PUBLIC

Code Assessment

of the Node Management Module Smart Contracts

August 29, 2023

Produced for

hopr

by

CHAINSECURITY

Contents

1 Executive Summary 3
2 Assessment Overview 5
3 Limitations and use of report 9
4 Terminology 10
5 Findings 11
6 Resolved Findings 12
7 Informational 21

1 Executive Summary

Dear Hoprnet Team,

Thank you for trusting us to help HOPRNet with this security audit. Our executive summary provides an overview of subjects covered in our audit of the latest reviewed contracts of Node Management Module according to Scope to support you in forming an opinion on their security risks.

HOPRNet implements a module for Safe multisignature contract that allows management and separation of the different keys that are needed for HOPR network functionality. Scope includes updated HoprChannels that can use such Safes and factory to deploy and configure them.

The most critical subjects covered in our audit are asset solvency, functional correctness and signature handling. Asset solvency and Signature handling are good. Functional correctness is high.

The general subjects covered are specification, front-running and integration with 3rd party systems. Security regarding all the aforementioned subjects is high.

In summary, we find that the codebase provides a good level of security.

It is important to note that security audits are time-boxed and cannot uncover all vulnerabilities. They complement but don't replace other vital measures to secure a project.

The following sections will give an overview of the system, our methodology, the issues uncovered and how they have been addressed. We are happy to receive questions and feedback to improve our service.

Sincerely yours,

ChainSecurity

1.1 Overview of the Findings

Below we provide a brief numerical overview of the findings and how they have been addressed.

Critical-Severity Findings	2
• Code Corrected	2
High-Severity Findings	0
Medium-Severity Findings	1
• Code Corrected	1
Low-Severity Findings	13
• Code Corrected	13

2 Assessment Overview

In this section, we briefly describe the overall structure and scope of the engagement, including the code commit which is referenced throughout this report.

2.1 Scope

The assessment was performed on the source code files inside the Node Management Module `https://github.com/hoprnet/hoprnet` repository based on the documentation files. Following files in the `packages/ethereum/contracts/src` directory of the repository were considered in scope:

```
Channels.sol
MultiSig.sol
interfaces/IAvatar.sol
interfaces/INetworkRegistryRequirement.sol
interfaces/INodeManagementModule.sol
interfaces/INodeSafeRegistry.sol
node-stake/NodeSafeRegistry.sol
node-stake/NodeStakeFactory.sol
node-stake/permissioned-module/CapabilityPermissions.sol
node-stake/permissioned-module/NodeManagementModule.sol
node-stake/permissioned-module/SimplifiedModule.sol
utils/EnumerableStringSet.sol
utils/EnumerableTargetSet.sol
utils/TargetUtils.sol
```

The table below indicates the code versions relevant to this report and when they were received.

V	Date	Commit Hash	Note
1	06 August 2023	41fa52c9c50b0029cd8329f04409b05d31eb063b	Initial Version
2	22 August 2023	e6ac2da904672da5c932b3e95ba6cbc37934643d	Version with fixes
3	27 August 2023	274b59e409e6bf48c6d7d675de2d9905dcf1f813	Version with fixes

For the solidity smart contracts, the compiler version `0.8.19` was chosen.

2.1.1 Excluded from scope

Any other files not explicitly mentioned in the scope section. In particular tests, scripts, external dependencies, and configuration files are not part of the audit scope. In particular, the `src/Crypto.sol` library is out-of-scope.

2.1.2 Assumptions

All imported files, contracts and libraries are assumed to function according to their high level specification. The libraries in `vendor/solidity` folder of the repository are assumed to be unmodified. The `src/Crypto.sol` library is assumed to be correct. In addition, the mechanism of deciding a winning ticket is out of scope. It is assumed to be fair to both parties of a channel, and no parties shall be able to predict a winning ticket in advance.

2.2 System Overview

This system overview describes the initially received version (Version 1) of the contracts as defined in the Assessment Overview.

Furthermore, in the findings section, we have added a version icon to each of the findings to increase the readability of the report.

HOPRNet is building a privacy focused network featuring a build-in incentive model. The provided `HoprChannels` (or `Channels`) contract allows nodes to create payment channels between each other and allow the transfer of wxHOPR tokens between them. The transfers are done via tickets, that have a certain predefined probability to win. HOPRNet also implements a NodeStakeFactory, that one can use to deploy a Safe multisignature wallet to hold the funds. Factory deployed Safe is deployed with configured NMM (Node Management Module) to manage the access of several HOPR chain nodes. The NMM implements default and granular access control towards the callable targets from the Safe.

2.2.1 Channels

Node can use the Channels contract to open, close, and keep track of unidirectional payment channels to other nodes. A channel is uniquely identifiable by the tuple `(address source,address target)`, which holds the funds that target address may claim for the work of relaying source's messages.

A ticket is issued by source with each message to target, which leads to probabilistic payments for the relaying work. To keep the winning probability fair, the success of redeeming a ticket is dependent on both the packet creator entropy as well as the ticket claimer entropy, which are unknown to each other in advance. Target has a commitment that is unknown to source. If source is not a packet creator, it has no control over the winning probability and thus cannot influence the win chance probability. And target does not know the proof of relay in advance and has to relay message to the next node to reveal it.

Channels expose interfaces that support both the nodes that use Safe with NMM and the nodes that do not use Safe. The former can use the `fundChannelSafe` and `redeemTicketSafe` entrypoints, while the latter can use the `fundChannel()` and `redeemTicket()` entrypoints.

2.2.1.1 Fund a channel

A channel can be funded by calling `fundChannelSafe()` or `fundChannel()` directly, which enforces the funding token amount is within a range and the two parties of the channel are different. Channel can be funded only if it is not in the `PENDING_TO_CLOSE` state. The balance of the channel will be increased by the transferred amount. In case the channel is `CLOSED`, the channel will be reopened with an `epoch` increased by 1 and `ticketIndex` set to 0.

One can also directly use `wxHOPR.send()` function to send the tokens directly to the Channel contract. This will triggers the `IERC777Recipient.tokensReceived()` callback and funds a single unidirectional channel or two (A to B and B to A) unidirectional channels depending on the `userData`.

2.2.1.2 Close a channel

To close the channel, the source needs to call `initiateOutgoingChannelClosure()` or `initiateOutgoingChannelClosureSafe()` to initiate the channel closure. The `noticePeriodChannelClosure` time period allows target node on the other side of the channel to redeem the remaining tickets. After the `noticePeriodChannelClosure`, source can eventually call `finalizeOutgoingChannelClosureSafe` or `finalizeOutgoingChannelClosure` to finalize the closure, which transfers the tokens left in the channel to channel src. Channel target can also close an incoming payment channel immediately by `closeIncomingChannel` or `closeIncomingChannelSafe`.

2.2.1.3 Redeem a ticket

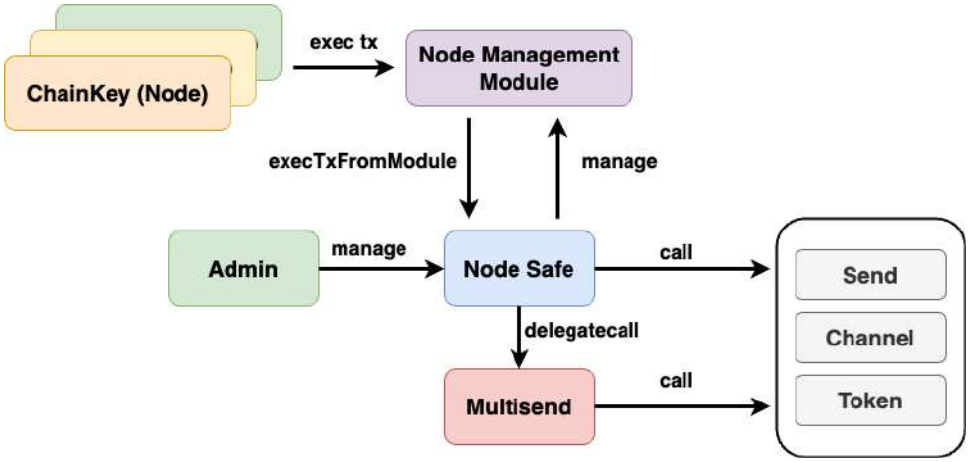
A node can redeem a ticket through `redeemTicketSafe` or `redeemTicket` with the issuer-signed ticket, the proof of relay, and the verifiable random function data. The following steps will be taken to validate a ticket:

1. The channel status must be `OPEN` or `PENDING_TO_CLOSE`, and the channel current epoch must match the ticket's epoch.
2. The ticket's index is larger than the channel's index, and channel's balance is sufficient.
3. The ticket falls within the winning area given the winning probability.
4. The random function verification succeeds.
5. The signature of the redeemable ticket is valid from the issuer, and the ticket is indeed for the redeemer.

The reward token amount will be transferred to the ticker redeemer, in case the channel from it to source is `CLOSED`. Otherwise, the reward tokens will be used to top up the redeemer's payment channel (from redeemer to issuer)

2.2.2 Safe and Management Module

To isolate the nodes operations and funds management, a Safe and a NMM (Node Management Module) is implemented. One can leverage the NodeStakeFactory to set up a new Safe and NMM, where multiple chainkeys could be added to the NMM, which enforces access control towards members and targets and triggers Safe to conduct certain calls or delegatecalls. Different components of the system can be found in the following figure.



Upon deployment, an array of admins are passed to the Safe, which has the ultimate privilege of co-signing to manage the modules and funds.

The NMM stores the access control information into a `Role` struct, which consists of a `TargetSet`, a mapping of role members, and a mapping of granular capabilities. Each target is a `uint256` which encodes the 20-byte target address and 12 one-byte flags. The first 3 flags are clearance, target type, and default permission. The rest are capability permission flags. Only the members of the Role can perform the operations to targets specified by the permission flags and the granular capabilities.

The NMM provides the main entrypoints for the registered nodes (chainkeys), which need to pass the access control to trigger specific actions on the Safe. The possible actions are limited to:

- call HoprChannels.fundChannelSafe
- call HoprChannels.redeemTicketSafe
- call HoprChannels.closeIncomingChannelSafe
- call HoprChannels.initiateOutgoingChannelClosureSafe
- call HoprChannels.finalizeOutgoingChannelClosureSafe

- call HoprToken.approve(address,uint256)
- call HoprToken.send(address,uint256,bytes)
- send native tokens
- delegatecall multisend

Granular access control is achieved by limiting the allowed parameters passed to the functions above. In case of a multisend operation, the individual transactions inside the data will be loaded and inspected. Afterwards, the node module will call the Safe with the data, and the Safe will eventually execute the calls or delegatecalls.

2.2.3 Roles and Trust Model

The admin key owners are privileged roles that are fully trusted to not misbehave. A compromised admin key will allow attacker to get full control of the Safe and NMM and thus the funds, chainkeys in NMM. The owners of individual chainkeys are considered trusted and must not perform malicious actions. Compromised chainkey will allow attacker to open and close channels. Other consequences in both cases are also possible.

3 Limitations and use of report

Security assessments cannot uncover all existing vulnerabilities; even an assessment in which no vulnerabilities are found is not a guarantee of a secure system. However, code assessments enable the discovery of vulnerabilities that were overlooked during development and areas where additional security measures are necessary. In most cases, applications are either fully protected against a certain type of attack, or they are completely unprotected against it. Some of the issues may affect the entire application, while some lack protection only in certain areas. This is why we carry out a source code assessment aimed at determining all locations that need to be fixed. Within the customer-determined time frame, ChainSecurity has performed an assessment in order to discover as many vulnerabilities as possible.

The focus of our assessment was limited to the code parts defined in the engagement letter. We assessed whether the project follows the provided specifications. These assessments are based on the provided threat model and trust assumptions. We draw attention to the fact that due to inherent limitations in any software development process and software product, an inherent risk exists that even major failures or malfunctions can remain undetected. Further uncertainties exist in any software product or application used during the development, which itself cannot be free from any error or failures. These preconditions can have an impact on the system's code and/or functions and/or operation. We did not assess the underlying third-party infrastructure which adds further inherent risks as we rely on the correct execution of the included third-party technology stack itself. Report readers should also take into account that over the life cycle of any software, changes to the product itself or to the environment in which it is operated can have an impact leading to operational behaviors other than those initially determined in the business specification.

4 Terminology

For the purpose of this assessment, we adopt the following terminology. To classify the severity of our findings, we determine the likelihood and impact (according to the CVSS risk rating methodology).

- *Likelihood* represents the likelihood of a finding to be triggered or exploited in practice
- *Impact* specifies the technical and business-related consequences of a finding
- *Severity* is derived based on the likelihood and the impact

We categorize the findings into four distinct categories, depending on their severity. These severities are derived from the likelihood and the impact using the following table, following a standard risk assessment procedure.

Likelihood	Impact		
	High	Medium	Low
High	Critical	High	Medium
Medium	High	Medium	Low
Low	Medium	Low	Low

As seen in the table above, findings that have both a high likelihood and a high impact are classified as critical. Intuitively, such findings are likely to be triggered and cause significant disruption. Overall, the severity correlates with the associated risk. However, every finding's risk should always be closely checked, regardless of severity.

5 Findings

In this section, we describe any open findings. Findings that have been resolved have been moved to the Resolved Findings section. The findings are split into these different categories:

- Security: Related to vulnerabilities that could be exploited by malicious actors
- Design: Architectural shortcomings and design inefficiencies
- Correctness: Mismatches between specification and implementation

Below we provide a numerical overview of the identified findings, split up by their severity.

Critical-Severity Findings	0
High-Severity Findings	0
Medium-Severity Findings	0
Low-Severity Findings	0

6 Resolved Findings

Here, we list findings that have been resolved during the course of the engagement. Their categories are explained in the Findings section.

Below we provide a numerical overview of the identified findings, split up by their severity.

Critical-Severity Findings	2

- HoprChannels ERC777 Reentrancy Code Corrected
- Winning Ticker Can Be Redeemed Multiple Times Code Corrected

High-Severity Findings	0

Medium-Severity Findings	1

- EIP-712 Incompliant Signed Message Code Corrected

Low-Severity Findings	13

- Incorrect indexEvent Input Code Corrected
- Dependencies Between Source File Folders Code Corrected
- DomainSeparator Is Not Recomputed After a Change of Chain ID Code Corrected
- HoprNodeSafeRegistry Is Not an INodeSafeRegistry Code Corrected
- HoprNodeStakeFactory Can Clones Any Module Code Corrected
- IHoprNodeSafeRegistry Is a Contract and Not an Interface Code Corrected
- Incorrect Flag Position Upper Bound Code Corrected
- Incorrect Specifications and Comments Code Corrected
- Missing Input Checks at tokensReceived Code Corrected
- Signatures Can Be Replayed Code Corrected
- TargetUtils Incorrect Iterator Bound Code Corrected
- Timestamp Is Not Updated With Snapshot Code Corrected
- isNodeSafeRegistered Returns True for Unregistered Pairs if safeAddress==0 Code Corrected

6.1 HoprChannels ERC777 Reentrancy

Security Critical Version 1 Code Corrected

CS-HPRNMM23-001

An attacker can leverage the ERC777 capability of the wxHOPR token to drain the funds of HoprChannels contract.

Attack vector:

Assume the attacker deploys the following pair of contracts at `ALICE` and `BOB` addresses respectively.

```
contract Bob {

    function close() public {
        HoprChannels channels = HoprChannels(0x...);
```

```
        channels.closeIncomingChannel(ALICE);

    }
}
contract Alice is IERC777Recipient {

    bool once = false;

    function tokensReceived(...) public {
        if (once) {
            return;
        }
        once = true;
        HoprChannels channels = HoprChannels(0x...);
        channels.fundChannel(BOB, 1);
        Bob(BOB).close();
    }

    function hack() public {

        _ERC1820_REGISTRY.setInterfaceImplementer(address(this),
        TOKENS_RECIPIENT_INTERFACE_HASH, address(this));

        HoprChannels channels = HoprChannels(0x...);
        channels.fundChannel(BOB, 10);
        Bob(BOB).close();
    }

}
```

Exploit scenario:

When `Alice.hack()` is called, the following happens:

1. Alice's contract registers itself as its own ERC777TokensRecipient.
2. Alice funds outgoing channel to Bob with 10 wxHOPR.
3. Bob closes the incoming channel with Alice.
4. During the execution of `closeIncomingChannel()` the 10 wxHOPR tokens are transferred to Alice.
5. The `tokensReceived()` function of Alice is called. During this call:
    1. Alice funds outgoing channel to Bob with 1 wxHOPR. Balance of the channel becomes 11 wxHOPR.
    2. Bob closes the incoming channel with Alice and 11 wxHOPR tokens are transferred to Alice.
    3. This time the `tokensReceived()` function of Alice does nothing. The balance of the channel is set to 0.
6. The `closeIncomingChannel()` that started on step 4. sets the balance of channel to 0.

As a result, attacker using 10+1 token can withdraw 10+11 wxHOPR tokens from the channel. The attacker can loop the reentrancy even more time for more profit.

Cause:

The change of channel balance to 0 happens after the reentrant call to `token.transfer()` in the `_closeIncomingChannelInternal()` function. Thus, checks-effects-interactions pattern is effectively violated. Similar violations happen in other functions of the `HoprChannels` contract:

- `_finalizeOutgoingChannelClosureInternal()` sets the channel balance to 0 after the reentrant call to `token.transfer()`.
- `_redeemTicketInternal()` calls `indexEvent` and emits events after `token.tranfer()` call in case when the earning channel is closed. This effectively can lead do the wrong order of events in the event log or a different snapshot root.

Code corrected:

The code has been corrected by moving the token transfer to the end of the function in all relevant functions.

6.2 Winning Ticker Can Be Redeemed Multiple Times

Security | Critical | Version 1 | Code Corrected

CS-HPRNMM23-002

Assume Alice has outgoing channel to Bob with 1 as `ticketIndex`. Following scenario is possible:

1. Alice provides Bob 4 non-winning tickets with `ticketIndex` 2, 3, 4, 5 and a winning ticket with `ticketIndex` 6. All of those tickets are non-aggregated and thus their `indexOffset` is 1.
2. Bob redeems the winning ticket. In the `_redeemTicketInternal` function, the `ticketIndex` of the spending channel is updated as: `spendingChannel.ticketIndex += indexOffset`. The `ticketIndex` of the spending channel is now 2.
3. Bob can redeem the ticket again, because the only requirement on `ticketIndex` is that it is greater than the `ticketIndex` of the spending channel.

Thus, same winning ticket can be redeemed multiple times. The `ticketHash` signature from Alice does not prevent this, because it does not contain any nonce or other replay protection mechanism.

Code corrected:

The `ticketIndex` of the spending channel has been updated as: `spendingChannel.ticketIndex = TicketIndex.wrap(baseIndex + baseIndexOffset)`.

And the check of ticket validity has been adjusted to require: `(baseIndexOffset >= 1) && (baseIndex >= currentIndex)`.

6.3 EIP-712 Incompliant Signed Message

Correctness | Medium | Version 1 | Code Corrected

CS-HPRNMM23-003

The EIP-712 compliant message should start with a two-byte prefix "0x1901" followed by the domainSeparator and the message hash struct. Whereas two 32-bytes are used in the following cases for the prefix because of `abi.encode`. Consequently the signatures generated by the mainstream EIP-712 compliant libraries cannot be verified in these smart contracts:

1. `registerSafeWithNodeSig()` in NodeSafeRegistry.
2. `_getTicketHash()` in Channels.

Code corrected:

The `abi.encodePacked` has been used instead of `abi.encode` to generate the message hash struct.

6.4 Incorrect indexEvent Input

Correctness | Low | Version 2 | Code Corrected

CS-HPRNMM23-021

The channel will call `indexEvent()` when emitting a new event. In the following case, `indexEvent()` is incorrectly invoked with an extra `channel.balance` field compared to the emitted event. As a result, the snapshot will contain an incorrect event.

```
indexEvent(abi.encodePacked(ChannelOpened.selector, self, account,
    channel.balance));
emit ChannelOpened(self, account);
```

Code corrected:

The redundant field `channel.balance` has been removed from the `indexEvent` input.

6.5 Dependencies Between Source File Folders

Design | Low | Version 1 | Code Corrected

CS-HPRNMM23-004

The `packages/ethereum/contracts` foundry project have following problems

- Some of the files in `src` folder depend on smart contracts from `test` folder.
- Some of the files in `src` folder depend on smart contracts from `script` folder.

Such dependencies are considered as bad practice and should be avoided. Potential risks are:

- Separation of concerns is not respected. Testing and setup code should not impact production code.
- Risk of deploying test code to production is increased.
- Maintenance of the project is more difficult. Changes in `test` or `script` folders may impact production code.

Code corrected:

HOPRNet responded:

> The placement of specific library files and contracts have been reorganized to align with improved structuring and imports.

6.6 DomainSeparator Is Not Recomputed After a Change of Chain ID

Design | Low | Version 1 | Code Corrected

CS-HPRNMM23-005

In contracts Channels and NodeSafeRegistry, the `domainSeparator` is defined as an immutable in the constructor and used in the signature verification. In case there is a fork, the contracts will still verify a signature based on the old `domainSeparator`, whereas the forked chain is associated with a different chain ID. Besides, the signature targeted to the original chain can be replayed to the forked chain. In order to support the potential forked chains and avoid signature replay, the `domainSeparator` needs to be recomputed based on on-chain chain ID.

Code corrected:

Function `updateDomainSeparator` has been introduced in contracts Channels and NodeSafeRegistry to update the `domainSeparator` based on the on-chain chain ID.

6.7 HoprNodeSafeRegistry Is Not an INodeSafeRegistry

Design | Low | Version 1 | Code Corrected

CS-HPRNMM23-006

The contract does not inherit from INodeSafeRegistry, which means the compiler will not check that the contract implements all functions correctly.

Code corrected:

The INodeSafeRegistry contract has been removed.

6.8 HoprNodeStakeFactory Can Clones Any Module

Design | Low | Version 1 | Code Corrected

CS-HPRNMM23-007

The `clone()` function of the HoprNodeStakeFactory receives `moduleSingletonAddress` as a parameter. However, no checks are performed to ensure that the address is a valid module. While this is not immediately a problem, since it concerns only the msg.sender itself. However, the event `NewHoprNodeStakeModule` does not mention the address of the module. This complicates the process of verifying that the module is indeed a valid module.

Code corrected:

Description of Changes:

HOPRNet has adjusted the NewHoprNodeStakeModule event to include an indexed parameter, improving event clarity. And events have been separated from "HoprNodeStakeFactory" into an abstract contract named "HoprNodeStakeFactoryEvents."

Said changes allow inspection of the module address, and thus, verification of the module's validity.

6.9 IHoprNodeSafeRegistry Is a Contract and Not an Interface

Design Low Version 1 Code Corrected

CS-HPRNMM23-008

The `I` prefix is usually used for interfaces, not contracts. `IHoprNodeSafeRegistry` lies inside of `interfaces` folder whereas it is actually a contract.

Code corrected:

The INodeSafeRegistry contract has been removed.

6.10 Incorrect Flag Position Upper Bound

Correctness Low Version 1 Code Corrected

CS-HPRNMM23-009

A customized type `Target` is an alias of `uint256`, which is used to store a target address associated with 12 one-byte flags (3 general flags with 9 capability permission flags). In contract `util/TargetUtils.sol`, `getDefaultCapabilityPermissionAt()` will return the capability permission flag at a certain index: `position`. However, the upper bound of position is 9 instead of 8. As a result, the function will not revert upon reading the out-of-bound 9th permission flag and will always return 0 due to 256 left shifts.

Code corrected:

The code has been corrected: a proper upper bound of 8 is checked before reading the permission flag.

6.11 Incorrect Specifications and Comments

Correctness Low Version 1 Code Corrected

CS-HPRNMM23-010

Several incorrect specifications and comments are identified:

1. `checkMultisendTransaction()` will disassemble the data into individual transactions for access checks. Each transaction consists of 1 byte operation, 20 bytes target address, 32 bytes value, and the actual transaction data. The inspected actual transaction data locates at an offset of 53 bytes instead of 85 bytes in the comments.
2. The input encoded data of `decodeFunctionSigsAndPermissions()` encodes function signature in a right-padded way and permissions in a left-padded way. The index of permissions grows from right to left, while the specifications incorrectly state the other direction.

Code corrected:

Both specifications and comments have been corrected.

6.12 Missing Input Checks at tokensReceived

Design | Low | Version 1 | Code Corrected

CS-HPRNMM23-011

Besides funding a channel by `fundChannelSafe()` or `fundChannel()`, a node can also directly send tokens to the Channel contract, which triggers the `tokensReceived()` callback and funds one channel or a bidirectional channel depending on the `userData`. `fundChannelSafe()` and `fundChannel()` enforce the balance and channel parties validations, nevertheless, `tokensReceived()` does not. As a result, a node can fund a channel with a balance out of restrictions or with same parties on both sides of a channel.

Code corrected:

Description of Changes:

- Moved validateBalance and validateChannelParties from external functions (fundChannelSafe and fundChannel) to the internal function _fundChannelInternal. This allows tokensReceived to perform checks on balance and channel parties.
- Moved _fundChannelInternal before token.transferFrom in fundChannelSafe and fundChannel functions

6.13 Signatures Can Be Replayed

Design | Low | Version 1 | Code Corrected

CS-HPRNMM23-012

The signature used in `HoprNodeSafeRegistry.registerSafeWithNodeSig()` doesn't have a nonce, so it can be replayed.

Thus, any arbitrary `msg.sender` can register a node again using the same signature, even if the Safe has deregistered it. Effectively, only the node chain address that used `registerSafeByNode()` can be deregistered.

In addition, for deregistration the node should be a member of a Safe. Otherwise, the node cannot deregister itself from the registry.

The correct deregister way is assumed to be:

1. Deregister at the registry.
2. Remove the node from the Safe NodeManagementModule.

If these actions are not performed in a single transaction, a malicious party can register the node again after step 1 and break this flow.

Code corrected:

A nonce has been added as a parameter in signed data. The nonce of the given chain address will be incremented on each registration.

6.14 TargetUtils Incorrect Iterator Bound

Correctness Low Version 1 Code Corrected

CS-HPRNMM23-013

A customized type `Target` is an alias of `uint256` to store a 20-byte target address associated with 12 one-byte flags (3 general flags and 9 capability permission flags). In contract `util/TargetUtils.sol`, `decodeDefaultPermissions()` will retrieve the address and individual flags from the packed target input. When decoding the capability permission flags, the iterator falsely starts from 0 and ends at 8 (176 + 8 * i, for i in [0,8]). As a result, the last general flag with the first 8 capability permission flags are returned as the 9 capability permission flags.

Code corrected:

The starting index has been fixed and is 184 now.

6.15 Timestamp Is Not Updated With Snapshot

Design Low Version 1 Code Corrected

CS-HPRNMM23-014

In an out-of-scope contract `Ledger`, `indexEvent()` will update the `lastestRoot.rootHash` when it is called, and it will also push the `lastestRoot` to the `lastestSnapshotRoot` if a `snapshotInterval` has elapsed. However, the `latestRoot.timestamp` is not updated together with the `latestSnapshotRoot`. Consequently the `lastestSnapshotRoot` will be updated every time.

Code corrected:

The `latestRoot.timestamp` is updated together with the `latestSnapshotRoot`.

6.16 isNodeSafeRegistered Returns True for Unregistered Pairs if safeAddress==0

Correctness Low Version 1 Code Corrected

CS-HPRNMM23-015

In NodeSafeRegistry, `isNodeSafeRegistered()` returns if the input chainkey address is registered with the input safe address. In case a chainkey is not registered and the input safeAddress is 0, this function will return true. This may be unexpected for the external systems.

Code corrected:

If node is not registered to any safe, `isNodeSafeRegistered()` will return false. Thus, the 0 address of safe will not be considered as a registered safe.

6.17 ERC777 Reentrancy in fundChannel

Informational | Version 1 | Code Corrected

CS-HPRNMM23-016

`fundChannelSafe()` and `fundChannel()` will call `token.transferFrom()` to pull tokens, which will trigger the callback to the token spender's registered hook. What happens before the transfer is:

1. Validation of the safe.
2. Validation of the input balance.
3. Validation of the parties addresses.

The only thing that the token spender can do is to register or deregister at the SafeRegistry which tricks the first modifier. For example, assume a node A without registering a safe at the beginning:

1. A first calls `fundChannel()`.
2. In the callback, A calls `registerSafeByNode()`.

As a result, A successfully funds a channel through `fundChannel()` while it is already registered with a safe. This reentrancy does not have an explicit influence to the contracts though it could break the assumptions.

Code corrected:

The code has been corrected to avoid reentrancy. The token transfer is now done at the end of the function, after all the state changes have been done.

6.18 Order of Evaluation Can Be Enforced

Informational | Version 1 | Code Corrected

CS-HPRNMM23-020

The Solidity documentation states:

> https://docs.soliditylang.org/en/latest/ir-breaking-changes.html#semantic-only-changes
>
> For the old code generator, the evaluation order of expressions is unspecified. For the new code generator, we try to evaluate in source order (left to right), but do not guarantee it. This can lead to semantic differences.

The new code generator is not yet the default. This means that the order of evaluation of expressions is not guaranteed. Explicit brackets can be used to enforce the order of evaluation in e.g. `decodeDefaultPermissions()`

```
targetPermission = TargetPermission(uint8(Target.unwrap(target) << 176 >> 248));

uint8(Target.unwrap(target) << (184 + 8 * i) >> 248)
```

Code corrected:

HOPRNet has added explicit brackets to enforce the evaluation order.

7 Informational

We utilize this section to point out informational findings that are less severe than issues. These informational issues allow us to point out more theoretical findings. Their explanation hopefully improves the overall understanding of the project's security. Furthermore, we point out findings which are unrelated to security.

7.1 DomainSeparator Requires Manual Recompute After a Fork

Informational **Version 2**

CS-HPRNMM23-017

`updateDomainSeparator()` is a public function that recompute and update the domainSeparator in case of a fork. It requires manual invocation upon a fork. Signatures on the forked chain are still replayable before the call to `updateDomainSeparator()`. In case of supporting a forked chain, we assume this function will be invoked immediately.

7.2 Users Can Flash Loan by Fund Channel Reentrancy

Informational **Version 2**

CS-HPRNMM23-018

`fundChannelSafe()` and `fundChannel()` will update the internal balance in `_fundChannelInternal()` before calling `token.transferFrom()` to pull wxHOPR tokens. A user can implement and register a token sender callback, which will be invoked before the token transfer. In this callback, it can flashloan any amount of wxHOPR within the current liquidity of the channel contract by calling `closeIncomingChannel()`. At the end of the callback, the flashloan will be repaid by the real transfer of the tokens. Here is an example:

```
contract Bob {

    function close() public {
        // close the channel to get desiredAmount
        HoprChannels channels = HoprChannels(0x...);
        channels.closeIncomingChannel(ALICE);

        // customized logic here
    }
}

contract Alice is IERC777Sender {

    function tokensToSend(...) public {
        Bob(BOB).close();
    }

    function flashloan(uint256 desiredAmount) public {
        _ERC1820_REGISTRY.setInterfaceImplementer(address(this),
        TOKENS_SENDER_INTERFACE_HASH, address(this));
```

```
        HoprChannels channels = HoprChannels(0x...);
        channels.fundChannel(BOB, desiredAmount);
    }
}
```

7.3 isContract Check Can Be Bypassed

Informational **Version 2**

CS-HPRNMM23-019

`addNodeSafe()` has a check `nodeChainKeyAddress.isContract()`. However, if this node is a contract, and it calls the registry during its construction, the check will fail. Thus, node that is a contract can still be added to the registry.