attributes, and add or remove assets from the project.

Also, because quick changes in the parameters may harm users while using the system, there is a time-off period in which the owner cannot increase its values for a particular market or collateral.

Nevertheless, there is a feasible scenario where the owner can easily bypass this process by removing and re-adding an already existent market or collateral. For instance, given a specific market, the owner can call the `removeMarket` function, which will only set the respective `isActive` flag for that market as false, and then call the `addMarket` function for that same market but with some new parameters that the owner may use for their own benefit.

Additionally, if these two calls are done in order using a high gas price and the first one with an even higher value, and then these frontrun a user's transaction performed to the `Holdefi` contract, the owner could potentially benefit from these modifications even in the short term.

Consider adding a time check to apply the same logic of the time-off period to a previously removed market or collateral so the owner cannot bypass the time restrictions while changing the parameters.

[H02] Markets are not being properly removed from the markets list

In the `HoldefiSettings` contract there are two data structures to track markets: `marketsList` which is an array of addresses, and `marketAssets`, which is a mapping that describes the properties of each market. Additionally, there are two functions to manipulate these markets: the `addMarket` function which adds new markets, and the `removeMarket` function which removes an already existent market.

The problem resides in that the `removeMarket` function does not remove a market from both mentioned data structures, as it only sets a given market as not active in the `Market` struct, but does not remove it from the `marketsList` array. This means that, when the `getMarketsList` function is called, it will list already removed markets, showing them as valid and active markets.

This could lead into an undesired manipulation of collaterals of a borrower in a specific market, that could generate market debts for inactive collaterals in that market, and could trigger other operations such as updating the promotion reserve and therefore update a market's supply index for a specific collateral in an inactive market. Also, this behavior of having a removed market that is still active could be confusing for the end-users.

Consider renaming the `removeMarket` function to `deactivateMarket`, to make its behavior clear. On all the functions that query `marketsList` consider checking if the market is inactive. If the function supports or requires to take into account inactive markets, consider clearly documenting this.

Alternatively, consider using different state variables to differentiate from active and inactive markets, instead of leaving them in a state that is not clearly defined.

Update: *Not fixed. Holdefi's statement for this issue:*

> "marketList", the borrow value of that token for the users that borrowed it, will not be calculated.
> We did not remove the market from "marketList" with awareness.
> Actually removing market or collateral in Holdefi stops users from depositing new tokens to contract but withdrawing and other functionalities are not changed because we don't want our users losing their tokens.
> Also updating the supply index and promotion reserve doesn't have any bad effect on the platform and these functions are not "sensitive operations". They just update index or reserve till current time.
> You suggested that we should check whether a market is active or not in the "clearDebts". But if we do that, the platform will have problems and we will not consider the users that borrowed that market before.

We have updated our suggestions to make them clearer.

[H03] Users can add non-existent ERC20 tokens into the promotion reserve

The `Holdefi` contract implements most of the functionalities of the Holdefi's protocol, such as the supply of assets into the system, add collateral, and to borrow other assets based on the collateral power that the account has, among others.

In particular, the `depositPromotionReserve` function, allows any account to deposit ERC20 token assets into the promotion reserve for a particular market.

The problem is that the `depositPromotionReserve` function does not check whether the market that the caller is trying to interact with is actually whitelisted by checking the `isActive` flag which enables the possibility to pass any address (except the zero address) as a possible market.

Here, an attacker could create a new contract that implements a simple `transferFrom` function that always returns `true`, so the requirement that checks the output of the real `transferFrom` function can be bypassed. By doing this, the attacker would be calling the `depositPromotionReserveInternal` function with a fake market address and number of tokens.

several parts, all checks will be bypassed, including the protections that the usage of the `SafeMath` library provides, ending up in writing in storage not only a fake deposited value of the particular ERC20 token market but also it will update the timestamp from when the inexistent assets were deposited, emulating the behavior of a real ERC20 token.

Consider restricting the call to the whitelisted markets by using the already implemented `isActive` flag, and restricting who is able to deposit ERC20 assets into the promotion reserve.

Update: *Not fixed. Holdefi's statement for this issue:*

> You have 2 suggestions for this issue: 1-implementing isActive flag 2-restricting who is able to deposit ERC20 assets first suggestion is good and we will implement it. But restricting users to deposit into promotion reserve will not change anything. Users can deposit their assets into promotion reserve and it's good for us. Also updating indexes is not bad for the protocol.

JTNDZGl2JTIwY2xhc3MlM0QlMjJidG4tY29udGFpbmVyJTIyJTNFJTBBJTBBJTNDYnV0dG9uJTIw
b25jbGljayUzRCUyMmN1c3RvbXNjcm9sbCUyOCUyOSUyMiUzRSUzQ2ElMjBocmVmJTNEJTIyJ
TIzY3JpdGljYWwlMjIlMjBjbGFzcyUzRCUyMmN1c3RvbS1saW5rJTIyJTNFJTNDJTIwUHJldmlvdX
MlM0MlMkZhJTNFJTNDJTJGYnV0dG9uJTNFJTBBJTBBJTNDYnV0dG9uJTIwb25jbGljayUzRCU
yMmN1c3RvbXNjcm9sbCUyOCUyOSUyMiUzRSUzQ2ElMjBocmVmJTNEJTIyJTIzbWVkaXVtJTIy
JTIwY2xhc3MlM0QlMjJjdXN0b20tbGluayUyMiUzRW5leHQlMjAlM0UlM0MlMkZhJTNFJTNDJTJG
YnV0dG9uJTNFJTBBJTBBJTNDJTJGZGl2JTNF

Medium Severity

[M01] Owner can overrule pauser actions

The `HoldefiPauser` contract implements the functionality to pause specific functions at will in case a problem appears in the platform. The only accounts that can perform such task are the `owner` and the `pauser` accounts.

Nevertheless, the time-off period that the pause event has to achieve is modifiable only by the owner by calling the `setPauseDuration` function and if the owner decides to suppress the

Consider setting a minimum duration for the time-off period to prevent that the pauser account loses its powers.

[M02] List of markets can endlessly grow

In the `HoldefiSettings` contract, the `addMarket` function is used to introduce new markets to the system by adding them into the `marketsAssets` mapping and in the `marketsList` array.

Given that the owner of the contract can add markets without any limit, and given that the `removeMarket` function does not remove markets properly, as it was pointed out in the issue *"[H02] Markets are not being properly removed from the markets list"*, the `marketsList` array can grow endlessly. Then, when the array is iterated through, it would cause an out of gas error, as the loop would be too large to be handled in a single transaction. This would happen in the following scenarios:

- When adding a new market, as the `marketsList` array is iterated to check whether a market already exists, preventing the market to be added
- When `clearing debts` in the `Holdefi` contract, preventing a borrower's collateral to be liquidated.

Consider limiting the size of the `marketsList` array to prevent out of gas errors. In addition to this, when removing a market using the `removeMarket` function, consider not only deactivating it in the `marketsAssets` struct, but also removing it from the `marketsList` array.

Update: *Not fixed. Holdefi's statement for this issue:*

> We explained it in H02. Your suggestion to remove the market from marketsList not only can't solve the issue but also will damage the platform.

[M03] Anyone can set the Holdefi contract's address in the CollateralsWallet contract

The `CollateralsWallet` contract is in charge of letting the `Holdefi` contract to withdraw a collateral asset if needed through the `withdraw` function. For this, the `holdefiContract`

The problem resides in the fact that the `setHoldefiContract` function does not have any restriction and can be called by any address and, thus, the `holdefiContract` storage value can be set by any external actor right after the contract creation, and will be able to call the `withdraw` function afterwards and steal the tokens held by the `CollateralsWallet` contract in any ERC20 contract, if there are any in the first place.

Consider setting the `holdefiContract` storage variable on the contract creation instead of setting it in a separate function, or consider using the `Ownable` contract for setting an owner of the `CollateralsWallet` contract so the `setHoldefiContract` can only be called by a trusted actor.

[M04] Insufficient incentives to liquidator

The `Holdefi` contract implements the liquidation process for those accounts that may have an under-collateralized balance or that may have been inactive for a whole year without interacting with the project.

The liquidation process is a very important part of every DeFi project because it allows to extinguish the problem of having the whole system under-collateralized under critical conditions of the market, and it needs a design that incentivizes its speed of execution.

Nevertheless, the only incentive that a liquidator has when calling the `liquidateBorrowerCollateral` function is to update its last time of activity. Not only that, but also it only updates the latest activity time for the collateral type involved in the liquidation process and not all the activity times of all collaterals. Liquidators that do not have the collateral involved in the contract, or any collateral at all, will not have any benefit for calling the liquidation process.

Furthermore, given that the cost of calling the `liquidateBorrowerCollateral` function will be high due to the `for` loop in the `clearDebts` function, as it was pointed out in the issue **"[M02] List of markets can endlessly grow"**, and even though buying discounted collateral assets could be considered as an incentive to the liquidators, the liquidators have a low incentive to reduce the liquidable accounts' borrows of the platform. This is because the account liquidation process is detached from the liquidated-collateral asset purchase, making that between those end up paying for the expensive liquidation process, without receiving any benefit.

Consider improving the incentive design to give the liquidators higher incentives to execute the liquidation process, or merging the functionalities from the `liquidateBorrowerCollateral` and the `buyLiquidatedCollateral` functions under one.

Update: *Not fixed. Holdefi's statement for this issue:*

> A 5% discount for buying collateral assets can incentivize the liquidator to liquidate collaterals and then buy it.

We have updated our suggestion to make it clearer.

[M05] Asset price can be reset by owner

The `addStableCoin` function of the `HoldefiPrices` contract is used to introduce new token prices to the Holdefi system with an initial price of `priceDecimal`, that will later be used for performing sensible operations such as withdrawing collaterals, borrowing market assets, and liquidating a borrower's collateral on the `Holdefi` contract.

The problem resides in that the `addStableCoin` function does not check whether an asset has already been added to the system or not, leading to scenarios where the owner of the contract could reset its value by calling this function multiple times, overwriting the current price of a given asset with the initial `priceDecimal` value.

Consider checking whether a given asset price has already been added to the system by checking its existence in the `assetPrices` mapping before calling the `setPrice` function inside the `addStableCoin` function.

[M06] Markets can become insolvent

When the value of all collateral is worth less than the value of all borrowed assets, we say a market is insolvent. The Holdefi codebase can do many things to reduce the risk of market insolvency, including: prudent selection of collateral-ratios, incentivizing third-party collateral liquidation, careful selection of which tokens are listed on the platform, etc. However, the risk of

- The price of the underlying (or borrowed) asset makes a big, quick move during a time of high network congestion — resulting in the market becoming insolvent before enough liquidation transactions can be mined. A similar situation was experienced at the beginning 2020 in the Ethereum network, and specially in Maker's pricing oracles.
- The liquidation incentives are not as strong they should be, allowing the accumulation of under-collateralized borrows.
- The price oracle temporarily goes offline during a time of high market volatility. This could result in the oracle not updating the asset prices until after the market has become insolvent. In this case, there will never have been an opportunity for liquidation to occur.
- The admin or oracle steals enough collateral that the market becomes insolvent.
- Administrators list an ERC20 token with a later-discovered bug that allows minting tokens arbitrarily. This potentially corrupt tokens can be used as collateral to borrow funds that were never intended to be used as a repayment.

In any case, the effects of an insolvent market could be disastrous. It may result in a "run on the bank" situation, with the last suppliers out losing their money.

This risk is not unique to the Holdefi project. All collateralized loans (even non-blockchain loans) have a risk of insolvency. However, it is important to know that this risk does exist, and that it can be difficult to recover from even a small dip into insolvency.

Consider adding more targeted tests for these scenarios to better understand the behavior of the protocol, and designing relevant mechanics to make sure the platform operates properly. Also consider communicating the potential risks to the users if needed.

[M07] Old owner can frontrun the ownerChanger

The `Ownable` contract is a modified version of the OpenZepplin `Ownable` contract where a third actor is included: the `ownerChanger`.

The `ownerChanger` is can accept an ownership transfer from the old owner to the `pendingOwner`.

A scenario could happen in which the old owner A tries to deceive the `ownerChanger` by calling the `transferOwnership` function with the address of the new owner B, address that the `ownerChanger` would approve, but right after the `ownerChanger` sends the transaction calling the `acceptTransferOwnership` function, the old owner A calls the `transferOwnership` function again with a higher gasPrice and passing another address C as the parameter. In that case, this last transaction would be mined first and just after that the `acceptTransferOwnership` function would confirm the ownership transfer to the undesired address C.

Although it is very unlikely that the old owner would perform such attack, consider changing the `acceptTransferOwnership` function to ask the `ownerChanger` the address of the new owner so it can be compared to the `pendingOwner` submitted by the old owner.

[M08] Excessive indirection

Due to the massive factorization of certain behaviors under the same function as addressed in the issue *"[L08] Overcomplicated return values"*, the level of indirection present severely degrades the readability of the code.

To give an example, if a liquidator calls the `liquidateBorrowerCollateral` function of the `Holdefi` contract, that same transaction would end up triggering a call to the `clearDebts` function, which then would call the `updateSupplyIndex` function, which would call afterwards the `getCurrentInterestIndex` function, and finally jump to the `HoldefiSettings` contract to call the `getInterests` function.

Right after the `updateSupplyIndex` call ends, the `updatePromotionReserve` would be called, which would call the `getCurrentPromotion` function, and would end up jumping again to the `HoldefiSettings` contract to call the `getInterests` function.

Note that the example of excessive indirection mentioned above is not the only one triggered by the `liquidateBorrowerCollateral` function, but just one of many.

Consider reducing excessive indirections throughout the code base by simplifying each function, so that they can fulfill one single and clear purpose, and also avoid over-factorizing behaviors in order to improve the readability and maintenance of the project. If there is a reason or limitation that forces this complexity, consider documenting it in the code.

Update: *Not fixed. Holdefi's statement for this issue:*

> As you mentioned, this does not pose a security risk per se. And it's not a medium severity issue. It's just a suggestion.
> Also, we can use the limited number of values in each function. If we want to simplify functions, we are faced with this error: Stack too deep, try removing local variables.

We have updated our suggestion to make it clearer.

[M09] Not using OpenZeppelin contracts

OpenZeppelin maintains a library of standard, audited, community-reviewed, and battle-tested smart contracts.
Instead of always importing these contracts, the Holdefi project reimplements them in some cases, while in other cases it just copies them.

This increases the amount of code that the Holdefi team will have to maintain and misses all the improvements and bug fixes that the OpenZeppelin team is constantly implementing with the help of the community.

In particular, the following contracts and libraries are being reimplemented or copied:

- the `Ownable` contract can be replaced with the OpenZeppelin's `Ownable` contract
- the `SafeMath` library can be replaced with the OpenZeppelin's `SafeMath` library
- The `ERC20` interface defined in line 3 of `CollateralsWallet.sol` and line 25 of `Holdefi.sol` can be replaced with the OpenZeppelin's `IERC20.sol` interface

Consider importing the OpenZeppelin contracts instead of reimplementing or copying them. These contracts can be extended to add the extra functionalities required by Holdefi.
Consider always using the full ERC interfaces so that obviously non-compliant implementations

Update: *Not fixed. Holdefi's statement for this issue:*

> If we use exactly OpenZeppelin contracts, we will miss some added features like ownerChanger. But we need them and can't remove them. For ERC20 interface, we don't need all functions so web just use a reduced version of IERC20.sol interface.

We have updated our suggestion to make it clearer.

[M10] Lack of events emission after sensitive actions

Throughout the Holdefi project codebase, there are several cases where sensitive actions are performed but there are no events being emitted, or the existent emitted events miss important parameters.
Our suggestions are:

In the `CollateralsWallet` contract:
– The `withdraw` function should emit a `CollateralAssetWithdrawn` event
– The fallback function should emit a `TransferReceived` event

In the `Holdefi` contract:
– The `updatePromotionReserve` function should emit a `PromotionReseveUpdated` event
– The `withdrawLiquidationReserve` function should emit a `LiquidationReserveWithdrawn` event
– The `withdrawPromotionReserve` function should emit a `PromotionReserveWithdrawn` event
– The `UpdateBorrowIndex` event should print the `borrowRate` received by the `getCurrentInterestIndex` function

In the `HoldefiPauser` contract:
– The `pause` function should emit a `OperationPaused` event
– The `unpause` function should emit a `OperationUnaused` event
– The `setPauser` function should emit a `PauserSet` event
– The `setPauseDuration` function should emit a `PauseDurationSet` event

`StablecoinAdded` event

As a general rule, consider emitting events appropriately when performing sensitive changes to storage variables, and consider emitting the most important variables involved in those changes.

[M11] Missing docstrings

All the contracts and functions in the Holdefi's codebase lack documentation. This hinders reviewers' understanding of the code's intention, which is fundamental to correctly assess not only security, but also correctness.

Additionally, docstrings improve readability and ease maintenance. They should explicitly explain the purpose or intention of the functions, the scenarios under which they can fail, the roles allowed to call them, the values returned, and the events emitted.

Consider thoroughly documenting all functions (and their parameters) that are part of the contracts' public API. Functions implementing sensitive functionality, even if those are not public, should be clearly documented as well. When writing docstrings, consider following the Ethereum Natural Specification Format (NatSpec).

Update: *Not fixed. Holdefi's statement for this issue:*

> This is not a bug. This is just a suggestion

JTNDZGl2JTIwY2xhc3MlM0QlMjJidG4tY29udGFpbmVyJTIyJTNFJTBBJTBBJTNDYnV0dG9uJTIw
b25jbGljayUzRCUyMmN1c3RvbXNjcm9sbCUyOCUyOSUyMiUzRSUzQ2ElMjBocmVmJTNEJTIyJ
TIzaGlnaCUyMiUyMGNsYXNzJTNEJTIyY3VzdG9tLWxpbmslMjIlM0UlM0MlMjBQcmV2aW91cyUz
QyUyRmElM0UlM0MlMkZidXR0b24lM0UlMEElMEElM0NidXR0b24lMjBvbmNsaWNrJTNEJTIyY3
VzdG9tc2Nyb2xsJTI4JTI5JTIyJTNFJTNDYSUyMGhyZWYlM0QlMjIlMjNsb3clMjIlMjBjbGFzcyUzR
CUyMmN1c3RvbS1saW5rJTIyJTNFbmV4dCUyMCUzRSUzQyUyRmElM0UlM0MlMkZidXR0b24l
M0UlMEElMEElM0MlMkZkaXYlM0U=

Low Severity

[L01] Transfer method is used to send ETH

caller's address with the `holdefiContract` address is compared.

Meanwhile, the `Holdefi` contract allows users to deposit assets as collaterals to then increase their collateral balance which allows them to borrow other assets of the platform.

In particular, if someone deposits ETH, the user has to call the `collateralize` payable function while sending the amount of ETH they want to add in the same transaction. Once the `Holdefi` contract checks if the ETH market is active, it transfers the ETH to the `CollateralsWallet` contract and stores the balance in an internal mapping.

The method used to transfer the ETH to the `CollateralsWallet` contract is a low level call, which is the current correct way to transfer ETH between addresses due to the fact that the `transfer` method relies on the fact that gas costs are always constant, but since the Istanbul hard fork, these gas prices cannot be considered constants and contracts that use them may break in the future.

Nevertheless, in all other ETH transfers to external accounts or other contracts, the `transfer` method is used. Some examples are the ones in L214, L325, L397, and L778 from the `Holdefi` contract.

Consider using the low level call to send ETH as it was implemented in L254-L255 or, even better, replacing all the current implementations to send ETH with the OpenZeppelin's `sendValue` method.

[L02] Collaterals and markets can be updated after being deactivated

The `HoldefiSettings` contract defines a mapping of `collateralAssets` to store information about collaterals, and a mapping of `marketAssets` to store information about markets. Even though in both data structures there is an `isActive` flag defined to check whether a `collateralAsset` or `marketAsset` is active or not, it is not being used on the `setValueToLoanRate`, `setPenaltyRate`, `setBonusRate`, `removeMarket`, `setSuppliersShareRate`, and the `setBorrowRate` functions, leading to scenarios where deactivated collaterals and markets can be updated, and undesired event emissions will be triggered.

a given collateral or market at the beginning of the functions mentioned above.

Update: *Not fixed. Holdefi's statement for this issue:*

> Removing a market o collateral in the Holdefi means that nobody can supply or borrow or add collateral. But old users can withdraw supply or repay borrow. So we can set new rates (VTL rate, bonus rate, …) for them.

We have updated our suggestion to make it clearer.

[L03] Borrowers are allowed to operatate after the maximum period has passed

The `Holdefi` contract establishes a maximum period of time in which a borrower can be sure that his assets will not be liquidated. This period is defined as the number of seconds in a regular year.

Nevertheless, this restriction is not being applied on all the functions, and it is only used as a limit in the `liquidateBorrowerCollateral`. This could lead into a hypothetical scenario where a borrower could continue with his borrowing for a period greater than that year in the case where there are no liquidators performing their liquidation duty.

Additionally, a borrower in such condition who has an account which has been a whole year without activity could frontrun a liquidator when the transaction that calls the `liquidateBorrowerCollateral` function enters into the mempool, maximizing the period of time for his position beyond the established limit.

Consider adding requirements in all functions related to the financial aspect of the contract to restrict the borrower's actions after the maximum period of time has been achieved instead of relying on the liquidators' actions only.

Alternatively, if there is a good reason for this design consider documenting it in the docstrings of the affected functions.

Update: *Not fixed. Holdefi's statement for this issue:*

> reason that can not get his/her debt back into the platform, suppliers be sure that their tokens will be back. There is no problem if borrower is alive and can perform any action with Holdefi contract after one year.

We have updated our suggestion to make it clearer.

[L04] OwnerChanger cannot renounce its role

The role management scheme implemented in the `Ownable` contract does not include a way for the `ownerChanger` to renounce the role they have been granted. This might become problematic in a scenario where the account wishes to renounce the role after the trusted device holding the private keys has been compromised.

Also, in the case where the current `owner` may want to transfer its ownership with the `transferOwnership` function, the `ownerChanger` could reject every single transfer at his will by not calling the `acceptTransferOwnership` function, and as the `ownerChanger` role is not transferable, the `owner` role transfer operation could get compromised.

Consider using multisig accounts for both roles to prevent that one of them becomes compromised, and adding the option to renounce the `ownerChanger` role if needed.

[L05] Pauser accounts can reset the pause period indefinitely

The `HoldefiPauser` contract implements a pausable functionality for certain cases. Each one of them are assigned to a specific index. Then, the timestamp of when a pause action has been triggered for a specific case is tracked in the `paused` array by using its index.

This feature provides child contracts with a system overhaul in cases that an unexpected event occurs and part of the code needs to be stopped. This feature can only be called by the owner or the pauser, but on the other hand, the `unpause` function can only be called by the owner.

Nevertheless, if a particular functionality has been paused, in which a `pauseDuration` should be waited to have the paused functionality back, the pauser can re-call the `pause` function and reset the timer for that functionality, being possible to extend indefinitely the paused period. If this

Consider modifying the functionality to prevent multiple resets of the pause period by the pausers accounts, or documenting this if it is the expected behavior.

[L06] Parameter's time-off periods are uneven

The `HoldefiSettings` contract uses a 10-day time-off period to prevent the owner to perform actions such as an increase in the `borrowRate`, the `suppliersShareRate`, the `valueToLoanRate`, and the `penaltyRate`.

Nevertheless, if the owner accidentally changes one of these values and sets up a lower value by mistake, a 10-day time-off period will have to be awaited before the parameters can be corrected, possibly affecting the finances of the platform in a drastic manner.

In addition to this, the awaiting period cannot be changed even by the owner. This means that once the platform is launched, if this period is too high, it will not allow the platform to compensate the dynamics of the market quickly enough.

Consider simulating and documenting the selection of these unchangeable values, and taking extra caution before updating these parameters.

Update: *Not fixed. Holdefi's statement for this issue:*

> It's not a bug. It's just a suggestion and a warning. We have to be careful about setting these parameters. But these parameters must exist for users to trust us and prove to them that we can't manipulate the platform.

[L07] Unneeded public visibility in some functions

In the Holdefi codebase, there are situations in which functions have a public visibility although those are not called by any other function from the same contract. Some of them are:

- In the `HoldefiPrices` contract, the `addStableCoin` function
- In the `HoldefiPauser` contract, the `batchPause` and the `batchUnpause` functions
- In the `HoldefiSettings` contract, the `getMarket` function. This function returns true if a certain market is whitelisted, but this information can be known by using the

functions and any similar case to `external`, and consider reducing the visibility of the `marketAssets` variable if the `getMarket` function is not removed from the codebase.

[L08] Overcomplicated return values

Throughout the project's codebase, functions that return several variables are implemented, but when calling these functions, only a few or just one return parameter is used.
Some examples are:

- The `getCollateral` function in the `HoldefiSettings` contract returns the `isActive`, `valueToLoanRate`, `penaltyRate`, and `bonusRate` attributes, but when this function is called in the `Holdefi` contract, only one of these variables is accessed, such as in the `collateralize`, `withdrawCollateral`, and `borrow` functions.
- The `getCurrentInterestIndex` function in `Holdefi` contract returns the `supplyIndex`, `supplyRate`, `borrowIndex`, `borrowRate`, and `currentTime` attributes, but only a few of these variables are being accessed, such as in the `updateSupplyIndex` and `updateBorrowIndex` functions.

Even though this does not pose a security risk, it is very difficult to understand which variables are being accessed when calling a function without checking its signature. Additionally, returning big data structures will increase the gas costs, leading to higher prices when executing transactions.

Consider identifying all the functions that follow this pattern and modularize them into smaller ones that return at most one variable each.

[L09] Lack of indexed parameters in events

Throughout the Holdefi's codebase, none of the parameters in the events defined in the contracts are indexed.

Consider indexing event parameters to avoid hindering the task of off-chain services searching and filtering for specific events.

[L10] Lack of input validation

- The `withdraw` function in the `CollateralsWallet` contract does not check that the `recipient` is not the zero address.
- The `setPrice` function in the `HoldefiPrices` contract does not check whether the `newPrice` is the same as the old price, triggering an event without any new relevant information.
- The `setPauseDuration` function in the `HoldefiPauser` contract does not check the `newPauseDuration` parameter, which could be zero or a very long period of time, possibly locking important functions of the `Holdefi` system.

Even though this issue does not pose a security risk, the lack of validation on user-controlled parameters may result in erroneous transactions considering that some clients or owners may default to sending null parameters if none are specified. Consider always adding validation checks to ensure that the parameters are always in the expected range of values.

[L11] Semantic overload

Throughout the codebase, there are cases in which a single variable has two purposes, and, based on which its value is, the code can perform differently. For example:

In the `HoldefiSettings` contract, the properties of a market or collateral are stored inside structs. These structs have a flag called `isActive` which is raised when a market or collateral is added to the whitelist, and flagged down when the same asset is removed from the whitelist.

Nevertheless, because the parameters of the assets are not erased after the `removeMarket` function is called, the `isActive` variable is used to check not only if the market exists but also if the market is active. The same analogous problem occurs with the collateral and its respective functions.

In the `HoldefiPauser` contract, an array to keep track of the timestamps of the moment in which the `pause` function for each functionality was triggered is used, but also this array is used as a flag to know when the functionality is unpaused by checking the array's getter.

This is known as Semantic Overload. If the multiple meanings of the variables and states are not totally clear when making changes to the code, it can introduce severe vulnerabilities. We strongly instead of using the same variable for different purposes.

Update: *Not fixed. Holdefi's statement for this issue:*

> You said that this is Semantic Overload and it may cause issues when it is not used properly, but you didn't explain that what is the issue.

We have updated our suggestion to make it clearer.

[L12] Default variable declaration values are used during calculations

In the codebase of the project, there are several places in which the returned variable is declared in the function definition without the explicit `return` instruction within the function's body, as addressed in the issue *"[N06] Named return variables"*, but also there are places where the local variable is not assigned with an initial value.

Usually, this does not bring problems because the default value of a variable would be overwritten by another one during the code execution, but in other cases, depending on how this variable is used, an issue may appear.

In the `getAccountTotalBorrowValue` function from the `Holdefi` contract, `totalBorrowValueScaled` is declared to represent the function's return value. This variable is then used in a `for` loop to accumulate the value of all borrows from a particular account, but during the first iteration, the default value of this variable is used for the calculation.

Similar cases can also be found in the `totalDebt`, `assetPrice`, and `assetValueScaled` variables from the `getAccountTotalBorrowValue` function, and in many other functions.

Although it does not represent a problem in the current scenario, consider always declaring variables with an initial value to reduce the attack surface.

[L13] Use of magic constants

There are several occurrences of magic constants in Holdefi's codebase. Some examples are:

[illegible]

– Lines 161, 174, 234, 248, 261, 298, 298, 298, 359, 382, 429, 464, and 481 when calling the `whenNotPaused` modifier with a parameter

Even though the meaning of some of this constants are mentioned as comments in the `HoldefiPauser` contract, there is no reference of them in the contract where the `wheNotPaused` modifier is being used, which makes the code harder to understand and maintain.

In addition to this, the references in the comments mentioned above are not consistent with the list of function codes for pausing operations in the specification, where the operation `3` is being skipped in the whitepaper.

Consider defining a constant variable for every magic constant (including booleans) in the contract where they are used, giving it a clear and self-explanatory name. For complex values, consider adding an inline comment explaining how they were calculated or why they were chosen. All of this will allow a better readability, easing the code's maintenance.

[L14] Tests not passing successfully

The testing suite finishes with three failing tests, two on the `AdminFunctionTests.js` file and one on the `OwnableTests.js` file.

Although the test suite was left outside of the audit's scope, please consider thoroughly reviewing the test suite to make sure all tests run successfully. Furthermore, it is advisable to only merge code that neither breaks the existing tests nor decreases coverage.

Update: *Not fixed. Holdefi's statement on this issue:*

> These fails are because of very little time difference, and it's not a bug. This will be pass if you rerun it. Some of our calculations are based on timestamp and because of problem with exactly syncing with blockchain (local, testnet or mainnet), sometimes some tests will fail because of minor differences.

[L15] README file is empty

developers often read, so these should be complete, clear, concise, and accurate.

Consider following Standard Readme to define the structure and contents for the README file.

Also, consider including an explanation of the core concepts of the repository, the usage workflows, the public APIs, instructions to test and deploy it, and how the code relates to other key components of the project.

Furthermore, it is highly advisable to include instructions for the responsible disclosure of any security vulnerabilities found in the project.
Consider adding a method for secure and encrypted communication with the team, like an email address with its GPG key.

Update: *Not fixed. Holdefi's statement for this issue:*

> It's not a bug as well. We will add information in the readme file of course.

JTNDZGl2JTIwY2xhc3MlM0QlMjJidG4tY29udGFpbmVyJTIyJTNFJTBBJTBBJTNDYnV0dG9uJTIw
b25jbGljayUzRCUyMmN1c3RvbXNjcm9sbCUyOCUyOSUyMiUzRSUzQ2ElMjBocmVmJTNEJTIyJ
TIzbWVkaXVtJTIyJTIwY2xhc3MlM0QlMjJjdXN0b20tbGluayUyMiUzRSUzQyUyMFByZXZpb3VzJT
NDJTJGYSUzRSUzQyUyRmJ1dHRvbiUzRSUwQSUwQSUzQ2J1dHRvbiUyMG9uY2xpY2slM0Ql
MjJjdXN0b21zY3JvbGwlMjglMjklMjIlM0UlM0NhJTIwaHJlZiUzRCUyMiUyM25vdGVzJTIyJTIwY2xh
c3MlM0QlMjJjdXN0b20tbGluayUyMiUzRW5leHQlMjAlM0UlM0MlMkZhJTNFJTNDJTJGYnV0dG9u
JTNFJTBBJTBBJTNDJTJGZGl2JTNF

Notes & Additional Information

[N01] Variables declared as uint instead of uint256

To favor explicitness, consider changing all instances of `uint` into `uint256` in the entire codebase.

[N02] Repeated access control in the code

The `CollateralsWallet` contract is used to store the collateral assets of the project and it implements a function to set the `Holdefi` contract's address, a function to withdraw the

the fallback functions check whether the caller is the `Holdefi` contract or not. This functionality could be factorized into a `onlyHoldefi` modifier as it is done in the `Ownable` contract instead of implementing the require statement in each function.

Consider creating a `onlyHoldefi` modifier and replacing the require statements in both functions with the modifier.

[N03] Not following the Checks-Effects-Interactions pattern

Throughout the project's codebase, there are situations in which the code is not following the check-effect-interaction pattern that helps to protect against reentrancy issues, such as in the `setBorrowRate` function or in the `setSuppliersShareRate` function from the `HoldefiSettings` contract.

Although in the codebase these particular cases do not pose a security risk, consider changing the order of operations to first write the state variables and then perform the external calls to other contracts for readability and consistency purposes, and to be safe against reentrancy attacks even if the called functions change.

Update: *Not fixed. Holdefi's statement for this issue:*

> We know that. We followed Checks-Effects-Interactions pattern in the whole code. But in this case, indexes should be updated with previous values, not new values.

In these cases, we still suggest to follow the defensive Checks-Effects-Interactions when calling other contracts. Consider saving the old values into local variables, and pass those when calling the external contracts. In the meanwhile, consider explaining in the code the reason of not follow the Checks-Effects-Interactions pattern.

[N04] Inconsistent coding style

Deviations from the Solidity Style Guide were identified throughout the entire codebase. Taking into consideration how much value a consistent coding style adds to the project's readability, enforcing a standard coding style with help of linter tools such as Solhint is recommended.

[N05] Floating Solidity compiler versions

Nevertheless, the Holdefi project is using a floating pragma version that could include these new vulnerabilities when the project is compiled.

Consider fixing all the dependencies to the same stable Solidity version.

[N06] Named return variables

There is an inconsistent use of named return variables across the entire codebase. Some examples can be found in the `getAccountCollateral` function from the `Holdefi` contract or in the `getCollateral` function of the `HoldefiSettings` contract.

Consider removing all named return variables, explicitly declaring them as local variables in the body of the function, and adding the necessary explicit return statements where appropriate. This should favor both explicitness and readability of the project.

[N07] Timestamp may not be reliable

The Holdefi's codebase uses the `block.timestamp` as part of the calculations and time checks.

Nevertheless, timestamps can be slightly altered by miners to favor them in contracts that have logics that depend strongly on them.

Consider taking into account this issue and warning the users that such scenario could happen. If the alteration of timestamps cannot affect the protocol in any way, consider documenting the reasoning and writing tests enforcing that these guarantees will be preserved even if the code changes in the future.

Update: *Not fixed. Holdefi's statement for this issue:*

> As you can see in the below link, If the scale of your time-dependent event can vary by 15 seconds and maintain integrity, it is safe to use a block.timestamp. Using block.timestamp for generating random numbers is not good in the lottery or this kind of context. But in our platform 10-15 seconds in 1 year doesn't change anything and miner can't make a profit.

We have updated our suggestion to make it clearer.

[N08] Unnecessary imports

In the `Holdefi` contract, consider removing the import statement for the `Ownable` contract as it is never used.

[N09] Unused events

Line 124 of `Holdefi.sol` declares a `RepayBorrow` event. As it is never emitted, consider removing the declaration or emitting the event in the appropriate place.

[N10] Misleading comments

On `Holdefi.sol`:
– Line 160: Should say 'Deposit ERC20 assets as supply'
– Line 184: The `withdrawSupply` not only handles tokens, but also ETH
– Line 592: The `getAccountSupply` function also returns the `currentSupplyIndex` variable
– Line 607: The `getAccountBorrow` function also returns the `currentBorrowIndex` variable

On `CollateralsWallet.sol`:
– Line 19: The withrdaw function not only handles tokens, but also ETH

Consider updating the comments to more accurately describe the purpose and effect of the codebase.

[N11] Misleading function and variable names

To favor explicitness and readability, some functions and variables from the whole repository may benefit from a better naming.
Our suggestions are:

In the `HoldefiPauser` contract:
– `isPause` to `isPaused`

– newUnpaused to functionsToUnpause

In the HoldefiSettings contract:
– secondsPerTenDays to periodBetweenUpdates
– getMarket to isMarketActive
– newOwnerChanger to ownerChanger
– All occurrencies where penaltyRate is part of a variable the name, to liquidationRatio
– Market to MarketSettings, to differentiate it from the Market struct defined in the Holdefi contract

In the Holdefi contract:
– All ocurrencies of totalBalance to totalBorrowedBalance

[N12] TODOs in code

On line 11 and line 38 of HoldefiPrices.sol, there are “TODO” comments that should be removed and instead tracked in the project’s issues backlog.

[N13] Typos in comments

Throughout the Holdefi’s codebase, there are a few typos in the code and in comments. We list them here.

- On line 8 of CollateralsWallet.sol, ‘collateralls’ should be ‘collaterals’
- On line 33 and on line 39 of HoldefiPauser.sol, ‘functions’ should be ‘function’
- On line 27 of Ownable.sol, ‘can not’ should be ‘cannot’
- On line 16 of HoldefiSettings.sol, ‘All these settings is callable by only owner’ should say ‘All these settings are callable only by the owner’

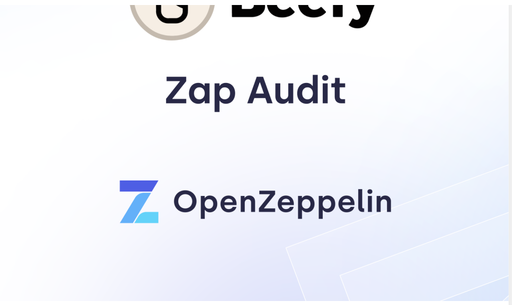
JTNDZGl2JTIwY2xhc3MlM0QlMjJidG4tY29udGFpbmVyJTIyJTNFJTBBJTBBJTNDYnV0dG9uJTIw
b25jbGljayUzRCUyMmN1c3RvbXNjcm9sbCUyOCUyOSUyMiUzRSUzQ2ElMjBocmVmJTNEJTIyJ
TIzbG93JTIyJTIwY2xhc3MlM0QlMjJjdXN0b20tbGluayUyMiUzRSUzQyUyMFByZXZpb3VzJTNDJT
JGYSUzRSUzQyUyRmJ1dHRvbiUzRSUwQSUwQSUzQ2J1dHRvbiUyMG9uY2xpY2slM0QlMjJjdX

Conclusions

No critical and 3 high severity issues were found. Some changes were proposed to follow best practices and reduce potential attack surface.

JTNDZGl2JTIwY2xhc3MlM0QlMjJidG4tY29udGFpbmVyJTIyJTNFJTBBJTBBJTNDYnV0dG9uJTIw
b25jbGljayUzRCUyMmN1c3RvbXNjcm9sbCUyOCUyOSUyMiUzRSUzQ2ElMjBocmVmJTNEJTIyJ
TIzbm90ZXMlMjIlMjBjbGFzcyUzRCUyMmN1c3RvbS1saW5rJTIyJTNFJTNDJTIwUHJldmlvdXMlM
0MlMkZhJTNFJTNDJTJGYnV0dG9uJTNFJTBBJTBBJTNDJTJGZGl2JTNF

Related Posts



Beefy Zap Audit

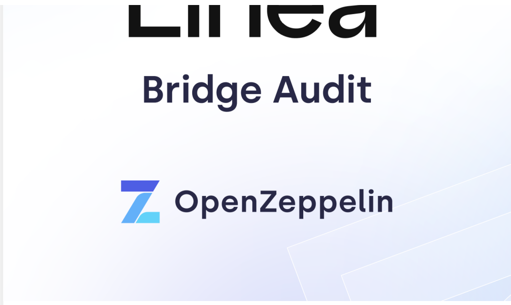
BeefyZapRouter serves as a versatile intermediary designed to execute users' orders through routes...

Security Audits

OpenBrush Contracts Library Security Review

OpenBrush is an open-source smart contract library written in the Rust programming language and the...

Security Audits



Linea Bridge Audit

Linea is a ZK-rollup deployed on top of Ethereum. It is designed to be EVM-compatible and aims to...

Security Audits