Swell Network

Swell Network Contract Review

Version: 2.1

Contents

Introduction 2
- Disclaimer . . . 2
- Document Structure . . . 2
- Overview . . . 2

Security Assessment Summary 3
- Findings Summary . . . 3

Detailed Findings 4

Summary of Findings 5
- Bots can set `ethReserves` and `swETHToETHRateFixed` to any value . . . 6
- Reentrancy protection is recommended for staking vault . . . 8
- Updating an operator's `controllingAddress` allows the previous operator to maintain control . . . 9
- Initial reprice can set `ethReserves` and `swETHToETHRateFixed` to incorrect values . . . 10
- No update to ETH reserve bookkeeping when ETH is deposited . . . 11
- Use of `transfer()` and `transferFrom()` . . . 12
- Insufficiently bounded loop when repricing `swETH` . . . 13
- `getNextValidatorDetails()` may return validators with zero addresses . . . 14
- Upgradeable contracts must disable initializers in the implementation contracts . . . 15
- Changing `SWETHv3` address should emit an event . . . 16
- Tokens returned from Balancer are handled with unnecessary flexibility . . . 17
- Permission to mint `swETH` could remain with `swNFT` owner . . . 18
- Inconsistent migration variable updates in `SWNFTv3` . . . 19
- Necessary off chain systems should be in place . . . 20
- Miscellaneous general comments . . . 21
- Miscellaneous gas optimisations . . . 22

A Test Suite 24

B Vulnerability Severity Classification 26

Introduction

Sigma Prime was commercially engaged to perform a time-boxed security review of the Swell Network smart contracts. The review focused solely on the security aspects of the Solidity implementation of the contract, though general recommendations and informational comments are also provided.

Disclaimer

Sigma Prime makes all effort but holds no responsibility for the findings of this security review. Sigma Prime does not provide any guarantees relating to the function of the smart contract. Sigma Prime makes no judgements on, or provides any security review, regarding the underlying business model or the individuals involved in the project.

Document Structure

The first section provides an overview of the functionality of the Swell Network smart contracts contained within the scope of the security review. A summary followed by a detailed review of the discovered vulnerabilities is then given which assigns each vulnerability a severity rating (see Vulnerability Severity Classification), an *open/closed/resolved* status and a recommendation. Additionally, findings which do not have direct security implications (but are potentially of interest) are marked as *informational*.

Outputs of automated testing that were developed during this assessment are also included for reference (in the Appendix: Test Suite).

The appendix provides additional documentation, including the severity matrix used to classify vulnerabilities within the Swell Network smart contracts.

Overview

Swell is an Ethereum liquid staking protocol. It provides users with a non custodial means of liquid staking via a transferable ERC-20 token called `swETH`. The protocol maintains a list of operators who can queue validators to be deployed once the required 32 ETH is available.

This review covers migration from the previous version of the protocol, which used NFTs to track positions. Part of the migration consists of burning the previous NFTs and converting them to the new `swETH`.

Security Assessment Summary

This review was conducted on the files hosted on the Sigma Prime repository and were assessed at commits ab2f6af and 7ba12d7.

Retesting was performed on commits 5144aff and dd969d0 respectively.

As confirmed with the development team, there were two versions of the `swETH` contract provided for review. `swell-v3-sigp/src/v3_swETH.sol` is a mock contract which is provided for the purpose of testing `swell-v3-sigp/src/v3_swNFT.sol`. It will not be deployed as part of the final project. `swell-v3-labrys-sigp/contracts/implementations/swETH.sol` is the version of the contract which will be deployed, but it was missing some functionality at the time of review.

Note: the OpenZeppelin libraries and dependencies were excluded from the scope of this assessment.

The manual code review section of the report is focused on identifying any and all issues/vulnerabilities associated with the business logic implementation of the contracts. This includes their internal interactions, intended functionality and correct implementation with respect to the underlying functionality of the Ethereum Virtual Machine (for example, verifying correct storage/memory layout). Additionally, the manual review process focused on all known Solidity anti-patterns and attack vectors. These include, but are not limited to, the following vectors: re-entrancy, front-running, integer overflow/underflow and correct visibility specifiers. For a more thorough, but non-exhaustive list of examined vectors, see [**?**, **?**].

To support this review, the testing team used the following automated testing tools:

- Mythril: `https://github.com/ConsenSys/mythril`
- Slither: `https://github.com/trailofbits/slither`
- c4udit: `https://github.com/byterocket/c4udit`
- Surya: `https://github.com/ConsenSys/surya`
- solstat: `https://github.com/0xKitsune/solstat`

Output for these automated tools is available upon request.

Note: in this review, Slither was unable to process some contracts and so those contracts were not assessed using this tool.

Findings Summary

The testing team identified a total of 16 issues during this assessment. Categorised by their severity:

- Medium: 3 issues.
- Low: 5 issues.
- Informational: 8 issues.

Detailed Findings

This section provides a detailed description of the vulnerabilities identified within the Swell Network smart contracts. Each vulnerability has a severity classification which is determined from the likelihood and impact of each issue by the matrix given in the Appendix: Vulnerability Severity Classification.

A number of additional properties of the contracts, including gas optimisations, are also described in this section and are labelled as "informational".

Each vulnerability is also assigned a **status**:

- ***Open:*** the issue has not been addressed by the project team.
- ***Resolved:*** the issue was acknowledged by the project team and updates to the affected contract(s) have been made to mitigate the related risk.
- ***Closed:*** the issue was acknowledged by the project team but no further actions have been taken.

Summary of Findings

ID	Description	Severity	Status
SWL2-01	Bots can set `ethReserves` and `swETHToETHRateFixed` to any value	Medium	Resolved
SWL2-02	Reentrancy protection is recommended for staking vault	Medium	Resolved
SWL2-03	Updating an operator's `controllingAddress` allows the previous operator to maintain control	Medium	Resolved
SWL2-04	Initial reprice can set `ethReserves` and `swETHToETHRateFixed` to incorrect values	Low	Closed
SWL2-05	No update to ETH reserve bookkeeping when ETH is deposited	Low	Resolved
SWL2-06	Use of `transfer()` and `transferFrom()`	Low	Resolved
SWL2-07	Insufficiently bounded loop when repricing `swETH`	Low	Closed
SWL2-08	`getNextValidatorDetails()` may return validators with zero addresses	Low	Resolved
SWL2-09	Upgradeable contracts must disable initializers in the implementation contracts	Informational	Resolved
SWL2-10	Changing `SWETHv3` address should emit an event	Informational	Resolved
SWL2-11	Tokens returned from Balancer are handled with unnecessary flexibility	Informational	Resolved
SWL2-12	Permission to mint `swETH` could remain with `swNFT` owner	Informational	Resolved
SWL2-13	Inconsistent migration variable updates in `SWNFTv3`	Informational	Resolved
SWL2-14	Necessary off chain systems should be in place	Informational	Resolved
SWL2-15	Miscellaneous general comments	Informational	Resolved
SWL2-16	Miscellaneous gas optimisations	Informational	Resolved

SWL2-01	Bots can set `ethReserves` and `swETHToETHRateFixed` to any value		
Asset	`swETH.sol`, `stakeAndVault.sol`		
Status	**Resolved:** See Resolution		
Rating	Severity: Medium	Impact: Medium	Likelihood: Medium

Description

Bots are trusted to update not only how many ETH rewards have been generated, but also to update the protocol's entire supply of ETH. This value is used to calculate the `swETH` to ETH conversion rate. It is therefore possible for a bot to set the value of `swETH` to a very high value. Any future withdraw functionality could thus be made to overpay.

`swETH.reprice()` takes a parameter `_preRewardETHReserves` which it trusts entirely. It uses `_preRewardETHReserves` to calculate the amount of new `swETH` to mint for rewards, but it also uses it to reset the contract's essential ETH bookkeeping values:

```
ethReserves = _preRewardETHReserves + _newETHRewards;

swETHToETHRateFixed = wrap(ethReserves).div(wrap(totalSupply())).unwrap();
```

If `_preRewardETHReserves` were set to a number, say a hundred times higher than the true reserve amount, then `swETH` would report a value approximately one hundred times too high when the following functions are called:

- `swETHToETHRate()`
- `ethToSwETHRate()`
- `getRate()`
- `ethReserves()` (Solidity automatically generated getter function)

Other contracts might rely on these values, which are too easily manipulated. `stakeAndVault` is one such contract, for example.

Furthermore, in future versions, if `swETH` has a `withdraw()` function for converting `swETH` back to ETH, it might reasonably rely on the value of `ethReserves`.

It would also be possible to perform a similar attack by calling `reprice()` with a very large value for the parameter `_newETHRewards`.

This issue is partially mitigated by security restrictions of the role `SwellLib.BOT`, which is the only role able to call `swETH.reprice()`.

Recommendations

Consider implementing bookkeeping within the `swETH` contract to maintain the value of `ethReserves` (See also SWL2-05). Implement a check on the value of the parameter `_newETHRewards` to prevent it from being disproportionately higher than `ethReserves`, and do not allow `reprice` to be called many times in a small time period.

Resolution

The issue has been resolved in commits d52f90e and 3b8909b. The solution implements a check to ensure a certain quantity of time has elapsed since the previous call to `reprice()` to prevent multiple calls in a small time period.

Additionally, checks have been added to ensure an upper and lower bound on the new ratio of `swETH` to ETH as a percentage of the old ratio to prevent drastic and possibly manipulative changes.

There is also a similar check on the new parameter, `swETHTotalSupply`, to ensure that it does not differ too greatly from the value of `totalSupply()`. This prevents and reprice which would preserve the `swETH` to ETH ratio whilst greatly increasing or decreasing the total amounts of ETH and `swETH`.

SWL2-02	Reentrancy protection is recommended for staking vault		
Asset	`SwellStakeVaultHelper.sol`		
Status	**Resolved:** See Resolution		
Rating	Severity: Medium	Impact: High	Likelihood: Low

Description

The staking vault contract makes external calls, which would allow a reentrant to the contract to steal the assets of the first user.

It is not within the scope of this review to fully assess whether any of the external calls do contain a reentrancy risk but, as they are not controlled by Swell, it is best to assume that some threat potential is present.

The main reentrancy threat occurs during the call to `balancerVault.joinPool()` on line [**132**]. If the call to this function passes control flow to an exploit contract after `BPT` contracts have been transferred to the vault, that contract would be able to call `SwellStakeVaultHelper.stakeAndVault()` with a very small amount of ETH. During this reentrancy call, the `BPT` balance of the vault on line [**150**] would include the `BPT` tokens of the first depositor. These would then be deposited into aura under the exploit contract's address. The original depositor's call to `SwellStakeVaultHelper.stakeAndVault()` would then resolve, with `amount` on line [**150**] being zero.

Recommendations

Add a reentrancy guard to `SwellStakeVaultHelper.stakeAndVault()`.

Resolution

The `nonReentrant` modifier from OpenZeppelin ReentrancyGuard has been added to the function `SwellStakeVaultHelper.stakeAndVault()` as recommended in commit 8acccd8.

SWL2-03	Updating an operator's `controllingAddress` allows the previous operator to maintain control		
Asset	`NodeOperatorRegistry.sol`		
Status	**Resolved:** See Resolution		
Rating	Severity: Medium	Impact: Medium	Likelihood: Medium

Description

Even after updating the `controllingAddress` using the function `updateOperatorControllingAddress()`, the old `controllingAddress` can still control the Operator and call the function `addNewValidatorDetails()`.

When the `controllingAddress` is updated using `updateOperatorControllingAddress()`, the old `controllingAddress` is still linked to the `operatorId`, this is because the function do not clear the `operatorId` from the old `controllingAddress`. As a result, `getOperatorIdForAddress(oldAddress) == getOperatorIdForAddress(newAddress) != 0` and so the old address can still control the `Operator` that it was controlling.

Added to that, it would be impossible in the future to add `Operator` with an old `controllingAddress` because of line [**293-295**].

```
282 function addOperator(
      string calldata _name,
284   address _operatorAddress,
      address _rewardAddress
286 )
      external
288   override
      checkRole(SwellLib.PLATFORM_ADMIN)
290   checkZeroAddress(_operatorAddress)
      checkZeroAddress(_rewardAddress)
292 {
      if (getOperatorIdForAddress[_operatorAddress] != 0) {
294     revert OperatorAlreadyExists(_operatorAddress);
      }
```

Recommendations

Clear the old `controllingAddress` after updating it, by doing `delete getOperatorIdForAddress[_operatorAddress]`.

Resolution

The recommendation has been implemented in commit a90c338.

SWL2-04	Initial reprice can set `ethReserves` and `swETHToETHRateFixed` to incorrect values		
Asset	`swETH.sol`		
Status	**Closed:** See Resolution		
Rating	Severity: Low	Impact: Medium	Likelihood: Low

Description

The checks implemented to address SWL2-01 have an exception for the very first repricing.

```
// Ensure that the reprice differences are within expected ranges, only if the reprice method has been called before
if (lastRepriceUNIX != 0) {
```

Whilst this exception enables the first repricing event, it does enable a possible denial of service attack if the first reprice event is executed with awkward values for the first two parameters of `reprice()` : `_preRewardETHReserves` and `_newETHRewards` .

Recommendations

This issue may be best addressed as a part of deployment by carefully managing the first address to be granted the role `SwellLib.BOT` and ensuring that this address executes a trusted initial call to `reprice()` .

Resolution

The development team acknowledged the issue and assured that the first call to `reprice()` would be carefully controlled as a part of the migration process and that the variable `lastRepriceUNIX` would therefore have a non zero value before any bot was granted the `SwellLib.BOT` role.

SWL2-05	No update to ETH reserve bookkeeping when ETH is deposited		
Asset	`swETH.sol`		
Status	**Resolved:** See Resolution		
Rating	Severity: Low	Impact: Low	Likelihood: Medium

Description

When ETH is deposited into the `swETH` contract, new `swETH` tokens are minted, but the value of the ETH reserves in the protocol are not updated. This could lead to possible accounting errors and potential resultant loss.

`swETH.deposit()` calls `_mint` on line [**134**] to mint new `swETH` to the depositor at the current conversion rate as determined by `swETHToETHRateFixed`. However, it makes no adjustment to the state variable `ethReserves`.

This issue is of limited impact as the ratio of `swETH` to ETH in the protocol has not changed, and so deposits or potential withdrawals would still be at the correct rate. However, any system that relies on the variable `ethReserves`, perhaps comparing it to `totalSupply()`, would operate under incorrect information and this could potentially lead to loss.

Recommendations

Update `ethReserves` when `swETH.deposit()` is called. Alternatively, consider making the state variable `ethReserves` private.

Resolution

The issue has been resolved in commit d52f90e. The variable `ethReserves` has been renamed to `lastRepriceETHReserves` to better indicate its bookkeeping role and the implied limitations. A complimentary variable `totalETHDeposited` has been added, which is incremented during calls to `deposit()`.

Additionally, `swETHTotalSupply` has been added as an argument to `reprice()` to be used in place of `totalSupply()` as this value could be out of sync with the values used by the bot.

SWL2-06	Use of `transfer()` and `transferFrom()`		
Asset	`NodeOperatorRegistry.sol`, `DepositManager.sol`		
Status	**Resolved:** See Resolution		
Rating	Severity: Low	Impact: Low	Likelihood: Low

Description

`NodeOperatorRegistry` uses `transfer()` and `transferFrom()` functions.

Some ERC20 tokens don't return a value in their `transfer` and `transferFrom()` functions. If there are no checks performed on the returned value (or if there is a value returned), the function may not revert on failed transfers and this may lead to further errors.

Additionally, some tokens do not return a boolean value. Since `IERC20` expects a `bool` to be returned it will attempt to decode the `returndata`. If the `returndata` is empty the decoding will fail and the transaction will revert.

`SafeERC20` functions `safeTransfer()` and `safeTransferFrom()` automatically check and assert the boolean return value of a transfer function.

Recommendations

Use SafeERC20 functions such as `safeTransfer()` and `safeTransferFrom()` to ensure that the return value of the transfer call is checked and handled properly, if there is a return value.

Resolution

The recommendation has been implemented in commit a90c338.

SWL2-07	Insufficiently bounded loop when repricing `swETH`		
Asset	`swETH.sol`		
Status	**Closed:** See Resolution		
Rating	Severity: Low	Impact: Low	Likelihood: Medium

Description

There is a loop in the function `reprice()` that does not have sufficient bounds check and may cause an out of gas revert.

When `swETH.reprice()` is called, it loops through all registered validators in `nodeOperatorRegistry` and mints rewards to each one. As the number of validators grows, the gas fees for this loop increase proportionally. After a certain number of validators have been added, the function `reprice()` will require more gas to execute than exists in the block gas limit for Ethereum, at this stage it will no longer be possible to call `reprice()`.

Recommendations

Consider restructuring the logic of the rewards part of the repricing operation to save gas. For example, something similar to the MasterChef staking rewards algorithm would allow very gas efficient updates during a call to `reprice()`. Individual operators could then periodically call a claim function for their `swETH` rewards.

An alternate option is to implement a hard cap on the number of operators that can be added to the `NodeOperatorRegistry`. This requires determining the number of nodes that can be iterated overly safely within the block gas limit.

Resolution

This issue has been acknowledged by Swell team. The development team has clarified that the number of operators is likely to be less than 30 operators in short to medium term. The development team will ensure that they do not add more operators than can be iterated within the block gas limit.

SWL2-08	getNextValidatorDetails() may return validators with zero addresses		
Asset	NodeOperatorRegistry.sol		
Status	**Resolved:** See Resolution		
Rating	Severity: Low	Impact: Low	Likelihood: Low

Description

If the requested number of validators `_numNewValidators` when calling `getNextValidatorDetails()` is bigger than the actual number of validators, this function will return validators with zeros addresses. It will return available validators' details at the start of the returned array. After that, all entries will be the zero address.

If the function `getNextValidatorDetails()` is called by other contracts, this could potentially cause issues as this function might return unusable data.

Recommendations

Consider returning the number of validators actually found as another return value. Alternatively, make this behaviour clear in the function comment of `getNextValidatorDetails()` and ensure that anything that calls `getNextValidatorDetails()` accounts for this behaviour and does not expect all return values to be usable validators.

Resolution

The recommendation has been implemented in commit 532e9ca. The `getNextValidatorDetails()` now also returns the number of validators found as a second return value.

SWL2-09	Upgradeable contracts must disable initializers in the implementation contracts
Asset	`AccessControlManager.sol`, `DepositManager.sol`, `NodeOperatorRegistry.sol`, `swETH.sol`
Status	**Resolved:** See Resolution
Rating	Informational

Description

Quoting OpenZeppelin:

> Avoid leaving a contract uninitialized.
>
> An uninitialized contract can be taken over by an attacker. This applies to both a proxy and its implementation contract, which may impact the proxy. To prevent the implementation contract from being used, you should invoke the `_disableInitializers` function in the constructor to automatically lock it when it is deployed.

Recommendations

Add the following code to the affected contracts:

```
constructor() {
    _disableInitializers();
}
```

Resolution

The recommendation has been implemented in commit e5fcdbb. The function `_disableInitializers()` has been added each constructor for the effected contracts.

SWL2-10	Changing `SWETHv3` address should emit an event
Asset	`v3_swNFT.sol`
Status	**Resolved:** See Resolution
Rating	Informational

Description

In `v3_swNFT.sol`, the `setSWETHv3` function allows an administrator to modify the address of the associated `SWETHv3` contract, but fails to emit an event when this occurs. Given the importance of the `SWETHv3` contract, an event should be emitted. This improves the composability of the protocol and makes it easier to review and test.

Recommendations

Define a new event, `swETHAddressChanged(address prev, address curr)`, and emit it in `v3_swNFT::setSWETHv3`.

Resolution

The recommendation has been implemented in commit 1e7b0b0. The function `setSWETHv3()` has been updated to emit the event `swETHAddressChanged()` with two address arguments `prev` and `curr`.

SWL2-11	Tokens returned from Balancer are handled with unnecessary flexibility
Asset	`SwellStakeVaultHelper.sol`
Status	**Resolved:** See Resolution
Rating	Informational

Description

`balancerVault.getPoolTokens()` is called on line [111] of `SwellStakeVaultHelper` and it returns an array of tokens. Logically, the code would not revert if the returned array contained only one token, or if `swETH` or `wETH` were listed more than once.

This is admittedly unlikely to occur. Nevertheless, the extra flexibility is not desirable from a security perspective.

Recommendations

Consider checking that `tokensFromPool.length` is 2 and then hardcode this value in place of `tokensFromPool.length` throughout `depositBalancer()`.

Resolution

The recommendation has been implemented in commit 8acccd8.

SWL2-12	Permission to mint `swETH` could remain with `swNFT` owner
Asset	`v3_swNFT.sol`
Status	**Resolved:** See Resolution
Rating	Informational

Description

After migration is complete, the `SWNFTv3` contract will still retain the ability to call `swETH.migrateMint()`.

Whilst the `owner` of `SWNFTv3` will be a trusted role, it may nevertheless be more secure to remove this ability once the migration is complete.

Recommendations

The ability can be removed on the `SWNFTv3` side by calling `SWNFTv3.renounceOwnership()`, which would remove the `owner` entirely. Alternatively, it may be desirable to deactivate the `swETH.migrateMint()` function once migration is complete.

Resolution

The development team has resolved this issue in commit 1e7b0b0 through addition of the modifier `hasIncompleteMigration`. The modifier deactivates the function `migrateBatch()` once the migration is complete.

SWL2-13	Inconsistent migration variable updates in `SWNFTv3`
Asset	`v3_swNFT.sol`
Status	**Resolved:** See Resolution
Rating	Informational

Description

The `SWNFTv3` contract contract maintains two public variables that track the status of the migration process. These variables are not consistently updated.

The function `migrateSingle()` does not increment `totalMigrated` or `lastMigratedId`, both of which are updated when migrations occur in `migrateBatch()`.

The function `setSWETHv3()` resets the value of `totalMigrated` to zero, however the value of `lastMigratedId` is not reset.

Recommendations

Review the observations above and, if desired, update the mentioned variables in the appropriate places. Note that `migrateBatch()` calls `migrateSingle()` and so incrementing `totalMigrated` in both would cause double counting.

Resolution

The development team has resolved this issue in commit 1e7b0b0 by changing the visibility of the `migrateSingle()` function to `private` and updating the migration variable inside this function.

SWL2-14	Necessary off chain systems should be in place
Asset	`DepositManager.sol`
Status	**Resolved:** See Resolution
Rating	Informational

Description

The check on line [**83-85**] in `setupValidators()` protects against a validator from front-running the deposit to the `DepositContract`. To have this effect, however, an off chain system of checks is required.

Front-running attacks by a validator can be used to make a 1 ETH deposit directly to the `DepositContract` with different withdrawal credentials. The first withdrawal credentials with a valid signature will be used. Hence, the withdrawal address in `setupValidators()` may not be used.

Protection against this attack occurs by passing a `_depositDataRoot` which is the root of all validators deposit data. This issue is raised as an informational note that it is important to ensure none of the public keys supplied to `setupValidators()` exist in the `_depositDataRoot` otherwise the attack may still occur. These checks are required to be performed off-chain.

Recommendations

Ensure the required off chain checks are in place.

Consider adding further clarification in the comments at the start of `setupValidators()` function concerning these checks.

Resolution

A more detailed natspec comment “`@param _depositDataRoot The deposit contracts deposit root which MUST match the current beacon deposit contract deposit data root otherwise the contract will revert due to the risk of the front-running vulnerability.`“ has been added to the interface `IDepositManager` in commit 5144aff.

SWL2-15	Miscellaneous general comments
Asset	`contracts/*`
Status	**Resolved:** See Resolution
Rating	Informational

Description

This section details miscellaneous findings discovered by the testing team that do not have direct security implications:

1. **Magic numbers** The codebase contains various magic numbers throughout, namely:
   - Ether deposit quantities in `DepositManager.sol` on line [**87**], line [**105**], and line [**108**]
   - Prefix for Ether withdrawals to Execution Layer in `DepositManager.sol` on line [**128**]
   - Array length during allocation in `DepositManager.sol` on line [**125**]
   - Address of the Ether deposit contract in `DepositManager.sol` on line [**54**]
2. **Typographic errors** There are some typos within the codebase, such as:
   - Comment in `SwellStakeAndVaultHelper.sol` on line [**51**]
   - `NodeOperatorRegistry.sol` on line [**103**] "operator's" should be "operators".
   - `NodeOperatorRegistry.sol` on line [**297**] "operatorId's" should be "operatorIds".
   - `NodeOperatorRegistry.sol` on line [**449**] "it's" should be "its".
3. **Function underutilised** Consider using `_encodeOperatorIdAndKeyIndex()` on line [**215**] of `NodeOperatorRegistry.sol` as the code functionality is the same.
4. **Duplicated checks**
   - In `v3_swNFTv3.harvestETH()`, the `hasETHRecipient()` test is duplicated in `Harvester.harvestETH()`
   - In `v3_swNFTv3.rescueERC20()`, the `hasERC20Recipient()` test is duplicated in `Harvester.rescueERC20()`
5. **Missing comment** `INodeOperatorRegistry.sol` line [**32**] is missing a comment for the last parameter.
6. **No need for a payable receive() function which reverts** `NodeOperatorRegistry` line [**68**] implements a `receive()` function which reverts. Note that, if there are no `payable` functions in a contract, Solidity will revert if any call to the contract has `value`. Adding a payable function just to revert is likely to increase the deployable size of the code and may even make the contract as a whole more expensive to call.

Recommendations

Ensure that the comments are understood and acknowledged, and consider implementing the suggestions above.

Resolution

The comments above have been acknowledged by the development team, and relevant changes actioned in commit 1e7b0b0, 8acccd8, ffcb73e and 242acbe where appropriate.

SWL2-16	Miscellaneous gas optimisations
Asset	`AccessControlManager.sol`, `DepositManager.sol`, `NodeOperatorRegistry.sol`, `swETH.sol`
Status	**Resolved:** See Resolution
Rating	Informational

Description

Throughout the course of the review, the testing team encountered various opportunities for reductions in gas consumption.

Initialisation to default values: it is cheaper not to explicitly set a value for a variable if the Solidity default value is what is desired.

- Loop counter in `DepositManager.sol` on line [**100**]
- Loop counter in `NodeOperatorRegistry.sol` on line [**184**], line [**256**], line [**383**], line [**407**], line [**460**], and line [**493**]
- `smallestOperatorActiveKeys` in `NodeOperatorRegistry` on line [**106**]
- `foundValidators` in `NodeOperatorRegistry` on line [**109**]
- `foundOperatorId` in `NodeOperatorRegistry` on line [**113**]
- Loop counter in `SwellStakeVaultHelper.sol` on line [**119**]
- `nftMigratedCount` in `v3_swNFT.sol` on line [**86**]
- `ethValueMinted` in `v3_swNFT.sol` on line [**87**]

Uncached array lengths during iteration: the array length will be assessed each iteration of the loop. If it does not change within the loop, gas can be saved by copying the array length into a variable (assuming that the number of iterations is greater than one).

- `_pubKeys.length` in `DepositManager.sol` on line [**87**]
- `validatorDetails.length` in `DepositManager.sol` on line [**100**]
- `operatorIdToValidatorDetails[operatorId].length()` in `NodeOperatorRegistry.sol` on line [**127**]
- `_pubKeys.length` in `NodeOperatorRegistry.sol` on line [**182**]
- `_pubKeys.length` in `NodeOperatorRegistry.sol` on line [**184**]
- `operatorIdToValidatorDetails[operatorId].length()` in `NodeOperatorRegistry.sol` on line [**194**]
- `_pubKeys.length` in `NodeOperatorRegistry.sol` on line [**220**]
- `_validatorDetails.length` in `NodeOperatorRegistry.sol` on line [**236**]
- `operatorIdToValidatorDetails[operatorId].length()` in `NodeOperatorRegistry.sol` on line [**249**]
- `validatorDetails.length` in `NodeOperatorRegistry.sol` on line [**250**]
- `validatorDetails.length` in `NodeOperatorRegistry.sol` on line [**256**]
- `_validatorDetails[i].length` in `NodeOperatorRegistry.sol` on line [**258**]
- `_validatorDetails[i].signature.length` in `NodeOperatorRegistry.sol` on line [**262**]

- `_validatorDetails[i].length` in `NodeOperatorRegistry.sol` on line [275]
- `_pubKeys.length` in `NodeOperatorRegistry.sol` on line [383]
- `_pubKeys.length` in `NodeOperatorRegistry.sol` on line [399]
- `_pubKeys.length` in `NodeOperatorRegistry.sol` on line [407]
- `activeValidatorIndexes.length()` in `NodeOperatorRegistry.sol` on line [445]
- `pubKey.length` in `NodeOperatorRegistry.sol` on line [457]
- `pubKey.length` in `NodeOperatorRegistry.sol` on line [460]
- `_SYMBOLS.length` in `NodeOperatorRegistry.sol` on line [461]
- `_SYMBOLS.length` in `NodeOperatorRegistry.sol` on line [462]
- `activeValidatorIndexes.length()` in `NodeOperatorRegistry.sol` on line [472]
- `activeValidatorIndexes.length()` in `NodeOperatorRegistry.sol` on line [476]
- `operatorIdToValidatorDetails[operatorId].length()` in `NodeOperatorRegistry.sol` on line [592]
- `operatorIdToValidatorDetails[operatorId].length()` in `NodeOperatorRegistry.sol` on line [606]
- `operatorIdToValidatorDetails[operatorId].length()` in `NodeOperatorRegistry.sol` on line [612]
- `_addresses.length` in `Whitelist.sol` on line [79]
- `_addresses.length` in `Whitelist.sol` on line [97]
- `tokensFromPool.length` in `SwellStakeVaultHelper.sol` on line [115]
- `tokensFromPool.length` in `SwellStakeVaultHelper.sol` on line [116]
- `tokensFromPool.length` in `SwellStakeVaultHelper.sol` on line [119]

Use of strict inequality on unsigned integers: It is cheaper to just compare against zero rather than whether the integer is explicitly greater than zero and, due to the unsignedness of the type, it is logically equivalent to do so.

- `SwellStakeVaultHelper.sol` on line [87]
- `SwellStakeVaultHelper.sol` on line [96]

Additionally, iteration over arbitrary, caller-provided data structures is present throughout the codebase.

Recommendations

Understand the trade-offs between gas efficiency and other properties such as readability, security, and maintainability; consider and evaluate each item in light of this.

Resolution

The comments above have been acknowledged by the development team, and relevant changes actioned in commit 1e7b0b0, 8acccd8 and f86a300 where appropriate.

Appendix A Test Suite

A non-exhaustive list of tests were constructed to aid this security review and are given along with this document. The `brownie` framework was used to perform these tests and the output is given below.

```
test_DepositManager.py::test_initialize PASSED                                                                               [ 16%]
test_DepositManager.py::test_setup_validators PASSED                                                                         [ 33%]
test_DepositManager.py::test_setup_validators_incorrect_role PASSED                                                          [ 50%]
test_DepositManager.py::test_setup_validators_incorrect_role_paused_bot_methods PASSED                                       [ 66%]
test_DepositManager.py::test_setup_validators_invalid_deposit_root PASSED                                                    [ 83%]
test_DepositManager.py::test_setup_validators_insufficient_eth_balance PASSED                                                [100%]
test_AccessControlManager.py::test_initialize PASSED                                                                         [  7%]
test_AccessControlManager.py::test_setters PASSED                                                                            [ 14%]
test_AccessControlManager.py::test_unpause_core_methods PASSED                                                               [ 21%]
test_AccessControlManager.py::test_unpause_bot_methods PASSED                                                                [ 28%]
test_AccessControlManager.py::test_unpause_operator_methods PASSED                                                           [ 35%]
test_AccessControlManager.py::test_unpause_withdrawals PASSED                                                                [ 42%]
test_AccessControlManager.py::test_pause_core_methods PASSED                                                                 [ 50%]
test_AccessControlManager.py::test_pause_bot_methods PASSED                                                                  [ 57%]
test_AccessControlManager.py::test_pause_operator_methods PASSED                                                             [ 64%]
test_AccessControlManager.py::test_pause_withdrawals PASSED                                                                  [ 71%]
test_AccessControlManager.py::test_set_swETH PASSED                                                                          [ 78%]
test_AccessControlManager.py::test_set_deposit_manager PASSED                                                                [ 85%]
test_AccessControlManager.py::test_set_node_operator_registry PASSED                                                         [ 92%]
test_AccessControlManager.py::test_set_swell_treasury PASSED                                                                 [100%]
test_NodeOperatorRegistry.py::test_withdrawERC20_incorrect_role PASSED                                                       [  1%]
test_NodeOperatorRegistry.py::test_withdrawERC20_no_balance PASSED                                                           [  3%]
test_NodeOperatorRegistry.py::test_withdrawERC20_zero_address PASSED                                                         [  5%]
test_NodeOperatorRegistry.py::test_getNextValidatorDetails_stress PASSED                                                     [  7%]
test_NodeOperatorRegistry.py::test_usePubKeysForValidatorSetup_call_not_from_deposit_manager PASSED                          [  9%]
test_NodeOperatorRegistry.py::test_usePubKeysForValidatorSetup_missing_validator_details PASSED                              [ 11%]
test_NodeOperatorRegistry.py::test_addNewValidatorDetails_paused PASSED                                                      [ 13%]
test_NodeOperatorRegistry.py::test_addNewValidatorDetails_empty_array PASSED                                                 [ 15%]
test_NodeOperatorRegistry.py::test_addNewValidatorDetails_nonexistent_operator PASSED                                        [ 17%]
test_NodeOperatorRegistry.py::test_addOperator_incorrect_role PASSED                                                         [ 19%]
test_NodeOperatorRegistry.py::test_addOperator_zero_address PASSED                                                           [ 21%]
test_NodeOperatorRegistry.py::test_add_operator PASSED                                                                       [ 23%]
test_NodeOperatorRegistry.py::test_updateOperatorRewardAddress_incorrect_role PASSED                                         [ 25%]
test_NodeOperatorRegistry.py::test_updateOperatorRewardAddress_zero_reward_address PASSED                                    [ 26%]
test_NodeOperatorRegistry.py::test_updateOperatorControllingAddress_incorrect_role PASSED                                    [ 28%]
test_NodeOperatorRegistry.py::test_updateOperatorControllingAddress_zero_controlling_address PASSED                          [ 30%]
test_NodeOperatorRegistry.py::test_update_operator_controlling_address PASSED                                                [ 32%]
test_NodeOperatorRegistry.py::test_update_operator_name PASSED                                                               [ 34%]
test_NodeOperatorRegistry.py::test_update_operator_reward_address PASSED                                                     [ 36%]
test_NodeOperatorRegistry.py::test_disable_operator PASSED                                                                   [ 38%]
test_NodeOperatorRegistry.py::test_disableOperator_incorrect_role PASSED                                                     [ 40%]
test_NodeOperatorRegistry.py::test_enableOperator_incorrect_role PASSED                                                      [ 42%]
test_NodeOperatorRegistry.py::test_enable_operator PASSED                                                                    [ 44%]
test_NodeOperatorRegistry.py::test_deletePendingValidators_incorrect_role PASSED                                             [ 46%]
test_NodeOperatorRegistry.py::test_add_new_validator_details PASSED                                                          [ 48%]
test_NodeOperatorRegistry.py::test_add_new_validator_details_operator_methods_paused PASSED                                  [ 50%]
test_NodeOperatorRegistry.py::test_add_new_validator_details_invalid_array PASSED                                            [ 51%]
test_NodeOperatorRegistry.py::test_add_new_validator_details_operator_disabled PASSED                                        [ 53%]
test_NodeOperatorRegistry.py::test_add_new_validator_details_invalid_pubkey_length PASSED                                    [ 55%]
test_NodeOperatorRegistry.py::test_add_new_validator_details_invalid_sig_length PASSED                                       [ 57%]
test_NodeOperatorRegistry.py::test_add_new_validator_details_duplicate_pubkey PASSED                                         [ 59%]
test_NodeOperatorRegistry.py::test_delete_pending_validators PASSED                                                          [ 61%]
test_NodeOperatorRegistry.py::test_delete_pending_validators_nonexistent_pubkey PASSED                                       [ 63%]
test_NodeOperatorRegistry.py::test_delete_pending_validators_active_validator PASSED                                         [ 65%]
test_NodeOperatorRegistry.py::test_use_pubkeys_for_validator_setup PASSED                                                    [ 67%]
test_NodeOperatorRegistry.py::test_use_pubkeys_for_validator_setup_wrong_caller PASSED                                       [ 69%]
test_NodeOperatorRegistry.py::test_use_pubkeys_for_validator_setup_operator_disabled PASSED                                  [ 71%]
test_NodeOperatorRegistry.py::test_use_pubkeys_for_validator_setup_operator_out_pubkeys PASSED                               [ 73%]
test_NodeOperatorRegistry.py::test_use_pubkeys_for_validator_setup_operator_mismatch PASSED                                  [ 75%]
test_NodeOperatorRegistry.py::test_delete_active_validators PASSED                                                           [ 76%]
test_NodeOperatorRegistry.py::test_delete_active_validators_incorrect_role PASSED                                            [ 78%]
test_NodeOperatorRegistry.py::test_delete_active_validators_no_active_validators PASSED                                      [ 80%]
test_NodeOperatorRegistry.py::test_delete_active_validators_validator_not_active PASSED                                      [ 82%]
test_NodeOperatorRegistry.py::test_get_next_validator_details_1 PASSED                                                       [ 84%]
test_NodeOperatorRegistry.py::test_get_next_validator_details_2 PASSED                                                       [ 86%]
test_NodeOperatorRegistry.py::test_get_operator PASSED                                                                       [ 88%]
test_NodeOperatorRegistry.py::test_get_operators_pending_validator_details PASSED                                            [ 90%]
test_NodeOperatorRegistry.py::test_get_reward_details_for_operator_id PASSED                                                 [ 92%]
test_NodeOperatorRegistry.py::test_get_operators_active_validator_details PASSED                                             [ 94%]
test_NodeOperatorRegistry.py::test_get_PoR_address_list PASSED                                                               [ 96%]
test_NodeOperatorRegistry.py::test_getPoRAddressList_end_index_exceeds_start_index PASSED                                    [ 98%]
test_NodeOperatorRegistry.py::test_getPoRAddressList_start_index_out_of_bounds PASSED                                        [100%]
test_swETH.py::test_setSwellTreasuryRewardPercentage PASSED                                                                  [  5%]
test_swETH.py::test_setNodeOperatorRewardPercentage PASSED                                                                   [ 11%]
test_swETH.py::test_setMinimumRepriceTime PASSED                                                                             [ 16%]
test_swETH.py::test_setMaximumRepriceDifferencePercentage PASSED                                                             [ 22%]
test_swETH.py::test_setMaximumRepriceswETHDifferencePercentage PASSED                                                        [ 27%]
test_swETH.py::test_deposit_pbt_nonzero_ether_value PASSED                                                                   [ 33%]
test_swETH.py::test_deposit_zero_ether_value PASSED                                                                          [ 38%]
test_swETH.py::test_deposit_success_one_ether_single_depositor PASSED                                                        [ 44%]
test_swETH.py::test_deposit_success_multiple_depositors PASSED                                                               [ 50%]
test_swETH.py::test_reprice PASSED                                                                                           [ 55%]
test_swETH.py::test_fallback PASSED                                                                                          [ 61%]
test_swETH.py::test_swETHToETHRate PASSED                                                                                    [ 66%]
test_swETH.py::test_ethToSwETHRate PASSED                                                                                    [ 72%]
test_swETH.py::test_getRate PASSED                                                                                           [ 77%]
test_swETH.py::test_withdrawERC20 PASSED                                                                                     [ 83%]
test_swETH.py::test_reprice_bad_initial PASSED                                                                               [ 88%]
test_swETH.py::test_reprice_good_ratio PASSED                                                                                [ 94%]
test_swETH.py::test_reprice_ratios PASSED                                                                                    [100%]

test_SWNFTv3.py::test_basic PASSED                                                                                           [  7%]
test_SWNFTv3.py::test_setSWETHv3 PASSED                                                                                      [ 15%]
test_SWNFTv3.py::test_setHarvestETHRecipient PASSED                                                                          [ 23%]
test_SWNFTv3.py::test_setRescueERC20Recipient PASSED                                                                         [ 30%]
test_SWNFTv3.py::test_harvestETH PASSED                                                                                      [ 38%]
test_SWNFTv3.py::test_rescueERC20 PASSED                                                                                     [ 46%]
test_SWNFTv3.py::test_migrateBatch PASSED                                                                                    [ 53%]
test_SWNFTv3.py::test_set_migration_finished PASSED                                                                          [ 61%]
test_SwellStakeVaultHelper.py::test_constructor PASSED                                                                       [ 69%]
test_SwellStakeVaultHelper.py::test_stakeAndVault_zero_precision PASSED                                                      [ 76%]
test_SwellStakeVaultHelper.py::test_stakeAndVault_stake_fraction_exceeds_precision PASSED                                    [ 84%]
test_SwellStakeVaultHelper.py::test_stakeAndVault_zero_ether_value PASSED                                                    [ 92%]
test_SwellStakeVaultHelper.py::test_stakeAndVault PASSED                                                                     [100%]
```

Appendix B Vulnerability Severity Classification

This security review classifies vulnerabilities based on their potential impact and likelihood of occurance. The total severity of a vulnerability is derived from these two metrics based on the following matrix.

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

Table 1: Severity Matrix - How the severity of a vulnerability is given based on the *impact* and the *likelihood* of a vulnerability.