PALADIN
BLOCKCHAIN SECURITY

Smart Contract Security Assessment

Final Report

For Vincask

27 November 2024

paladinsec.co

info@paladinsec.co

Table of Contents

Disclaimer

1 Overview

This report has been prepared for Vincask contracts on the Ethereum network. Paladin provides a user-centred examination of the smart contracts to look for vulnerabilities, logic errors or other issues from both an internal and external perspective.

1.1 Summary

Project Name	Vincask
URL	TBC
Platform	Ethereum
Language	Solidity
Preliminary Contracts	https://github.com/0xGuvnor/vincask_contracts/tree/fe7e01f35da357cc4e15516743eadc45598baffa
Resolution	https://github.com/0xGuvnor/vincask_contracts/commit/76d574d1f5599af7a6d576d5612d4c0e9b8ad265

1.2 Contracts Assessed

Name	Contract	Live Code Match
`VinCask`		
`VinCaskX`		

1.3 Findings Summary

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)	Failed Resolution
● Governance	0	-	-	-	-
● High	5	5	-	-	-
● Medium	3	2	-	-	1
● Low	8	6	-	2	-
● Informational	1	-	1	-	-
Total	**17**	**13**	**1**	**2**	**1**

Classification of Issues

Severity	Description
● **Governance**	Issues under this category are where the governance or owners of the protocol have certain privileges that users need to be aware of, some of which can result in the loss of user funds if the governance's private keys are lost or if they turn malicious, for example.
● **High**	Exploits, vulnerabilities or errors that will certainly or probabilistically lead towards loss of funds, control, or impairment of the contract and its functions. Issues under this classification are recommended to be fixed with utmost urgency.
● **Medium**	Bugs or issues that may be subject to exploit, though their impact is somewhat limited. Issues under this classification are recommended to be fixed as soon as possible.
● **Low**	Effects are minimal in isolation and do not pose a significant danger to the project or its users. Issues under this classification are recommended to be fixed nonetheless.
● **Informational**	Consistency, syntax or style best practices. Generally pose a negligible level of risk, if any.

1.3.1 VinCask

ID	Severity	Summary	Status
01	HIGH	`safeMultiMintAndBurnForAdmin` incorrectly reduces `maxCirculatingSupply` and `totalSupply`	✓ RESOLVED
02	HIGH	Missing access control on `safeMultiMintWithCreditCard`	✓ RESOLVED
03	HIGH	`safeMultiMintWithCreditCard` charges a fixed amount	✓ RESOLVED
04	HIGH	Minting functions can be reentered and cause the maximum token supply restrictions to be bypassed	✓ RESOLVED
05	HIGH	Inconsistent updates of `tokensBurned` cause `getCirculatingSupply` to show incorrect information	✓ RESOLVED
06	MEDIUM	Not all minting methods have `whenNotPaused` modifier	✓ RESOLVED
07	MEDIUM	`setStableCoin` can cause incorrect pricing of the mints due to a difference in decimals	✓ RESOLVED
08	MEDIUM	The `mintCompliance` modifier restricts minting based on circulating supply instead of the total minted amount	FAILED
09	LOW	Contract does not support fee on transfer tokens	ACKNOWLEDGED
10	LOW	`removeWhitelistAddress` might revert or be very expensive	✓ RESOLVED
11	LOW	`multiRedeem` can be called outside of website UI	ACKNOWLEDGED
12	LOW	`multiApprove` approves only the contract itself	✓ RESOLVED
13	LOW	Use `safeTransferFrom` for the stablecoin transfers	✓ RESOLVED
14	LOW	`MULTI_SIG` should be editable in case it gets compromised	✓ RESOLVED
15	LOW	Ambiguous use of `maxCirculatingSupply` and `totalSupply`	✓ RESOLVED
16	INFO	Typographical issues	PARTIAL

1.3.1 VinCaskX

ID	Severity	Summary	Status
17	LOW	`safeMint` access control can cause problems in the future transfer of ownership	✓ RESOLVED

2 Findings

2.1 VinCask

`VinCask` is an NFT contract that entitles its token holders to redeem their tokens for a bottle of VinCask whisky in real life. Tokens will be available for purchase via a defined stable coin and credit card (by using Crossmint). Whitelisted users will be able to mint certain quantities free of charge. Once redemptions are turned on by the contract owner, users will be able to burn their NFT in exchange for a VinCaskX NFT that will be used as proof of redemption.

Minting

- Users pay the minting price in the defined `stableCoin` token by via `safeMultiMintWithStableCoin`
- `safeMultiMintWithCreditCard` allows users to mint by paying with a credit card
- Whitelisted users are able to mint free of charge by calling the `safeMultiMintForWhitelist` function
- Admin is able to record physical sales with the `safeMultiMintAndBurnForAdmin` function

Redeem

- Redemptions are done through the `multiRedeem` function.

2.1.1 Privileged Functions

- `safeMultiMintForWhitelist`
- `safeMultiMintAndBurnForAdmin`
- `increaseCirculatingSupply`
- `setMintPrice`
- `setStableCoin`
- `setWhitelistAddress`
- `removeWhitelistAddress`
- `pause`
- `unpause`
- `openRedemption`
- `closeRedemption`
- `renounceOwnership`
- `transferOwnership`

2.1.2 Issues & Recommendations

Issue #01	**`safeMultiMintAndBurnForAdmin` incorrectly reduces `maxCirculatingSupply` and `totalSupply`**
Severity	● HIGH SEVERITY
Description	`safeMultiMintAndBurnForAdmin` is intended to be used by the owner for physical sales by minting and burning tokens at the same time. `maxCirculatingSupply` and `totalSupply` should not be reduced in this case since the minting or burning of an NFT should not update the restraints the contract enforces for how much tokens should be minted overall.
Recommendation	To fix the function we should remove the reduction of `maxCirculatingSupply` and `totalSupply`.
Resolution	✓ RESOLVED The recommended fix from issue 16 was implemented and the function emits an event and increment state variables (renamed the variables to `totalMintedCount` and `adminBurnedCount` for better naming clarity).

Issue #02	Missing access control on `safeMultiMintWithCreditCard`
Severity	● HIGH SEVERITY
Description	`safeMultiMintWithCreditCard` is intended to only be called by Crossmint when a user wants to pay with a credit card. Anybody can call this function as there is no access control.
Recommendation	Consider giving access only to one or some of CrossMint's treasury addresses as it is mentioned in their documentation. Make sure there is an owner privileged function that can change this treasury address.
Resolution	✓ RESOLVED Access control for a CrossMint treasury address was implemented to call the function, along with a setter function for the admin to change the address.

Issue #03	`safeMultiMintWithCreditCard` charges a fixed amount
Severity	● HIGH SEVERITY
Description	As per the CrossMint documentation (link), the `safeMultiMintWithCreditCard` function should be minting tokens free of charge. Furthermore, hardcoding the price is not recommended because if the purchasing token is changed with another one that has different decimals this hardcoded price will cause users to pay either too much or too little.
Recommendation	Consider removing the hardcoded price.
Resolution	✓ RESOLVED The hardcoded price was removed.

Issue #04	Minting functions can be reentered and cause the maximum token supply restrictions to be bypassed
Severity	● HIGH SEVERITY
Description	ERC721 calls `_checkOnERC721Received` when `_safeMint` or `_safeTransfer` are called. `_checkOnERC721Received` makes a call with `onERC721Received` to the receiving entity if it is a contract. This means if we mint an NFT to a contract, it can execute code when its `onERC721Received` method gets called. This allows the receiving contract to re-enter into VinCask's function. Before `_safeMultiMint` gets called, the `mintCompliance` modifier will check if the minted quantity is not going to be beyond the maximum token supply. However, if the receiving contract re-enters a minting function in the first iteration of the minting, this check can be bypassed. Example: 1. Contract calls `safeMultiMintWithStableCoin(10)` 2. Circulating supply is 90, `maxCirculatingSupply` is 100 3. The `mintCompliance` check passes 90 + 10 is not more than 100 4. On the first iteration of the minting the receiving contract calls `safeMultiMintWithStableCoin(9)` 5. The `mintCompliance` check passes 91 + 9 is not more than 100 6. The minting iterations of both multi mint calls finish and the circulating supply is now 90 + 10 + 9 = 109
Recommendation	Consider placing reentrancy guards on all mint/burn functions. You can use the OpenZeppelin `ReentrancyGuard.sol` contract. Additionally, consider using `_mint` instead of the `_safeMint` of the ERC721.
Resolution	✓ RESOLVED Reentrancy guards were added on all mint/burn functions from OpenZeppelin's `ReentrancyGuard.sol`.

Issue #05	**Inconsistent updates of `tokensBurned` cause `getCirculatingSupply` to show incorrect information**
Severity	● HIGH SEVERITY
Description	Circulating supply is `tokenCounter - tokensBurned`. `getCirculatingSupply` is used by all mint functions to ensure the maximum token restrictions of the contract are enforced. `tokensBurned` however is not increased when we burn tokens in `multiRedeem`. This causes the maximum token restrictions to be bypassed.
Recommendation	Consider increasing `tokensBurned` in `multiRedeem`.
Resolution	✓ RESOLVED Renamed `getCirculatingSupply` to `getTotalMintedForCap` and `tokensBurned` to `adminBurnedCount` for clarity.

Issue #06	**Not all minting methods have whenNotPaused modifier**
Severity	● MEDIUM SEVERITY
Description	`safeMultiMintForWhitelist` is missing the whenNotPaused modifier.
Recommendation	Consider adding the modifier.
Resolution	✓ RESOLVED

Issue #07	**`setStableCoin` can cause incorrect pricing of the mints due to a difference in decimals**
Severity	● MEDIUM SEVERITY
Description	If the owner wants to change the pricing currency of the tokens with one that does not have the same number of decimals, users would be able to mint with an incorrect price. Example: - Price currency USDC, 6 decimals, 100 USDC is the price per mint (100e6) - Owner changes the currency to a stablecoin with 18 decimals. - Attacker backruns the transaction before the owner has the chance to change the price from 100e6 to 100e18 - The attacker is able to mint an NFT for 100e6 in the new stablecoin currency which is not 100 USD but 0.0000000001 USD
Recommendation	Consider updating the stablecoin together with the minting price in the same method.
Resolution	✓ RESOLVED A state variable `stableCoinDecimals` was added to keep track of decimals and `mintPrice` will be assumed to have 18 decimals by default.

Issue #08	**The `mintCompliance` modifier restricts minting based on circulating supply instead of the total minted amount**
Severity	● MEDIUM SEVERITY
Description	Since the number of whisky bottles are limited, `mintCompliance` should restrict minting functions from minting more NFTs than there are bottles in reality. As of now `mintCompliance` limits the minting based on circulating supply. This is incorrect since circulating supply will be reduced during redemptions and with the current logic, this will allow for the minting of more NFTs.
Recommendation	Consider editing `mintCompliance` to revert based on the minted tokens instead of the circulating supply.
Resolution	● FAILED RESOLUTION `MintCompliance` modifier checks if all minted tokens minus the locally sold ones are more than the minting cap. If the minting cap and total supply have to respect the fixed amount of whiskeys in real life, consider replacing `uint256 currentMinted = getTotalMintedForCap();` with `uint256 currentMinted = totalMintedCount;` inside of the `mintCompliance` modifier.

Issue #09	Contract does not support fee on transfer tokens
Severity	● LOW SEVERITY
Description	Some stablecoins have a functionality for charging fees on transfers. If turned on, Vincask will be minting tokens for less than the defined minting price.
Recommendation	If the team wishes to support fee on transfer stablecoins, consider recording the balance of the multi-signature contract before and after the stablecoin transfer to see the real amount that was received and revert if the amount is not enough. The amount of tokens needed to be transferred to the multi-signature contract is calculated in `_safeMultiMint`. This would DOS the minting of tokens that were bought with stablecoins if we implement the before and after transfer balance check from above because there is no way for the user to make the contract take more tokens than the defined minting price and if there is a fee the amount the multi-signature contract will receive is always going to be below the desired price. To mitigate this the user could be let to define the amount of stablecoins he is willing to pay so he could send more in order to compensate if the stablecoin turned on their fee functionality.
Resolution	● ACKNOWLEDGED The team does not intend to support fee on transfer tokens, only either USDC or USDT.

Issue #10	`removeWhitelistAddress` might revert or be very expensive
Severity	● LOW SEVERITY
Description	Since there is no limit to how many users are going to be whitelisted, if the whitelist becomes too large, this can cause the `removeWhitelistAddress` function to reach the block gas limit and revert when trying to remove an address that is on the end of the array. If the array indeed becomes too large to reach the addresses at the end, the owner will be forced to remove users in the front first in order to make the array smaller and then make another attempt to remove the desired address at the end. Additionally, the function is going to be more expensive in terms of gas the more iteration `_findIndex` has to go through to find the address.
Recommendation	Consider searching for the address index off-chain and when calling `removeWhitelistAddress`, just provide the index that needs to be removed.
Resolution	✓ RESOLVED The whitelist is intended to probably have 5 or less addresses for members within the team, so gas should not be much of an issue. A check was added to ensure the list does not get larger than 10 addresses as a safeguard.

Issue #11	`multiRedeem` can be called outside of website UI
Severity	● LOW SEVERITY
Description	The contract comments state: "For redemptions to be eligible, they MUST be initiated through the website UI as users have to complete a form with their redemption details." This is not enforced and anybody can call `multiRedeem`.
Recommendation	There are a couple of ways to get the redemption details from users. `multiRedeem` can be guarded by a signature verification check that the user needs to get from your backend in order to successfully call `multiRedeem`. The con here is that the system becomes very centralized. The second way is to not have access control on `multiRedeem` as it is now and if anybody wants to get the whiskey in real life, they will have to go to your website, get verified that their wallet holds the VinCaskX NFT and input their redemption details in.
Resolution	● ACKNOWLEDGED The team commented: "A prominent notice will be placed on the website UI to remind users to go through the UI to redeem instead of calling the contract directly. In cases where the redemption is called directly to the contract, the user can still redeem their whisky by contacting the company directly and we will manually verify their details."

Issue #12	**`multiApprove` approves only the contract itself**
Severity	● LOW SEVERITY
Description	In the function comment, it is stated that `multiApprove` is intended to give approvals only to certain tokens instead of all like `setApprovalForAll` does. `multiApprove` however can only give approvals to the contract itself which is not very helpful since the contract does not need to be approved in order to do something with the tokens of its users. You can test this by going to `VincaskTest.t.sol` => `test_RedeemedNftHasSameTokenId()` => and comment out "`vin.multiApprove(tokenIdArray);`". When you run the test you will see that it will pass just like before. Furthermore, we consider that this function is actually risky and should be deleted—`setApprovalForAll` should always be used by the users to know about the risks of approving multiple tokens to an address.
Recommendation	Consider adding an address "`to`" argument to `multiApprove` and use it to give specific approvals to this address instead of to the contract itself. Additionally, consider deleting this function as this function prevents the user from understanding the risks of `setApprovalForAll`.
Resolution	✓ RESOLVED The `multiApprove` function was deleted.

Issue #13	Use `safeTransferFrom` for the stablecoin transfers
Severity	● LOW SEVERITY
Description	Some tokens return booleans while others do not. For this reason, it is considered a best practice to use the `SafeERC20.sol` OpenZeppelin library when making token transfers.
Recommendation	Consider replacing `stableCoin.transferFrom()` with `stableCoin.safeTransferFrom()`.
Resolution	✓ RESOLVED

Issue #14	`MULTI_SIG` should be editable in case it gets compromised
Severity	● LOW SEVERITY
Description	The `MULTI_SIG` address where all stablecoin tokens go is immutable. This is problematic since there is no way for the owner of VinCask to change it if the `MULTI_SIG` address gets compromised. Additionally, the ability to change the royalty receiver is also not implemented which results in having the same receiver even if the multi-signature wallet gets compromised.
Recommendation	Consider allowing the owner to be able to change the `MULTI_SIG` address. Additionally, if this is implemented, consider adding a function to modify the royalty receiver as well by making an `onlyOwner` function that calls `_setDefaultRoyalty`.
Resolution	✓ RESOLVED `MULTI_SIG` was renamed to `multiSig` as it is no longer an immutable variable. Setter functions were added to update `multiSig` and royalties.

Issue #16	Ambiguous use of `maxCirculatingSupply` and `totalSupply`
Severity	● LOW SEVERITY
Description	`maxCirculatingSupply` currently tries limit the existing circulating supply while `totalSupply` represents the maximum supply existent —currently this is ambiguous and should be simplified to only one variable to avoid ambiguity and mistakes.
Recommendation	Consider having `maxCirculatingSupply` as the maximum supply that can be minted at all time—this should basically check the `tokenIdCounter` to not go over this amount. It should never decrease, only increase with the mint. `totalSupply` should start from 0 and increase in the mint function and decrease with the burn, which means `getCirculatingSupply` should actually be replaced with `totalSupply` . `tokensBurned` should just be an informational variable that increments when any token is burnt.
Resolution	✓ RESOLVED `maxCirculatingSupply` and `totalSupply` variables have been renamed to `mintingCap` and `maxSupply` respectively for better clarity.

Issue #17	Typographical issues
Severity	● INFORMATIONAL
Description	Use a concrete Solidity version. — There is no need for most state variables to be private. If they are public, Solidity automatically creates getter functions for them. Also as of now users cannot easily check the address of `VincaskX` through the `Vincask` contract. — In the constructor, `tokenCounter` and `redemptionOpen` by default are 0 and false. It is not necessary to set these values. — `increaseCirculatingSupply`, `setMintPrice`, `setStableCoin`, `setWhitelistAddress`, `removeWhitelistAddress`, pause and unpause are not emitting events. Consider adding events for these functions. — Within `setWhitelistAddress`, replace `if (_mintLimit <= 0)”` `with “if (_mintLimit == 0)` because `uint256` cannot be less than 0. — On line 161, there is no need to define a variable. `tokenCounter` could be used directly. — `safeMultiMintAndBurnForAdmin` could just emit an event and increase the `tokenId` and not call mint and burn of the same `tokenId` that just consumes unnecessary gas. — The `_safeMultiMint` on line 401 can just use `_safeMint(_to, tokenCounter)` and avoid the unnecessary declaration of `tokenId`.

	The `VinCaskX` contract is unnecessarily imported, an interface should be used instead to save gas on deployment. — `totalSupply` should be a public state variable as `totalSupply` will be picked by Etherscan to be shown in the token page. — Remove `_beforeTokenTransfer` on line 460 as it is not needed. — Consider checking in the constructor if `maxSupply` is more or equal to `mintingCap`. — Before deployment do not forget to change `_baseURI` to the correct URI on both `Vincask` and `VincaskX`. It is also recommended to have a variable for the URI that only the owner can change in case in the future the metadata URI has to be changed for some reason.
Recommendation	Consider fixing the typographical issues.
Resolution	● PARTIALLY RESOLVED

2.2 VinCaskX

`VinCaskX` is used by VinCask to mint tokens during redemptions that are used as proof.

2.2.1 Privileged Functions

- `safeMint`
- `renounceOwnership`
- `transferOwnership`

2.2.2 Issues & Recommendations

Issue #18	**`safeMint` access control can cause problems in the future transfer of ownership**
Severity	● LOW SEVERITY
Description	The `safeMint` function is called by the VinCask contract during redemptions and in order to do that the VinCask contract has to be the owner of VinCaskX. The issue is that VinCask does not have defined methods for the ownership management of VinCaskX. If for whatever reason the ownership of VinCaskX needs to be changed this will not be possible once the VinCask contract becomes the owner.
Recommendation	Consider implementing `AccessControl` from OpenZeppelin and have the `DEFAULT_OWNER` as the multi-signature wallet and give a special role that can mint to the VinCask contract.
Resolution	✓ RESOLVED Implemented `AccessControl` instead of `Ownable` from `OpenZeppelin` and created a `MINTER_ROLE`. In the deployment script, `DeployVinCask.s.sol`, `VinCask` is granted the `MINTER_ROLE` and `DEFAULT_ADMIN_ROLE` is granted to the multi-signature address.

PALADIN
BLOCKCHAIN SECURITY