82.com

Security Assessment

CertiK Assessed on May 8th, 2025

CertiK Assessed on May 8th, 2025

82.com

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

Executive Summary

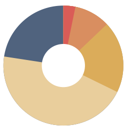
TYPES	ECOSYSTEM	METHODS
NFT	Ethereum (ETH)	Formal Verification, Manual Review, Static Analysis
LANGUAGE	**TIMELINE**	**KEY COMPONENTS**
Solidity	Delivered on 05/08/2025	N/A

CODEBASE	COMMITS
base	5a43734e468fe7c70a6700c0531a8b79f705e4de
update_20250502	22571719e7b3ddfb92a9367af1bdb72352e7e4c4
View All in Codebase Page	View All in Codebase Page

Highlighted Centralization Risks

- Contract upgradeability
- Privileged role can mint tokens
- Fees are unbounded
- Has blacklist/whitelist

Vulnerability Summary



Total Findings	Resolved	Partially Resolved	Acknowledged	Declined
31	11	1	19	0

2	Centralization	2 Acknowledged	Centralization findings highlight privileged roles & functions and their capabilities, or instances where the project takes custody of users' assets.
1	Critical	1 Resolved	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.
1	Major	1 Resolved	Major risks may include logical errors that, under specific circumstances, could result in fund losses or loss of project control.
6	Medium	1 Resolved, 5 Acknowledged	Medium risks may not pose a direct risk to users' funds, but they can affect the overall functioning of a platform.

🟨	14	Minor	7 Resolved, 1 Partially Resolved, 6 Acknowledged	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.
🟦	7	Informational	1 Resolved, 6 Acknowledged	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 | 82.COM

CODEBASE | 82.COM

Repository

base

update_20250502

Commit

5a43734e468fe7c70a6700c0531a8b79f705e4de

22571719e7b3ddfb92a9367af1bdb72352e7e4c4

AUDIT SCOPE | 82.COM

13 files audited ● 13 files with Acknowledged findings

ID	Repo	File	SHA256 Checksum
● DNF	octopus-net/octopus-contract	domainNft/DomainNFTLogic.sol	ec38703e68abea5e60d4b4b3e5ac4ee9f0658eed2e5f97572b1c87c443d61ebc
● DNT	octopus-net/octopus-contract	domainNft/DomainNFTProxy.sol	3f01c20f88b6dddc19ec8125d171690b88f1b6541d7b1a251c9bfddfd13d7ccb
● TWM	octopus-net/octopus-contract	modules/TokenWithdrawModule.sol	3e4630b8e7d4269391b0fdb91362493f34e1b475afc8d6d1cbef31327d30d9e7
● IPC	octopus-net/octopus-contract	safe/IProxyCreationCallback.sol	cacbc0044b6cbd89caa1c16b185fc662f066a85798532ee735835e8c044e9bf3
● SAE	octopus-net/octopus-contract	safe/Safe.sol	b7c49796ce8e61ae1ae740dd5ffcb56d3eda8b685d5f9f9fe8488a6391dfa1b9
● SPB	octopus-net/octopus-contract	safe/SafeProxy.sol	4c23eec50cbf85dd2ddc3b7eb9efd85430bc47c9ad79dc6b98833603fbe09cb0
● SPF	octopus-net/octopus-contract	safe/SafeProxyFactory.sol	8aa936ea0d16eb96c7cd63398697064836f1737d2fc10d5f012eaa0e91b3f8d9
● SML	octopus-net/octopus-contract	setting/SettingManagerLogic.sol	2e18cfb897efc31437badb14b54064c40a65058246cf95786281195daaef025d
● SMP	octopus-net/octopus-contract	setting/SettingManagerProxy.sol	375fe9a1b75f9186b2a09c75a409d6152f529b860a4748829edc2dfe25493d18

ID	Repo	File	SHA256 Checksum
● NFT	octopus-net/octopus-contract	trading/NFTOfferMarketLogic.sol	4bac637a3ba739d348fadc7e291e141a56bcd8dfcb30e7213dde6093dd867f7e
● NFO	octopus-net/octopus-contract	trading/NFTOfferMarketProxy.sol	86d178c8167868cda8fc607645e6742772c89e1eca6f233596ed61eb66cd366b
● TAL	octopus-net/octopus-contract	transfer/TransferAgentLogic.sol	40b7ba2ffa1f0e7c666e27d21bfd7759a9e5357d96654f69a78ace7a5127e033
● TAP	octopus-net/octopus-contract	transfer/TransferAgentProxy.sol	5daa411cce3b4e792a9fe175d8659694c3f47d0fe0d01e9a32dc525c64eb3cf7

APPROACH & METHODS | 82.COM

This report has been prepared for 82.com to discover issues and vulnerabilities in the source code of the 82.com 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.

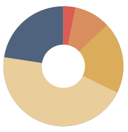
The auditing process pays special attention to the following considerations:

- 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 | 82.COM



31	1	2	1	6	14	7
Total Findings	Critical	Centralization	Major	Medium	Minor	Informational

This report has been prepared to discover issues and vulnerabilities for 82.com. Through this audit, we have uncovered 31 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
COI-05	Unrestricted Public Mint Function Allows Unauthorized Token Creation	Access Control	Critical	● Resolved
COI-06	**Centralized Control Of Contract Upgrade**	**Centralization**	**Centralization**	● **Acknowledged**
COI-07	**Centralization Risks**	**Centralization**	**Centralization**	● **Acknowledged**
COI-09	Potential Front Run In `NFTOfferMarketLogic`	Concurrency	Major	● Resolved
COI-08	Function Calls User-Provided Addresses With No Access Control Modifier	Access Control	Medium	● Acknowledged
COI-10	Lack Of Storage Gap Or NameSpaced Storage Layout In Upgradeable Contract	Design Issue	Medium	● Acknowledged
COI-11	Users Are Forced To Approve For All	Volatile Code	Medium	● Acknowledged
COI-12	Preemptive Freezing Of Non-Existent Tokens Can Block Future Minting	Logical Issue	Medium	● Resolved
COI-13	Ambiguous ERC20 Transfers To Same Recipient May Indicate Misconfigured Fee Logic	Logical Issue	Medium	● Acknowledged
COI-14	Insufficient Safeguards Allow Arbitrary Calls Despite Signature Validation	Logical Issue	Medium	● Acknowledged

ID	Title	Category	Severity	Status
COI-15	Self-Administered Role Allows For Complete Role Takeover	Design Issue	Minor	● Acknowledged
COI-16	Missing Zero Address Validation	Volatile Code	Minor	● Acknowledged
COI-17	Incorrect Setup Of EIP712 Domain Separator	Design Issue	Minor	● Acknowledged
COI-18	`_setRoleAdmin` To `DEFAULT_ADMIN_ROLE` Is Redundant	Logical Issue	Minor	● Resolved
COI-19	Potential Reentrancy Attack (Out-Of-Order Events)	Concurrency	Minor	● Acknowledged
COI-20	`_nextTokenId` Reflects The Current Token ID	Logical Issue	Minor	● Resolved
COI-21	Missing Input Validation In `viewRoleMember()` Leads To Silent Or Unexpected Failures	Logical Issue	Minor	● Partially Resolved
COI-23	Unsafe Unlimited Token Approvals And Operator Permissions Granted To Transfer Agent	Design Issue	Minor	● Acknowledged
COI-24	Unbounded Token Whitelist May Lead To Gas Exhaustion	Denial of Service	Minor	● Acknowledged
COI-25	Stale Token Type Mapping Remains After Token Removal	Logical Issue	Minor	● Resolved
COI-26	Uninitialized Critical Configuration Variables May Lead To Undefined Behavior	Logical Issue	Minor	● Resolved
COI-27	Redundant Check	Volatile Code	Minor	● Resolved
COI-28	Potential CrossFunction Reentrancy In `NFTOfferMarketLogic`	Concurrency	Minor	● Resolved
COI-36	Initialize State Variable In Constructor Or Declaration In Upgradeable Contract	Logical Issue	Minor	● Resolved

ID	Title	Category	Severity	Status
COI-29	Inconsistent Solidity Versions	Language Version	Informational	● Acknowledged
COI-30	Potential Reentrancy Attack (In Case Of Unlimited Gas)	Concurrency	Informational	● Acknowledged
COI-31	Unused State Variable	Coding Issue	Informational	● Acknowledged
COI-32	Local Variable Shadowing	Coding Style	Informational	● Acknowledged
COI-33	Too Many Digits	Magic Numbers	Informational	● Acknowledged
COI-34	Missing Emit Events	Coding Style	Informational	● Acknowledged
COI-35	Information On Upgrade Handling	Design Issue	Informational	● Resolved

COI-05 | UNRESTRICTED PUBLIC MINT FUNCTION ALLOWS UNAUTHORIZED TOKEN CREATION

Category	Severity	Location	Status
Access Control	● Critical	domainNft/DomainNFTLogic.sol (base): 59~64, 69~74	● Resolved

Description

Anyone can call the `mint()` function to create new tokens and assign them to any address, as there are no access controls or restrictions in place to limit who can invoke it. This allows unauthorized users to arbitrarily mint tokens, potentially leading to inflation, abuse, or loss of trust in the system.

Recommendation

We recommend adding proper access control, such as `onlyOwner()` or role-based modifiers, to restrict who can call the mint function.

Alleviation

[82.com, 04/23/2025]: The unrestricted mint vulnerability (COI-5) has been resolved in commit b01d571 through implementation of role-based access control, effectively restricting minting privileges to authorized addresses only.

COI-06 | CENTRALIZED CONTROL OF CONTRACT UPGRADE

Category	Severity	Location	Status
Centralization	● **Centralization**	**domainNft/DomainNFTLogic.sol (base): 11; setting/SettingManagerLogic.sol (base): 8; trading/NFTOfferMarketLogic.sol (base): 218; transfer/TransferAgentLogic.sol (base): 9**	● **Acknowledged**

Description

Upgrade authorization is restricted to roles such as `ROLE_ADMIN` , `DEFAULT_ADMIN_ROLE` , or `onlyOwner` , granting a single account or a small group full control over replacing the contract's implementation. This introduces a centralization risk, as any compromise or misuse of these privileged roles can lead to arbitrary logic being deployed, potentially enabling fund theft, data corruption, or bypassing critical security checks. Without additional safeguards like multi-signature governance, timelocks, or community oversight, the upgrade mechanism becomes a single point of failure.

Risk Note

Important Note: Certain identification procedures were attempted to be applied to the project team in order to better understand the centralization situation and potential risks of the project. We strongly advise end users to conduct further research and exercise due diligence before engaging with the project given the centralization related risks. It is crucial for end users to independently verify and assess all available information

Recommendation

We recommend that the team make efforts to restrict access to the admin of the proxy contract. A strategy of combining a time-lock and a multi-signature (⅔, ⅗) wallet can be used to prevent a single point of failure due to a private key compromise. In addition, the team should be transparent and notify the community in advance whenever they plan to migrate to a new implementation contract.

Here are some feasible short-term and long-term suggestions that would mitigate the potential risk to a different level and suggestions that would permanently fully resolve the risk.

Short Term:

A combination of a time-lock and a multi signature (⅔, ⅗) wallet mitigate the risk by delaying the sensitive operation and avoiding a single point of key management failure.

- A time-lock with reasonable latency, such as 48 hours, for awareness of privileged operations;
  AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to a private key compromised;

AND

- A medium/blog link for sharing the time-lock contract and multi-signers addresses information with the community.

For remediation and mitigated status, please provide the following information:

- Provide the deployed time-lock address.
- Provide the **gnosis** address with **ALL** the multi-signer addresses for the verification process.
- Provide a link to the **medium/blog** with all of the above information included.

Long Term:

A combination of a time-lock on the contract upgrade operation and a DAO for controlling the upgrade operation mitigate the contract upgrade risk by applying transparency and decentralization.

- A time-lock with reasonable latency, such as 48 hours, for community awareness of privileged operations;
  AND
- Introduction of a DAO, governance, or voting module to increase decentralization, transparency, and user involvement;
  AND
- A medium/blog link for sharing the time-lock contract, multi-signers addresses, and DAO information with the community.

For remediation and mitigated status, please provide the following information:

- Provide the deployed time-lock address.
- Provide the **gnosis** address with **ALL** the multi-signer addresses for the verification process.
- Provide a link to the **medium/blog** with all of the above information included.

Permanent:

Renouncing ownership of the `admin` account or removing the upgrade functionality can *fully* resolve the risk.

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

Note: we recommend the project team consider the long-term solution or the permanent solution. The project team shall make a decision based on the current state of their project, timeline, and project resources.

Alleviation

[82.com, 04/23/2025]: We acknowledge the centralization concerns regarding contract upgrade mechanisms. This finding remains under active internal evaluation as we assess the appropriate balance between operational efficiency and decentralization. Any potential adjustments will be determined through our formal governance process.

COI-07 | CENTRALIZATION RISKS

Category	Severity	Location	Status
Centralization	● Centralization	domainNft/DomainNFTLogic.sol (base): 47, 48, 53, 58; safe/Safe.sol (base): 104; setting/SettingManagerLogic.sol (base): 62, 81, 91, 110, 115, 129, 133, 137, 141, 145, 149; trading/NFTOfferMarketLogic.sol (base): 33, 36, 229, 264, 272; transfer/TransferAgentLogic.sol (base): 33, 34, 38, 55, 60	● Acknowledged

Description

Several contracts grant powerful privileges to specific roles or addresses. If these roles are compromised, an attacker may gain control over critical operations.

DomainNFTLogic

- **ROLE_ADMIN**: Authorize upgrades
- **ROLE_ADMIN**: Set base URI
- **ROLE_FROZEN**: Freeze or unfreeze specific NFT token IDs

SafeV2

- **_owners**: Verify multiple signatures

SettingManagerLogic

- **DEFAULT_ADMIN_ROLE**: Authorize contract upgrades
- **FEE_MANAGER_ROLE**: Set transaction fee receiver
- **FEE_MANAGER_ROLE**: Set withdrawal fee rate and receiver
- **FEE_MANAGER_ROLE**: Set NFT creator and owner royalty rates
- **FEE_MANAGER_ROLE**: Set transaction fee rate
- **SAFE_MANAGER_ROLE**: Add or remove safe proxy addresses
- **TOKEN_MANAGER_ROLE**: Add or remove whitelisted tokens

MultiTokenManager

- **_owner**: Set `settingManager` address

- **_owner**: Set `transferAgent` address

NFTOfferMarketLogic

- **_owner**: Cancel sell and buy orders
- **_owner**: Authorize contract upgrade

TransferAgentLogic

- **_owner**: Add or remove whitelisted exchanges
- **_owner**: Authorize upgrade to new implementation
- **_whitelisted**: Transfer ERC721 and ERC20 tokens
- **_whitelisted**: Remove whitelisted exchange addresses

Risk Note

Important Note: Certain identification procedures were attempted to be applied to the project team in order to better understand the centralization situation and potential risks of the project. We strongly advise end users to conduct further research and exercise due diligence before engaging with the project given the centralization related risks. It is crucial for end users to independently verify and assess all available information

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

[82.com, 04/23/2025]: We acknowledge the centralization risks identified in COI-7. While maintaining the current privileged role structure for operational continuity, we are actively evaluating decentralized governance models (including DAO integration and multi-sig implementations) as part of our long-term roadmap to progressively reduce single points of failure in the protocol's administration.

COI-09 | POTENTIAL FRONT RUN IN `NFTOfferMarketLogic`

Category	Severity	Location	Status
Concurrency	● Major	trading/NFTOfferMarketLogic.sol (base): 150~151, 158~159, 166~167, 175~176	● Resolved

Description

The contracts in `NFTOfferMarketLogic` allow for the creation of buy and sell orders at a specific price. However, this price can be updated without modifying the order hash.

This could lead to an exploit where a buyer or a seller front runs the `acceptBuyOrder()` or `acceptSellOrder()` call, respectively, by a seller to modify the price of the related NFT.

Scenario

Buy Order

1. **Initial State:** Bob owns NFT #42.
2. **Alice's Offer:** Alice calls `createBuyOrder()` for this specific token and a price of 100. The contract computes an order hash (e.g. `0xABC123`) based on several parameters, but **not** the `price`.
3. **Bob's Acceptance:** Bob invokes `acceptBuyOrder()` to sell his NFT under the terms Alice originally offered (price = 100).
4. **Front-Run by Alice:** Before Bob's transaction is mined, Alice front-runs by calling `updateBuyOrder()` with the same parameters as before, except the price is modified to equal 1. This reuses the same order hash `0xABC123` because the hash omits the `price` field.
5. **Order Resolution:** When Bob's `acceptBuyOrder` finally executes, the contract looks up order `0xABC123` and finds the **updated** price: 1, not 100.
6. **Adverse Outcome:** Bob unintentionally transfers his NFT for just 1 token instead of 100.

Sell Order

1. **Bob's Setup:** To smoothly purchase NFTs, Bob grants the marketplace contract a large ERC-20 allowance (e.g. 10 000 tokens).
2. **Alice's Sell Order:** Alice calls `createSellOrder()` for NFT #43 and a price of 100. The contract computes an order hash (e.g. `0xABC123`) from parameters **excluding** the `price`.
3. **Bob's Acceptance:** Bob invokes `acceptSellOrder()` expecting to pay 100 tokens in return for NFT #43.
4. **Front-Run by Alice:** Before Bob's tx is mined, Alice front-runs by calling `updateSellOrder()` with the same parameters as before, except the price is modified to equal 10 000. Since the hash omits `price`, it remains

`0xABC123` .

5. **Order Resolution:** When Bob's pending `acceptSellOrder()` executes, the contract retrieves order `0xABC123` and now sees the **updated** price of 10 000.
6. **Adverse Outcome:** Bob unintentionally spends his entire allowance (10 000 tokens) instead of the intended 100.

Recommendation

We recommend including all sensible informations in the hash of an order to prevent exploiting it through an update.

Alleviation

[82.com, 04/23/2025]: We acknowledge this as a critical front-running vulnerability. Our implemented solution (commit 9b4f623) introduces price tolerance parameters in order execution functions - transactions now automatically revert if execution price exceeds the user-specified acceptable range, effectively mitigating this attack vector while maintaining market functionality.

COI-08 FUNCTION CALLS USER-PROVIDED ADDRESSES WITH NO ACCESS CONTROL MODIFIER

Category	Severity	Location	Status
Access Control	● Medium	setting/SettingManagerLogic.sol (base): 70	● Acknowledged

Description

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

[82.com, 04/23/2025]: We acknowledge the security considerations regarding arbitrary contract calls. While we currently lack an on-chain mechanism to verify contract maliciousness, this function is intentionally restricted to admin-only access. We're maintaining the current implementation based on our trust model for privileged roles, while continuing to explore more robust verification solutions for future iterations.

COI-10 LACK OF STORAGE GAP OR NAMESPACED STORAGE LAYOUT IN UPGRADEABLE CONTRACT

Category	Severity	Location	Status
Design Issue	● Medium	trading/NFTOfferMarketLogic.sol (base): 16, 72	● Acknowledged

Description

When updating upgradeable smart contracts for new features or bug fixes, keeping the state variables' declaration order unchanged is essential to avoid storage layout issues.

A practical solution is to include unused state variables or explicitly named storage gaps (like `__gap`) in the base contracts. This foresight allows reserved slots for future use, ensuring that any additions to the contract's state won't disrupt the storage pattern of derived contracts or the compatibility with previously deployed versions. After `ERC-7201` , it is also possible to place all storage variables of a contract into one or more structs like `Namespaced Storage Layout` .

The problem of "Lack of Storage Gap Or NameSpaced Storage Layout in Upgradeable Contract" occurs when **these storage gaps are not incorporated into the base contract's logic nor the base contract defines the namespace storage layout**. As a result, if new state variables are added to the base contract, they might overwrite existing variables in the child contracts due to storage slot collisions.

In the current contract, the contract allows for future upgrades and is also inherited by other contracts. However, the storage gap is missing for the the current contract, nor is the `namespaced storage layout` used.

For detailed guidelines and best practices, refer to the following OpenZeppelin documentation:

- https://docs.openzeppelin.com/contracts/4.x/upgradeable#storage_gaps
- https://docs.openzeppelin.com/upgrades-plugins/1.x/writing-upgradeable#storage-gaps

Recommendation

To mitigate this issue:

1. For enhanced flexibility in future upgrades of the logic contract, it is prudent to reserve a storage gap of an appropriate size in the base contract. This is achieved by declaring a fixed-size array, typically of `uint256` elements, each occupying a 32-byte slot, in the base contract. Label this array with the identifier `__gap` or any name prefixed with `__gap_` to indicate its purpose as a reserved space clearly.
2. it is also possible by placing all storage variables of a contract into one or more structs like `Namespaced Storage Layout` .

More detailed info :

- https://docs.openzeppelin.com/contracts/4.x/upgradeable#storage_gaps

Alleviation

[82.com, 04/23/2025]: We acknowledge the importance of storage gaps for upgradeable contracts. This will be systematically implemented across all upgradeable contracts in our next development cycle to ensure forward compatibility and prevent storage collisions during future upgrades.

COI-11 | USERS ARE FORCED TO APPROVE FOR ALL

Category	Severity	Location	Status
Volatile Code	● Medium	trading/NFTOfferMarketLogic.sol (base): 90~91	● Acknowledged

Description

The function '_validateExternalConfig()' requires the `ERC721` owner/operator to approve the current contract in order to transfer the specific token from the token owner. However, it must be approved for all, which is a level of approval that many users are not comfortable with. This is because it uses `isApprovedForAll()` to check validation. Thus, if only a single token is approved, the user will not be able to invoke the function, which is clearly unreasonable.

Recommendation

We recommend that users should be allowed to approve only a single token.

Alleviation

[82.com, 04/23/2025]: We recognize the security considerations regarding forced token approvals. While maintaining the current implementation for user experience optimization (as with COI-23), we've documented this as a potential future enhancement pending further evaluation of security/usability tradeoffs.

COI-12 | PREEMPTIVE FREEZING OF NON-EXISTENT TOKENS CAN BLOCK FUTURE MINTING

Category	Severity	Location	Status
Logical Issue	● Medium	domainNft/DomainNFTLogic.sol (base): 48~52, 59~64, 95	● Resolved

Description

Tokens that have not yet been minted can be marked as frozen through the `frozenTokenId()` function, allowing anyone with the appropriate role to preemptively freeze future token IDs. This could inadvertently or maliciously block the minting of new tokens, as frozen tokens trigger a revert during the minting process due to checks in the `_update()` function, potentially halting the contract's core functionality.

Recommendation

We recommend adding a check to ensure that only existing or already minted token IDs can be frozen to prevent disruption of the minting process.

Alleviation

[82.com, 04/23/2025]: We acknowledge this design oversight where the system previously trusted admin inputs to only reference minted NFT IDs. The vulnerability has been patched in commit 0fcc1e1 by implementing validation to prevent freezing unminted tokens, enhancing both security and protocol integrity.

COI-13 | AMBIGUOUS ERC20 TRANSFERS TO SAME RECIPIENT MAY INDICATE MISCONFIGURED FEE LOGIC

Category	Severity	Location	Status
Logical Issue	● Medium	trading/NFTOfferMarketLogic.sol (base): 68~69	● Acknowledged

Description

Two separate amounts, `feeInfo.ownFee` and `feeInfo.remainingAmount` , are both transferred to the same `_to` address, which may indicate either redundant or unintended logic if `_to` is not meant to receive both values. If `_to` is expected to be the actual recipient of the remaining amount only, and `ownFee` was meant for another stakeholder, such as the `NFT` owner, sending both to `_to` could result in overpayment.

Recommendation

We recommend clarifying or fixing the logic to ensure that `feeInfo.ownFee` and `feeInfo.remainingAmount` are sent to the correct intended recipients and not redundantly to the same address unless explicitly required by the business logic.

Alleviation

[82.com, 04/23/2025]: We acknowledge the observation regarding ambiguous ERC20 transfers. The current implementation remains unchanged as the royalty distribution mechanism is still under active development. We will implement necessary adjustments once the final royalty framework is formally established.

COI-14 | INSUFFICIENT SAFEGUARDS ALLOW ARBITRARY CALLS DESPITE SIGNATURE VALIDATION

Category	Severity	Location	Status
Logical Issue	● Medium	modules/TokenWithdrawModule.sol (base): 27~28, 55	● Acknowledged

Description

Despite requiring signature validation through `checkSignatures()`, both `tokenTransfer()` and `nftTransfer()` allow arbitrary calls to external contracts using the Safe's `execTransactionFromModule()`, which can be abused by a malicious Safe configuration where the internal logic of `checkSignatures()` or the Safe itself is compromised or intentionally permissive. Since the contract trusts the Safe's internal permission system without enforcing its own validation on target addresses or expected method selectors, an attacker could craft a Safe instance or manipulate its module setup to execute unintended transactions, leading to unauthorized token movements or interactions with malicious contracts.

Recommendation

We recommend enforcing strict validation on target contract addresses and expected function signatures within `tokenTransfer()` and `nftTransfer()` to prevent abuse through malicious or misconfigured Safe instances, even if signatures appear valid.

Alleviation

[82.com, 04/23/2025]: Issue acknowledged. I will fix the issue in the future, which will not be included in this audit engagement.

COI-15 | SELF-ADMINISTERED ROLE ALLOWS FOR COMPLETE ROLE TAKEOVER

Category	Severity	Location	Status
Design Issue	● Minor	domainNft/DomainNFTLogic.sol (base): 36; setting/SettingManagerLogic.sol (base): 52	● Acknowledged

Description

The current contract implements `AccessControl` Mechanism.

In the setup, the `privileged role` is assigned as its own admin role, which creates a significant security risk. This setup allows any account with the `privileged role` to have administrative control over itself, including the ability to revoke the role from other accounts.

```
    _setRoleAdmin(PRIVILEGED_ROLE, PRIVILEGED_ROLE);
```

If a malicious actor gains the `privileged role`, they can remove this role from all other accounts, effectively taking complete control.

Recommendation

To mitigate this issue, assign a different, higher-level administrative role to `PRIVILEGED_ROLE`. This higher-level role should have the authority to manage the `PRIVILEGED_ROLE` assignments, thus preventing any single account with the `PRIVILEGED_ROLE` from having unchecked control.

Alleviation

[82.com, 04/23/2025]: After careful evaluation, we recognize the security implications of self-administered roles. While we maintain the current implementation for operational continuity, this finding has been documented as a potential optimization candidate pending further internal review and governance discussions.

COI-16 | MISSING ZERO ADDRESS VALIDATION

Category	Severity	Location	Status
Volatile Code	● Minor	domainNft/DomainNFTLogic.sol (base): 40; setting/SettingManagerLogic.sol (base): 60; trading/NFTOfferMarketLogic.sol (base): 34, 37	● Acknowledged

Description

The cited address input is missing a check that it is not `address(0)` .

Recommendation

We recommend adding a check the passed-in address is not `address(0)` to prevent unexpected errors.

Alleviation

[82.com, 04/23/2025]: As a security best practice, we acknowledge the importance of zero-address validation for critical input parameters. This issue will be prioritized in our upcoming optimization cycle to enhance contract robustness and prevent unintended behavior.

COI-17 | INCORRECT SETUP OF EIP712 DOMAIN SEPARATOR

Category	Severity	Location	Status
Design Issue	● Minor	safe/Safe.sol (base): 154	● Acknowledged

Description

According to the `EIP712` standard, the hash of the `EIP712Domain` separator should be:

```
bytes32 private constant _TYPE_HASH =
        keccak256("EIP712Domain(string name,string version,uint256 chainId,address
verifyingContract)");
```

However, the current setup is incorrect, which is incompatible with EIP712.

Recommendation

To mitigate this issue, it is recommended to follow the `EIP712` standard.

Alleviation

[82.com, 04/23/2025]: Regarding COI-17 (EIP712 Domain Separator), the current implementation originates from the inherited Safe contract framework. To ensure the stability of existing signature verification, we will maintain the current implementation and address this during future foundational contract upgrades, while enhancing signature monitoring mechanisms in the interim.

COI-18 | `_setRoleAdmin` TO `DEFAULT_ADMIN_ROLE` IS REDUNDANT

Category	Severity	Location	Status
Logical Issue	● Minor	setting/SettingManagerLogic.sol (base): 52	● Resolved

Description

The specific role is assigned with `DEFAULT_ADMIN_ROLE` as the role admin. However, the assignment here is redundant, as `DEFAULT_ADMIN_ROLE` will be the role admin for any role by default.

Recommendation

To mitigate this issue, it is recommended to remove the redundant the assignment.

Alleviation

[82.com, 04/23/2025]: egarding COI-18 (Redundant Role Assignment), technical review confirmed the explicit DEFAULT_ADMIN_ROLE assignment was redundant. We've removed this redundant code in GitHub commit c5ad4d9.

COI-19 | POTENTIAL REENTRANCY ATTACK (OUT-OF-ORDER EVENTS)

Category	Severity	Location	Status
Concurrency	● Minor	modules/TokenWithdrawModule.sol (base): 22~42, 43~47, 48~59; safe/Safe.sol (base): 55~88, 93; trading/NFTOfferMarketLogic.sol (base): 45, 50, 197~206, 207~216; transfer/TransferAgentLogic.sol (base): 55~59, 60~64	● Acknowledged

Description

A reentrancy attack can occur when the contract creates a function that makes an external call to another untrusted contract before resolving any effects. If the attacker can control the untrusted contract, they can make a recursive call back to the original function, repeating interactions that would have otherwise not run after the external call resolved the effects.

This finding is considered minor because the reentrancy only causes out-of-order events.

External call(s)

```
36            _executeTokenTransfer(_tokenAddress, _safeAddress, _beneficiary, _amount
);
```

- This function call executes the following external call(s).
- In `TokenWithdrawModuleV2._executeTokenTransfer` ,
    - `require(bool,string)(SafeV2(address(_safeAddress)).execTransactionFromModule(_tokenAddress,0,data,Enum.Operation.Call),Could not execute token transfer)`

```
40                _executeTokenTransfer(_tokenAddress,_safeAddress,feeReceiver,
_feeAmount);
```

- This function call executes the following external call(s).
- In `TokenWithdrawModuleV2._executeTokenTransfer` ,
    - `require(bool,string)(SafeV2(address(_safeAddress)).execTransactionFromModule(_tokenAddress,0,data,Enum.Operation.Call),Could not execute token transfer)`

Events emitted after the call(s)

```
        emit TokenTransferred(_tokenAddress,_safeAddress,_recipient,_amount);
```

- Executed via the following function call(s):
    - `_executeTokenTransfer(_tokenAddress,_safeAddress,feeReceiver,_feeAmount)`

External call(s)

```
        require(SafeV2(payable(_safeAddress)).execTransactionFromModule(
_tokenAddress,0,data,Enum.Operation.Call),"Could not execute NFT transfer");
```

Events emitted after the call(s)

```
        emit NFTTransferred(_tokenAddress,_safeAddress,_recipient,_tokenId);
```

External call(s)

```
                Guard(guard).checkTransaction(to, value, data, operation,
safeTxGas, baseGas, gasPrice, gasToken, refundReceiver, signatures, msg.sender);
```

```
                payment = handlePayment(gasUsed, baseGas, gasPrice, gasToken,
refundReceiver);
```

- This function call executes the following external call(s).
- In `SafeV2.handlePayment` ,
    - `require(bool,string)(receiver.send(payment),GS011)`

Events emitted after the call(s)

```
            else emit ExecutionFailure(txHash, payment);
```

```
            if (success) emit ExecutionSuccess(txHash, payment);
```

External call(s)

```
        require(SafeV2(payable(_safeAddress)).execTransactionFromModule(
_tokenAddress,0,data,Enum.Operation.Call),"Could not execute token transfer");
```

Events emitted after the call(s)

```
        emit TokenTransferred(_tokenAddress,_safeAddress,_recipient,_amount);
```

External call(s)

```
        erc20.safeTransferFrom(_from, _to, _amount);
```

Events emitted after the call(s)

```
        emit ERC20Transferred(_exchange, _from, _to, _amount);
```

External call(s)

```
        nft.safeTransferFrom(_from, _to, _nftTokenId);
```

Events emitted after the call(s)

```
        emit ERC721Transferred(_nftAddress, _from, _to, _nftTokenId);
```

External call(s)

```
        _transferERC20FromSupportingFee(_erc20Token, order.price, msg.sender,
_promisee, _erc721Token, _nftTokenId);
```

- This function call executes the following external call(s).
- In `MultiTokenManager._transferERC20From` ,
    - `transferAgent.transferERC20(_erc20Token,_from,_to,_amount)`

```
        _transferERC721From(_erc721Token, _nftTokenId, _promisee, msg.sender);
```

- This function call executes the following external call(s).
- In `MultiTokenManager._transferERC721From` ,

    - `transferAgent.transferERC721(_erc721Token,_nftTokenId,_from,_to)`

Events emitted after the call(s)

```
205         emit OrderAccepted(orderHash, _orderId, OrderType.Sell, _promisee, msg.
sender, _erc721Token, _nftTokenId, _erc20Token, order.price);
```

External call(s)

```
212         _transferERC721From(_erc721Token, _nftTokenId, msg.sender, _promisee);
```

- This function call executes the following external call(s).
- In `MultiTokenManager._transferERC721From` ,
    - `transferAgent.transferERC721(_erc721Token,_nftTokenId,_from,_to)`

```
213         _transferERC20FromSupportingFee(_erc20Token, order.price, _promisee,
msg.sender, _erc721Token, _nftTokenId);
```

- This function call executes the following external call(s).
- In `MultiTokenManager._transferERC20From` ,
    - `transferAgent.transferERC20(_erc20Token,_from,_to,_amount)`

Events emitted after the call(s)

```
215         emit OrderAccepted(orderHash, _orderId, OrderType.Buy, _promisee, msg.
sender, _erc721Token, _nftTokenId, _erc20Token, order.price);
```

Recommendation

We recommend using the Checks-Effects-Interactions Pattern to avoid the risk of calling unknown contracts or applying OpenZeppelin ReentrancyGuard library - `nonReentrant` modifier for the aforementioned functions to prevent reentrancy attack.

Alleviation

[82.com, 04/23/2025]: Regarding COI-19 (Potential Reentrancy Risk), we have identified optimization opportunities in event emission ordering. The specific adjustments will be finalized and implemented in upcoming development cycles.

COI-20 | `_nextTokenId` REFLECTS THE CURRENT TOKEN ID

Category	Severity	Location	Status
Logical Issue	● Minor	domainNft/DomainNFTLogic.sol (base): 60~61	● Resolved

Description

The `DomainNFTLogic.mint()` function uses pre-incrementation when assigning a new token ID:

```
uint256 tokenId = ++_nextTokenId;
_safeMint(to, tokenId);
```

As a result, `tokenId` is assigned the incremented value of `_nextTokenId` which then holds the value of the ID of the next token to be minted.

Recommendation

We recommend switching to post-incrementation:

```
uint256 tokenId = _nextTokenId++;
```

This ensures `_nextTokenId` truly represents the *next* available token ID.

Alleviation

[82.com, 04/25/2025]: Regarding COI-20 (Token ID Counting Logic), we identified a discrepancy between _nextTokenId naming and its actual behavior. The optimized solution now: 1) Initializes value at 1; 2) Uses post-increment (tokenId = _nextTokenId++) for precise semantics. This ensures full consistency between variable naming and logic, implemented in GitHub commit 97f0da6.

COI-21 | MISSING INPUT VALIDATION IN `viewRoleMember()` LEADS TO SILENT OR UNEXPECTED FAILURES

Category	Severity	Location	Status
Logical Issue	● Minor	domainNft/DomainNFTLogic.sol (base): 81~87; setting/SettingManagerLogic.sol (base): 122~127; transfer/TransferAgentLogic.sol (base): 48~53	● Partially Resolved

Description

The `DomainNFTLogic.viewRoleMember()` function is intended to extract from a role member list the addresses list up to `size` entries starting at `cursor`.

However, there are two edge cases:

- **`cursor == memberCount`**
  Returns an empty array with no indication that the request was out-of-range.
- **`size == 0`**
  Also yields an empty array without errors.

Similar silent failures can also take place in:

- `SettingManagerLogic.viewWhitelistedSafes()`
- `TransferAgentLogic.viewWhitelistedExchange()`

Recommendation

We recommend adding explicit input checks at the start of the function to guard against out-of-bounds or zero-length requests.

Alleviation

[82.com, 04/23/2025]: Regarding COI-21 (Missing Input Validation), we've implemented boundary checks in viewRoleMember() including: 1) Explicit error for out-of-range cursor 2) Exception for zero-size requests. This fix prevents silent failures, with implementation at GitHub commit 7921d6d.

[CertiK, 04/25/2025]: The team heeded the advice and partially resolved the issue in commit 7921d6db7d0afe84c3dde6871389b68f8b2464a7.

Similar issues still exist in:

- `SettingManagerLogic.viewWhitelistedSafes()`
- `TransferAgentLogic.viewWhitelistedExchange()`

[CertiK, 04/28/2025]: The team heeded the advice and fully resolved the issue in commit 8b42c54fce2c7bb494e4c6e57842563e5e784de7.

COI-23 UNSAFE UNLIMITED TOKEN APPROVALS AND OPERATOR PERMISSIONS GRANTED TO TRANSFER AGENT

Category	Severity	Location	Status
Design Issue	● Minor	safe/SafeProxy.sol (base): 33~42	● Acknowledged

Description

Blindly granting unlimited token approvals and universal operator permissions introduces significant security risks, as it gives the `transferAgent` full control over all approved `ERC20` and `ERC721` tokens without constraints or expiration. If the `transferAgent` becomes compromised or behaves maliciously, it could drain all user tokens approved through this function. Additionally, using `type(uint256).max` for `ERC20` approvals is a known anti-pattern, as it lacks fine-grained control and cannot be easily revoked mid-way without an extra approval transaction, increasing vulnerability to front-running or misuse.

Recommendation

We recommend setting precise approval amounts for `ERC20` tokens and granting `ERC721` operator permissions only when necessary, preferably with revocation mechanisms or time-limited scopes to minimize security risks.

Alleviation

[82.com, 04/23/2025]: Regarding COI-23 (Unlimited Approval Risk), the current full-amount approval is a UX-oriented design choice to minimize repeated approvals in high-frequency trading scenarios. We've flagged this for optimization and will evaluate precise-amount approval implementation in the next version, while mitigating potential risks through real-time monitoring and user warnings.

COI-24 | UNBOUNDED TOKEN WHITELIST MAY LEAD TO GAS EXHAUSTION

Category	Severity	Location	Status
Denial of Service	● Minor	safe/SafeProxy.sol (base): 31~42; setting/SettingManagerLogic.sol (base): 81~90, 98~109	● Acknowledged

Description

Allowing unlimited tokens to be added to the `_whitelistedTokens` set without any cap can lead to a situation where the `viewWhitelistedTokensByType()` function consumes excessive gas due to its iteration over all entries in the set. Since `viewWhitelistedTokensByType()` is a public view function that reads the entire `_whitelistedTokens` collection and filters by type, its gas usage grows linearly with the number of tokens added. In extreme cases, this may cause the function to exceed the block gas limit, making it unusable and potentially breaking interfaces or integrations that rely on it.

Recommendation

We recommend enforcing a maximum limit on the number of whitelisted tokens or implementing pagination in the `viewWhitelistedTokensByType()` function to prevent potential gas exhaustion.

Alleviation

[82.com, 04/23/2025]: Regarding COI-24 (Unbounded Whitelist Gas Risk), we have strictly limited the whitelist to 10 tokens, well below the risk threshold. A hard-coded limit will be considered in future upgrades, while the current design remains unchanged.

COI-25 | STALE TOKEN TYPE MAPPING REMAINS AFTER TOKEN REMOVAL

Category	Severity	Location	Status
Logical Issue	● Minor	setting/SettingManagerLogic.sol (base): 91~95	● Resolved

Description

When a token is removed from the `_whitelistedTokens` set using `removeToken()` , its corresponding entry in the `_tokenTypes` mapping is not deleted, which leads to stale data remaining in storage. This leftover mapping can cause inconsistencies if other parts of the contract rely on `_tokenTypes` for logic, potentially resulting in incorrect token type interpretation or unauthorized behavior. Additionally, it introduces unnecessary storage costs and may increase the complexity of future upgrades or audits.

Recommendation

We recommend explicitly deleting the `_tokenTypes[token]` entry when a token is removed from the whitelist to ensure storage consistency and prevent stale data from affecting contract logic.

Alleviation

[82.com, 04/23/2025]: Regarding COI-25 (Stale Data After Token Removal), we've added _tokenTypes[token] cleanup logic in removeToken() function to ensure token type data is deleted when a token is removed from _whitelistedTokens.See implementation in GitHub commit 8f77b9c.

COI-26 UNINITIALIZED CRITICAL CONFIGURATION VARIABLES MAY LEAD TO UNDEFINED BEHAVIOR

Category	Severity	Location	Status
Logical Issue	● Minor	setting/SettingManagerLogic.sol (base): 49~61, 133~140; trading/NFTOfferMarketLogic.sol (base): 33~38	● Resolved

Description

Critical configuration variables such as:

- `_transactionFeeRate` ,
- `_withdrawalFeeReceiver` ,
- `transferAgentAddress` ,
- `settingManagerAddress`

are not initialized within the `initialize()` function, which may lead to undefined or invalid behavior if these variables are accessed before being explicitly set through their respective setter functions. This creates a dependency on external calls post-deployment for proper configuration and increases the risk of misconfiguration or unintended contract behavior, especially if fee-related operations or contract integrations rely on these values being present and valid from the start.

Recommendation

We recommend initializing `_transactionFeeRate` , `_withdrawalFeeReceiver` , `transferAgentAddress` , and `settingManagerAddress` within the initialize function or enforcing non-zero checks before their usage to ensure the contract operates with valid configurations from the beginning.

Alleviation

[82.com, 04/23/2025]: Regarding COI-26 (Uninitialized Variables), we have added default value assignments for all critical configuration variables (including transactionFeeRate, withdrawalFeeReceiver, etc.) in the contract initialization function to ensure safe state upon deployment. This fix prevents unexpected behaviors caused by uninitialized variables, with implementation details in GitHub commit 7f828cd .

COI-27 | REDUNDANT CHECK

Category	Severity	Location	Status
Volatile Code	● Minor	setting/SettingManagerLogic.sol (base): 120~121; trading/NFTOfferMarketLogic.sol (base): 95~96	● Resolved

Description

In `NFTOfferMarketLogic._validateExternalConfig()` , the following check is made

```
95          if (!settings.isSafeWhitelisted(settings.getTransactionFeeReceiver()))
return "FeeReceiverNotWhitelisted";
```

However, in the current `SettingManagerLogic` contract implementation, `isSafeWhitelisted()` is defined as follows

```
120      function isSafeWhitelisted(address safe) external view returns (bool) {
return _whitelistedSafes.contains(safe) || safe == _transactionFeeReceiver; }
```

Therefore, by providing `_transactionFeeReceiver` as the input parameter, the `NFTOfferMarketLogic._validateExternalConfig()` check will always pass.

Recommendation

We recommend modifying the code to ensure the check is effective.

Alleviation

[82.com, 04/23/2025]: Regarding **COI-27 (Redundant Check)**, we confirmed that this validation became redundant during code iterations. To optimize contract efficiency, we removed the check in GitHub Commit 22885b1 and added test coverage to ensure functional integrity.

COI-28 POTENTIAL CROSSFUNCTION REENTRANCY IN `NFTOfferMarketLogic`

Category	Severity	Location	Status
Concurrency	● Minor	trading/NFTOfferMarketLogic.sol (base): 248~249, 251~252, 254~255, 259~260	● Resolved

Description

In the contract `NFTOfferMarketLogic` , the following functions trigger the transfer of ERC721:

- `acceptSellOrder()`
- `acceptBuyOrder()`
- `batchAcceptSellOrder()`
- `batchAcceptBuyOrder()`

which all share a pattern similar to:

```
        _transferERC721From(_erc721Token, _nftTokenId, msg.sender, _promisee);
        _transferERC20FromSupportingFee(_erc20Token, order.price, _promisee,
msg.sender, _erc721Token, _nftTokenId);
        delete makerOrderMapping[orderHash];
        emit OrderAccepted(orderHash, _orderId, OrderType.Buy, _promisee,
msg.sender, _erc721Token, _nftTokenId, _erc20Token, order.price);
```

ERC-721 `safeTransferFrom` invokes the recipient's `onERC721Received()` hook when the receiver is a contract. A malicious hook implementation can reenter back into the contract before its state is fully updated.
Because the functions mentioned above are protected by a `nonReentrant` modifier, calling one of these functions when reentering the contract won't be possible; however, calling the other unprotected functions would still be possible.

This can create confusion when monitoring transactions, as out-of-order events could be produced.
Moreover, if other contracts in the protocol rely on `makerOrderMapping` for safety checks, this could lead to read-only reentrancies as the ERC721 transfer before the mapping deletion could allow triggering external functions with out of date information.

Scenario

Post Buy Order cancel

1. Alice has created a buy order for Bob's NFT

2. Alice's contract contains a malicious implementation of the ERC721 `onERC721Received()` function, calling back `cancelBuyOrder()` for the exact same order
3. Bob triggers the `acceptBuyOrder()` to sell his NFT
4. Before `makerOrderMapping[orderHash]` is deleted and `OrderAccepted` event is emitted, `cancelBuyOrder()` will be triggered, deleting the `makerOrderMapping[orderHash]` and emitting an `OrderCancelled` event
5. As a result, the sell executes as expected, but the mapping `makerOrderMapping[orderHash]` is deleted twice, and an event `OrderCancelled` is emitted before an `OrderAccepted` event for the exact same order.

Multiple similar scenarios are possible, causing issues to monitor transactions, and potentially causing more severe damages if off-chain code relies on events to trigger other transactions.

Read-only reentrancy

Using the ERC721 `onERC721Received()` , the malicious buyer calls an external contract relying on `NFTOfferMarketLogic` public mapping `makerOrderMapping[orderHash]` to check if a specific order exists. Since this call takes place before deleting `makerOrderMapping[orderHash]` for the specific order, the external contract will work on outdated invalid data.

Recommendation

We recommend applying the check-effect interaction pattern by deleting the mapping and emitting the event **before** triggering ERC721 transfer.

Alleviation

[82.com, 04/25/2025]: Thanks for catching that. We've applied CEI pattern consistently - state updates now precede external calls in all functions. Fixed in commit 6a6e5bbc.

COI-36 | INITIALIZE STATE VARIABLE IN CONSTRUCTOR OR DECLARATION IN UPGRADEABLE CONTRACT

Category	Severity	Location	Status
Logical Issue	● Minor	domainNft/DomainNFTLogic.sol (Update1): 15	● Resolved

Description

In Solidity, initialization logic inside a constructor or global variable declaration in a logic contract is not reflected in the proxy contract's state when using the proxy pattern for upgradeable contracts. This is because the initialization code in the constructor or the global variable declaration is not part of the contract's runtime bytecode, and the proxy contract does not execute it.

Recommendation

We recommend moving the initialization within the global state variable declaration to an initialize function to avoid unexpected behavior and confusion.

Alleviation

[82.com, 04/29/2025]: Thanks for catching this I've renamed '_nextTokenId' to '_currentTokenId' and switched to pre-increment '++i' to ensure IDs start from 1. See the changes here: 2257171.

COI-29 | INCONSISTENT SOLIDITY VERSIONS

Category	Severity	Location	Status
Language Version	● Informational	domainNft/DomainNFTProxy.sol (base): 2; interfaces/IDomainNFT.sol (base): 2; interfaces/ISettingManager.sol (base): 2; interfaces/ITransferAgent.sol (base): 2; modules/TokenWithdrawModule.sol (base): 2; safe/IProxyCreationCallback.sol (base): 2; safe/Safe.sol (base): 2; safe/SafeProxy.sol (base): 2; safe/SafeProxyFactory.sol (base): 2; setting/SettingManagerLogic.sol (base): 2; setting/SettingManagerProxy.sol (base): 2; trading/NFTOfferMarketLogic.sol (base): 2; trading/NFTOfferMarketProxy.sol (base): 2; transfer/TransferAgentLogic.sol (base): 2; transfer/TransferAgentProxy.sol (base): 2	● Acknowledged

Description

The codebase contains multiple Solidity versions, which can lead to unexpected behavior, potential vulnerabilities, difficulties in maintaining the code, and inconsistencies in the execution of the smart contract. Using different versions may also result in increased complexity during code auditing, as different security features and bug fixes are present in different versions of the compiler.

Versions used: `^0.8.20` , `>=0.7.0<0.9.0` , `^0.8.22` , `>=0.8.0<0.9.0`

```
pragma solidity ^0.8.20;
```

`^0.8.20` is used in modules/TokenWithdrawModule.sol file.

```
pragma solidity ^0.8.20;
```

```
pragma solidity >=0.7.0 <0.9.0;
```

`>=0.7.0<0.9.0` is used in node_modules/@safe-global/safe-contracts/contracts/interfaces/ISignatureValidator.sol file.

```
pragma solidity >=0.7.0 <0.9.0;
```

```
pragma solidity ^0.8.22;
```

`^0.8.22` is used in interfaces/ITransferAgent.sol file.

```
pragma solidity ^0.8.22;
pragma solidity >=0.8.0 <0.9.0;
```

`>=0.8.0<0.9.0` is used in safe/Safe.sol file.

```
pragma solidity >=0.8.0 <0.9.0;
```

Versions used: `^0.8.20`, `^0.8.22`

```
pragma solidity ^0.8.0;
```

`^0.8.20` is used in node_modules/@openzeppelin/contracts/utils/structs/EnumerableSet.sol file.

```
pragma solidity ^0.8.0;
```

```
pragma solidity ^0.8.22;
```

`^0.8.22` is used in transfer/TransferAgentLogic.sol file.

```
pragma solidity ^0.8.22;
```

Versions used: `^0.8.20`, `^0.8.22`

`^0.8.20` is used in node_modules/@openzeppelin/contracts/utils/StorageSlot.sol file.

```
pragma solidity ^0.8.22;
```

`^0.8.22` is used in transfer/TransferAgentProxy.sol file.

```
pragma solidity ^0.8.22;
```

Versions used: `^0.8.20`, `^0.8.22`

`^0.8.20` is used in node_modules/@openzeppelin/contracts/utils/StorageSlot.sol file.

```
pragma solidity ^0.8.22;
```

`^0.8.22` is used in trading/NFTOfferMarketLogic.sol file.

```
pragma solidity ^0.8.22;
```

Versions used: `^0.8.20`, `^0.8.22`

`^0.8.20` is used in node_modules/@openzeppelin/contracts/utils/StorageSlot.sol file.

```
pragma solidity ^0.8.22;
```

`^0.8.22` is used in trading/NFTOfferMarketProxy.sol file.

```
pragma solidity ^0.8.22;
```

Versions used: `^0.8.0` , `^0.8.2` , `^0.8.1` , `^0.8.22`

```
pragma solidity ^0.8.0;
```

`^0.8.0` is used in node_modules/@openzeppelin/contracts/utils/StorageSlot.sol file.

```
pragma solidity ^0.8.0;
```

```
pragma solidity ^0.8.2;
```

`^0.8.2` is used in node_modules/@openzeppelin/contracts/proxy/ERC1967/ERC1967Upgrade.sol file.

```
pragma solidity ^0.8.2;
```

```
pragma solidity ^0.8.1;
```

`^0.8.1` is used in node_modules/@openzeppelin/contracts/utils/Address.sol file.

```
pragma solidity ^0.8.1;
```

```
pragma solidity ^0.8.22;
```

`^0.8.22` is used in domainNft/DomainNFTProxy.sol file.

```
pragma solidity ^0.8.22;
```

Versions used: `>=0.7.0<0.9.0` , `^0.8.22` , `>=0.8.0<0.9.0`

```
pragma solidity >=0.7.0 <0.9.0;
```

`>=0.7.0<0.9.0` is used in node_modules/@safe-global/safe-contracts/contracts/interfaces/ISignatureValidator.sol file.

```
pragma solidity >=0.7.0 <0.9.0;
pragma solidity ^0.8.22;
```

`^0.8.22` is used in interfaces/ITransferAgent.sol file.

```
pragma solidity ^0.8.22;
```

```
pragma solidity >=0.8.0 <0.9.0;
```

`>=0.8.0<0.9.0` is used in safe/SafeProxy.sol file.

```
pragma solidity >=0.8.0 <0.9.0;
```

Versions used: `>=0.7.0<0.9.0` , `^0.8.22` , `>=0.8.0<0.9.0`

```
pragma solidity >=0.7.0 <0.9.0;
```

`>=0.7.0<0.9.0` is used in node_modules/@safe-global/safe-contracts/contracts/interfaces/ISignatureValidator.sol file.

```
pragma solidity >=0.7.0 <0.9.0;
```

```
pragma solidity ^0.8.22;
```

`^0.8.22` is used in interfaces/ITransferAgent.sol file.

```
pragma solidity ^0.8.22;
```

```
pragma solidity >=0.8.0 <0.9.0;
```

`>=0.8.0<0.9.0` is used in safe/SafeProxyFactory.sol file.

```
pragma solidity >=0.8.0 <0.9.0;
```

Versions used: `^0.8.0` , `^0.8.2` , `^0.8.1` , `^0.8.22`

```
pragma solidity ^0.8.0;
```

`^0.8.0` is used in node_modules/@openzeppelin/contracts/utils/StorageSlot.sol file.

```
pragma solidity ^0.8.0;
```

```
pragma solidity ^0.8.2;
```

`^0.8.2` is used in node_modules/@openzeppelin/contracts/proxy/ERC1967/ERC1967Upgrade.sol file.

```
pragma solidity ^0.8.2;
```

```
pragma solidity ^0.8.1;
```

`^0.8.1` is used in node_modules/@openzeppelin/contracts/utils/Address.sol file.

```
pragma solidity ^0.8.1;
```

```
pragma solidity ^0.8.22;
```

`^0.8.22` is used in setting/SettingManagerProxy.sol file.

```
pragma solidity ^0.8.22;
```

Versions used: `^0.8.0`, `^0.8.2`, `^0.8.1`, `^0.8.22`

```
pragma solidity ^0.8.0;
```

`^0.8.0` is used in node_modules/@openzeppelin/contracts/utils/structs/EnumerableSet.sol file.

```
pragma solidity ^0.8.0;
```

```
pragma solidity ^0.8.2;
```

`^0.8.2` is used in node_modules/@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol file.

```
pragma solidity ^0.8.2;
```

```
pragma solidity ^0.8.1;
```

`^0.8.1` is used in node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol file.

```
pragma solidity ^0.8.1;
```

```
pragma solidity ^0.8.22;
```

`^0.8.22` is used in setting/SettingManagerLogic.sol file.

```
pragma solidity ^0.8.22;
```

Recommendation

It is recommended to standardize on a single, up-to-date Solidity version throughout the codebase to ensure consistent behavior, benefit from the latest security features, and improve maintainability.

Alleviation

[82.com, 05/06/2025]: Issue acknowledged. I won't make any changes for the current version.

COI-30 | POTENTIAL REENTRANCY ATTACK (IN CASE OF UNLIMITED GAS)

Category	Severity	Location	Status
Concurrency	● Informational	safe/Safe.sol (base): 42~54, 55~88, 93	● Acknowledged

Description

A reentrancy attack can occur when the contract creates a function that makes an external call to another untrusted contract before resolving any effects. If the attacker can control the untrusted contract, they can make a recursive call back to the original function, repeating interactions that would have otherwise not run after the external call resolved the effects.

This finding is only informational because it only involves `transfer` and `send` calls. Those functions may not protect from reentrancies in case of gas price changes.

External call(s)

```
78                  payment = handlePayment(gasUsed, baseGas, gasPrice, gasToken,
 refundReceiver);
```

- This function call executes the following external call(s).
- In `SafeV2.handlePayment` ,
    - `require(bool,string)(receiver.send(payment),GS011)`

Events emitted after the call(s)

```
81              else emit ExecutionFailure(txHash, payment);
```

```
80              if (success) emit ExecutionSuccess(txHash, payment);
```

External call(s)

```
51              handlePayment(payment, 0, 1, paymentToken, paymentReceiver);
```

- This function call executes the following external call(s).
- In `SafeV2.handlePayment` ,

- `require(bool,string)(receiver.send(payment),GS011)`

Events emitted after the call(s)

```
        emit SafeSetup(msg.sender, _owners, _threshold, to, fallbackHandler,
address(settingManager), address(transferAgent));
```

Recommendation

We recommend using the Checks-Effects-Interactions Pattern to avoid the risk of calling unknown contracts or applying OpenZeppelin ReentrancyGuard library - `nonReentrant` modifier for the aforementioned functions to prevent reentrancy attack.

Alleviation

[82.com, 05/06/2025]: Issue acknowledged. I won't make any changes for the current version.

COI-31 | UNUSED STATE VARIABLE

Category	Severity	Location	Status
Coding Issue	● Informational	safe/Safe.sol (base): 19~163	● Acknowledged

Description

Some state variables are not used in the codebase. This can lead to incomplete functionality or potential vulnerabilities if these variables are expected to be utilized.

Variable `_deprecatedDomainSeparator` in `SafeV2` is never used in `SafeV2`.

```
30      bytes32 private _deprecatedDomainSeparator;
```

```
19  contract SafeV2 is Singleton,NativeCurrencyPaymentFallback,ModuleManager,
OwnerManager,SignatureDecoder,SecuredTokenTransfer,ISignatureValidatorConstants,
FallbackManager,StorageAccessible,GuardManager{
```

Recommendation

It is recommended to ensure that all necessary state variables are used, and remove redundant variables.

Alleviation

[82.com, 05/06/2025]: Issue acknowledged. I won't make any changes for the current version.

COI-32 | LOCAL VARIABLE SHADOWING

Category	Severity	Location	Status
Coding Style	● Informational	trading/NFTOfferMarketLogic.sol (base): 114, 127, 141	● Acknowledged

Description

A local variable is shadowing another component defined elsewhere. This means that when the contract accesses the variable by its name, it will use the one defined locally, not the one defined in the other place. The use of the variable may lead to unexpected results and unintended behavior.

```
141             try nft.ownerOf(_nftTokenId) returns (address owner) { if (owner !=
_trade) return "NotOwner"; } catch { return "NotOwner"; }
```

- Local variable `owner` in `OrderManager._validateAccept` shadows the function `owner` in `OwnableUpgradeable` .

```
114             try nft.ownerOf(_nftTokenId) returns (address owner) { if (owner !=
_trade) return "NotOwner"; } catch { return "NotOwner"; }
```

- Local variable `owner` in `OrderManager._validateCreate` shadows the function `owner` in `OwnableUpgradeable` .

```
127             try nft.ownerOf(_nftTokenId) returns (address owner) { if (owner !=
_trade) return "NotOwner"; } catch { return "NotOwner"; }
```

- Local variable `owner` in `OrderManager._validateUpdate` shadows the function `owner` in `OwnableUpgradeable` .

Recommendation

It is recommended to remove or rename the local variable that shadows another definition to prevent potential issues and maintain the expected behavior of the smart contract.

Alleviation

[82.com, 05/06/2025]: Issue acknowledged. I won't make any changes for the current version.

COI-33 | TOO MANY DIGITS

Category	Severity	Location	Status
Magic Numbers	● Informational	setting/SettingManagerLogic.sol (base): 171	● Acknowledged

Description

Literals with many digits are difficult to read and review.

```
171      function _feeDenominator() internal pure virtual returns (uint96) { return
1000000; }
```

Recommendation

We recommend using scientific notation to improve readability.

Alleviation

[82.com, 05/06/2025]: Issue acknowledged. I won't make any changes for the current version.

COI-34 | MISSING EMIT EVENTS

Category	Severity	Location	Status
Coding Style	● Informational	domainNft/DomainNFTLogic.sol (base): 47, 58; trading/NFT OfferMarketLogic.sol (base): 33, 36	● Acknowledged

Description

There should always be events emitted in the sensitive functions that are controlled by centralization roles.

Recommendation

It is recommended emitting events for the sensitive functions that are controlled by centralization roles.

Alleviation

[82.com, 05/06/2025]: Issue acknowledged. I won't make any changes for the current version.

COI-35 | INFORMATION ON UPGRADE HANDLING

Category	Severity	Location	Status
Design Issue	● Informational		● Resolved

Description

Contracts in the `Coin988` protocol are upgradeable. Please clarify whether the source code provided is for an upgrade of an existing deployment or whether this is the implementation code for a proxy being deployed for the first time. This information is needed in order to determine whether storage collisions during upgrades should be considered.

Recommendation

If the currently audited codebase is an upgrade to an existing deployment, we recommend providing the address for the currently existing contract logic in order to assess the potential for storage collisions. Otherwise, please confirm this is the contract logic to be used with the first deployment of the project.

Alleviation

[82.com, 04/22/2025]: Yes, all smart contracts are written on the premise of the first deployment.

OPTIMIZATIONS | 82.COM

ID	Title	Category	Severity	Status
COI-01	Redundant Code Components	Volatile Code, Code Optimization	Optimization	● Acknowledged
COI-02	Redundant `burn()` Override	Code Optimization	Optimization	● Acknowledged
COI-03	Unused Declarations Increase Contract Size And Reduce Clarity	Code Optimization	Optimization	● Acknowledged
COI-04	Redundant Length Checks Before Looping Over Token Arrays	Gas Optimization	Optimization	● Acknowledged

COI-01 | REDUNDANT CODE COMPONENTS

Category	Severity	Location	Status
Volatile Code, Code Optimization	● Optimization	trading/NFTOfferMarketLogic.sol (base): 25	● Acknowledged

Description

The `NFTOfferMarketLogic.isWhiteList` modifier is declared but never used.

Recommendation

We advise removing the redundant statement for production environments.

Alleviation

[82.com, 05/06/2025]: Issue acknowledged. I won't make any changes for the current version.

COI-02 | REDUNDANT `burn()` OVERRIDE

Category	Severity	Location	Status
Code Optimization	● Optimization	domainNft/DomainNFTLogic.sol (base): 65~66	● Acknowledged

Description

The `DomainNFTLogic` overrides the internal `_update()` to revert when a token is marked frozen:

```
    function _update(address to, uint256 tokenId, address auth) internal
override(ERC721Upgradeable, ERC721EnumerableUpgradeable) returns (address) {
        if (isFrozenTokenId[tokenId]) { revert ERROR_NFT_FROZEN(); }
        return super._update(to, tokenId, auth);
    }
```

It also provides a public `burn(uint256)` override that performs the same freeze check before calling `super._update()` :

```
    function burn(uint256 _tokenId) public override {
        if (isFrozenTokenId[_tokenId]) { revert ERROR_NFT_FROZEN(); }
        super._update(address(0), _tokenId, _msgSender());
    }
```

However, the inherited `ERC721Burnable.burn()` already calls `_update(address(0), tokenId, _msgSender())` , which dispatches to the custom `_update` override. Thus, the `DomainNFTLogic.burn()` is redundant and can be removed.

Recommendation

We recommend removing the redundant public `burn(uint256)` override.

Alleviation

[82.com, 05/06/2025]: Issue acknowledged. I won't make any changes for the current version.

COI-03 | UNUSED DECLARATIONS INCREASE CONTRACT SIZE AND REDUCE CLARITY

Category	Severity	Location	Status
Code Optimization	● Optimization	domainNft/DomainNFTLogic.sol (base): 24, 25, 27; safe/Safe.sol (base): 26, 30, 31; setting/SettingManagerLogic.sol (base): 36; trading/NFTOfferMarketLogic.sol (base): 18, 220; transfer/TransferAgentLogic.sol (base): 17	● Acknowledged

Description

Several declared errors, events, variables, and mappings are never used throughout the contract, resulting in unnecessary code bloat that increases contract size, reduces readability, and may incur higher deployment costs without adding any functional value.

Recommendation

We recommend removing all unused declarations to optimize contract size, improve readability, and reduce deployment costs.

Alleviation

[82.com, 05/06/2025]: Issue acknowledged. I won't make any changes for the current version.

COI-04 | REDUNDANT LENGTH CHECKS BEFORE LOOPING OVER TOKEN ARRAYS

Category	Severity	Location	Status
Gas Optimization	● Optimization	safe/SafeProxy.sol (base): 33~42	● Acknowledged

Description

The checks `if (erc20Tokens.length > 0)` and `if (erc721Tokens.length > 0)` are redundant because looping over an empty array naturally results in zero iterations without causing errors or consuming unnecessary gas. These conditionals add extra bytecode and complexity without providing functional benefits, as the `for` loops are already safe to execute even when the arrays are empty.

Recommendation

We recommend removing the redundant length checks before the for loops, as empty arrays will not enter the loop and are safe to iterate over directly.

Alleviation

[82.com, 05/06/2025]: Issue acknowledged. I won't make any changes for the current version.

FORMAL VERIFICATION | 82.COM

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 formal verification to prove that important functions in the smart contracts adhere to their expected behaviors.

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 contracts derived from AccessControl v4.4

We verified properties of the public interface of contracts that provide an AccessControl-v4.4 compatible API. This involves:

- The `hasRole` function, which returns `true` if an account has been granted a specific `role`.
- The `getRoleAdmin` function, which returns the admin role that controls a specific `role`.
- The `grantRole` and `revokeRole` functions, which are used for granting a `role` to an account and revoking a `role` from an `account`, respectively.
- The `renounceRole` function, which allows the calling account to revoke a `role` from itself.

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

Property Name	Title
accesscontrol-getroleadmin-succeed-always	`getRoleAdmin` Function Always Succeeds
accesscontrol-hasrole-succeed-always	`hasRole` Function Always Succeeds
accesscontrol-hasrole-change-state	`hasRole` Function Does Not Change State
accesscontrol-renouncerole-succeed-role-renouncing	`renounceRole` Successfully Renounces Role
accesscontrol-getroleadmin-change-state	`getRoleAdmin` Function Does Not Change State
accesscontrol-grantrole-correct-role-granting	`grantRole` Correctly Grants Role
accesscontrol-revokerole-correct-role-revoking	`revokeRole` Correctly Revokes Role
accesscontrol-default-admin-role	AccessControl Default Admin Role Invariance
accesscontrol-renouncerole-revert-not-sender	`renounceRole` Reverts When Caller Is Not the Confirmation Address

Verification Results

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

- False: The property is violated by the project.
- Inconclusive: The proof engine cannot prove or disprove the property due to timeouts or exceptions.
- Inapplicable: The property does not apply to the project.

Detailed Results For Contract SettingManagerLogic (setting/SettingManagerLogic.sol) In Commit 5a43734e468fe7c70a6700c0531a8b79f705e4de

Verification of contracts derived from AccessControl v4.4

Detailed Results for Function `getRoleAdmin`

Property Name	Final Result	Remarks
accesscontrol-getroleadmin-succeed-always	● True	
accesscontrol-getroleadmin-change-state	● True	

Detailed Results for Function `hasRole`

Property Name	Final Result	Remarks
accesscontrol-hasrole-succeed-always	● True	
accesscontrol-hasrole-change-state	● True	

Detailed Results for Function `renounceRole`

Property Name	Final Result	Remarks
accesscontrol-renouncerole-succeed-role-renouncing	● True	
accesscontrol-renouncerole-revert-not-sender	● True	

Detailed Results for Function `grantRole`

Property Name	Final Result	Remarks
accesscontrol-grantrole-correct-role-granting	● True	

Detailed Results for Function `revokeRole`

Property Name	Final Result	Remarks
accesscontrol-revokerole-correct-role-revoking	● True	

Detailed Results for Function `DEFAULT_ADMIN_ROLE`

Property Name	Final Result	Remarks
accesscontrol-default-admin-role	● Inconclusive	

APPENDIX | 82.COM

Finding Categories

Categories	Description
Gas Optimization	Gas Optimization findings do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.
Coding Style	Coding Style findings may not affect code behavior, but indicate areas where coding practices can be improved to make the code more understandable and maintainable.
Magic Numbers	Magic Number findings refer to numeric literals that are expressed in the code in their raw format, but should instead be declared as constants to improve readability and maintainability.
Language Version	Language Version findings indicate that the code uses certain compiler versions or language features with known security issues.
Coding Issue	Coding Issue findings are about general code quality including, but not limited to, coding mistakes, compile errors, and performance issues.
Denial of Service	Denial of Service findings indicate that an attacker may prevent the program from operating correctly or responding to legitimate requests.
Concurrency	Concurrency findings are about issues that cause unexpected or unsafe interleaving of code executions.
Access Control	Access Control findings are about security vulnerabilities that make protected assets unsafe.
Volatile Code	Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases and may result in vulnerabilities.
Logical Issue	Logical Issue findings indicate general implementation issues related to the program logic.
Centralization	Centralization findings detail the design choices of designating privileged roles or other centralized controls over the code.
Design Issue	Design Issue findings indicate general issues at the design level beyond program logic that are not covered by other finding categories.

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. Each such contract was compiled into a mathematical model that 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.

The following assumptions and simplifications apply to our model:

- 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 specifications

All properties are expressed in a behavioral interface specification language that CertiK has developed for Solidity, which allows us to specify the behavior of each function in terms of the contract state and its parameters and return values, as well as contract properties that are maintained by every observable state transition. Observable state transitions occur when the contract's external interface is invoked and the invocation does not revert, and when the contract's Ether balance is changed by the EVM due to another contract's "self-destruct" invocation. The specification language has the usual Boolean connectives, as well as the operator `\old` (used to denote the state of a variable before a state transition), and several types of specification clause:

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

- `requires [cond]` - the condition `cond`, which refers to a function's parameters, return values, and contract state variables, must hold when a function is invoked in order for it to exhibit a specified behavior.
- `ensures [cond]` - the condition `cond`, which refers to a function's parameters, return values, and both `\old` and current contract state variables, is guaranteed to hold when a function returns if the corresponding requires condition held when it was invoked.
- `invariant [cond]` - the condition `cond`, which refers only to contract state variables, is guaranteed to hold at every observable contract state.
- `constraint [cond]` - the condition `cond`, which refers to both `\old` and current contract state variables, is guaranteed to hold at every observable contract state except for the initial state after construction (because there is no previous state); constraints are used to restrict how contract state can change over time.

Description of the Analyzed AccessControl-v4.4 Properties

Properties related to function `getRoleAdmin`

accesscontrol-getroleadmin-change-state

The `getRoleAdmin` function must not change any state variables.

Specification:

```
assignable \nothing;
```

accesscontrol-getroleadmin-succeed-always

The `getRoleAdmin` function must always succeed, assuming that its execution does not run out of gas.

Specification:

```
reverts_only_when false;
```

Properties related to function `hasRole`

accesscontrol-hasrole-change-state

The `hasRole` function must not change any state variables.

Specification:

```
assignable \nothing;
```

accesscontrol-hasrole-succeed-always

The `hasRole` function must always succeed, assuming that its execution does not run out of gas.

Specification:

```
reverts_only_when false;
```

Properties related to function `renounceRole`

accesscontrol-renouncerole-revert-not-sender

The `renounceRole` function must revert if the caller is not the same as `account`.

Specification:

```
reverts_when account != msg.sender;
```

accesscontrol-renouncerole-succeed-role-renouncing

After execution, `renounceRole` must ensure the caller no longer has the renounced role.

Specification:

```
ensures !hasRole(role, account);
```

Properties related to function `grantRole`

accesscontrol-grantrole-correct-role-granting

After execution, `grantRole` must ensure the specified account has the granted role.

Specification:

```
ensures hasRole(role, account);
```

Properties related to function `revokeRole`

accesscontrol-revokerole-correct-role-revoking

After execution, `revokeRole` must ensure the specified account no longer has the revoked role.

Specification:

```
ensures !hasRole(role, account);
```

Properties related to function `DEFAULT_ADMIN_ROLE`

accesscontrol-default-admin-role

The default admin role must be invariant, ensuring consistent access control management.

Specification:

```
invariant DEFAULT_ADMIN_ROLE() == 0x00;
```

DISCLAIMER | CERTIK

Elevating Your Entire Web3 Journey