shieldify

Terplayer

Hodl

SECURITY REVIEW

Date: 15 May 2025

CONTENTS

1. About Shieldify

Positioned as the first hybrid Web3 Security company, Shieldify shakes things up with a unique subscription-based auditing model that entitles the customer to unlimited audits within its duration, as well as top-notch service quality thanks to a disruptive 6-layered security approach. The company works with very well-established researchers in the space and has secured multiple millions in TVL across protocols, also can audit codebases written in Solidity, Vyper, Rust, Cairo, Move and Go.

Learn more about us at shieldify.org.

2. Disclaimer

This security review does not guarantee bulletproof protection against a hack or exploit. Smart contracts are a novel technological feat with many known and unknown risks. The protocol, which this report is intended for, indemnifies Shieldify Security against any responsibility for any misbehavior, bugs, or exploits affecting the audited code during any part of the project's life cycle. It is also pivotal to acknowledge that modifications made to the audited code, including fixes for the issues described in this report, may introduce new problems and necessitate additional auditing.

3. About Terplayer - Hodl

HODL is a DeFi protocol built on Berachain, designed to provide Bitcoin-denominated fixed-income opportunities. By utilizing **BeraBTC** as collateral, each HODL token is backed by a growing amount of **BeraBTC**, starting from a 1:1 ratio, and crucially, the value of HODL can only increase and never fall.

Users mint HODL at the spot price with a built-in upward mechanism: minting fees, redemption fees, and donations all push the HODL price higher. In essence, HODL offers a new category of fixed-income derivative anchored in Bitcoin's long-term appreciation, while providing predictable value growth.

HODL users can also borrow **BeraBTC** against their HODL holdings at a near-100% loan-to-value (LTV) ratio. Thanks to HODL's non-depreciating price model, collateral risks are eliminated. For convenience, HODL further supports one-click leverage, enabling users to maximize their HODL position with seamless borrowing and reminting loops.

Learn more: Docs

4. Risk Classification

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

4.1 Impact

- **High** - results in a significant risk for the protocol's overall well-being. Affects all or most users

- **Medium** – results in a non-critical risk for the protocol affects all or only a subset of users, but is still unacceptable
- **Low** – losses will be limited but bearable – and covers vectors similar to griefing attacks that can be easily repaired

4.2 Likelihood

- **High** – almost certain to happen and highly lucrative for execution by malicious actors
- **Medium** – still relatively likely, although only conditionally possible
- **Low** – requires a unique set of circumstances and poses non-lucrative cost-of-execution to rewards ratio for the actor

5. Security Review Summary

The security review lasted 6 days with a total of 96 hours dedicated by 2 researchers from the Shieldify team.

Overall, the code is well-written. The audit report identified one Medium-severity and one Low-severity issue. The vulnerabilities primarily stem from missing validation checks.

The Terplayer team has done a great job with the development and has been highly responsive to the Shieldify research team's inquiries and promptly implemented the provided recommendations.

5.1 Protocol Summary

Project Name	**Terplayer - Hodl**
Repository	hodl
Type of Project	DeFi, Lending, Borrowing
Audit Timeline	6 days
Review Commit Hash	b180fa8d03bdec56d4c6aea8c3f678428dd55429
Fixes Review Commit Hash	e3e9f41e3580db584b1564cbd968ebf3a7e404b2

5.2 Scope

The following smart contracts were in the scope of the security review:

File	**nSLOC**
src/BeraBtcPreDeposit.sol	159
src/HodlDonateModal.sol	31
src/Hodl.sol	595
Total	**785**

6. Findings Summary

The following issues have been identified, sorted by their severity:

- **Medium** issues: **1**
- **Low** issues: **1**
- **Info** issues: **1**

ID	Title	Severity	Status
[M-01]	Missing Minimum Output Validation in HODL Token Purchase	Medium	Acknowledged
[L-01]	Once **start** Is Set to **true**, It Cannot Be Changed Back to **false**	Low	Fixed
[I-01]	Using Outdated Compiler Version	Info	Fixed

7. Findings

[M-01] Missing Minimum Output Validation in HODL Token Purchase

Severity

Medium Risk

Description

The HODL contract contains an issue in the `_buyHodl()` function where it fails to validate the minimum amount of HODL tokens that users should receive when purchasing. This function is called internally by the public `buy()` function.

The issue occurs because the contract updates the HODL token price based on the contract's `beraBTC` balance after each transaction through the `safetyCheck()` function. This design enforces that the HODL price can only increase and never decrease:

```
require(lastPrice <= newPrice, "The price of hodl cannot decrease");
lastPrice = newPrice;
```

However, the `_buyHodl()` function does not include a minimum output check (`minOut`) for the amount of HODL tokens to be received:

```
function _buyHodl(uint256 BTC, address receiver, address inviter)
    internal {
     liquidate();
     require(receiver != address(0x0), "Reciever cannot be 0x0 address");
     require(BTC >= MIN, "Transaction amount must be above minimum");
// Mint HODL to sender
     uint256 receiveHodl = getBuyAmount(BTC);
     beraBTC.safeTransferFrom(msg.sender, address(this), BTC);
     mint(receiver, receiveHodl);
// Missing minOut validation here
     uint256 feeAddressAmount = (BTC * FEES_BUY) / FEE_BASE_10000;
     require(feeAddressAmount > MIN, "must trade over min");
     sendBTC(FEE_ADDRESS, feeAddressAmount);
     safetyCheck(BTC);
     emit Buy(msg.sender, receiver, inviter, BTC, receiveHodl);
}
```

For example, if a block has both a `donate()` and a `buy()` transaction, the order they're processed changes the outcome. If `donate` comes before `buy()`, the `buy()` gets less `HODL` —and vice versa. Users have no way to avoid this unpredictability.

Location of Affected Code

File: src/Hodl.sol

Impact

Users can not set a max price they're willing to pay, so in a rising market, they might end up buying at a higher cost than expected.

Recommendation

Add a `minOut` parameter to let users set their minimum acceptable output.

Team Response

Acknowledged.

[L-01] Once `start` Is Set to `true`, It Cannot Be Changed Back to `false`

Severity

Low Risk

Description

The `setStart()` function in the smart contract permanently sets `start = true` without any mechanism to reverse or pause the contract's operational state. Once this is set, the contract remains in a "started" state indefinitely, preventing administrators from halting operations even in emergencies, such as detected vulnerabilities, market manipulation, or critical failures.

Location of Affected Code

File: src/Hodl.sol#L168

```
function setStart(uint256 _beraBTCAmount,address _receiver) public
    onlyOwner {
     require(!start, "Trading already started");
     require(_beraBTCAmount > 0 && _receiver != address(0x0), "Must set
        beraBTC amount and receiver address");
     require(FEE_ADDRESS != address(0x0), "Must set fee address");
     start = true;
    _buyHodl(_beraBTCAmount, _receiver, address(0x0));
     emit Started(true);
}
```

Impact

The inability to pause or deactivate the contract poses significant risks. If a bug is discovered after `setStart()`, the protocol cannot be frozen to prevent further damage, potentially leading to unchecked financial losses, exploitation, or governance failures.

Recommendation

The contract should implement a way to toggle the started state between `true` and `false`. This can be achieved by adding a new privileged function like `set_started(bool)` that allows the admin to change the contract's operational state. Additionally, the contract can be made `Pausable`.

Team Response

Fixed.

[I-01] Using Outdated Compiler Version

Severity

Info Risk

Description

All contracts across the codebase use the following pragma statement:

```
pragma solidity ^0.8.0;
```

Contracts should be deployed with the same compiler version and flags used during development and testing. An outdated pragma version might introduce bugs that affect the contract system negatively, or recent compiler versions may have unknown security vulnerabilities.

Impact

Using a floating and outdated compiler version increases the risk of unintentional behavior due to unpatched bugs or undocumented changes, and may reduce compatibility with modern tooling or security analysis.

Recommendation

It is recommended to lock the pragma to the latest and specific version of the compiler.

Team Response

Fixed.

shieldify

Thank you!