// HALBORN

Benqi - LiquidStaking

Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **November 13th, 2021 - February 14th, 2022**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY 3
CONTACTS 3
1 EXECUTIVE OVERVIEW 4
1.1 INTRODUCTION 5
1.2 AUDIT SUMMARY 5
1.3 TEST APPROACH & METHODOLOGY 5
RISK METHODOLOGY 6
1.4 SCOPE 8
2 ASSESSMENT SUMMARY & FINDINGS OVERVIEW 9
3 FINDINGS & TECH DETAILS 10
3.1 (HAL-01) INVALID SETTER CAUSING POOL INCONSISTENCY - CRITICAL 12
Code Location 12
Risk Level 12
Recommendations 12
Remediation Plan 12
3.2 (HAL-02) USAGE OF MEMORY INSTEAD OF STORAGE - CRITICAL 13
Description 13
Risk Level 14
Recommendation 14
Remediation Plan 14
3.3 (HAL-03) CONTRACT LOCKED - CRITICAL 15
Risk Level 15
Remediation Plan 16
3.4 (HAL-04) CONTRACT MISSING OWNER - CRITICAL 17

Risk Level 17
Recommendation 17
Remediation Plan 17
3.5 (HAL-05) UNDERFLOW ON THE USAGE OF BLOCK.TIMESTAMP - MEDIUM 18
Risk Level 18
Recommendation 18
Remediation Plan 19
3.6 (HAL-06) OUT-OF-BOUNDS ACCESS - HIGH 20
POC 20
Recommendation 23
Remediation Plan 23
3.7 (HAL-07) CODE DUPLICATION - INFORMATIONAL 24
Risk Level 24
Recommendation 24
Remediation Plan 24
4 Manual Testing 24
4.1 NodeCoordinator 26

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	01/14/2022	Ferran Celades
1.0	Final Version	01/15/2022	Gabi Urrutia

CONTACTS

CONTACT	COMPANY	EMAIL
Rob Behnke	Halborn	Rob.Behnke@halborn.com
Steven Walbroehl	Halborn	Steven.Walbroehl@halborn.com
Gabi Urrutia	Halborn	Gabi.Urrutia@halborn.com
Ferran Celades	Halborn	Ferran.Celades@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Benqi engaged Halborn to conduct a security audit on their smart contracts beginning on November 13th, 2021 and ending on February 14th, 2022 . The security assessment was scoped to the smart contracts provided to the Halborn team.

1.2 AUDIT SUMMARY

The team at Halborn was provided two weeks for the engagement and assigned a full-time security engineer to audit the security of the smart contract. The security engineer is a blockchain and smart-contract security expert with advanced penetration testing, smart-contract hacking, and deep knowledge of multiple blockchain protocols.

The purpose of this audit is to:

- Ensure that smart contract functions operate as intended
- Identify potential security issues with the smart contracts

In summary, Halborn identified some security risks that were mostly addressed by the Benqi team.

1.3 TEST APPROACH & METHODOLOGY

Halborn performed a combination of manual and automated security testing to balance efficiency, timeliness, practicality, and accuracy regarding the scope of this audit. While manual testing is recommended to uncover flaws in logic, process, and implementation, automated testing techniques help enhance coverage of the bridge code and can quickly identify items that do not follow security best practices. The following phases and associated tools were used throughout the term of the audit:

- Research into architecture and purpose
- Smart contract manual code review and walkthrough
- Graphing out functionality and contract logic/connectivity/functions (solgraph)
- Manual assessment of use and safety for the critical Solidity variables and functions in scope to identify any arithmetic related vulnerability classes
- Manual testing by custom scripts
- Static Analysis of security for scoped contract, and imported functions (Slither)
- Testnet deployment (Brownie, Remix IDE)

RISK METHODOLOGY:

Vulnerabilities or issues observed by Halborn are ranked based on the risk assessment methodology by measuring the **LIKELIHOOD** of a security incident and the **IMPACT** should an incident occur. This framework works for communicating the characteristics and impacts of technology vulnerabilities. The quantitative model ensures repeatable and accurate measurement while enabling users to see the underlying vulnerability characteristics that were used to generate the Risk scores. For every vulnerability, a risk level will be calculated on a scale of 5 to 1 with 5 being the highest likelihood or impact.

RISK SCALE - LIKELIHOOD

5 - Almost certain an incident will occur.
4 - High probability of an incident occurring.
3 - Potential of a security incident in the long term.
2 - Low probability of an incident occurring.
1 - Very unlikely issue will cause an incident.

RISK SCALE - IMPACT

5 - May cause devastating and unrecoverable impact or loss.
4 - May cause a significant level of impact or loss.
3 - May cause a partial impact or loss to many.
2 - May cause temporary impact or loss.

1 - May cause minimal or un-noticeable impact.

The risk level is then calculated using a sum of these two values, creating a value of 10 to 1 with 10 being the highest level of security risk.



10 - CRITICAL
9 - 8 - HIGH
7 - 6 - MEDIUM
5 - 4 - LOW
3 - 1 - VERY LOW AND INFORMATIONAL

1.4 SCOPE

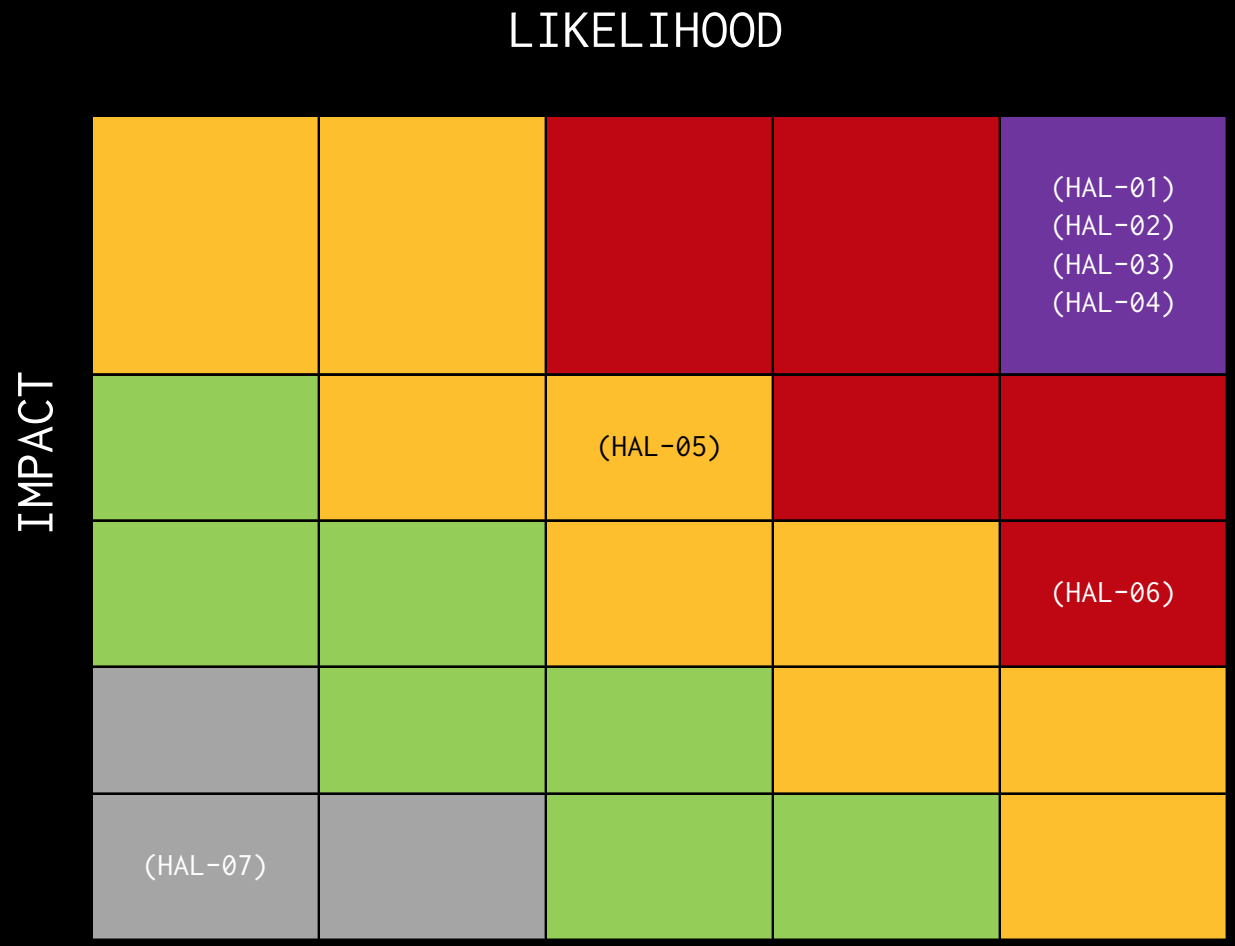
IN-SCOPE:
The security assessment was scoped on all the contract present on the Benqi-fi/liquid-staking repository.

The contracts were renamed and moved around several times. The engagement did start with commit 21f7e06324c7eb471b2e8d3a2dd226712dec5090.

OUT-OF-SCOPE:
Other smart contracts in the repository, external libraries and economical attacks.

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
4	1	1	0	1



EXECUTIVE OVERVIEW

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
INVALID SETTER CAUSING POOL INCONSISTENCY	Critical	SOLVED - 11/24/2021
USAGE OF MEMORY INSTEAD OF STORAGE	Critical	SOLVED - 10/26/2021
CONTRACT LOCKED	Critical	SOLVED - 10/26/2021
CONTRACT MISSING OWNER	Critical	SOLVED - 11/18/2021
UNDERFLOW ON THE USAGE OF BLOCK.TIMESTAMP	Medium	SOLVED - 10/26/2021
OUT-OF-BOUNDS ACCESS	High	SOLVED - 11/24/2021
CODE DUPLICATION	Informational	SOLVED - 11/18/2021

FINDINGS & TECH DETAILS

3.1 (HAL-01) INVALID SETTER CAUSING POOL INCONSISTENCY - CRITICAL

The setTotalPooledAvaxCap function under StackedAvax does set totalPooledAvax instead of totalPooledAvaxCap.

Code Location:

Listing 1: (Lines 5)

```
1     function setTotalPooledAvaxCap(uint newTotalPooledAvaxCap)
          external {
2         require(hasRole(ROLE_SET_TOTAL_POOLED_AVAX_CAP, msg.sender
              ), "ROLE_SET_TOTAL_POOLED_AVAX_CAP");
3
4         uint oldTotalPooledAvaxCap = totalPooledAvaxCap;
5         totalPooledAvax = newTotalPooledAvaxCap;
6
7         emit TotalPooledAvaxCapUpdated(oldTotalPooledAvaxCap,
              newTotalPooledAvaxCap);
8     }
```

Risk Level:

Likelihood - 5
Impact - 5

Recommendations:

This issue was caused by code completion or any sort of copy-pasting mistake. This could be prevented by having a test suite with full coverage even on simple functions.

Remediation Plan:

SOLVED: The issue was fixed in commit df3a83720e400346db06ae07487eb22486d0b197

3.2 (HAL-02) USAGE OF MEMORY INSTEAD OF STORAGE - CRITICAL

Description:

In NodeCoordinator the startRemovalCooldownPeriodForNode function modifies the removalCooldownEpoch state of the node. However, the loading of the object is done using the memory keyword, those making the changes of the removalCooldownEpoch variable not reflected on the object itself but a copy in memory.

```
>>> node.startRemovalCooldownPeriodForNode(random_address, {'from':accounts[0]})
Transaction sent: 0x413821cceb8c06304b1ee81a1409130eca26031ada35f54c6cd8ba91676da741
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 24
  NodeCoordinator.startRemovalCooldownPeriodForNode confirmed - Block: 28   Gas used: 26371 (0.22%)

<Transaction '0x413821cceb8c06304b1ee81a1409130eca26031ada35f54c6cd8ba91676da741'>
>>> node.nodes(random_address)
("0x66aB6D9362d4F35596279692F0251Db635165871", 1000, 0, 0)
>>>
```

Figure 1: The stored variables are not reflected since the values are only kept in memory

This means that the removalCooldownEpoch value will always be 0 causing other functions to miss-calculate the cooldown period or not execute at all. As an example, the removeNode will be never callable due to the following line:

Listing 2

```
1 require(node.removalCooldownEpoch != 0, "Removal cooldown not
      started");
```

On the same function, if the previous check was not present, the following statement would always validate, allowing the unlocking to take place since block.timestamp (currently at 13492020) would always be bigger than UNLOCK_PERIOD (1209600):

Listing 3

```
1 require(
2     block.timestamp - node.removalCooldownEpoch >= UNLOCK_PERIOD,
```

```
3     "Removal cooldown has not been completed"
4 );
```

Risk Level:

Likelihood - 5
Impact - 5

Recommendation:

Change the memory keyword to storage on the`startRemovalCooldownPeriodForNode` function.

Remediation Plan:

SOLVED: The issue was solved by changing the memory keyword to storage

3.3 (HAL-03) CONTRACT LOCKED - CRITICAL

The StackedAvax contract does always revert when calling mint, since the totalSupply().sub(address(this).balance) on _accrueReward of the _beforeTokenTransfer will always underflow. The funds sent in the mint call will always be present in the contract balance upon reaching this line, which makes the address(this).balance always greater than the minted tokens.

```
>>> t = avax.mint({'from':accounts[0],'value':100000000})
Transaction sent: 0xcb569d6de18a7b1e92e67fe9eb18e75249a7bc09c93b04230bdcf0c0b32af185
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 2
  StakedAvax.mint confirmed (SafeMath: subtraction overflow) - Block: 3   Gas used: 64214 (0.54%)

>>> t.call_trace()

Call trace for '0xcb569d6de18a7b1e92e67fe9eb18e75249a7bc09c93b04230bdcf0c0b32af185':
Initial call cost  [21064 gas]
StakedAvax.mint  0:211  [21090 / 43150 gas]
└── ERC20Upgradeable._mint  75:211  [59 / 22060 gas]
    └── StakedAvax._beforeTokenTransfer  92:211  [93 / 22001 gas]
        └── StakedAvax._distributeReward  118:211  [15 / 21908 gas]
            └── StakedAvax._accrueReward  122:211  [20889 / 21893 gas]
                ├── SafeMathUpgradeable.sub  135:149  [49 gas]
                ├── ERC20Upgradeable.totalSupply  161:166  [815 gas]
                └── SafeMathUpgradeable.sub  170:211  [140 gas]
>>>
```

Figure 2: Call trace showing that the mint function will always revert

This will prevent any tokens from being generated and the totalSupply will stay always zero. Furthermore, the contract expects that all send token will go through the fallback function. However, some methods exist to skip that, such as using a selfdestruct on a contract with funds. This could cause the address(this).balance to spike up and causing the same calculation errors. Take a look here:

https://docs.soliditylang.org/en/develop/security-considerations.html#sending-and-receiving-ether

Risk Level:

Likelihood - 5
Impact - 5

Remediation Plan:

SOLVED: The code was changed to have a different path that would never trigger the same issue

3.4 (HAL-04) CONTRACT MISSING OWNER - CRITICAL

The initialize method under NodeCoordinator is missing the __Ownable_init call to transfer the ownership to the actual caller, without this the owner will always be address(0), preventing anyone from actually owning the contract or calling any owner-only methods.

https://docs.soliditylang.org/en/develop/security-considerations.html#sending-and-receiving-ether

Risk Level:

Likelihood - 5
Impact - 5

Recommendation:

Add __Ownable_init under the initialize method of the NodeCoordinator.

Remediation Plan:

SOLVED: It was fixed under the 1aed81f46c3dfa49d8a63cff783a3d6391cba2fd commit.

3.5 (HAL-05) UNDERFLOW ON THE USAGE OF BLOCK.TIMESTAMP - MEDIUM

The timestamp of the block can be manipulated by the miner, and so should not be used for critical components of the contract. The `removeNode` function, under `NodeCoordinator`, does use the `block.timestamp` against the `removalCooldownEpoch` to determine if the `UNLOCK_PERIOD` period is passed.

However, the operation `block.timestamp - node.removalCooldownEpoch >= UNLOCK_PERIOD` could underflow if the faked `block.timestamp` is less than the stored `removalCooldownEpoch`, causing the subtraction operation to return a huge value always bigger than `UNLOCK_PERIOD` and resulting in the funds being unlocked.

Risk Level:

Likelihood - 3
Impact - 4

Recommendation:

It is recommended to change the `require` operation to a safer approach, such as the following one, or to remove the usage of `block.timestamp` completely.

Listing 4

```
require(
    UNLOCK_PERIOD + node.removalCooldownEpoch >= block.timestamp,
    "Removal cooldown has not been completed"
);
```

Remediation Plan:

SOLVED: The issue was solved by using SafeMath operations, those detecting overflows/underflows

3.6 (HAL-06) OUT-OF-BOUNDS ACCESS - HIGH

The `redeemOverdueShares` on `StakedAvax` code does not work as expected. During the loop, a `pop` is performed on the same list as the one iterating on. This will cause the `unlocksByUser[msg.sender].length` to be decremented (Not keeping the old value) and thus since the `-1` array index is moved on the popped item only half of the array will be claimed. In case the old length value was kept during the iteration, that would cause an out-of-bounds since the array will contain less item if they were popped.

The `_dropExpiredExchangeRateEntries` function under `StakedAvax`, on the last loop `for (uint i = 1; i < shiftCount; i = i.sub(1))`, the `sub(1)` should be `add(1)`, otherwise an out-of-bounds when `i=0` on the `length.sub(i)` would happen. Furthermore, the `_dropExpiredExchangeRateEntries` does not properly remove expired rates and leaves the last expiration on `shiftCount` index untouched, probably the last loop end index should be `shiftCount + 1`.

If no historical data is present on `_getExchangeRateByUnlockTimestamp` the internal `high` value will be `max(uint256)-1` causing `mid` to be half of it, aka `0x7fff...` and cause out-of-bounds

POC:

The Figure 3 is an example with 5 available unlocks, showing how after redemption, 2 of them are still on the list

The Figure 4 does show the error under the `_dropExpiredExchangeRateEntries` function

Furthermore, the end index should be `shiftCount + 1`, since the following example on Listing 5 showcases the missed element

```
>>> s.unlocksByUser(accounts[0], 0)
(1637892535, 1000)
>>> s.unlocksByUser(accounts[0], 1)
(1637892536, 1000)
>>> s.unlocksByUser(accounts[0], 2)
(1637892537, 1000)
>>> s.unlocksByUser(accounts[0], 3)
(1637892538, 1000)
>>> s.unlocksByUser(accounts[0], 4)
(1637892539, 1000)
>>> s.unlocksByUser(accounts[0], 5)
  File "<console>", line 1, in <module>
  File "brownie/network/contract.py", line 1712, in __call__
    return self.call(*args, block_identifier=block_identifier)
  File "brownie/network/contract.py", line 1517, in call
    raise VirtualMachineError(e) from None
VirtualMachineError: invalid opcode: invalid opcode
>>> s.redeemOverdueShares()
Transaction sent: 0xa4f29b1c34d748443c439dc9f0fb93762fd3946227e6c880154b7c5de16460c9
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 50
  StakedAvax.redeemOverdueShares confirmed - Block: 51   Gas used: 72549 (0.60%)

<Transaction '0xa4f29b1c34d748443c439dc9f0fb93762fd3946227e6c880154b7c5de16460c9'>
>>> s.unlocksByUser(accounts[0], 0)
(1637892539, 1000)
>>> s.unlocksByUser(accounts[0], 1)
(1637892538, 1000)
>>> s.unlocksByUser(accounts[0], 2)
  File "<console>", line 1, in <module>
  File "brownie/network/contract.py", line 1712, in __call__
    return self.call(*args, block_identifier=block_identifier)
  File "brownie/network/contract.py", line 1517, in call
    raise VirtualMachineError(e) from None
VirtualMachineError: invalid opcode: invalid opcode
>>>
```

Figure 3: Skipped elements due to modifying a list while iterating

```
>>> t = s.accrueRewards(1000)
Transaction sent: 0x83ff8e945452101ebc36485bcd2567c31c8f4dd1b985e93363a1cd302ca176c7
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 26
  StakedAvax.accrueRewards confirmed (Index out of range) - Block: 27   Gas used: 12000000 (100.00%)
```

Figure 4: Out of bounds on the _dropExpiredExchangeRateEntries

Listing 5

```
expirationThreshold = 5
historical = [ 1, 2, 3, 4, 5, 6, 7]

# After the while (shiftCount ...

shiftCount = 4

# After the for (uint i = 0; i < historical ...

historical = [5, 6, 7, 4, 5, 6, 7]

# After the last loop (changing sub to add!!)

historical = [5, 6, 7, 4]
```

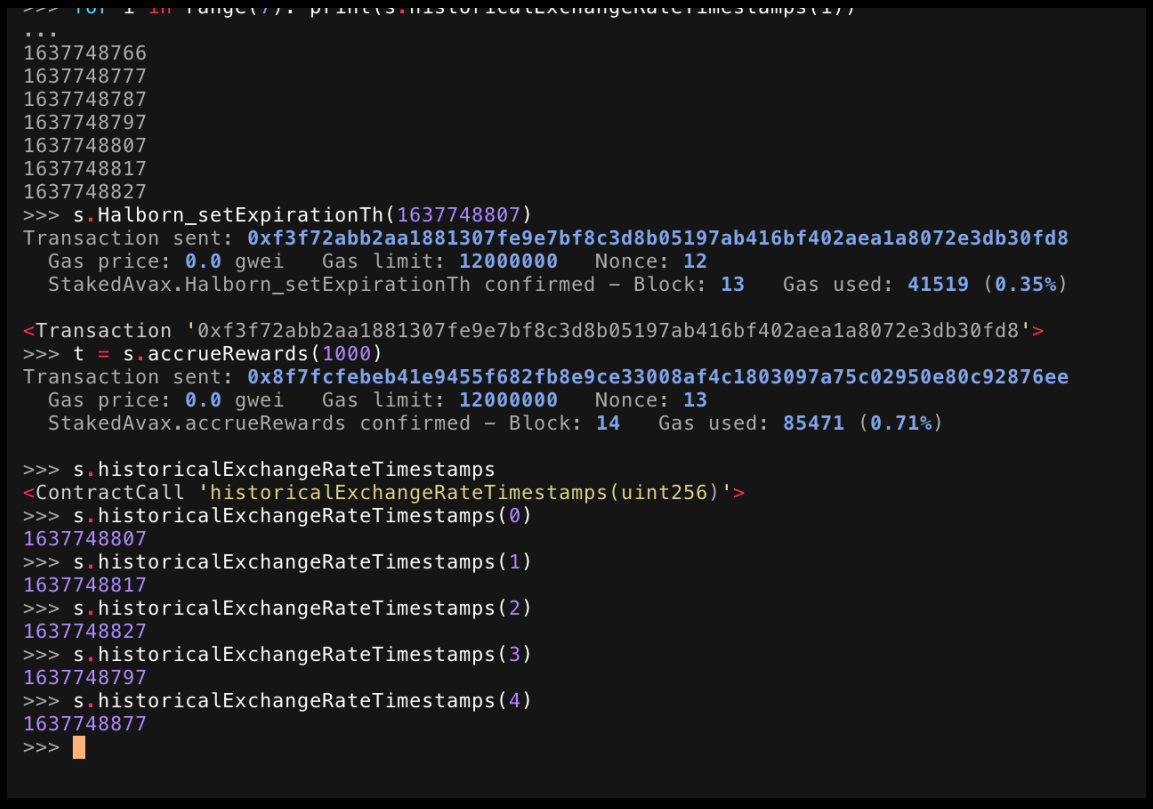


Figure 5: Invalid historical access list, the last element is skipped

If no historical data is present when calling the _getExchangeRateByUnlockTimestamp function, the code will report an ouf-of-bounds invalid instruction as seen in Figure 6



Figure 6: Invalid operation when no historical data is present

Recommendation:

The loop should check for the full length before looping, and never modify an iterated list/container.

Remediation Plan:

SOLVED: It was fixed under the 40bfd4a55114ffbb2f0794ae72ddf81962d3fd96, f53a0f7de26f7cc6faf1cdb1ea33ffbf0b3a272e, f53a0f7de26f7cc6faf1cdb1ea33ffbf0b3a272e and fefb8e5d79bbd860236ed86a030a580511f4b92e commits.

3.7 (HAL-07) CODE DUPLICATION - INFORMATIONAL

The sAVAXPerToken and tokensPerSAVAX function can be obtained by calling getSAVAXByWsAVAX and getWsAVAXBySAVAX with 1 ether as argument. If something can be done on the frontend or on other parts of the code do it there, don't duplicate code.

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to not duplicate code as this is prone to double update errors

Remediation Plan:

SOLVED: It was fixed under the bd0a46840427b72c1b5c39fbafafa96e92cf2bd5 commit.

MANUAL TESTING

4.1 NodeCoordinator

Adding less amount than the minimumQiStakeAmount results in a revert:

```
>>> random_address = '0x52bc44d5378309ee2abf1539bf71de1b7d7be3b5'
>>> node.addNode(random_address, 100)
Transaction sent: 0x3ddcf67a0d0d2dba2b6f40c8c9a37dc7f3320e041a9e14b7ebfbbc3fb1741a7b
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 19
  NodeCoordinator.addNode confirmed (QI amount is too low) - Block: 20   Gas used: 23660 (0.20%)

<Transaction '0x3ddcf67a0d0d2dba2b6f40c8c9a37dc7f3320e041a9e14b7ebfbbc3fb1741a7b'>
>>>
```

The balance needs to be approved on the token for the node coordinator:

```
>>> node.addNode(random_address, 1000)
Transaction sent: 0x72c60138781bd24222eb7beb313d9da746aae342aa314bfb78636898e602d0a4
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 21
  NodeCoordinator.addNode confirmed (ERC20: transfer amount exceeds allowance) - Block: 22   Gas used: 58104 (0.48%)

<Transaction '0x72c60138781bd24222eb7beb313d9da746aae342aa314bfb78636898e602d0a4'>
>>> token.approve(node, 1000)
Transaction sent: 0xe285622766464707789b80a990f4e456d0ce248195cef00b675115a3d0d20539
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 22
  HalbornToken.approve confirmed - Block: 23   Gas used: 44046 (0.37%)

<Transaction '0xe285622766464707789b80a990f4e456d0ce248195cef00b675115a3d0d20539'>
>>> node.addNode(random_address, 1000)
Transaction sent: 0xc3435eb65e6bb2d7ecc5eb3ab4fdfa615721ee108bef429d88e8bdf8e7faab0c
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 23
  NodeCoordinator.addNode confirmed - Block: 24   Gas used: 94294 (0.79%)

<Transaction '0xc3435eb65e6bb2d7ecc5eb3ab4fdfa615721ee108bef429d88e8bdf8e7faab0c'>
>>>
```

Calling the startRemovalCooldownPeriodForNode from a different account than the register:

```
>>> node.startRemovalCooldownPeriodForNode(random_address, {'from':accounts[1]})
Transaction sent: 0x8b333eea3722454070c5b2c2d23ab8195522d190e9eea3608c79f40d1c508d1d
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 2
  NodeCoordinator.startRemovalCooldownPeriodForNode confirmed (Caller must be node registerer)

<Transaction '0x8b333eea3722454070c5b2c2d23ab8195522d190e9eea3608c79f40d1c508d1d'>
>>>
```

The state is not reflected on the node after setting the removal cooldown period:

```
>>> node.startRemovalCooldownPeriodForNode(random_address, {'from':accounts[0]})
Transaction sent: 0x413821cceb8c06304b1ee81a1409130eca26031ada35f54c6cd8ba91676da741
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 24
  NodeCoordinator.startRemovalCooldownPeriodForNode confirmed - Block: 28   Gas used: 26371 (0.22%)

<Transaction '0x413821cceb8c06304b1ee81a1409130eca26031ada35f54c6cd8ba91676da741'>
>>> node.nodes(random_address)
("0x66aB6D9362d4F35596279692F0251Db635165871", 1000, 0, 0)
>>>
```

Remove node is not possible since the node storage value is not updated:

```
>>> node.startRemovalCooldownPeriodForNode(random_address)
Transaction sent: 0x4e6bc03d46650bf07ce00fc2b11933939f74a51f2e899f6db9c3b5e14682b92f
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 8
  NodeCoordinator.startRemovalCooldownPeriodForNode confirmed - Block: 9   Gas used: 26371 (0.22%)

<Transaction '0x4e6bc03d46650bf07ce00fc2b11933939f74a51f2e899f6db9c3b5e14682b92f'>
>>> node.nodes(random_address)
("0x66aB6D9362d4F35596279692F0251Db635165871", 1000, 0, 0)
>>> node.removeNode(random_address)
Transaction sent: 0x70442b6e7b5800dd7cf8ec82e25530a6f7a3151ea2a5061137a1d7e1817b9156
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 9
  NodeCoordinator.removeNode confirmed (Removal cooldown not started) - Block: 10   Gas used: 25430 (0.21%)

<Transaction '0x70442b6e7b5800dd7cf8ec82e25530a6f7a3151ea2a5061137a1d7e1817b9156'>
>>>
```

Successfully removing the node if the storage was updated:

```
>>> chain.sleep(1000000000)
>>> node.removeNode(random_address)
Transaction sent: 0x06f0445f183ac028dc0c90a9cddb9ed70bf20f91ef4538a87caad110cf10006d
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 7
  NodeCoordinator.removeNode confirmed - Block: 8   Gas used: 30627 (0.26%)

<Transaction '0x06f0445f183ac028dc0c90a9cddb9ed70bf20f91ef4538a87caad110cf10006d'>
>>> node.removeNode(random_address)
Transaction sent: 0xccc8790436326b03870c1b8767c3eb98fbbb945c8db94cdc8c59551b85e533d5
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 8
  NodeCoordinator.removeNode confirmed (Caller must be node registerer) - Block: 9   Gas used: 25404 (0.21%)

<Transaction '0xccc8790436326b03870c1b8767c3eb98fbbb945c8db94cdc8c59551b85e533d5'>
>>> node.removeNode(random_address)
Transaction sent: 0x70442b6e7b5800dd7cf8ec82e25530a6f7a3151ea2a5061137a1d7e1817b9156
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 9
  NodeCoordinator.removeNode confirmed (Caller must be node registerer) - Block: 10   Gas used: 25404 (0.21%)

<Transaction '0x70442b6e7b5800dd7cf8ec82e25530a6f7a3151ea2a5061137a1d7e1817b9156'>
>>> node.nodes(random_address)
("0x0000000000000000000000000000000000000000", 0, 0, 0)
>>>
```

Multiples unlocks and checking the cooldown requests:

> Will all my unlock requests be fulfilled

```
>>> avax.unlock(500)
Transaction sent: 0x51e58c5ba2093fde7b63d8d8c9c6fa173a3d0ef39c29941a043159a5e86655b0
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 5
  StakedAvax.unlock confirmed - Block: 6   Gas used: 72758 (0.61%)

<Transaction '0x51e58c5ba2093fde7b63d8d8c9c6fa173a3d0ef39c29941a043159a5e86655b0'>
>>> avax.unlock(500)
Transaction sent: 0x068bff5844b06f50bcb37c4b9fbf07576a3f4316e9545b46fe2ed14a8bd1a7a1
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 6
  StakedAvax.unlock confirmed - Block: 7   Gas used: 72758 (0.61%)

<Transaction '0x068bff5844b06f50bcb37c4b9fbf07576a3f4316e9545b46fe2ed14a8bd1a7a1'>
>>> avax.userCooldownState(accounts[0], 1)
(1635934059, 1)
>>> avax.userCooldownState(accounts[0], 2)
(1635934149, 500)
>>> avax.userCooldownState(accounts[0], 3)
(1635934151, 500)
>>> avax.userCooldownState(accounts[0], 4)
(0, 0)
>>> avax.userCooldownState(accounts[0], 0)
(1635934057, 1)
>>>
```

Canceling unlocks:

```
<Transaction '0xa9228848e75af071c6e87f3000f843ccf8c4bfe5aef778df7c425160264cc21a'>
>>> avax.cancelUnlock(2)
Transaction sent: 0x89edb503a944e26985ce08d1f6b1e86f78d282b6104eb54bf5257f7897c59516
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 11
  StakedAvax.cancelUnlock confirmed - Block: 12   Gas used: 20829 (0.17%)

<Transaction '0x89edb503a944e26985ce08d1f6b1e86f78d282b6104eb54bf5257f7897c59516'>
>>> avax.cancelUnlock(3)
Transaction sent: 0x251a397760a383510cf310161ed5e1370a3c7e8be2dc703972ce13b685589ac8
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 12
  StakedAvax.cancelUnlock confirmed - Block: 13   Gas used: 20829 (0.17%)

<Transaction '0x251a397760a383510cf310161ed5e1370a3c7e8be2dc703972ce13b685589ac8'>
>>> avax.cancelUnlock(4)
Transaction sent: 0x76ab437120b5fc6a3f2170c7f1d7fe94a4560a751e9b6e88e4212b1be59d3126
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 13
  StakedAvax.cancelUnlock confirmed (Invalid index) - Block: 14   Gas used: 28357 (0.24%)

<Transaction '0x76ab437120b5fc6a3f2170c7f1d7fe94a4560a751e9b6e88e4212b1be59d3126'>
>>> avax.cancelUnlock(5)
Transaction sent: 0x95b9802b15e4389e796ad180f0592995dda044d64d40425710605ae08f96f60b
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 14
  StakedAvax.cancelUnlock confirmed (Invalid index) - Block: 15   Gas used: 28357 (0.24%)

<Transaction '0x95b9802b15e4389e796ad180f0592995dda044d64d40425710605ae08f96f60b'>
>>> avax.userCooldownState(accounts[0], 0)
(0, 0)
>>> avax.userCooldownState(accounts[0], 1)
(0, 0)
>>> avax.userCooldownState(accounts[0], 2)
(0, 0)
>>> avax.userCooldownState(accounts[0], 3)
(0, 0)
>>>
```

Redeem period by default is zero, it should be on the initializer

```
>>> avax.redeem(4)
Transaction sent: 0x6911f8d9fc446c36cb75e3b327904cf2944894cc7b00b41518c045e7e2b7bdc9
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 20
  StakedAvax.redeem confirmed (Redeem window has closed) - Block: 21   Gas used: 32718 (0.27%)

<Transaction '0x6911f8d9fc446c36cb75e3b327904cf2944894cc7b00b41518c045e7e2b7bdc9'>
>>> avax.redeemPeriod
<ContractCall 'redeemPeriod()'>
>>> avax.redeemPeriod()
0
>>>
```

THANK YOU FOR CHOOSING

// HALBORN