Quantstamp

Executive Summary

This audit report was prepared by Quantstamp, the leader in blockchain security.

Type	NFT
Timeline	2023-10-02 through 2023-10-10
Language	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	README.md
Source Code	• deathtothecorporation/milady-os-contracts 🔗 #e523ce9 🔗 • deathtothecorporation/milady-os-contracts 🔗 #32f97cd 🔗
Auditors	• Danny Aksenov Senior Auditing Engineer • Zeeshan Meghji Auditing Engineer • Nikita Belenkov Auditing Engineer

Documentation quality	Medium	▬▬▬▬
Test quality	Low	▬
Total Findings	19	**Fixed: 10** **Acknowledged: 7** **Mitigated: 2**
High severity findings ⓘ	3	**Fixed: 2** **Mitigated: 1**
Medium severity findings ⓘ	1	**Acknowledged: 1**
Low severity findings ⓘ	6	**Fixed: 3** **Acknowledged: 3**
Undetermined severity findings ⓘ	0	
Informational findings ⓘ	9	**Fixed: 5** **Acknowledged: 3** **Mitigated: 1**

Summary of Findings

Overall, the code is well written and easy to follow, not to mention having a low degree of centralization, which we like to see. However, there are several concerns that we would like to raise with the Vaporware team. Out of all issues, the most concerning are the high level issues: VAP-1, VAP-2, VAP-3. We believe that the current implementation makes use of a reward disbursement system that doesn't work as intended and needs to be reworked. Additionally, the test coverage that is currently available highlights the need to cover more possible branches within the code to reduce the risk of encountering any unexpected behaviors. Finally, we would also like to highlight that `EIP-6551` is still in draft and has not fully been accepted yet, which may lead to integration issues down the line.

Update: The Vaporware team has successfully addressed the issues with the rewards distribution system and acknowledged that `EIP-6551` is still in draft. However, while the test coverage has increased (see the update on VAP-3 for more details), it is still lower than what we would like to see. We recommended improving test coverage prior to going live.

ID	DESCRIPTION	SEVERITY	STATUS
VAP-1	Broken Reward Accounting System	● High ⓘ	Fixed
VAP-2	Misconfigured Owner Leads to No Liquid Accessories	● High ⓘ	Fixed
VAP-3	Low Test Coverage	● High ⓘ	Mitigated
VAP-4	Reward Disbursement Can Be Front-Run	● Medium ⓘ	Acknowledged
VAP-5	Ownership Can Be Renounced	● Low ⓘ	Acknowledged
VAP-6	Privileged Roles and Ownership	● Low ⓘ	Acknowledged

ID	DESCRIPTION	SEVERITY	STATUS
VAP-7	Missing Input Validation	• Low ⓘ	Fixed
VAP-8	Unbounded Curve Parameter Can Lead to Overflow	• Low ⓘ	Acknowledged
VAP-9	Vulnerable Solidity Version	• Low ⓘ	Fixed
VAP-10	Reentrancy Guards Can Reduce Attack Surface	• Low ⓘ	Fixed
VAP-11	`revenueRecipient` Cannot Be Changed if It Gets Compromised	• Informational ⓘ	Fixed
VAP-12	Bonding Curve Can only Be Set Once	• Informational ⓘ	Acknowledged
VAP-13	Missing Event Emissions	• Informational ⓘ	Acknowledged
VAP-14	Events Emitted After External Calls	• Informational ⓘ	Fixed
VAP-15	Draft Eip Used	• Informational ⓘ	Acknowledged
VAP-16	Mixing Test Code with Production Code	• Informational ⓘ	Fixed
VAP-17	Unlocked Pragma	• Informational ⓘ	Fixed
VAP-18	`transfer()` Could Run Out of Gas and Lead to Dos	• Informational ⓘ	Fixed
VAP-19	`miladyAuthority` Should Be a Multi Sig	• Informational ⓘ	Mitigated

Assessment Breakdown

Quantstamp's objective was to evaluate the repository for security-related issues, code quality, and adherence to specification and best practices.

> ⓘ **Disclaimer**
>
> Only features that are contained within the repositories at the commit hashes specified on the front page of the report are within the scope of the audit and fix review. All features added in future revisions of the code are excluded from consideration in this report.

Possible issues we looked for included (but are not limited to):

- Transaction-ordering dependence
- Timestamp dependence
- Mishandled exceptions and call stack limits
- Unsafe external calls
- Integer overflow / underflow
- Number rounding errors
- Reentrancy and cross-function vulnerabilities
- Denial of service / logical oversights
- Access control
- Centralization of power
- Business logic contradicting the specification
- Code clones, functionality duplication
- Gas usage
- Arbitrary token minting

Methodology

1. Code review that includes the following
   1. Review of the specifications, sources, and instructions provided to Quantstamp to make sure we understand the size, scope, and functionality of the smart contract.
   2. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.

3. Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to Quantstamp describe.
2. Testing and automated analysis that includes the following:
    1. Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run those test cases.
    2. Symbolic execution, which is analyzing a program to determine what inputs cause each part of a program to execute.
3. Best practices review, which is a review of the smart contracts to improve efficiency, effectiveness, clarity, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.
4. Specific, itemized, and actionable recommendations to help you take steps to secure your smart contracts.

Scope

Files Included

- `src/Deployer.sol`
- `src/HardcodedGoerliDeployer.sol`
- `src/LiquidAccessories.sol`
- `src/MiladyAvatar.sol`
- `src/Rewards.sol`
- `src/SoulboundAccessories.sol`
- `src/TGA/TBARegistry.sol`
- `src/TGA/TokenGatedAccount.sol`
- `src/TGA/IERC6551Account.sol`
- `src/TGA/IERC6551Executable.sol`
- `src/TGA/IERC6551Registry.sol`
- `src/TGA/TokenGatedAccount.sol`

Findings

VAP-1 Broken Reward Accounting System

● **High** ⓘ   Fixed

> ✅ **Update**
> Marked as "Fixed" by the client. Addressed in: `b84563414bd0965100b155dee7720dc0788ceb08` .

File(s) affected: `Rewards.sol`

Description: The reward system is designed to distribute funds between users that are currently wearing the accessory for which the rewards are distributed. The rewards are generated, when the same accessory is minted for a different `Milady` NFT. But it has been observed that this reward system does not behave as expected and allocates funds in uneven proportions. See the exploit below:

Exploit Scenario:

1. User A mints an accessory and equips it
2. The contract is funded with 20 ETH for that accessory via `addRewardsForAccessory()`
3. Users B and C, equip this accessory
4. The contract is funded again with 20 ETH for that accessory via `addRewardsForAccessory()`
5. User B unequips the item and receives, `20/3 = 6` ETH
6. User C unequips the item and receives, `20/2 = 10` ETH
7. The contract is funded again with 10 ETH for that accessory via `addRewardsForAccessory()`
8. User A tries to unequip their item and should receive `50/1 = 50 ETH` , but the contract does not have enough funds and the transaction fails.

** The issue occurs in step 6, as User C is entitled to 6 ETH, but receives 10 ETH instead

Additionally, please find the following PoC demonstrating this issue:

```
pragma solidity ^0.8.13;

import "forge-std/Test.sol";
import "forge-std/console.sol";
import "openzeppelin/token/ERC1155/IERC1155Receiver.sol";
import "../src/Rewards.sol";
import "../src/Deployer.sol";
import "./MiladyOSTestBase.sol";
import "./Miladys.sol";

contract RewardsTest is MiladyOSTestBase {
    function test_reward_poc() external {
        // prepare milady with its soulbound accessories
```

```solidity
        MiladyAvatar.PlaintextAccessoryInfo[] memory milady0AccessoriesPlaintext = new
MiladyAvatar.PlaintextAccessoryInfo[](3);
        milady0AccessoriesPlaintext[0] = MiladyAvatar.PlaintextAccessoryInfo("hat", "red hat");

        uint[] memory milady0Accessories =
avatarContract.batchPlaintextAccessoryInfoToAccessoryIds(milady0AccessoriesPlaintext);
        vm.prank(MILADY_AUTHORITY_ADDRESS);

        // User A mints accessory
        soulboundAccessoriesContract.mintAndEquipSoulboundAccessories(0, milady0Accessories);

        // deposit rewards for hat accessory
        rewardsContract.addRewardsForAccessory{value:20}(milady0Accessories[0]);

        require(rewardsContract.getAmountClaimableForMiladyAndAccessories(0, milady0Accessories) == 20);

        //Uses B and C mint accessories

        uint[] memory milady1Accessories =
avatarContract.batchPlaintextAccessoryInfoToAccessoryIds(milady0AccessoriesPlaintext);
        vm.prank(MILADY_AUTHORITY_ADDRESS);
        soulboundAccessoriesContract.mintAndEquipSoulboundAccessories(1, milady1Accessories);

        uint[] memory milady2Accessories =
avatarContract.batchPlaintextAccessoryInfoToAccessoryIds(milady0AccessoriesPlaintext);
        vm.prank(MILADY_AUTHORITY_ADDRESS);
        soulboundAccessoriesContract.mintAndEquipSoulboundAccessories(2, milady2Accessories);


        // deposit rewards for hat accessory

        rewardsContract.addRewardsForAccessory{value:20}(milady0Accessories[0]);


        require(rewardsContract.getAmountClaimableForMiladyAndAccessories(1, milady0Accessories) == 6);


        // User B and C unequip their accessories

        uint balancePreClaim = address(this).balance;


        vm.prank(address(avatarContract));
        rewardsContract.deregisterMiladyForRewardsForAccessoryAndClaim(1, milady0Accessories[0],
payable(address(this)));
         vm.prank(address(avatarContract));
        rewardsContract.deregisterMiladyForRewardsForAccessoryAndClaim(2, milady0Accessories[0],
payable(address(this)));

        uint balancePostClaim = address(this).balance;
        require(balancePostClaim - balancePreClaim == 16);

        // deposit rewards for hat accessory

        rewardsContract.addRewardsForAccessory{value:10}(milady0Accessories[0]);


        // User A tries to unequip their item and fails

        vm.prank(address(avatarContract));
        vm.expectRevert();
        //reverts because there are not enough funds in the contract
        rewardsContract.deregisterMiladyForRewardsForAccessoryAndClaim(0, milady0Accessories[0],
payable(address(this)));
    }
}
```

Recommendation: Consider the scenario above and redesign the distribution system.

VAP-2 Misconfigured Owner Leads to No Liquid Accessories

• **High** ⓘ `Fixed`

> ✅ **Update**
>
> Marked as "Fixed" by the client. Addressed in: `0b9529b307ddf1e0dcfac05d00a4f95d107434a8` .

File(s) affected: `LiquidAccessories.sol` , `Deployer.sol`

Description: `LiquidAccessories` is an `Ownable` contract that transfers ownership of the contract to `msg.sender` . If this is done via the `Deployer` contract, then the owner of the `LiquidAcessories` contract will be the `Deployer` contract and not the `caller` of the `Deployer` contract. This will result in a situation, where `defineBondingCurveParameter()` , which protected by the `onlyOwner()` modifier, will not be able to be called as the `Deployer` contract does not have any functionality that allows it to call upon this function. This will result in a situation, where no liquid accessories can be minted, since the `bondingCurves` will not be able to be set. This scenario is not captured by the tests, as the tests are using `vm.prank()` to impersonate the address, which in this case will be the `Deployer` contract to test the functionality.

Recommendation: Either rewrite the deployment logic of `Deployer` to assign the appropriate `owner` to `LiquidAccessories` or introduce some sort of functionality in the `Deployer` contract that will allow the calling of `defineBondingCurveParameter()` .

VAP-3 Low Test Coverage

• **High** ⓘ `Mitigated`

> ⓘ **Alert**
>
> Test coverage has improved as of commit `ee38e358d52c984741230812707aedbe61bc2b3f` . However, branch and functional coverage continues to be low, especially for `MiladyAvatar.sol` and `SoulboundAccessories.sol` .
>
> | File | % Lines | % Statements | % Branches | % Funcs |
> |---|---|---|---|---|
> | src/LiquidAccessories.sol | 98.67% (74/75) | 98.96% (95/96) | 57.14% (24/42) | 92.31% (12/13) |
> | src/MiladyAvatar.sol | 66.20% (47/71) | 69.89% (65/93) | 50.00% (13/26) | 46.67% (14/30) |
> | src/Rewards.sol | 100.00% (35/35) | 100.00% (39/39) | 83.33% (15/18) | 85.71% (6/7) |
> | src/SoulboundAccessories.sol | 62.07% (18/29) | 61.11% (22/36) | 64.29% (9/14) | 50.00% (3/6) |

> ⓘ **Update**
>
> Marked as "Mitigated" by the client. Addressed in: `e7385ab69b497f3fe6a410a1634ae413ffb9e623` , `be5849999593bfd0a2168b25b7b58883753f738c` , `64bb8c29e3b8316e175a02a110cd39a39ae43a88` , `eed6cc929803ed167dccfcde75b5f07deba1dd25` , `68ec5e9eacdc49ff895f991f6ed62713cf786740` , `63633d8409b993ab8cae902f537d9d5cb44ca8c0` , `ef01f3cc1f854299909223e2d6bc9eef3593de94` , `89dfb3b3a24b9ed156933b7759f24f16b8b7d801` , `a6964966261c05ae71bda145df42265bb06bced0` . The client provided the following explanation:
>
> > We were not able to achieve 90% test coverage within our internal deadlines. We focused on tests that seemed most critical for the operation of our system, and also ignored view functions in favor of tests that cover state changes.

File(s) affected: `src/*`

Description: The current overall branch coverage for this audit was low. Please see the `Code Coverage` section of the report to reference the numbers. It is important to get overall branch coverage to at least 90%, ideally even 100%, to reduce the risk of encountering any unexpected behaviors. It is also important to test for access control issues, otherwise you may run the risk of encountering situations like VAP-2.

Recommendation: Get the overall branch coverage to at least 90% before deploying to production.

VAP-4 Reward Disbursement Can Be Front-Run

• **Medium** ⓘ `Acknowledged`

> ⓘ **Update**
>
> Marked as "Acknowledged" by the client. The client provided the following explanation:
>
> > We ignore ether accruing into the Rewards contract that does not come from accessory mints. Insofar as this assumption is invalid, then whoever is funding this action may attract the described attack; if so, all participants are still net positive after the attack, and the system will not break. In other words, if someone randomly throws $100 straight into the Rewards contract for some particular item, and some attacker finds profit in sandwiching this action to extract value from it, all legitimate users are not harmed (and in fact profit slightly) and no other aspects of our system suffer. If we concern ourselves only with the rewards that come in from the LiquidAccessory minting action: For an attacker to profit, he would have to earn from rewards not only the gas cost of the action, but also the 20% lost when reselling the accessory to the contract due to bonding curve spread. Now, the rewards the attacker would get would be half of the bonding curve spread - split among all current wearers of the item. Thus, for this attack to be profitable, we would have to imagine a ludicrously high

curveParameter and an extremely low amount of users wearing the item, compared to the amount minted. We consider this confluence too unlikely to address in the design of the contracts.

File(s) affected: `Rewards.sol`

Description: Rewards may be added to the `Rewards` contract in a number of different ways:

1. When minting liquid accessories through `LiquidAccessories.mintAccessories()`
2. Adding rewards directly through `Rewards.addRewardsForAccessory()`

Malicious users may frontrun calls when others are adding these rewards by equipping the item immediately before the reward is added and unequipping immediately after. A user may also acquire the accessory as part of the front-running.

Recommendation:

1. When adding rewards, consider using a private mempool such as with Flashbots.
2. Ensure that the rewards being added do not provide sufficient incentive for front-running. For example, adding fewer rewards on each call makes the front-running more expensive.

VAP-5 Ownership Can Be Renounced

• Low ⓘ   Acknowledged

> ⓘ **Update**
>
> Marked as "Acknowledged" by the client. The client provided the following explanation:
>
> > We consider the possibility of the owner renouncing ownership as the same class of mistake as transferring ownership to an incorrect ownership address. It's all down to responsible opsec. While we could override and disable the renounceOwnership functionality of the Ownable contract, this would add complexity to the project without actually addressing this broader concern of mishandling the ownership right in general. Thus we opt against implementing this suggestion.

File(s) affected: `LiquidAccessories.sol`

Description: If the owner renounces their ownership, all ownable contracts will be left without an owner. Consequently, any function guarded by the `onlyOwner` modifier will no longer be able to be executed.

Recommendation: Confirm that this is the intended behavior. If not, override and disable the `renounceOwnership()` function in the affected contracts. For extra security, consider using a two-step process when transferring the ownership of the contract (e.g. `Ownable2Step` from OpenZeppelin).

VAP-6 Privileged Roles and Ownership

• Low ⓘ   Acknowledged

> ⓘ **Update**
>
> Marked as "Acknowledged" by the client. The client provided the following explanation:
>
> > There are essentially three recommendations here: 1. document ownership roles; 2. use multisigs for privileged roles; 3. move toward more decentralized control of privileged roles.
> > 1. We will document this in user-facing documentation when we begin on documentation in general, which has not yet become a priority.
> > 2. We will use multisig for the owner of the LiquidAccessories bonding curve parameter setting, but this is infeasible for the miladyAuthority role. This is expected to be held by a secure server which responds automatically to requests from users to onboard, and must be able to act autonomously.
> > 3. Making the miladyAuthority role more decentralized is infeasible, as it relies on having privileged, authoritative information on the canonical metadata status of Miladys. Similarly, the bonding curve will be determined by canonical item rarity in the original set, which relies on authoritative access to the same data set. Thus there is no realistic road to either of these roles becoming more decentralized over time.

File(s) affected: `LiquidAccessories.sol` , `SoulboundAccessories.sol`

Description: Smart contracts will often have `owner` variables to designate the person with special privileges to make modifications to the smart contract. The following instances have been identified:

1. `LiquidAccessories`
   1. `owner` : Define the bonding curve parameter, which determines how expensive it is to mint the accessory.
2. `SoulboundAccessories`
   1. `miladyAuthority` : Onboard the Milady and mint its canonical accessories. This role could technically choose not to onboard selective Milady NFT IDs.

Recommendation: We recommend that the team describe all privileged roles in user-facing or public-facing documentation. Privileged roles should also be assigned to secure multi-sig wallets to reduce the risk of a private key compromise or malicious behavior from a single party. We also encourage the progressive decentralization of these roles over time.

VAP-7 Missing Input Validation   ● **Low** ⓘ   Fixed

> ℹ **Update**
>
> The client has implemented only several of the recommended checks, however we find their reasoning for not implementing the others to be sufficient.

> ✅ **Update**
>
> Marked as "Fixed" by the client. Addressed in: `99c87d3361289dc1b84b4cbef008f3c37be9c5bb` . The client provided the following explanation:
>
>> In many cases, validation can only protect against a vanishingly small amount of incorrect inputs. While it sounds reasonable to test for an empty string, or the zero address, in practice an empty string is just as incorrect as the string "" "", and address(0) is just as incorrect as address(1). Since it's infeasible to actually validate these inputs in any meaningful way, we opt for simplicity in code and remove the validation altogether, relying on deployer prudence - which we must rely on in any case, to no greater degree than if we'd had these narrow tests defined in require or assert statements. It's also worth noting that, where addresses are specified for receiving ether, if the user wishes (for some game theoretical reason) to burn the rewards, the most straightforward way to do this is to specify the zero address as the recipient. We see no reason to remove this option from the user. In the case of testing for non-empty arrays, we opt against testing for and explicitly rejecting these cases, as it complicates the code for no extra security gain. These functions perfectly fine when interpreting these calls as ""a list of 0 orders"". Requiring that the minRewardOut is greater than zero is not necessary. Consider the case in which the user would like to process the order no matter the reward. If we disallow setting minRewardOut to 0, the user would have to then specify an amount of 1 to effectively indicate to the contract ""I don't care how much I get out"". Since nothing about the intended behavior breaks if the user specifies a minRewardOut of 0, we opt to keep this option in.

File(s) affected: `SoulBoundAccessories.sol` , `LiquidAccessories.sol` , `Deployer.sol` , `TBARegistry.sol` , `TokenGatedAccount.sol` , `Rewards.sol`

Related Issue(s): SWC-123

Description: The following validation checks are missing from various contracts. The lack of these checks may lead to the contracts entering an incorrect state.
1. `Deployer.sol`
    1. `constructor()`
        1. Validate that the string parameters are not empty.
        2. Validate that the address parameters are not the zero address.
2. `LiquidAccessories.sol`
    1. `mintAccessories()`
        1. Validate that the address parameters are not the zero address.
        2. Validate that `_accessoryIds` , `_amounts` are not empty arrays.
    2. `_mintAccessoryAndDisburseRevenue()`
        1. Validate that `_amount` should be non-zero.
    3. `burnAccessories()`
        1. Validate that the address parameters are not the zero address.
        2. Validate that `_minRewardOut` is greater than zero.
    4. `_burnAccessory()`
        1. `_amount` should be non-zero.
3. `Rewards`
    1. `deregisterMiladyForRewardsForAccessoryAndClaim()` : Validate that the address parameters are not the zero address.
    2. `claimRewardsForMilady()` : Validate that the address parameters are not the zero address.
4. `SoulboundAccessories`
    1. * `mintAndEquipSoulboundAccessories()` : Validate that `_accessories` is not an empty list.
5. `TBARegistry`
    1. `createAccount()` : Validate that the address parameters are not the zero address.
6. `TokenGatedAccount`
    1. `bond()` : Validate that the address parameters are not the zero address.
    2. `execute()` : Validate that the address parameters are not the zero address.

Recommendation: We recommend adding the validation checks.

VAP-8 Unbounded Curve Parameter Can Lead to Overflow   ● **Low** ⓘ   Acknowledged

File(s) affected: `LiquidityAccessories.sol`

Description: The `defineBondingCurveParameter()` function allows setting of a curve parameter for every accessory. This curve paramter is used to calculate the price of the accessory needed for minting or refunded on burning. The following equation is used:

```
0.005 ether + _curveParameter * _itemNumber * _itemNumber
```

As `_curveParameter` is an arbitrary constant, if set too high, the whole calculation can overflow and DoS the minting process.

Recommendation: Consider upper bounding the `_curveParameter`, so that it would not overflow the calculation.

VAP-9 Vulnerable Solidity Version   • Low ⓘ   Fixed

File(s) affected: `src/*`

Description: As security standards develop, so does the Solidity language. In order to stay up to date with current practices, it's important to use a recent version of Solidity and recent conventions. The version used in all the contracts in scope, `0.8.13` , is known to have vulnerabilities, for more information see here.

Recommendation: Consider using Solidity version `0.8.18` instead and refer to the list of recommended versions for up-to-date suggestions.

VAP-10 Reentrancy Guards Can Reduce Attack Surface   • Low ⓘ   Fixed

File(s) affected: `LiquidAccessories.sol` , `MiladyAvatar.sol` , `Rewards.sol` , `SoulboundAccessories.sol` , `TBARegistry.sol` , `TokenGatedAccount.sol`

Description: There are several functions that transfer value and make numerous external calls. We have identified the following functions that make external calls:

1. `LiquidAccessories`
   1. `mintAccessories()`
   2. `burnAccessories()`
2. `MiladyAvatar`
   1. `updateEquipSlotsByAccessoryIds()`
   2. `equipSoulboundAccessories()`
   3. `preTransferUnequipById()`
3. `Rewards`
   1. `addRewardsForAccessory()`
   2. `deregisterMiladyForRewardsForAccessoryAndClaim()`
   3. `claimRewardsForMilady()`
4. `SoulboundAccessories.mintAndEquipSoulboundAccessories()`
5. `TBARegistry.createAccount()`
6. `TokenGatedAccount.execute()`

Recommendation: While we have not found any explicit attack vectors and the use of `transfer()` limits the possibility of reentrancy due to the hardcoded gas limit, there are some inherit risk to relying on `transfer()` see `VAP-18` . If looking to move away from using `transfer()` ,

consider adding re-entrancy guards to reduce the possible attack surface.

VAP-11
`revenueRecipient` Cannot Be Changed if It Gets Compromised

● **Informational** ⓘ   Fixed

> ✓ **Update**
>
> Marked as "Fixed" by the client. Addressed in: `e7385ab69b497f3fe6a410a1634ae413ffb9e623` .

File(s) affected: `LiquidityAccessories.sol`

Description: `revenueRecipient` is the address where the whole of `freeRevenue` or half of it is send depending if anyone is wearing that specific accessory or not. It is currently not possible to change this address if the private key of it gets compromised.

Recommendation: Consider adding ability to change `revenueRecipient` address or make `revenueRecipient` a multi-sig account.

VAP-12 Bonding Curve Can only Be Set Once

● **Informational** ⓘ   Acknowledged

> ⓘ **Update**
>
> Marked as "Acknowledged" by the client. The client provided the following explanation:
>
>> The suggestion to enable changing the bonding curve is not feasible, as all logic in the contract related to the bonding curve assumes a pure mathematical function. At the very least, raising the bonding curve after an item has been purchased would render the contract insolvent, and lowering it would trap ether in the contract. As pointed out, this introduces the risk that the parameter is set incorrectly, and cannot be updated after the fact. This is essentially the same class of risk as VAP-19. In both cases, to specify the authority address to a specific implementation of a multisig actually limits our options upon deployment. By leaving this as simply an address, we are free to choose any smart contract wallet, including a multisig. In the case of all owner roles, including this one, the plan is to deploy with an EOA, then transfer all ownership roles to company multisigs; after doing this and verifying the state of the contracts are as expected, the system will be declared canonical.

File(s) affected: `LiquidAccessories.sol`

Description: `defineBondingCurveParameter()` has the following requirement check:

```
require(bondingCurves[_accessoryId].curveParameter == 0, "Parameter already set");
```

This means that the `curveParameter` can only be set once. This can potentially be an issue if the Vaporware team would either accidentally misconfigure the value of a certain accessory or would like to update the value of a certain accessory.

Recommendation: Consider whether this functionality aligns with the Vaporware team's business logic, and if not consider introducing functionality to update the value of the `curveParameter` .

VAP-13 Missing Event Emissions

● **Informational** ⓘ   Acknowledged

> ⓘ **Update**
>
> Marked as "Acknowledged" by the client. The client provided the following explanation:
>
>> We consider the suggested new events superfluous. In the case of 1-4, the set variables are public, and cn be verified that way. In the case of 5, we consider an event for a transaction execution superfluous.

File(s) affected: `LiquidAccessories.sol` , `TokenGatedAccount.sol` , `MiladyAvatar.sol` , `SoulboundAccessories.sol`

Description: In order to validate the proper deployment and initialization of the contracts, it is a good practice to emit events. Also, any important state transitions can be logged, which is beneficial for monitoring the contract, and also tracking eventual bugs or hacks. Below, we present a non-exhaustive list of functions that could emit events to improve application management:

1. `LiquidAccessories.defineBondingCurveParameter()`
2. `LiquidAccessories.setAvatarContract()`
3. `SoulboundAccessories.setAvatarContract()`
4. `MiladyAvatar.setOtherContracts()`
5. `TokenGatedAccount.execute()`

Recommendation: Consider emitting the events mentioned above.

VAP-14 Events Emitted After External Calls

● **Informational** ⓘ `Fixed`

> ℹ **Update**
>
> Marked as "Acknowledged" by the client. The client provided the following explanation:
>
> > Implemented re-entrancy guards as per VAP-10

File(s) affected: `MiladyAvatar.sol` , `Rewards.sol` , `SoulboundAccessories.sol` , `TBARegistry.sol`

Description: It is generally recommended to emit events before external calls. Otherwise, it is possible that events will be emitted in the wrong order if reentrancy occurs. The following events are emitted after external calls.

1. `MiladyAvatar`
   1. `AccessoryEquipped`
   2. `AccessoryUnequipped`
2. `Rewards`
   1. `MiladyDeregisteredForRewards`
   2. `RewardsClaimed`
3. `SoulboundAccessories.SoulboundAccessoriesMinted`
4. `TBARegistry.AccountCreated()`

Recommendation: Emit the events after the external call. Alternately, use reentrancy guards to prevent the possibility of reentrancy changing the correct order of event emission.

VAP-15 Draft Eip Used

● **Informational** ⓘ `Acknowledged`

> ℹ **Update**
>
> Marked as "Acknowledged" by the client. The client provided the following explanation:
>
> > We acknowledge and accept the risk that the EIP 6551 spec will change after our system is deployed, which would render our TBA/TGA code technically noncompliant with the standard. We opt against implementing upgrade strategies which would let us respond to this scenario to regain compliance, as this would drastically complicate our system, and 6551 compliance is not a priority compared to other project goals.

File(s) affected: `IERC6551Account.sol` , `IERC6551Executable.sol` , `IERC6551Registry.sol` , `TBARegistry.sol` , `TokenGatedAccount.sol`

Description: The contracts rely heavily on the `EIP-6551` , which has not officially been accepted yet. If changes are made to this standard before it is accepted, the contracts may not integrate well with other applications that expect the official standard to be used.

Recommendation: The team should consider migration possibilities if there are modifications to the official version of `ERC-6551` .

VAP-16 Mixing Test Code with Production Code

● **Informational** ⓘ `Fixed`

> ✅ **Update**
>
> Marked as "Fixed" by the client. Addressed in: `cdf932c6f5182e00d79d9fa12f3d137fcf58b345` .

File(s) affected: `HardcodedGoerliDeployer.sol`

Description: The contract `HardcodedGoerliDeployer` is intended to facilitate testing rather than being deployed as part of the final project. It is ideal to keep production code separate from testing code. A failure to separate them could lead to mistakes in development or the mistaken use of the testing code within production contracts.

Recommendation: Move `HardcodedGoerliDeployer` into a `test` folder to clearly distinguish it from the production code.

VAP-17 Unlocked Pragma

● **Informational** ⓘ `Fixed`

> ✅ **Update**
>
> Marked as "Fixed" by the client. Addressed in: `0a12db6c53dbb7461e4de22203a02a639a2727d0` .

File(s) affected: `Deployer.sol` , `HardcodedGoerliDeployer.sol` , `LiquidAccessories.sol` , `MiladyAvatar.sol` , `Rewards.sol` , `SoulboundAccessories.sol` , `IERC6551Account.sol` , `IERC6551Executable.sol` , `IERC6551Registry.sol` , `TBARegistry.sol` , `TokenGatedAccount.sol`

Related Issue(s): SWC-103

Description: Every Solidity file specifies in the header a version number of the format `pragma solidity (^)0.8.*` . The caret (`^`) before the version number implies an unlocked pragma, meaning that the compiler will use the specified version *and above*, hence the term "unlocked".

Recommendation: For consistency and to prevent unexpected behavior in the future, we recommend to remove the caret to lock the file onto a specific Solidity version.

VAP-18 `transfer()` Could Run Out of Gas and Lead to Dos   • **Informational** ⓘ   Fixed

> ✓ **Update**
>
> Marked as "Fixed" by the client. Addressed in: `8c98aeb5e773ae5cc25c8cb259335789dfb9a481` .

File(s) affected: `LiquidityAccessories.sol` , `Rewards.sol`

Description: Use of the `address.transfer()` function is discouraged, since this function forwards only 2300 gas, which might break some non-trivial fallback functions if any future change increases the gas cost of opcodes operations.

Recommendation: Consider replacing calls to `transfer()` with raw `call.value(amount)("")` calls and make sure to check the return value of these calls for success and to respect the Checks-Effects-Interactions Pattern.

VAP-19 `miladyAuthority` Should Be a Multi Sig   • **Informational** ⓘ   Mitigated

> ⓘ **Update**
>
> The client provided the following explanation for their fix
>
> > The recommendation to make the miladyAuthority address a multisig is infeasible, for the following reason:
>
> The miladyAuthority address will be a server that responds to "onboarding requests" from users, immediately. This can't require manual intervention by a human, so if we are committed to using a multisig, we are really discussing having two separate servers, each with its own key.
>
> However, this starts to explode the scope: You'd have to develop two servers independently - ideally, even by two different developers - otherwise, a hack that works on one server will work on the other, and the address has not been made meaningfully any more secure. Furthermore, you'd have the added complexity of having one server *propose* the call with a given payload, and then the other server would need to then decode that call and the payload, to verify it agreed with the veracity of the payload.
>
> Instead of this, we've opted to add an ownership key, which operationally will be a multisig (as with the other ownership keys in the system). In the case of the miladyAuthority address being compromised, the owner can first change the miladyAuthority address to a fresh address, and then reverse any damage done by the attacker by making a series of unmintAndUnequipSoulboundAccessories calls.
>
> With this fix, while the miladyAuthority key is still relatively unprotected, the damage it can do is entirely reversible. This also drastically lowers the incentive for the attack in the first place. Meanwhile, the owner key (which can do the "real irreversible damage" that used to be available to the miladyAuthority) can be held safely behind a multisig which is rarely (if ever) needed to be used.
>
> While adding another ownership role to the system could be said to increase centralization, we consider it a minor increase in centralization. In addition, in the future when all soulbound accessories are minted (we may opt at some point to just mint the remainder in a batch transaction) we can both change the miladyAuthority to the zero address and finally revoke ownership, removing this point of centralization completely.

> ✓ **Update**
>
> Marked as "Fixed" by the client. Addressed in: `84d534fe0aec02c780ac894cfa250417fc3af86a` and `b3803b523899aea8f411380ab7d092ea84a5f92e` .

File(s) affected: `SoulboundAccessories.sol`

Description: The `miladyAuthority` account is used to mint the soulbound accessories as an onboarding system for new users. It is very important to keep this account safe as this is an increased privilege account.

Recommendation: Consider making `miladyAuthority` account a multi-sig account.

Definitions

- **High severity** – High-severity issues usually put a large number of users' sensitive information at risk, or are reasonably likely to lead to catastrophic impact for client's reputation or serious financial implications for client and users.

- **Medium severity** – Medium-severity issues tend to put a subset of users' sensitive information at risk, would be detrimental for the client's reputation if exploited, or are reasonably likely to lead to moderate financial impact.

- **Low severity** – The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low impact in view of the client's business circumstances.

- **Informational** – The issue does not post an immediate risk, but is relevant to security best practices or Defence in Depth.

- **Undetermined** – The impact of the issue is uncertain.

- **Fixed** – Adjusted program implementation, requirements or constraints to eliminate the risk.

- **Mitigated** – Implemented actions to minimize the impact or likelihood of the risk.

- **Acknowledged** – The issue remains in the code but is a result of an intentional business or design decision. As such, it is supposed to be addressed outside the programmatic means, such as: 1) comments, documentation, README, FAQ; 2) business processes; 3) analyses showing that the issue shall have no negative consequences in practice (e.g., gas analysis, deployment settings).

Code Documentation

`Fixed` 1. Code readability can be improved by documenting all functions and events using the NatSpec standard.

`Fixed` 2. Consider having a code comment for each mapping which documents what each key and value represents. For example: `(accessoryId => RewardInfoForAccessory)` for the `Rewards.rewardInfoForAccessory` mapping.

Adherence to Best Practices

1. `Fixed` Use `unchecked` blocks for incrementing the iterator in for-loops to save gas.
2. `Fixed` Some typos can be corrected:
   1. `MiladyAvatar.initiaDeployer` ⇒ `MiladyAvatar.initialDeployer`
3. `Acknowledged` It is more gas-efficient to use custom errors in revert instead of strings.
4. `Fixed` Several state variables which are only modified in the constructor can be marked as `immutable`, including but not limited to:
   1. `Rewards.miladysContract`
   2. `Rewards.avatarContractAddress`
   3. `MiladyAvatar.miladysContract`
5. `Fixed` In `MiladyAvatar._equipAccessoryIfOwned()`, consider replacing the `assert()` statement with `require()`, as the former would consume all the remaining gas upon failure.
6. `Acknowledged` `_updateEquipSlotByTypeAndVariant()` recomputes accessoryId that is already known.

Appendix

File Signatures

The following are the SHA-256 hashes of the reviewed files. A file with a different SHA-256 hash has been modified, intentionally or otherwise, after the security review. You are cautioned that a different SHA-256 hash could be (but is not necessarily) an indication of a changed condition or potential vulnerability that was not within the scope of the review.

Contracts

- `b03...350 ./src/Rewards.sol`
- `91a...274 ./src/MiladyAvatar.sol`
- `ece...a04 ./src/SoulboundAccessories.sol`
- `54b...f39 ./src/HardcodedGoerliDeployer.sol`
- `2a3...1b1 ./src/LiquidAccessories.sol`
- `510...1a9 ./src/Deployer.sol`
- `b16...b9f ./src/TGA/TokenGatedAccount.sol`
- `ed3...939 ./src/TGA/IERC6551Executable.sol`
- `240...706 ./src/TGA/TBARegistry.sol`
- `d51...d92 ./src/TGA/IERC6551Registry.sol`
- `efc...aba ./src/TGA/IERC6551Account.sol`

Tests

- `f6d...33c` `./test/GhostNFT.t.sol`
- `6c6...aa7` `./test/TestUtils.sol`
- `76f...613` `./test/TestConstants.sol`
- `ff8...82d` `./test/BitPacking.t.sol`
- `ed8...2f8` `./test/SoulboundAccessories.t.sol`
- `18e...291` `./test/MiladyOSTestBase.sol`
- `b8c...bc6` `./test/Rewards.t.sol`
- `955...8ea` `./test/TestSetup.sol`
- `886...a41` `./test/Miladys.sol`
- `c7b...ab7` `./test/TokenGatedAccount.t.sol`
- `8fe...41c` `./test/Harnesses.t.sol`
- `92d...19f` `./test/LiquidAccessories.t.sol`
- `9c5...040` `./test/SpecTests/SoulboundAccessories.t.sol`
- `fca...ae5` `./test/SpecTests/MiladyAvatar.t.sol`
- `e2f...b05` `./test/SpecTests/Rewards.t.sol`

Toolset

The notes below outline the setup and steps performed in the process of this audit.

Setup

Tool Setup:
- Slither ↗ v0.9.3

Steps taken to run the tools:
1. Install the Slither tool: `pip3 install slither-analyzer`
2. Run Slither from the project directory: `slither .`

Automated Analysis

Slither

All the issues that have been identified by Slither have either been false positives or covered in the report. We would like to however draw attention to the following output regarding re-entrancy detection in the contracts:

```
INFO:Detectors:
Reentrancy in MiladyAvatar._equipAccessoryIfOwned(uint256,uint256) (src/MiladyAvatar.sol#87-102):
    External calls:
    - _unequipAccessoryByTypeIfEquipped(_miladyId,accType) (src/MiladyAvatar.sol#95)
        - rewardsContract.deregisterMiladyForRewardsForAccessoryAndClaim(_miladyId,equipSlots[_miladyId]
[_accType],getPayableAvatarTBA(_miladyId)) (src/MiladyAvatar.sol#112)
    State variables written after the call(s):
    - equipSlots[_miladyId][accType] = _accessoryId (src/MiladyAvatar.sol#97)
    MiladyAvatar.equipSlots (src/MiladyAvatar.sol#54) can be used in cross function reentrancies:
    - MiladyAvatar._equipAccessoryIfOwned(uint256,uint256) (src/MiladyAvatar.sol#87-102)
    - MiladyAvatar._unequipAccessoryByTypeIfEquipped(uint256,uint128) (src/MiladyAvatar.sol#107-118)
    - MiladyAvatar.equipSlots (src/MiladyAvatar.sol#54)
Reentrancy in MiladyAvatar._unequipAccessoryByTypeIfEquipped(uint256,uint128) (src/MiladyAvatar.sol#107-
118):
    External calls:
    - rewardsContract.deregisterMiladyForRewardsForAccessoryAndClaim(_miladyId,equipSlots[_miladyId]
[_accType],getPayableAvatarTBA(_miladyId)) (src/MiladyAvatar.sol#112)
    State variables written after the call(s):
    - equipSlots[_miladyId][_accType] = 0 (src/MiladyAvatar.sol#116)
    MiladyAvatar.equipSlots (src/MiladyAvatar.sol#54) can be used in cross function reentrancies:
    - MiladyAvatar._equipAccessoryIfOwned(uint256,uint256) (src/MiladyAvatar.sol#87-102)
    - MiladyAvatar._unequipAccessoryByTypeIfEquipped(uint256,uint128) (src/MiladyAvatar.sol#107-118)
    - MiladyAvatar.equipSlots (src/MiladyAvatar.sol#54)
```

As mentioned in the report, there are potential areas where re-entrancy can occur, however as long as the contracts that external calls are being made to are trusted along with value transfer being done via the `transfer()` function, the chances of malicious re-entrancy attacks seem low. It would not hurt however to add additional re-entrancy guards to be safe, please see VAP-10 for a list of functions that could benefit from this.

Test Suite Results

All tests are passing.

```
Running 1 test for test/SoulboundAccessories.t.sol:SoulboundAccessoriesTests
[PASS] test_minting() (gas: 431183)
Test result: ok. 1 passed; 0 failed; 0 skipped; finished in 1.11s

Running 3 tests for test/Rewards.t.sol:RewardsTest
[PASS] test_auditRewardIssuePoc() (gas: 918616)
[PASS] test_basicRewards() (gas: 1083584)
[PASS] test_rewardsWithExiting() (gas: 624644)
Test result: ok. 3 passed; 0 failed; 0 skipped; finished in 1.11s

Running 3 tests for test/LiquidAccessories.t.sol:LiquidAccessoriesTests
[PASS] test_autoUnequip() (gas: 556644)
[PASS] test_autoUnequipBatch() (gas: 613613)
[PASS] test_revenueFlow() (gas: 608354)
Test result: ok. 3 passed; 0 failed; 0 skipped; finished in 1.12s

Running 1 test for test/BitPacking.t.sol:BitPacking
[PASS] test_packBackAndForth(uint256) (runs: 256, μ: 6980, ~: 6980)
Test result: ok. 1 passed; 0 failed; 0 skipped; finished in 1.18s

Running 4 tests for test/GhostNFT.t.sol:GhostNFT
[PASS] test_balanceOfForTGAIs1(uint256) (runs: 256, μ: 30368, ~: 30368)
[PASS] test_balanceOfRandomAcccountIs0(address) (runs: 256, μ: 25697, ~: 25697)
[PASS] test_idValidity(uint256) (runs: 256, μ: 9550, ~: 8753)
[PASS] test_ownershipTracks() (gas: 148511)
Test result: ok. 4 passed; 0 failed; 0 skipped; finished in 1.31s

Running 1 test for test/SpecTests/SoulboundAccessories.t.sol:SoulboundAccessoriesTests
[PASS] test_SA_SAC_3(address) (runs: 256, μ: 1870367, ~: 1870367)
Test result: ok. 1 passed; 0 failed; 0 skipped; finished in 188.41ms

Running 2 tests for test/CurveTests.sol:CurveTests
[PASS] test_BurnCalculations(uint256,uint256,uint256,uint256) (runs: 256, μ: 191753, ~: 206668)
[PASS] test_MintCalculations(uint256,uint256,uint256,uint256) (runs: 256, μ: 185955, ~: 209745)
Test result: ok. 2 passed; 0 failed; 0 skipped; finished in 2.05s

Running 7 tests for test/SpecTests/MiladyAvatar.t.sol:MiladyAvatarTests
[PASS] test_MA_ESA_3(uint256,uint256,uint256) (runs: 256, μ: 590435, ~: 577384)
[PASS] test_MA_ESA_8(uint256,uint256,uint256) (runs: 256, μ: 529336, ~: 544958)
[PASS] test_MA_PTUBI_1(uint256,uint256) (runs: 256, μ: 16016, ~: 16016)
[PASS] test_MA_PTUBI_4(uint256,uint256) (runs: 256, μ: 254062, ~: 255217)
[PASS] test_MA_UABSTAV_4(uint256,uint256) (runs: 256, μ: 33265, ~: 33265)
[PASS] test_MA_UABSTAV_7(uint256,uint256) (runs: 256, μ: 446590, ~: 448215)
[PASS] test_MA_UABSTAV_9(uint256,uint256) (runs: 256, μ: 324798, ~: 325878)
Test result: ok. 7 passed; 0 failed; 0 skipped; finished in 254.15s

Running 8 tests for test/SpecTests/LiquidAccessories.t.sol:LiquidAccessoriesTests
[PASS] test_LA_BAS_2(uint256,uint256,address) (runs: 256, μ: 715154, ~: 714542)
[PASS] test_LA_BA_1(uint256,uint256,uint256) (runs: 256, μ: 180405, ~: 220155)
[PASS] test_LA_BTT_2(uint256,uint256,uint256) (runs: 256, μ: 457476, ~: 454402)
[PASS] test_LA_MADR_3(uint256,uint256,address) (runs: 256, μ: 167379, ~: 167431)
[PASS] test_LA_MADR_4(uint256,uint256,address) (runs: 256, μ: 729610, ~: 729457)
[PASS] test_LA_MA_4(uint256,uint256,address,address) (runs: 256, μ: 294420, ~: 286661)
[PASS] test_LA_MA_8(uint256,uint256,address,address) (runs: 256, μ: 321481, ~: 315528)
[PASS] test_LA_SAC_3() (gas: 2292859)
Test result: ok. 8 passed; 0 failed; 0 skipped; finished in 308.09s

Running 8 tests for test/SpecTests/Rewards.t.sol:RewardsTests
[PASS] test_R_AR_1(uint256) (runs: 256, μ: 13567, ~: 13567)
[PASS] test_R_AR_2(uint256) (runs: 256, μ: 22929, ~: 22929)
[PASS] test_R_AR_3(uint256,uint256,uint256) (runs: 256, μ: 339853, ~: 340883)
[PASS] test_R_CRFM_1(uint256,uint256,address) (runs: 256, μ: 25441, ~: 25441)
[PASS] test_R_CRFM_3(uint256,uint256,uint256,address) (runs: 256, μ: 694465, ~: 693998)
[PASS] test_R_DMFRFAAC_1(address,uint256,uint256,address) (runs: 256, μ: 14406, ~: 14406)
[PASS] test_R_DMFRFAAC_4(uint256,uint256) (runs: 256, μ: 373257, ~: 374493)
```

```
[PASS] test_R_RMFRFA_1(address,uint256,uint256) (runs: 256, μ: 9137, ~: 9137)
Test result: ok. 8 passed; 0 failed; 0 skipped; finished in 308.09s
Ran 10 test suites: 38 tests passed, 0 failed, 0 skipped (38 total tests)
```

Code Coverage

Overall branch coverage is low for this project, especially for `MiladyAvatar.sol` and `SoulboundAccessories.sol`. We would like to see at least 90% in this area, ideally 100%.

File	% Lines	% Statements	% Branches	% Funcs
src/LiquidAccessories.sol	98.67% (**74**/75)	98.96% (**95**/96)	57.14% (**24**/42)	92.31% (**12**/13)
src/MiladyAvatar.sol	66.20% (**47**/71)	69.89% (**65**/93)	50.00% (**13**/26)	46.67% (**14**/30)
src/Rewards.sol	100.00% (**35**/35)	100.00% (**39**/39)	83.33% (**15**/18)	85.71% (**6**/7)
src/SoulboundAccessories.sol	62.07% (**18**/29)	61.11% (**22**/36)	64.29% (**9**/14)	50.00% (**3**/6)

Changelog

- 2023-10-11 - Initial report
- 2023-10-26 - Final report

About Quantstamp

Quantstamp is a global leader in blockchain security. Founded in 2017, Quantstamp's mission is to securely onboard the next billion users to Web3 through its best-in-class Web3 security products and services.

Quantstamp's team consists of cybersecurity experts hailing from globally recognized organizations including Microsoft, AWS, BMW, Meta, and the Ethereum Foundation. Quantstamp engineers hold PhDs or advanced computer science degrees, with decades of combined experience in formal verification, static analysis, blockchain audits, penetration testing, and original leading-edge research.

To date, Quantstamp has performed more than 500 audits and secured over $200 billion in digital asset risk from hackers. Quantstamp has worked with a diverse range of customers, including startups, category leaders and financial institutions. Brands that Quantstamp has worked with include Ethereum 2.0, Binance, Visa, PayPal, Polygon, Avalanche, Curve, Solana, Compound, Lido, MakerDAO, Arbitrum, OpenSea and the World Economic Forum.

Quantstamp's collaborations and partnerships showcase our commitment to world-class research, development and security. We're honored to work with some of the top names in the industry and proud to secure the future of web3.

Notable Collaborations & Customers:
- Blockchains: Ethereum 2.0, Near, Flow, Avalanche, Solana, Cardano, Binance Smart Chain, Hedera Hashgraph, Tezos
- DeFi: Curve, Compound, Maker, Lido, Polygon, Arbitrum, SushiSwap
- NFT: OpenSea, Parallel, Dapper Labs, Decentraland, Sandbox, Axie Infinity, Illuvium, NBA Top Shot, Zora
- Academic institutions: National University of Singapore, MIT

Timeliness of content

The content contained in the report is current as of the date appearing on the report and is subject to change without notice, unless indicated otherwise by Quantstamp; however, Quantstamp does not guarantee or warrant the accuracy, timeliness, or completeness of any report you access using the internet or other means, and assumes no obligation to update any information following publication or other making available of the report to you by Quantstamp.

Notice of confidentiality

This report, including the content, data, and underlying methodologies, are subject to the confidentiality and feedback provisions in your agreement with Quantstamp. These materials are not to be disclosed, extracted, copied, or distributed except to the extent expressly authorized by Quantstamp.

Links to other websites

You may, through hypertext or other computer links, gain access to web sites operated by persons other than Quantstamp. Such hyperlinks are provided for your reference and convenience only, and are the exclusive responsibility of such web sites&aspo; owners. You agree that Quantstamp are not responsible for the content or operation of such web sites, and that Quantstamp shall have no liability to you or any other person or entity for the use of third-party web sites. Except as described below, a hyperlink from this web site to another web site does not imply or mean that Quantstamp endorses the content on that web site or the operator or operations of that site. You are solely responsible for determining the extent to which you may use any content at any other web sites to which you link from the report. Quantstamp assumes no responsibility for the use of third-party software on any website and shall have no liability whatsoever to any person or entity for the accuracy or completeness of any output generated by such software.

Disclaimer

The review and this report are provided on an as-is, where-is, and as-available basis. To the fullest extent permitted by law, Quantstamp disclaims all warranties, expressed or implied, in connection with this report, its content, and the related services and products and your use thereof, including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement. You agree that your access and/or use of the report and other results of the review, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your sole risk. FOR AVOIDANCE OF DOUBT, THE REPORT, ITS CONTENT, ACCESS, AND/OR USAGE THEREOF, INCLUDING ANY ASSOCIATED SERVICES OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER ADVICE. This report is based on the scope of materials and documentation provided for a limited review at the time provided. You acknowledge that Blockchain technology remains under development and is subject to unknown risks and flaws and, as such, the report may not be complete or inclusive of all vulnerabilities. The review is limited to the materials identified in the report and does not extend to the compiler layer, or any other areas beyond the programming language, or other programming aspects that could present security risks. The report does not indicate the endorsement by Quantstamp of any particular project or team, nor guarantee its security, and may not be represented as such. No third party is entitled to rely on the report in any way, including for the purpose of making any decisions to buy or sell a product, service or any other asset. Quantstamp does not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by a third party, or 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, or any related services and products, any hyperlinked websites, or any other websites or mobile applications, and we will not be a party to or in any way be responsible for monitoring any transaction between you and any third party. As with the purchase or use of a product or service through any medium or in any environment, you should use your best judgment and exercise caution where appropriate.

Vaporware