Blaize.Security

January 24th, 2024 / V. 1.0

AVAX APES

SMART CONTRACT AUDIT

TABLE OF CONTENTS

AUDIT RATING

SCORE **9.9** /10

The scope of the project includes Avax Apes smart contracts:

wAvaxApes.sol

Repository: https://github.com/apesavax/wAvaxApes

Branch: main

Initial commit:

- acf863f642a13957a73f2867f9357ec6280a4ca8

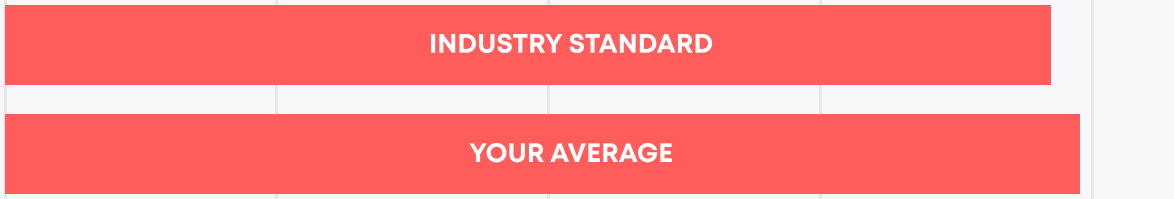
Final commit:

- 0a604358ce50beba4f76a9715528894345c19f45

TECHNICAL SUMMARY

During the audit, we examined the security of smart contracts for the Alfred 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 **Avax Apes** smart contracts conducted between **January 19th, 2024** and **January 24th, 2024.**

Testable code

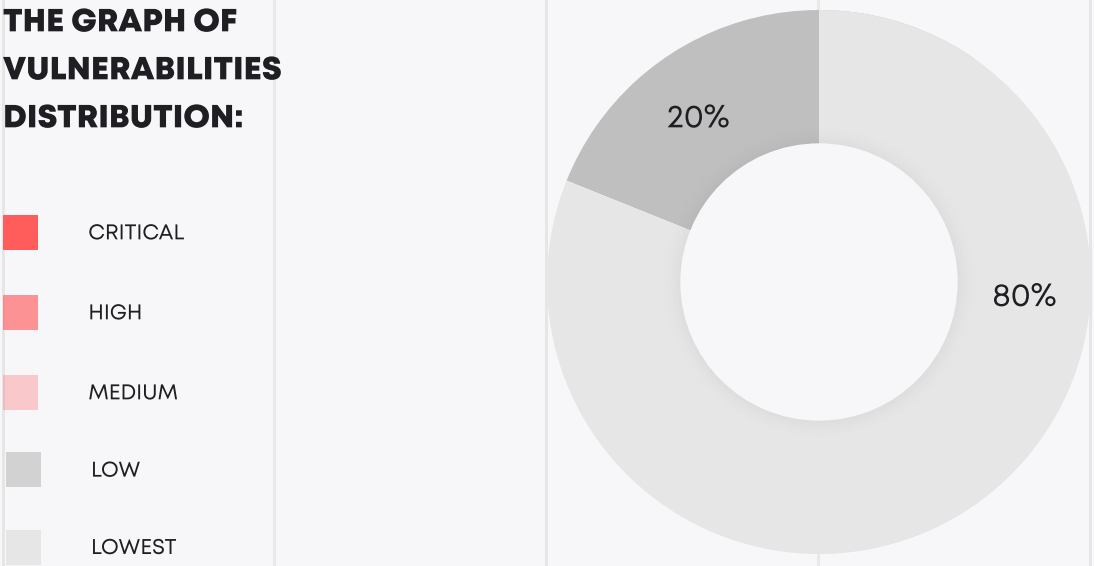


Auditors approved code as testable within the industry standard.

The audit scope includes all tests and scripts, documentation, and requirements presented by the **Avax Apes** team. The coverage is calculated based on the set of Hardhat framework tests and scripts from additional testing strategies, and includes testable code from manual and exploratory rounds.
However, to ensure the security of the contract, the **Blaize.Security** team suggests that the **Avax Apes** team follow post-audit steps:

1. launch **active protection** over the deployed contracts to have a system of early detection and alerts for malicious activity. We recommend the AI-powered threat prevention platform **VigiLens**, by the **CyVers** team.
2. 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 5 problems were found. All of the issues were fixed by the Avax Apes team.

	FOUND	FIXED/VERIFIED
Critical	0	0
High	0	0
Medium	0	0
Low	1	1
Lowest	4	4

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, yet is relevant for best practices

AUDITING STRATEGY AND TECHNIQUES APPLIED/PROCEDURE

Blaize.Security auditors start the audit by developing an **auditing strategy** - an individual plan where the team plans methods, techniques, approaches for the audited components. That includes a list of activities:

Manual audit stage

- Manual line-by-line code by at least 2 security auditors with crosschecks and validation from the security lead;
- Protocol decomposition and components analysis with building an interaction scheme, depicting internal flows between the components and sequence diagrams;
- Business logic inspection for potential loopholes, deadlocks, backdoors;
- Math operations and calculations analysis, formula modeling;
- Access control review, roles structure, analysis of user and admin capabilities and behavior;
- Review of dependencies, 3rd parties, and integrations;
- Review with automated tools and static analysis;
- Vulnerabilities analysis against several checklists, including internal Blaize.Security checklist;
- Storage usage review;
- Gas (or tx weight or cross-contract calls or another analog) optimization;
- Code quality, documentation, and consistency review.

For advanced components:

- Cryptographical elements and keys storage/usage audit (if applicable);
- Review against OWASP recommendations (if applicable);
- Blockchain interacting components and transactions flow (if applicable);
- Review against CCSSA (C4) checklist and recommendations (if applicable);

Testing stage:

- Development of edge cases based on manual stage results for false positives validation;
- Integration tests for checking connections with 3rd parties;
- Manual exploratory tests over the locally deployed protocol;
- Checking the existing set of tests and performing additional unit testing;
- Fuzzy and mutation tests (by request or necessity);
- End-to-end testing of complex systems;

In case of any issues found during audit activities, the team provides detailed recommendations for all findings.

EXECUTIVE SUMMARY

Blaize Security team has conducted an audit for the Avax Apes NFT protocol. The protocol consists of a single contract, wAvaxApes. The contract is designed as a wrapper around the OG Apes collection and allows users to wrap or unwrap their NFTs of the OG collection.

During the audit, auditors reviewed all the aspects of the contract, specifically:

- Check that the contract corresponds to the standard ERC-721 functionality.
- Check that the contract implements the ERC2981 royalty interface correctly.
- Check the compatibility with the OG smart contract and wrap/unwrap functionality.

Auditors have found 1 low-risk and 4 lowest-risk issues during the manual audit. The low-risk issue describes the possible lock up of OG or Wrapped Avax Apes NFTs on the contract's balance in case of accidentally sending tokens directly to the contract. The Avax Apes team successfully fixed the issue by implementing functions for manually wrapping and unwrapping stuck tokens. Other issues were connected to the usage of constants, redundant assignments of value during deployment, the ability for the owner to update the base URI of the collection, and possible lock of NFTs from other collections accidentally sent to the contract's balance. The Avax Apes team successfully fixed or verified all of the discovered issues. Auditors have verified the safety of the contract's implementation and assets stored on the contract's balance. No critical issues were found during the audit.

Blaize Security team has also prepared a suite of tests to validate the correctness of the contract's code. All the core functions of the wAvaxApes, along with the basic ERC-721 functionality, were carefully checked. Additionally, auditors have checked the following scenarios to detect if any possible security issues are possible:

- Pass repeatable IDs of tokens in the array or IDs of tokens not owned by the user to check for the possibility of double-spending of assets.
- Check the correctness of manual wrap/unwrap (owner functionality) and the flow for an owner to use it only for stuck tokens.
- Check compatibility with OG and ability to receive royalty share of AVAX from OG collection during unwrapping.

The contract has successfully passed all the security tests.

The total security of the smart contract is high enough. The contract is well-written and has sufficient quality of natSpec documentation. The smart contract's total score after applying the fixes is 9.9 out of 10, with the mark slightly decreased for certain redundancy of the implemented code and the absence of native tests.

	RATING
Security	10
Logic optimization	10
Code quality	9.9
Test coverage*	9.9
Total	9.9

Note* : there were no native tests available in the protocol, testable code was fully checked by auditors during the testing stage.

PROTOCOL OVERVIEW

Roles & Responsibilities

1. Owner

The owner is maintained via an Ownable Smart contract by OpenZeppelin. Aside from the basic owner logic of Ownable (transfer and renounce of ownership), the owner can:

- Flip the pause state of the smart contract.
- Set royalty BPS.
- Set royalty address.
- Update the Base URI of the collection.
- Mint the remainder of 5000 NFTs, capping the total supply to 10000 NFTs.
- Withdraw any ERC-20 tokens stuck on the contract.
- Withdraw AVAX sent to the contract (for example, via Marketplaces).

2. User

Users can interact with the contract, specifically by calling the wrap() and unwrap() functions. By calling the wrap function, a user has to approve the OG NFTs that he is going to wrap or set approval for all to the wAvaxApes smart contract and then pass the IDs of the NFTs as an array parameter. By calling the unwrap function, the user has to pass the IDs of the NFTs as an array parameter. The NFTs must be stored on the user's balance during the operations.

Settings

1. Pause.

The `pause` setting affects users' ability to wrap NFTs. It should be noted that it doesn't affect the unwrap function, which always allows users to claim their OG NFTs back. Only the owner can change the state of the `pause` setting.

2. OG address.

OG address is the address of the OG collection, which users can wrap into the Wrapped Avax Apes NFT collection. It is set only once during deployment and can't be changed later.

3. Base URI.

Base URI is the base link to the IPFS where the image for the Avax Apes collection is stored. URI for each NFT can be queried with the tokenURI() function, passing the ID of the token as a parameter. The Base URI and the token ID are then concatenated, and the function returns the URI of the specific token of the Avax Apes collection.

4. Royalty BPS and Address.

These settings are part of the ERC2981 royalty standard. Usually, these settings are used by other protocols (e.g., Marketplaces) to query the royalty information. The Royalty BPS parameter reflects the % of the price for which the token is traded, and the Royalty Address reflects the address (usually, the address of the artist), which will receive the % of the price. It is set initially during deployment as follows:

- Royalty BPS is set to 5%.
- Royalty Address is set to the 0x5BcD0455E8b83A2d3Ad3E1e8Ca635779D7d5a253.

Furthermore, the owner is able to update these parameters:

- Royalty BPS can be set to any value less than 10%.
- Royalty address can be set to any address except zero address.

Deployment script

The deployment script is located in the ./script/deploy.js.
During deployment, 5000 tokens are minted to the wAvaxApes contract's balance. The name of the collection is set to the "wAvax Apes." The symbol of the collection is set to the "WAVAXAPE."

List of valuable assets

1. Wrapped Avax Apes NFTs.

After minting, Wrapped Avax Apes NFTs are initially stored on the contract's balance. They transferred from the balance during the wrap() in exchange for OG NFTs. They are transferred back to the contract's balance from the user when the user wants to receive the OG NFT back.

The IDs of both collections are equal. For example, for OG NFT with id 1, the user will receive the Wrapped Avax Apes NFT with id 1 as well and vice versa. Thanks to these, access to the NFTs is only possible if user has a corresponding token of OG collection.

2. OG NFTs.

OG NFTs appear on the contract's balance during the wrap function() and are stored until users who possess the NFTs of Wrapped Avax Apes with corresponding IDs decide to unwrap them and receive OG NFTs back.

3. ERC-20 tokens.

Though the contract is not designed to interact with the ERC-20 tokens, the failsafe mechanism is implemented for the owner to be able to withdraw accidentally stuck tokens.

4. AVAX.

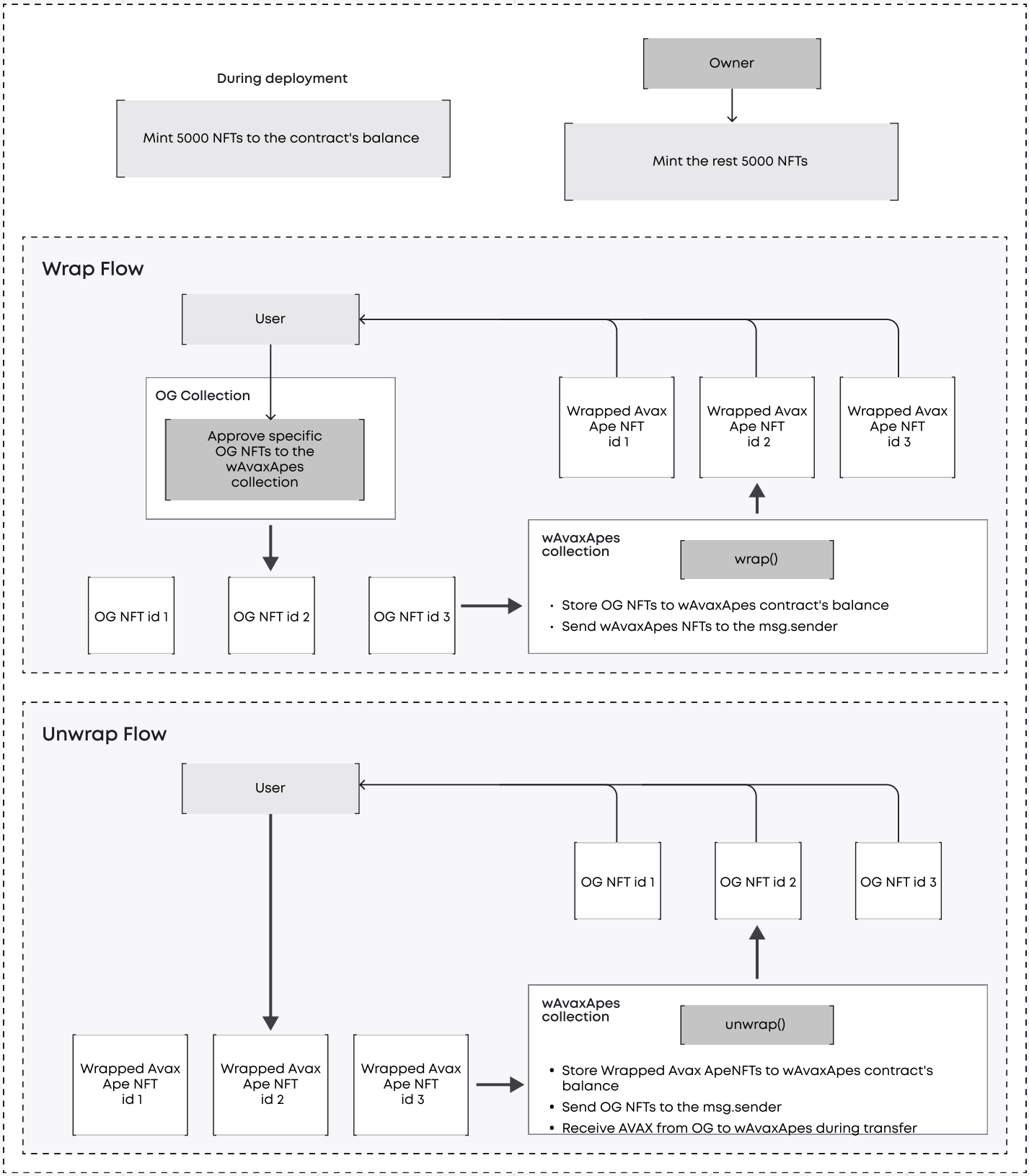
The contract is designed to receive tokens via the receive() function. According to natSpec, AVAX might be sent to the contract from the internal marketplace. The owner is then able to withdraw AVAX to his balance. Furthermore, auditors have tested that the contract is able to receive AVAX from the OG collection during unwrapping.

Description

wAvaxApes smart contract is an ERC-721 collection. It inherits the standard ERC721AQueryable, Ownable, and ReentrancyGuard. It also implements custom pause functionality and the ERC2981 royalty standard (where the main function, royaltyInfo(), corresponds to the standard and the implementation by OpenZeppelin).

The Wrapped Avax Apes collection allows the owners of the OG collection to wrap their tokens into tokens of the Wrapped Avax collection. Each OG collection token corresponds to the Wrapped Avax Apes collection tokens. For example, the OG token with ID 1 corresponds to the Wrapped Avax Apes token with ID 1. Users are able to wrap tokens only when the contract is not paused. Users can unwrap Avax Apes and receive OG tokens back at any time. The total supply of the collection is 10000. The first 5000 are minted during deployment to the balance of the Wrapped Avax Ape collection contract. The rest 5000 can be minted once by the owner.

AVAXAPEX


During deployment
Mint 5000 NFTs to the contract's balance
Owner
Mint the rest 5000 NFTs
Wrap Flow
User
OG Collection
Approve specific OG NFTs to the wAvaxApes collection
Wrapped Avax Ape NFT id 1
Wrapped Avax Ape NFT id 2
Wrapped Avax Ape NFT id 3
OG NFT id 1
OG NFT id 2
OG NFT id 3
wAvaxApes collection
wrap()
• Store OG NFTs to wAvaxApes contract's balance
• Send wAvaxApes NFTs to the msg.sender
Unwrap Flow
User
OG NFT id 1
OG NFT id 2
OG NFT id 3
Wrapped Avax Ape NFT id 1
Wrapped Avax Ape NFT id 2
Wrapped Avax Ape NFT id 3
wAvaxApes collection
unwrap()
• Store Wrapped Avax ApeNFTs to wAvaxApes contract's balance
• Send OG NFTs to the msg.sender
• Receive AVAX from OG to wAvaxApes during transfer

COMPLETE ANALYSIS

LOW-1	✔ Resolved

OG or Wrapped Avax Apes tokens may stuck on the contract's balance.

OG tokens and Wrapped Avax Apes may accidentally be sent to the contract's balance directly. In this case, the tokens will be stuck without the ability to retrieve them.

Recommendation:

Consider adding a failsafe mechanism to retrieve accidentally sent OG or Wrapped Avax Apes tokens OR implement a mechanism where, in case tokens are sent directly to the contract's balance, they are still wrapped or unwrapped. Otherwise, verify if this is not an issue for the protocol and notify users not to send their NFTs directly.

Post-audit.
The Avax Apes team has added functions for manually withdrawing accidentally stuck tokens.

LOWEST-1	✔ Resolved

Variable should be marked as constant.
wAvaxApes.sol: line 46, variable `OG`.
The OG variable is set only once during deployment and never changed. Thus, such a variable can be marked as constant.

Recommendation:

Mark variable `OG`as constant.

LOWEST-2	✔ Resolved

Redundant assignment of value during deployment.
wAvaxApes.sol: line 44, variable `paused`.
The value of the boolean variable is assigned to false during deployment. However, since all variables are assigned to initial values, assigning it explicitly to `false` is redundant and may increase gas spending during deployment.

Recommendation:
Remove assignment of `false` to variable.

LOWEST-3	✔ Verified

Owner is able to update base URI.
wAvaxApes.sol: updateBaseURI().
The base URI is the core value of NFTs, which references the images of tokens stored in the decentralized storage. Since users usually pay to own the image, the base URI remains immutable to ensure that users won't lose ownership of the image. However, the function updateBaseURI() allows the contract owner to change this value, potentially changing the images owned by users. The issue is marked as info since the contract represents the wrapped collection, which doesn't change the URI of the OG collection.

Recommendation:
Verify if the functionality of updating the URI is necessary for the collection. Notify users in advance if the URI is going to be changed.

From client.
The Avax Apes team has verified that the functionality should remain when IPFS data is corrupted or lost, and the URI should be updated.

LOWEST-4	✔ Resolved

Tokens of other NFT collections may be transferred to the contract's balance.

The contract overrides the onERC721Received hook to check the collection of transferred tokens. The transfer will be reverted if the collection is not OG or Wrapped Avax Apes. However, such measures still don't prevent sending tokens of other collections since the hook is only invoked when the safeTransferFrom() function is used. Actors may still use transferFrom instead and be able to send any tokens to the contract, which makes the measures implemented in onERC721Received redundant.

Recommendation:

Verify if it is crucial for the contract not to receive NFTs from other collections. Remove the validation from onERC721Received, since users may still transfer NFTs to the contract's balance.

Post-audit.

The Avax Apes team has decided to leave the validation as the safety measure. Additionally, emergencyWithdrawERC721() function was added, allowing the owner to withdraw any NFT (except OG and Wrapped Avax Apes collections) from the contract's balance.

STANDARD CHECKLIST

	wAvaxApes.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, PREPARED BY BLAIZE SECURITY TEAM

Scenarios

Scenario 1

- ✓ The user passes the same ID in the array to the wrap() function (1119ms)
- ✓ The user passes the same ID in the array to the unwrap() function (1350ms)

Scenario 2

- ✓ The user transfer token to the contract OG and wAvaxApes (1201ms)

Scenario 3

- ✓ The user tries to wrap & unwrap tokens which not owned by him (1121ms)

Scenario 4

- ✓ The user wrap NFT -> transfer wNFT to wAvaxApes -> unwrap wNFT (527ms)

Scenario 5

- ✓ The contract has NFTs from OG and wAvax Apes with the same id -> owner call manualWrap -> check manualUnwrap not allowed after (1038ms)
- ✓ The contract has NFTs from OG and Avax Apes with the same id -> owner call manualUnwrap -> check manualWrap not allowed after (572ms)

Scenario 6

- ✓ After wrap of NFT, wAvaxApes should be rewarded as the owner of the NFT (571ms)

wAvaxApes

Mint remaining tokens

- ✓ Should revert when trying to mint more than 10_000
- ✓ Should revert if caller is not owner

Wrap

- ✓ Should allow to wrap OG Apes (1328ms)
- ✓ Should revert when paused

Unwrap

- ✓ Should allow to unwrap OG Apes (1362ms)
- ✓ Should revert if caller of copy is not owner (1297ms)

Manual Wrap
✓ Should allow to manual wrap (516ms)
✓ Should revert if caller is not owner
✓ Should revert if token is not deposited
Manual Unwrap
✓ Should allow to manual unwrap (548ms)
✓ Should revert if caller is not owner
✓ Should revert if token is not deposited (512ms)
Pause control
✓ Should allow to pause
✓ Should allow to unpause
✓ Should revert if caller is not owner
Royalty BPS
✓ Should allow to set new royalty bps
✓ Should revert if caller is not owner
✓ Should revert if new royalty bps is exceeding max royalty bps
Royalty address
✓ Should allow to set new royalty address
✓ Should revert if caller is not owner
✓ Should revert if new royalty address is zero address
Royalty info
✓ Should allow to get royalty info
Emergency withdraw ERC20
✓ Should allow to emergency withdraw ERC20
✓ Should revert if caller is not owner
Emergency withdraw AVAX
✓ Should allow to emergency withdraw AVAX
✓ Should revert if caller is not owner
Emergency withdraw ERC721
✓ Should allow to emergency withdraw ERC721 (46ms)
✓ Should revert if collection contract is OG or wAvaxApes (39ms)
✓ Should revert if caller is not owner
List of held tokens
✓ Should allow to get list of held tokens (1007ms)
TokenURI getter
✓ Should allow to get tokenURI (524ms)

```
✓ Should revert if token does not exist
✓ Should return empty string if baseURI is not set
  # BaseURI setter
✓ Should allow to set baseURI
✓ Should revert if caller is not owner
  # Interface supporter (ERC165)
✓ Should return false if interface is not supported
  # ERC721Receiver
✓ Should revert if sender is not acceptable

  45 passing (18s)
```

TEST COVERAGE RESULTS

FILE	% STMTS	% BRANCH	% FUNCS
wAvaxApes.sol	97.83	84.62	100

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, which helps to investigate and detect the weaknesses and vulnerable parts that may accelerate the technology improvements and faster error elimination.