SECURITY REVIEW OF GOLDILOCKS

hyacinth

SECURITY REVIEW GOLDILOCKS

Summary

Auditors: 0xWeiss

Marketplace: Hyancinth

Client: Goldilocks

Report Delivered: October 2024

Protocol Summary

Protocol Name	Goldilocks
Language	Solidity
Codebase	https://github.com/0xgeeb/goldilocks-core
Commit	a912b1b0efcff1bb43704a0b13ae3bca0781290e
Previous Audits	Yes, 2

About 0xWeiss

Marc Weiss, or **0xWeiss**, is an independent smart contract security researcher. Having found numerous security vulnerabilities in various protocols, he does his best to contribute to the blockchain ecosystem and its protocols by putting time and effort into security research & reviews. Reach out on Twitter @0xWeisss or on Telegram @0xWeiss.

Audit Summary

Goldilocks engaged 0xWeiss through Hyacinth Audits to review the security of its token contract. 0xWeiss reviewed the source code in scope. At the end, there were 7 issues identified. All findings have been recorded in the following report. Notice that the examined smart contracts are not resistant to internal exploitation. For a detailed understanding of risk severity, source code vulnerability, and potential attack vectors, refer to the complete audit report below.

Vulnerability Summary

Severity	Total	Pending	Acknowledged	Par. resolved	Resolved
HIGH	1	0	1	0	0
MEDIUM	3	0	2	0	1
LOW	3	0	1	0	2
INF	0	0	0	0	0

Audit Scope

ID	File Path
GOLD	src/core/**.sol

Severity Classification

Severity	Classification
HIGH	Exploitable, causing loss/manipulation of assets or data.
MEDIUM	Risk of future exploits that may or may not impact the smart contract execution.
LOW	Minor code errors that may or may not impact the smart contract execution.
INF	No impact issues. Code improvement

Methodology

The auditing process pays special attention to the following considerations:

● Testing the smart contracts against both common and uncommon attack vectors.

● Assessing the codebase to ensure compliance with current best practices and industry standards.

● Ensuring contract logic meets the specifications and intentions of the client.

● Cross-referencing contract structure and implementation against similar smart contracts produced by industry leaders.

● Thorough line-by-line manual review of the entire codebase by industry experts.

Findings and Resolutions

ID	Category	Severity	Status
GOLD-1	Logical error	**HIGH**	Acknowledged
GOLD-2	Logical error	**MEDIUM**	Acknowledged
GOLD-3	Logical error	**MEDIUM**	Resolved
GOLD-4	Architectural error	**MEDIUM**	Acknowledged
GOLD-5	Logical error	**LOW**	Resolved
GOLD-6	Logical error	**LOW**	Resolved
GOLD-7	Logical error	**LOW**	Acknowledged

GOLD-1 | Loan data is never deleted when fully repaid or liquidated which will DOS new borrow orders for users

Severity	Category	Status
HIGH	Logical error	Acknowledged

Description of the issue

Users have a maximum amount of loans they can open so that when looping through all of them via `_lookupLoan` there can't be a DOS for having too many loans open.

```
uint256 userLoansLength = loans[msg.sender].length;
if(userLoansLength == MAX_LOANS) revert TooManyLoans();
```

Every time a user borrows, the number of loans increments by 1:

```
>> loans[msg.sender].push(loan);
IERC721(collateralNFT).transferFrom(msg.sender, address(this), collateralNFTId);
```

The problem is that this loan is never removed from the array when it has been fully repaid or liquidated, DOSing the user forever from opening new loans. User boosts will be locked without them being able to borrow making the boosts un-usable

Recommendation

Pop or delete the Loan struct for every index when it is fully repaid or liquidated.

Resolution

Acknowledged, we are okay with a 25 loan limit per address.

GOLD-2 | Foot-gun architecture when re-boosting

Severity	Category	Status
MEDIUM	Logical error	Acknowledged

Description of the issue

The following issue arises from the fact that you can boost multiple times different NFTs.

```
  function boost(
  address[] calldata partnerNFTs,
  uint256[] calldata partnerNFTIds
) external {
  uint256 partnerNFTsLength = partnerNFTs.length;
  for(uint256 i; i < partnerNFTsLength;) {
   if(partnerNFTBoosts[partnerNFTs[i]] == 0) revert InvalidBoostNFT();
   unchecked {
    ++i;  }}
  if(partnerNFTsLength != partnerNFTIds.length) revert ArrayMismatch();
  boosts[msg.sender] = _buildBoost(partnerNFTs, partnerNFTIds);
```

These boosts are always packed together and have an expiry date which is ` expiry: block.timestamp + boostLockDuration,`.

There no matter how much NFTs you boost, you will have them on the same boost with the same expiry. This means that the architecture is prone to self-DOS by boosting or re-boosting NFTs that are already expired but not withdrawn, as this would re-boost them and increase the expiry once again, not allowing to withdraw the NFT until it is expired.

Recommendation

Either do not automatically re-boost expired boosts when a user tries to boost again. Because if they were to boost again they can always specify such NFT again, or make it very clear in NATSPEC and disclose to the community such that if they re-boost and the previous boost is not withdrawn they will re-lock the old boost too

Resolution

Acknowledged

GOLD-3 |NFT boosts changes are not realized in current boosts

Severity	Category	Status
MEDIUM	Logical error	Resolved

Description of the issue

When users boost they do it by using the current value of the NFT: `partnerNFTBoosts` which it is added to the boost magnitude `magnitude`:

```
for(uint256 i; i < nftsLength;) {
    magnitude += partnerNFTBoosts[nfts[i]];
    unchecked {
      ++i;
    }
```

This boost is then locked for `boostLockDuration` at that specific magnitude.

In the scenario that the value of the NFT is changed `partnerNFTBoosts[nfts[i]]`, then any locks of such NFT would be affected because they would retain the old values for the rest of the boost.

Recommendation

Given that the fix would require a quite complex architectural change, do not change the value of an NFT while boosts for that certain NFT are locked.

Resolution

While the functionality to do so is there, to not trigger this issue, the developer team has specified that no values will be changed while boosts are active to not cause this problem.

GOLD-4 | All NFTs from the same collection are valued at the same fair value

Severity	Category	Status
MEDIUM	Architectural error	Acknowledged

Description of the issue

Currently there is a `nftFairValues` mapping that stores what would be the fair value for an NFT collection. The problem is that all the NFTs from such collections are valued at the same fair value while rarities inside the same collection are completely different. So a 1 of 1 NFT which might be valued at 50k USD, would have the same "fair value" as an NFT from the floor price of the collection priced at 1k USD.

All NFTs from the same collection are valued at the same fair value which will miss-price rare NFTs and affect the borrowable amount of a user.

Recommendation

Do not only assign a fair value per collection, but also have the option to assign a fair value per Id.

Resolution

Goldilend intentionally treats all NFT's from within a single collection equivalently, and ignores e.g. rarity differences. The fair value for a collection represents the maximum that Goldilend is willing to lend out for any specific NFT within the collection.

GOLD-(5-7) | List of Low issues

Severity	Category
LOW	Compilation

GOLD 5 - Total valuation to nft fair value ratio can be broken

Description of the issue

The function `  function initializeBeras()` initializes nft fair values and enables borrowing.

```
function initializeBeras(
    uint256 _totalValuation,
    address[] calldata _nfts,
    uint256[] calldata _nftFairValues
  ) external {
    if(msg.sender != multisig) revert NotMultisig();
    if(berasInitialized) revert AlreadyInitialized();
    berasInitialized = true;
    totalValuation = _totalValuation;
    uint256 nftFairValuesLength = _nftFairValues.length;
    for(uint256 i; i < nftFairValuesLength;) {
      nftFairValues[_nfts[i]] = _nftFairValues[i];
      unchecked {
        ++i;}}
    borrowingActive = true; }
```

`nftFairValues` should represent 1/1000's of the `totalValuation`, so they should sum to 1000. There is no hard requirement for this in the code, which would allow this "invariant" to be broken.

Recommendation

Specifically check that this scenario does not happen when calling `initializeBeras()` and the ratio is kept

Resolution

Fixed at PR

GOLD 6 - Incorrect state tracked for initialization variables

Description of the issue

The `initializeParameters` function does not emit any event when initializing variables that already have events declared, tracking the state incorrectly:

```
function initializeParameters(
  uint256 _multisigShare,
  uint256 _apdaoShare,
  uint256 _minDuration,
  uint256 _maxDuration,
  uint256 _protocolInterestRate,
  uint256 _slope,
  uint256 _annualPrgEmissions,
  uint256 _boostLockDuration) external {
  if(msg.sender != multisig) revert NotMultisig();
  if(parametersInitialized) revert AlreadyInitialized();
  parametersInitialized = true;
  multisigShare = _multisigShare;
  apdaoShare = _apdaoShare;
  minDuration = _minDuration;
  maxDuration = _maxDuration;
  protocolInterestRate = _protocolInterestRate;
  slope = _slope;
  annualPrgEmissions = _annualPrgEmissions;
  boostLockDuration = _boostLockDuration; }
```

Some examples are:

`NewProtocolInterestRate`, `NewShareRates`, `NewSlope`, `NewDurations`

Recommendation

Emit the events also in the `initializeParameters` function

Resolution

Fixed at PR

GOLD 7 - Donations can backfire and cause minted lock amount dilutions

Description of the issue

Locking iBGT is very similar to the `deposit()` function in vaults and lending protocols. Here, the first depositor can also be diluted to mint 0 tokens by depositing less than the donation amount.

While Goldilocks does the right thing by not checking `balanceOf()` to check `poolSize`, they included a `donate()` function in a prior audit that allows to increase `poolSize` without increasing the total supply.

```
function lock(uint256 amount) external {
  uint256 mintAmount = _GiBGTMintAmount(amount);
  poolSize += amount;
  SafeTransferLib.safeTransferFrom(ibgt, msg.sender, address(this), amount);
  _refreshiBGT(amount);
  _mint(msg.sender, mintAmount);
  emit iBGTLock(msg.sender, amount);
}
```

This allows for a rounding issue to happen:

- lock 1 wei, totalsupply = 1 wei, _poolSize = 1 wei
- donate 1000e6, totalsupply = 1 wei, _poolSize = 1000e6 + 1 wei
- lock < 1000e6 will mint 0 tokens due to rounding

Donations are permissioned, which makes this issue very low likelihood.

Recommendation

Be very careful when donations happen and try to always use a private rpc to execute such a function to not allow front/back-running.

Resolution

Acknowledged, do not currently use a private RPC but will for this donate function.

DISCLAIMER

This report is not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any “product” or “asset” created by any team or project that contracts 0xWeiss to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology. Blockchain technology and cryptographic assets present a high level of ongoing risk.

My position is that each company and individual are responsible for their own due diligence and continuous security. My goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze. Therefore, I do not guarantee the explicit security of the audited smart contract, regardless of the verdict.