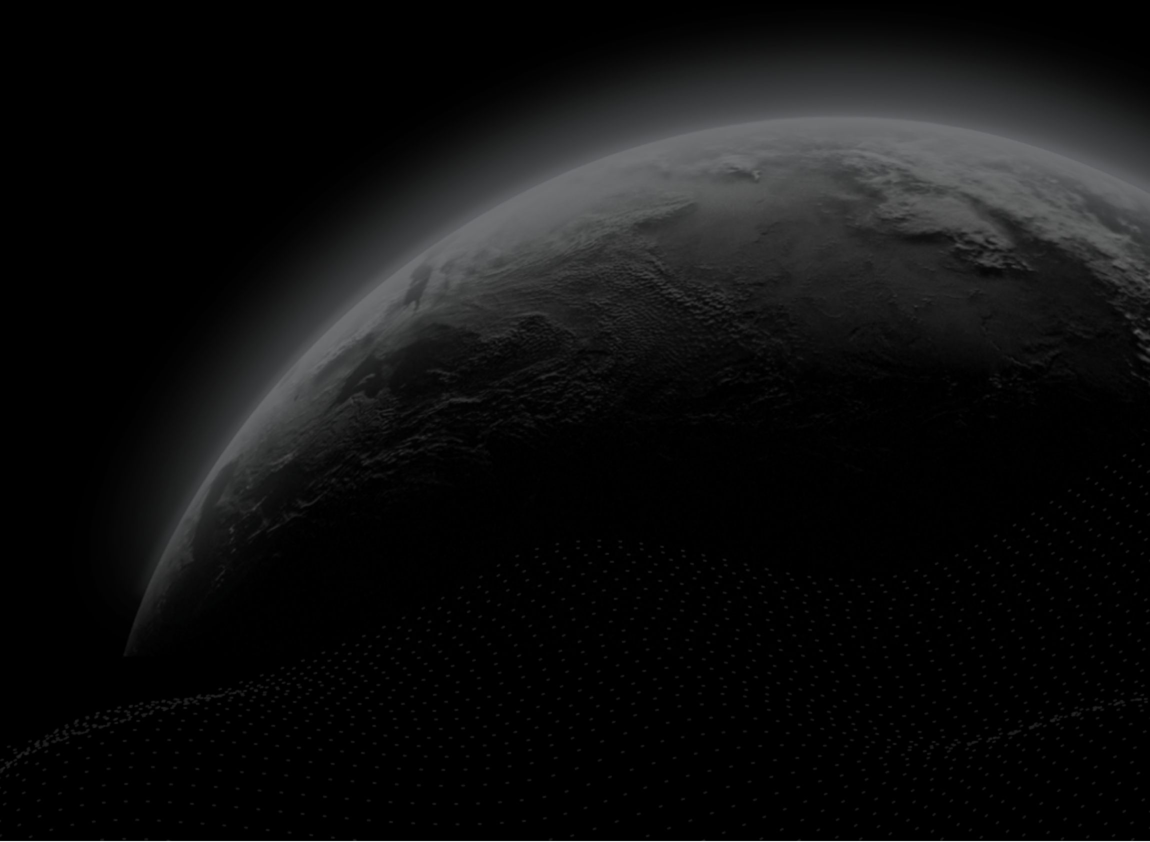
CERTIK

Security Assessment

Unseen - Audit

CertiK Assessed on Aug 5th, 2024



CertiK Assessed on Aug 5th, 2024

Unseen - Audit

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

Executive Summary

TYPES
DeFi

ECOSYSTEM
EVM Compatible

METHODS
Formal Verification, Manual Review, Static Analysis

LANGUAGE
Solidity

TIMELINE
Delivered on 08/05/2024

KEY COMPONENTS
N/A

CODEBASE
github.com/Rektstudios/unseen-audits
View All in Codebase Page

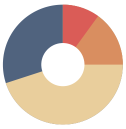
COMMITS
b0e0abf927c12a15cc878213e26eec6d94734570
View All in Codebase Page

Highlighted Centralization Risks

ⓘ Transfers can be paused

ⓘ Initial multi-sig address token share is 100%

Vulnerability Summary



20	15	1	0	4	0
Total Findings	Resolved	Mitigated	Partially Resolved	Acknowledged	Declined

🟥 2	Critical	1 Resolved, 1 Mitigated	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.
🟧 3	Major	1 Resolved, 2 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.
🟨 0	Medium		Medium risks may not pose a direct risk to users' funds, but they can affect the overall functioning of a platform.
🟨 9	Minor	8 Resolved, 1 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.
🟦 6	Informational	5 Resolved, 1 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 | UNSEEN - AUDIT

Summary

Findings

Optimizations

Formal Verification

CODEBASE | UNSEEN - AUDIT

Repository

github.com/Rektstudios/unseen-audits

Commit

b0e0abf927c12a15cc878213e26eec6d94734570

AUDIT SCOPE | UNSEEN - AUDIT

51 files audited ● 13 files with Acknowledged findings ● 7 files with Resolved findings ● 31 files without findings

ID	Repo	File	SHA256 Checksum
● BFR	Rektstudios/unseen-audits	contracts/fee-collector/BaseFeeCollector.sol	dbbf46593a2a17e204d86768d0d27e14f2b117d4fe1e8ca0ece38a28f51c84f6
● FEO	Rektstudios/unseen-audits	contracts/fee-collector/FeeCollector.sol	84685bd8df4ced2229c28b24c121b9f741b7cbd0cba8c6bad243f8341a667cf4
● EXA	Rektstudios/unseen-audits	contracts/marketplace/exchange/ExchangeCore.sol	878a2b9491e2e186b8b644f6a234fd7bdec1cad68722b252d6a9439f504b2da4
● AUT	Rektstudios/unseen-audits	contracts/marketplace/registry/AuthenticatedProxy.sol	1c92a614a38a9503d84a72ba93da9a429dccc8095d2f9e36d07f2cedbf3c9bf2
● PRO	Rektstudios/unseen-audits	contracts/marketplace/registry/ProxyRegistry.sol	fc3ff62744e26bcd382855b9ab74077842074319629911f7e3b31fa9ea885d08
● UNR	Rektstudios/unseen-audits	contracts/marketplace/UnseenRegistry.sol	4ba9ca82958adbd28f96b6b5e5d7fadbdcbb148765cb92bed45c08f1e0f01bfa
● TTR	Rektstudios/unseen-audits	contracts/thegenerates/abstract/TokenTransferValidator.sol	431d5d0b66ccb340503d1634b608f17d408b893c0ed663768653805deb9aea87
● CMR	Rektstudios/unseen-audits	contracts/thegenerates/extensions/ContractMetadata.sol	d645e241e0077c21e9e557dd3b2a07fb2167b3a378df163c123ea9f630064d32
● TGE	Rektstudios/unseen-audits	contracts/thegenerates/extensions/TGenContract.sol	5011eb628684d69c8e58be996874391d06c54f3debbd38958d49aabb33987e47
● THN	Rektstudios/unseen-audits	contracts/thegenerates/TheGenerates.sol	75d86a2528d78a9ac26296f0300d4371e2619e102913b22d87f7d84a55f31910
● UNU	Rektstudios/unseen-audits	contracts/uncn/uncn.sol	afaf6d1f5bc045ac44a05d272a22728abf91c8cd84c2bc609efd3135f96829cd
● VLR	Rektstudios/unseen-audits	contracts/vesting/abstracts/VestingLockup.sol	312a241e58eb75c50f3d0e4d58ff2153aa769d1f4f2015d1a2bdff172e410fc2
● UVR	Rektstudios/unseen-audits	contracts/vesting/UnseenVesting.sol	e400874ce22f0c6abe1dd0878e513c68d6d6a593ec0096f6a86a153bcd3546a6

ID	Repo	File	SHA256 Checksum
🟢 ERA	Rektstudios/unseen-audits	contracts/extensions/ERC4907A.sol	d3eea3ac6856fb4371b97ded5d30a7277c0d2aa14a0e4532e247fa405dc4729b
🟢 ARR	Rektstudios/unseen-audits	contracts/lib/ArrayUtils.sol	2f64edd108e6cb934acd085568782667eabf8b45668a656be6c7a4c36aad6f0b
🟢 SMR	Rektstudios/unseen-audits	contracts/marketplace/staticMarket/StaticMarket.sol	ca0a91e25f60172b30d263a5d30002ac04fa4ec44d6ff8024441d0d1950f5ec7
🟢 GLO	Rektstudios/unseen-audits	contracts/marketplace/GlobalMaker.sol	db4659bc9a50cd6a049764c22be2506653b83fb2f47efb4674d581f7fa0e3464
🟢 UNE	Rektstudios/unseen-audits	contracts/marketplace/UnseenAtomicizer.sol	0c76836392c215e4f29ee41c4bf597edd82f80f180bdbfd757086862bde9fce3
🟢 MAE	Rektstudios/unseen-audits	contracts/thegenerates/abstract/MarketsPreapproved.sol	af1bc668ccbc36fc138d8b722de71d51407a6df54f8fa874635c8b0f009f11cd
🟢 TIR	Rektstudios/unseen-audits	contracts/thegenerates/extensions/TheGeneratesImplementation.sol	f084bd7850f77485abe1972d55c3c4de9f846e33990ac1700e3cc5f0e2179ea8
⚪ IBR	Rektstudios/unseen-audits	contracts/interfaces/IBaseFeeCollector.sol	c438d8d33f0728971f8f660666930d1ad62a2a086581cd158564256a7bd44979
⚪ IBE	Rektstudios/unseen-audits	contracts/interfaces/IBaseFeeCollectorEventsAndErrors.sol	dddab83d1fbec03020b53cd21e9f555dae4c26de6adf70029174467f4ae0bda0
⚪ IEA	Rektstudios/unseen-audits	contracts/interfaces/IERC4907A.sol	722c9833e2382b8dd832651235b28faae94aa7a3e5d3dc0447a9174c109f3c64
⚪ IFR	Rektstudios/unseen-audits	contracts/interfaces/IFeeCollector.sol	3b85ee080f6974c20584dd2ea9c89cd4c9e1e642d23a9a03487555470bcf6843
⚪ RCR	Rektstudios/unseen-audits	contracts/lib/ERC1271.sol	2fc85523576449e71948cada48e0760ac6baa8742344c698dd8774e8d503bd3e
⚪ ECM	Rektstudios/unseen-audits	contracts/lib/ERC1271Mod.sol	542cbbc6491bc9be01e523906c6eaa00f7e582b8b00460f2109799f360a1a7e4
⚪ STA	Rektstudios/unseen-audits	contracts/lib/StaticCaller.sol	96dfdde9a2993cd115728423e534ece41db646df5696415df60ed83fa5674381
⚪ ERB	Rektstudios/unseen-audits	contracts/marketplace/exchange/Exchange.sol	8e5521d02e203a45a544ff8d4cf0cc1c8ade1fee6cf2356e879a0cfa21b532c7

ID	Repo	File	SHA256 Checksum
MEE	Rektstudios/unseen-audits	contracts/marketplace/lib/MarketErrorsAndEvents.sol	03472cdcb1be7d3143677ea349d80ce3e49cfa31dd3dd70ca031d84a79180c74
PIR	Rektstudios/unseen-audits	contracts/marketplace/registry/ProxyRegistryInterface.sol	8e1f42fd2323eead9d4b419a8eeaf7153d7c736556e93442c01c930b74455d4f
UNX	Rektstudios/unseen-audits	contracts/marketplace/UnseenExchange.sol	d81af9b2c040ba31fdbac0c1189ca62d1eb227cef9929f12453b7267c74dd602
UNT	Rektstudios/unseen-audits	contracts/marketplace/UnseenStatic.sol	abc79da8f2925eea7972810c609f0b6eb7d3e833bcb0c8581d7f453fa58a3d45
ICR	Rektstudios/unseen-audits	contracts/thegenerates/interfaces/IContractMetadata.sol	dd160cf31c04362f4f2854bba31ec96b649c0bde8d46cf0abc5ab052a5bba80e
ITR	Rektstudios/unseen-audits	contracts/thegenerates/interfaces/ICreatorToken.sol	c25cbc38e6b96c9949238efab62708a837948bfc665ab0df1c74d5b46eb5094c
IRC	Rektstudios/unseen-audits	contracts/thegenerates/interfaces/IERC173.sol	44a7d6cc39126e6631f0d00a219927a0d14c05b093cc81aae57567958a175656
IGR	Rektstudios/unseen-audits	contracts/thegenerates/interfaces/ITheGenerates.sol	b68ff2b593492376f406aefefc47e6961923673dd0c093732b4165374c7fd632
IGC	Rektstudios/unseen-audits	contracts/thegenerates/interfaces/ITheGeneratesConfigurer.sol	fabb2ad319538bb23baee04cd021e5a056036b936a16a91bb776e6a865e7d2a0
IVR	Rektstudios/unseen-audits	contracts/thegenerates/interfaces/ITransferValidator.sol	7bb61567b857efa5fc5ff985e1aca7e852c087ecc2462c77165a9f8e6fbbb104
EAR	Rektstudios/unseen-audits	contracts/thegenerates/lib/ErrorsAndEvents.sol	39ceadf15d3c13ec90e354f0a3ddff52a721c6def4d66e4794f1e6a4500df020
TSR	Rektstudios/unseen-audits	contracts/thegenerates/lib/TheGeneratesStorage.sol	4c29fc3c95a85e621c379eb04c28f19a99af4fe81fc8d2a36348f0b23140eb4f
DTR	Rektstudios/unseen-audits	contracts/thegenerates/types/DataTypes.sol	f2e71f1abe5f13bd2e3daa20bced1e8bedd93426dae8f5b671c0cdb69a90ca2d
GCR	Rektstudios/unseen-audits	contracts/thegenerates/TheGeneratesConfigurer.sol	d5cbe786660738976a0c9c8c8ee92d19c441f86a57be081196ccdb3480a4c2a2
IUR	Rektstudios/unseen-audits	contracts/vesting/interfaces/IUnseenVesting.sol	9867419ef6b7078bba93c147863330f3a66c96710d3f638ec7c079cd387276b3

ID	Repo	File	SHA256 Checksum
IUF	Rektstudios/unseen-audits	contracts/vesting/interfaces/IUnseenVestingNFTDescriptor.sol	ff15b8a55e1636202471167518b774411cca3586659d7488072266decf25c5cd
ILR	Rektstudios/unseen-audits	contracts/vesting/interfaces/IVestingLockup.sol	b10c15326da1892b6a3bf84005c3e76328106fe0ace75c3c2eb275991506a61c
ERT	Rektstudios/unseen-audits	contracts/vesting/libraries/Errors.sol	d37efbc707226496aca59b164155d435337e3fe5153649f8c2e8eb0ec759b6db
HRB	Rektstudios/unseen-audits	contracts/vesting/libraries/Helpers.sol	524aae8581e2c47ffa363d349f6c774e6b5c5b43c7cc11c697ac9bff78c4d1ae
NFT	Rektstudios/unseen-audits	contracts/vesting/libraries/NFTSVG.sol	2178aa218334985d7ee034c6cb49b6675254fd018b428f614bf12841c512f39b
SVG	Rektstudios/unseen-audits	contracts/vesting/libraries/SVGElements.sol	d52d1ce39cbea611e3bdca72da2040375cd5f067ccbb075540a1677972e3943f
DAY	Rektstudios/unseen-audits	contracts/vesting/types/DataTypes.sol	9fad181c2e2005947cec3949adf5bd8580cab3334b42e051f8e5a43226b549fa
UVN	Rektstudios/unseen-audits	contracts/vesting/UnseenVestingNFTDescriptor.sol	3989c33b1792b3a594a1b2a5a62e087ab9bf92c39d73f5be1571ce3a3912f216

APPROACH & METHODS | UNSEEN - AUDIT

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

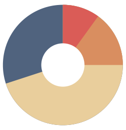
The auditing process pays special attention to the following considerations:

- 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 | UNSEEN - AUDIT



20	2	3	0	9	6
Total Findings	Critical	Major	Medium	Minor	Informational

This report has been prepared to discover issues and vulnerabilities for Unseen - 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
CON-01	User Asset Can Be Taken Over By Another User's Authenticated Proxy	Access Control, Logical Issue	Critical	Resolved
REI-01	**Registry Owner Can Directly Or Indirectly Control Any User's Authenticated Proxy**	**Volatile Code, Centralization**	**Critical**	**Mitigated**
CON-02	**Centralization Related Risks**	**Centralization**	**Major**	**Acknowledged**
UNU-01	Init Function Can Be Called Multiple Times	Volatile Code	Major	Resolved
UNU-02	**Initial Token Distribution**	**Centralization**	**Major**	**Acknowledged**
ARR-01	Potential Memory Overwriting On `guardedArrayReplace()`	Volatile Code	Minor	Resolved
CMR-01	No Upper Limit For `feeNumerator`	Logical Issue	Minor	Resolved
CON-03	Third-Party Dependencies	Volatile Code	Minor	Acknowledged
EXA-02	`Executor` Used Instead Of `Maker` In Order Validation	Inconsistency, Logical Issue	Minor	Resolved
FEC-01	Use `.call()` Instead Of `.send()` Or `.transfer()`	Volatile Code	Minor	Resolved

ID	Title	Category	Severity	Status
GLO-01	Lack Of Zero Address Validation Of `ecrecover()` Return Value	Coding Style	Minor	● Resolved
TIR-01	Public Drop Start Time And End Time Cannot Be The Same	Logical Issue, Inconsistency	Minor	● Resolved
UNE-01	Remaining Native Tokens May Not Be Refunded To Users	Logical Issue	Minor	● Resolved
UVR-01	ERC721 Tokens May Be Minted To A Contract That Can Not Handle Them	Logical Issue, Volatile Code	Minor	● Resolved
CON-04	Missing Emit Events	Coding Style	Informational	● Resolved
CON-05	Functions With `_` As Name Prefix Are Not `private` Or `internal`	Coding Style	Informational	● Resolved
MAK-01	Typos In Comments	Coding Style	Informational	● Resolved
SMR-01	Missing Error Messages	Coding Style	Informational	● Resolved
THG-01	Commingling Of Proxy And Logic Contracts	Coding Style	Informational	● Acknowledged
VLR-01	EIP4906 Not Strictly Followed	Coding Issue	Informational	● Resolved

CON-01 USER ASSET CAN BE TAKEN OVER BY ANOTHER USER'S AUTHENTICATED PROXY

Category	Severity	Location	Status
Access Control, Logical Issue	● Critical	contracts/extensions/ERC4907A.sol (v3): 34; contracts/marketplace/exchange/ExchangeCore.sol (v3): 356; contracts/marketplace/registry/AuthenticatedProxy.sol (v3): 110, 128, 151; contracts/marketplace/registry/ProxyRegistry.sol (v3): 138; contracts/thegenerates/abstract/MarketsPreapproved.sol (v3): 27, 43	● Resolved

Description

The `ProxyRegistry` contract plays an important role in the `TheGenerates` contract. Specifically, in the `MarketsPreapproved` contract (which is inherited by `TheGenerates` contract), the `isApprovedForAll()` function returns `true` if the `operator` is a proxy for the user. Subsequently, the `isAuthorized` modifier defined in the `ERC4907A` contract would pass for the user's `tokenId` , which means that the proxy can control the NFT corresponding to the `tokenId` . The proxy is recorded in the `ProxyRegistry` contract, specifically the `proxies` mapping.

The intended behavior is that a user can only set its own proxy, and not any other user's proxy, so a user can control the NFT either directly or via its proxy. However, the `transferAccessTo()` function in the `ProxyRegistry` contract is capable of updating the `proxies[to]` mapping, and the access control in the `ProxyRegistry` and `AuthenticatedProxy` contracts is insufficient. This enables user Bob to change the `proxies[Alice]` to his own proxy, and subsequently takes control of any `tokenId` that Alice owns via Bob's proxy.

Note that in the `AuthenticatedProxy` contract, the `transferProxyOwnership()` function enables the proxy owner to transfer the ownership to a new owner, and update the mapping in the registry. However, the `AuthenticatedProxy` contract also contains a `proxy()` function and a `proxyAssert()` function that can also call the `transferAccessTo()` function of the Registry, but does NOT change the `owner` variable. This allows Bob to retain ownership of his proxy, while updating the `proxies[Alice]` value in the `ProxyRegistry` contract.

A proof of concept test case is added to the `exchange.registry.spec.ts` file to demonstrate the above.

This vulnerability impacts all other contracts that utilize the registry and proxy contracts, including `TheGenerates` and `UnseenExchange` . For `TheGenerates` , a malicious user could effectively take control of another user's NFT. For `UnseenExchange` , the ability to manipulate the `proxy` address in line 356 of the `ExchangeCore` contract allows a malicious user to brick legitimate atomic match transactions.

Proof of Concept

The following test case is added to the `exchange.registry.spec.ts` file and it passes.

```
  context('new test', function () {

    it('Bob sets Alice proxy recorded at the registry to be his own and retain
ownership of his proxy', async function () {

    // the first 4 bytes are the function selector for the
`transferAccessTo(address,address)` function
      let data = ethers.utils.hexConcat(['0xdcfa9222',
ethers.utils.defaultAbiCoder.encode(['address', 'address'], [bob.address,
alice.address])]);

      const { authProxy } = await getAuthenticatedProxy(bob);

      // confirms that Bob is the owner of the proxy before the transaction
      expect(await authProxy.owner()).to.eq(bob.address);

      // Bob calls the registry.transferAccessTo() function not via the
transferProxyOwnership() function, but via the `proxy()` or `proxyAssert()` function
      await authProxy
          .connect(bob)
          .proxyAssert(registry.address, 0, data);

      // confirms that the registry has been updated, such that proxies[alice] ==
Bob's proxy
      expect(await registry.proxies(alice.address)).to.eq(authProxy.address);

      // confirms that Bob still retains ownership of his proxy
      expect(await authProxy.owner()).to.eq(bob.address);
  });
```

The test result shows that Bob can update the record of Alice at the Registry to be his own proxy

```
  Exchange Registry - (Unseen v1.0.0)
    new test
      ✔ Bob sets Alice proxy recorded at the registry to be his own and retain
ownership of his proxy
```

Recommendation

Consider adding restrictions in the `AuthenticatedProxy` contract that the `dest` address in the `proxy()` and `proxyAssert()` functions cannot be the `unseenRegistry` / `ProxyRegistry` address. When that is true, the attack scenario above would no longer work because the only way Bob can call the `ProxyRegistry` contract is via the `transferProxyOwnership()` function which means that he would also lose ownership of his proxy if he attempts to update the `proxies[Alice]` value in the Registry contract.

Alleviation

[Unseen team, 8/6/2024]: The team heeded the advice and resolved the issue in commits 953e5c922099e39a485713e317218116b795f3a5, 8c2aba83c9b47963b2dff91ebae144f92959acac, and e0e5af13683e4fd4012ef91b8e52d4737d1d6e68.

REI-01 | REGISTRY OWNER CAN DIRECTLY OR INDIRECTLY CONTROL ANY USER'S AUTHENTICATED PROXY

Category	Severity	Location	Status
Volatile Code, Centralization	● **Critical**	**contracts/marketplace/registry/AuthenticatedProxy.sol (v3): 133; contracts/marketplace/registry/ProxyRegistry.sol (v3): 75~83**	● **Mitigated**

Description

Line 133 of the `AuthenticatedProxy` contract is intended to control who can call the `proxy` function. The owner of the proxy can call the `proxy()` function which is intended. Additionally, if `revoked` is `false` (and it defaults to `false` when first deployed) AND the caller is authenticated in the `ProxyRegistry` contract, then the check also passes. In the `ProxyRegistry` contract, the owner can grant authentication to any address, which means that the owner of the `ProxyRegistry` contract can perform arbitrary function call / delegatecall of any user's proxy via an affiliated address. This is a critical risk, as the owner affiliated address can drain the asset of any user from its proxy, or destroy any user's proxy contract via a delegate call to `selfdestrct()` etc.

Proof of Concept

The following test case is added to the `exchange.registry.spec.ts` file to demonstrate that the registry owner can give permission to Alice to drain another user Bob's assets from his proxy.

```
  it('The Registry owner can take over any user proxy by default either directly or
via another user', async function () {

      // User Bob has an authenticated proxy and tranfers 1000 tokens to it
      const { authProxy } = await getAuthenticatedProxy(bob);

      await mockERC20.connect(bob).transfer(authProxy.address, 1000, { gasLimit:
100_000 });
      expect(await mockERC20.balanceOf(authProxy.address)).to.eq(1000);

      // The owner grant another user Alice authentication
      await registry.connect(owner).grantAuthentication(alice.address);
      expect (await registry.contracts(alice.address)).to.eq(true);

      // Alice can transfer Bob's token at his proxy to her own address
      // the first 4 bytes are the function selector for the
`transfer(address,uint256)` function
      let data = ethers.utils.hexConcat(['0xa9059cbb',
ethers.utils.defaultAbiCoder.encode(['address', 'uint256'], [alice.address,
1000])]);

      await authProxy
      .connect(alice)
      .proxyAssert(mockERC20.address, 0, data, { gasLimit: 100_000 });

      // confirms that Alice has drained the token from Bob's proxy
      expect(await mockERC20.balanceOf(alice.address)).to.eq(1000);
      expect(await mockERC20.balanceOf(authProxy.address)).to.eq(0);
  });
```

The test passes showing that Alice has successfully drained Bob's tokens from his proxy.

```
      ✔ The Registry owner can take over any user proxy by default either directly
or via another user (45ms)
```

Recommendation

We recommend removing the ability of the registry owner to grant authentication to arbitrary addresses, and/or only allowing the proxy owner to call the `proxy()` function in the `AuthenticatedProxy` contract.

Alleviation

[Unseen team, 7/26/2024]: Added a 1 week delay in granting authentication in commit https://github.com/Rektstudios/unseen-audits/tree/349f8192fe55b7caa5332d785eb2ecd38a9f0821. Will provide multisig address with all signers once contract is deployed.

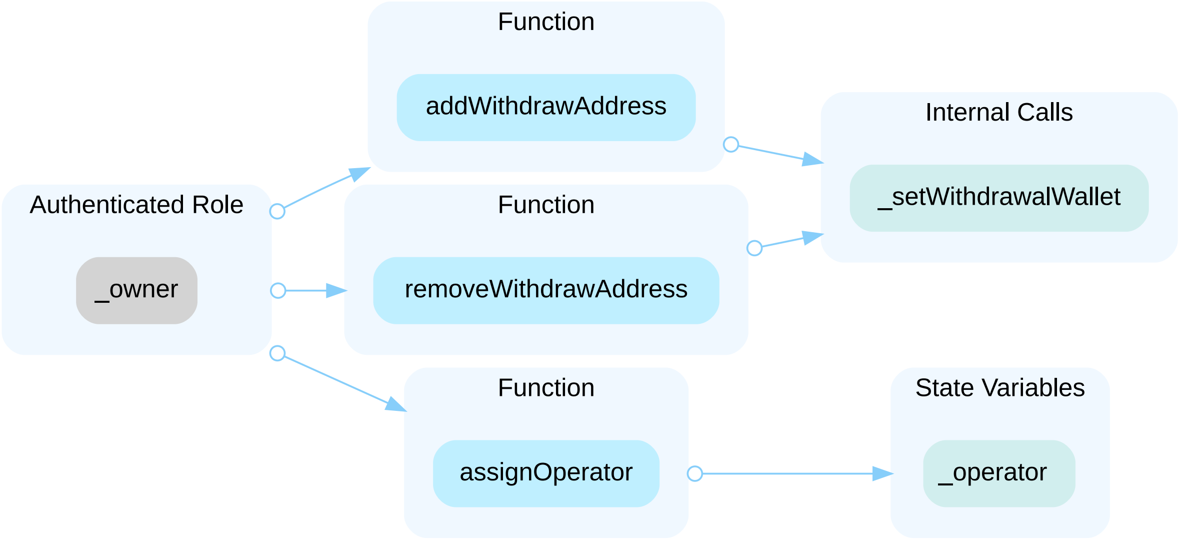
[Certik, 8/5/2024] While this approach has indeed reduced the risk, it's crucial to note that it has not completely eliminated it. We have verified that the deployed contract (https://polygonscan.com/address/0x658447ffdbf61bb431ccd2e84f0af192a9b147cf#readContract) has the multisig (https://polygonscan.com/address/0x8870cD5AED8A586929a11468DdB38d8A1370D509) as the owner, with a 1 week delay in granting authentication.

CON-02 | CENTRALIZATION RELATED RISKS

Category	Severity	Location	Status
Centralization	● **Major**	**contracts/fee-collector/BaseFeeCollector.sol (v3): 128, 144, 155; contracts/fee-collector/FeeCollector.sol (v3): 56; contracts/marketplace/UnseenRegistry.sol (v3): 54; contracts/marketplace/exchange/ExchangeCore.sol (v3): 178, 557, 575; contracts/marketplace/registry/AuthenticatedProxy.sol (v3): 101, 110; contracts/marketplace/registry/ProxyRegistry.sol (v3): 75, 91; contracts/thegenerates/extensions/ContractMetadata.sol (v3): 225, 233; contracts/uncn/uncn.sol (v3): 100, 109, 118, 147; contracts/vesting/UnseenVesting.sol (v3): 371, 401; contracts/vesting/abstracts/VestingLockup.sol (v3): 275**	● **Acknowledged**

Description

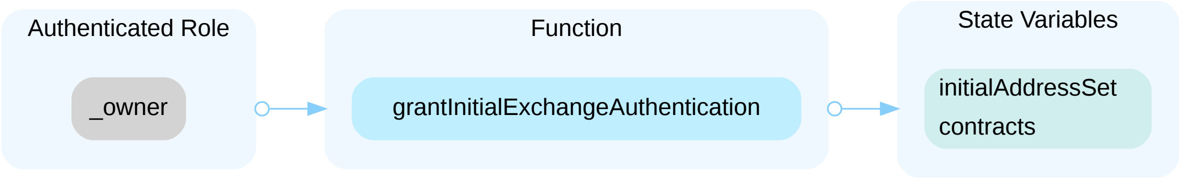
In the contract `BaseFeeCollector` the role `_owner` has authority over the functions shown in the diagram below. Any compromise to the `_owner` account may allow the hacker to take advantage of this authority and change the withdraw address and set `_operator` address.



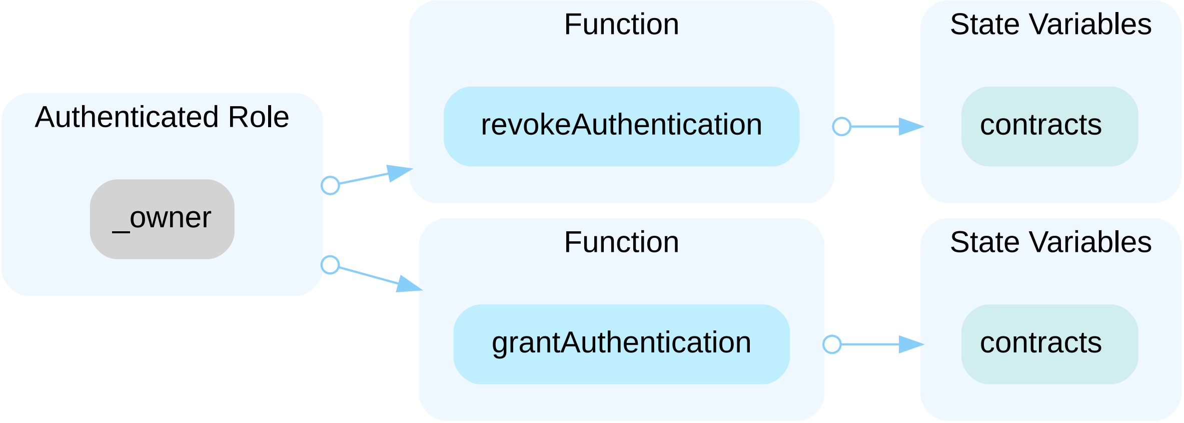
In the contract `BaseFeeCollector` the role `_operator` has authority over the functions `withdrawERC20Tokens()` and `withdraw()`. Any compromise to the `_operator` account may allow the hacker to take advantage of this authority and withdraw ERC20 and native token from the contract.

In the contract `FeeCollector` the role `_operator` has authority over the function `unwrapAndWithdraw()`. Any compromise to the `_operator` account may allow the hacker to take advantage of this authority and unwrap token into its associated native token and withdraw it from the contract.

In the contract `UnseenRegistry` 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 and grant authentication to an arbitrary non-zero address.

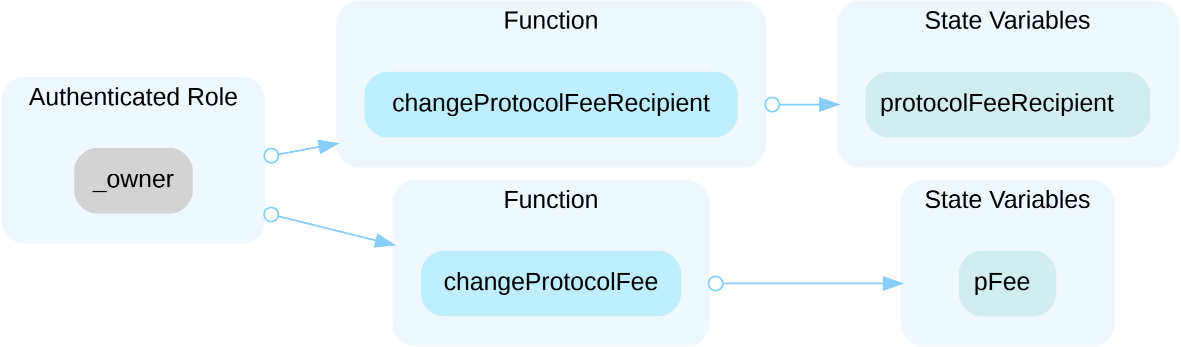


In the contract `ProxyRegistry` the role `_owner` has authority over the functions shown in the diagram below. Any compromise to the `_owner` account may allow the hacker to take advantage of this authority and grant and revoke authentication on any non-zero address.

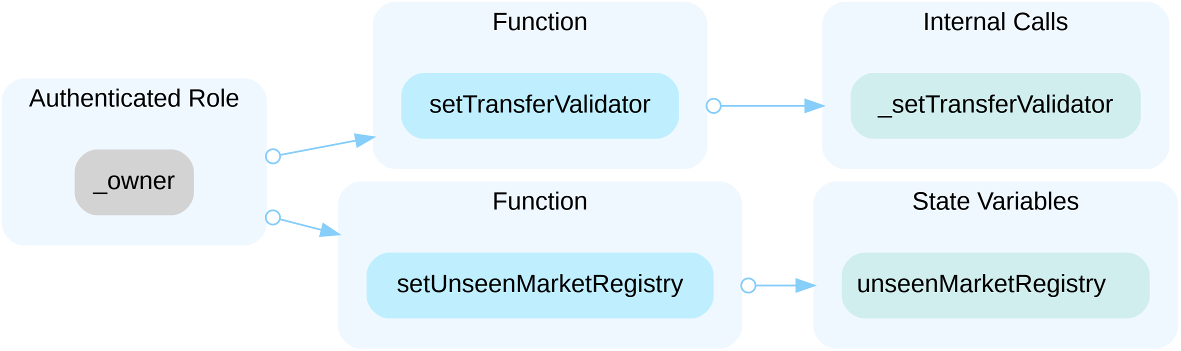


In the contract `AuthenticatedProxy` the role `_proxyowner` has authority over the functions `setRevoke()` , `transferProxyOwnership()` , and `proxy()` . Any compromise to the `_proxyowner` account may allow the hacker to take advantage of this authority and execute arbitrary external call and delegate call.

In the contract `ExchangeCore` 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 and change protocol fee and fee recipient.

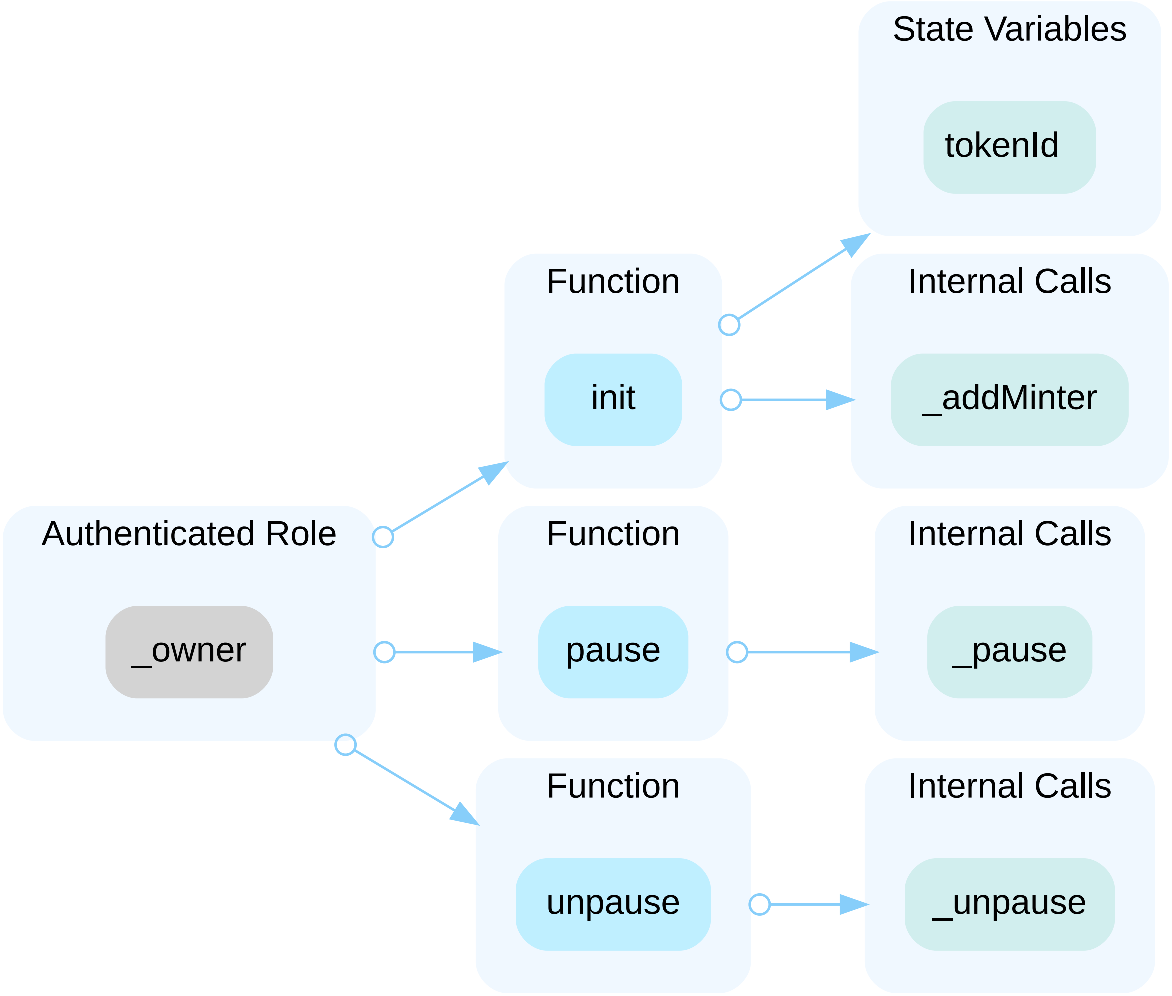


In the contract `ContractMetadata` the role `_owner` has authority over the functions shown in the diagram below. Any compromise to the `_owner` account may allow the hacker to take advantage of this authority and change transfer validator address and `unseenMarketRegistry` address.

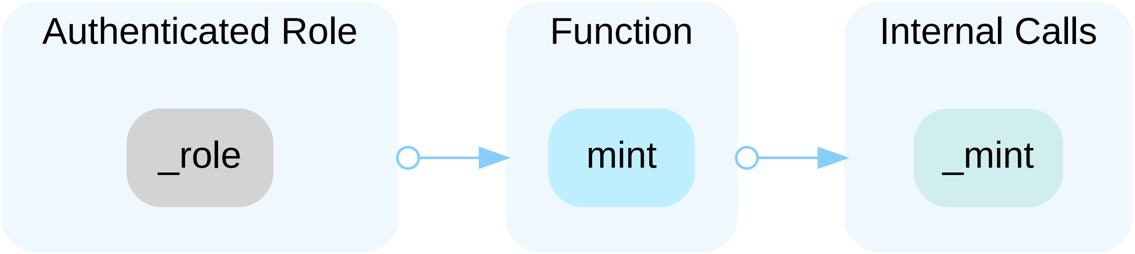


In the contract `ContractMetadata` the role `_owner` and `_CONFIGURER` have authority over the functions `setBaseURI()`, `setContractURI()`, `emitBatchMetadataUpdate()`, `setMaxSupply()`, `setProvenanceHash()`, and `setDefaultRoyalty()`. Any compromise to the `_owner` or `_CONFIGURER` account may allow the hacker to take advantage of this authority and change important project parameters such as the base URI and contract URI, token max supply and default royalties, etc.

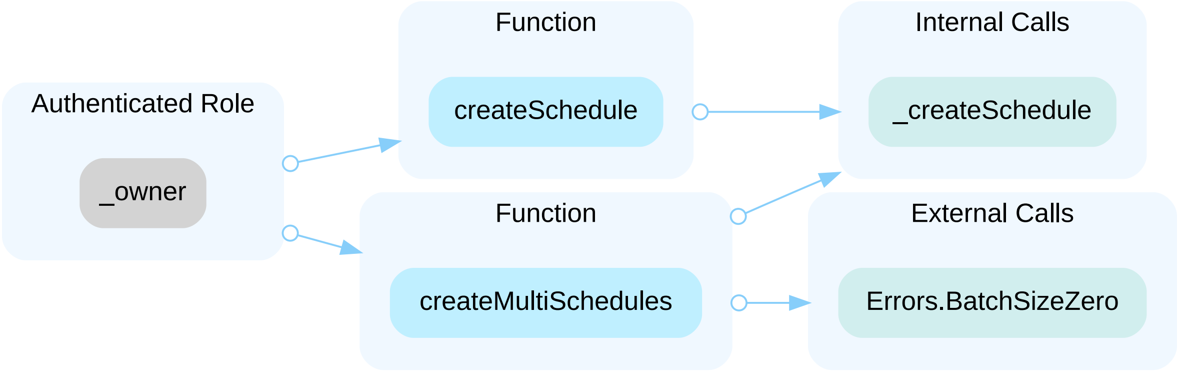
In the contract `UnseenToken` the role `_owner` has authority over the functions shown in the diagram below. Any compromise to the `_owner` account may allow the hacker to take advantage of this authority and pause or unpause the contract, and change `tokenId` and add minter.



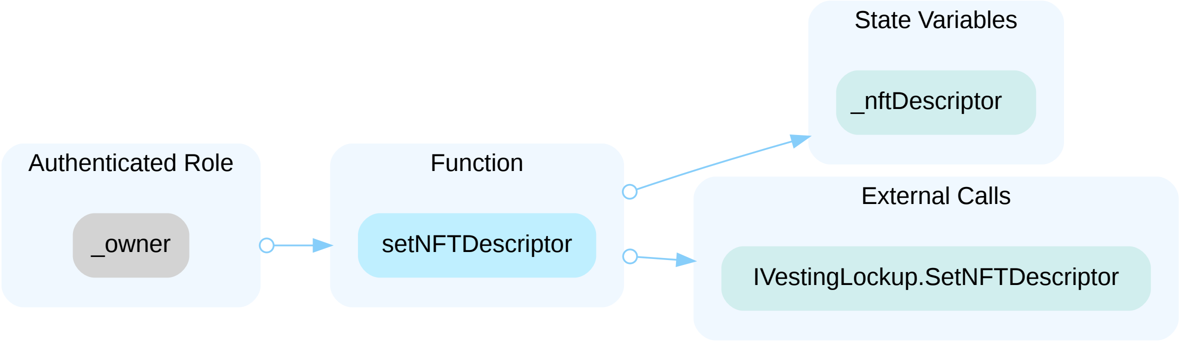
In the contract `UnseenToken` the role `MINTER` has authority over the functions shown in the diagram below. Any compromise to the `_role` account may allow the hacker to take advantage of this authority and mint tokens to arbitrary address.



In the contract `UnseenVesting` 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 and create vesting schedule.



In the contract `VestingLockup` 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 and change NFT descriptor.



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

[Unseen team, 7/26/2024]: After deployment of the protocols . I will provide tx of ownership initialize and will add the multisig on polygon with the signers.

[Certik, 8/5/2024]: We have verified that the owner of the relevant contracts is the multisig address https://polygonscan.com/address/0x8870cD5AED8A586929a11468DdB38d8A1370D509

UNU-01 | INIT FUNCTION CAN BE CALLED MULTIPLE TIMES

Category	Severity	Location	Status
Volatile Code	● Major	contracts/uncn/uncn.sol (v3): 113~123	● Resolved

Description

The `init` function is supposed to be called by the contract owner only one time, as the comment in line 116 indicates. However, the check in line 120 does not prevent the function from being called more than once. As a result, the owner can change the `tokenId` and add new minters at will. This represents a significant risk to the token holders, as a new minter can mint a large amount of tokens and also arbitrarily burn tokens from other addresses.

```
113     /**
114      * @notice Initialize the bridge parmas.
115      *
116      * This can only be called by the contract owner and one time.
117      */
118     function init(bytes32 tokenId_, address tokenManager) external onlyOwner {
119         if (tokenId_ == bytes32(0)) revert TokenIdZero();
120         if (tokenId_ == tokenId) revert TokenIdIsAlreadySet();
121         tokenId = tokenId_;
122         _addMinter(tokenManager);
123     }
```

Proof of Concept

The following test file is added

```
import { loadFixture } from '@nomicfoundation/hardhat-network-helpers';
import { expect } from 'chai';
import { ethers, network } from 'hardhat';
import { UnseenToken } from '@typechained';
import type { Wallet } from 'ethers';
import { randomHex } from '@utils/encoding';
import { faucet } from '@utils/faucet';

describe(`Unseen Token`, async function () {
    const { provider } = ethers;

    let owner: Wallet;
    let interchainTokenService: Wallet;
    let alice: Wallet;
    let bob: Wallet;

    async function setupFixture() {
        owner = new ethers.Wallet(randomHex(32), provider);
        interchainTokenService = new ethers.Wallet(randomHex(32), provider);
        alice = new ethers.Wallet(randomHex(32), provider);
        bob = new ethers.Wallet(randomHex(32), provider);
        for (const wallet of [owner, alice, bob]) {
            await faucet(wallet.address, provider);
          }
        return { owner, interchainTokenService, alice, bob };
      }

    before(async () => {
        ({ owner, interchainTokenService, alice, bob } = await
loadFixture(setupFixture));
    });

    after(async () => {
        await network.provider.request({
          method: 'hardhat_reset',
        });
      });


    it("Init can be called multiple times", async function () {
        const uncn = await ethers.deployContract("UnseenToken", [owner.address,
interchainTokenService.address, 1000]);

        // confirms that the token has been deployed successfully
        expect(await uncn.totalSupply()).to.equal(1000);
        expect(await uncn.owner()).to.equal(owner.address);

        // set token ID to 1 and add Alice as a Minter
        await uncn.connect(owner).init(ethers.utils.formatBytes32String("1"),
alice.address);
        expect(await
uncn.interchainTokenId()).to.eq(ethers.utils.formatBytes32String("1"));
        expect(await uncn.hasRole(alice.address, 0)).to.eq(true);

        // set token ID to 2 and add Bob as a Minter
        await uncn.connect(owner).init(ethers.utils.formatBytes32String("2"),
bob.address);
        expect(await
uncn.interchainTokenId()).to.eq(ethers.utils.formatBytes32String("2"));
        expect(await uncn.hasRole(alice.address, 0)).to.eq(true);
        expect(await uncn.hasRole(bob.address, 0)).to.eq(true);
    });
});
```

The test passes

```
  Unseen Token
    ✔ Init can be called multiple times (120ms)


  1 passing (2s)
```

Recommendation

We recommend fixing the check in line 120, such that the `init` function can only be called once. As an example:

```
120          if (tokenId !=  bytes32(0)) revert TokenIdIsAlreadySet();
```

Alleviation

[Unseen team, 7/26/2024]: Removed the init function in commit https://github.com/Rektstudios/unseen-audits/tree/349f8192fe55b7caa5332d785eb2ecd38a9f0821.

UNU-02 | INITIAL TOKEN DISTRIBUTION

Category	Severity	Location	Status
Centralization	● **Major**	**contracts/uncn/uncn.sol (v3): 92**	● **Acknowledged**

Description

All `_initialSupply` of the `UNCN` tokens are sent to the contract owner address. This is a centralization risk because the owner can distribute tokens without obtaining the consensus of the community. Any compromise to these addresses may allow a hacker to steal and sell tokens on the market, resulting in severe damage to the project.

Recommendation

It is recommended that the team be transparent regarding the initial token distribution process. The token distribution plan should be published in a public location that the community can access. The team should make efforts to restrict access to the private keys of the deployer account or EOAs. A multi-signature (⅔, ⅗) wallet can be used to prevent a single point of failure due to a private key compromise. Additionally, the team can lock up a portion of tokens, release them with a vesting schedule for long-term success, and deanonymize the project team with a third-party KYC provider to create greater accountability.

Alleviation

[Unseen team, 7/26/2024]: https://playunseen.com has the tokenomics, its gonna be updated but always available for public.

Multisig that will hold tokens after deployment :

0x8870cD5AED8A586929a11468DdB38d8A1370D509 (POLYGON)

signer 1:

0x19c52c2e39dF7D4403259E85062090Abf12453b1

signer 2:

0x77518eC664BBD77ca449510508E68055fc53e1EA

signer 3:

0x57384Ce0Aff4d4a390899B333a3d2ccE67e3928e

[Unseen team, 8/4/2024]: Token address on polygon :

https://polygonscan.com/address/0xf2b028ed5977f136982fdfa429814cf19f09693f#readContract

[Certik, 8/5/2024]: We have verified that the token owner and the recipient of all initial tokens is the referenced multisig address, and the 3 signer addresses match the above.

ARR-01 | POTENTIAL MEMORY OVERWRITING ON `guardedArrayReplace()`

Category	Severity	Location	Status
Volatile Code	● Minor	contracts/lib/ArrayUtils.sol (v3): 32~33, 57~58	● Resolved

Description

In the library `ArrayUtils`, there is a function `guardedArrayReplace()` that is used to replace bytes in an array with bytes in another array, guarded by a bitmask. This function should process the array in two steps: quotient and remainder.

The local variable `words` is used to separate the two steps, and it is calculated by `array.length / 0x20`. The first quotient step works well following this calculation. However, the later remainder case also uses this value to determine if there is a set of leftover bytes by checking `words != 0`. Since `words` holds the quotient, the remainder is not properly handled, causing issues when the function processes the `words != 0` condition.

This bug can lead to arbitrary storage writes. If `array.length` is exactly divisible by `0x20`, the copying happens correctly within the loop. In other cases, the function enters the flawed logic and attempts to copy an extra word beyond the array bounds, leading to out-of-bounds access. This incorrect logic can be exploited to overwrite memory at the end of the target array, potentially affecting any unknown memory area.

Recommendation

We recommend properly reviewing the design and fixing the flawed handling of the remainder.

Alleviation

[Unseen team, 7/22/2024]: The team heeded the advice and resolved the issue in commit: https://github.com/Rektstudios/unseen-audits/commit/953e5c922099e39a485713e317218116b795f3a5

CMR-01 | NO UPPER LIMIT FOR `feeNumerator`

Category	Severity	Location	Status
Logical Issue	● Minor	contracts/thegenerates/extensions/ContractMetadata.sol (v3): 216	● Resolved

Description

The function `setDefaultRoyalty` receives the `feeNumerator` is used for royalty payment. In the marketplaces which support EIP-2981, `salePrice * ( feeNumerator/10000`) will be sent to the royalty recipient. But there's no upper limit for `feeNumerator` when setting the default royalty and it can be as high as 100%.

Recommendation

We recommend setting a reasonable upper limit of `FeeRate` such as 1000. (10%)

Alleviation

[Unseen team, 7/22/2024]: The team heeded the advice and resolved the issue in commit: https://github.com/Rektstudios/unseen-audits/commit/953e5c922099e39a485713e317218116b795f3a5

CON-03 | THIRD-PARTY DEPENDENCIES

Category	Severity	Location	Status
Volatile Code	● Minor	contracts/thegenerates/abstract/TokenTransferValidator.sol (v3): 23~30; contracts/thegenerates/extensions/ContractMetadata.sol (v3): 339~347; contracts/uncn/uncn.sol (v3): 86	● Acknowledged

Description

The code base contains interaction with third-party or out-of-scope contracts. For example, the `TokenTransferValidator` contract contains a setter function that sets the `_transferValidator` address, and the `ContractMetadata` contract utilizes the `_transferValidator` in the `_beforeTokenTransfers` hook. If the external `_transferValidator` contract fails to properly validate token transfers, then users' token transfer could be adversely impacted. Additionally, the `UnseenToken` contract interacts with third-party `interchainTokenService_` contract that is out of the scope of this audit.

The scope of the audit treats third-party/out-of-scope entities as black boxes and assumes their functional correctness. However, in the real world, third parties can be compromised and this may lead to lost or stolen assets.

Recommendation

We recommend that the project team constantly monitor the functionality of third-party or out-of-scope contracts to mitigate any side effects that may occur when unexpected changes are introduced.

Alleviation

[Unseen team, 7/26/2024]: TokenTransferValidator is to enforce royalties on other marketplaces. Is it also used on top of our anti-cheat system to protect draining our reward wallet from Gamers who cheat. So a bot will trigger the freeze of that token until we investigate the case. interchainTokenService_ is removed.

EXA-02 `Executor` USED INSTEAD OF `Maker` IN ORDER VALIDATION

Category	Severity	Location	Status
Inconsistency, Logical Issue	● Minor	contracts/marketplace/exchange/ExchangeCore.sol (v3): 173~261, 459, 471	● Resolved

Description

The `validateOrderAuthorization()` function is intended to "validate if maker signed the order hash" (comment in line 173). However, in line 459 and 471, the `executor` address is passed in to the function instead. If the `maker` and `executor` are different addresses, this represents the inconsistency between the intended design and the actual behavior of the code.

Recommendation

Recommend conforming the code with the intended design.

Alleviation

[Unseen team, 7/26/2024]: The executor is either maker if the maker intend to use his proxy to trade , or global maker kind for a shared proxy. If the executer passed to validateOrderAuthorization is a contract then we do the checks `isValidSignature()` to make sure the "from" param in the calldata which will be used in call execution , is equal to the recovered signer. Additional test cases are included in https://github.com/Rektstudios/unseen-audits/commit/349f8192fe55b7caa5332d785eb2ecd38a9f0821

FEC-01 | USE `.call()` INSTEAD OF `.send()` OR `.transfer()`

Category	Severity	Location	Status
Volatile Code	● Minor	contracts/fee-collector/BaseFeeCollector.sol (v3): 119; contracts/fee-collector/FeeCollector.sol (v3): 96	● Resolved

Description

The `.send()` function intends to transfer an ETH amount with a fixed amount of 2300 gas. This function is not equipped to handle changes in the underlying `.send()` and `.transfer()` functions which may supply different amounts of gas in the future. Additionally, if the recipient implements a fallback function containing some sort of logic, this may inevitably revert, meaning the vault and owner of the contract will never be able to call certain sensitive functions.

Reference: Solidity: Sending Ether (transfer, send, call)

Recommendation

Consider using `.call()` instead with the checks-effects-interactions pattern implemented correctly. Careful consideration needs to be made to prevent reentrancy.

Alleviation

[Unseen team, 7/22/2024]: The team heeded the advice and resolved the issue in commit: https://github.com/Rektstudios/unseen-audits/commit/953e5c922099e39a485713e317218116b795f3a5

GLO-01 | LACK OF ZERO ADDRESS VALIDATION OF `ecrecover()` RETURN VALUE

Category	Severity	Location	Status
Coding Style	● Minor	contracts/marketplace/GlobalMaker.sol (v3): 82	● Resolved

Description

The `ecrecover()` function can return the zero address (0x0000000000000000000000000000000000000000) if the signature was invalid or the message was malformed. When the smart contract does not check the output of `ecrecover()` and assumes it is always a valid address, vulnerability may arise.

Recommendation

We recommend adding sanity validation for the return data of `ecrecover()` to ensure that the return address is not the zero address unless the zero address is a valid and intended result within the contract's logic.

We would suggest using OpenZeppelin's ECDSA Library contract as it implements correctly recovering the address from the signature.

Alleviation

[Unseen team, 7/22/2024]: The team heeded the advice and resolved the issue in commit: https://github.com/Rektstudios/unseen-audits/commit/953e5c922099e39a485713e317218116b795f3a5

TIR-01 | PUBLIC DROP START TIME AND END TIME CANNOT BE THE SAME

Category	Severity	Location	Status
Logical Issue, Inconsistency	● Minor	contracts/thegenerates/extensions/TheGeneratesImplementation.sol (v3): 451~467, 568~573	● Resolved

Description

The `_checkActive()` function would revert if `startTime == endTime` , which indicates that the `startTime` and `endTime` of any mint (Public, AllowList, or Signed) cannot be the same. However, in the `updatePublicDrop()` function, the check in line 568 - 573 allows `publicDrop.startTime == publicDrop.endTime` , which is inconsistent with the minting logic. This means that it is possible to set public drop parameters that is guaranteed to fail when users attempt to mint.

Recommendation

Consider updating the condition in line 568 to `if (publicDrop.startTime >= publicDrop.endTime)`

Alleviation

[Unseen team, 7/22/2024]: The team heeded the advice and resolved the issue in commit: https://github.com/Rektstudios/unseen-audits/commit/953e5c922099e39a485713e317218116b795f3a5

UNE-01 | REMAINING NATIVE TOKENS MAY NOT BE REFUNDED TO USERS

Category	Severity	Location	Status
Logical Issue	● Minor	contracts/marketplace/UnseenAtomicizer.sol (v3): 50	● Resolved

Description

When a contract function accepts native tokens and executes a `.call()` or similar function that also forwards native tokens, it's important to check for and refund any remaining balance. This is because some of the supplied value may not be used during the call execution due to gas constraints, a revert in the called contract, or simply because not all the value was needed. If this remaining balance is not refunded, it will be locked in the contract.

Recommendation

We recommended either returning the remaining balance to the sender or handling it in a way that ensures it is not permanently stuck. Neglecting to do so can lead to loss of funds and degradation of the contract's reliability. Furthermore, it's good practice to ensure fairness and trust with your users by returning unused funds.

Alleviation

[Unseen team, 7/26/2024]: Removed the `payable` keyword in commit https://github.com/Rektstudios/unseen-audits/tree/349f8192fe55b7caa5332d785eb2ecd38a9f0821.

UVR-01 | ERC721 TOKENS MAY BE MINTED TO A CONTRACT THAT CAN NOT HANDLE THEM

Category	Severity	Location	Status
Logical Issue, Volatile Code	● Minor	contracts/vesting/UnseenVesting.sol (v3): 371, 401, 755	● Resolved

Description

When the owner creates a new schedule either via `createSchedule()` or `createMultiSchedules()` , there is no check that the `params.recipient` can handle the received ERC721. If `params.recipient` is a smart contract that cannot handle the ERC721, the corresponding vesting schedule could be unwithdrawable.

Recommendation

Consider using `_safeMint()` from the OpenZeppelin ERC721 implementation instead of `_mint()` . If the `params.recipient` refers to a smart contract, it must implement {IERC721Receiver-onERC721Received} for the transaction to succeed. Note that due to the `onERC721Received` call, this could introduce the risk of reentrancy from `params.recipient` , so reentrancy protection is also recommended.

Alleviation

[Unseen team, 7/22/2024]: The team heeded the advice and resolved the issue in commit: https://github.com/Rektstudios/unseen-audits/commit/953e5c922099e39a485713e317218116b795f3a5

CON-04 | MISSING EMIT EVENTS

Category	Severity	Location	Status
Coding Style	● Informational	contracts/extensions/ERC4907A.sol (v3): 96~103, 113~119, 129~150; contracts/fee-collector/BaseFeeCollector.sol (v3): 128~138, 144~149; contracts/marketplace/exchange/ExchangeCore.sol (v3): 382~409; contracts/marketplace/registry/ProxyRegistry.sol (v3): 138~154; contracts/thegenerates/extensions/ContractMetadata.sol (v3): 225~228, 233~240; contracts/uncn/uncn.sol (v3): 100~102, 109~111, 118~123, 147~152; contracts/vesting/UnseenVesting.sol (v3): 371~394, 401~406; contracts/vesting/abstracts/VestingLockup.sol (v3): 275~291, 365~388	● Resolved

Description

It is important to emit events for sensitive actions, particularly those that can be executed by centralized roles or administrators. This ensures transparency and enables tracking of critical changes, which is essential for security and trust in the system. Missing event logs can indeed result in a lack of visibility and potential information loss.

Recommendation

It is recommended to emit events in sensitive functions that are controlled by centralization roles.

Alleviation

[Unseen team, 7/22/2024]: The team heeded the advice and resolved the issue in commit: https://github.com/Rektstudios/unseen-audits/commit/953e5c922099e39a485713e317218116b795f3a5

CON-05 | FUNCTIONS WITH `_` AS NAME PREFIX ARE NOT `private` OR `internal`

Category	Severity	Location	Status
Coding Style	● Informational	contracts/extensions/ERC4907A.sol (v3): 215~219; contracts/shim/Shim.sol (v3): 13~16	● Resolved

Description

Functions with names starting with `_` should be declared as `private` / `internal` .

Recommendation

Consider changing function visibility to private or removing `_` from the start of the function name.

Alleviation

[Unseen team, 7/22/2024]: The team heeded the advice and resolved the issue in commit: https://github.com/Rektstudios/unseen-audits/commit/953e5c922099e39a485713e317218116b795f3a5

MAK-01 | TYPOS IN COMMENTS

Category	Severity	Location	Status
Coding Style	● Informational	contracts/marketplace/exchange/ExchangeCore.sol (v3): 140; contracts/marketplace/registry/AuthenticatedProxy.sol (v3): 25	● Resolved

Description

There are several typos in the contracts, please see the locations above, where the words with typos are listed:

- targer
- implemenation

Recommendation

We recommend correcting all of the typos in the contracts to provide better readability for open-source purposes.

Alleviation

[Unseen team, 7/22/2024]: The team heeded the advice and resolved the issue in commit: https://github.com/Rektstudios/unseen-audits/commit/953e5c922099e39a485713e317218116b795f3a5

SMR-01 | MISSING ERROR MESSAGES

Category	Severity	Location	Status
Coding Style	● Informational	contracts/marketplace/staticMarket/StaticMarket.sol (v3): 80, 81~84, 135, 136~139, 238, 239, 271, 272, 312	● Resolved

Description

The **require** can be used to check for conditions and throw an exception if the condition is not met. It is better to provide a string message containing details about the error that will be passed back to the caller.

Recommendation

We advise adding error messages to the linked **require** statements.

Alleviation

[Unseen team, 7/22/2024]: The team heeded the advice and resolved the issue in commit: https://github.com/Rektstudios/unseen-audits/commit/953e5c922099e39a485713e317218116b795f3a5

THG-01 | COMMINGLING OF PROXY AND LOGIC CONTRACTS

Category	Severity	Location	Status
Coding Style	● Informational	contracts/thegenerates/TheGenerates.sol (v3): 27; contracts/thegenerates/extensions/ContractMetadata.sol (v3): 24; contracts/thegenerates/extensions/TGenContract.sol (v3): 40	● Acknowledged

Description

In most commonly used proxy patterns (e.g. transparent proxy, UUPS, and diamond proxy), there is a clear separation of proxy and logic, in that all contract logic (with the possible exception of contract upgrade) are implemented in the logic contract, and the proxy contract delegate function calls via the `fallback()` function.

However, for `TheGenerates` , the usage of proxy and logic are commingled. `TheGenerates` inherits `TGenContract` which contains a `fallback()` function that delegate calls to the `TheGeneratesConfigurer` contract. At the same time, all contracts in the inheritance chain, including `TheGenerates` , `TGenContract` , `ContractMetadata` , `MarketsPreapproved` , `TokenTransferValidator` etc. all contain function logic as well. Furthermore, within the `fallback()` function itself, some logic are delegated to the external `TheGeneratesConfigurer` contract, and some are redirected to functions implemented in the contract itself.

Recommendation

Consider applying clear separation of proxy and logic in the code base for better modular design and maintainability.

Alleviation

[Unseen team, 7/26/2024]: Reasoning Behind the Current Setup:

- Size Limitations: The size of the smart contracts necessitated the separation of some logic into the TheGeneratesConfigurer contract to ensure we remain within the permissible contract size limits.
- Separation of Concerns: By offloading specific validation and configuration logic to TheGeneratesConfigurer, we aimed to modularize the contract to enhance maintainability and clarity.

VLR-01 | EIP4906 NOT STRICTLY FOLLOWED

Category	Severity	Location	Status
Coding Issue	● Informational	contracts/vesting/abstracts/VestingLockup.sol (v3): 20	● Resolved

Description

According to EIP 4906 (https://eips.ethereum.org/EIPS/eip-4906), The `supportsInterface` method MUST return `true` when called with `0x49064906` . The `VestingLockup` contract inherits `IERC4906` but the `supportsInterface` method is not updated to conform to the EIP 4906 standard.

Recommendation

Consider updating the `supportsInterface` function in `VestingLockup` to conform to the EIP 4906 standard. For example, the EIP provided the following reference implementation:

```
    function supportsInterface(bytes4 interfaceId) public view virtual
override(IERC165, ERC721) returns (bool) {
        return interfaceId == bytes4(0x49064906) ||
super.supportsInterface(interfaceId);
    }
```

Alleviation

[Unseen team, 7/22/2024]: The team heeded the advice and resolved the issue in commit: https://github.com/Rektstudios/unseen-audits/commit/953e5c922099e39a485713e317218116b795f3a5

OPTIMIZATIONS | UNSEEN - AUDIT

ID	Title	Category	Severity	Status
EXA-01	Variables That Could Be Declared As `constant`	Gas Optimization	Optimization	● Resolved

EXA-01 | VARIABLES THAT COULD BE DECLARED AS `constant`

Category	Severity	Location	Status
Gas Optimization	● Optimization	contracts/marketplace/exchange/ExchangeCore.sol (v3): 28	● Resolved

Description

The linked variables could be declared as constant since these state variables are never modified. Compared to regular state variables, the gas costs of constant variables are much lower.

Recommendation

It is recommended to declare these variables as `constant` .

Alleviation

[Unseen team, 7/22/2024]: The team heeded the advice and resolved the issue in commit: https://github.com/Rektstudios/unseen-audits/commit/953e5c922099e39a485713e317218116b795f3a5

FORMAL VERIFICATION | UNSEEN - 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 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 Pausable ERC-20 Compliance

We verified properties of the public interface of those token contracts that implement the pausable 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-allowance-succeed-always	`allowance` Always Succeeds
erc20-allowance-correct-value	`allowance` Returns Correct Value
erc20-balanceof-correct-value	`balanceOf` Returns the Correct Value
erc20-balanceof-succeed-always	`balanceOf` Always Succeeds
erc20-totalsupply-correct-value	`totalSupply` Returns the Value of the Corresponding State Variable
erc20-approve-succeed-normal	`approve` Succeeds for Valid Inputs
erc20-approve-correct-amount	`approve` Updates the Approval Mapping Correctly
erc20-totalsupply-succeed-always	`totalSupply` Always Succeeds
erc20-allowance-change-state	`allowance` Does Not Change the Contract's State
erc20-balanceof-change-state	`balanceOf` Does Not Change the Contract's State

Property Name	Title
erc20-totalsupply-change-state	`totalSupply` Does Not Change the Contract's State
erc20-approve-false	If `approve` Returns `false`, the Contract's State Is Unchanged
erc20-approve-never-return-false	`approve` Never Returns `false`
erc20-approve-revert-zero	`approve` Prevents Approvals For the Zero Address

Verification of Standard ERC-721A Properties

We verified *partial* properties of the public interfaces of those token contracts that implement the ERC721A interface. This involves:

- functions `totalSupply` `balanceOf` `ownerOf` `getApproved` and `isApprovedForAll`, which collectively provide comprehensive information about the token,
- function `safeTransferFrom` that safely transfers token from `from` to `to`.

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

Property Name	Title
erc165-supportsinterface-correct-false	`supportsInterface` Returns `False` for Id 0xffffffff
erc721a-supportsinterface-metadata	`supportsInterface` Signals that ERC721Metadata is Implemented
erc165-supportsinterface-correct-erc165	`supportsInterface` Signals Support for ERC165
erc721a-balanceof-no-change-state	`balanceOf` Does Not Change the Contract's State
erc721a-ownerof-no-change-state	`ownerOf` Does Not Change the Contract's State
erc165-supportsinterface-no-change-state	`supportsInterface` Does Not Change the Contract's State
erc721a-totalsupply-change-state	ERC721A `totalSupply` Change State
erc721a-getapproved-change-state	`getApproved` Does Not Change the Contract's State
erc721a-balanceof-revert	`balanceOf` Fails on the Zero Address
erc721a-supportsinterface-correct-erc721	`supportsInterface` Signals Support for `ERC721`

Verification Results

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

Detailed Results For Contract UnseenToken (contracts/uncn/uncn.sol) In Commit b0e0abf927c12a15cc878213e26eec6d94734570

Verification of Pausable ERC-20 Compliance

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 `balanceOf`

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

Detailed Results for Function `totalSupply`

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

Detailed Results for Function `approve`

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

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 TGenContract (contracts/thegenerates/extensions/TGenContract.sol) In Commit b0e0abf927c12a15cc878213e26eec6d94734570

Verification of Standard ERC-721A Properties

Detailed Results for Function `supportsInterface`

Property Name	Final Result	Remarks
erc165-supportsinterface-correct-false	○ Inapplicable	The property does not apply to the contract
erc721a-supportsinterface-metadata	● True	
erc165-supportsinterface-correct-erc165	● True	
erc165-supportsinterface-no-change-state	● True	
erc721a-supportsinterface-correct-erc721	● True	

Detailed Results for Function `balanceOf`

Property Name	Final Result	Remarks
erc721a-balanceof-no-change-state	● True	
erc721a-balanceof-revert	○ Inconclusive	

Detailed Results for Function `ownerOf`

Property Name	Final Result	Remarks
erc721a-ownerof-no-change-state	● True	

Detailed Results for Function `totalSupply`

Property Name	Final Result	Remarks
erc721a-totalsupply-change-state	● True	

Detailed Results for Function `getApproved`

Property Name	Final Result	Remarks
erc721a-getapproved-change-state	● True	

Detailed Results For Contract TheGenerates (contracts/thegenerates/TheGenerates.sol) In Commit b0e0abf927c12a15cc878213e26eec6d94734570

Verification of Standard ERC-721A Properties

Detailed Results for Function `supportsInterface`

Property Name	Final Result	Remarks
erc165-supportsinterface-no-change-state	● True	
erc721a-supportsinterface-metadata	● True	
erc721a-supportsinterface-correct-erc721	● True	
erc165-supportsinterface-correct-false	○ Inapplicable	The property does not apply to the contract
erc165-supportsinterface-correct-erc165	● True	

Detailed Results for Function `balanceOf`

Property Name	Final Result	Remarks
erc721a-balanceof-revert	○ Inconclusive	
erc721a-balanceof-no-change-state	● True	

Detailed Results for Function `totalSupply`

Property Name	Final Result	Remarks
erc721a-totalsupply-change-state	● True	

Detailed Results for Function `ownerOf`

Property Name	Final Result	Remarks
erc721a-ownerof-no-change-state	● True	

Detailed Results for Function `getApproved`

Property Name	Final Result	Remarks
erc721a-getapproved-change-state	● True	

Detailed Results For Contract ContractMetadata (contracts/thegenerates/extensions/ContractMetadata.sol) In Commit

b0e0abf927c12a15cc878213e26eec6d94734570

Verification of Standard ERC-721A Properties

Detailed Results for Function `supportsInterface`

Property Name	Final Result	Remarks
erc721a-supportsinterface-metadata	● True	
erc165-supportsinterface-correct-false	● Inapplicable	The property does not apply to the contract
erc721a-supportsinterface-correct-erc721	● True	
erc165-supportsinterface-correct-erc165	● True	
erc165-supportsinterface-no-change-state	● True	

Detailed Results for Function `balanceOf`

Property Name	Final Result	Remarks
erc721a-balanceof-no-change-state	● True	
erc721a-balanceof-revert	● Inconclusive	

Detailed Results for Function `ownerOf`

Property Name	Final Result	Remarks
erc721a-ownerof-no-change-state	● True	

Detailed Results for Function `getApproved`

Property Name	Final Result	Remarks
erc721a-getapproved-change-state	● True	

Detailed Results for Function `totalSupply`

Property Name	Final Result	Remarks
erc721a-totalsupply-change-state	● True	

APPENDIX | UNSEEN - AUDIT

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.
Coding Issue	Coding Issue findings are about general code quality including, but not limited to, coding mistakes, compile errors, and performance issues.
Access Control	Access Control findings are about security vulnerabilities that make protected assets unsafe.
Inconsistency	Inconsistency findings refer to different parts of code that are not consistent or code that does not behave according to its specification.
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.

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 ERC-20-Pausable Properties

Properties related to function `allowance`

erc20-allowance-change-state

Function `allowance` must not change any of the contract's state variables.

Specification:

```
assignable \nothing;
```

erc20-allowance-correct-value

Invocations of `allowance(owner, spender)` must return the allowance that address `spender` has over tokens held by address `owner`.

Specification:

```
ensures \result == allowance(\old(owner), \old(spender));
```

erc20-allowance-succeed-always

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

Specification:

```
reverts_only_when false;
```

Properties related to function `balanceOf`

erc20-balanceof-change-state

Function `balanceOf` must not change any of the contract's state variables.

Specification:

```
assignable \nothing;
```

erc20-balanceof-correct-value

Invocations of `balanceOf(owner)` must return the value that is held in the contract's balance mapping for address `owner` .

Specification:

```
ensures \result == balanceOf(\old(account));
```

erc20-balanceof-succeed-always

Function `balanceOf` must always succeed if it does not run out of gas.

Specification:

```
reverts_only_when false;
```

Properties related to function `totalSupply`

erc20-totalsupply-change-state

The `totalSupply` function in contract UnseenToken must not change any state variables.

Specification:

```
assignable \nothing;
```

erc20-totalsupply-correct-value

The `totalSupply` function must return the value that is held in the corresponding state variable of contract UnseenToken.

Specification:

```
ensures \result == totalSupply();
```

erc20-totalsupply-succeed-always

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

Specification:

```
reverts_only_when false;
```

Properties related to function `approve`

erc20-approve-correct-amount

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:

```
requires spender != address(0);
ensures \result ==> allowance(msg.sender, \old(spender)) == \old(amount);
```

erc20-approve-false

If function `approve` returns `false` to signal a failure, it must undo all state changes that it incurred before returning to the caller.

Specification:

```
ensures !\result ==> \assigned (\nothing);
```

erc20-approve-never-return-false

The function `approve` must never returns `false`.

Specification:

```
ensures \result;
```

erc20-approve-revert-zero

All calls of the form `approve(spender, amount)` must fail if the address in `spender` is the zero address.

Specification:

```
ensures \old(spender) == address(0) ==> !\result;
```

erc20-approve-succeed-normal

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:

```
requires spender != address(0);
ensures \result;
reverts_only_when false;
```

Description of the Analyzed ERC-721A-Standard Properties

Properties related to function `supportsInterface`

erc165-supportsinterface-correct-erc165

Invocations of `supportsInterface(id)` must signal that the interface `ERC165` is implemented.

Specification:

```
requires interfaceId == 0x01ffc9a7;
ensures \result;
```

erc165-supportsinterface-correct-false

Invocations of `supportsInterface(id)` with `id` 0xffffffff must return `false`.

Specification:

```
requires interfaceId == 0xffffffff;
ensures !\result;
```

erc165-supportsinterface-no-change-state

Function `supportsInterface` must not change any of the contract's state variables.

Specification:

```
assignable \nothing;
```

erc721a-supportsinterface-correct-erc721

Invocations of `supportsInterface(id)` must signal that the interface `ERC721` is implemented.

Specification:

```
requires interfaceId == 0x80ac58cd;
ensures esult;
```

erc721a-supportsinterface-metadata

A call of `supportsInterface(interfaceId)` with the interface id of ERC721Metadata must return true.

Specification:

```
requires interfaceId == 0x5b5e139f;
ensures \result;
```

Properties related to function `balanceOf`

erc721a-balanceof-no-change-state

Function `balanceOf` must not change any of the contract's state variables.

Specification:

```
assignable \nothing;
```

erc721a-balanceof-revert

Invocations of `balanceOf(owner)` must fail if the address `owner` is the zero address.

Specification:

```
reverts_when owner == address(0);
```

Properties related to function `ownerOf`

erc721a-ownerof-no-change-state

Function `ownerOf` must not change any of the contract's state variables.

Specification:

```
assignable \nothing;
```

Properties related to function `totalSupply`

erc721a-totalsupply-change-state

The `totalSupply` function in contract TGenContract must not change any state variables.

Specification:

```
assignable \nothing;
```

erc721a-totalsupply-change-state

The `totalSupply` function in contract ContractMetadata must not change any state variables.

Specification:

```
assignable \nothing;
```

erc721a-totalsupply-change-state

The `totalSupply` function in contract TheGenerates must not change any state variables.

Specification:

```
assignable \nothing;
```

Properties related to function `getApproved`

erc721a-getapproved-change-state

Function `getApproved` must not change any of the contract's state variables.

Specification:

```
assignable \nothing;
```

DISCLAIMER | CERTIK

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to you ("Customer" or the "Company") in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes, nor may copies be delivered to any other person other than the Company, without CertiK's prior written consent in each instance.

This report is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts CertiK to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. CertiK's position is that each company and individual are responsible for their own due diligence and continuous security. CertiK's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

The assessment services provided by CertiK is subject to dependencies and under continuing development. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.

ALL SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF ARE PROVIDED "AS IS" AND "AS AVAILABLE" AND WITH ALL FAULTS AND DEFECTS WITHOUT WARRANTY OF ANY KIND. TO THE MAXIMUM EXTENT PERMITTED UNDER APPLICABLE LAW, CERTIK HEREBY DISCLAIMS ALL WARRANTIES, WHETHER EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS. WITHOUT LIMITING THE FOREGOING, CERTIK SPECIFICALLY DISCLAIMS ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT, AND ALL WARRANTIES ARISING FROM COURSE OF DEALING, USAGE, OR TRADE PRACTICE. WITHOUT LIMITING THE FOREGOING, CERTIK MAKES NO WARRANTY OF ANY KIND THAT THE SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF, WILL MEET CUSTOMER'S OR ANY OTHER PERSON'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULT, BE COMPATIBLE OR WORK WITH ANY SOFTWARE, SYSTEM, OR OTHER SERVICES, OR BE SECURE, ACCURATE, COMPLETE, FREE OF HARMFUL CODE, OR ERROR-FREE. WITHOUT LIMITATION TO THE FOREGOING, CERTIK PROVIDES NO WARRANTY OR

UNDERTAKING, AND MAKES NO REPRESENTATION OF ANY KIND THAT THE SERVICE WILL MEET CUSTOMER'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULTS, BE COMPATIBLE OR WORK WITH ANY OTHER SOFTWARE, APPLICATIONS, SYSTEMS OR SERVICES, OPERATE WITHOUT INTERRUPTION, MEET ANY PERFORMANCE OR RELIABILITY STANDARDS OR BE ERROR FREE OR THAT ANY ERRORS OR DEFECTS CAN OR WILL BE CORRECTED.

WITHOUT LIMITING THE FOREGOING, NEITHER CERTIK NOR ANY OF CERTIK'S AGENTS MAKES ANY REPRESENTATION OR WARRANTY OF ANY KIND, EXPRESS OR IMPLIED AS TO THE ACCURACY, RELIABILITY, OR CURRENCY OF ANY INFORMATION OR CONTENT PROVIDED THROUGH THE SERVICE. CERTIK WILL ASSUME NO LIABILITY OR RESPONSIBILITY FOR (I) ANY ERRORS, MISTAKES, OR INACCURACIES OF CONTENT AND MATERIALS OR FOR ANY LOSS OR DAMAGE OF ANY KIND INCURRED AS A RESULT OF THE USE OF ANY CONTENT, OR (II) ANY PERSONAL INJURY OR PROPERTY DAMAGE, OF ANY NATURE WHATSOEVER, RESULTING FROM CUSTOMER'S ACCESS TO OR USE OF THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS.

ALL THIRD-PARTY MATERIALS ARE PROVIDED "AS IS" AND ANY REPRESENTATION OR WARRANTY OF OR CONCERNING ANY THIRD-PARTY MATERIALS IS STRICTLY BETWEEN CUSTOMER AND THE THIRD-PARTY OWNER OR DISTRIBUTOR OF THE THIRD-PARTY MATERIALS.

THE SERVICES, ASSESSMENT REPORT, AND ANY OTHER MATERIALS HEREUNDER ARE SOLELY PROVIDED TO CUSTOMER AND MAY NOT BE RELIED ON BY ANY OTHER PERSON OR FOR ANY PURPOSE NOT SPECIFICALLY IDENTIFIED IN THIS AGREEMENT, NOR MAY COPIES BE DELIVERED TO, ANY OTHER PERSON WITHOUT CERTIK'S PRIOR WRITTEN CONSENT IN EACH INSTANCE.

NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS.

THE REPRESENTATIONS AND WARRANTIES OF CERTIK CONTAINED IN THIS AGREEMENT ARE SOLELY FOR THE BENEFIT OF CUSTOMER. ACCORDINGLY, NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH REPRESENTATIONS AND WARRANTIES AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH REPRESENTATIONS OR WARRANTIES OR ANY MATTER SUBJECT TO OR RESULTING IN INDEMNIFICATION UNDER THIS AGREEMENT OR OTHERWISE.

FOR AVOIDANCE OF DOUBT, THE SERVICES, INCLUDING ANY ASSOCIATED ASSESSMENT REPORTS OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

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.