Blaize.Security

January 5th 2023 / V. 1.0

cupcake

SMART CONTRACT AUDIT

TABLE OF CONTENTS

AUDIT RATING

Cupcake contract's source code was taken from the repository provided by the Cupcake team.

SCORE **9.6**/10

The scope of the project includes the **Cupcake** set of contracts**:**

Contract.sol
CandyMachine.sol
CandyMachineFactory.sol
RentableWrapper.sol

Initial repository:
https://github.com/leopoldjoy/cupcake-contract#-cupcake

Initial commit:

- c82dd521149f350f6f006ea3fecd341b29e8424b

Final commit:

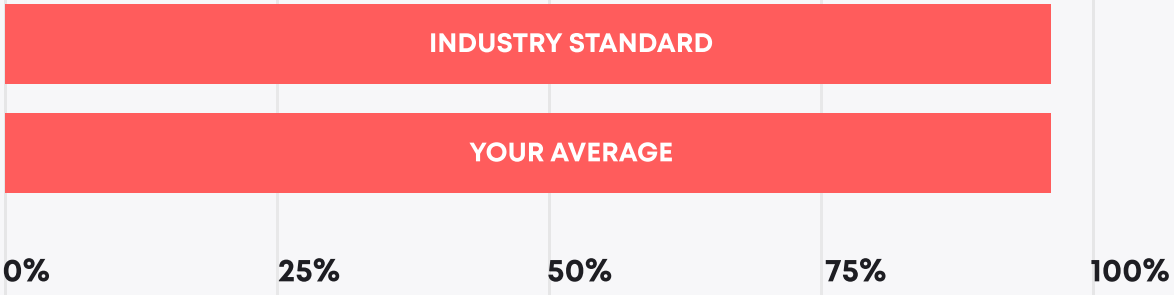
- df54a75c867ca58519c1836946b7e70c51cda4eb

At the end of the audit contracts were merged to the **main repository** and auditors verified their integrity.
Repository:
https://github.com/cupcake/contract
FInal commit:
9fef304dde4802b241e9d4da565bf5c41bdd8a39

TECHNICAL SUMMARY

During the audit, we inspected the security of the smart contracts of the Cupcake protocol. Our task was to find and describe any security issues in the smart contracts of the platform. This report presents the findings of the security audit of the **Cupcake** smart contracts conducted during **November 28th, 2022 - December 19th, 2022.**

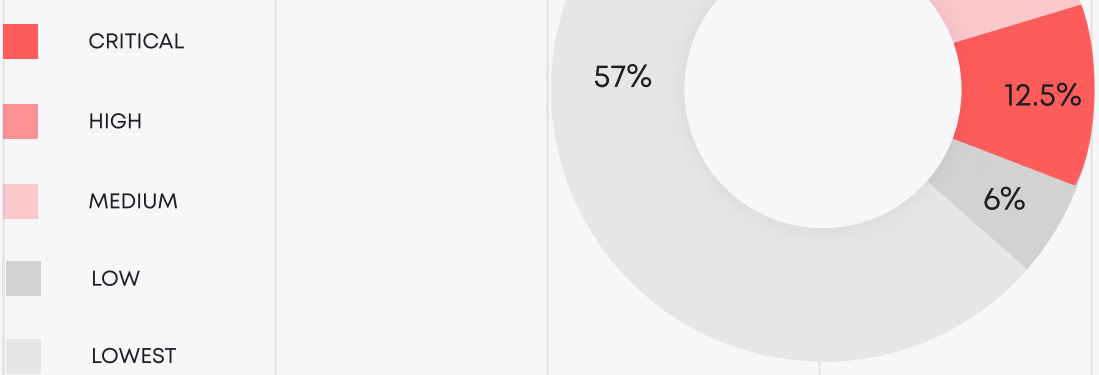
Testable code



The testable code has sufficient coverage, which corresponds the industry standard of 95%.

The scope of the audit includes the unit test coverage, which is based on the smart contracts code, documentation, and requirements presented by the Cupcake team. The coverage is calculated based on the set of the Hardhat framework tests and scripts from additional testing strategies. However, in order to ensure full security of the contract, the Blaize.Security team suggests the Cupcake team launch a bug bounty program to encourage further active analysis of the smart contracts.

THE GRAPH OF VULNERABILITIES DISTRIBUTION:



The table below shows the number of the detected issues and their severity. A total of 16 problems were found. All the issues were successfully fixed by the Cupcake team.

	FOUND	FIXED/VERIFIED
Critical	2	2
High	1	1
Medium	3	3
Low	1	1
Lowest	9	9

SEVERITY DEFINITION

Critical

The system contains several issues ranked as very serious and dangerous for users and the secure work of the system. Requires immediate fixes and a further check.

High

The system contains a couple of serious issues, which lead to unreliable work of the system and migh cause a huge data or financial leak. Requires immediate fixes and a further check.

Medium

The system contains issues that may lead to medium financial loss or users' private information leak. Requires immediate fixes and a further check.

Low

The system contains several risks ranked as relatively small with the low impact on the users' information and financial security. Requires fixes.

Lowest

The system does not contain any issues critical to the secure work of the system, but best practices should be implemented.

AUDITING STRATEGY AND TECHNIQUES APPLIED/PROCEDURE

We have scanned this smart contract for commonly known and more specific vulnerabilities:

- Unsafe type inference;
- Timestamp Dependence;
- Reentrancy;
- Implicit visibility level;
- Gas Limit and Loops;
- Transaction-Ordering Dependence;
- Unchecked external call - Unchecked math;
- DoS with Block Gas Limit;
- DoS with (unexpected) Throw;
- Byte array vulnerabilities;
- Malicious libraries;
- Style guide violation;
- ERC20 API violation;
- Uninitialized state/storage/ local variables;
- Compile version not fixed.

Procedure

We checked the contract for the following parameters:

- Whether the contract is secure;
- Whether the contract corresponds to the documentation;
- Whether the contract meets the best practices in the efficient use of gas, code readability.

Automated analysis:

Scanning contract by several publicly available automated analysis tools such as Mythril, Solhint, Slither, and Smartdec. Manual verification of all the issues found with these tools.

Manual audit:

Manual analysis of smart contracts for security vulnerabilities. We checked smart contract logic and compared it with the one described in the documentation.

EXECUTIVE SUMMARY

During the audit, the Blaize Security team has checked the whole set of smart contracts provided by the Cupcake team. The protocol consists of 4 contracts:

- CandyMachine.sol, an ERC1155 smart-contract with a custom minting functionality based on the Chainlink VRF Oracle in order to randomly choose URI of the minted token;
- CandyMachineFactory.sol, a factory contract designed for deploying new instances of CandyMachine contracts;
- RentableWrapper.sol, an ERC721 smart-contract designed to wrap the existing external NFTs to extend their interface with EIP-4907 user interface;
- Contract.sol, a contract designed for the distribution of ERC20, ERC721, and ERC1155 assets in different modes, which are called tags type.

The goal of the audit was to analyze the listed smart contracts in terms of well-known security vulnerabilities, check the contracts against the Blaize.Security internal vulnerabilities check-list, validate the security of users' funds, the safety of ERC721 implementation (including transfer and mint operations), check that contracts correspond the best practises in terms of code quality and gas optimization. The team of auditors found 2 critical, 1 high, and 3 medium-severity issues in the contracts, as well as several low and informational ones. One of the critical issues was connected to the generation of random numbers on-chain. This issue was successfully fixed by the Cupcake team by integrating the Chainlink VRF Oracle. The other critical issue was connected to the possible deletion of information about a wrapped asset in RentableWrapper.sol. The issue was fixed as well by deleting only necessary information instead of the whole data about the NFT.

Other issues were connected to the violation of the upgradable proxy pattern, the safety of ERC20 asset transfer, gas optimizations, and the validation of the business logic of the contracts. A complete list of all detected vulnerabilities can be seen in the Complete analysis section.

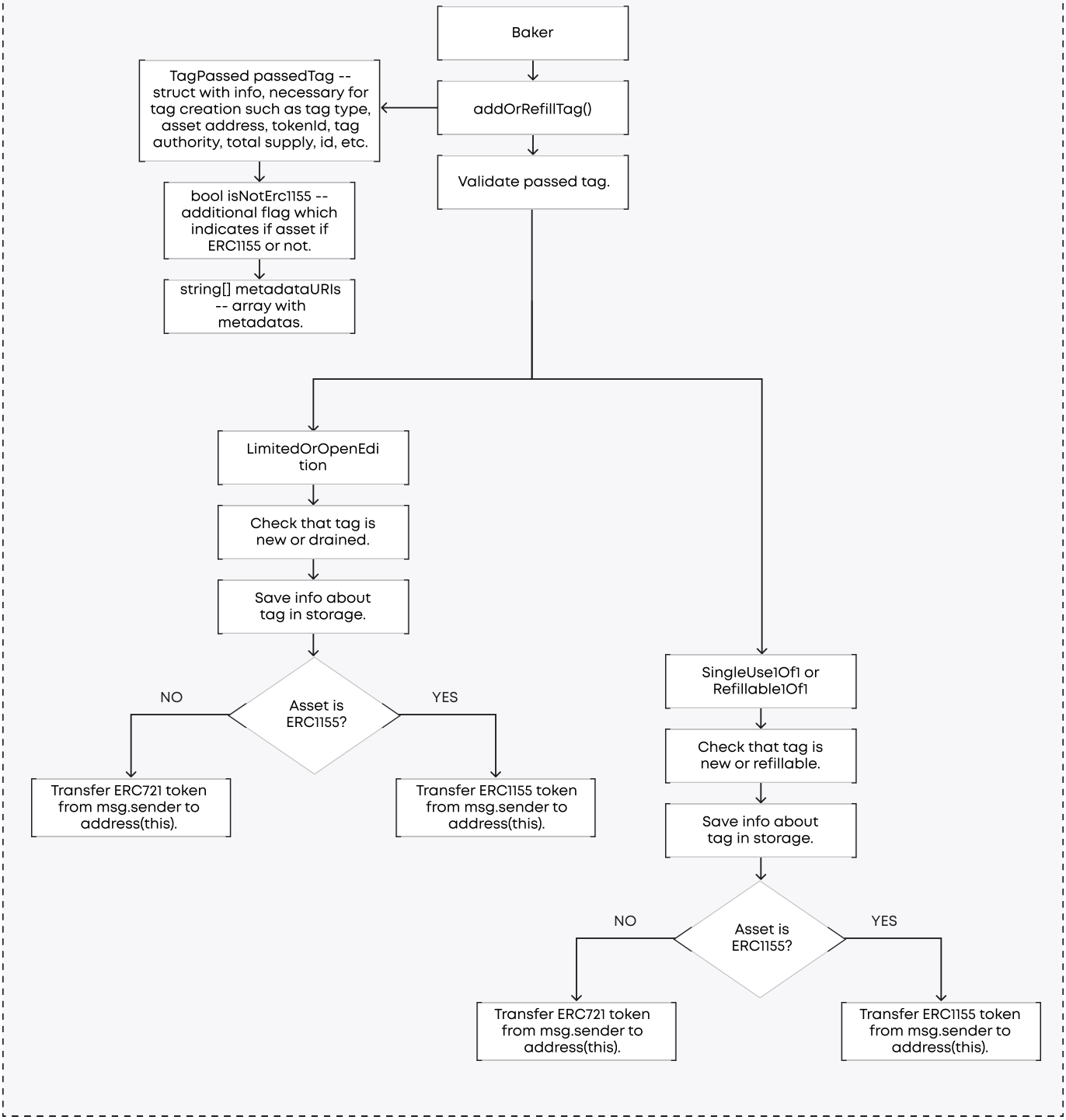
There are two more aspects worth mentioning. First of all, all the contracts, except for CandyMachine, are upgradable, which means that the owner of the contracts can change their logic at any time. The other thing is connected to the ability of users to use arbitrary assets for wrapping and creating tags. Since such an issue might be a potential threat for the users of the protocol, auditors proposed several approaches to protect the protocol against it. According to the Cupcake team, all assets passed to the contracts will be checked on the dApp with additional scripts. It is also validated inside the smart contracts that the passed assets implement the necessary interface.

The overall security of the smart contracts is high enough. The contracts are well-written and have a good natspec documentation. Yet, they lack additional documentation, so we recommended Cupcake to prepare additional documentation describing the logic of the contracts. The Cupcake team has also provided a sufficient set of unit tests that validate all the essential operations in the smart contracts. Nevertheless, our team has prepared our own set of unit tests and additional scenarios.

	RATING
Security	9.8
Gas usage and logic optimization	9.7
Code quality	9.5
Test coverage	9.5
Total	9.6

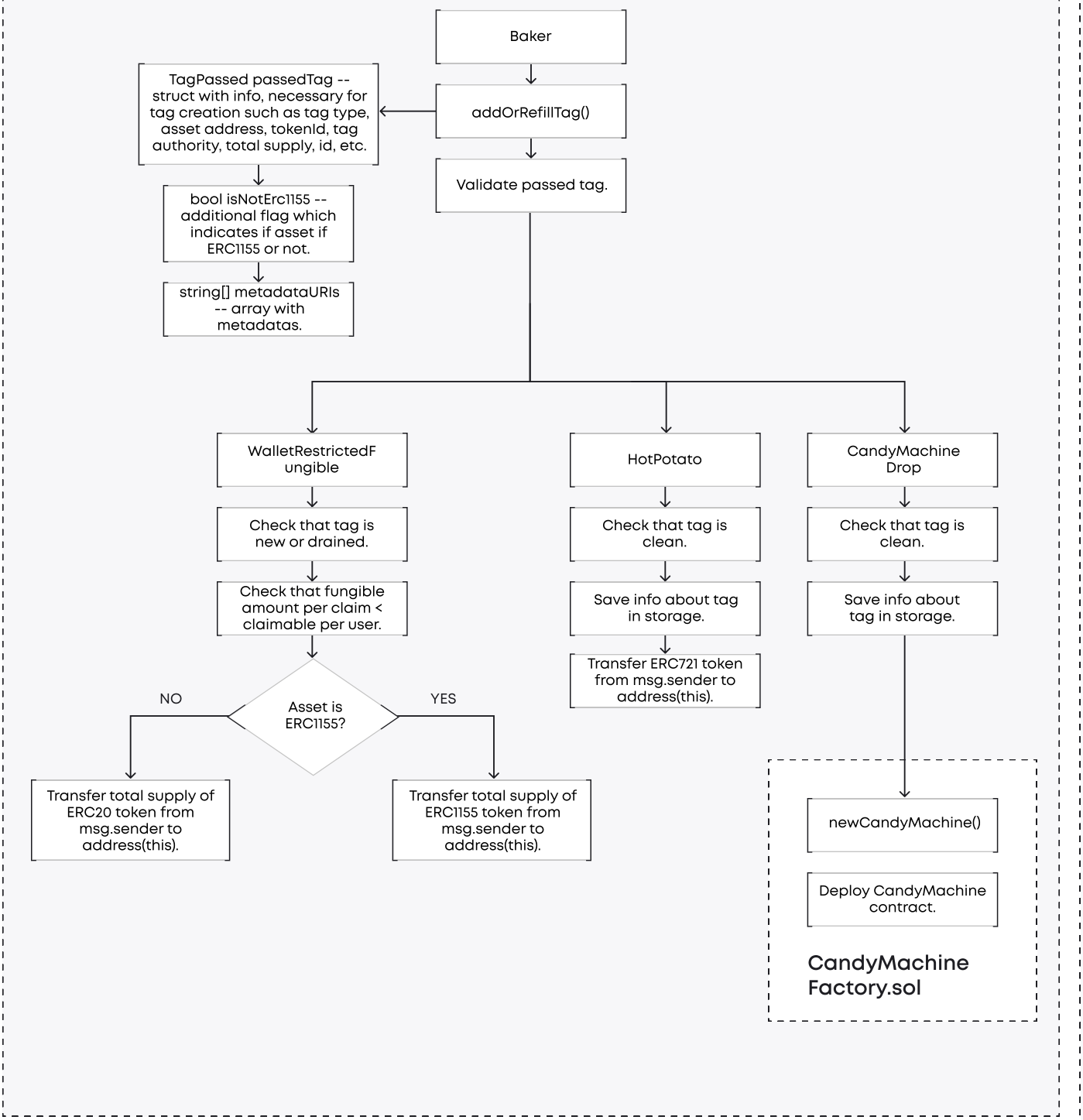
CUPCAKE PROTOCOL

Tag creation



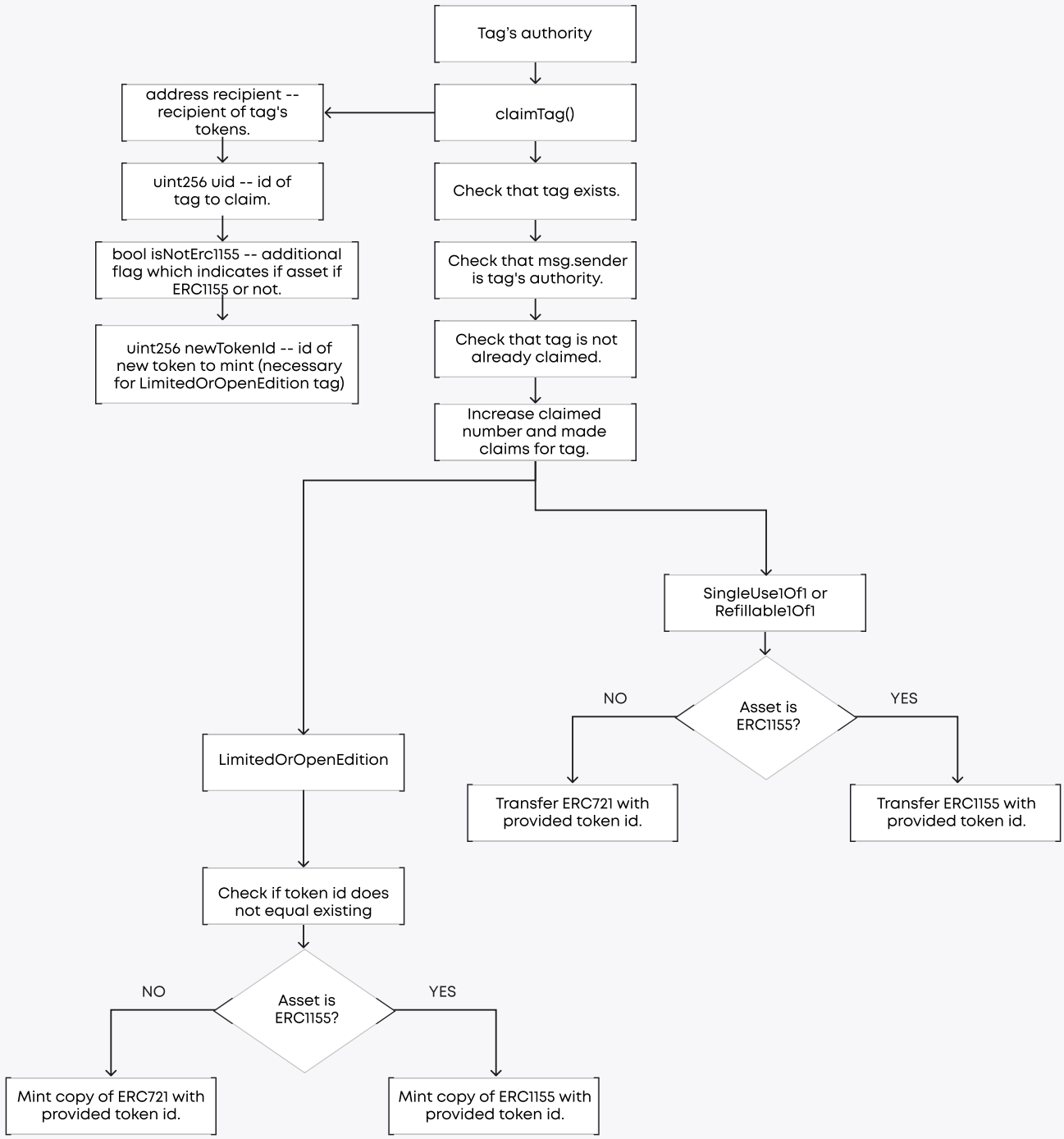
CUPCAKE PROTOCOL

Tag creation



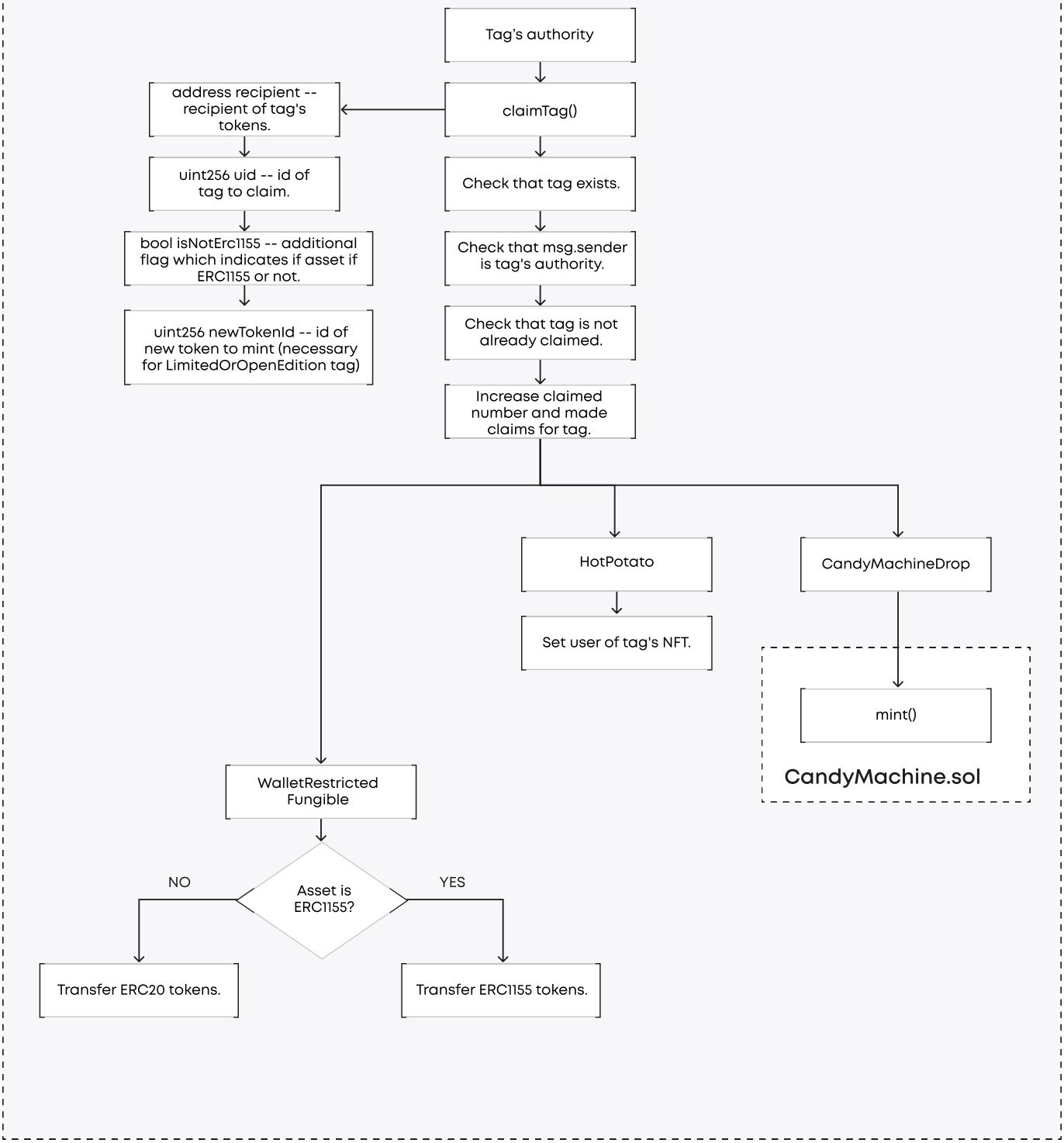
CUPCAKE PROTOCOL

Claim tag



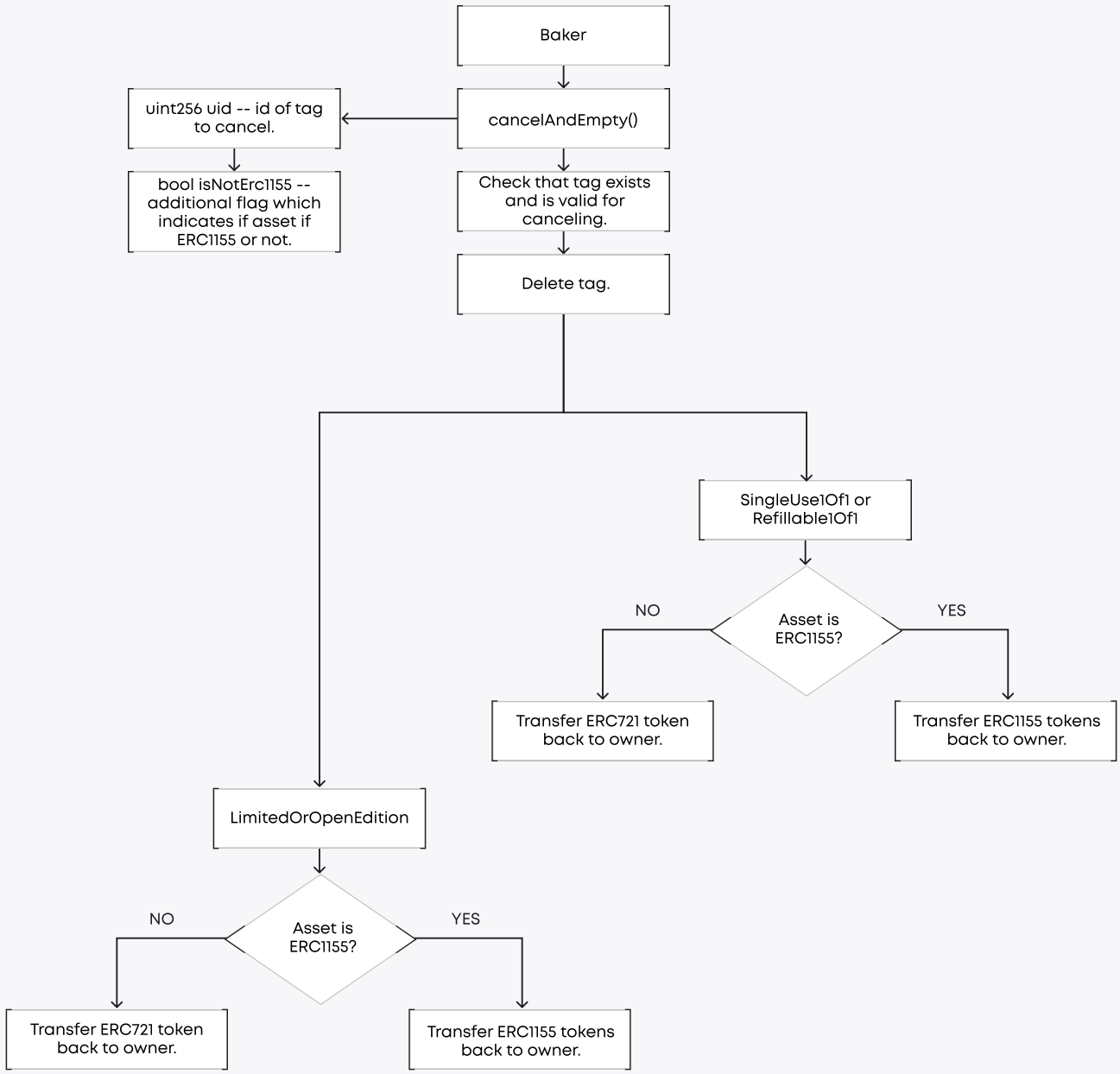
CUPCAKE PROTOCOL

Claim tag



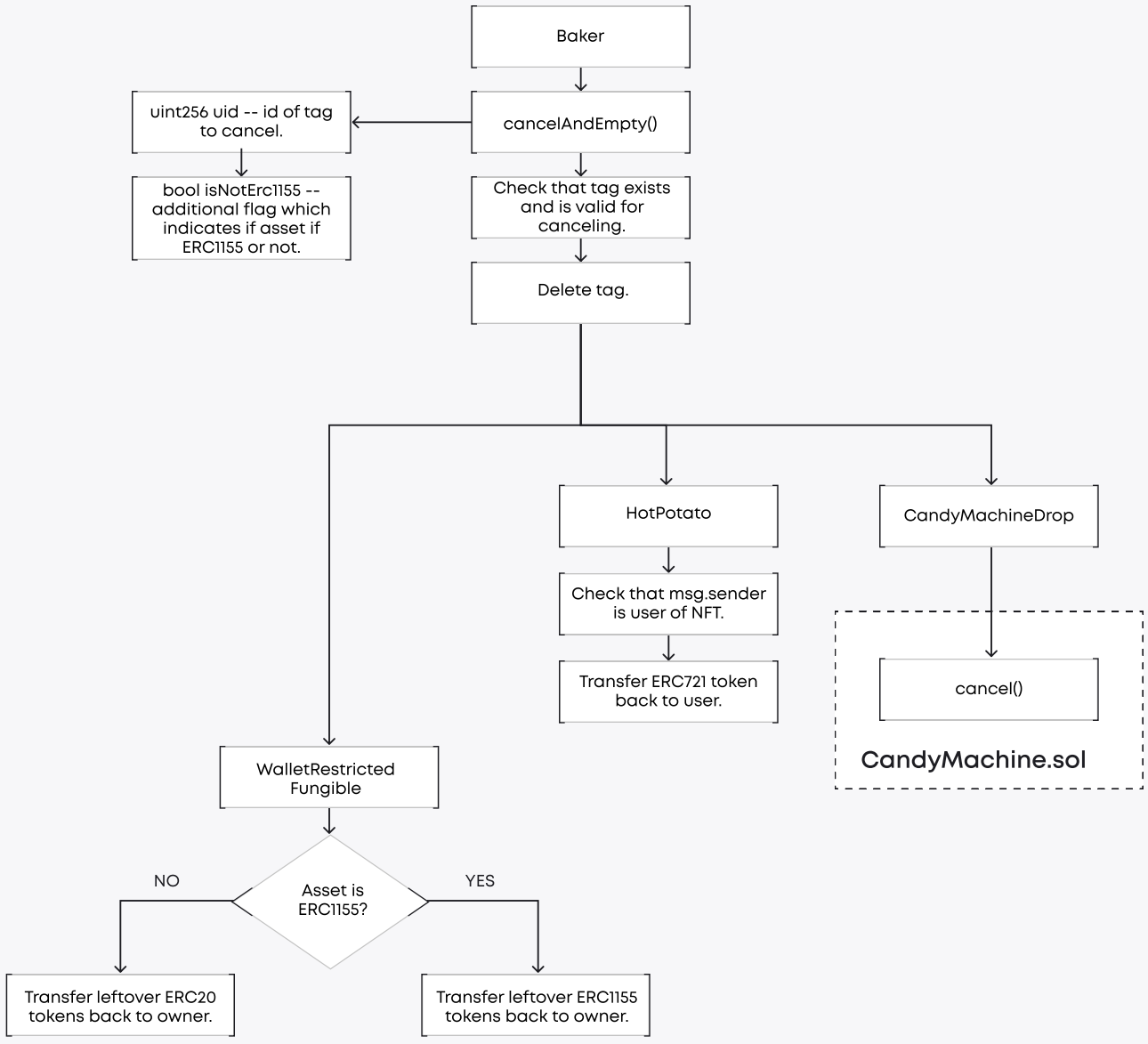
CUPCAKE PROTOCOL

Cancel tag



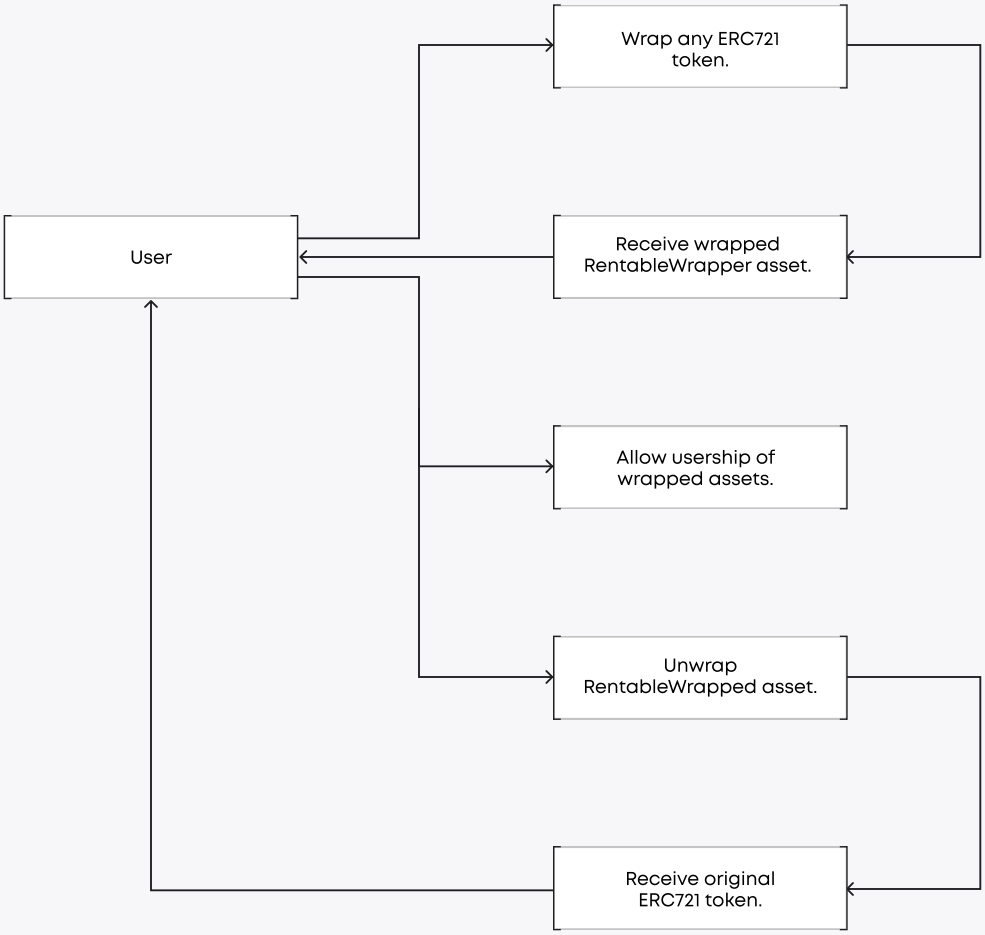
CUPCAKE PROTOCOL

Cancel tag



CUPCAKE PROTOCOL

RentableWrapper.sol



COMPLETE ANALYSIS

CRITICAL-1	✔ Resolved

Miners can manipulate the result of a random number.

CandyMachine.sol: _randomNumber().
In order to determine the ID of the minted token, a pseudo-random number is generated based on block.timestamp and a nonce.
Miners can manipulate the transaction and some of the blocks to manipulate the outcomes of the ID to mint.
The issue is described in the following article: https://betterprogramming.pub/how-to-generate-truly-random-numbers-in-solidity-and-blockchain-9ced6472dbdf

Recommendation:

Consider using off-chain oracles for generating random numbers.

Post-audit:

VRF Chainlink Oracle is now used for generating random numbers.

CRITICAL-2	✔ Resolved

Information about the token can be deleted during transferring.

RentableWrapper.sol: _beforeTokenTransfer().
During the transfer, the _beforeTokenTransfer hook is executed. This hook validates that in case the specified `tokenId` has a user and the expiry is less than block.timestamp, the whole information about the token is removed from the mapping (line 218). Due to this, the token can no longer be unwrapped after this information is deleted. Based on the validations performed in this hook, only the information connected to the user and expiry should be deleted.

Recommendation:

Delete only the information about the user and expiry.

Post-audit:
Only the information about the user and expiration is deleted now if needed.

HIGH-1	✔ Resolved

Upgradable pattern violation.

CandyMachineFactory.sol: newCandyMachine().
CandyMachine.sol inherits an initializable contract (ERC1155URIStorageUpgradeable.sol), which is most often used when a contract is supposed to be an implementation for an upgradable proxy. However, when the instance of CandyMachine.sol is deployed within the factory contract, no proxy is deployed and an implementation is initialized and used in the future. The issue is marked as high since it is unclear if CandyMachine.sol should be deployed as a proxy. Based on the code of CandyMachine.sol, it is supposed to be an upgradable proxy but it is not deployed as a proxy in the factory.

Recommendation:

Verify if CandyMachine.sol should be an upgradable proxy contract or implement it as non-upgrable contract with the constructor instead of initialized().

Post-audit: The contract was implemented as non-upgradable.

MEDIUM-1	✔ Resolved

Wrong statement is checked in the function.

RentableWrapper.sol: isWrapped(), line 145.
Based on the naming and the usage of the function isWrapped(), it should return true when the provided `wrappedTokenId`is wrapped and false otherwise. However, the current statement, which checks if the asset of a provided token is zero address, returns true when `wrappedTokenId` isn't wrapped and false otherwise. The issue is marked as medium since the function doesn't prevent any vital actions on the contracts, though it prevents the correct execution of the uri() function, which might be important for the dApp.

Recommendation:

Correct the statement.

MEDIUM-2	✔ Resolved

Transferring zero amount of ERC1155 tokens.

Contract.sol: addOrRefillTag(), line 201, 218; claimTag(), line 333; cancelAndEmpty(), line 394.
Transferring a zero amount of ERC1155 tokens is redundant since it doesn't affect the balances. Thus, in case no tokens should be transferred, the transfer can be removed. In case tokens should be transferred, the amount greater than 0 should be passed. The issue is marked as medium since in claimTag(), the recipient receives a zero amount of ERC1155 tokens, which might be potentially misleading for users who may expect to actually receive tokens.

Recommendation:

Verify if the current logic is correct. Remove the transfer or pass an amount greater than 0.

Post-audit:
An appropriate amount of tokens is transferred now.

MEDIUM-3	✔ Resolved

SafeERC20 should be used.

Contract.sol: addOrRefillTag(), line 225; claimTag(), line 365; cancelAndEmpty(), line 437.
Though a return value is validated with `require`, such an approach to validate the transfer or ERC20 token might revert in case the implementation of the token has a non-standard transfer, which doesn't return boolean values (e.g. USDT). The issue is marked as medium since the tag can be created by anyone and in case a token with a non-standard implementation is used, the function will revert.

Recommendation:

Use SafeERC20 to validate transfers.

LOW-1	✔ Resolved

The authority tag is not validated to be owner.

Contract.sol: addOrRefillTag().
Function for claiming tag contains a restriction, that msg.sender is equal to the tag's authority. However, the claimTag() function also has the onlyOwner modifier that restricts the function from being called by anyone except for the owner. Thus, in case passedTag.tagAuthority` is not equal to the owner during the execution of the addOrRefillTag() function, the tag cannot be claimed until the ownership of the contract is transferred to the tag authority. The issue is marked as low since only the owner can execute addOrRefillTag() and should validate input parameters such as `passedTag.tagAuthority.`

Recommendation.
Validate that `passedTag.tagAuthority`is equal to the owner OR validate that claimTag() should only be executed by the owner whose address is equal to tag authority.

Post-audit:
During claiming the tag, a bakery address is passed to generate the correct hash of the tag. Only tag authority can claim the tag.

LOWEST-1	✔ Resolved

Missing storage variables visibility.

CandyMachineStorage.sol: lines 17-19, variables `numURIsExisting,` `nonce,` `owner.` ContractStorage.sol: lines 60, 118, variable `candyMachineFactoryAddr.` The visibility of these storage variables is not explicitly marked. Though all variables have private visibility by default, it is recommended to explicitly mark the visibility in order to improve code readability.

Recommendation:

Mark the visibility of the storage variables.

LOWEST-2	✔ Resolved

Gas optimization suggestions.

1) CandyMachine.sol: initialize(), line 43.
Assigning zero during the initialization is redundant since all variables in Solidity are initialized with zero values by default.

2) CandyMachine.sol: mint(), line 90.
Setting the `numURIsExisting` storage variable to zero before revert() is redundant since transactions will always revert when a branch on lines 90-91 is entered. Such an operation decreases code readability (In case the statement in `if` is true and `numURIsExisting` must be set to zero, a revert should not be performed). Thus, it is recommended to remove assigning `numURIsExisting` to zero and use `require` instead of `if`.

3) Contract.sol: claimTag(); cancelAndEmpty(), lines 372-376.
The storage pointer of tags[tagHash] should be used in order to decrease gas spendings and increase code readability.

4) Contract.sol: addOrRefillTag(), line 259; claimTag(), line347; cancelAndEmpty(), 419.
The interfaces of CandyMachine and CandyMachineFactory should be imported and used instead of the contract to decrease the code size of Contract.sol and decrease gas expenditure during the deployment of the contract.

5) Contract.sol: cancelAndEmpty(), line 455.
The statement “require(totalSupply - numClaimed > 0, "tag totally depleted")” in branching for tag “WalletRestrictedFungible” can’t be FALSE as there is already a validation that checks for totalSupply to be greater than numClaimed before all branching (line 421).

LOWEST-3	✔ Resolved

Library functions are never used.

CandyMachineStorage.sol, CandyMachine.sol, RentableWrapperStorage.sol, RentableWrapper.sol, ContractStorage.sol, Contract.sol.
The usage of SafeMathUpgradeable library is declared in all the following contracts, though none of the library's functions is used within any of them.

Recommendation.
Remove an unnecessary library.

LOWEST-4	✔ Resolved

ERC1155 with zero ID are used only.

Contract.sol: addOrRefillTag(), lines 226, 236;
When the tag type is `WalletRestrictedFungible`, the ID of tokens is always used as 0. Thus, tags with any other ids of tokens can't be used.
The issue is marked as the lowest since it doesn't affect the security of the contract, though such logic should be validated.

Recommendation.
Validate that only tokens with 0 should be used when tag type is `WalletRestrictedFungible` or use a passed ID `passedTag.erc721TokenId`.

Post-audit:
Passed `erc721TokenId` is used as an ID now.

LOWEST-5	✔ Resolved

NFTs can't be rescued from the contract's balance.

Contract.sol
When a tag with the `LimitedOrOpenEdition` tag type is created, ERC721 token is transferred to the contract's balance. In case the whole supply within this tag is claimed, the original ERC721 token that was transferred during tag creation can't be transferred from the contract's balance. The issue is marked as lowest since it looks like an intended logic, though it still should be validated.

Recommendation.
Validate that ERC721 token from a tag with the `LimitedOrOpenEdition` type shouldn't be rescued from the contract's balance.

Post-audit: NFT with the `LimitedOrOpenEdition` tag can be withdrawn.

LOWEST-6	✔ Verified

Usage of an arbitrary asset.

Contract.sol
An arbitrary address of the asset is used across the functions of Contract.sol. Only the owner can execute the functions where the asset is passed and should validate its correctness and conformity to the necessary interface. For example, the safety of assets with the IERC721CopyableUpgradeable interface should be validated before creating a tag with such assets. The issue is marked as lowest since only the owner can pass the address of asset and verify its safety and conformity to the necessary interface.
From the client: According to the Cupcake team, the validation of assets will be performed in additional scripts on the dApp. Also, as a minimal measure, it is now validated that the provided asset supports the necessary interface.
Post-audit: The function for creating a tag can now be executed by any user, though the validation of the asset on the dApp should be enough to protect the platform.

LOWEST-7	✔ Resolved

Claiming tag may fail.

Contract.sol, claimTag()
When claiming the CandyMachineDrop tag, the claimTag method fails proportionately as tagsClaimed approaches totalSupply. It happens because a random generator cannot generate a valid randomNum when more mintedTokenIds[randomNum] equals TRUE. The “i” iterator becomes greater than numURIsExisting and the statement “while (mintedTokenIds[randomNum] && i < numURIsExisting)” becoms FALSE, quitting the loop and resulting in revert “candyMachine depleted” in line 92.
When claiming the last NFT from the total supply, there is about 40% possibility that the method will fail (according to the automatic test).

Recommendation.
Consider changing to a different approach where a valid URI ID is always determined in one function call.

LOWEST-8	✔ Verified

Tokens might be blocked in the contract.

Contract.sol: addOrRefillTag().
When adding the WalletRestrictedFungible tag with perClaim not being a divisor of totalSupply, some amount of tokens is going to be blocked in the Contract. For example, if we add the tag with totalSupply=10 and perClaim=6, we cannot claim the tag the second time because there are not enough tokens left in totalSupply (4 tokens) to match the claim (6 tokens). Therefore, 4 tokens are blocked in the Contract with no chance to be claimed.

Recommendation:
Add a validation for perClaim to be a divisor of totalSupply.
Post-audit: The Cupcake team provided a detailed explanation on the issue. According to the team, though funds can’t be claimed, they can still be rescued with the cancelAndEmpty() function. It was confirmed by the auditors as well.

LOWEST-9	✔ Verified

Usage of an arbitrary ERC721 token.

RentableWrapper.sol: wrap(), unwrap(). When the wrap() and unwrap() functions are executed, the user can specify the address of an ERC721 asset, including malicious implementations. Though there are validations of the ownership in lines 79, 94, 114, 129, a malicious implementation of ERC721 can still pass them. The issue is marked as high since usage of malicious assets can mislead other users of the protocol.

Recommendation.

Implement a sanitizing policy. Either use a whitelist on smart contracts, or validate the provided assets in off-chain scripts, or notify users of any unverified wrapped assets.

From the client:

According to the Cupcake team, a validation of the assets will be performed in additional scripts on the dApp.

Post-audit:

Issue was marked as lowest after the clarifications from the client's side, since it was verified that it has no impact on contract itself.

	Contract.sol
✓ Re-entrancy	Pass
✓ Access Management Hierarchy	Pass
✓ Arithmetic Over/Under Flows	Pass
✓ Delegatecall Unexpected Ether	Pass
✓ Default Public Visibility	Pass
✓ Hidden Malicious Code	Pass
✓ Entropy Illusion (Lack of Randomness)	Pass
✓ External Contract Referencing	Pass
✓ Short Address/Parameter Attack	Pass
✓ Unchecked CALL Return Values	Pass
✓ Race Conditions/Front Running	Pass
✓ General Denial Of Service (DOS)	Pass
✓ Uninitialized Storage Pointers	Pass
✓ Floating Points and Precision	Pass
✓ Tx.Origin Authentication	Pass
✓ Signatures Replay	Pass
✓ Pool Asset Security (backdoors in the underlying ERC-20)	Pass

	CandyMachine.sol
✓ Re-entrancy	Pass
✓ Access Management Hierarchy	Pass
✓ Arithmetic Over/Under Flows	Pass
✓ Delegatecall Unexpected Ether	Pass
✓ Default Public Visibility	Pass
✓ Hidden Malicious Code	Pass
✓ Entropy Illusion (Lack of Randomness)	Pass
✓ External Contract Referencing	Pass
✓ Short Address/Parameter Attack	Pass
✓ Unchecked CALL Return Values	Pass
✓ Race Conditions/Front Running	Pass
✓ General Denial Of Service (DOS)	Pass
✓ Uninitialized Storage Pointers	Pass
✓ Floating Points and Precision	Pass
✓ Tx.Origin Authentication	Pass
✓ Signatures Replay	Pass
✓ Pool Asset Security (backdoors in the underlying ERC-20)	Pass

	CandyMachineFactory.sol
✔ Re-entrancy	Pass
✔ Access Management Hierarchy	Pass
✔ Arithmetic Over/Under Flows	Pass
✔ Delegatecall Unexpected Ether	Pass
✔ Default Public Visibility	Pass
✔ Hidden Malicious Code	Pass
✔ Entropy Illusion (Lack of Randomness)	Pass
✔ External Contract Referencing	Pass
✔ Short Address/Parameter Attack	Pass
✔ Unchecked CALL Return Values	Pass
✔ Race Conditions/Front Running	Pass
✔ General Denial Of Service (DOS)	Pass
✔ Uninitialized Storage Pointers	Pass
✔ Floating Points and Precision	Pass
✔ Tx.Origin Authentication	Pass
✔ Signatures Replay	Pass
✔ Pool Asset Security (backdoors in the underlying ERC-20)	Pass

	RentableWrapper.sol
✓ Re-entrancy	Pass
✓ Access Management Hierarchy	Pass
✓ Arithmetic Over/Under Flows	Pass
✓ Delegatecall Unexpected Ether	Pass
✓ Default Public Visibility	Pass
✓ Hidden Malicious Code	Pass
✓ Entropy Illusion (Lack of Randomness)	Pass
✓ External Contract Referencing	Pass
✓ Short Address/Parameter Attack	Pass
✓ Unchecked CALL Return Values	Pass
✓ Race Conditions/Front Running	Pass
✓ General Denial Of Service (DOS)	Pass
✓ Uninitialized Storage Pointers	Pass
✓ Floating Points and Precision	Pass
✓ Tx.Origin Authentication	Pass
✓ Signatures Replay	Pass
✓ Pool Asset Security (backdoors in the underlying ERC-20)	Pass

CODE COVERAGE AND TEST RESULTS FOR ALL FILES BY THE BLAIZE.SECURITY TEAM

CandyMachine

Initialize
✓ Should work if metadata and owner passed correctly (47ms)
✓ Should revert if passed empty array for URIs
✓ Should revert is passed zero address as owner
✓ Should revert if called twice
Minting
✓ Should work if not cancelled and not finished
✓ Should revert if finished (48ms)
✓ Should revert if cancelled
✓ Should revert if called by not owner
Cancel
✓ Should work if called by owner
✓ Should work if called by owner and token already minted (38ms)
✓ Should revert if called by not owner

CandyMachineFactory

Initialize
✓ Should revert if called twice
NewCandyMachine
✓ Should work if metadata passed correctly (45ms)
✓ Should revert if passed empty array for URIs

Contract

Initialize
✓ Should revert when passing zero-address as candy factory
✓ Should revert when called twice
LimitedOrOpenEdition
✓ add tag (721) (38ms)
✓ add tag (1155)
✓ claim tag (721) (51ms)
✓ should revert when hacker tries to claim tag (721) (41ms)
✓ claim tag (1155) (48ms)
✓ refill tag (721) (87ms)
✓ refill tag (1155) (71ms)

- ✓ revert when claiming tag with newTokenId already minted (57ms)
- ✓ cancel and empty (721) (53ms)
- ✓ cancel and empty (1155) (49ms)

WalletRestrictedFungible

- ✓ add tag (20)
- ✓ add tag (1155)
- ✓ revert when adding tag which is undrained (45ms)
- ✓ revert when adding tag with fungiblePerClaim > perUser
- ✓ claim tag (20) (41ms)
- ✓ should revert when claiming more than total supply (40ms)
- ✓ claim tag (1155) (45ms)
- ✓ should revert when claiming tag which is totally deleted (38ms)
- ✓ refill tag (20) (53ms)
- ✓ refill tag (1155) (64ms)
- ✓ cancel and empty (20) (45ms)
- ✓ cancel and empty (1155) (46ms)
- ✓ should revert when cancelling tag which is totally deleted (20) (44ms)
- ✓ should revert when cancelling tag which is totally deleted (1155) (42ms)

CandyMachineDrop

- ✓ add tag
- ✓ revert when adding existing tag (43ms)
- ✓ claim tag (80ms)
- ✓ revert when claiming tag when per user was drained (73ms)
- ✓ cancel tag after claiming multiple tags (67ms)
- ✓ cancel tag after claiming multiple tags (87ms)
- ✓ claim all tags from total supply (124ms)

Checking wrong conditions

- ✓ revert when adding existing tag
- ✓ revert when adding tag which is fully undrained (48ms)
- ✓ revert when adding tag which is partly drained (44ms)
- ✓ revert when adding tag with zero total supply
- ✓ revert when adding tag with zero perUser
- ✓ revert when adding tag with perUser > totalSupply
- ✓ revert when cancelling already canceled tag (66ms)

Additional tests
- ✓ add tag of Type1 with uid=1 -> drain tag -> add tag of Type2 with uid=1 (40ms)
- ✓ add tag of Type1 with uid=1 -> add tag of Type2 with uid=1
- ✓ erc1155 token with id from passedTag should be transferred to Contract when adding tag
- ✓ minting tag where neither Contract nor tagAuthority not a owners of nft
- ✓ claim all tags from total supply (CandyMachineDrop)

SingleUse1Of1
- ✓ Should work with ERC721 if passed correct data (46ms)
- ✓ Should work with ERC1155 if passed correct data
- ✓ Shouldn't be able to create twice (45ms)
- ✓ Should revert if ERC721 token not approved
- ✓ Should revert if ERC1155 token not approved
- ✓ Should be able to cancelAndEmpty with ERC721 (45ms)
- ✓ Should be able to cancelAndEmpty with ERC1155
- ✓ Should revert if cancelling claimed tag (47ms)

Refillable1Of1
- ✓ Should work with ERC721 if passed correct data (51ms)
- ✓ Should work with ERC1155 if passed correct data
- ✓ Should be able to create twice (105ms)
- ✓ Should refill without draining
- ✓ Should be able to cancelAndEmpty with ERC721 (48ms)
- ✓ Should be able to cancelAndEmpty with ERC1155
- ✓ Should revert if cancelling claimed tag (50ms)

HotPotato
- ✓ Should work with ERC4907 if passed correct data (70ms)
- ✓ Shouldn't be able to create twice (41ms)
- ✓ Shouldn't be able to create if not approved
- ✓ Should be able to cancelAndEmpty if claimed by the Alice (53ms)
- ✓ Shouldn't be able to cancelAndEmpty if claimed (59ms)

RentableWrapper

Initialize
- ✓ Should name set and symbol
- ✓ Should revert if called twice

Wrap
- ✓ Should work if Alice owns nft, nft approved and nft address passed correctly (57ms)
- ✓ Should revert if Alice does not own nft
- ✓ Should revert if nft is not approved
- ✓ Should revert if nft address is not correct
- ✓ Should revert if nft is already wrapped (38ms)
- ✓ Should revert if nft doesn't exist

Unwrap
- ✓ Should work if Alice owns wrapped token and wNFT user is not assigned (70ms)
- ✓ Should work if Alice owns wrapped token and wNFT user is set to Alice (71ms)
- ✓ Should revert if Alice does not own wrapped token (48ms)
- ✓ Should revert if wNFT user is set to Bob (51ms)
- ✓ Should revert if wNFT does not exist

SetUser
- ✓ Alice should be able to set herself while she owns wrapped token and expires value is in the future (60ms)
- ✓ userOf should return owner of NFT before time expires (63ms)
- ✓ userOf should return zero address of NFT before time expires (65ms)
- ✓ Alice should be able to set Bob while she owns wrapped token and expires value is in the future (55ms)
- ✓ Charlie should be able to set Bob if he is approved for wrapped token and expires value is in the future (57ms)
- ✓ Should revert if expires value is in the past (55ms)
- ✓ Should revert if Alice does not own wrapped token (63ms)
- ✓ Should revert if Charlie is not approved for wrapped token (55ms)

Transfer wrapped token
- ✓ User and expires values should be reset after transfer if user expired, but asset and underlyingTokenId should remain
- ✓ Transfer shouldn't reset user and expires values if user is not expired (55ms)
- ✓ Transfer shouldn't reset user and expires values if from=to (54ms)
- ✓ Transfer shouldn't reset user and expires values if user is not set (53ms)

Getters
- ✓ isWrapped should return true if token is wrapped
- ✓ isWrapped should return false if token is not wrapped
- ✓ tokenURI should return tokenURI if token is wrapped
- ✓ tokenURI should revert if token is not wrapped
- ✓ userExpires should return 0 if user is not set (104ms)
- ✓ userExpires should return user expires if user is set (50ms)
- ✓ Supports interface ERC4907
- ✓ Supports interface IERC721MetadataUpgradeable

110 passing (10s)

TEST COVERAGE RESULTS

FILE	% STMTS	% BRANCH	% FUNCS
CandyMachine.sol	100	91.67	100
CandyMachineFactory.sol	100	50	75
Contract.sol	97.39	89.29	87.5
RentableWrapper.sol	100	83.33	91.67
All files	99.34	78.57	88.54

CODE COVERAGE AND TEST RESULTS FOR ALL FILES BY THE CUPCAKE TEAM

RentableWrapper

Initialize

- ✓ should should not revert when name and symbol are empty (212ms)
- ✓ should set owner (76ms)
- ✓ should set name (66ms)
- ✓ should set symbol (64ms)

wrap

- ✓ should revert when token is not owned by wrap executer
- ✓ should revert when token is not approved for use
- ✓ should wrap token and issue wrapped token and emit a Wrap event (4056ms)
- ✓ should revert when attempting to (recursively) wrap a wrapper token (3899ms)

unwrap

- ✓ should revert when wrapper token is not owned by unwrap executer
- ✓ should revert when owner of wrapper token is not the current user (3373ms)
- ✓ should unwrap token and take wrapped token back and emit an Unwrap event (155ms)
- ✓ should (not revert and) unwrap token and take wrapped token back and emit an Unwrap event when non-owner usership has expired (3859ms)

transferFrom

- ✓ should revert when owner of wrapper token is not the current user (2400ms)
- ✓ should allow transfer when there is no user of the token
- ✓ should allow transfer when owner of wrapper token is also the current user (383ms)

isWrapped

- ✓ should return false when no such wrapped token exists for the provided tokenId
- ✓ should return true when a wrapped token exists for the provided tokenId
- ✓ should return false when a wrapped token has been unwrapped for the provided tokenId (38ms)

tokenURI

- ✓ should return tokenURI when wrapped token has tokenURI set
- ✓ should return empty tokenURI when wrapped token has no tokenURI set

setUser
- ✓ should revert when expires is not in the future
- ✓ should revert when not called by the owner (3111ms)
- ✓ should set user correctly (257ms)
- ✓ should emit an UpdateUser event when user is set correctly (963ms)
- ✓ should set user correctly when called by other address that is approved by owner (2112ms)

userOf
- ✓ should return null address when no user is set
- ✓ should return address when user is set (2378ms)
- ✓ should return null address when user has expired (2723ms)

userExpires
- ✓ should revert when owner of wrapper token is not the current user (395ms)

CandyMachine

constructor
- ✓ should revert when metadataURIs array is empty
- ✓ should revert when ownerArg is the null address
- ✓ should populate URIs when passed (57ms)

mint
- ✓ should set unique metadata URIs for all assets that are minted (2958ms)
- ✓ should revert after all assets have already been minted (3937ms)
- ✓ should emit a TransferSingle event when an asset is minted (3918ms)
- ✓ should revert when a non-owner address calls

cancel
- ✓ should override all metadata URIs before any assets have been minted
- ✓ should override two metadata URIs when only one asset has been minted (3845ms)
- ✓ should override one metadata URIs when two assets have been minted (3967ms)
- ✓ should override zero metadata URIs when all three assets have been minted (3970ms)
- ✓ should emit a Cancellation event when the cancellation occurs (3957ms)
- ✓ should revert when a non-owner address calls

CandyMachineFactory

initialize
- ✓ should set owner (76ms)

newCandyMachine
- ✓ should create a new CandyMachine contract that is functional (7727ms)
- ✓ should revert when trying to create a CandyMachine contract with an empty array of metadata URIs
- ✓ should emit a Creation event when new newCandyMachine is created

Contract

initialize
- ✓ should revert when the null address is provided for the CandyMachineFactory address argument (41ms)
- ✓ should set the owner (41ms)

addOrRefillTag

LimitedOrOpenEdition
- ✓ should revert when totalSupply is set to zero
- ✓ should revert when perUser is set to zero
- ✓ should revert when perUser is greater than totalSupply
- ✓ should revert when tag is already in use and undrained

ERC-721
- ✓ should revert and not create ERC-721 tag when non-IERC721CopyableUpgradeable-compliant asset is passed
- ✓ should create ERC-721 tag
- ✓ should create ERC-721 tag, ignoring passed value for fungiblePerClaim (always set to 0)
- ✓ should create ERC-721 tag even when tag existed before but is now drained (72ms)

ERC-1155
- ✓ should revert and not create ERC-1155 tag when non-IERC1155CopyableUpgradeable-compliant asset is passed
- ✓ should create ERC-1155 tag
- ✓ should create ERC-1155 tag, ignoring passed value for fungiblePerClaim (always set to 0)
- ✓ should create ERC-1155 tag even when tag existed before but is now drained (74ms)

SingleUse1Of1
- ✓ should revert when tag has been used / uncleared

ERC-721
- ✓ should revert and not create ERC-721 tag when non-IERC721-compliant asset is passed

- ✓ should revert if tag ever existed (50ms)
- ✓ should create ERC-721 tag
- ✓ should create ERC-721 tag, ignoring passed values for totalSupply, perUser, and fungiblePerClaim

ERC-1155

- ✓ should revert and not create ERC-1155 tag when non-IERC1155-compliant asset is passed
- ✓ should revert if tag ever existed (48ms)
- ✓ should create ERC-1155 tag
- ✓ should create ERC-1155 tag, ignoring passed value for totalSupply, perUser, and fungiblePerClaim

Refillable1Of1

- ✓ should revert when tag has been used / uncleared

ERC-721

- ✓ should revert and not create ERC-721 tag when non-IERC721-compliant asset is passed
- ✓ should create ERC-721 tag
- ✓ should create ERC-721 tag, ignoring passed values for totalSupply, perUser, and fungiblePerClaim
- ✓ should create / refill tag if it existed before but has already been claimed (71ms)

ERC-1155

- ✓ should revert and not create ERC-1155 tag when non-IERC1155-compliant asset is passed
- ✓ should create ERC-1155 tag
- ✓ should create ERC-1155 tag, ignoring passed value for totalSupply, perUser, and fungiblePerClaim
- ✓ should create / refill tag if it existed before but has already been claimed (73ms)

WalletRestrictedFungible

- ✓ should revert when tag is already in use and unused (not even partially drained)
- ✓ should revert when tag is already in use and partially drained (39ms)
- ✓ should revert when fungiblePerClaim is greater than perUser
- ✓ should revert when totalSupply is set to zero
- ✓ should revert when fungiblePerClaim is set to zero

ERC-20
- ✓ should revert and not create ERC-20 tag when non-IERC20-compliant asset is passed
- ✓ should create ERC-20 tag
- ✓ should create / refill tag if it existed before but has already been totally claimed (67ms)

ERC-1155
- ✓ should revert and not create ERC-1155 tag when non-IERC1155-compliant asset is passed
- ✓ should create ERC-1155 tag
- ✓ should create / refill tag if it existed before but has already been totally claimed (82ms)

HotPotato
- ✓ should revert when tag is currently in use
- ✓ should revert and not create tag when non-IERC4907-compliant asset is passed
- ✓ should revert when tag already existed and has even been claimed (52ms)
- ✓ should create tag
- ✓ should create tag, ignoring passed values for totalSupply, perUser, and fungiblePerClaim

CandyMachineDrop
- ✓ should revert when perUser is set to zero
- ✓ should revert when perUser is greater than the number of metadataURIs provided
- ✓ should revert when tag is currently in use
- ✓ should revert when tag already existed and has even been claimed (5867ms)
- ✓ should create tag (3540ms)
- ✓ should create tag, ignoring passed values for assetAddress, erc721TokenId, totalSupply, and fungiblePerClaim (3615ms)

claimTag

LimitedOrOpenEdition
- ✓ should revert when totalSupply is zero
- ✓ should revert when non-tagAuthority signer sends transaction
- ✓ should revert when the numClaimed equals totalSupply (51ms)
- ✓ should revert when the claimsMade for a specified recipient equals perUser
- ✓ should revert when newTokenId equals existing tokenId

ERC-721
✓ should claim tag
ERC-1155
✓ should claim tag
SingleUse1Of1
✓ should revert when totalSupply is zero
✓ should revert when non-tagAuthority signer sends transaction
✓ should revert when the numClaimed equals totalSupply / when the claimsMade for a specified recipient equals perUser
ERC-721
✓ should claim tag
ERC-1155
✓ should claim tag
Refillable1Of1
✓ should revert when non-tagAuthority signer sends transaction
ERC-721
✓ should revert if tag has been claimed and not refilled yet (38ms)
✓ should claim tag
✓ should create refill tag after a claim (60ms)
ERC-1155
✓ should revert if tag has been claimed and not refilled yet
✓ should claim tag
✓ should create refill tag after a claim (70ms)
WalletRestrictedFungible
✓ should revert when totalSupply is zero
✓ should revert when non-tagAuthority signer sends transaction
✓ should revert when the numClaimed equals totalSupply (66ms)
✓ should revert when the claimsMade for a specified recipient equals perUser
ERC-20
✓ should claim tag
ERC-1155
✓ should claim tag
HotPotato
✓ should revert when non-tagAuthority signer sends transaction
✓ should claim tag
CandyMachineDrop
✓ should revert when non-tagAuthority signer sends transaction

✓ should revert if tag has been depleted (12012ms)
✓ should claim tag (4017ms)
cancelAndEmpty
LimitedOrOpenEdition
✓ should revert when totalSupply is zero
✓ should reset claimsMade for the tag
ERC-721
✓ should cancel and empty tag
ERC-1155
✓ should cancel and empty tag
SingleUse1Of1
✓ should revert when totalSupply is zero
✓ should revert when the numClaimed equals totalSupply
ERC-721
✓ should cancel and empty tag
ERC-1155
✓ should cancel and empty tag
Refillable1Of1
✓ should revert when totalSupply is zero
✓ should revert when the numClaimed equals totalSupply
✓ should reset claimsMade for the tag (52ms)
ERC-721
✓ should cancel and empty tag
ERC-1155
✓ should cancel and empty tag
WalletRestrictedFungible
✓ should revert when totalSupply is zero
✓ should revert when the numClaimed equals totalSupply (43ms)
✓ should reset claimsMade for the tag
ERC-20
✓ should cancel and empty tag
ERC-1155
✓ should cancel and empty tag
HotPotato
✓ should revert when totalSupply is zero
✓ should reset claimsMade for the tag
✓ should revert when bakery is not current user

✓ should cancel and empty tag
CandyMachineDrop
✓ should revert when totalSupply is zero
✓ should revert when the numClaimed equals totalSupply (8009ms)
✓ should reset claimsMade for the tag (4032ms)
✓ should cancel and empty tag
getClaimsMade
✓ should return claimsMade properly based on invalidation mechanism (60ms)

157 passing (3m)

TEST COVERAGE RESULTS

FILE	% STMTS	% BRANCH	% FUNCS
CandyMachine.sol	100	91.67	100
CandyMachineFactory.sol	100	25	75
Contract.sol	100	93.45	87.5
RentableWrapper.sol	94.59	69.44	83.33
All files	98.64	54.44	88.95

DISCLAIMER

The information presented in this report is an intellectual property of the customer, including all the presented documentation, code databases, labels, titles, ways of usage, as well as the information about potential vulnerabilities and methods of their exploitation. This audit report does not give any warranties on the absolute security of the code. Blaize.Security is not responsible for how you use this product and does not constitute any investment advice.

Blaize.Security does not provide any warranty that the working product will be compatible with any software, system, protocol, or service and operate without interruption. We do not claim the investigated product is able to meet your or anyone else's requirements and be fully secure, complete, accurate, and free of any errors and code inconsistency.

We are not responsible for all subsequent changes, deletions, and relocations of the code within the contracts that are the subjects of this report.

You should perceive Blaize.Security as a tool that helps investigate and detect any weaknesses and vulnerable parts that may accelerate the technology improvements and faster error elimination.