UnHash

2022

Borrow Protocol Smart Contracts

security audit

Table of

Contents

01 | Introduction 03

02 | Security Assessment Principles 04

03 | Executive Summary 05-06

04 | Detected Issues 07-11

05 | About About UnHash 12

06 | Disclaimer 13

Part 1 Introduction

General Provisions

Borrow Protocol connects lenders and borrowers in a completely trustless, decentralized way. Various collaterals may be used, including NFT tokens. Major security improvements over competing protocols include the absence of administrative loan parameters and oracles.

UnHash was approached by the team to provide a security audit of the protocol smart contracts.

Scope of the audit

The scope of the audit is the following smart contracts:

https://github.com/unitprotocol/borrow-module/tree/6882304805c21e746a8ccca078bf6592869fbebb/contracts
, git commit: 6882304805c21e746a8ccca078bf6592869fbebb.

Part 2 Security Assessment Principles

Classification of Issues

CRITICAL:

Bugs leading to Ether or token theft, fund access locking or any other loss of Ether/tokens to be transferred to any party (for example, dividends).

MAJOR:

Bugs that can trigger a contract failure. Further recovery is possible only by manual modification of the contract state or replacement.

WARNINGS:

Bugs that can break the intended contract logic or expose it to DoS attacks.

COMMENTS:

Other issues and recommendations reported to/ acknowledged by the team.

Security Assessment Methodology

Security assessment is a multi-stage process. Each stage is further divided into multiple sub-stages and check routines. Eventually, the code under inspection is checked numerous times.

The unique UnHash methodology is developed through the course of many years of security analysis, and it can be briefly described as follows.

Unbiased code inspection

Reverse-engineering of models behind the software. That is where most logical inconsistencies pop up.

Guided code inspection

Documentation and code-guided inspection. They make sure the code works as intended, verifying with the models from the previous step. Checking invariants.

Automated analysis

Use of the best tools in the field, as well as a proprietary analyzer.

Checklist analysis

Use of a checklist, built through the course of several years of in-field experience.

Project- and sphere-specific analysis

Part 3 Executive Summary

UnHash was approached by the team to provide a security audit of the protocol smart contracts.

The audited code is a simple borrowing solution that connects a lender and a borrower directly. No pooled assets are used. A loan interest rate is auctioned. Any collateral and loaned asset can be used as long as both parties agree. There is only one case when liquidation occurs: if a loan is not repaid in a timely fashion.
The parameters of a loan are immutable. As a result, there is no need for administrative actions on existing loans. That results in a trustless solution. Some frequent attack vectors, e.g. a rug-pull, are eliminated completely.
Another important characteristic is the absence of oracles in the solution. As a result, any asset can be used and oracle attack vectors are also eliminated.

Key phases of the audit were conducted from 12 to 28 July 2022. The codebase is relatively simple, and that fact contributes to the security of the protocol. A multi-pass security evaluation was performed according to our standards.
The codebase itself is implemented on a high level. No major or critical issues were identified. Only a few warnings were issued, mostly about improving the security of some malicious collateral handling.

Issues summary

Issue type	Number
Critical	0
Major	0
Warning	6
Comment	10

Part 3 Executive Summary

The key security factors of the system are simplicity, prudent code quality, and reentrancy guards.

According to our analysis, the commits 5bc5afd07396f61c4c8273f0cb8ef6ee96037ba6 and 53105d72a0509e8209a37095fcedf7cbd15a0c15 do not have any vulnerabilities.

Part 4 Detected Issues

CRITICAL

Not found.

MAJOR

Not found.

WARNING

01 Assets01.sol#L15

Such a detector won't work for some NFTs, especially some old ones, like Cryptopunks and CryptoKitties.

Can be checked on a forked node by the following code:

ERC721Tester.sol

```
// SPDX-License-Identifier: bsl-1.1
pragma solidity ^0.8.0;

import "@openzeppelin/contracts/token/ERC721/IERC721.sol";
import
"@openzeppelin/contracts/utils/introspection/ERC165Checker.sol";


contract ERC721Tester {
    using ERC165Checker for address;

    function is721(address addr) external view returns (bool) {
       return addr.supportsInterface(type(IERC721).interfaceId);
    }
}
```

Test721Check.js

```
const { expect } = require("chai");
const { ethers } = require("hardhat");

const {deployContract} = require("../scripts/utils");
```

```
describe("Test721Check", function () {
    beforeEach(async function () {
        this.t = await deployContract("ERC721Tester");
    });

    it("721Check", async function () {
     for (const addr of
['0xb47e3cd837dDF8e4c57F05d70Ab865de6e193BBB',
                '0xBC4CA0EdA7647A8aB7C2061c2E118A18a936f13D',
                '0xa7d8d9ef8D8Ce8992Df33D8b8CF4Aebabd5bD270',
                '0x06012c8cf97BEaD5deAe237070F9587f8E7A266d',
                '0xF87E31492Faf9A91B02Ee0dEAAd50d51d56D5d4d']) {
            console.log(addr, await this.t.is721(addr));
        }
    });
});
```

Acknowledged

Client: This is true, thank you. We can't interact with kitties and punks as NFT since they don't have some/all required methods. But also it is wrong to interpret them as ERC20 since they are not ERC721.
In a fix a new param is introduced - asset type. Dapp will check the contract with assetViewer is it ERC721/ERC20. But the user should check if we guessed right.

Part 4 Detected Issues

02 BorrowModule01.sol#L39

Malicious collateral may change its type (from 721 to 20 and vice versa). Remembering the type from `getFrom` may be an extra safety precaution.

Fixed as of 5bc5afd

03 BorrowModule01.sol#L87

The check prevents usage of ERC721 with `tokenId == 0`.

Fixed as of 5bc5afd

04 AssetViewer.sol#L35

The gas stipend may not be enough for tokens that do some dynamic accrual estimation. The ERC 20 doesn't limit gas usage. Some significantly bigger number is needed to still prevent reverts.

Fixed as of 5bc5afd

Part 4 Detected Issues

05 BorrowModule01.sol#L155

Note that there may be a race condition regarding `loan.interestRate` when `parameters.getAuctionDuration()` is changed in parallel with the mining of `accept`. The resulting rate would be unfavorable for some of the parties.

Acknowledged

06 BorrowModule01.sol#L180

Note that tokens transferring less than requested (b/c of fees, etc) break the logic. Also, any accrued interest/coupons are lost.
Note that any non-standard function invocations requested by `collateral` or `debtCurrency` (e.g. token migrations, claims, etc, but especially token migrations) are not possible and may permanently lock some of the collaterals.

Acknowledged

Client: Yes, we understand this. Maybe we should add a note in the dapp to avoid the usage of such nonstandard tokens.

COMMENT

01 ParametersStorage.sol#L25

Why is max multiplied by 10? Max is max.

Fixed as of 5bc5afd

02 ParametersStorage.sol#L32

Events about the added manager and changed `treasury` should be emitted.

Fixed as of 5bc5afd

03 BorrowModule01.sol#L72

It makes sense to index `loanId` and the address.

Fixed as of 5bc5afd

Part 4 Detected Issues

04 BorrowModule01.sol#L193

Misprint in `_loadId`.

Fixed as of 5bc5afd

05 BorrowModule01.sol#L85

Here and below: it's worth having a common prefix for all error messages to distinguish them in a complex transaction.

Fixed as of 5bc5afd

06 BorrowModule01.sol#L127

There is no way to cancel an outdated auction. `activeAuctions` will bloat.

Acknowledged

Client: Yes, it is expected. We have methods for selecting auctions with offset/limit to handle this.

07 BorrowModule01.sol#L327

`* 1 days / 365 days` can be replaced with `/ 365`.

Fixed as of 5bc5afd

08 BorrowModule01.sol#L174

A borrower also pays interest on the platform fee to the lender (the fee is not deducted from `loan.debtAmount` during interest calculation) - is that intended?

Acknowledged

09 Assets01.sol#L10

We don't see a point in a `01` suffix in a library name since this library is compiled into a contract bytecode.

Client: We plan to support several versions of the main contract at once. Since it depends on `Assets` and `Assets` may change in the future with changes in the main contact (for example to support a new type of asset) I think that it makes sense to add versions to `Assets` too.

Couldn't it be handled with git tags/branches? File-specific versions sound like the CVS era.

Fixed as of 5bc5afd

10 ParametersStorage.sol#L48

Strictly speaking, `@notice` is not a notice, but "Explains to an end user what this does", according to NatSpec.

Fixed as of 5bc5afd

Part 4 Detected Issues

Part 5 About UnHash

About UnHash

UnHash is a security auditing and consulting company valuing software security as the top priority. It provides security assessment and consulting for the most innovative DLT software projects.

Contact us:

https://unhash.io

hello@unhash.io

Part 6 Disclaimer

UnHash does not guarantee any commercial, investment or other benefits from the use or launch of products based on the audited code. UnHash may be involved in the promotion of the code and/or products based on it only with its written consent.