BAIL security

BAILSEC.IO

EMAIL : OFFICE@BAILSEC.IO

TWITTER : @BAILSECURITY

TELEGRAM : @HELLOATBAILSEC

FINAL REPORT:

OverHere

October 2023

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

Project	OverHere
Website	TBA
Type	AMM
Language	Solidity
Methods	Manual Analysis
Github repository	https://github.com/Whatzhub/merch-project/tree/ad0034d735cb1cd0d94044cf874e65d540c79ffb/contracts/merch-core/master

2. Detections Overview

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
High	1	1		
Medium	1			1
Low	5			5
Informational	9	1		8
Governance	1			1
Quality assurance	2			2
Total	19	2		17

2.1 Detection Definition

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
Qauality assurance	Aggregated minor issues, ensuring a high quality codebase.

3. Detections

Asset

The Asset.sol contract serves as a specialized ERC20 token with distinct functionalities and roles within a particular decentralized finance (DeFi) mechanism.

Unlike traditional tokens, it operates with a max supply that isn't fixed but is instead adjustable by the contract's owner, thereby providing a layer of dynamic supply management that can be tailored to the economic needs of the system.

Furthermore, it introduces a concept of "virtual supply," which is manipulated by a separate, privileged "Master" contract during buy and sell operations to influence the bonding curve without altering actual token balances during a redeem, effectively adjusting perceived supply and consequently impacting prices.

Moreover this contract is acting as a custodian for the base asset, which could be another token or ETH, the Asset contract engages with this base asset during certain functions, namely during buy and sell operations, ensuring a mechanism to provide liquidity or back the value of the issued tokens. Therefore, in simple terms, the base asset is just the asset which is used for the purchase and sale.

Unique to its design, the token restricts transferability, limiting transfers to the contract itself and not to arbitrary addresses, a feature that serves to regulate the token's flow and confines its operations within defined boundaries, possibly mitigating speculative trading or unauthorized transfers.

Issue	Governance issues
Severity	Governance
Description	The contract owner can freely change the Master contract, which can result in a full loss of funds. Users should be aware of this privilege.
Recommendations	Since we assume that this flexibility is desired, and consider the OverHere team as highly trusted, we highly recommend ensuring a proper safety architecture around the owner address, for example, using a multisig contract as owner, to mitigate the scenario of a compromised owner.
Comments / Resolution	Acknowledged

Issue	Missing minimum check for setMaxSupply
Severity	Low
Description	The setMaxSupply function allows the owner to set an arbitrary maxSupply. However, it does not check if the new maxSupply is larger than the current virtualSupply, which can result in an unfair outcome if tokens are being burned and then minting is prohibited.
Recommendations	A check should be implemented which ensures that the maxSupply is in fact larger than the current virtualSupply.
Comments / Resolution	Acknowledged

Issue	Quality assurance
Severity	**Quality assurance**
Description	All typographical and minor issues have been aggregated in the Quality Assurance section for clarity and comprehensive review. L 3: import '@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol'; This import is unused, it can therefore be removed.
Recommendations	We recommend fixing all aforementioned issues.
Comments / Resolution	Acknowledged

Master

Executive Summary:

The audited Master Contract governs the mechanism of buying, selling, and redeeming of "Merch" assets, underpinned by a bonding curve pricing model and additional operational logic to ensure coherent functionality, security, and adherence to the projected use case scenarios. Below, we outline the contract's primary mechanics and associated considerations.

1.Purchase and Sale Mechanics:

a) Purchase of Merch Assets: Users can purchase one or more Merch assets, payable with a specified base asset, indicated to be WETH. The price is dictated by a bonding curve, initializing at 0.07407 WETH and potentially escalating to 201 WETH for the 500th token, as example.
b) Sale of Merch Assets: Users are also empowered to sell Merch assets, with the sale price dynamically determined by the aforementioned bonding curve.

2.Pricing and Supply Sensitivity:

The contract encapsulates a sharp price appreciation relative to supply, operated via the employed bonding curve, which tends to exponentially increase the asset price as the virtual supply expands, ostensibly to manage scarcity and value.

3.Redemption Mechanics:

Users retain the ability to redeem Merch tokens for physical items (e.g., an FTX rug), with the redemptive logic executed off-chain, presumably triggered by an on-chain event emission during the redeem function, to ensure secure and verifiable redemption processes

4.Free Structuring:

A "haircut" fee is applied to both purchase and sale transactions, requiring users to pay additional assets above the quoted price during purchases and deducting from the return quote during sales, ensuring a built-in fee mechanism.

5.Whitelisting and Purchase Limitations:

a) Whitelist Enforcement: The contract employs a whitelist mechanism, dictating that only whitelisted addresses can execute purchases during a defined period, with each address limited to the acquisition of 1 asset (1e18).

b) Sell and Redemption Limitations: Selling and redeeming actions are restricted until the conclusion of the whitelist period to protect against premature transaction executions.

6.Governance

The contract owner retains the capability to manage whitelisted addresses via an external whitelist contract and to modify the whitelist period and corresponding contract to manage user access and purchase capabilities effectively. Moreover, the contract owner may add multiple MERCH assets, with their underlying base asset.

It is important to note that the libraries are out of scope, therefore the assumption was made that the arithmetical operations work as expected. We encourage the OverHere team to fuzz all libraries and execute edge-case testing, libraries handling such arithmetic calculations are often subject to reverts due to overflows when specific parameters become severely large.

Issue	Whitelist mechanism can be exploited
Severity	**High**
Description	The whitelist mechanism intends to only distribute one asset (1e18) per whitelisted user during the whitelist period. This is ensured via a balance check: `if (block.timestamp < whitelistEndTime) {` `require(whitelist.check(address(msg.sender)) &&` `IERC20(token).balanceOf(address(msg.sender)) == 0, 'not eligible to buy');` `amount = 1 * WAD;` `}` However, the buy function exposes a 'to' parameter, which can be abused by a sophisticated attacker to purchase thousands of assets to another address, effectively bypassing the whitelist mechanism. This would effectively break the whole business logic as the price will become ultra-large and other investors might be unable to participate.
Recommendations	Remove the to parameter to prevent this attack
Comments / Resolution	Resolved

Issue	BaseAsset cannot be a transfer-tax token
Severity	Medium
Description	In the scenario that the baseAsset is a transfer-tax token, this will break the contract in multiple scenarios since the to received amount will be less than what was accounted for.
Recommendations	Consider either not using such tokens or implementing logic which supports these tokens.
Comments / Resolution	Acknowledged, the OverHere team indicated that such tokens will not be used.

Issue	Quote calculation will revert under certain circumstances
Severity	Informational
Description	Currently, the bonding curve calculation is using the following mathematical operations: ``` int256 lhs_1 = DSMath.toInt256(magicNum1).wmul(LogExpMath.exp(DSMath.toInt256(magicNum2.wmul(virtualSupply + amount)))); int256 lhs_2 = DSMath.toInt256(magicNum3.wmul(virtualSupply + amount)); int256 lhs = lhs_1 - lhs_2; int256 rhs_1 = DSMath.toInt256(magicNum1).wmul(LogExpMath.exp(DSMath.toInt256(magicNum2.wmul(virtualSupply)))); int256 rhs_2 = DSMath.toInt256(magicNum3.wmul(virtualSupply)); int256 rhs = rhs_1 - rhs_2; quote = SignedSafeMath.toUint256(lhs - rhs); ``` To understand the precise overflow location, we need to dismantle the highlighted expression: 1) Add virtualSupply + amount

2) Multiply (virtualSupply+amount) with magicNum2
-> decrease the value since magicNum2 is represented as 0.0137

3) Use this value as exponent for e ≈ 2.71828

4) Multiply the result in 3) with magicNum1

return ((x * y) + (WAD / 2)) / WAD;

The highlighted spot will likely overflow, depending on the final constants which are set.

Currently, these constants are as follows:

```
    uint256 internal constant magicNum1 = 1569340000000000000;
    uint256 internal constant magicNum2 = 13700000000000000;
    uint256 internal constant magicNum3 = 14600000000000000;
```

*The OverHere development team indicated that these values can be subject to a change.

For these values, the calculation will already overflow if the supply exceeds 6650 assets. While it is unlikely that the business logic of this contract even intends to distribute so many assets, since the price for 1 asset at index 6648 would be 78680092980863437021403158933030325427516 nominal tokens, that is not completely impossible, if a token with a very low price is used as base asset.

It is important to highlight that even if the magicNum2 variable is decreased, a revert could still happen within the exp function, if the exponent becomes larger than 130e18.

However, this issue is only rated as a low vulnerability due to this assumption.

Some examples of simple fuzz tests:

```
Logs:
  Bound Result 3176
  Bound Result 145

Traces:
  [756468] FuzzAlgo::setUp()
    ├─ [701334] → new Master@0x5615dEB798BB3E4dFa0139dFa1b3D433Cc23b72f
    │   └─ ← 3503 bytes of code
    └─ ← ()

  [28605] FuzzAlgo::testFuzzBuy(3176, 3067)
    ├─ [0] console::log(Bound Result, 3176) [staticcall]
    │   └─ ← ()
    ├─ [0] console::log(Bound Result, 145) [staticcall]
    │   └─ ← ()
    ├─ [21833] FuzzAlgo::fuzzBuy(3176, 145) [staticcall]
    │   ├─ [16688] Master::_buyQuoteFunc(3176, 145) [staticcall]
    │   │   └─ ← "value must be positive"
    │   └─ ← "value must be positive"
    └─ ← "value must be positive"

Test result: FAILED. 0 passed; 1 failed; 0 skipped; finished in 50.74ms

Ran 1 test suites: 0 tests passed, 1 failed, 0 skipped (1 total tests)
```

```
Logs:
  Bound Result 1115200387632
  Bound Result 214

Traces:
  [756468] FuzzAlgo::setUp()
    ├─ [701334] → new Master@0x5615dEB798BB3E4dFa0139dFa1b3D433Cc23b72f
    │   └─ ← 3503 bytes of code
    └─ ← ()

  [27915] FuzzAlgo::testFuzzBuy(1115200387632 [1.115e12], 214)
    ├─ [0] console::log(Bound Result, 1115200387632 [1.115e12]) [staticcall]
    │   └─ ← ()
    ├─ [0] console::log(Bound Result, 214) [staticcall]
    │   └─ ← ()
    ├─ [21833] FuzzAlgo::fuzzBuy(1115200387632 [1.115e12], 214) [staticcall]
    │   ├─ [16688] Master::_buyQuoteFunc(1115200387632 [1.115e12], 214) [staticcall]
    │   │   └─ ← "value must be positive"
    │   └─ ← "value must be positive"
    └─ ← "value must be positive"

Test result: FAILED. 0 passed; 1 failed; 0 skipped; finished in 6.32s

Ran 1 test suites: 0 tests passed, 1 failed, 0 skipped (1 total tests)
```

```
Logs:
  Bound Result 158399
  Bound Result 4

Traces:
  [756468] FuzzAlgo::setUp()
    ├─ [701334] → new Master@0x5615dEB798BB3E4dFa0139dFa1b3D433Cc23b72f
    │   └─ ← 3503 bytes of code
    └─ ← ()

  [29442] FuzzAlgo::testFuzzSell(1599999999999999999999 [1.599e21], 4)
    ├─ [0] console::log(Bound Result, 158399 [1.583e5]) [staticcall]
    │   └─ ← ()
    ├─ [0] console::log(Bound Result, 4) [staticcall]
    │   └─ ← ()
    ├─ [18159] Master::_sellQuoteFunc(10000000000000000000 [1e20], 4) [staticcall]
    │   └─ ← "value must be positive"
    └─ ← "value must be positive"
```

While our fuzzing tests only showed reverts for scenarios which are unusual for the normal business logic, we highly recommend the OverHere team to execute extensive fuzzing to further identify potential edge-cases. Moreover, to be aligned with the fuzzing results, buys/sells below 1e3 should be prevented with an explicit revert error.

Recommendations	The nature of that limitation lies within the bonding-curve algorithm itself. A fix of this issue would most probably require creating a completely new algorithm. Therefore, we simply recommend a strict limitation of the maximum supply to a reasonable value. However, since the screenshot not only shows an overflow and multiple fuzz tests have reverted for values <1e3, it might be desirable to not allow purchases/sells below a certain threshold e.g. 1e3, in an effort to prevent potential reverts due to that.
Comments / Resolution	Acknowledged, the client indicated the following: “As the MERCH token is tied to the redemption of physical products, the maximum supply logic is tied to the amount of products a merchant may provide. The current intent is to map the supply of MERCH tokens to limited edition products, but may change due to business reasons. “ Therefore, such a high max supply will never be reached.

Issue	Haircut rate can be set up to 100%
Severity	Low
Description	Currently, the haircut rate can be set up to 100%, which would essentially mean that a user will pay 100% on top of the buy amount or will receive no payout for a sale. Such a large limit should not be part of the normal business logic, as it could effectively result in a loss of user funds
Recommendations	It should be ensured that the haircut rate can only be set to a

	reasonable value, as example 30%.
Comments / Resolution	Acknowledged, the client indicated that the contract owner will be a multisig contract and that this flexibility is desired.

Issue	Quotation will round down for assets with less than 18 decimals
Severity	Low
Description	Whenever the price is fetched from the bonding-curve, the return value is with 18 decimals precision. As an example we can take the first two index prices (1e18 denomination): 70479075766447587 73465247675957049 If the corresponding base asset is a 6 decimal token, this will truncate these values to the following amounts: 70479 73465 Since the correctness of the bonding-curve is based on the increased value, this will result in a broken invariant, because an attacker can therefore use the truncation to their own benefit, potentially receiving the same quote for a higher index. As an example: 1e18 assets will cost 70479 when the current index is 1e18 but so will it also if the current index is 1.0002e18 This is only a small deviation, but can result under certain circumstances to negative side-effects such as stuck funds or reverts due to insufficient balances or could even be exploited by a malicious

	party.
Recommendations	The issue lies within the nature of the truncation. A possible solution is to remove the truncation and work with the real return values. This would then however mean a different curve is necessary to properly reflect the desired decimals. However, since we assume that only 1e18 decimal tokens will be used for this codebase, this issue is only rated as low severity.
Comments / Resolution	Acknowledged, the client indicated that only tokens with 18 decimals are being used.

Issue	quotePotentialSwap might unexpectedly return zero as quote and haircut
Severity	Low
Description	The quotePotentialSwap function is based on two scenarios: a) !isFromTokenAsset && isToTokenAsset b) isFromTokenAsset && !isToTokenAsset In a very rare edge-case that both tokens are added as assets, this will return 0 as haircut and quote, which would essentially allow users to purchase assets for free. *This issue was only rated as a low severity instead of high since we assess such a scenario as highly unlikely, though theoretically it could still happen.
Recommendations	We therefore recommend simply reverting if both variables, isFromTokenAsset and isToTokenAsset returns true.
Comments / Resolution	Acknowledged, the client indicated that only carefully selected tokens will be used and therefore this is not seen as an issue.

Issue	Lack of validation
Severity	Low
Description	During the initialize function, there is no validation for the haircutRate and whitelistEndTime parameters. Moreover, the whitelistEndTime should have a reasonable upper limit within the setWhitelistEndTime function, to ensure safety for users as a large limit could prevent selling and redeeming.
Recommendations	These parameters should be validated accordingly.
Comments / Resolution	Acknowledged, the client indicated that the if there are any mistakes during deployment, the contract owner will amend these.

Issue	Reentrancy attack allows for repeatedly draining the fee
Severity	Informational
Description	Within the _mintFee function, the transfer out is happening before the fee is being reset. In the scenario of an ERC777 token, this could lead to a reentrancy attack, to reenter before the reset, allowing to repeatedly take out the accumulated fee. This issue is only rated as informational since we do not expect such tokens to be used as base assets, moreover, the governance has full control over the funds anyways.
Recommendations	Ideally, the fee amount is being cached, the fee is reset and the transfer is happening afterwards.
Comments / Resolution	Resolved.

Issue	Checks-effects interactions pattern is violated
Severity	**Informational**
Description	As already mentioned in the _mintFee function, a similar issue is present within the buy function, as the fee is incremented after external calls, which violates the cei pattern: https://medium.com/returnvalues/smart-contract-security-patterns-79e03b5a1659 However, this scenario cannot be exploited but still violates best practices for smart contract programming.
Recommendations	Any state variable change should be executed before external calls.
Comments / Resolution	Acknowledged.

Issue	Frontend phishing possibility
Severity	**Informational**
Description	Multiple functions allow a 'to' parameter as beneficiary. In the event of a compromised frontend, this can result in a loss of user funds.
Recommendations	Maintaining a strong emphasis on the security of the web2 component is crucial, especially given the increasing prevalence of frontend attacks in today's landscape.
Comments / Resolution	Acknowledged

Issue	Inconsistency in _mintFee call
Severity	Informational
Description	Within the sell function, the _mintFee function is called, however, this is not the scenario within the buy function. This exposes an inconsistency which should not be part of a well-designed smart contract.
Recommendations	We recommend removing this call and solely allow the contract owner to collect the fee. (Implementing a pull-over-push pattern)
Comments / Resolution	Acknowledged.

Issue	mintAllFee is unused
Severity	Informational
Description	The aforementioned function is internal and unused, moreover, it might run out of gas if too many merch tokens are listed.
Recommendations	An external function should be implemented to call this function.
Comments / Resolution	Acknowledged.

Issue	Implementation can be initialized
Severity	**Informational**
Description	Currently, the disableInitializer function is not called during the contract deployment, this will result in the implementation being potentially initialized directly. In that scenario this does not expose a problem, however, it is generally considered as best-practice to disable the implementation from being initialized directly.
Recommendations	We recommend calling the _disableInitializer function during contract deployment: https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/proxy/utils/Initializable.sol#L192
Comments / Resolution	Acknowledged.

Issue	Quality Assurance
Severity	**Quality assurance**
Description	All typographical and minor issues have been aggregated in the Quality Assurance section for clarity and comprehensive review. L 44: address public merchantFeeTo; This variable is uninitialized. This issue is also present for the whitelist variable. However, this scenario might be desired to prevent purchases at the beginning. L 47:

	/// @notice Dividend collected by each asset (unit: WAD) The unit can also be in other decimals than 1e18. —-- L 114: function initialize(uint256 haircutRate_, uint256 whitelistEndTime_) public virtual initializer The virtual keyword can be removed as this contract isn't meant to be inherited. —-- Implementation of a pause mechanism: Since governance has full control over the funds anyways, it might make sense to implement a pause mechanism as an additional security feature for unforeseen events. —--
Recommendations	These issues should be fixed and recommendations implemented.
Comments / Resolution	Acknowledged.

Whitelist:

The `Whitelist` contract introduces a structured approach towards access management within the blockchain context, integrating a combination of whitelisting status and time-sensitive activation. The core structure employed, `WhitelistInfo`, encapsulates two pivotal data points:

- `isWhitelisted`: A boolean status indicating the presence of an address on the whitelist
- `blockTimestamp`: A Unix timestamp determining the point from which the whitelisted status is considered active

This allows the contract owner to not only add and remove addresses to/from the whitelist, but also setting a timestamp which must have passed, to consider an address as whitelisted.

The simple check function is then used within the Master contract but can be extendedly used in other contracts as well.

Issue	blockTimestamp should be reasonable
Severity	Informational
Description	Currently, there is no check for the aforementioned parameter to be a reasonable value. For example setting this value too far in the future does not make any sense.
Recommendations	We recommend implementing a reasonable validation.
Comments / Resolution	Acknowledged

Issue	Simple change of timestamp is not supported
Severity	Informational
Description	Whenever an address is already whitelisted, it is not possible to just adjust the blockTimestamp. Therefore it needs to be completely removed and then added again.
Recommendations	We recommend implementing a scenario which allows for only adjusting the blockTimestamp for already whitelisted wallets.
Comments / Resolution	Acknowledged