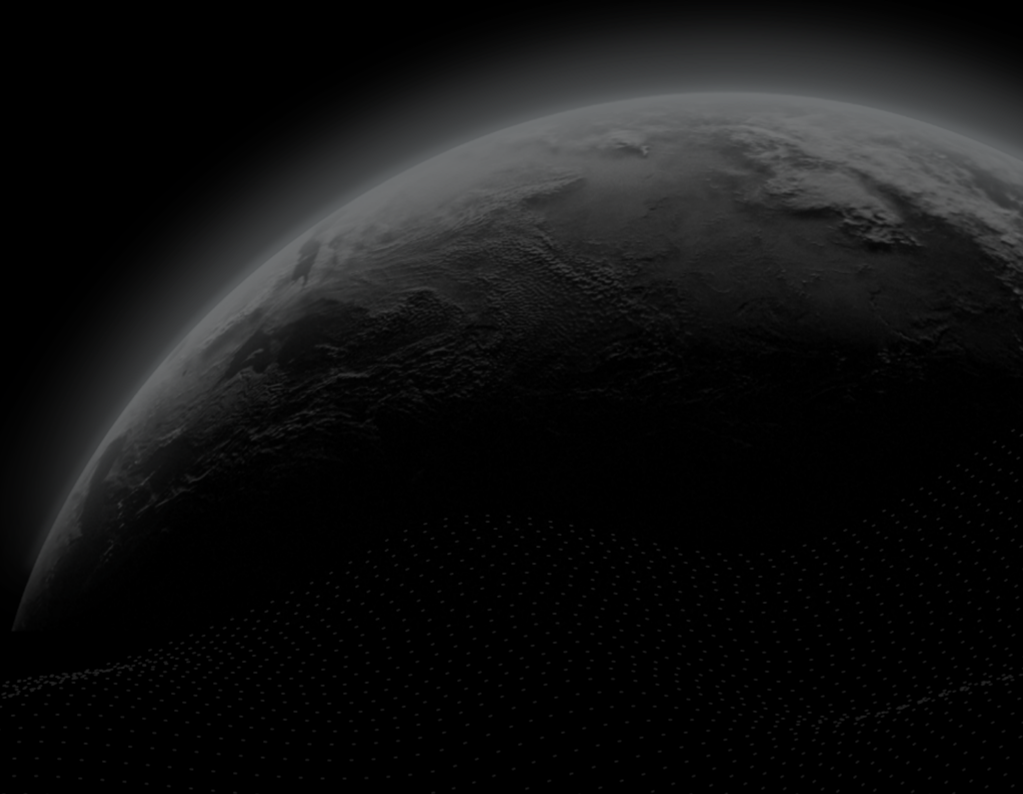
CERTIK

Security Assessment

Webacy (Audit)

CertiK Verified on Apr 3rd, 2023



CertiK Verified on Apr 3rd, 2023

Webacy (Audit)

The security assessment was prepared by CertiK, the leader in Web3.0 security.

Executive Summary

TYPES	ECOSYSTEM	METHODS
Service	Ethereum (ETH)	Formal Verification, Manual Review, Static Analysis

LANGUAGE	TIMELINE	KEY COMPONENTS
Solidity	Delivered on 04/03/2023	N/A

CODEBASE	COMMITS
https://github.com/Webacy-Prod/mega-contracts ...View All	c0c2768c3dd1e802a33ccba4bfc4f8cdfa4649c5 ...View All

Vulnerability Summary

20	17	2	0	1	0	0
Total Findings	Resolved	Mitigated	Partially Resolved	Acknowledged	Declined	Unresolved

🟥	0	Critical		Critical risks are those that impact the safe functioning of a platform and must be addressed before launch. Users should not invest in any project with outstanding critical risks.
🟧	4	Major	1 Resolved, 2 Mitigated, 1 Acknowledged	Major risks can include centralization issues and logical errors. Under specific circumstances, these major risks can lead to loss of funds and/or control of the project.
🟨	6	Medium	6 Resolved	Medium risks may not pose a direct risk to users' funds, but they can affect the overall functioning of a platform.
🟨	4	Minor	4 Resolved	Minor risks can be any of the above, but on a smaller scale. They generally do not compromise the overall integrity of the project, but they may be less efficient than other solutions.
🟦	6	Informational	6 Resolved	Informational errors are often recommendations to improve the style of the code or certain operations to fall within industry best practices. They usually do not affect the overall functioning of the code.

TABLE OF CONTENTS | WEBACY (AUDIT)

Summary

Executive Summary

Vulnerability Summary

Codebase

Audit Scope

Approach & Methods

Findings

GLOBAL-01 : Centralization Risks

WCP-01 : Centralized Control of Contract Upgrade

WCP-02 : Function Calls User-Provided Addresses With No Access Control Modifier

WCP-03 : Dangerous Usage of `tx.origin`

ASW-01 : Missing checks on provided `uid`

ASW-02 : Missing checks on `claimExpiryTime`

MWC-01 : Incorrect Fee Handling

TAW-01 : Ineffective Balance Check

WCP-04 : `Ischarity` not checked when calling `transferUnclaimedAssets`

WCP-05 : Incorrect Allowance Check

ASW-03 : Missing Input Validation

MWC-03 : Missing Blacklist Address Check

WCP-06 : Missing checks on approved token amount

WCP-07 : Check Effect Interaction Pattern Violated

ASW-04 : Discussion on function `transferUnclaimedAssets`

ASW-05 : Incorrect Claimable Assets Calculation

GLOBAL-02 : Usage of transfer pool

GLOBAL-03 : Lack of Unit-test File

WCP-08 : Questionable Implementation of Function `checkIfUIDExists`

WCP-09 : Redundant Checks

Formal Verification

Considered Functions And Scope

Verification Results

Appendix

Disclaimer

CODEBASE | WEBACY (AUDIT)

Repository

https://github.com/Webacy-Prod/mega-contracts

Commit

c0c2768c3dd1e802a33ccba4bfc4f8cdfa4649c5

AUDIT SCOPE | WEBACY (AUDIT)

19 files audited ● 2 files with Acknowledged findings ● 3 files with Mitigated findings ● 1 file with Resolved findings ● 13 files without findings

ID	File	SHA256 Checksum
● ASW	projects/Webacy/contracts/AssetsStore.sol	eac945dec58c890f3b5f3d01a602ff22e7706ea9c56d0fd15b8ded7e9291cae7
● MWC	projects/Webacy/contracts/Member.sol	f2d90b275aa6a9b3df88b438b321499bfa12af66a2ef7eb75bc4c0a065bb45cb
● ASF	projects/Webacy/contracts/factories/AssetStoreFactory.sol	c29d14831ee4d957e6986325cedf965fc3d59f4f6d2737e8cdcd479234066d08
● PDW	projects/Webacy/contracts/ProtocolDirectory.sol	6413f1f855b00adea1ca27e03b401d957341668e879f4923c7e8a74459072689
● RCW	projects/Webacy/contracts/RelayerContract.sol	359f3f9b63f2818f72aadcbd611e9bf5c850bd1efaa35271f8a0bbc473fe8077
● TAW	projects/Webacy/contracts/libraries/TokenActions.sol	16390211a16ee3fc6f25544e98c0b20188822b53f32c3fdfafd4181d1c5d106b
● IAS	projects/Webacy/contracts/interfaces/IAssetStore.sol	46615376d4aaf5c0930c55f1181b0e3e6af7e468fbe6440f04981f9b142fd880
● IAF	projects/Webacy/contracts/interfaces/IAssetStoreFactory.sol	6c6247e4cbb800caed7c069eccd4be5b764a0adc1d9bc914bcaeea7ef98e7379
● IMW	projects/Webacy/contracts/interfaces/IMember.sol	2400d06918f78690e52fd7c6a4e712e7a37c9b1d78e7dede46c5215a6d46ef76
● IPD	projects/Webacy/contracts/interfaces/IProtocolDirectory.sol	ebbee951f3b1c3a6e602a595c1f2b063d0ec378768cecc860fbd0471c0509f82
● ASC	projects/Webacy/contracts/structs/ApprovalsStruct.sol	1eb4210ac664eead9750f78abc76f9258145f5e0353f19aff26762b8f5de342b
● BAS	projects/Webacy/contracts/structs/BackupApprovalStruct.sol	15589b8892dbe71f597fcb9755853c548cec901181ae135b6adc427967458c25
● BSW	projects/Webacy/contracts/structs/BeneficiaryStruct.sol	4ab8307375271b2b2743f962a8edf6ef439075185ce8e5501456e6abbbf8e6a7

ID	File	SHA256 Checksum
MSW	projects/Webacy/contracts/structs/MemberStruct.sol	f40fbad0062342866e030c5fa2f361aadfcea0a752febbf4b43f40b37f8fcc32
TSW	projects/Webacy/contracts/structs/TokenStruct.sol	02bff0a6b895f84576819b391aa2901932b08c1d152bbcb39c5d2c9fc09bedf4
ERC	projects/Webacy/contracts/utils/ERC1155.sol	b87641627e17432bcad3a6f494a46ffab01bca0c01092687a862d094dcd08659
ERW	projects/Webacy/contracts/utils/ERC20.sol	e4a33145b286445902263c3e47fc9b9745f1f9840f89d2932be6cff08c2ae88e
ERK	projects/Webacy/contracts/utils/ERC721.sol	fba95a8b473698953eba5c1380384b962116cb37aad863aabdb833be07219093
PNF	projects/Webacy/contracts/utils/ParadigmNFT.sol	df71cb8ea02916dc15ca13b1afb4ff5ea4417581a96825cbf1b0ace85481cf60

APPROACH & METHODS | WEBACY (AUDIT)

This report has been prepared for Webacy (Audit) to discover issues and vulnerabilities in the source code of the Webacy (Audit) project as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Static Analysis and Manual Review techniques.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

The security assessment resulted in findings that ranged from critical to informational. We recommend addressing these findings to ensure a high level of security standards and industry practices. We suggest recommendations that could better serve the project from the security perspective:

- Testing the smart contracts against both common and uncommon attack vectors;
- Enhance general coding practices for better structures of source codes;
- Add enough unit tests to cover the possible use cases;
- Provide more comments per each function for readability, especially contracts that are verified in public;
- Provide more transparency on privileged activities once the protocol is live.

FINDINGS | WEBACY (AUDIT)

20	0	4	6	4	6
Total Findings	Critical	Major	Medium	Minor	Informational

This report has been prepared to discover issues and vulnerabilities for Webacy (Audit). Through this audit, we have uncovered 20 issues ranging from different severity levels. Utilizing the techniques of Static Analysis & Manual Review to complement rigorous manual code reviews, we discovered the following findings:

ID	Title	Category	Severity	Status
GLOBAL-01	**Centralization Risks**	**Centralization / Privilege**	**Major**	● **Mitigated**
WCP-01	**Centralized Control Of Contract Upgrade**	**Centralization / Privilege**	**Major**	● **Mitigated**
WCP-02	**Function Calls User-Provided Addresses With No Access Control Modifier**	**Centralization / Privilege**	**Major**	● **Acknowledged**
WCP-03	Dangerous Usage Of `tx.origin`	Volatile Code	Major	● Resolved
ASW-01	Missing Checks On Provided `uid`	Logical Issue	Medium	● Resolved
ASW-02	Missing Checks On `claimExpiryTime`	Logical Issue	Medium	● Resolved
MWC-01	Incorrect Fee Handling	Logical Issue	Medium	● Resolved
TAW-01	Ineffective Balance Check	Logical Issue	Medium	● Resolved
WCP-04	`Ischarity` Not Checked When Calling `transferUnclaimedAssets`	Logical Issue	Medium	● Resolved
WCP-05	Incorrect Allowance Check	Logical Issue	Medium	● Resolved

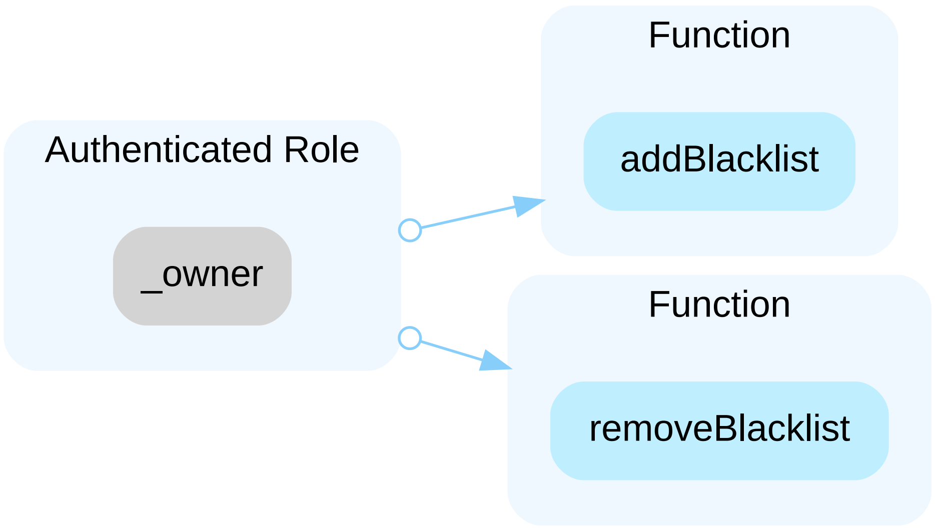
ID	Title	Category	Severity	Status
ASW-03	Missing Input Validation	Logical Issue	Minor	● Resolved
MWC-03	Missing Blacklist Address Check	Control Flow	Minor	● Resolved
WCP-06	Missing Checks On Approved Token Amount	Logical Issue	Minor	● Resolved
WCP-07	Check Effect Interaction Pattern Violated	Logical Issue	Minor	● Resolved
ASW-04	Discussion On Function `transferUnclaimedAssets`	Logical Issue	Informational	● Resolved
ASW-05	Incorrect Claimable Assets Calculation	Logical Issue	Informational	● Resolved
GLOBAL-02	Usage Of Transfer Pool	Logical Issue	Informational	● Resolved
GLOBAL-03	Lack Of Unit-Test File	Coding Style	Informational	● Resolved
WCP-08	Questionable Implementation Of Function `checkIfUIDExists`	Logical Issue	Informational	● Resolved
WCP-09	Redundant Checks	Logical Issue	Informational	● Resolved

GLOBAL-01 | CENTRALIZATION RISKS

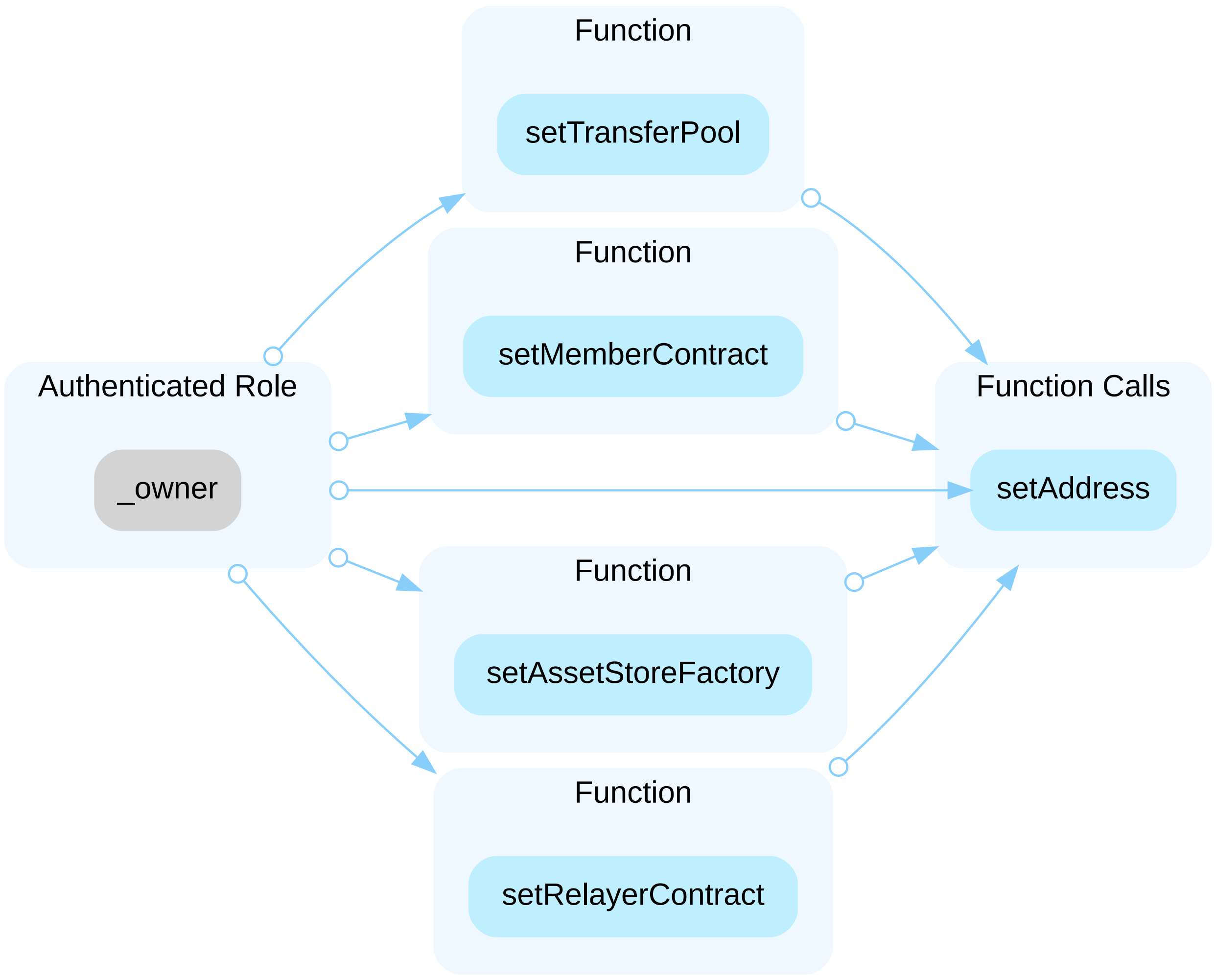
Category	Severity	Location	Status
Centralization / Privilege	● **Major**		● **Mitigated**

Description

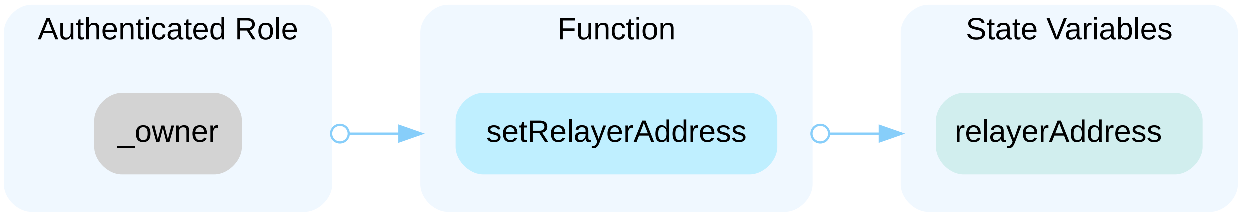
In the contract `Member` the role `_owner` has authority over the functions shown in the diagram below. Any compromise to the `_owner` account may allow the hacker to take advantage of this authority.



In the contract `ProtocolDirectory` the role `_owner` has authority over the functions shown in the diagram below. Any compromise to the `_owner` account may allow the hacker to take advantage of this authority.



In the contract `RelayerContract` the role `_owner` has authority over the functions shown in the diagram below. Any compromise to the `_owner` account may allow the hacker to take advantage of this authority.

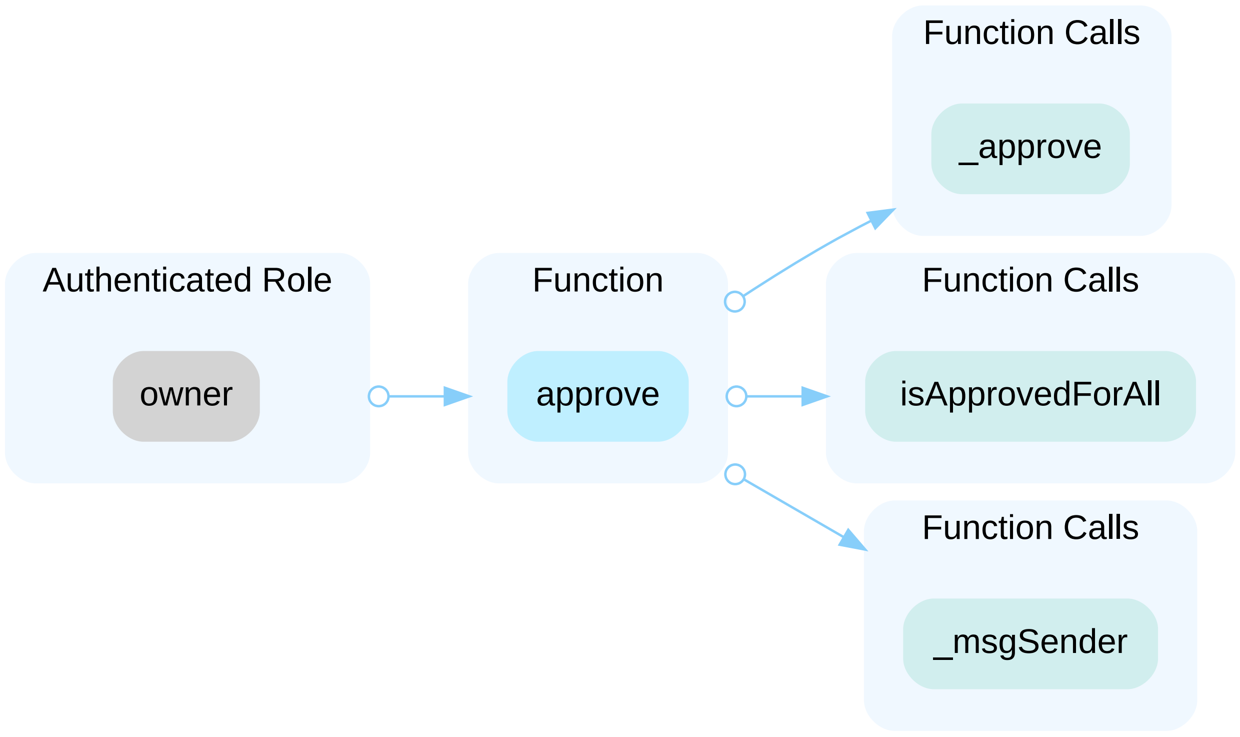


In the contract `RelayerContract` the role `relayer` has authority over the following functions:

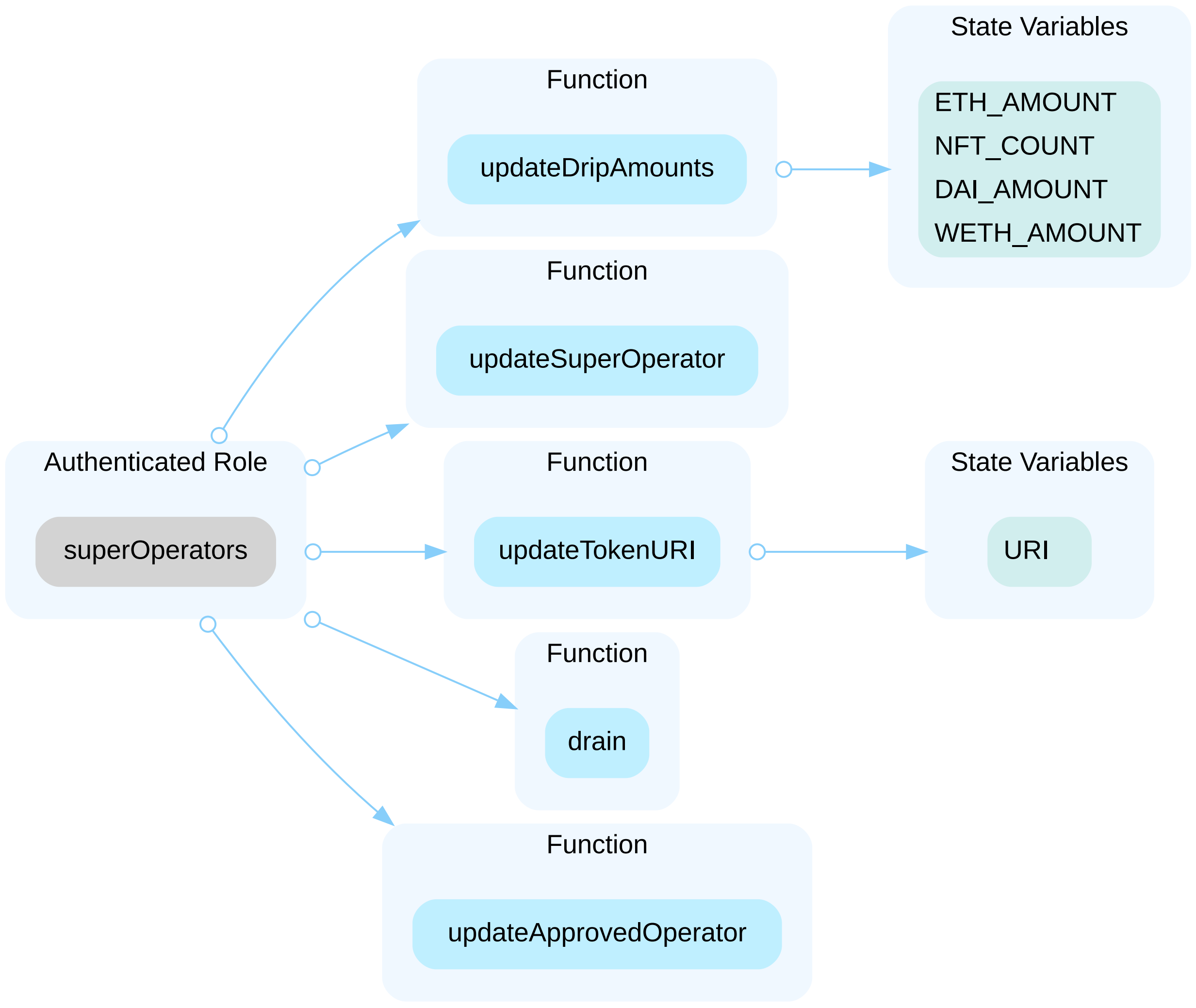
- function setApprovalActiveForUID()
- function transferUnclaimedAssets()
- function triggerAssetsForCharity()

Any compromise to the `relayer` account may allow the hacker to take advantage of this authority.

In the contract `ERC721` the role `owner` has authority over the functions shown in the diagram below. Any compromise to the `owner` account may allow the hacker to take advantage of this authority.



In the contract `MultiFaucet` the role `superOperators` has authority over the functions shown in the diagram below. Any compromise to the `superOperators` account may allow the hacker to take advantage of this authority.



Recommendation

The risk describes the current project design and potentially makes iterations to improve in the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multisignature wallets. Indicatively, here are some feasible suggestions that would also mitigate the potential risk at a different level in terms of short-term, long-term and permanent:

Short Term:

Timelock and Multi sign (⅔, ⅗) combination *mitigate* by delaying the sensitive operation and avoiding a single point of key management failure.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key compromised;

AND

- A medium/blog link for sharing the timelock contract and multi-signers addresses information with the public audience.

Long Term:

Timelock and DAO, the combination, *mitigate* by applying decentralization and transparency.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.
  AND
- A medium/blog link for sharing the timelock contract, multi-signers addresses, and DAO information with the public audience.

Permanent:

Renouncing the ownership or removing the function can be considered *fully resolved*.

- Renounce the ownership and never claim back the privileged roles.
  OR
- Remove the risky functionality.

Alleviation

[Webacy team]: We will implement time-lock on the smart contracts using open-zeppelin defender and there is already existing multi-sign ownership of the smart contracts with 2/4 approvals required for authorizing a transaction.

Certik: Based on our on-chain investigations, Protocol Directory Contract, Member Proxy Contract, Membership Factory Proxy Contract, assetsStore Proxy Contract, Relayer Contract, Whitelist Users Proxy Contract, and Blacklist Users Proxy Contract all have the same multi-sign ownership 0x435cb8f189FBD3D98ab07dD213442aEbA2b0D622. We advise the team to provide more details about the multi-sign ownership.

WCP-01 | CENTRALIZED CONTROL OF CONTRACT UPGRADE

Category	Severity	Location	Status
Centralization / Privilege	● **Major**	**projects/Webacy/contracts/AssetsStore.sol: 38; projects/Webacy/contracts/Member.sol: 43; projects/Webacy/contracts/ProtocolDirectory.sol: 19; projects/Webacy/contracts/RelayerContract.sol: 24; projects/Webacy/contracts/factories/AssetStoreFactory.sol: 23**	● **Mitigated**

Description

The attached contracts are upgradeable contracts, the owner can upgrade the contract without the community's commitment. If an attacker compromises the account, he can change the implementation of the contract and drain tokens from the contract.

Recommendation

The risk describes the current project design and potentially makes iterations to improve in the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multisignature wallets. Indicatively, here are some feasible suggestions that would also mitigate the potential risk at a different level in terms of short-term, long-term and permanent:

Short Term:

Timelock and Multi sign (⅔, ⅗) combination *mitigate* by delaying the sensitive operation and avoiding a single point of key management failure.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key compromised;
  AND
- A medium/blog link for sharing the timelock contract and multi-signers addresses information with the public audience.

Long Term:

Timelock and DAO, the combination, *mitigate* by applying decentralization and transparency.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.
  AND
- A medium/blog link for sharing the timelock contract, multi-signers addresses, and DAO information with the public audience.

Permanent:

Renouncing the ownership or removing the function can be considered *fully resolved*.

- Renounce the ownership and never claim back the privileged roles.
  OR
- Remove the risky functionality.

Alleviation

[Webacy team]: We will implement time-lock on the smart contracts using open-zeppelin defender and there is already existing multi-sign ownership of the smart contracts with 2/4 approvals required for authorizing a transaction.

Certik: Based on our on-chain investigations, Protocol Directory Contract, Member Proxy Contract, Membership Factory Proxy Contract, assetsStore Proxy Contract, Relayer Contract, Whitelist Users Proxy Contract, and Blacklist Users Proxy Contract all have the same multi-sign ownership 0x435cb8f189FBD3D98ab07dD213442aEbA2b0D622. We advise the team to provide more details about the multi-sign ownership.

WCP-02 FUNCTION CALLS USER-PROVIDED ADDRESSES WITH NO ACCESS CONTROL MODIFIER

Category	Severity	Location	Status
Centralization / Privilege	● **Major**	**projects/Webacy/contracts/AssetsStore.sol; projects/Webacy/contracts/Member.sol**	● **Acknowledged**

Description

User stored ERC20, ER721, and ERC1155 token addresses in approvals after calling function `storeAssetsAndBackUpApprovals` , `storeAssetsApprovals` in AssetsStore contract and function `storeBackupAssetsApprovals` in Member contract. Those addresses will later be used in claiming assets.

Calling a user-provided address is dangerous, especially in a public function with no access control restriction. An attacker could deploy a malicious contract and use the vulnerable function to trigger a call to the malicious contract, potentially stealing user funds or causing other serious damages.

Recommendation

We recommend several different types of mitigations, depending on the context:

1. Remove the vulnerable function, or restrict what addresses can be called from it.
2. Include access control mechanisms, whether it be through making the function `internal` or restricting which contracts can call this function.

Alleviation

[Webacy team]: We have introduced a new modifier called onlyMember that checks and validates if the msg.sender is a onChain Member or not. Allow sufficient access control to only allow members to access the function.

Certik: The new modifier onlyMember is redundant since the original check can perform the same functionality. The recommended approach here is to use a white list to store all tokens. Thus only allowed external token addresses can be called.

[Webacy team]: We will be implementing a whitelist to store all tokens in the near future.

WCP-03 | DANGEROUS USAGE OF `tx.origin`

Category	Severity	Location	Status
Volatile Code	● Major	projects/Webacy/contracts/AssetsStore.sol; projects/Webacy/contracts/Member.sol; projects/Webacy/contracts/factories/AssetStoreFactory.sol	● Resolved

Description

`tx.origin` check will be no longer valid after EIP-3074 is added in the coming months. EIP3074 introduces two EVM instructions AUTH and AUTHCALL. The first sets a context variable authorized based on an ECDSA signature. The second sends a call as authorized. This essential delegates control of the EOA to a smart contract. This means there will be a way for smart contracts to send transactions in the context of an Externally Owned Account, thus bypassing this check.

Also tx.origin is widely discouraged as there are possible phishing attempts.

For a contract, the `tx.origin` is not transparent. In a simple call chain A->B->C->D, inside D msg.sender will be C, and tx.origin will be A. If the origin is really desired in D, then each of the functions in the contracts B, C, D could take an extra parameter to propagate the origin: A would pass its address (this) to B, B would pass the value to C, and C would pass it to D. `tx.origin` based protection can be abused by a malicious contract if a legitimate user interacts with the malicious contract. Hence it's not recommended to use tx.origin for authorization in solidity documentation. refer to: https://docs.soliditylang.org/en/v0.7.0/security-considerations.html#tx-origin

Recommendation

We do not recommend the team use tx.origin to get the user info as it is vulnerable to phishing attacks.

Alleviation

[Webacy team]: The use of tx.origin has been replaced with msg.sender in order to capture address information of function executor.

Moreover, we have added _memberAddress as an additional parameter to be passed into the function call for both storeBackupAssetsApprovals() and storeAssetsApprovals() in order to allow internal calls from unifying function storeAssetsAndBackUpApprovals()

ASW-01 | MISSING CHECKS ON PROVIDED `uid`

Category	Severity	Location	Status
Logical Issue	● Medium	projects/Webacy/contracts/AssetsStore.sol: 542	● Resolved

Description

The function forgets to check if the approval is associated with the provided uid and just simply sets `MemberApprovals[uid][i].claimed` to true when `MemberApprovals[uid][i].approvalId` is equal to `_beneficiaryApproval.approvalId` .

The approvals in `BeneficiaryClaimableAsset[_charityBeneficiaryAddress]` might not all belong to the same uid.

Recommendation

We advise the team to check if `BeneficiaryClaimableAsset[_charityBeneficiaryAddress._uid` is the same as the input parameter `_uid` .

Alleviation

[Webacy team]: The UID check has been added to the function implementation. Fixed in commit: https://github.com/Webacy-Prod/mega-contracts/commit/6cd716a678308df450236f271747ef8c9832fadc.

ASW-02 | MISSING CHECKS ON `claimExpiryTime`

Category	Severity	Location	Status
Logical Issue	● Medium	projects/Webacy/contracts/AssetsStore.sol: 429, 521	● Resolved

Description

The function `claimAsset` and `sendAssetsToCharity` forget to check if the claim has expired.

Recommendation

We advise the team to add related checks.

Alleviation

[Webacy team]: Necessary claimable functionality and tests for timebased checks have been added to both the functions and test suite. Fixed in https://github.com/Webacy-Prod/mega-contracts/commit/6bef7d49b46fd70f967b7cdf20bac4f949176e76.

Certik: The checks here should require `claimExpiryTime` to be larger than current timestamp.

[Webacy team]: Fixed in https://github.com/Webacy-Prod/mega-contracts/commit/f68854a6e44c70d22360b1c866bf12016074a149.

MWC-01 | INCORRECT FEE HANDLING

Category	Severity	Location	Status
Logical Issue	● Medium	projects/Webacy/contracts/Member.sol: 635~636	● Resolved

Description

Based on the context, the webacyfees should be transferred to `IProtocolDirectory(directoryContract).getTransferPool()` instead of `_backUpWallet` .

Recommendation

We advise the team to provide related changes in the contract.

Alleviation

[Webacy team]: This issue has been fixed and the transfer is now to getTransferPool(). Fixed in commit: https://github.com/Webacy-Prod/mega-contracts/commit/5b0543cc4703923cca8c503b46449fd6c238f94c.

TAW-01 | INEFFECTIVE BALANCE CHECK

Category	Severity	Location	Status
Logical Issue	● Medium	projects/Webacy/contracts/libraries/TokenActions.sol: 47, 53~56, 65~68	● Resolved

Description

This function is used in AssetsStore and Member contracts to check if the wallet has enough balance. However, the input parameter `tokenAmount` here actually represents the percentage of allocated tokens in the wallet balance. The function `checkAssetContract` just uses `tokenAmount` to compare with the balance, which makes the check meaningless.

Furthermore, the real allocated amount is calculated when setting the approval to active. The calculation in `setApprovalActive` uses the current balance of the approved wallet, which makes the check unnecessary in the other aspect.

Recommendation

We advise the team to review the original design and recheck the implementation.

Alleviation

[Webacy Team]: We are using percentages for crypto-will, and this has been implemented for backups as well in order to avoid confusion in terms of value calculation. As regards the error mentioned above this has been fixed. Fixed in commit: https://github.com/Webacy-Prod/mega-contracts/commit/92b43238815c0b7c3043cb3f6357500ed11d404b and https://github.com/Webacy-Prod/mega-contracts/commit/b500529e9cceab8c49aaf4c96f4a14d076193eea.

Certik: All changes have been implemented except the function `editBackUp`.

Update: Fixed in https://github.com/Webacy-Prod/mega-contracts/commit/781aa8660a7ab4f4bee30273adca5b3a54071e61.

WCP-04 | `Ischarity` NOT CHECKED WHEN CALLING `transferUnclaimedAssets`

Category	Severity	Location	Status
Logical Issue	● Medium	projects/Webacy/contracts/AssetsStore.sol: 364, 521; projects/Webacy/contracts/RelayerContract.sol: 84, 116	● Resolved

Description

The function `sendAssetsToCharity` in the AssetsStore contract can only be called by the relayer contract and will send assets to the charity beneficiary based on the approval. In the relayer contract, the function `triggerAssetsForCharity` which triggers the `sendAssetsToCharity` function, can also only be called by the relayer. The relayer is an external address set by the owner after calling the function `setRelayerAddress`. This design provides the relayer a chance to transfer assets from the approved wallet to the transfer pool when the asset is supposed to transfer to the charity.

The relayer can decide not to call function `triggerAssetsForCharity` and wait for the claim to be expired. Then he can call function `transferUnclaimedAssets` to transfer the asset since the function doesn't check if the beneficiary in this approval is a charity.

Recommendation

We advise the team to limit the usage of `transferUnclaimedAssets` by checking `_approval[i].beneficiary.Ischarity`.

Alleviation

[Webacy team]: We have added an additional conditional in the `transferUnclaimedAssets` in order to ensure that charities are avoided during the transfer of unclaimed assets.

WCP-05 | INCORRECT ALLOWANCE CHECK

Category	Severity	Location	Status
Logical Issue	● Medium	projects/Webacy/contracts/AssetsStore.sol: 550~552; projects/Webacy/contracts/Member.sol: 621	● Resolved

Description

The check on line 550 in AssetsStore contract compares the allowance from the approved wallet to the relayer and the `_tokenAmount` , while the token transfer on line 551 is unrelated to the relayer.

The allowance on line 621 in Member contract refers to the allowance from the approved wallet to the Member contract, while the following token transfers on line 627-632 is unrelated to the Member contract.

Recommendation

We advise the team to review the original design, and perform related changes.

Alleviation

[Webacy Team]: This issue has been resolved for asset store and the allowance check for spender has changed to the assetstore contract similar to the issue raised for member contract. The design is intended as the approvals for these tokens were done against assetstore and member contract and the spenders are these contracts for which the user has previously done approvals for. Moreover, the allowance is calculated based on what the contract has allowance for to spend and accordingly transferred. Do let us know if there are any more clarifications needed. Fixed in https://github.com/Webacy-Prod/mega-contracts/commit/ff291d27a388b655772206c7dc1a82c200ccdd60.

Certik: Both mentioned allowance is not set in the contract, which causes confusion. The allowance check in asset store contract is removed.

ASW-03 | MISSING INPUT VALIDATION

Category	Severity	Location	Status
Logical Issue	● Minor	projects/Webacy/contracts/AssetsStore.sol: 190~191, 511~512	● Resolved

Description

The contract is missing checks to make sure those arrays all have the same length. Right now the checks can only make sure they have the same length in pairs.

Recommendation

We recommend adding the missing checks.

Alleviation

[Webacy Team]: We have changed the conditional from pair check to obtain one of the lengths of the parameters passed and do a cross check with other parameters which hopefully will cover the missing checks. Fixed in https://github.com/Webacy-Prod/mega-contracts/commit/b86f6f04cb9fa47a46f45c4e942dd9ecbba0ed99.

MWC-03 | MISSING BLACKLIST ADDRESS CHECK

Category	Severity	Location	Status
Control Flow	● Minor	projects/Webacy/contracts/Member.sol: 282, 295, 502, 745, 779	● Resolved

Description

The functions `addWallet` , `_addWallet` , `storeBackupAssetsApprovals` , `editBackUp` , and `editAllBackUp` all don't have a blacklist address check.

Recommendation

We advise the team to add the related checks.

Alleviation

[Webacy Team]: Fixed in https://github.com/Webacy-Prod/mega-contracts/commit/8d206b987606e4efc84d7454e6312fdadff5642f. The backup wallet will be checked before execute panic.

WCP-06 | MISSING CHECKS ON APPROVED TOKEN AMOUNT

Category	Severity	Location	Status
Logical Issue	● Minor	projects/Webacy/contracts/AssetsStore.sol; projects/Webacy/contracts/Member.sol	● Resolved

Description

The sum of the approved token amount among all approvals may exceed the approved wallet balance since there are no checks to ensure it. If so, the beneficiary who claims later might not have the chance to get what he is supposed to claim.

Recommendation

We advise the team to review this design and perform related changes.

Alleviation

[Webacy Team]: These balance checks are done in the Token Actions Library to check if the user has sufficient balance and allowance in order to this transfer, please check the library.

The sum of individual tokens is not needed to be checked as we are only dealing with individual tokens at a time from the dapp. However, we will add a check in the contracts to ensure that only individual tokens are passed.

WCP-07 | CHECK EFFECT INTERACTION PATTERN VIOLATED

Category	Severity	Location	Status
Logical Issue	● Minor	projects/Webacy/contracts/AssetsStore.sol: 364~421, 439~468, 542~565; projects/Webacy/contracts/Member.sol: 609~723	● Resolved

Description

The order of external call/transfer and storage manipulation must follow the check-effect-interaction pattern.

Recommendation

We advise the client to check if storage manipulation is before the external call/transfer operation.LINK

Alleviation

[Webacy Team]: This issue has been fixed and now follows the check-effect interaction pattern.

Certik: Although the commit description mentions WCP-07, the actual fixed repo is https://github.com/Webacy-Prod/mega-contracts/commit/ff291d27a388b655772206c7dc1a82c200ccdd60 instead of https://github.com/Webacy-Prod/mega-contracts/commit/d70052030448e8bcfc1580d179172e051ab7a1d3.

Update: Fixed in commit https://github.com/Webacy-Prod/mega-contracts/commit/c828f243e3e7d7c7d316824ea914574ec2e29a1a.

ASW-04 | DISCUSSION ON FUNCTION `transferUnclaimedAssets`

Category	Severity	Location	Status
Logical Issue	● Informational	projects/Webacy/contracts/AssetsStore.sol: 379~382	● Resolved

Description

The function `transferUnclaimedAssets` transfers `_tokenAmount` tokens to the transfer pool instead of just using `BeneficiaryClaimableAsset[_charityBeneficiaryAddress][i].approvedTokenAmount`. The `_tokenAmount` is calculated with the current wallet balance and the `approvedTokenAmount` is calculated with the balance when setting the approval to be active.

Recommendation

We would advise the team to check if it's an intended design.

Alleviation

[Webacy Team]: Yes this as intended design, as there can be difference in values during claiming times which can be less than actual approved amount. So in order to calculate the latest amount that the wallet contains to ensure a successful transaction the wallet balance is calculated when the approval is set to active.

ASW-05 | INCORRECT CLAIMABLE ASSETS CALCULATION

Category	Severity	Location	Status
Logical Issue	● Informational	projects/Webacy/contracts/AssetsStore.sol: 574	● Resolved

Description

In the documentation, the function `getClaimableAssets` allows users to get a list of all claimable assets. However, the list doesn't exclude all expired assets.

Recommendation

We advise the team to restrict the list to claimable assets only.

Alleviation

[Webacy Team]: We have mitigated this issue. Fixed in https://github.com/Webacy-Prod/mega-contracts/commit/e95f01a3f754157456b8b3b8575b4243886bb6d2.

GLOBAL-02 | USAGE OF TRANSFER POOL

Category	Severity	Location	Status
Logical Issue	● Informational		● Resolved

Description

In the project implementation, the transfer pool will accept the webacy fees and unclaimed assets from users. However, the usage of funds in transfer pool is not clear.

Recommendation

We advise the team to confirm this design and provide more illustrations on this.

Alleviation

[Webacy Team]: The usage of funds is to obtain the necessary charges and fees related to running Webacy functionality and operations. Transferring unclaimed funds is claimed when beneficiaries that are made to claim a users assets does not claim the users assets in due time and the assets are no longer of interest or value and is stuck inside the owners wallet, once the time period expires. Therefore, it is considered that the assets are unclaimed and transferred to the Webacy wallet.

GLOBAL-03 | LACK OF UNIT-TEST FILE

Category	Severity	Location	Status
Coding Style	● Informational		● Resolved

Description

Using unit tests to test smart contracts is one of the best ways to identify potential logic errors and security vulnerabilities in the smart contract. The unit test files in this project only contain the happy path tests.

Recommendation

We advise the team to add more test cases to cover more edge cases and improve the test coverage.

Alleviation

[Webacy Team]: We have added more tests cases in order to improve the test coverage as well as cover more path tests that have not previously been covered.

WCP-08 | QUESTIONABLE IMPLEMENTATION OF FUNCTION `checkIfUIDExists`

Category	Severity	Location	Status
Logical Issue	● Informational	projects/Webacy/contracts/AssetsStore.sol: 197~199; projects/Webacy/contracts/Member.sol: 164, 164	● Resolved

Description

The implementation of function `checkIfUIDExists` only checks if the `_walletAddress` has been added to any members. The implementation doesn't match the function name and also affects the project logic.

When the function `checkIfUIDExists` is used as a check in function `storeAssetsApprovals` of `AssetsStore` contract, function `storeAssetsApprovals` then calls `createMember` in member contract to create a new member with the unique uid. If the caller (wallet) has not been added to any members while the member has already been created, the function would revert.

Recommendation

We advise the team to change the implementation of function `checkIfUIDExists` or change the function name.

Alleviation

[Webacy Team]: Function name has been changed to checkIfWalletExists. Fixed in https://github.com/Webacy-Prod/mega-contracts/commit/d70052030448e8bcfc1580d179172e051ab7a1d3.

WCP-09 | REDUNDANT CHECKS

Category	Severity	Location	Status
Logical Issue	● Informational	projects/Webacy/contracts/AssetsStore.sol: 435~436; projects/Webacy/contracts/Member.sol: 337~343	● Resolved

Description

In the function `claimAsset` of AssetsStore contract, the aforementioned statement is always true since the beneficiary can not be changed after it's stored.

In the function `_addBackupWallet` of Member contract, If the check on line 325 doesn’t revert, checkIfUIDExists(_user) must return true and _member.wallets.length can not be 0.

Recommendation

We advise removing the aforementioned check.

Alleviation

[Webacy Team]: We have removed the aforementioned check. Fixed in https://github.com/Webacy-Prod/mega-contracts/commit/1f8ebb061f34c4d532bb55f5c85e4a95501d20a8.

FORMAL VERIFICATION | WEBACY (AUDIT)

Formal guarantees about the behavior of smart contracts can be obtained by reasoning about properties relating to the entire contract (e.g. contract invariants) or to specific functions of the contract. Once such properties are proven to be valid, they guarantee that the contract behaves as specified by the property. As part of this audit, we applied automated formal verification (symbolic model checking) to prove that well-known functions in the smart contracts adhere to their expected behavior.

Considered Functions And Scope

In the following, we provide a description of the properties that have been used in this audit. They are grouped according to the type of contract they apply to.

Verification of ERC-20 Compliance

We verified properties of the public interface of those token contracts that implement the ERC-20 interface. This covers

- Functions `transfer` and `transferFrom` that are widely used for token transfers,
- functions `approve` and `allowance` that enable the owner of an account to delegate a certain subset of her tokens to another account (i.e. to grant an allowance), and
- the functions `balanceOf` and `totalSupply` , which are verified to correctly reflect the internal state of the contract.

The properties that were considered within the scope of this audit are as follows:

Property Name	Title
erc20-transfer-revert-zero	Function `transfer` Prevents Transfers to the Zero Address
erc20-transfer-succeed-self	Function `transfer` Succeeds on Admissible Self Transfers
erc20-transfer-succeed-normal	Function `transfer` Succeeds on Admissible Non-self Transfers
erc20-transfer-correct-amount	Function `transfer` Transfers the Correct Amount in Non-self Transfers
erc20-transfer-change-state	Function `transfer` Has No Unexpected State Changes
erc20-transfer-correct-amount-self	Function `transfer` Transfers the Correct Amount in Self Transfers
erc20-transfer-exceed-balance	Function `transfer` Fails if Requested Amount Exceeds Available Balance
erc20-transfer-recipient-overflow	Function `transfer` Prevents Overflows in the Recipient's Balance
erc20-transfer-false	If Function `transfer` Returns `false` , the Contract State Has Not Been Changed
erc20-transfer-never-return-false	Function `transfer` Never Returns `false`

Property Name	Title
erc20-transferfrom-revert-to-zero	Function `transferFrom` Fails for Transfers To the Zero Address
erc20-transferfrom-revert-from-zero	Function `transferFrom` Fails for Transfers From the Zero Address
erc20-transferfrom-succeed-normal	Function `transferFrom` Succeeds on Admissible Non-self Transfers
erc20-transferfrom-correct-amount	Function `transferFrom` Transfers the Correct Amount in Non-self Transfers
erc20-transferfrom-correct-amount-self	Function `transferFrom` Performs Self Transfers Correctly
erc20-transferfrom-succeed-self	Function `transferFrom` Succeeds on Admissible Self Transfers
erc20-transferfrom-fail-exceed-balance	Function `transferFrom` Fails if the Requested Amount Exceeds the Available Balance
erc20-transferfrom-correct-allowance	Function `transferFrom` Updated the Allowance Correctly
erc20-transferfrom-change-state	Function `transferFrom` Has No Unexpected State Changes
erc20-transferfrom-fail-exceed-allowance	Function `transferFrom` Fails if the Requested Amount Exceeds the Available Allowance
erc20-totalsupply-succeed-always	Function `totalSupply` Always Succeeds
erc20-transferfrom-false	If Function `transferFrom` Returns `false`, the Contract's State Has Not Been Changed
erc20-transferfrom-fail-recipient-overflow	Function `transferFrom` Prevents Overflows in the Recipient's Balance
erc20-transferfrom-never-return-false	Function `transferFrom` Never Returns `false`
erc20-totalsupply-correct-value	Function `totalSupply` Returns the Value of the Corresponding State Variable
erc20-totalsupply-change-state	Function `totalSupply` Does Not Change the Contract's State
erc20-balanceof-succeed-always	Function `balanceOf` Always Succeeds
erc20-balanceof-correct-value	Function `balanceOf` Returns the Correct Value
erc20-balanceof-change-state	Function `balanceOf` Does Not Change the Contract's State
erc20-allowance-succeed-always	Function `allowance` Always Succeeds
erc20-allowance-correct-value	Function `allowance` Returns Correct Value

Property Name	Title
erc20-allowance-change-state	Function `allowance` Does Not Change the Contract's State
erc20-approve-revert-zero	Function `approve` Prevents Giving Approvals For the Zero Address
erc20-approve-succeed-normal	Function `approve` Succeeds for Admissible Inputs
erc20-approve-correct-amount	Function `approve` Updates the Approval Mapping Correctly
erc20-approve-change-state	Function `approve` Has No Unexpected State Changes
erc20-approve-false	If Function `approve` Returns `false`, the Contract's State Has Not Been Changed
erc20-approve-never-return-false	Function `approve` Never Returns `false`

Verification of ERC-721 Compliance

We verified the properties of the public interface of those token contracts that implement the ERC-721 interface without pause.

The properties that were considered within the scope of this audit are as follows:

Property Name	Title
erc721-supportsinterface-correct-erc721	Function `supportsInterface` Signals that the Contract Supports `ERC721`
erc721-balanceof-succeed-normal	Function `balanceOf` Succeeds on Admissible Inputs
erc721-balanceof-correct-count	Function `balanceOf` Returns the Correct Value
erc721-balanceof-revert	Function `balanceOf` Fails on the Zero Address
erc721-balanceof-no-change-state	Function `balanceOf` Does Not Change the Contract's State
erc721-transferfrom-succeed-normal	Function `transferFrom` Succeeds on Admissible Inputs
erc721-ownerof-succeed-normal	Function `ownerOf` Succeeds For Valid Tokens
erc721-ownerof-correct-owner	Function `ownerOf` Returns the Correct Owner
erc721-ownerof-revert	Function `ownerOf` Fails On Invalid Tokens
erc721-ownerof-no-change-state	Function `ownerOf` Does Not Change the Contract's State
erc721-getapproved-succeed-normal	Function `getApproved` Succeeds For Valid Tokens

Property Name	Title
erc721-getapproved-correct-value	Function `getApproved` Returns Correct Approved Address
erc721-isapprovedforall-succeed-normal	Function `isApprovedForAll` Always Succeeds
erc721-getapproved-change-state	Function `getApproved` Does Not Change the Contract's State
erc721-getapproved-revert-zero	Function `getApproved` Fails on Invalid Tokens
erc721-isapprovedforall-correct	Function `isApprovedForAll` Returns Correct Approvals
erc721-isapprovedforall-change-state	Function `isApprovedForAll` Does Not Change the Contract's State
erc721-isapprovedforall-correct-false	Function `isApprovedForAll` Returns Non-Approval For Invalid Inputs
erc721-approve-succeed-normal	Function `approve` Return for Admissible Inputs
erc721-approve-set-correct	Function `approve` Sets Approve
erc721-approve-revert-not-allowed	Function `approve` Prevents Unpermitted Approvals
erc721-setapprovalforall-succeed-normal	Function `setApprovalForAll` Return for Admissible Inputs
erc721-approve-revert-invalid-token	Function `approve` Fails For Calls with Invalid Tokens
erc721-approve-change-state	Function `approve` Has No Unexpected State Changes
erc721-setapprovalforall-set-correct	Function `setApprovalForAll` Approves Operator
erc721-setapprovalforall-multiple	Function `setApprovalForAll` Can Set Multiple Operators
erc721-setapprovalforall-change-state	Function `setApprovalForAll` Has No Unexpected State Changes
erc721-setapprovalforall-revert-zero	Function `setApprovalForAll` Prevents Giving Approvals to the Zero Address
erc721-transferfrom-correct-increase	Function `transferFrom` Transfers the Complete Token in Non-self Transfers
erc721-transferfrom-correct-one-token-self	Function `transferFrom` Performs Self Transfers Correctly
erc721-transferfrom-correct-approval	Function `transferFrom` Updates the Approval Correctly
erc721-transferfrom-correct-owner-from	Function `transferFrom` Removes Token Ownership of From
erc721-transferfrom-correct-owner-to	Function `transferFrom` Transfers Ownership

Property Name	Title
erc721-transferfrom-correct-balance	Function `transferFrom` Sum of Balances is Constant
erc721-transferfrom-correct-state-balance	Function `transferFrom` Keeps Balances Constant Except for From and To
erc721-transferfrom-correct-state-owner	Function `transferFrom` Has Expected Ownership Changes
erc721-transferfrom-correct-state-approval	Function `transferFrom` Has Expected Approval Changes
erc721-transferfrom-revert-invalid	Function `transferFrom` Fails for Invalid Tokens
erc721-transferfrom-revert-from-zero	Function `transferFrom` Fails for Transfers From the Zero Address
erc721-transferfrom-revert-to-zero	Function `transferFrom` Fails for Transfers To the Zero Address
erc721-supportsinterface-metadata	Function `supportsInterface` Returns that Interface ERC721Metadata Implemented
erc721-supportsinterface-succeed-always	Function `supportsInterface` Always Succeeds
erc721-transferfrom-revert-not-owned	Function `transferFrom` Fails if `From` Is Not Token Owner
erc721-supportsinterface-correct-erc165	Function `supportsInterface` Signals that the Contract Supports ERC165
erc721-supportsinterface-correct-false	Function `supportsInterface` Returns `False` for Id 0xffffffff
erc721-supportsinterface-no-change-state	Function `supportsInterface` Does Not Change the Contract's State
erc721-transferfrom-revert-exceed-approval	Function `transferFrom` Fails for Token Transfers without Approval

Verification Results

For the following contracts, model checking established that each of the properties that were in scope of this audit (see scope) are valid:

Detailed Results For Contract WebacyToken (projects/Webacy/contracts/utils/ERC20.sol) In Commit b62f7ff5f75d7202a8d14978da8c7dd0183204de

Verification of ERC-20 Compliance

Detailed results for function `transfer`

Property Name	Final Result	Remarks
erc20-transfer-revert-zero	● True	
erc20-transfer-succeed-self	● True	
erc20-transfer-succeed-normal	● True	
erc20-transfer-correct-amount	● True	
erc20-transfer-change-state	● True	
erc20-transfer-correct-amount-self	● True	
erc20-transfer-exceed-balance	● True	
erc20-transfer-recipient-overflow	● True	
erc20-transfer-false	● True	
erc20-transfer-never-return-false	● True	

Detailed results for function `transferFrom`

Property Name	Final Result	Remarks
erc20-transferfrom-revert-to-zero	● True	
erc20-transferfrom-revert-from-zero	● True	
erc20-transferfrom-succeed-normal	● True	
erc20-transferfrom-correct-amount	● True	
erc20-transferfrom-correct-amount-self	● True	
erc20-transferfrom-succeed-self	● True	
erc20-transferfrom-fail-exceed-balance	● True	
erc20-transferfrom-correct-allowance	● True	
erc20-transferfrom-change-state	● True	
erc20-transferfrom-fail-exceed-allowance	● True	
erc20-transferfrom-false	● True	
erc20-transferfrom-fail-recipient-overflow	● True	
erc20-transferfrom-never-return-false	● True	

Detailed results for function `totalSupply`

Property Name	Final Result	Remarks
erc20-totalsupply-succeed-always	● True	
erc20-totalsupply-correct-value	● True	
erc20-totalsupply-change-state	● True	

Detailed results for function `balanceOf`

Property Name	Final Result	Remarks
erc20-balanceof-succeed-always	● True	
erc20-balanceof-correct-value	● True	
erc20-balanceof-change-state	● True	

Detailed results for function `allowance`

Property Name	Final Result	Remarks
erc20-allowance-succeed-always	● True	
erc20-allowance-correct-value	● True	
erc20-allowance-change-state	● True	

Detailed results for function `approve`

Property Name	Final Result	Remarks
erc20-approve-revert-zero	● True	
erc20-approve-succeed-normal	● True	
erc20-approve-correct-amount	● True	
erc20-approve-change-state	● True	
erc20-approve-false	● True	
erc20-approve-never-return-false	● True	

In the remainder of this section, we list all contracts where model checking of at least one property was not successful. There are several reasons why this could happen:

- Model checking reports a counterexample that violates the property. Depending on the counterexample,this occurs if
    - The specification of the property is too generic and does not accurately capture the intended behavior of the smart contract. In that case, the counterexample does not indicate a problem in the underlying smart contract. We report such instances as being "inapplicable".
    - The property is applicable to the smart contract. In that case, the counterexample showcases a problem in the smart contract and a correspond finding is reported separately in the Findings section of this

report. In the following tables, we report such instances as "invalid". The distinction between spurious and actual counterexamples is done manually by the auditors.

- The model checking result is inconclusive. Such a result does not indicate a problem in the underlying smart contract. An inconclusive result may occur if
    - The model checking engine fails to construct a proof. This can happen if the logical deductions necessary are beyond the capabilities of the automated reasoning tool. It is a technical limitation of all proof engines and cannot be avoided in general.
    - The model checking engine runs out of time or memory and did not produce a result. This can happen if automatic abstraction techniques are ineffective or of the state space is too big.

Detailed Results For Contract WebacyNFT (projects/Webacy/contracts/utils/ERC721.sol) In Commit b62f7ff5f75d7202a8d14978da8c7dd0183204de

Verification of ERC-721 Compliance

Detailed results for function `supportsInterface`

Property Name	Final Result	Remarks
erc721-supportsinterface-correct-erc721	● True	
erc721-supportsinterface-metadata	● True	
erc721-supportsinterface-succeed-always	● True	
erc721-supportsinterface-correct-erc165	● True	
erc721-supportsinterface-correct-false	● True	
erc721-supportsinterface-no-change-state	● True	

Detailed results for function `balanceOf`

Property Name	Final Result	Remarks
erc721-balanceof-succeed-normal	● True	
erc721-balanceof-correct-count	● True	
erc721-balanceof-revert	● True	
erc721-balanceof-no-change-state	● True	

Detailed results for function `transferFrom`

Property Name	Final Result	Remarks
erc721-transferfrom-succeed-normal	● True	
erc721-transferfrom-correct-increase	● True	
erc721-transferfrom-correct-one-token-self	● True	
erc721-transferfrom-correct-approval	● True	
erc721-transferfrom-correct-owner-from	● True	
erc721-transferfrom-correct-owner-to	● True	
erc721-transferfrom-correct-balance	● True	
erc721-transferfrom-correct-state-balance	● True	
erc721-transferfrom-correct-state-owner	● True	
erc721-transferfrom-correct-state-approval	● True	
erc721-transferfrom-revert-invalid	● True	
erc721-transferfrom-revert-from-zero	● True	
erc721-transferfrom-revert-to-zero	● True	
erc721-transferfrom-revert-not-owned	● True	
erc721-transferfrom-revert-exceed-approval	● True	

Detailed results for function `ownerOf`

Property Name	Final Result	Remarks
erc721-ownerof-succeed-normal	● True	
erc721-ownerof-correct-owner	● True	
erc721-ownerof-revert	● True	
erc721-ownerof-no-change-state	● True	

Detailed results for function `getApproved`

Property Name	Final Result	Remarks
erc721-getapproved-succeed-normal	● True	
erc721-getapproved-correct-value	● True	
erc721-getapproved-change-state	● True	
erc721-getapproved-revert-zero	● True	

Detailed results for function `isApprovedForAll`

Property Name	Final Result	Remarks
erc721-isapprovedforall-succeed-normal	● True	
erc721-isapprovedforall-correct	● True	
erc721-isapprovedforall-change-state	● True	
erc721-isapprovedforall-correct-false	● Inapplicable	Intended behavior

Detailed results for function `approve`

Property Name	Final Result	Remarks
erc721-approve-succeed-normal	● True	
erc721-approve-set-correct	● True	
erc721-approve-revert-not-allowed	● True	
erc721-approve-revert-invalid-token	● True	
erc721-approve-change-state	● True	

Detailed results for function `setApprovalForAll`

Property Name	Final Result	Remarks
erc721-setapprovalforall-succeed-normal	● True	
erc721-setapprovalforall-set-correct	● True	
erc721-setapprovalforall-multiple	● True	
erc721-setapprovalforall-change-state	● True	
erc721-setapprovalforall-revert-zero	● Inapplicable	Context not considered

Detailed Results For Contract Webacy2NFT (projects/Webacy/contracts/utils/ERC721.sol) In Commit b62f7ff5f75d7202a8d14978da8c7dd0183204de

Verification of ERC-721 Compliance

Detailed results for function `supportsInterface`

Property Name	Final Result	Remarks
erc721-supportsinterface-correct-erc721	● True	
erc721-supportsinterface-succeed-always	● True	
erc721-supportsinterface-metadata	● True	
erc721-supportsinterface-correct-erc165	● True	
erc721-supportsinterface-correct-false	● True	
erc721-supportsinterface-no-change-state	● True	

Detailed results for function `balanceOf`

Property Name	Final Result	Remarks
erc721-balanceof-succeed-normal	● True	
erc721-balanceof-correct-count	● True	
erc721-balanceof-revert	● True	
erc721-balanceof-no-change-state	● True	

Detailed results for function `ownerOf`

Property Name	Final Result	Remarks
erc721-ownerof-succeed-normal	● True	
erc721-ownerof-correct-owner	● True	
erc721-ownerof-no-change-state	● True	
erc721-ownerof-revert	● True	

Detailed results for function `transferFrom`

Property Name	Final Result	Remarks
erc721-transferfrom-succeed-normal	● True	
erc721-transferfrom-correct-increase	● True	
erc721-transferfrom-correct-one-token-self	● True	
erc721-transferfrom-correct-owner-from	● True	
erc721-transferfrom-correct-approval	● True	
erc721-transferfrom-correct-owner-to	● True	
erc721-transferfrom-correct-balance	● True	
erc721-transferfrom-correct-state-balance	● True	
erc721-transferfrom-correct-state-owner	● True	
erc721-transferfrom-correct-state-approval	● True	
erc721-transferfrom-revert-invalid	● True	
erc721-transferfrom-revert-to-zero	● True	
erc721-transferfrom-revert-from-zero	● True	
erc721-transferfrom-revert-not-owned	● True	
erc721-transferfrom-revert-exceed-approval	● True	

Detailed results for function `getApproved`

Property Name	Final Result	Remarks
erc721-getapproved-succeed-normal	● True	
erc721-getapproved-correct-value	● True	
erc721-getapproved-change-state	● True	
erc721-getapproved-revert-zero	● True	

Detailed results for function `isApprovedForAll`

Property Name	Final Result	Remarks
erc721-isapprovedforall-succeed-normal	● True	
erc721-isapprovedforall-correct	● True	
erc721-isapprovedforall-change-state	● True	
erc721-isapprovedforall-correct-false	● Inapplicable	Intended behavior

Detailed results for function `approve`

Property Name	Final Result	Remarks
erc721-approve-succeed-normal	● True	
erc721-approve-set-correct	● True	
erc721-approve-revert-not-allowed	● True	
erc721-approve-change-state	● True	
erc721-approve-revert-invalid-token	● True	

Detailed results for function `setApprovalForAll`

Property Name	Final Result	Remarks
erc721-setapprovalforall-succeed-normal	● True	
erc721-setapprovalforall-set-correct	● True	
erc721-setapprovalforall-multiple	● True	
erc721-setapprovalforall-change-state	● True	
erc721-setapprovalforall-revert-zero	● Inapplicable	Context not considered

Detailed Results For Contract ERC721 (projects/Webacy/contracts/utils/ParadigmNFT.sol) In Commit b62f7ff5f75d7202a8d14978da8c7dd0183204de

Verification of ERC-721 Compliance

Detailed results for function `balanceOf`

Property Name	Final Result	Remarks
erc721-balanceof-succeed-normal	● True	
erc721-balanceof-correct-count	● True	
erc721-balanceof-no-change-state	● True	
erc721-balanceof-revert	● True	

Detailed results for function `supportsInterface`

Property Name	Final Result	Remarks
erc721-supportsinterface-correct-erc721	● True	
erc721-supportsinterface-succeed-always	● True	
erc721-supportsinterface-metadata	● True	
erc721-supportsinterface-correct-erc165	● True	
erc721-supportsinterface-no-change-state	● True	
erc721-supportsinterface-correct-false	● True	

Detailed results for function `transferFrom`

Property Name	Final Result	Remarks
erc721-transferfrom-succeed-normal	● True	
erc721-transferfrom-correct-increase	● True	
erc721-transferfrom-correct-one-token-self	● True	
erc721-transferfrom-correct-approval	● True	
erc721-transferfrom-correct-owner-from	● True	
erc721-transferfrom-correct-owner-to	● True	
erc721-transferfrom-correct-balance	● True	
erc721-transferfrom-correct-state-owner	● True	
erc721-transferfrom-correct-state-balance	● True	
erc721-transferfrom-correct-state-approval	● True	
erc721-transferfrom-revert-invalid	● True	
erc721-transferfrom-revert-from-zero	● True	
erc721-transferfrom-revert-to-zero	● True	
erc721-transferfrom-revert-not-owned	● True	
erc721-transferfrom-revert-exceed-approval	● True	

Detailed results for function `ownerOf`

Property Name	Final Result	Remarks
erc721-ownerof-succeed-normal	● True	
erc721-ownerof-correct-owner	● True	
erc721-ownerof-no-change-state	● True	
erc721-ownerof-revert	● True	

Detailed results for function `getApproved`

Property Name	Final Result	Remarks
erc721-getapproved-succeed-normal	● True	
erc721-getapproved-correct-value	● True	
erc721-getapproved-change-state	● True	
erc721-getapproved-revert-zero	● True	

Detailed results for function `isApprovedForAll`

Property Name	Final Result	Remarks
erc721-isapprovedforall-succeed-normal	● True	
erc721-isapprovedforall-correct	● True	
erc721-isapprovedforall-change-state	● True	
erc721-isapprovedforall-correct-false	● Inapplicable	Context not considered

Detailed results for function `approve`

Property Name	Final Result	Remarks
erc721-approve-succeed-normal	● True	
erc721-approve-set-correct	● True	
erc721-approve-revert-not-allowed	● True	
erc721-approve-revert-invalid-token	● True	
erc721-approve-change-state	● True	

Detailed results for function `setApprovalForAll`

Property Name	Final Result	Remarks
erc721-setapprovalforall-succeed-normal	● True	
erc721-setapprovalforall-set-correct	● True	
erc721-setapprovalforall-multiple	● True	
erc721-setapprovalforall-revert-zero	● Inapplicable	Context not considered
erc721-setapprovalforall-change-state	● True	

Detailed Results For Contract MultiFaucet (projects/Webacy/contracts/utils/ParadigmNFT.sol) In Commit b62f7ff5f75d7202a8d14978da8c7dd0183204de

Verification of ERC-721 Compliance

Detailed results for function `supportsInterface`

Property Name	Final Result	Remarks
erc721-supportsinterface-correct-erc721	● True	
erc721-supportsinterface-metadata	● True	
erc721-supportsinterface-succeed-always	● True	
erc721-supportsinterface-correct-erc165	● True	
erc721-supportsinterface-correct-false	● True	
erc721-supportsinterface-no-change-state	● True	

Detailed results for function `balanceOf`

Property Name	Final Result	Remarks
erc721-balanceof-succeed-normal	● True	
erc721-balanceof-correct-count	● True	
erc721-balanceof-revert	● True	
erc721-balanceof-no-change-state	● True	

Detailed results for function `transferFrom`

Property Name	Final Result	Remarks
erc721-transferfrom-succeed-normal	● True	
erc721-transferfrom-correct-one-token-self	● True	
erc721-transferfrom-correct-increase	● True	
erc721-transferfrom-correct-approval	● True	
erc721-transferfrom-correct-owner-from	● True	
erc721-transferfrom-correct-owner-to	● True	
erc721-transferfrom-correct-balance	● True	
erc721-transferfrom-correct-state-balance	● True	
erc721-transferfrom-correct-state-owner	● True	
erc721-transferfrom-correct-state-approval	● True	
erc721-transferfrom-revert-invalid	● True	
erc721-transferfrom-revert-from-zero	● True	
erc721-transferfrom-revert-to-zero	● True	
erc721-transferfrom-revert-not-owned	● True	
erc721-transferfrom-revert-exceed-approval	● True	

Detailed results for function `ownerOf`

Property Name	Final Result	Remarks
erc721-ownerof-succeed-normal	● True	
erc721-ownerof-correct-owner	● True	
erc721-ownerof-revert	● True	
erc721-ownerof-no-change-state	● True	

Detailed results for function `getApproved`

Property Name	Final Result	Remarks
erc721-getapproved-succeed-normal	● True	
erc721-getapproved-correct-value	● True	
erc721-getapproved-revert-zero	● True	
erc721-getapproved-change-state	● True	

Detailed results for function `isApprovedForAll`

Property Name	Final Result	Remarks
erc721-isapprovedforall-succeed-normal	● True	
erc721-isapprovedforall-correct	● True	
erc721-isapprovedforall-change-state	● True	
erc721-isapprovedforall-correct-false	● Inapplicable	Context not considered

Detailed results for function `approve`

Property Name	Final Result	Remarks
erc721-approve-succeed-normal	● True	
erc721-approve-set-correct	● True	
erc721-approve-revert-not-allowed	● True	
erc721-approve-revert-invalid-token	● True	
erc721-approve-change-state	● True	

Detailed results for function `setApprovalForAll`

Property Name	Final Result	Remarks
erc721-setapprovalforall-succeed-normal	● True	
erc721-setapprovalforall-set-correct	● True	
erc721-setapprovalforall-multiple	● True	
erc721-setapprovalforall-change-state	● True	
erc721-setapprovalforall-revert-zero	● Inapplicable	Context not considered

APPENDIX | WEBACY (AUDIT)

Finding Categories

Categories	Description
Centralization / Privilege	Centralization / Privilege findings refer to either feature logic or implementation of components that act against the nature of decentralization, such as explicit ownership or specialized access roles in combination with a mechanism to relocate funds.
Logical Issue	Logical Issue findings detail a fault in the logic of the linked code, such as an incorrect notion on how block.timestamp works.
Control Flow	Control Flow findings concern the access control imposed on functions, such as owner-only functions being invoke-able by anyone under certain circumstances.
Volatile Code	Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.
Coding Style	Coding Style findings usually do not affect the generated byte-code but rather comment on how to make the codebase more legible and, as a result, easily maintainable.

Checksum Calculation Method

The "Checksum" field in the "Audit Scope" section is calculated as the SHA-256 (Secure Hash Algorithm 2 with digest size of 256 bits) digest of the content of each file hosted in the listed source repository under the specified commit.

The result is hexadecimal encoded and is the same as the output of the Linux "sha256sum" command against the target file.

Details on Formal Verification

Some Solidity smart contracts from this project have been formally verified using symbolic model checking. Each such contract was compiled into a mathematical model which reflects all its possible behaviors with respect to the property. The model takes into account the semantics of the Solidity instructions found in the contract. All verification results that we report are based on that model.

Technical Description

The model also formalizes a simplified execution environment of the Ethereum blockchain and a verification harness that performs the initialization of the contract and all possible interactions with the contract. Initially, the contract state is initialized non-deterministically (i.e. by arbitrary values) and over-approximates the reachable state space of the contract throughout any actual deployment on chain. All valid results thus carry over to the contract's behavior in arbitrary states after it has been deployed.

Assumptions and Simplifications

The following assumptions and simplifications apply to our model:

- Gas consumption is not taken into account, i.e. we assume that executions do not terminate prematurely because they run out of gas.
- The contract's state variables are non-deterministically initialized before invocation of any function. That ignores contract invariants and may lead to false positives. It is, however, a safe over-approximation.
- The verification engine reasons about unbounded integers. Machine arithmetic is modeled using modular arithmetic based on the bit-width of the underlying numeric Solidity type. This ensures that over- and underflow characteristics are faithfully represented.
- Certain low-level calls and inline assembly are not supported and may lead to a contract not being formally verified.
- We model the semantics of the Solidity source code and not the semantics of the EVM bytecode in a compiled contract.

Formalism for Property Specification

All properties are expressed in linear temporal logic (LTL). For that matter, we treat each invocation of and each return from a public or an external function as a discrete time step. Our analysis reasons about the contract's state upon entering and upon leaving public or external functions.

Apart from the Boolean connectives and the modal operators "always" (written `[]`) and "eventually" (written `<>`), we use the following predicates as atomic propositions. They are evaluated on the contract's state whenever a discrete time step occurs:

- `started(f, [cond])` Indicates an invocation of contract function `f` within a state satisfying formula `cond`.
- `willSucceed(f, [cond])` Indicates an invocation of contract function `f` within a state satisfying formula `cond` and considers only those executions that do not revert.
- `finished(f, [cond])` Indicates that execution returns from contract function `f` in a state satisfying formula `cond`. Here, formula `cond` may refer to the contract's state variables and to the value they had upon entering the function (using the `old` function).
- `reverted(f, [cond])` Indicates that execution of contract function `f` was interrupted by an exception in a contract state satisfying formula `cond`.

The verification performed in this audit operates on a harness that non-deterministically invokes a function of the contract's public or external interface. All formulas are analyzed w.r.t. the trace that corresponds to this function invocation.

Description of the Analyzed ERC-20 Properties

The specifications are designed such that they capture the desired and admissible behaviors of the ERC-20 functions `transfer`, `transferFrom`, `approve`, `allowance`, `balanceOf`, and `totalSupply`. In the following, we list those property specifications.

Properties related to function `transfer`

erc20-transfer-revert-zero

Function `transfer` Prevents Transfers to the Zero Address. Any call of the form `transfer(recipient, amount)` must fail if the recipient address is the zero address. Specification:

```
[](started(contract.transfer(to, value), to == address(0)) ==>
  <>(reverted(contract.transfer) || finished(contract.transfer(to, value), return
    == false)))
```

erc20-transfer-succeed-normal

Function `transfer` Succeeds on Admissible Non-self Transfers. All invocations of the form `transfer(recipient, amount)` must succeed and return `true` if

- the `recipient` address is not the zero address,
- `amount` does not exceed the balance of address `msg.sender` ,
- transferring `amount` to the `recipient` address does not lead to an overflow of the recipient's balance, and
- the supplied gas suffices to complete the call. Specification:

```
[](started(contract.transfer(to, value), to != address(0) && to != msg.sender &&
    value >= 0 && value <= _balances[msg.sender] && _balances[to] + value <
    0x10000000000000000000000000000000000000000000000000000000000000000 &&
    _balances[to] >= 0 && _balances[msg.sender] <
    0x10000000000000000000000000000000000000000000000000000000000000000) ==>
  <>(finished(contract.transfer(to, value), return == true)))
```

erc20-transfer-succeed-self

Function `transfer` Succeeds on Admissible Self Transfers. All self-transfers, i.e. invocations of the form `transfer(recipient, amount)` where the `recipient` address equals the address in `msg.sender` must succeed and return `true` if

- the value in `amount` does not exceed the balance of `msg.sender` and
- the supplied gas suffices to complete the call. Specification:

```
[](started(contract.transfer(to, value), to != address(0) && to == msg.sender &&
    value >= 0 && value <= _balances[msg.sender] && _balances[msg.sender] >= 0 &&
    _balances[msg.sender] <
    0x10000000000000000000000000000000000000000000000000000000000000000) ==>
  <>(finished(contract.transfer(to, value), return == true)))
```

erc20-transfer-correct-amount

Function `transfer` Transfers the Correct Amount in Non-self Transfers. All non-reverting invocations of `transfer(recipient, amount)` that return `true` must subtract the value in `amount` from the balance of `msg.sender` and add the same value to the balance of the `recipient` address. Specification:

```
[](willSucceed(contract.transfer(to, value), to != msg.sender && _balances[to] >= 0
    && value >= 0 && _balances[to] + value <
    0x10000000000000000000000000000000000000000000000000000000000000000 &&
    _balances[msg.sender] >= 0 && _balances[msg.sender] <
    0x10000000000000000000000000000000000000000000000000000000000000000) ==>
  <>(finished(contract.transfer(to, value), return == true ==>
      _balances[msg.sender] == old(_balances[msg.sender]) - value && _balances[to]
      == old(_balances[to]) + value)))
```

erc20-transfer-correct-amount-self

Function `transfer` Transfers the Correct Amount in Self Transfers. All non-reverting invocations of `transfer(recipient, amount)` that return `true` and where the `recipient` address equals `msg.sender` (i.e. self-transfers) must not change the balance of address `msg.sender`. Specification:

```
[](willSucceed(contract.transfer(to, value), to == msg.sender && _balances[to] >= 0
    && _balances[to] <
    0x10000000000000000000000000000000000000000000000000000000000000000) ==>
  <>(finished(contract.transfer(to, value), return == true ==> _balances[to] ==
      old(_balances[to]))))
```

erc20-transfer-change-state

Function `transfer` Has No Unexpected State Changes. All non-reverting invocations of `transfer(recipient, amount)` that return `true` must only modify the balance entries of the `msg.sender` and the `recipient` addresses. Specification:

```
[](willSucceed(contract.transfer(to, value), p1 != msg.sender && p1 != to) ==>
  <>(finished(contract.transfer(to, value), return == true ==> (_totalSupply ==
        old(_totalSupply) && _allowances == old(_allowances) && _balances[p1] ==
        old(_balances[p1]) && other_state_variables ==
        old(other_state_variables)))))
```

erc20-transfer-exceed-balance

Function `transfer` Fails if Requested Amount Exceeds Available Balance. Any transfer of an amount of tokens that exceeds the balance of `msg.sender` must fail. Specification:

```
[](started(contract.transfer(to, value), value > _balances[msg.sender] &&
    _balances[msg.sender] >= 0 && value <
    0x10000000000000000000000000000000000000000000000000000000000000000) ==>
  <>(reverted(contract.transfer) || finished(contract.transfer(to, value), return
      == false)))
```

erc20-transfer-recipient-overflow

Function `transfer` Prevents Overflows in the Recipient's Balance. Any invocation of `transfer(recipient, amount)` must fail if it causes the balance of the `recipient` address to overflow. Specification:

```
[](started(contract.transfer(to, value), to != msg.sender && _balances[to] + value
    >= 0x10000000000000000000000000000000000000000000000000000000000000000 &&
    _balances[to] >= 0 && _balances[to] <
    0x10000000000000000000000000000000000000000000000000000000000000000 &&
    _balances[msg.sender] <
    0x10000000000000000000000000000000000000000000000000000000000000000 && value >
    0 && value <= _balances[msg.sender]) ==> <>(reverted(contract.transfer) ||
    finished(contract.transfer(to, value), return == false) ||
    finished(contract.transfer(to, value), _balances[to] > old(_balances[to]) +
      value -
      0x10000000000000000000000000000000000000000000000000000000000000000)))
```

erc20-transfer-false

If Function `transfer` Returns `false`, the Contract State Has Not Been Changed. If the `transfer` function in contract `contract` fails by returning `false`, it must undo all state changes it incurred before returning to the caller. Specification:

```
[](willSucceed(contract.transfer(to, value)) ==> <>(finished(contract.transfer(to,
        value), return == false ==> (_balances == old(_balances) && _totalSupply ==
        old(_totalSupply) && _allowances == old(_allowances) &&
        other_state_variables == old(other_state_variables)))))
```

erc20-transfer-never-return-false

Function `transfer` Never Returns `false`. The transfer function must never return `false` to signal a failure. Specification:

```
[](!(finished(contract.transfer, return == false)))
```

Properties related to function `transferFrom`

erc20-transferfrom-revert-from-zero

Function `transferFrom` Fails for Transfers From the Zero Address. All calls of the form `transferFrom(from, dest, amount)` where the `from` address is zero, must fail. Specification:

```
[](started(contract.transferFrom(from, to, value), from == address(0)) ==>
  <>(reverted(contract.transferFrom) || finished(contract.transferFrom, return ==
      false)))
```

erc20-transferfrom-revert-to-zero

Function `transferFrom` Fails for Transfers To the Zero Address. All calls of the form `transferFrom(from, dest, amount)` where the `dest` address is zero, must fail. Specification:

```
[](started(contract.transferFrom(from, to, value), to == address(0)) ==>
  <>(reverted(contract.transferFrom) || finished(contract.transferFrom, return ==
      false)))
```

erc20-transferfrom-succeed-normal

Function `transferFrom` Succeeds on Admissible Non-self Transfers. All invocations of `transferFrom(from, dest, amount)` must succeed and return `true` if

- the value of `amount` does not exceed the balance of address `from` ,
- the value of `amount` does not exceed the allowance of `msg.sender` for address `from` ,
- transferring a value of `amount` to the address in `dest` does not lead to an overflow of the recipient's balance, and
- the supplied gas suffices to complete the call. Specification:

```
[](started(contract.transferFrom(from, to, value), from != address(0) && to !=
    address(0) && from != to && value <= _balances[from] && value <=
    _allowances[from][msg.sender] && _balances[to] + value <
    0x10000000000000000000000000000000000000000000000000000000000000000 && value >=
    0 && _balances[to] >= 0 && _balances[from] >= 0 && _balances[from] <
    0x10000000000000000000000000000000000000000000000000000000000000000 &&
    _allowances[from][msg.sender] >= 0 && _allowances[from][msg.sender] <
    0x10000000000000000000000000000000000000000000000000000000000000000) ==>
  <>(finished(contract.transferFrom(from, to, value), return == true)))
```

erc20-transferfrom-succeed-self

Function `transferFrom` Succeeds on Admissible Self Transfers. All invocations of `transferFrom(from, dest, amount)` where the `dest` address equals the `from` address (i.e. self-transfers) must succeed and return `true` if:

- The value of `amount` does not exceed the balance of address `from` ,
- the value of `amount` does not exceed the allowance of `msg.sender` for address `from` , and
- the supplied gas suffices to complete the call. Specification:

```
[](started(contract.transferFrom(from, to, value), from != address(0) && from == to
    && value <= _balances[from] && value <= _allowances[from][msg.sender] && value
    >= 0 && _balances[from] <
    0x10000000000000000000000000000000000000000000000000000000000000000 &&
    _allowances[from][msg.sender] <
    0x10000000000000000000000000000000000000000000000000000000000000000) ==>
  <>(finished(contract.transferFrom(from, to, value), return == true)))
```

erc20-transferfrom-correct-amount

Function `transferFrom` Transfers the Correct Amount in Non-self Transfers. All invocations of `transferFrom(from, dest, amount)` that succeed and that return `true` subtract the value in `amount` from the balance of address `from` and add the same value to the balance of address `dest` . Specification:

```
[](willSucceed(contract.transferFrom(from, to, value), from != to && value >= 0 &&
    _balances[from] >= 0 && _balances[from] <
    0x10000000000000000000000000000000000000000000000000000000000000000 &&
    _balances[to] >= 0 && _balances[to] + value <
    0x10000000000000000000000000000000000000000000000000000000000000000) ==>
  <>(finished(contract.transferFrom(from, to, value), return == true ==>
      _balances[from] == old(_balances[from]) - value && _balances[to] ==
      old(_balances[to] + value))))
```

erc20-transferfrom-correct-amount-self

Function `transferFrom` Performs Self Transfers Correctly. All non-reverting invocations of `transferFrom(from, dest, amount)` that return `true` and where the address in `from` equals the address in `dest` (i.e. self-transfers) do not change the balance entry of the `from` address (which equals `dest`). Specification:

```
[](willSucceed(contract.transferFrom(from, to, value), from == to && value >= 0 &&
    value < 0x10000000000000000000000000000000000000000000000000000000000000000 &&
    _balances[from] >= 0 && _balances[from] <
    0x10000000000000000000000000000000000000000000000000000000000000000) ==>
  <>(finished(contract.transferFrom(from, to, value), return == true ==>
      _balances[from] == old(_balances[from]))))
```

erc20-transferfrom-correct-allowance

Function `transferFrom` Updated the Allowance Correctly. All non-reverting invocations of `transferFrom(from, dest, amount)` that return `true` must decrease the allowance for address `msg.sender` over address `from` by the value in `amount` . Specification:

```
[](willSucceed(contract.transferFrom(from, to, value), value >= 0 && value <
    0x10000000000000000000000000000000000000000000000000000000000000000 &&
    _balances[from] >= 0 && _balances[from] <
    0x10000000000000000000000000000000000000000000000000000000000000000 &&
    _balances[to] >= 0 && _balances[to] <
    0x10000000000000000000000000000000000000000000000000000000000000000 &&
    _allowances[from][msg.sender] >= 0 && _allowances[from][msg.sender] <
    0x10000000000000000000000000000000000000000000000000000000000000000) ==>
  <>(finished(contract.transferFrom(from, to, value), return == true ==>
      ((_allowances[from][msg.sender] == old(_allowances[from][msg.sender]) -
      value) || (_allowances[from][msg.sender] ==
      old(_allowances[from][msg.sender]) && (from == msg.sender ||
        old(_allowances[from][msg.sender]) ==
        0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF))))))
```

erc20-transferfrom-change-state

Function `transferFrom` Has No Unexpected State Changes. All non-reverting invocations of `transferFrom(from, dest, amount)` that return `true` may only modify the following state variables:

- The balance entry for the address in `dest` ,
- The balance entry for the address in `from` ,
- The allowance for the address in `msg.sender` for the address in `from` . Specification:

```
[](willSucceed(contract.transferFrom(from, to, amount), p1 != from && p1 != to &&
    (p2 != from || p3 != msg.sender)) ==> <>(finished(contract.transferFrom(from,
      to, amount), return == true ==> (_totalSupply == old(_totalSupply) &&
      _balances[p1] == old(_balances[p1]) && _allowances[p2][p3] ==
      old(_allowances[p2][p3]) && other_state_variables ==
      old(other_state_variables)))))
```

erc20-transferfrom-fail-exceed-balance

Function `transferFrom` Fails if the Requested Amount Exceeds the Available Balance. Any call of the form `transferFrom(from, dest, amount)` with a value for `amount` that exceeds the balance of address `from` must fail. Specification:

```
[](started(contract.transferFrom(from, to, value), value > _balances[from] &&
    _balances[from] >= 0 && _balances[from] <
    0x10000000000000000000000000000000000000000000000000000000000000000) ==>
  <>(reverted(contract.transferFrom) || finished(contract.transferFrom, return ==
      false)))
```

erc20-transferfrom-fail-exceed-allowance

Function `transferFrom` Fails if the Requested Amount Exceeds the Available Allowance. Any call of the form `transferFrom(from, dest, amount)` with a value for `amount` that exceeds the allowance of address `msg.sender` must fail. Specification:

```
[](started(contract.transferFrom(from, to, value), value >
    _allowances[from][msg.sender] && _allowances[from][msg.sender] >= 0 && value <
    0x10000000000000000000000000000000000000000000000000000000000000000) ==>
  <>(reverted(contract.transferFrom) || finished(contract.transferFrom(from, to,
        value), return == false) || finished(contract.transferFrom(from, to,
        value), return == true && (msg.sender == from ||
        _allowances[from][msg.sender] ==
        0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF))))
```

erc20-transferfrom-fail-recipient-overflow

Function `transferFrom` Prevents Overflows in the Recipient's Balance. Any call of `transferFrom(from, dest, amount)` with a value in `amount` whose transfer would cause an overflow of the balance of address `dest` must fail. Specification:

```
[](started(contract.transferFrom(from, to, value), from != to && _balances[to] +
    value >= 0x10000000000000000000000000000000000000000000000000000000000000000 &&
    value < 0x10000000000000000000000000000000000000000000000000000000000000000 &&
    _balances[to] >= 0 && _balances[to] <
    0x10000000000000000000000000000000000000000000000000000000000000000) ==>
  <>(reverted(contract.transferFrom) || finished(contract.transferFrom(from, to,
        value), return == false) || finished(contract.transferFrom(from, to,
        value), _balances[to] > old(_balances[to]) + value -
      0x10000000000000000000000000000000000000000000000000000000000000000)))
```

erc20-transferfrom-false

If Function `transferFrom` Returns `false`, the Contract's State Has Not Been Changed. If `transferFrom` returns `false` to signal a failure, it must undo all incurred state changes before returning to the caller. Specification:

```
[](willSucceed(contract.transferFrom(from, to, value)) ==>
  <>(finished(contract.transferFrom(from, to, value), return == false ==>
    (_balances == old(_balances) && _totalSupply == old(_totalSupply) &&
    _allowances == old(_allowances) && other_state_variables ==
    old(other_state_variables)))))
```

erc20-transferfrom-never-return-false

Function `transferFrom` Never Returns `false`. The `transferFrom` function must never return `false`. Specification:

```
[](!(finished(contract.transferFrom, return == false)))
```

Properties related to function `totalSupply`

erc20-totalsupply-succeed-always

Function `totalSupply` Always Succeeds. The function `totalSupply` must always succeeds, assuming that its execution does not run out of gas. Specification:

```
[](started(contract.totalSupply) ==> <>(finished(contract.totalSupply)))
```

erc20-totalsupply-correct-value

Function `totalSupply` Returns the Value of the Corresponding State Variable. The `totalSupply` function must return the value that is held in the corresponding state variable of contract contract. Specification:

```
[](willSucceed(contract.totalSupply) ==> <>(finished(contract.totalSupply, return
      == _totalSupply)))
```

erc20-totalsupply-change-state

Function `totalSupply` Does Not Change the Contract's State. The `totalSupply` function in contract contract must not change any state variables. Specification:

```
[](willSucceed(contract.totalSupply) ==> <>(finished(contract.totalSupply,
      _totalSupply == old(_totalSupply) && _balances == old(_balances) &&
      _allowances == old(_allowances) && other_state_variables ==
      old(other_state_variables))))
```

Properties related to function `balanceOf`

erc20-balanceof-succeed-always

Function `balanceOf` Always Succeeds. Function `balanceOf` must always succeed if it does not run out of gas. Specification:

```
[](started(contract.balanceOf) ==> <>(finished(contract.balanceOf)))
```

erc20-balanceof-correct-value

Function `balanceOf` Returns the Correct Value. Invocations of `balanceOf(owner)` must return the value that is held in the contract's balance mapping for address `owner` . Specification:

```
[](willSucceed(contract.balanceOf) ==> <>(finished(contract.balanceOf(owner),
      return == _balances[owner])))
```

erc20-balanceof-change-state

Function `balanceOf` Does Not Change the Contract's State. Function `balanceOf` must not change any of the contract's state variables. Specification:

```
[](willSucceed(contract.balanceOf) ==> <>(finished(contract.balanceOf(owner),
      _totalSupply == old(_totalSupply) && _balances == old(_balances) &&
      _allowances == old(_allowances) && other_state_variables ==
      old(other_state_variables))))
```

Properties related to function `allowance`

erc20-allowance-succeed-always

Function `allowance` Always Succeeds. Function `allowance` must always succeed, assuming that its execution does not run out of gas. Specification:

```
[](started(contract.allowance) ==> <>(finished(contract.allowance)))
```

erc20-allowance-correct-value

Function `allowance` Returns Correct Value. Invocations of `allowance(owner, spender)` must return the allowance that address `spender` has over tokens held by address `owner`. Specification:

```
[](willSucceed(contract.allowance(owner, spender)) ==>
  <>(finished(contract.allowance(owner, spender), return ==
    _allowances[owner][spender])))
```

erc20-allowance-change-state

Function `allowance` Does Not Change the Contract's State. Function `allowance` must not change any of the contract's state variables. Specification:

```
[](willSucceed(contract.allowance(owner, spender)) ==>
  <>(finished(contract.allowance(owner, spender), _totalSupply == old(_totalSupply)
    && _balances == old(_balances) && _allowances == old(_allowances) &&
    other_state_variables == old(other_state_variables))))
```

Properties related to function `approve`

erc20-approve-revert-zero

Function `approve` Prevents Giving Approvals For the Zero Address. All calls of the form `approve(spender, amount)` must fail if the address in `spender` is the zero address. Specification:

```
[](started(contract.approve(spender, value), spender == address(0)) ==>
  <>(reverted(contract.approve) || finished(contract.approve(spender, value),
    return == false)))
```

erc20-approve-succeed-normal

Function `approve` Succeeds for Admissible Inputs. All calls of the form `approve(spender, amount)` must succeed, if

- the address in `spender` is not the zero address and
- the execution does not run out of gas. Specification:

```
[](started(contract.approve(spender, value), spender != address(0)) ==>
  <>(finished(contract.approve(spender, value), return == true)))
```

erc20-approve-correct-amount

Function `approve` Updates the Approval Mapping Correctly. All non-reverting calls of the form `approve(spender, amount)` that return `true` must correctly update the allowance mapping according to the address `msg.sender` and the values of `spender` and `amount`. Specification:

```
[](willSucceed(contract.approve(spender, value), spender != address(0) && value >=
    0 && value <
    0x10000000000000000000000000000000000000000000000000000000000000000) ==>
  <>(finished(contract.approve(spender, value), return == true ==>
     _allowances[msg.sender][spender] == value)))
```

erc20-approve-change-state

Function `approve` Has No Unexpected State Changes. All calls of the form `approve(spender, amount)` must only update the allowance mapping according to the address `msg.sender` and the values of `spender` and `amount` and incur no other state changes. Specification:

```
[](willSucceed(contract.approve(spender, value), spender != address(0) && (p1 !=
      msg.sender || p2 != spender)) ==> <>(finished(contract.approve(spender,
        value), return == true ==> _totalSupply == old(_totalSupply) && _balances
      == old(_balances) && _allowances[p1][p2] == old(_allowances[p1][p2]) &&
      other_state_variables == old(other_state_variables))))
```

erc20-approve-false

If Function `approve` Returns `false`, the Contract's State Has Not Been Changed. If function `approve` returns `false` to signal a failure, it must undo all state changes that it incurred before returning to the caller. Specification:

```
[](willSucceed(contract.approve(spender, value)) ==>
  <>(finished(contract.approve(spender, value), return == false ==> (_balances ==
      old(_balances) && _totalSupply == old(_totalSupply) && _allowances ==
      old(_allowances) && other_state_variables == old(other_state_variables)))))
```

erc20-approve-never-return-false

Function `approve` Never Returns `false` . The function `approve` must never returns `false` . Specification:

```
[](!(finished(contract.approve, return == false)))
```

Description of ERC-721 Properties

The specifications are designed such that they capture the desired and admissible behaviors of the ERC-721 functions `transferFrom` , `balanceOf` , `ownerOf` , `getApproved` , `isApprovedForAll` , `approve` , `setApprovalForAll` `supportsInterface` , `tokenURI` , `tokenByIndex` , `tokenByIndex` , `decimals` and `totalSupply` . In the following, we list those property specifications.

Properties related to function `transferFrom`

erc721-transferfrom-succeed-normal

Function `transferFrom` Succeeds on Admissible Inputs. All invocations of `transferFrom(from, to, tokenId)` must succeed if

- address `from` is the owner of token `tokenId` ,
- the sender is approved to transfer token `tokenId` ,
- transferring the token to the address `to` does not lead to an overflow of the recipient's balance, and
- the supplied gas suffices to complete the call. Specification:

```
[](started(contract.transferFrom(from, to, tokenId), from != address(0) && to !=
    address(0) && _owner[tokenId]==from && ((from == msg.sender) ||
      (_approved[tokenId] == msg.sender) || _approvedAll[from][msg.sender]) &&
  _balances[to] >= 0 && _balances[from] >= 1 && _balances[to] <
  0x10000000000000000000000000000000000000000000000000000000000000000 - 1 &&
  _balances[from] <
  0x10000000000000000000000000000000000000000000000000000000000000000) ==> <>
finished(contract.transferFrom(from, to, tokenId)))
```

erc721-transferfrom-correct-increase

Function `transferFrom` Transfers the Complete Token in Non-self Transfers. All invocations of `transferFrom(from, to, tokenId)` that succeed must subtract a token from the balance of address `from` and add the token to the balance of address `to` . Specification:

```
[](willSucceed(contract.transferFrom(from, to, tokenId), from != to &&
    _balances[from] > 0 && _balances[from] <
    0x10000000000000000000000000000000000000000000000000000000000000000 &&
    _balances[to] <
    0x10000000000000000000000000000000000000000000000000000000000000000 - 1) ==>
  <>(finished(contract.transferFrom(from, to, tokenId), _balances[from] ==
      (old(_balances[from]) - 1) && _balances[to] == (old(_balances[to]) + 1))))
```

erc721-transferfrom-correct-one-token-self

Function `transferFrom` Performs Self Transfers Correctly. All non-reverting invocations of `transferFrom(from, to, tokenId)` that return `true` and where the address `from` equals the address `to` (i.e. self-transfers) must not change the balance entry of the address `from` (which equals `to`). Specification:

```
[](willSucceed(contract.transferFrom(from, to, tokenId), from == to &&
    _owner[tokenId] == from) ==> <>(finished(contract.transferFrom(from, to,
        tokenId), _balances[from] == old(_balances[from]))))
```

erc721-transferfrom-correct-approval

Function `transferFrom` Updates the Approval Correctly. All non-reverting invocations of `transferFrom(from, to, tokenId)` that return must remove any approval for token `tokenId`. Specification:

```
[](willSucceed(contract.transferFrom(from, to, tokenId), p1 != address(0)) ==>
  <>(finished(contract.transferFrom(from, to, tokenId), (_approved[tokenId] !=
        p1))))
```

erc721-transferfrom-correct-owner-from

Function `transferFrom` Removes Token Ownership of From. All non-reverting and non-self invocations of `transferFrom(from, to, tokenId)` that return, must remove the ownership of token `tokenId` from address `from`. Specification:

```
[](willSucceed(contract.transferFrom(from, to, tokenId), from != to && from !=
    address(0) && to != address(0) && (msg.sender==from ||
      _approved[tokenId]==msg.sender || _approvedAll[from][msg.sender])) ==>
  <>(finished(contract.transferFrom(from, to, tokenId), (_owner[tokenId] !=
        from))))
```

erc721-transferfrom-correct-owner-to

Function `transferFrom` Transfers Ownership. All non-reverting invocations of `transferFrom(from, to, tokenId)` must transfer the ownership of token `tokenId` to the address `to`. Specification:

```
[](willSucceed(contract.transferFrom(from, to, tokenId), from != address(0) && to
    != address(0) && _balances[from] >= 0 && _balances[from] <
    0x10000000000000000000000000000000000000000000000000000000000000000 &&
    _balances[to] >= 0 && _balances[to] <
    0x10000000000000000000000000000000000000000000000000000000000000000 &&
    (msg.sender==from || _approved[tokenId]==msg.sender ||
    _approvedAll[from][msg.sender])) ==> <>(finished(contract.transferFrom(from,
      to, tokenId), (_owner[tokenId] == to))))
```

erc721-transferfrom-correct-balance

Function `transferFrom` Sum of Balances is Constant. All non-reverting invocations of `transferFrom(from, to, tokenId)` must keep the sum of token balances constant. Specification:

```
[](willSucceed(contract.transferFrom(from, to, tokenId), from!=address(0) &&
    _balances[from]>0 && to!=address(0) && _balances[from] <
    0x10000000000000000000000000000000000000000000000000000000000000000 &&
    _balances[to] <
    0x10000000000000000000000000000000000000000000000000000000000000000 ) ==>
  <>(finished(contract.transferFrom(from, to, tokenId),
      (old(_balances[from])-_balances[from]) ==
    (_balances[to]-old(_balances[to])))))
```

erc721-transferfrom-correct-state-balance

Function `transferFrom` Keeps Balances Constant Except for From and To. All non-reverting invocations of `transferFrom(from, to, tokenId)` must only modify the balance of the addresses `from` and `to`. Specification:

```
[](willSucceed(contract.transferFrom(from, to, tokenId), p1 != from && p1 != to )
  ==> <>(finished(contract.transferFrom(from, to, tokenId), _balances[p1] ==
      old(_balances[p1]))))
```

erc721-transferfrom-correct-state-owner

Function `transferFrom` Has Expected Ownership Changes. All non-reverting invocations of `transferFrom(from, to, tokenId)` must only modify the ownership of token `tokenId`. Specification:

```
[](willSucceed(contract.transferFrom(from, to, tokenId), t1 != tokenId) ==>
  <>(finished(contract.transferFrom(from, to, tokenId), _owner[t1] ==
      old(_owner[t1]) && _owner[t1] == old(_owner[t1]))))
```

erc721-transferfrom-correct-state-approval

Function `transferFrom` Has Expected Approval Changes. All non-reverting invocations of `transferFrom(from, to, tokenId)` must remove only approvals for token `tokenId` Specification:

```
[](willSucceed(contract.transferFrom(from, to, tokenId), t1 != tokenId) ==>
  <>(finished(contract.transferFrom(from, to, tokenId), _approved[t1] ==
      old(_approved[t1]))))
```

erc721-transferfrom-revert-invalid

Function `transferFrom` Fails for Invalid Tokens. All calls of the form `transferFrom(from, to, tokenId)` must fail for any invalid token. Specification:

```
[](started(contract.transferFrom(from, to, tokenId), _owner[tokenId] == address(0))
  ==> <>(reverted(contract.transferFrom)))
```

erc721-transferfrom-revert-from-zero

Function `transferFrom` Fails for Transfers From the Zero Address. All calls of the form `transferFrom(from, to, tokenId)` must fail if the `from` address is zero. Specification:

```
[](started(contract.transferFrom(from, to, tokenId), from == address(0)) ==>
  <>(reverted(contract.transferFrom(from, to, tokenId))))
```

erc721-transferfrom-revert-to-zero

Function `transferFrom` Fails for Transfers To the Zero Address. All calls of the form `transferFrom(from, to, tokenId)` must fail if the address `to` is the zero address. Specification:

```
[](started(contract.transferFrom(from, to, tokenId), to == address(0)) ==>
  <>(reverted(contract.transferFrom(from, to, tokenId))))
```

erc721-transferfrom-revert-not-owned

Function `transferFrom` Fails if `From` Is Not Token Owner. Any call of the form `transferFrom(from, to, tokenId)` must fail if address 'from' is not the owner of token `tokenId` . Specification:

```
[](started(contract.transferFrom(from, to, tokenId), _owner[tokenId]!= from) ==>
  <>(reverted(contract.transferFrom)))
```

erc721-transferfrom-revert-exceed-approval

Function `transferFrom` Fails for Token Transfers without Approval. Any call of the form `transferFrom(from, to, tokenId)` must fail if the sender is neither the token owner nor an operator of the token owner nor approved for token `tokenId` . Specification:

```
[](started(contract.transferFrom(from, to, tokenId), msg.sender!=from &&
    _approved[tokenId]!=msg.sender && !_approvedAll[from][msg.sender]) ==>
  <>(reverted(contract.transferFrom)))
```

Properties related to function `supportsInterface`

erc721-supportsinterface-correct-erc721

Function `supportsInterface` Signals that the Contract Supports `ERC721`. Invocations of `supportsInterface(id)` must signal that the interface `ERC721` is implemented. Specification:

```
[](willSucceed(contract.supportsInterface(id), id==0x80ac58cd) ==> <>
  finished(contract.supportsInterface(id), return==true))
```

erc721-supportsinterface-metadata

Function `supportsInterface` Returns that Interface ERC721Metadata Implemented. A call of `supportsInterface(interfaceId)` with the interface id of ERC721Metadata must return true. Specification:

```
[](willSucceed(contract.supportsInterface(interfaceId), interfaceId==0x5b5e139f)
  ==> <> finished(contract.supportsInterface(interfaceId), return==true))
```

erc721-supportsinterface-succeed-always

Function `supportsInterface` Always Succeeds. Function `supportsInterface` must always succeed if it does not run out of gas. Specification:

```
[](started(contract.supportsInterface(id)) ==> <>
  finished(contract.supportsInterface(id)))
```

erc721-supportsinterface-correct-erc165

Function `supportsInterface` Signals that the Contract Supports ERC165. Invocations of `supportsInterface(id)` must signal that the interface `ERC165` is implemented. Specification:

```
[](willSucceed(contract.supportsInterface(id), id==0x01ffc9a7) ==> <>
  finished(contract.supportsInterface(id), return==true))
```

erc721-supportsinterface-correct-false

Function `supportsInterface` Returns `False` for Id 0xffffffff. Invocations of `supportsInterface(id)` with `id` 0xffffffff must return `false`. Specification:

```
[](willSucceed(contract.supportsInterface(id), id==0xffffffff) ==> <>
  finished(contract.supportsInterface(id), return==false))
```

erc721-supportsinterface-no-change-state

Function `supportsInterface` Does Not Change the Contract's State. Function `supportsInterface` must not change any of the contract's state variables. Specification:

```
[](willSucceed(contract.supportsInterface(id)) ==>
  <>(finished(contract.supportsInterface(id), other_state_variables ==
    old(other_state_variables))))
```

Properties related to function `balanceOf`

erc721-balanceof-succeed-normal

Function `balanceOf` Succeeds on Admissible Inputs. All invocations of `balanceOf(owner)` must succeed if the address `owner` is not zero and it does not run out of gas. Specification:

```
[](started(contract.balanceOf(owner), owner!=address(0)) ==>
  <>(finished(contract.balanceOf)))
```

erc721-balanceof-correct-count

Function `balanceOf` Returns the Correct Value. Invocations of `balanceOf(owner)` must return the value that is held in the balance mapping for address `owner`. Specification:

```
[](willSucceed(contract.balanceOf) ==> <>(finished(contract.balanceOf(owner),
      return == _balances[owner])))
```

erc721-balanceof-revert

Function `balanceOf` Fails on the Zero Address. Invocations of `balanceOf(owner)` must fail if the address `owner` is the zero address. Specification:

```
[](started(contract.balanceOf(owner), owner==address(0)) ==>
  <>(reverted(contract.balanceOf(owner))))
```

erc721-balanceof-no-change-state

Function `balanceOf` Does Not Change the Contract's State. Function `balanceOf` must not change any of the contract's state variables. Specification:

```
[](willSucceed(contract.balanceOf) ==> <>(finished(contract.balanceOf, _balances ==
      old(_balances) && other_state_variables == old(other_state_variables))))
```

Properties related to function `ownerOf`

erc721-ownerof-succeed-normal

Function `ownerOf` Succeeds For Valid Tokens. Function `ownerOf(token)` must always succeed for valid tokens if it does not run out of gas. Specification:

```
[](started(contract.ownerOf(token), _owner[token]!=address(0)) ==>
  <>(finished(contract.ownerOf)))
```

erc721-ownerof-correct-owner

Function `ownerOf` Returns the Correct Owner. Invocations of `ownerOf(token)` must return the owner for a valid token `token` that is held in the contract's owner mapping. Specification:

```
[](willSucceed(contract.ownerOf(token), _owner[token]!=address(0)) ==>
  <>(finished(contract.ownerOf(token), return == _owner[token])))
```

erc721-ownerof-revert

Function `ownerOf` Fails On Invalid Tokens. Invocations of `ownerOf(token)` must fail for an invalid token. Specification:

```
[](started(contract.ownerOf(token), _owner[token]==address(0)) ==>
  <>(reverted(contract.ownerOf(token))))
```

erc721-ownerof-no-change-state

Function `ownerOf` Does Not Change the Contract's State. Function `ownerOf` must not change any of the contract's state variables. Specification:

```
[](willSucceed(contract.ownerOf) ==> <>(finished(contract.ownerOf, _owner ==
      old(_owner) && other_state_variables == old(other_state_variables))))
```

Properties related to function `getApproved`

erc721-getapproved-succeed-normal

Function `getApproved` Succeeds For Valid Tokens. Function `getApproved` must always succeed for valid tokens, assuming that its execution does not run out of gas. Specification:

```
[](started(contract.getApproved(token), _owner[token]!=address(0)) ==>
  <>(finished(contract.getApproved)))
```

erc721-getapproved-correct-value

Function `getApproved` Returns Correct Approved Address. Invocations of `getApproved(token)` must return the approved address of a valid `token` . Specification:

```
[](willSucceed(contract.getApproved(token)) ==>
  <>(finished(contract.getApproved(token), return == _approved[token] || return ==
    address(0))))
```

erc721-getapproved-revert-zero

Function `getApproved` Fails on Invalid Tokens. Invocations of `getApproved(token)` with an invalid token must fail. Specification:

```
[](started(contract.getApproved(token), _owner[token]==address(0)) ==>
  <>(reverted(contract.getApproved)))
```

erc721-getapproved-change-state

Function `getApproved` Does Not Change the Contract's State. Function `getApproved` must not change any of the contract's state variables. Specification:

```
[](willSucceed(contract.getApproved) ==> <>(finished(contract.getApproved,
      _approved == old(_approved) && other_state_variables ==
      old(other_state_variables))))
```

Properties related to function `isApprovedForAll`

erc721-isapprovedforall-succeed-normal

Function `isApprovedForAll` Always Succeeds. Function `isApprovedForAll` does always succeed, assuming that its execution does not run out of gas. Specification:

```
[](started(contract.isApprovedForAll(owner, operator)) ==>
  <>(finished(contract.isApprovedForAll)))
```

erc721-isapprovedforall-correct

Function `isApprovedForAll` Returns Correct Approvals. Invocations of `isApprovedForAll(owner, operator)` must return whether a non-zero address `operator` is approved for tokens of a non-zero address `owner`, or return false. Specification:

```
[](willSucceed(contract.isApprovedForAll(owner, operator), owner!=address(0) &&
    operator!=address(0)) ==> <>(finished(contract.isApprovedForAll(owner,
        operator), return == _approvedAll[owner][operator])))
```

erc721-isapprovedforall-correct-false

Function `isApprovedForAll` Returns Non-Approval For Invalid Inputs. Invocations of `isApprovedForAll(owner, operator)` must return `false` if called with any invalid address. Specification:

```
[](started(contract.isApprovedForAll(owner, operator), owner==address(0) ||
    operator==address(0)) ==> <>(finished(contract.isApprovedForAll, return ==
      false)))
```

erc721-isapprovedforall-change-state

Function `isApprovedForAll` Does Not Change the Contract's State. Function `isApprovedForAll` does not change any of the contract's state variables. Specification:

```
[](willSucceed(contract.isApprovedForAll) ==>
  <>(finished(contract.isApprovedForAll, _approvedAll == old(_approvedAll) &&
    other_state_variables == old(other_state_variables))))
```

Properties related to function `approve`

erc721-approve-succeed-normal

Function `approve` Return for Admissible Inputs. All calls of the form `approve(to, tokenId)` must return if

- the sender is the owner or an authorized operator of the owner
- the token `tokenId` is valid and
- the execution does not run out of gas. Specification:

```
[](started(contract.approve(to, tokenId), (_owner[tokenId]!=address(0)) &&
    (_owner[tokenId]==msg.sender || _approvedAll[_owner[tokenId]][msg.sender]) &&
  (_owner[tokenId]!=to)) ==> <>(finished(contract.approve)))
```

erc721-approve-set-correct

Function `approve` Sets Approve. Any returning call of the form `approve(to, tokenId)` must approve the address `to` for token `tokenId`. Specification:

```
[](willSucceed(contract.approve(to, tokenId), (_owner[tokenId]!=address(0)) &&
    (_owner[tokenId]==msg.sender || _approvedAll[_owner[tokenId]][msg.sender])) ==>
<>(finished(contract.approve(to, tokenId), _approved[tokenId]==to)))
```

erc721-approve-revert-not-allowed

Function `approve` Prevents Unpermitted Approvals. All calls of the form `approve(to, tokenId)` must fail if the message sender is not permitted to access token `tokenId`. Specification:

```
[](started(contract.approve(to, tokenId), _owner[tokenId]!=msg.sender &&
    !_approvedAll[_owner[tokenId]][msg.sender]) ==> <>(reverted(contract.approve)))
```

erc721-approve-revert-invalid-token

Function `approve` Fails For Calls with Invalid Tokens. All calls of the form `approve(to, tokenId)` must fail for an invalid token. Specification:

```
[](started(contract.approve(to, tokenId), _owner[tokenId] == address(0)) ==>
  <>(reverted(contract.approve)))
```

erc721-approve-change-state

Function `approve` Has No Unexpected State Changes. All calls of the form `approve(to, tokenId)` must only update the allowance mapping according to a valid token `tokenId` and the address `to`, and incur no other state changes. Specification:

```
[](willSucceed(contract.approve(approved, tokenId), t1!=tokenId) ==>
  <>(finished(contract.approve(approved, tokenId),
    _approved[t1]==old(_approved[t1]) && other_state_variables ==
    old(other_state_variables))))
```

Properties related to function `setApprovalForAll`

erc721-setapprovalforall-succeed-normal

Function `setApprovalForAll` Return for Admissible Inputs. Calls of the form `setApprovalForAll(operator, approved)` must return if

- the message sender is not the `operator`,
- `operator` is not the zero address and
- the execution does not run out of gas. Specification:

```
[](started(contract.setApprovalForAll(operator, approved), (msg.sender!=operator)
    && (operator!=address(0))) ==> <>(finished(contract.setApprovalForAll)))
```

erc721-setapprovalforall-set-correct

Function `setApprovalForAll` Approves Operator. All non-reverting calls of the form `setApprovalForAll(operator, approved)` must set the approval of a non-zero address `operator` according to the Boolean value `approved`. Specification:

```
[](willSucceed(contract.setApprovalForAll(operator, approved),
    operator!=address(0)) ==> <>(finished(contract.setApprovalForAll(operator,
        approved), _approvedAll[msg.sender][operator]==approved)))
```

erc721-setapprovalforall-multiple

Function `setApprovalForAll` Can Set Multiple Operators. Calls of the form `setApprovalForAll(operator, approved)` must be able to set multiple operators for the tokens of the message sender. Specification:

```
[](willSucceed(contract.setApprovalForAll(operator, approved), op1!=address(0) &&
    approved && _approvedAll[msg.sender][op1]  ) ==>
  <>(finished(contract.setApprovalForAll(operator, approved),
    _approvedAll[msg.sender][operator] && _approvedAll[msg.sender][op1])))
```

erc721-setapprovalforall-revert-zero

Function `setApprovalForAll` Prevents Giving Approvals to the Zero Address. All calls of the form `setApprovalForAll(operator, approved)` must fail if the address `operator` is the zero address. Specification:

```
[](started(contract.setApprovalForAll(operator, approved), operator == address(0))
  ==> <>(reverted(contract.setApprovalForAll)))
```

erc721-setapprovalforall-change-state

Function `setApprovalForAll` Has No Unexpected State Changes. All calls of the form `setApprovalForAll(operator, approved)` must only update the approval mapping according to the message sender, the address `operator` and the Boolean value `approved` but incur no other state changes. Specification:

```
[](started(contract.setApprovalForAll(op, approved), ow1!=msg.sender || op1!=op)
  ==> <>(finished(contract.setApprovalForAll(op, approved),
      _approvedAll[ow1][op1]==old(_approvedAll[ow1][op1]) &&
      _approvedAll[msg.sender][op]==approved && other_state_variables ==
      old(other_state_variables)) || reverted(contract.setApprovalForAll(op,
        approved))))
```

DISCLAIMER | CERTIK

CertiK | Securing the Web3 World

Founded in 2017 by leading academics in the field of Computer Science from both Yale and Columbia University, CertiK is a leading blockchain security company that serves to verify the security and correctness of smart contracts and blockchain-based protocols. Through the utilization of our world-class technical expertise, alongside our proprietary, innovative tech, we’re able to support the success of our clients with best-in-class security, all whilst realizing our overarching vision; provable trust for all throughout all facets of blockchain.