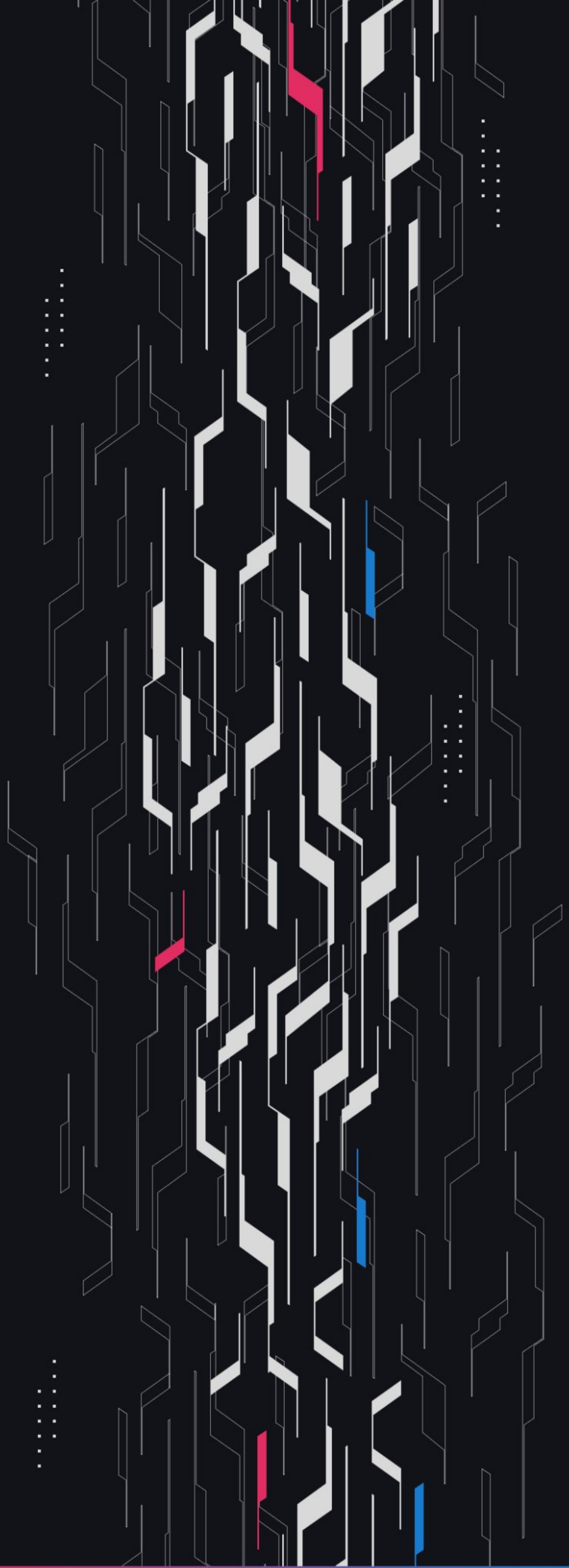
GUARDIAN

Yuga Labs

NFT Shadows 2

Security Assessment

February 5th, 2025



Summary

Audit Firm Guardian

Prepared By Windhustler, Arnie, 0xrajkumar, Unnamed researcher, ilchovski

Client Firm Yuga Labs

Final Report Date February 5, 2025

Audit Summary

Yuga Labs engaged Guardian to review the security of their cross-chain NFT ownership mirroring protocol. From the 22nd of January to the 29th 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 2 High/Critical issues were uncovered and promptly remediated by the Yuga Labs team.

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

Smart Contract Risk Assessment

Addendum

Project Overview

Project Summary

Project Name	Yuga Labs
Language	Solidity
Codebase	https://github.com/yuga-labs/NFT-Shadows
Commit(s)	1906dc8f001b4689f947af8f5da8514b54b9488a

Audit Summary

Delivery Date	February 5, 2025
Audit Methodology	Static Analysis, Manual Review, Test Suite, Contract Fuzzing

Vulnerability Summary

Vulnerability Level	Total	Pending	Declined	Acknowledged	Partially Resolved	Resolved
🔴 Critical	0	0	0	0	0	0
🟠 High	2	0	0	0	0	2
🟡 Medium	4	0	0	3	0	1
🟢 Low	14	0	0	5	0	9

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
H-01	Working With Stale State Can Cause Invalid Delegation Rights	Logical Error	● High	Resolved
H-02	If Beacon Delegates To Address(0) It Will Be Permanent	Logical Error	● High	Resolved
M-01	Working With Stale State Can Cause DoS	Logical Error	● Medium	Acknowledged
M-02	Delegation Is Broken For Punks	Unexpected Behavior	● Medium	Acknowledged
M-03	Deployment Scripts Configuration Issues	Configuration	● Medium	Resolved
M-04	NFT Owner Can Cause exclusiveOwnerByRights() Call To Revert	Unexpected Behavior	● Medium	Acknowledged
L-01	lzReceive Can Be DoSed If The Beacon Contract Is Not Enforcing Executors	DoS	● Low	Acknowledged
L-02	Unused Errors	Optimization	● Low	Resolved
L-03	Same Timestamp Blocks May Lead To Incorrect Data Reads	Logical Error	● Low	Acknowledged
L-04	Missing ERC 4906 Support	Compatibility	● Low	Resolved
L-05	Compute Options Hardcode Confirmations To 0	Unexpected Behavior	● Low	Acknowledged
L-06	User Cannot Increase Gas Limit Of Send()	Unexpected Behavior	● Low	Resolved
L-07	Partial Support For Smart Contract Wallets	Logical Error	● Low	Acknowledged

Findings & Resolutions

ID	Title	Category	Severity	Status
L-08	Send Can Be Used To Execute A Read Request Leading To DoS	DoS	● Low	Resolved
L-09	Read Channel Cannot Be Changed	Suggestion	● Low	Resolved
L-10	User Cannot Specify refundAddress	Suggestion	● Low	Resolved
L-11	Unused Imports	Optimization	● Low	Resolved
L-12	NFTShadow Deviates From ERC721C Standard	Compatibility	● Low	Resolved
L-13	Usage Of Tx.origin Over Msg.sender	Unexpected Behavior	● Low	Acknowledged
L-14	Cannot Update The Config Of A Collection	Suggestion	● Low	Resolved

H-01 | Working With Stale State Can Cause Invalid Delegation Rights

Category	Severity	Location	Status
Logical Error	● High	Beacon.sol: 748	Resolved

Description

When a user wants to update delegate rights on the base chain or mint an NFT on the shadow chain they call lzread. However in lzread we use EVMCallComputeV1 with the current timestamp. Meanwhile in lzmap we rely on a reading state that could potentially be outdated. This is because the state is based on the timestamp set in EVMCallComputeV1 and by the time _executeMessage is called that state may no longer be accurate.

Let's take example: (lzread from Base chain to Shadow chain). Initially, an NFT is locked on the base chain but unlocked and owned by User A on Optimism, with lzRead delegating rights to User A. At timestamp x, someone calls lzRead, and while the owner on Optimism has changed to User B, the base chain still lists delegatedOwners[collectionAddress][tokenId] as User A at timestamp x.

At a later timestamp y, another lzRead call occurs, and now the owner on Optimism is User C, the base chain still lists delegatedOwners[collectionAddress][tokenId] as User A at timestamp y. When processing these updates, delegation rights are transferred from User A to User B for the first call and from User A to User C for the second call, leaving delegatedOwners[collectionAddress][tokenId] set to User C. This creates an issue where both User B and User C have delegation rights, as the delegation for User B cannot be properly revoked.

Recommendation

To mitigate this issue, we should avoid reading staleOwner in lzmap. Instead we should read the previous owner in _updateOwnership.

Resolution

Yuga Labs Team: The issue was resolved in commit 553a6c6.

H-02 | If Beacon Delegates To Address(0) It Will Be Permanent

Category	Severity	Location	Status
Logical Error	● High	Beacon.sol: 947-973	Resolved

Description

The Beacon can delegate to address(0) in case the owner on the remote chain have burned the NFT. In this scenario the NFT remains unlocked and a read request from the base chain will set delegatedOwners[collectionAddress][tokenId] to address(0). This will remove the active delegation of the stale owner and will create a new delegation to address(0) via IDelegateRegistry.delegateERC721.

There is another case where the owner of the NFT on the remote chain delegates to address(0) via IDelegateRegistry.delegateERC721. Upon a read request the Beacon on the base chain will create an active delegation to address(0) that it will not be able to remove ever again for this NFT since we encounter this check in _updateDelegations() when the staleOwner is address(0): if (staleOwner = address(0)) {.

This means that when the NFT gets a new owner on the remote chain from now on, the base chain will have 2 active delegations. One to address(0) and one to the current owner. Having two active delegations can confuse projects that implement logic for distributing rewards based on your current active delegations (and split the rewards), require you to have 1 active delegation or does not handle well delegation to address(0).

Recommendation

Consider including a if (newOwner = address(0)) { check.

Resolution

Yuga Labs Team: The issue was resolved in commit 7c0c359.

M-01 | Working With Stale State Can Cause DoS

Category	Severity	Location	Status
Logical Error	● Medium	Beacon.sol: 751	Acknowledged

Description

When a user wants to update delegate rights on the base chain or mint an NFT on the shadow chain they call lzread. However in lzread we use EVMCallComputeV1 with the current timestamp. Meanwhile in lzmap we rely on a reading state that could potentially be outdated. This is because the state is based on the timestamp set in EVMCallComputeV1, and by the time _executeMessage is called, that state may no longer be accurate.

Example: (lzread from Shadow chain to Shadow chain or Base chain)
For now Shadow chain to Shadow chain example it taken. ApeChain is Shadow chain 1 and Arbitrum is Shadow chain 2. Initially, an NFT is locked on the Arbitrum but unlocked and owned by User A on ApeChain, with lzRead delegating rights to User A. At timestamp x, someone calls lzRead from Arbitrum, and while the owner on ApeChain has changed to User B, the Arbitrum chain lists staleOwner as User A at timestamp x.

At a later timestamp y, another lzRead call occurs from Arbitrum, and now the owner on ApeChain is User C, the Arbitrum chain lists staleOwner as User A at timestamp x. When processing these updates, ownership is transferred from User A to User B for the first call because User A was owner and from User A to User C ownership transfer for the second call will not work because Current owner will be User B. This causes DOS.

Recommendation

To mitigate this issue, we should avoid reading staleOwner in lzmap. Instead, we should read the previous owner in _updateOwnership.

Resolution

Yuga Labs Team: Acknowledged.

M-02 | Delegation Is Broken For Punks

Category	Severity	Location	Status
Unexpected Behavior	● Medium	Beacon.sol: 569	Acknowledged

Description

Try

```
IExclusiveDelegateResolver(EXCLUSIVE_DELEGATE_RESOLVER_ADDRESS).exclusiveOwnerByRights(shadowCollectionAddress, tokenId, SHADOW_TOKEN_RIGHTS
```

The logic attempts to find the owner via the exclusive delegate resolver, However we are using the punk adapter address (shadowCollectionAddress) instead of the punk721 address to find the owner. This will result in delegated wallets not being able to mint a shadow nft.

The functionality of the contract should allow non locked nft to mint shadow NFT's using the delegation functionality, however since we query the wrong address in the delegate resolver, it is not possible to retrieve the delegated user via the beacon contract.

Given that users usually hold punks in cold wallets and use delegations to the hot wallet, this will limit the functionality of shadow nfts as they cannot be minted to delegated wallet addresses of the punk holders.

Recommendation

If the address is the punk adapter, use the punks721 address when querying the delegate resolver.

Resolution

Yuga Labs Team: Acknowledged.

M-03 | Deployment Scripts Configuration Issues

Category	Severity	Location	Status
Configuration	● Medium	./script/.sol	Resolved

Description

1. Incorrect Send Library Configuration:

• The ChainConfigurator contract currently sets ChainConfig.sendLibrary for Ethereum to 0xD231084BfB234C107D3eE2b22F97F3346fDAF705
• This is the sendUln301 library meant for LayerZero EndpointV1

2. Maximum Message Size:

• The ConfigureBeacons contract run script sets ExecutorConfig.maxMessageSize to 1_000_000
• This is significantly higher than the default executor configuration of 10_000 bytes

3. Send Library Configuration:

• The ConfigureBeacons contract run script only sets LayerZero send libraries when they differ from defaults

Recommendation

1. Send Library Update:

• Set ChainConfig.sendLibrary to 0x21F33EcF7F65D61f77e554B4B4380829908cD076 (SendUln302)
• This ensures compatibility with LayerZero EndpointV2

2. Message Size Standardization:
• Set ExecutorConfig.maxMessageSize to the default value of 10_000 bytes

3. Library Configuration:
• Always explicitly set LayerZero libraries
• Do this even when values match defaults

Resolution

Yuga Labs Team: The issue was resolved in commit b333410.

M-04 | NFT Owner Can Cause exclusiveOwnerByRights() Call To Revert

Category	Severity	Location	Status
Unexpected Behavior	● Medium	Beacon.sol: 587	Acknowledged

Description

unlockedExclusiveOwnerByRights() gets called by an LayerZeroV2 DVN off chain to read the new owner of a NFT from a remote chain. Based on certain conditions the function will make a internal call wrapped in a try catch block to the exclusive delegate resolver with the aim to return the address to which the owner of the NFT have delegated and return it as the new owner instead.

IExclusiveDelegateResolver(EXCLUSIVE_DELEGATE_RESOLVER_ADDRESS).exclusiveOwnerByRights() can loop over all outgoingDelegations of a particular owner of an NFT while it loads all in memory, and performs checks in a loop. An owner on a remote chain could create/have thousands of active delegations via IDelegateRegistry with different delegation type than ERC721.

When the off chain read request is processed IExclusiveDelegateResolver(EXCLUSIVE_DELEGATE_RESOLVER_ADDRESS).exclusiveOwnerByRights() inside unlockedExclusiveOwnerByRights() will loop for a very long time and will consume a lot of gas. Off chain calls do not consume gas but they still simulate the consumption of gas in order to revert if such off chain call loops for ever or is too computationally expensive like reaching the block gas limit or some other limit set by the node.

Seems like the owner of the NFT will not be able to make unlockedExclusiveOwnerByRights() to revert, which would be considered a critical issue since it will block future read requests from being executed, because Layer Zero sets a 300M gas limit on such off chain view calls and second because the inner call is wrapped in a try catch block. 63/64 part of the gas would be forwarded to the internal call and the rest 1/64 would be sufficient to finish the rest of the function.

The impact of this would be that the read request will retrieve the actual owner of the NFT instead of the delegated address and that the off chain services will more time to process the request since it might be more computationally intensive. This could harm the owner of the NFT or the systems that integrate and base their protocol logic on the result of the read request.

Recommendation

Users and protocol that integrate with the system should be informed for this possible scenario.

Resolution

Yuga Labs Team: Acknowledged.

L-01 | lzReceive Can Be DoSed If The Beacon Contract Is Not Enforcing Executors

Category	Severity	Location	Status
DoS	● Low	Beacon.sol: 728	Acknowledged

Description

The Beacon contract includes a feature to restrict which addresses can execute cross-chain messages through its lzReceive function. This is implemented via an allowedExecutors mapping and an enforceExecutors toggle.

However, during deployment enforceExecutors is set to false by default. This means any address can call EndpointV2:lzReceive, creating a vulnerability where an attacker could:

1. Monitor for verified messages on each chain
2. Front-run the Executor and call lzReceive with minimal gas (just above SAFE_CALL_BUFFER)
3. Successfully execute the message while preventing the intended execution

This creates a Denial of Service (DoS) vulnerability for all cross-chain communication, as legitimate messages can be intercepted and failed deliberately.

Recommendation

Make sure to always enforce executors and whitelist addresses in the allowedExecutors mapping.

Resolution

Yuga Labs Team: Acknowledged.

L-02 | Unused Errors

Category	Severity	Location	Status
Optimization	● Low	CryptoPunkShadowAdapter.sol: 20	Resolved

Description

There were several unused errors in the codebase. For instance:

- NotOwner
- TokenIsLocked
- CallbackFailed

Recommendation

Consider finding and removing all such instances of unused errors to enhance readability of code.

Resolution

Yuga Labs Team: The issue was resolved in commit afaec9b.

L-03 | Same Timestamp Blocks May Lead To Incorrect Data Reads

Category	Severity	Location	Status
Logical Error	● Low	Beacon.sol: 883	Acknowledged

Description

On Arbitrum, two blocks can share the same timestamp. Suppose we read a timestamp (let's call it x) from Ethereum using LZRead. On Arbitrum, there could be two blocks with the same timestamp x.

If we read from the first block with timestamp x, the owner might be y, but in the next block, the owner could change to z. In this case, we would incorrectly delegate to owner y. It causes a synchronization issue.

Recommendation

Reading from the `current timestamp + 1` instead of the `current timestamp` in `EVMCallRequestV1` would resolve this issue.

Resolution

Yuga Labs Team: Acknowledged.

L-04 | Missing ERC 4906 Support

Category	Severity	Location	Status
Compatibility	● Low	NFTShadow.sol: 342	Resolved

Description

The NFTShadow contract intends to support ERC-4906 but does not reflect this in the supportsInterface function.

When the metadataRenderer is updated using setMetadataRenderer we are not emitting BatchMetadataUpdate. However technically when the metadataRenderer is changed the tokenURI function can return a different tokenURI.

Recommendation

Update the supportsInterface function to return true if the queried interfaceId is 0x49064906 as indicated by the ERC standard: https://eips.ethereum.org/EIPS/eip-4906. We should also emit BatchMetadataUpdate in the setMetadataRenderer function.

Resolution

Yuga Labs Team: The issue was resolved in commit 38d4af9.

L-05 | Compute Options Hardcode Confirmations To 0

Category	Severity	Location	Status
Unexpected Behavior	● Low	Beacon.sol: 894	Acknowledged

Description

Read requests allow for a compute settings that describe the computations which could be done via map and reduce after the original read from the remote chain was retrieved. In these settings we can specify which of these functions should be called, on which chain and timestamp.

The confirmations are set to 0 which means that the off-chain service can compute the result from the remote chain on the source chain on an unconfirmed block.

In the current implementation the request call is in the same block in which the map and reduce computation will be called on off-chain.

This makes it very unlikely that the 0 confirmations will cause a major issue since off-chain services will most likely process read request once the block is confirmed but still confirmations should be changed to a value larger than 0 as specified in the LayerZero documentation.

Recommendation

Consider updating the hardcoded 0 confirmations to a non zero value.

Resolution

Yuga Labs Team: Acknowledged.

L-06 | User Cannot Increase Gas Limit Of Send()

Category	Severity	Location	Status
Unexpected Behavior	● Low	Beacon.sol: 368	Resolved

Description

Beacon.send() calculates how much gas the send message needs via getSendOptions() -> _calculateLzReceiveGasAllocation().

The user is not able to supply more gas for the send message as he can do in read(). The read function needs callbackGasLimit because of the possible callback the caller might want to execute.

In case the calculated in _calculateLzReceiveGasAllocation() gas is not sufficient enough for the send message of a collection the owner will not be able to change the gas settings later on so the user will have no go on the remote chain and retry the message himself.

Recommendation

Consider allowing the caller of send() to be able to increase the gas limit of the call.

Resolution

Yuga Labs Team: The issue was resolved in commit 18a7b40.

L-07 | Partial Support For Smart Contract Wallets

Category	Severity	Location	Status
Logical Error	● Low	Beacon.sol	Acknowledged

Description

When a user bridges an NFT through the Beacon contract to a shadow chain:
1. The NFT is unlocked and minted to a specified beneficiary address on the destination chain
2. Anyone can then trigger a read request to sync the ownership across chains, resulting in:
- The native chain: Beacon delegates the NFT to the beneficiary address
- Other shadow chains: NFT is locked but minted to the beneficiary address

This creates a potential issue for multi-sigs, as users may not control the same multisig address across all chains. Users need to be aware that:
1. Their NFT ownership will automatically sync to the same address on all chains
2. They may lose effective control if they don't control the multisig on every chain

The same ownership syncing behavior applies when bridging back to the native chain:
1. User bridges from shadow chain back to native chain
2. NFT is transferred out from Beacon to the user's address
3. Read requests will sync this new ownership state across all shadow chains

A possible workaround is for users to delegate rights of the unlocked NFT to an address they control across all chains.

Recommendation

The protocol documentation should clearly explain how ownership synchronizes automatically across all chains when bridging NFTs in either direction. Users, especially those using multi-sigs, need to understand they must control the same address across all chains to maintain access.

Resolution

Yuga Labs Team: Acknowledged.

L-08 | Send Can Be Used To Execute A Read Request Leading To DoS

Category	Severity	Location	Status
DoS	● Low	Beacon.sol: 397	Resolved

Description

Beacon::send function is used to send a cross-chain message. It allows the caller to specify the destination chain endpoint without any checks inside the function.
1. During deployment, the readChannel has its peer set to the Beacon contract itself, which means this pathway is enabled.
2. The send function allows messages to any destination chain ID (dstEid) without validation inside the function.
3. Currently, this is safe because:
• The ExecutorFeeLib reverts if read message options contain the LzReceiveOption, which is the case in the getSendOptions function.
• This prevents read messages from being sent through the send function.

However, this safety relies on external contract behavior. If the ExecutorFeeLib, ReadLibrary, or Executor logic changes to accept these options, an attacker could use send to transmit read messages and block the LayerZero pathway since the message encoded is not according to the read request specification.

Recommendation

Add explicit validation in the Beacon::send function:

```
if (dstEid > _READ_CHANNEL_EID_THRESHOLD) revert();
```

This ensures read messages can only be sent through proper channels, regardless of changes in dependent contracts.

Resolution

Yuga Labs Team: The issue was resolved in commit 0414bbc.

L-09 | Read Channel Cannot Be Changed

Category	Severity	Location	Status
Suggestion	● Low	Beacon.sol: 204	Resolved

Description

The read channel currently is set in the constructor and there is no setter function to change it later on. Currently 4294967295 is the active read channel but in case it becomes inactive for some reason the owner will not be able to change it.

Recommendation

Consider adding a setter function for `readChannel`.

Resolution

Yuga Labs Team: The issue was resolved in commit fceef0f.

L-10 | User Cannot Specify refundAddress

Category	Severity	Location	Status
Suggestion	● Low	NFTShadow.sol: 209	Resolved

Description

LayerZero send function allows users to specify a refund address to which the excess msg.value provided for fees could be returned to. NFTShadow.send() its hardcoded to msg.sender.

Recommendation

Consider allowing the user to specify a refund address.

Resolution

Yuga Labs Team: The issue was resolved in commit 6759453.

L-11 | Unused Imports

Category	Severity	Location	Status
Optimization	● Low	NFTShadow.sol: 9-12	Resolved

Description

There are several unused imports in the NFTShadow contract. For instance:

- OptionsBuilder
- IOAppMapper
- IOAppReducer

Recommendation

Consider removing these to enhance the readability of the codebase.

Resolution

Yuga Labs Team: The issue was resolved in commit 9d2ca3d.

L-12 | NFTShadow Deviates From ERC721C Standard

Category	Severity	Location	Status
Compatibility	● Low	NFTShadow.sol: 250	Resolved

Description

The NFTShadow contract deviates from the ERC721C standard when it emits the TransferValidatorSet event instead of emitting the TransferValidatorUpdated event in the setTransferValidator function.

Recommendation

Consider emitting the correct event to maintain compliance with the ERC721C standard.

Resolution

Yuga Labs Team: The issue was resolved in commit 5a6ea17.

L-13 | Usage Of Tx.origin Over Msg.sender

Category	Severity	Location	Status
Unexpected Behavior	● Low	Beacon.sol: 241-260	Acknowledged

Description

Locations where `tx.origin` is used: `tx.origin` is used only in the constructor of the contracts which can be a problem only if the contracts are deployed via a multi-sig or where the EOA triggering the transaction is not the intended owner of these contracts.

With the current deployment setup this will not be an issue. It is considered a good practice to use `msg.sender` over `tx.origin`.

Recommendation

Consider replacing `tx.origin` with `msg.sender`.

Resolution

Yuga Labs Team: Acknowledged.

L-14 | Cannot Update The Config Of A Collection

Category	Severity	Location	Status
Suggestion	● Low	Beacon.sol: 274	Resolved

Description

If a collection is registered with the wrong settings such as wrong base address or insufficient `baseCollectionPerNftOwnershipUpdateCost` then this cannot be undone.

If settings are not correct this can cause reverts in the receive function that need to be retried or can make the collection incompatible with the beacon.

Recommendation

Consider allowing the owner to be able to change these settings.

Resolution

Yuga Labs Team: The issue was resolved in commit afe6f36.

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