GA GUARDIAN

Yuga Labs

NFT Shadows

Security Assessment

January 17th, 2025

Summary

Audit Firm Guardian

Prepared By Owen Thurm, Daniel Gelfand, Kamensec, 0x3b, Arnie Jimenez

Client Firm Yuga Labs

Final Report Date January 17, 2025

Audit Summary

Yuga Labs engaged Guardian to review the security of their cross-chain NFT mirroring protocol. From the 30th of December to the 6th of January, a team of 5 auditors reviewed the source code in scope. All findings have been recorded in the following report.

Issues Detected Throughout the engagement 3 High/Critical issues were uncovered and promptly remediated by the Yuga Labs team. Several issues impacted the fundamental behavior of the protocol, following their remediation Guardian believes the protocol to uphold the functionality described for the NFT Mirror.

For a detailed understanding of risk severity, source code vulnerability, and potential attack vectors, refer to the complete audit report below.

✅ Verify the authenticity of this report on Guardian's GitHub: https://github.com/guardianaudits

📊 Code coverage & PoC test suite: https://github.com/GuardianAudits/yuga-2/

Table of Contents

Project Information

Project Overview 4

Audit Scope & Methodology 5

Smart Contract Risk Assessment

Findings & Resolutions 7

Addendum

Disclaimer 41

About Guardian Audits 42

Project Overview

Project Summary

Project Name	Yuga Labs
Language	Solidity
Codebase	https://github.com/yuga-labs/NFTMIrror
Commit(s)	Commit (initial): 763a1a9ddbd340f24babba19cc4c7c73c39aa4f4 Commit (final): 92953d35d8fd8abe7144d75887233cd1ae3179da

Audit Summary

Delivery Date	January 17, 2025
Audit Methodology	Static Analysis, Manual Review, Test Suite, Contract Fuzzing

Vulnerability Summary

Vulnerability Level	Total	Pending	Declined	Acknowledged	Partially Resolved	Resolved
Critical	1	0	0	0	0	1
High	2	0	0	0	0	2
Medium	12	0	0	0	0	12
Low	16	0	0	3	0	13

Audit Scope & Methodology

Vulnerability Classifications

Severity	Impact: *High*	Impact: *Medium*	Impact: *Low*
Likelihood: *High*	🔴 Critical	🟠 High	🟡 Medium
Likelihood: *Medium*	🟠 High	🟡 Medium	🟢 Low
Likelihood: *Low*	🟡 Medium	🟢 Low	🟢 Low

Impact

High Significant loss of assets in the protocol, significant harm to a group of users, or a core functionality of the protocol is disrupted.

Medium A small amount of funds can be lost or ancillary functionality of the protocol is affected. The user or protocol may experience reduced or delayed receipt of intended funds.

Low Can lead to any unexpected behavior with some of the protocol's functionalities that is notable but does not meet the criteria for a higher severity.

Likelihood

High The attack is possible with reasonable assumptions that mimic on-chain conditions, and the cost of the attack is relatively low compared to the amount gained or the disruption to the protocol.

Medium An attack vector that is only possible in uncommon cases or requires a large amount of capital to exercise relative to the amount gained or the disruption to the protocol.

Low Unlikely to ever occur in production.

Audit Scope & Methodology

Methodology

Guardian is the ultimate standard for Smart Contract security. An engagement with Guardian entails the following:

- Two competing teams of Guardian security researchers performing an independent review.
- A dedicated fuzzing engineer to construct a comprehensive stateful fuzzing suite for the project.
- An engagement lead security researcher coordinating the 2 teams, performing their own analysis, relaying findings to the client, and orchestrating the testing/verification efforts.

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. Comprehensive written tests as a part of a code coverage testing suite.
- Contract fuzzing for increased attack resilience.

Findings & Resolutions

ID	Title	Category	Severity	Status
C-01	Messaging Channel Blocked By Locked Chain Reads	DoS	Critical	Resolved
H-01	Ownership Cannot Be Synced To Base Chain	DoS	High	Resolved
H-02	Reorgs Invalidate The Locked Invariant	Reorg	High	Resolved
M-01	ShadowFactory Cannot Deploy NFTs	DoS	Medium	Resolved
M-02	There Is No Function For triggerMetadataRead	DoS	Medium	Resolved
M-03	Release Stuck On Zero Address Beneficiary	DoS	Medium	Resolved
M-04	Insufficient Gas	DoS	Medium	Resolved
M-05	Enforced Options Not Applied On releaseOnEid	Configuration	Medium	Resolved
M-06	Different Gas Limits May Lead To DOS	DoS	Medium	Resolved
M-07	Callback Gas Is Never Specified	DoS	Medium	Resolved
M-08	releaseOnEid In NFTShadow May Lead To DOS	DoS	Medium	Resolved
M-09	Same Address Controlled By Multiple Parties	Logical Error	Medium	Resolved
M-10	Read Response Censoring	DoS	Medium	Resolved

Findings & Resolutions

ID	Title	Category	Severity	Status
M-11	Insufficient Return Data Length Options	Gaming	Medium	Resolved
M-12	Burned Tokens Minted Arbitrarily	Access Control	Medium	Resolved
L-01	Lacking Read Channel Validation	Validation	Low	Resolved
L-02	Caller Is Always The Refund Receiver	Compatibility	Low	Resolved
L-03	Multiple Delegation Rights Risk	Gaming	Low	Acknowledged
L-04	Collection Misconfiguration Leads To DoS	Validation	Low	Resolved
L-05	safeCall Does Not Validate Contract Existence	Warning	Low	Acknowledged
L-06	baseCollectionAddress Clashing	Documentation	Low	Acknowledged
L-07	Unnecessary Reason Variable	Optimization	Low	Resolved
L-08	Empty Registry Call Revert	DoS	Low	Resolved
L-09	Missing ERC 5192 Support	Composability	Low	Resolved
L-10	Smart Contract Wallets Warning	Warning	Low	Resolved
L-11	Locked ShadowNFT Ownership Warning	Warning	Low	Resolved

Findings & Resolutions

ID	Title	Category	Severity	Status
L-12	Lacking Read Response Staleness Checks	Validation	Low	Resolved
L-13	Forks Will Enable 2 Shadow NFTs To Be Unlocked	Validation	Low	Resolved
L-14	CryptoPunks Not Supported	Logical Error	Low	Resolved
L-15	Incorrect Initialize Comment	Logical Error	Low	Resolved
L-16	Missing Validation On Triggers	Validation	Low	Resolved

C-01 | Messaging Channel Blocked By Locked Chain Reads

Category	Severity	Location	Status
DoS	● Critical	Beacon.sol: 324, 331	Resolved

Description

The `unlockedExclusiveOwnerByRights` function reverts for chains that have the NFT locked. However this will cause the read messaging channel for the OApp on that chain to be blocked, preventing all subsequent reads on the chain.

This occurs because the DVNs cannot process the read request, producing an error of `UnresolvableCommand` which prevents the DVN from verifying their response for the nonce.

When DVNs have not verified the response for a nonce, validation in the `EndpointV2` contract prevents any subsequent messages for the OApp from being processed in `lzReceive`:

https://github.com/LayerZero-Labs/LayerZero-v2/blob/7da76840e41dc593d3c2007ce35b911b1d816b4b/packages/layerzero-v2/evm/protocol/contracts/MessagingChannel.sol#L138

This produces a block of read messages on the chain which the invalid read request was triggered from. The delegate of the OApp can call the skip function on the endpoint in order to skip the affected nonce which cannot be verified and resume `lzReceive` processing.

However with the commonality of reverts in the target `unlockedExclusiveOwnerByRights` the delegate will not successfully be able to unstuck the read channels, especially on chains where gas costs are high like Ethereum. As a result the ownership syncing feature is completely DoS'd by either malicious or normal use.

Recommendation

Instead of reverting in the `unlockedExclusiveOwnerByRights` function when the token is locked on the target chain, consider returning a boolean value in addition to the exclusive owner address which indicates whether the chain is locked or not.

Resolution

Yuga Labs Team: The issue was resolved in commit 2f73109.

H-01 | Ownership Cannot Be Synced To Base Chain

Category	Severity	Location	Status
DoS	High	Beacon.sol: 595	Resolved

Description

In the `_updateOwnership` function the `executeCallback` function is always invoked on the `_shadow` address, even when the collection is native to the current chain.

In the case that the collection is native to the current chain the `_shadow` contract address is the address of the actual NFT collection, which does not implement the `executeCallback` function.

As a result receptions of `lzRead` requests on the base collection chain will always revert, thus preventing the syncing of ownership to the base chain.

Recommendation

Only perform the `executeCallback` invocation on the `_shadow` address if `isNative` is `false`.

Resolution

Yuga Labs Team: The issue was resolved in commit cc3f87a.

H-02 | Reorgs Invalidate The Locked Invariant

Category	Severity	Location	Status
Reorg	High	Beacon.sol 229	Resolved

Description

function `releaseOnEid()` transfers token from origin chain owner to the `Beacon.sol` contract, then delegates the rights to he owner, completing the `lzSend() -> lzReceive()` request on the destination chain. For completed request the `NFTShadow` is minted and is now owned by the `beneficiary` on the destination chain. In the case of a reorg on the base chain, the NFT ownership will revert back to the original owner, become unlocked on the origin chain and the destination chain.
Example:

1. Owner 0xA bridges BAYC ID:11 to polygon

Chain: ETH
NFT: BAYC
Owner: Beacon
Delegate: 0xA
Block: 1000

Chain: Polygon
NFT: BAYC Shadow
Owner: 0xA
Block: 100000

2. 0xA transfers to 0xB on polygon

Chain: ETH
NFT: BAYC
Owner: Beacon
Block: 1000

Chain: Polygon
NFT: BAYC Shadow
Owner: 0xB
Block: 100001

H-02 | Reorgs Invalidate The Locked Invariant

Category	Severity	Location	Status
Reorg	High	Beacon.sol 229	Resolved

Description

3. 0xB bridges back to eth mainnet. Note this will release the token to beneficiary from mainnet Beacon contract.

Chain: ETH
NFT: BAYC
Owner: 0xB
Block: 1001

Chain: Polygon
NFT: BAYC Shadow
Owner: 0xB
Block: 100003

4. A 3 block Reorg occurs on polygon.

Chain: ETH
NFT: BAYC
Owner: 0xB
Block: 1002

Chain: Polygon
NFT: BAYC Shadow
Owner: 0xA
Block: 100000

A now holds an irredeemable NFT on a side chain, whilst 0xB holds the mainnet native token outside of the constraints of the protocol.

Recommendation

Ensure probabilistic finality on origin chain prior to finalization of the `lzSend` for `releaseOnEid` and `triggerOwnershipUpdate` by configuring an appropriate confirmations threshold depending on the chains involved in each bridging pathway.

Resolution

Yuga Labs Team: The issue was resolved in commit e3616ec.

M-01 | ShadowFactory Cannot Deploy NFTs

Category	Severity	Location	Status
DoS	Medium	ShadowFactory.sol: 19	Resolved

Description

`ShadowFactory` is used to create shadow NFTs. It implements `Ownable` and has `onlyOwner` on `deployAndRegister` to prevent anyone but the owner from using this function.

However, the issue is that this owner is never initialized, meaning that after deployment, the owner would be address 0 and no one would be able to call `deployAndRegister`.

Recommendation

Add `_initializeOwner` inside the constructor, or add another function to call it and initialize it. In the second scenario, `_guardInitializeOwner` also needs to be called.

Resolution

Yuga Labs Team: The issue was resolved in commit 33ee610.

M-02 | There Is No Function For triggerMetadataRead

Category	Severity	Location	Status
DoS	Medium	ea4a45b6857036524f682a25fb30762c2254ceca	Resolved

Description

`MetadataReadRenderer` has a function to update the metadata for NFTs on different chains. Where the main function - `triggerMetadataRead` must be called by the `NFTShadow` contract to invoke the change, as it gets the `baseCollectionAddress` from `IBeacon(beacon).shadowToBase(msg.sender)`.

Meaning that `msg.sender` must be `NFTShadow`. However the `NFTShadow` lacks a method which invokes `triggerMetadataRead`.

Recommendation

Add a function inside `NFTShadow` that can invoke `triggerMetadataRead`, make sure it has `onlyOwner`

Resolution

Yuga Labs Team: The issue was resolved in commit ea4a45b.

M-03 | Release Stuck On Zero Address Beneficiary

Category	Severity	Location	Status
DoS	Medium	Beacon.sol: 196	Resolved

Description

In the `releaseOnEid` function there is no validation that the beneficiary address is not the zero address. Therefore a layerzero message can be sent with the beneficiary set as the zero address.

The execution of this message will perpetually fail as all attempts to execute the `executeMessage` function will revert as the Solady ERC721 library does not support mints or transfers to the zero address.

Therefore the Shadow NFTs and base collection NFT will remain locked on all chains with this message stuck in perpetuity.

Recommendation

In the `releaseOnEid` function validate that the beneficiary address is not the zero address.

Resolution

Yuga Labs Team: The issue was resolved in commit 1b80e4a.

M-04 | Insufficient Gas

Category	Severity	Location	Status
DoS	Medium	NFTShadow.sol: 376	Resolved

Description

Inside `releaseOnEid` `getSendOptions` is used to calculate the amount of gas needed for the `_lzReceive` call. However due to rigid nature of the math and vast possible routes that the call could go, it would often result in an revert.

Not only that but the math currently lacks the `GAS_RESERVE`, which is taken before that call is executed, resulting in the total gas allocated to be `30k + (20k * NFT count)`
Lets look into an example call:

Total gas for 5 NFTs gets to be - 80k - 50k + 20k * 5 -> 130k
1. The call goes as usual `executeMessage` -> `_executeMessage`, but then we enter the else for `_updateLockState`
2. `_updateLockState` would in tern perform a couple of checks and call `_clearDelegationsAndReleaseTokens` if we are on the native chain
3. `_clearDelegationsAndReleaseTokens` would execute `transferFrom` for every NFT
4. This is followed by `_clearDelegations` which would call `DELEGATE_REGISTRY.delegateERC721` again for every NFT

Both `transferFrom` and `delegateERC721` access storage, be it warm or cold, costing huge amounts of gas.

Recommendation

Calculate the gas properly and make sure it covers all routes, even the most expensive one

Resolution

Yuga Labs Team: The issue was resolved in commit 3c4b869.

M-05 | Enforced Options Not Applied On releaseOnEid

Category	Severity	Location	Status
Configuration	Medium	Beacon.sol: 182	Resolved

Description

In the `releaseOnEid` function the `options` provided to the `_lzSend` call are not combined with the apps enforced options using the `combineOptions` function. As a result the enforced options for the send message type are not applied.

Recommendation

Apply `combineOptions` to the user provided options in the `releaseOnEid` function.

Resolution

Yuga Labs Team: The issue was resolved in commit 41d4919.

M-06 | Different Gas Limits May Lead To DOS

Category	Severity	Location	Status
DoS	Medium	Beacon.sol: 196	Resolved

Description

Different chains have different gas limits. Due to the fact that a user can send as many tokens as they have using `releaseOnEid`, it may be possible that the tx has enough gas limit on the source chain, but not enough gas Limit to execute on destination chain.

For example ETH Mainnet gas limit is 30M, however Avalanche C-chain is 15M, half the gas limit. This can lead to permanently stuck NFT's when the execution of the `_lzReceive` will exceed the max block gas limit and any retry attempts will fail.

This issue may be more prevalent on chains with an even lower gas limit such as some chains have as low as 8M gas limit. This small of a gas limit can be reached with about 57 tokens being sent.

Recommendation

It is recommended to add a limit of 10-20 on the amount of NFTs that can be bridged in a single tx.

Resolution

Yuga Labs Team: The issue was resolved in commit 152ab98.

M-07 | Callback Gas Is Never Specified

Category	Severity	Location	Status
DoS	Medium	Beacon.sol: 595	Resolved

Description

The `_updateOwnership` function concludes with a callback to the shadow NFT via `_shadow.executeCallback(guid);`. While this callback is included in the gas cost, it adds only a negligible increase of 20k gas:

```
uint128 baseGasCost = withCallback
     _BASE_OWNERSHIP_UPDATE_COST_WITH_CALLBACK // 100k
     _BASE_OWNERSHIP_UPDATE_COST;              // 80k
```

This small gas amount is insufficient to cover any callback, especially a more complex one. Additionally, the callback has no predefined gas limit, which means it could unintentionally consume more gas than expected, causing the function to revert with an OOG error.

Recommendation

Consider passing the gas limit as a parameter and enforcing it during the callback execution.

Resolution

Yuga Labs Team: The issue was resolved in commit a707fbd.

M-08 | releaseOnEid In NFTShadow May Lead To DOS

Category	Severity	Location	Status
DoS	Medium	NFTShadow.sol: 376	Resolved

Description

The releaseOnEid function in `NFTShadow` assumes that we are only releasing to non native chains. This can be observed by the send calculations below

```
function getSendOptions(uint256[] calldata tokenIds)

      public pure returns (bytes memory) {uint128 totalGasRequired =
      _BASE_OWNERSHIP_UPDATE_COST + (_INCREMENTAL_OWNERSHIP_UPDATE_COST *
      uint128(tokenIds.length));
      returnOptionsBuilder.newOptions().addExecutorLzReceiveOption(totalGasRequired, 0);}
```

The function assumes that we are always doing an ownership update but does not account for the fact that if we release to chain that includes the native collection that we will instead use delegations which are more expensive.

This may result in a DOS due to not enough gas being sent to the destination chain to execute the message. However the tx can be retried after failure.

Recommendation

If the Eid that we are releasing on includes that native collection, calculate the send costs to include all the delegations cost.

After talks with dev this was the potential solution: maybe worth removing quotes and options generation from the Shadow and just using the Beacon

Resolution

Yuga Labs Team: The issue was resolved in commit 3c4b869.

M-09 | Same Address Controlled By Multiple Parties

Category	Severity	Location	Status
Logical Error	Medium	Beacon.sol: 216	Resolved

Description

The `releaseOnEid` function transfers the delegation rights to the beneficiary address on the origin chain. This means that the owner of the beneficiary address must be the same on both the origin and destination chain.

If this is not the case one party is able to steal potential delegated airdrops, or the other party is unable to claim.

In the case of multisig accounts across different chains, it is viable that contract addresses are not the same across different changes despite trading across EVM compatible chains.

Recommendation

Delegated rights should not be granted from the Beacon contract immediately upon calling the `releaseOnEid` function.

Instead the global delegation rights should only be granted on the origin chain after waiting for the `releaseOnEid` lifecycle to succeed, and then syncing the ownership state to the origin chain with the `triggerOwnershipUpdate` function.

This way for those users who happen to control the smart contract at the beneficiary address on the destination chain, but not on the origin chain, they can preemptively delegate Shadow token rights on the destination chain to an address that they do control across all chains, preventing the inaccurate delegation on the origin chain.

Resolution

Yuga Labs Team: The issue was resolved in commit 49a83a7.

M-10 | Read Response Censoring

Category	Severity	Location	Status
DoS	Medium	Beacon.sol: 486	Resolved

Description

Because the `_lzReceive` function invokes the `executeMessage` function in a try/catch and specifically leaves behind 50,000 gas units to complete the outer function execution it is possible for a malicious actor to intentionally censor an lzRead response by executing the `lzReceive` function with a minimal amount of gas.

This way the inner `executeMessage` function is not forwarded enough gas to complete execution and fails. As a result the message is added to the `payloadHashes` mapping in the `Beacon` contract. Both normal layerzero messages and layerzero read responses can be censored this way.

But specifically layerzero read requests are not allowed to be retried in the Beacon contract so they are effectively censored.

To pull off this attack a malicious actor can simply observe that the last required DVN has submitted their verification and then call the `commitVerification` and `lzReceive` functions on the receive library and `EndpointV2` contracts accordingly.

Recommendation

Consider adding validation at the beginning of the `_lzReceive` function which requires that the transaction has enough gas to complete the receive logic, ideally this validation will also account for any callback which might occur.

Resolution

Yuga Labs Team: The issue was resolved in commit dd5130c.

M-11 | Insufficient Return Data Length Options

Category	Severity	Location	Status
Gaming	Medium	Global	Resolved

Description

In the `triggerMetadataRead` and `triggerOwnershipUpdate` functions there is no enforcement of the expected `returndata` length for the read request.

The expected `returndata` length must be specified so that the executor can be adequately compensated for executing the `lzReceive` function once the message has been verified.

If users call the `triggerMetadataRead` and `triggerOwnershipUpdate` functions with extra options including the correct return data size then the executor will not execute the `lzReceive` function upon DVN verification and users will have to manually execute the message through the endpoint.

Furthermore the return data size cannot be enforced at the enforced options level since it varies based upon the query being made.

Recommendation

For the Beacon contract Implement the use of the `getReadOptions` function at the Beacon contract level instead of including this in the `NFTShadow` contract so that all read requests are properly configured, not just those coming through the `NFTShadow` contract.

For the `MetaDataReadRenderer` only expose the `triggerMetadataRead` function through the `NFTShadow` collection contract and enforce that sufficient expected return data is specified in the options for each collection's respective `tokenUri` length.

Resolution

Yuga Labs Team: The issue was resolved in commit 9169911.

M-12 | Burned Tokens Minted Arbitrarily

Category	Severity	Location	Status
Access Control	Medium	NFTShadow.sol: 222, ERC721: 594, NFTShadow.sol: 490-492	Resolved

Description

The `NFTShadow` contract makes incorrect assumptions about access control on minting. On an unlocked chain, after a token is burned, it remains unlocked. While unlocked it is possible for any user to to re `mint()`, `updateOwnership()` and steal delegations on other chains.

In the comments "Only the Beacon contract can mint tokens, enforced by `_beforeTokenTransfer`". However the condition is actually as follows:

`if (msg.sender = BEACON_CONTRACT_ADDRESS) if (tokenIsLocked(tokenId)) revert CallerNotBeacon(); // if not beacon and locked} // if you are beacon, or unlocked`

Recommendation

If the `msg.sender` is not the `BEACON_CONTRACT_ADDRESS` and the token is `unlocked`, we will be able to mint to any recipient. After minting, we can `updateOwnership` on any other chain to steal delegation rights.

Resolution

Yuga Labs Team: The issue was resolved in commit cb807e8.

L-01 | Lacking Read Channel Validation

Category	Severity	Location	Status
Validation	Low	Beacon.sol: 121	Resolved

Description

In the Beacon contract constructor the `readChannel` value is assigned without any validation performed on it.

Recommendation

Consider validating that the `readChannel` value is above the `_READ_CHANNEL_EID_THRESHOLD` as expected.

Resolution

Yuga Labs Team: The issue was resolved in commit 188f4ac.

L-02 | Caller Is Always The Refund Receiver

Category	Severity	Location	Status
Compatibility	Low	NFTShadow.sol: 456	Resolved

Description

In the `_triggerOwnershipUpdate` function the `msg.sender` is always assigned as the refund receiver for the beacon `triggerOwnershipUpdate` invocation.

It is likely that smart contracts will interact with the `NFTShadow` contract to make use of their callbacks, however if these contracts do not implement a receive function for the native refund then they will not be compatible.

Recommendation

Consider either adding a `refundReceiver` parameter to the `NFTShadow` functions or documenting this clearly for integrators with the `NFTShadow` contract.

Resolution

Yuga Labs Team: The issue was resolved in commit 231cd07.

L-03 | Multiple Delegation Rights Risk

Category	Severity	Location	Status
Gaming	Low	Beacon.sol	Acknowledged

Description

If a peer is assigned for the base chain with the `setPeer` function then an exploit becomes possible where multiple users obtain global rights for the same token id from the Beacon contract.

Consider the following scenario:
• address A holds bayc 12
• address A gives shadow delegation rights to address B on ETH
• address B is now considered the `exclusiveOwnerByRights`
• address A calls `triggerOwnershipUpdate` on ETH, using ETH as the target EID to read from (this is possible if a peer is assigned for the ETH eid on ETH)
• address B is reported as the owner and is given the global rights from the Beacon contract
• address A calls `releaseOnEid` on the Beacon contract on ETH and receives global rights from the beacon contract as well

Now two addresses have global rights issued from the Beacon contract for token 12, this can be used to steal airdrops or vests that belong to global claiming rights for users which unknowingly receive NFTs which have a second address with allocated global rights from the Beacon.

Recommendation

This exploit is not possible if the peer for the native EID is not assigned on the native chain. This is not a typical configuration that would be expected, however this finding simply serves as an additional warning against such a configuration as it enables the attack described.

Resolution

Yuga Labs Team: Acknowledged.

L-04 | Collection Misconfiguration Leads To DoS

Category	Severity	Location	Status
Validation	Low	Beacon.sol: 138	Resolved

Description

In the `registerCollection` function there is no validation preventing a configuration of the `baseCollectionChainId` to 0.

However this mis-configuration will lead to a complete DoS of the OAPP since the `_collectionIsNative` function validates that the `baseCollectionChainId` is nonzero.

Recommendation

Consider validating that the `baseCollectionChainId` is nonzero in the `registerCollection` function to avoid any potential mis-configurations which may halt in-flight messages.

Resolution

Yuga Labs Team: The issue was resolved in commit 152ab98.

L-05 | safeCall Does Not Validate Contract Existence

Category	Severity	Location	Status
Warning	Low	ExcessivelySafeCall.sol: 40	Acknowledged

Description

The `excessivelySafeCall.excessivelySafeCall()` does not validate the `to.code.length > 0`. As a result EOA recipients and undeployed contract addresses will not execute required function call, but return success on low level `call()` operation.

Recommendation

Check `to.code.length > 0` before call is complete, or alternatively check that the returnDataSize is non zero.

Resolution

Yuga Labs Team: Acknowledged.

L-06 | baseCollectionAddress Clashing

Category	Severity	Location	Status
Documentation	Low	Beacon.sol: 138	Acknowledged

Description

The `registerCollection` function assumes that the `baseCollectionAddress` is unique to every NFT collection, however since this is an omnichain application it is possible for multiple NFT collections to correspond to the same `baseCollectionAddress` on their respective chains.

Recommendation

Be aware that collections which clash like this are not supported by the Beacon system.

Resolution

Yuga Labs Team: Acknowledged.

L-07 | Unnecessary Reason Variable

Category	Severity	Location	Status
Optimization	Low	Beacon.sol: 486	Resolved

Description

In the `_lzReceive` function the `executeMessage` is invoked with `safeCall` which is configured to copy 0 bytes of the return data into memory.

However the `bytes memory reason` value is still used to emit in the revert reason in the `MessageCached` event. This reason will always be empty because 0 bytes are specified as the `_maxCopy` value.

Notice that it is important that the `_maxCopy` value is small or 0 because without limiting the return data size then a malicious callback could revert with excessively large revert data to produce OOG errors in the `_lzReceive` function and enable the retrying of read requests.

Recommendation

Either remove the `reason` string from the `MessageCached` event as it is currently always empty or allocate a small amount of return data bytes to be able to read the revert reason in the `safeCall` invocation.

Resolution

Yuga Labs Team: The issue was resolved in commit cd47b90.

L-08 | Empty Registry Call Revert

Category	Severity	Location	Status
DoS	Low	Beacon.sol: 695	Resolved

Description

In the `_updateDelegations` function the `multicallData` array may be sparsely populated in the case where the `staleOwner = address(0)` and `IERC721(collectionAddress).ownerOf(tokenId) = address(this)` conditions are not satisfied.

This will lead to a revert when using the registry multicall since the `DelegateRegistry` contract does not implement a fallback function to receive empty data calls.

The only case where these conditions will be met is in a race condition where a `releaseOnEid` message is executed before the outdated read result is processed. In this case the correct behavior is to revert since this is an unexpected state.

Recommendation

Consider explicitly reverting when the `staleOwner = address(0)` and `IERC721(collectionAddress).ownerOf(tokenId) = address(this)` conditions are not satisfied.

Resolution

Yuga Labs Team: The issue was resolved in commit fd89120.

L-09 | Missing ERC 5192 Support

Category	Severity	Location	Status
Composability	Low	NFTShadow.sol: 323	Resolved

Description

The `NFTShadow` contract aims to support the `ERC5192` interface standard for soulbound tokens, however this is not reflected in the `supportsInterface` function.

Furthermore, the ERC5192 standard includes a `locked(uint256 tokenId)` function. However the NFTShadow contract instead implements a `tokenIsLocked(uint256 tokenId)` which is not congruent with the ERC5192 standard.

Recommendation

Update the `supportsInterface` function to return true if the queried `interfaceId` is 0xb45a3c0e as indicated by the ERC standard: https://eips.ethereum.org/EIPS/eip-5192. Update the `tokenIsLocked(uint256 tokenId)` function to be named `locked(uint256 tokenId)`.

Resolution

Yuga Labs Team: The issue was resolved in commit 0c87009.

L-10 | Smart Contract Wallets Warning

Category	Severity	Location	Status
Warning	Low	Global	Resolved

Description

Users should be careful to ensure that their highest priority combination of ownership/delegate registry rights as determined by the `ExclusiveDelegateResolver` is never granted to an address that they don't control across all chains supported by the Beacon system.

In this case the alternative owner of the same address on any chain which the user does not control it can trigger an ownership update to sync ownership to themselves on that chain.

The malicious actor cannot steal the NFT as the NFT is locked on this chain, but they will be able to claim airdrops and allocations which are attributed to this NFT on the respective chain.

Recommendation

Be sure to document this for users and integrators.

Resolution

Yuga Labs Team: The issue was resolved in commit d44e445.

L-11 | Locked ShadowNFT Ownership Warning

Category	Severity	Location	Status
Warning	Low	Global	Resolved

Description

Because the ownership of `ShadowNFT`s on locked chains is based upon the result of the resolver's `exclusiveOwnerByRights` function, the tokens on these chains can be unexpectedly transferred away from the current owning address.

For example:
• Shadow NFT 12 is unlocked on APE and owned by 0xA
• Shadow NFT 12 is locked on Arbitrum and owned by 0xA
• 0xA delegates with shadow rights for NFT 12 on APE to 0xB
• 0xB triggers an ownership update for Arbitrum
• 0xB now controls NFT 12 on Arbitrum
• 0xA revokes the delegation to 0xB
• 0xA triggers an ownership update for Arbitrum
• NFT 12 is transferred away from 0xB back to 0xA

This may be unexpected for 0xB since it had not approved any other address.

Recommendation

Potential integrations and users have to be warned that locked `ShadowNFT` ownership cannot be trusted.

Resolution

Yuga Labs Team: The issue was resolved in commit d44e445.

L-12 | Lacking Read Response Staleness Checks

Category	Severity	Location	Status
Validation	Low	Beacon.sol	Resolved

Description

It is possible that an executor or DVN verification is delayed longer than expected, in which case the result of a read operation may be recorded on the triggering chain much later than expected.

In these scenarios if the stale read result is allowed then a potential outdated mis-accounting of the owner assigned can occur.

Recommendation

Consider adding validation such that read requests cannot be executed greater than some amount of time after their initiation to avoid stale results.

Resolution

Yuga Labs Team: The issue was resolved in commit bdffafe.

L-13 | Forks Will Enable 2 Shadow NFTs To Be Unlocked

Category	Severity	Location	Status
Validation	Low	Beacon.sol: 520	Resolved

Description

If a TX fails on the destination chain it gets stored inside `payloadHashes`. However there is no protection against hard forks, and in the case of one, the message would be in the storage inside the contract on both chains (main and fork).

This will allow the user to `retryFailedMessage` on both chains and possibly create 2 shadow NFTs. Given that only 1 shadow or native token must be unlocked at once, this breaks a core invariant which is that only 1 token should be unlocked at one time across all chains.

Recommendation

Include `block.chainId` inside the hash making and it's verification.

Resolution

Yuga Labs Team: The issue was resolved in commit 95b9d6e.

L-14 | CryptoPunks Not Supported

Category	Severity	Location	Status
Logical Error	Low	Beacon.sol	Resolved

Description

In order to `triggerOwnershipUpdate` and mint a shadow the `ownerOf` function must successfully resolve within `unlockedExclusiveOwnerByRights`.

However, `CryptoPunks` do not comply to ERC-721 standard and do not contain many crucial functions for the Shadows to operate properly such as the `ownerOf` function. Consequently, `CryptoPunks` are not currently compatible with Shadows.

Recommendation

If support for this collection is not necessary, do not register the `CryptoPunks` collection and clearly document this. Otherwise, add `CryptoPunks` ownership checks and transfer functions within the Beacon.

Resolution

Yuga Labs Team: The issue was resolved in commit b0b8a5e.

L-15 | Incorrect Initialize Comment

Category	Severity	Location	Status
Logical Error	Low	MetadataReadRenderer.sol	Resolved

Description

The constructor of the `MetadataReadRenderer` incorrectly states that it “Initializes the Beacon contract” which is just a copy of the Beacon’s documentation.

Recommendation

Correct the NatSpec for the constructor of the `MetadataReadRenderer`.

Resolution

Yuga Labs Team: The issue was resolved in commit 3d8c6bc.

L-16 | Missing Validation On Triggers

Category	Severity	Location	Status
Validation	Low	Global	Resolved

Description

Functions `triggerOwnershipUpdate` and `triggerMetadataRead` do not validate that the arrays passed are non-empty. If empty parameters are passed, then the function could early return to prevent building a command for a message that will ultimately fail.

Recommendation

Validate that the array parameters passed are non-empty.

Resolution

Yuga Labs Team: The issue was resolved in commit 0aae7e9.

Disclaimer

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 Guardian 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. Guardian’s position is that each company and individual are responsible for their own due diligence and continuous security. Guardian’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 Guardian 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.

Notice that smart contracts deployed on the blockchain are not resistant from internal/external exploit. Notice that active smart contract owner privileges constitute an elevated impact to any smart contract’s safety and security. Therefore, Guardian does not guarantee the explicit security of the audited smart contract, regardless of the verdict.

About Guardian Audits

Founded in 2022 by DeFi experts, Guardian Audits is a leading audit firm in the DeFi smart contract space. With every audit report, Guardian Audits upholds best-in-class security while achieving our mission to relentlessly secure DeFi.

To learn more, visit https://guardianaudits.com

To view our audit portfolio, visit https://github.com/guardianaudits

To book an audit, message https://t.me/guardianaudits