Audit Report

cw-hyperlane

v1.0

February 13, 2024

Table of Contents

License

Disclaimer

This audit has been performed by

Oak Security

https://oaksecurity.io/
info@oaksecurity.io

Introduction

Purpose of This Report

Oak Security has been engaged by Abacus Works Inc to perform a security audit of cw-hyperlane.

The objectives of the audit are as follows:

1. Determine the correct functioning of the protocol, in accordance with the project specification.
2. Determine possible vulnerabilities, which could be exploited by an attacker.
3. Determine smart contract bugs, which might lead to unexpected behavior.
4. Analyze whether best practices have been applied during development.
5. Make recommendations to improve code safety and readability.

This report represents a summary of the findings.

As with any code audit, there is a limit to which vulnerabilities can be found, and unexpected execution paths may still be possible. The author of this report does not guarantee complete coverage (see disclaimer).

Codebase Submitted for the Audit

The audit has been performed on the following target:

Repository	https://github.com/many-things/cw-hyperlane
Commit	`ac8cd58c6bc5a831bf4b36d65f8a8d81f4edaf1a`
Scope	The scope of this audit included all contracts and packages within the `cw-hyperlane` repository.
Fixes verified at commit	`659594588d78b1a32d644b903fc6bf321a9b632d` Note that changes to the codebase beyond fixes after the initial audit have not been in scope of our fixes review.

Methodology

The audit has been performed in the following steps:

1. Gaining an understanding of the code base's intended purpose by reading the available documentation.
2. Automated source code and dependency analysis.
3. Manual line-by-line analysis of the source code for security vulnerabilities and use of best practice guidelines, including but not limited to:
    a. Race condition analysis
    b. Under-/overflow issues
    c. Key management vulnerabilities
4. Report preparation

Functionality Overview

Hyperlane is a permissionless interoperability layer built for modular blockchains. cw-hyperlane enables chains supporting CosmWasm to integrate with the Hyperlane protocol. This audit covers the functionality associated with Hyperlane's core, hooks, isms, and warp contracts.

How to Read This Report

This report classifies the issues found into the following severity categories:

Severity	Description
Critical	A serious and exploitable vulnerability that can lead to loss of funds, unrecoverable locked funds, or catastrophic denial of service.
Major	A vulnerability or bug that can affect the correct functioning of the system, lead to incorrect states or denial of service.
Minor	A violation of common best practices or incorrect usage of primitives, which may not currently have a major impact on security, but may do so in the future or introduce inefficiencies.
Informational	Comments and recommendations of design decisions or potential optimizations, that are not relevant to security. Their application may improve aspects, such as user experience or readability, but is not strictly necessary. This category may also include opinionated recommendations that the project team might not share.

The status of an issue can be one of the following: **Pending,** **Acknowledged**, or **Resolved**.

Note that audits are an important step to improving the security of smart contracts and can find many issues. However, auditing complex codebases has its limits and a remaining risk is present (see disclaimer).

Users of the system should exercise caution. In order to help with the evaluation of the remaining risk, we provide a measure of the following key indicators: **code complexity**, **code readability**, **level of documentation**, and **test coverage**. We include a table with these criteria below.

Note that high complexity or low test coverage does not necessarily equate to a higher risk, although certain bugs are more easily detected in unit testing than in a security audit and vice versa.

Code Quality Criteria

The auditor team assesses the codebase's code quality criteria as follows:

Criteria	Status	Comment
Code complexity	Medium	The scope of the audit covered a technical implementation with many contracts and complex interactions.
Code readability and clarity	Medium	The code was generally readable, but did not have sufficient code commenting and had some potentially misleading variable naming.
Level of documentation	Medium	While the Hyperlane protocol has extensive documentation, cw-hyperlane did not have specific documentation to detail the CosmWasm implementation.
Test coverage	Medium	Test coverage is in `75.1%`, assessed with llvm-cov

Summary of Findings

No	Description	Severity	Status
1	Risk of Denial-of-Service of the merkle hook	Major	Acknowledged
2	Lack of `info.funds` transfer to `postDispatch` leads to failure of subsequent operations	Major	Resolved
3	Resource intensive Multisig ISM `verify_message` query	Minor	Acknowledged
4	Warp contracts do not specify hook in `transfer_remote` function	Minor	Acknowledged
5	Multisig ISM may cause out-of-gas errors	Minor	Resolved
6	Static multisig threshold design presents scalability concerns	Minor	Resolved
7	A zero threshold will make it impossible to verify a message	Minor	Resolved
8	Pausable endpoint not exposed	Minor	Resolved
9	Strategic planning is crucial for Mailbox contract upgrades	Minor	Acknowledged
10	Missing address validation and normalization	Minor	Partially Resolved
11	Lack of input validation	Minor	Resolved
12	Aggregate hook does not consider alternate denominations	Minor	Resolved
13	Missing entry point to remove ISM entries	Minor	Resolved
14	Maintainability considerations for Merkle tree	Minor	Partially Resolved
15	Misleading `ContractErrors` and events emitted	Informational	Resolved
16	Remove debugging messages	Informational	Resolved
17	Remove duplicated code	Informational	Resolved
18	Unused events	Informational	Resolved
19	Missing entry point to remove ISM entries	Informational	Resolved

20	Mailbox should explicitly block same domain dispatch messages	Informational	Acknowledged
21	Usage of panics for error handling	Informational	Resolved
22	Lack of attached funds may cause inefficiencies	Informational	Resolved
23	Storage elements are not all available through queries	Informational	Partially Resolved
24	Potentially misleading attribute emitted	Informational	Resolved
25	Remove unused config for Multisig ISM	Informational	Resolved
26	Empty attributes	Informational	Resolved
27	Fix spelling errors errors	Informational	Resolved
28	“Migrate only if newer” pattern is not followed	Informational	Acknowledged
29	Usage of vulnerable dependencies	Informational	Acknowledged

Detailed Findings

1. Risk of Denial-of-Service of the merkle hook

Severity: Major

The `PostDispatch` execute message in `contracts/hooks/merkle/src/lib.rs:80` is callable by any sender and results in an insertion into the Merkle tree so long as the message matches the `latest_dispatch_id`. The `latest_dispatch_id` is a known value that can be queried from the mailbox. This will allow attackers to spam the Merkle tree by repeatedly calling `PostDispatch` with duplicates of the latest dispatched message. The duplicated message identifiers are not rejected and pollute the tree.

Consider the case of inserting into a Merkle tree 1,000,000th leaf. The Merkle tree, including its intermediate nodes (one less in number than its leaves), amounts to 1,999,999 32-byte nodes, totaling 63,999,968 bytes. The default Cosmos SDK rate for smart contract storage writes is 30 gas per byte, plus a flat write fee. Consequently, the gas required for storing a Merkle tree with 1,000,000 message IDs is approximately 1,919,999,040.

The actual monetary cost varies with the network's specific conditions and load at any given time. For example, in Osmosis, the minimum gas price can be 0.0025 uosmo, and the average is around 0.025 uosmo. Assuming an OSMO price of 0.33 USD, the minimum cost for this operation would be about 1.5 USD, and the average cost would be about 15 USD. The tree size grows linearly with its leaves. Thus, for the billionth message, the minimum gas cost could surpass 1,500 USD, while the average fee may reach around 15,610 USD.

As a consequence, the cost of operating the hook grows proportionally to the amount of messages processed and can reach levels unacceptable for users. A malicious party could deliberately invest funds to render the hook inoperative.

Recommendation

We recommend optimizing the Merkle tree structure for append operations, allowing the addition of new elements without the need to fully load and rewrite the unchanged portions of the tree. Additionally, rejecting the duplicated messages and whitelisting the callers of `PostDispatch` is recommended to protect the hook from attackers exploiting its escalating operational costs.

Status: Acknowledged

The client has acknowledged this finding, stating that the agent is resilient to duplicate insertions to the merkle tree. The client also states that this implementation serves to protect the merkle tree from spam at the agent level rather than the contract level as discussed in the recommendation.

2. Lack of `info.funds` transfer to `postDispatch` leads to failure of subsequent operations

Severity: Major

The `TransferRemote` message allows funds to be transferred via callbacks through the `Mailbox` contract directly to the destination chain if the `route` for `dest_domain` has been found.

However, when calling the `mailbox::dispatch` method in `contracts/warp/native/src/contract.rs:204`, no funds are transferred as `info.funds` that could cover the costs of gas of subsequent transactions and calls. Therefore, they will return errors and revert, making the functionality unusable.

This vulnerability was also independently identified by the client during the audit and resolved by introducing the "approve and transfer from" method.

Recommendation

We recommend ensuring that the contract properly handles funds during the message dispatch.

Status: Resolved

3. Resource intensive Multisig ISM `verify_message` query

Severity: Minor

The Multisig ISM contract includes an intensive computation for `verify_message` in `contracts/isms/multisig/src/query.rs:19-66`. The verification performs a linear pass through all provided signatures, recovering public keys from them, and performing a nested linear scan to find matching validators. This results in a quadratic computational complexity of the verification. Furthermore, two resource-intensive operations, `secp256k1_verify` and `eth_addr`, are executed for each signature. The code involved in this issue has been updated in Phase 2 of this audit but the issue remains.

Heavy computations in smart contracts lead to high gas fees and increase the risk of Denial-of-Service attacks. Note that queries to external contracts in CosmWasm impose gas restrictions, so the caller of this query may experience out-of-gas errors. The result of this is that messages may fail to be processed by the mailbox if the query exceeds wasmd smart query limits.

Recommendation

We recommend offloading as much computation as possible from the smart contract to relayers. The contract should only verify already processed data. Relayers can perform off-chain matching and submit validator indexes along with the signatures. This approach would simplify the verification process to a straightforward linear scan over the validators

vector within the contract. Alternatively, relayers could also handle the recovery of public keys from signatures and convert them to the required format. This may significantly reduce the computational burden on the contract, although it would increase the transaction size due to additional data being submitted by relayers.

Status: Acknowledged

The client states that they do not expect the computation to be problematic for the expected signature validation scenarios.

4. Warp contracts do not specify hook in `transfer_remote` function

Severity: Minor

The `transfer_remote` function in both warp contracts currently sends a mailbox dispatch message with a hook parameter hardcoded to `None`. This undermines the design of the protocol by not utilizing the hooks feature.

This issue was also independently identified by the client during this audit.

Recommendation

We recommend setting the hook value in the `transfer_remote` function call in both warp contracts.

Status: Resolved

5. Multisig ISM may cause out-of-gas errors

Severity: Minor

The Multisig ISM contract contains the `enroll_validator` function in `contracts/isms/multisig/src/execute/validator.rs:26`. This function exhibits inefficiencies that escalate costs as the number of validators rises:

1. `validators.0.iter` performs a linear pass through all already enrolled validators.
2. `validators.0.sort_by` operates with *O(N logN)* complexity, where *N* is the total number of validators.

The same inefficiencies impact the `unenroll_validator` function in this file.

This issue is classified with minor severity since only the owner has the authority to enroll and unenroll validators.

Recommendation

We recommend keeping the validators vector sorted in the smart contract's storage, eliminating the need for explicit sorting.

With a pre-sorted vector:

1. Searches for duplicates can be executed in *O(log N)* using binary search.
2. New entries can be inserted in *O(N)* by locating the insertion point and shifting the vector's tail. Advanced data structures can further reduce insertion complexity to *O(logN)*. Both of these ways to insert an entry ensure the collection of validators remains sorted post-insertion, effectively eliminating the computationally expensive sorting step.

Status: Resolved

6. Static multisig threshold design presents scalability concerns

Severity: Minor

The Multisig ISM (Interchain Security Module) relies on a static threshold parameter to verify messages based on the number of validators that have verified the message. This parameter's storage is defined on `contracts/isms/multisig/src/state.rs:15` as a mapping from secured domains to threshold values. A fixed threshold number is not reliable because as the number of validators increases, the probability of collusion or leaked keys also rises. Therefore, the threshold should always be adjusted according to the current number of validators.

Recommendation

We recommend using a proportion of the validators as the threshold instead of a fixed threshold amount. This approach ensures a consistent level of security regardless of the number of validators.

Status: Acknowledged

The client states that this will be managed at the operational level. ISM owners will be given the discretion to manage this parameter at their discretion.

7. A zero threshold will make it impossible to verify a message

Severity: Minor

The aggregate ism contract allows the minimum value of the `THRESHOLD` state parameter to be set to zero in `contracts/isms/aggregate/src/lib.rs:55`. In the functions responsible for verifying the message in `contracts/isms/aggregate/src/lib.rs:126-128` and

`contracts/isms/multisig/src/query.rs:55-59`, after obtaining confirmation from the first validator, the threshold value is decreased by one. Since the value was initially equal to zero, an underflow occurs, which will result in a panic. As a consequence, this operation cannot be completed successfully, leading to a state of unusability.

Recommendation

We recommend enforcing a minimum value for the `THRESHOLD` parameter such that at least one validator is required to verify the message.

Status: Resolved

8. Pausable endpoint not exposed

Severity: Minor

The `mailbox` contract is instantiated with the pausable feature in `contracts/core/mailbox/src/contract.rs:39`. However, the pausable-related execute entry points are not exposed in the `execute` function in lines `45-64`. Therefore, it is not possible to pause the contract after being instantiated, rendering this security mechanism ineffective.

Additionally, the contract is instantiated with pausable set to `false`. To ensure no users attempt to pass the dispatch or process messages before the `default_ism`, `default_hook`, and `required_hook` have been set, it could be beneficial to instantiate the contract in a paused state and once the initial setup occurs unpause the contract.

Recommendation

We recommend exposing the pausable execute entry points, as well as instantiating the contract in a paused state.

Status: Resolved

9. Strategic planning is crucial for Mailbox contract upgrades

Severity: Minor

The check at `contracts/core/mailbox/src/execute.rs:217-223` rejects messages from older `Mailbox` versions. Upgrading the destination contract with messages in transit would lead to failure of their delivery albeit being paid and verified.

To manage this, dispatching new messages might be temporarily halted using the `Pausable` hook. For domain-specific pausing, the `Router` hook can be employed. However, as `Mailbox` instances across different domains could have separate ownership, coordinated

upgrades necessitate prior agreements among all involved parties for synchronous lane updates.

Recommendation

When deploying a `Mailbox`, we recommend configuring sending messages only to `Mailbox` contracts managed by the same party. An alternative solution may be supporting the reception of messages meant for a previous version.

Status: Acknowledged

10. Missing address validation and normalization

Severity: Minor

Multiple contracts within the scope of this audit lack address validation or normalization steps. Some of the affected instances cause just a transaction failure when an incorrect address is provided, wasting gas. But others would render some features unusable until a valid address is recorded.

The following instances were found:

- `dispatch_msg.recipient_addr` in `contracts/core/mailbox/src/execute.rs:151-156`. Although it is not possible to validate the address as it belongs to a different chain, the address could be normalized to lowercase.
- `recipient` in `contracts/core/mailbox/src/execute.rs:215`
- `msg.recipient` in `contracts/hooks/routing-custom/src/lib.rs:185`
- `refund_address` in `contracts/igps/core/src/execute.rs:114`
- `router` in `packages/router/src/lib.rs:98`
- `set.route` in `packages/router/src/lib.rs:74`

Recommendation

We recommend implementing the fixes mentioned above to improve address handling.

Status: Partially Resolved

11. Lack of input validation

Severity: Minor

Multiple contracts within the scope of this audit lack input validation, for example of `HRP`, the gas token, and the oracle configurations. Invalid inputs can render the contracts unusable until a correct value is provided. In some cases parameter updates require deploying a new contract.

- `msg.hrp` in `contracts/core/va/src/contract.rs:46`
- `msg.gas_token` in `contracts/igps/core/src/contract.rs:34`
- `msg.hrp` in `contracts/igps/core/src/contract.rs:35`
- `msg.hrp` in `contracts/core/mailbox/src/contract.rs:27`
- `msg.hrp` in `contracts/isms/multisig/src/contract.rs:35`
- `config` in `contracts/igps/oracle/src/contract.rs:54` and `71`

Recommendation

We recommend:

- Implementing a best-effort validation on `HRP` so it only contains lowercase letters.
- Querying the `gas_token` supply to ensure that it is a valid token.
- Checking that the provided oracle `config` does not include an exchange rate of zero.

Status: Resolved

12. Aggregate hook does not consider alternate denominations

Severity: Minor

The `quote_dispatch` function in `contracts/hooks/aggregate/src/lib.rs:147` does not account for the possibility of having gas coins of different denominations. It incorrectly assumes that all gas coins are of the same denomination, which cannot be guaranteed. Ultimately this could cause an error further in the execution of the dispatch when the necessary funds are not present.

Recommendation

We recommend ensuring that all gas denominations are the same as the denomination of the `gas_total` being calculated.

Status: Resolved

13. Missing entry point to remove ISM entries

Severity: Minor

The `routing` contract implements a `Set` entry point in `contracts/isms/routing/src/contract.rs:55` to add ISM entries to the storage. However, there is no `Remove` entry point to delete an undesired entry.

Recommendation

We recommend implementing a `Remove` entry point to delete routes from the storage.

Status: Resolved

14. Maintainability considerations for Merkle tree

Severity: Minor

The Merkle package in `packages/interface/src/types/merkle.rs` contains several issues not posing any immediate threat, but potentially affecting future versions of the system.

1. In line `11`, the type of the constant `ZERO_HASHES` is incorrectly parameterized by constant `HASH_LENGTH`. The number of hashes should relate however to the Merkle tree's depth, not the hash value length. Currently, changing `TREE_DEPTH` results in runtime errors during Merkle tree operations.
2. In line `8`, the constant `MAX_LEAVES` is incorrect. The maximum number of a Merkle tree's leaves must be a power of `2`. The impact is that the very last leaf slot cannot be filled. For example, if `TREE_DEPTH` is set to 3, the 8th value is wrongly rejected by the tree.

Recommendation

We recommend implementing the following fixes:

1. The correct type signature for the constant `ZERO_HASHES` should be `[&str; TREE_DEPTH]`.
2. The correct value for `MAX_LEAVES` should be `2_u128.pow(TREE_DEPTH as u32)`.
3. Implementing unit tests for the Merkle tree is advisable, given its complexity and the optimization evident.
4. For better efficiency, using byte vectors for `ZERO_BYTES` and `ZERO_HASHES` is recommended over string types.

Status: Partially Resolved

15. Misleading `ContractErrors` and events emitted

Severity: Informational

The codebase contains some events and errors that may mislead users. Misleading errors have been found in:

- The `announce` function in `contracts/core/va/src/contract.rs:145-148` returns `Unauthorized` when `REPLAY_PROTECITONS` contains `replay_id` already.
- `ExecuteMsg::PostDispatch` in `contracts/hooks/merkle/src/lib.rs:94-98` returns `Unauthorized` when `latest_dispatch_id` is not the same as `decoded_msd.id`.

Additionally, the `SetRemoteGasData` function in `contracts/igps/oracle/src/contract.rs:63` does not emit the "owner" attribute.

This is different from the `SetRemoteGasDataConfigs` function, which contains the same functionality, but accepts a vector instead of a single object.

Recommendation

We recommend adjusting both error messages to describe the source of the error and to unify the attributes emitted in events in the indicated functions.

Status: Resolved

16. Remove debugging messages

Severity: Informational

The codebase contains several debugging messages, e.g. using `deps.api.debug`. It is best practice to remove these debug messages before releasing the code in production.

Recommendation

We recommend removing the following debug statements (locations based on phase 2 code freeze):

- `contracts/core/mailbox/src/execute.rs:50`
- `contracts/core/mailbox/src/execute.rs:235`
- `contracts/isms/multisig/src/query.rs:24`
- `contracts/isms/routing/src/contract.rs:84`

Status: Resolved

17. Remove duplicated code

Severity: Informational

The `enroll_validators` function in `contracts/isms/multisig/src/execute.rs:77-110` utilizes duplicated code to the definition of `enroll_validators` from the same file.

Recommendation

We recommend removing the duplicated function and instead importing the existing one.

Status: Resolved

18. Unused events

Severity: Informational

There are several events in the codebase that are currently unused:

- `Paused` in `contracts/hooks/aggregate/src/error.rs`
- `RouteNotFound` in `contracts/isms/aggregate/src/error.rs`
- `OwnershipTransferNotStarted` and `OwnershipTransferAlreadyStarted` in `contracts/isms/multisig/src/error.rs`
- `emit_init_transfer_ownership`, `emit_finish_transfer_ownership` and `emit_revoke_transfer_ownership` in `contracts/isms/multisig/src/event.rs`
- `InsufficientFunds` and `MessageNotFound` in `contracts/core/mailbox/src/error.rs`

Recommendation

We recommend using the above events or removing their definitions.

Status: Resolved

19. Mailbox should explicitly block same domain dispatch messages

Severity: Informational

The `dispatch` function in `contracts/core/mailbox/src/execute.rs:161` does not explicitly block messages where the destination domain is the same as the local domain. While this does not pose any security concerns because the dispatch messages cannot cause any harm to the contracts it may negatively impact user experience.

Recommendation

We recommend enforcing that the destination domain is different from the local domain.

Status: Acknowledged

20. Usage of panics for error handling

Severity: Informational

There are several instances of usage of the `expect` and `unwrap` functions for error handling in the codebase.

The usage of `expect` and `unwrap` is generally discouraged because they raise panics without a user-friendly error message. Panics also causes the wasm execution to abort, which does not allow handling the panic from the calling context.

Recommendation

We recommend returning errors with meaningful error messages rather than using panics, which will increase the user experience and maintainability of the codebase.

Status: Resolved

21. Lack of attached funds may cause inefficiencies

Severity: Informational

The `mailbox` contract contains a `get_required_value` function in `contracts/core/mailbox/src/execute.rs:27-72` which is called by the `dispatch` function in line `167`. The returned values determine the amount of funds attached to the `PostDispatch` messages that are sent to hooks and relayers.

If no funds have been attached to the call, line `41` returns `None` values which will result in messages without attached funds, even if that is a requirement. This will cause the transaction to fail at a later stage, unnecessarily wasting gas. Similarly, line `62` returns the received funds in case they are less than the required funds which will cause a failure.

Recommendation

We recommend returning an error in the affected lines to remove inefficiencies.

Status: Resolved

22. Storage elements are not all available through queries

Severity: Informational

The `va` contract does not expose the `HRP`, `MAILBOX`, and `LOCAL_DOMAIN` storage state values through smart queries in `contracts/core/va/src/contract.rs:75-82`. This forces users and other contracts to perform a raw query to read the stored value, tying their code to the current implementation of the `va` contract, which is error-prone.

Recommendation

We recommend exposing a smart query that returns the above-mentioned elements.

Status: Partially Resolved

23. Potentially misleading attribute emitted

Severity: Informational

In the `instantiate` function in `contracts/warp/native/src/contract.rs:80`, the `denom` attribute for a bridged denom may be misleading. The function is creating a `tokenfactory` token which has a different format and the denom emitted will only be the token's subdenom. The correct format is `factory/{creator address}/{subdenom}`.

Recommendation

We recommend emitting the full `tokenfactory` denom for bridged tokens.

Status: Resolved

24. Remove unused config for Multisig ISM

Severity: Informational

The `Config` struct in `contracts/isms/multisig/src/state.rs:6` is defined but it is not used anywhere in the contract. It is best practice to remove unused code.

Recommendation

We recommend removing the `Config` struct in `contracts/isms/multisig/src/state.rs:6`.

Status: Resolved

25. Empty attributes

Severity: Informational

The `Set` execute message in `contracts/isms/routing/src/contract.rs:68` currently returns empty attributes. It is best practice to return descriptive attributes to reflect the state changes being made.

Recommendation

We recommend emitting attributes in `contracts/isms/routing/src/contract.rs:68`.

Status: Resolved

26. Fix spelling errors

Severity: Informational

In `contracts/igps/core/src/query.rs:19`, the beneficiary variable is misspelled as "beneficairy". Additionally, in `contracts/core/va/src/contract.rs:159` and `162` "REPLAY_PROTECITONS" is misspelled.

Recommendation

We recommend fixing the spelling errors mentioned above.

Status: Resolved

27. "Migrate only if newer" pattern is not followed

Severity: Informational

The contracts within the scope of this audit are currently migrated without regard to their version. This can be improved by adding validation to ensure that the migration is only performed if the supplied version is newer.

Recommendation

We recommend following the "migrate only if newer" pattern defined in the CosmWasm documentation.

Status: Acknowledged

28. Usage of vulnerable dependencies

Severity: Informational

The codebase utilizes packages with known vulnerabilities. As reported in https://rustsec.org/advisories/RUSTSEC-2022-0093 and https://rustsec.org/advisories/RUSTSEC-2023-0052, the `ed25519-dalek` and `webpki` crates are affected by issues of high impact.

These vulnerabilities are not directly exploitable in a CosmWasm smart contract and do not affect any of the current code, but we note this as an informational finding to raise awareness for potential risks of using these packages and affected functions in future functionality.

Recommendation

We suggest verifying that the current code development process does not include any vulnerable dependencies, as well as periodically checking publicly known issues in the dependencies used.

Status: Acknowledged