Sonic Staking Security Review

Auditors

Gerard Persoon, Lead Security Researcher

R0bert, Lead Security Researcher

0xWeiss, Security Researcher

Tnch, Security Researcher

Report prepared by: Lucas Goiriz

December 13, 2024

Contents

1 About Spearbit

Spearbit is a decentralized network of expert security engineers offering reviews and other security related services to Web3 projects with the goal of creating a stronger ecosystem. Our network has experience on every part of the blockchain technology stack, including but not limited to protocol design, smart contracts and the Solidity compiler. Spearbit brings in untapped security talent by enabling expert freelance auditors seeking flexibility to work on interesting projects together.

Learn more about us at spearbit.com

2 Introduction

Beethoven X is a next generation decentralized investment platform that provides innovative, capital-efficient, and sustainable solutions for all DeFi users.

Disclaimer: This security review does not guarantee against a hack. It is a snapshot in time of Sonic Staking according to the specific commit. Any modifications to the code will require a new security review.

3 Risk classification

Severity level	Impact: High	Impact: Medium	Impact: Low
Likelihood: high	Critical	High	Medium
Likelihood: medium	High	Medium	Low
Likelihood: low	Medium	Low	Low

3.1 Impact

- High - leads to a loss of a significant portion (>10%) of assets in the protocol, or significant harm to a majority of users.
- Medium - global losses <10% or losses to only a subset of users, but still unacceptable.
- Low - losses will be annoying but bearable--applies to things like griefing attacks that can be easily repaired or even gas inefficiencies.

3.2 Likelihood

- High - almost certain to happen, easy to perform, or not easy but highly incentivized
- Medium - only conditionally possible or incentivized, but still relatively likely
- Low - requires stars to align, or little-to-no incentive

3.3 Action required for severity levels

- Critical - Must fix as soon as possible (if already deployed)
- High - Must fix (before deployment if not already deployed)
- Medium - Should fix
- Low - Could fix

4 Executive Summary

Over the course of 4 days in total, Beethoven engaged with Spearbit to review the sonic-staking protocol. In this period of time a total of **27** issues were found.

Summary

Project Name	Beethoven
Repository	sonic-staking
Commit	4f0e016c
Type of Project	DeFi, Staking
Audit Timeline	Dec 9th to Dec 10th
Fix period	Dec 11

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	0	0	0
High Risk	0	0	0
Medium Risk	2	2	0
Low Risk	9	8	1
Gas Optimizations	5	2	3
Informational	11	4	7
Total	**27**	**16**	**11**

5 Findings

5.1 Medium Risk

5.1.1 Undelegations from pool cannot be paused

Severity: Medium Risk

Context: SonicStaking.sol#L387

Description: Accounts that are granted the `OPERATOR` or `DEFAULT_ADMIN_ROLE` role are entitled to pause multiple features of the `SonicStaking` contract. According to the documentation this should include, among others, undelegation of assets.

However, while the `undelegate` function correctly implements this security mechanism to check whether undelegations are paused (by reading the `undelegatePaused` state variable flag), the `undelegateFromPool` function does not. This means that users can still undelegate assets from the pool even after an operator or admin intended to pause all types of undelegations.

Recommendation: Update the `undelegateFromPool` function to check that undelegations are not paused before executing undelegations from the pool.

Beethoven: Fixed in PR 43.

Spearbit: Fixed.

5.1.2 Unhandled error in emergency clawback execution may prevent accounting lost funds after slashing

Severity: Medium Risk

Context: SonicStaking.sol#L555

Description: The `operatorExecuteClawBack` function of the `SonicStaking` contract is intended for an operator to withdraw assets to the pool. It's intended to be the only way to reduce the rate. An operator can acknowledge that the amount withdrawn may be less than what's owed (setting `emergency` to `true`), having lost these funds in slashing events of the `SFC` contract.

To execute the actual withdrawal of funds from the `SFC` contract, the `operatorExecuteClawBack` function calls `SFC::withdraw`, indicating the validator's ID from which it intends to withdraw. However, this validator may be fully slashed. If so, the call to `SFC::withdraw` will revert with a `StakeIsFullySlashed` error. This error is unhandled in the `operatorExecuteClawBack` function, and therefore will pop-up the revert too, preventing the clawback execution. This means that neither `totalPool` nor `pendingClawBackAmount` will be updated, thus failing to account for the lost funds in the fully slashed validator. At this point the `SonicStaking` contract will have lost assets, but would not be able to realize the losses and reduce the rate accordingly.

The only alternative left would be to wait until the validator's slashing refund ratio is updated in the `SFC` contract, which may allow avoiding the `StakeIsFullySlashed` error and withdrawing funds from the validator again.

Recommendation: To be able to realize the losses and reduce the rate even in the event of a fully slashed validator, consider updating the function to better handle this edge case. Any implementation should consider that the `slashingRefundRatio` for a validator in the `SFC` contract might not be set yet, and that it defaults to zero (meaning, the slashed validator's penalty defaults to 100%). So realizing losses in the `SonicStaking` contract must not be done before the validator is intendedly assigned a refund ratio by the owner of the `SFC` contract.

Consider also adding this functionality to `withdraw()` to clean up the `withdrawId` and for symmetry.

Beethoven: Implemented the alternative approach (not `try-catch`) in PR 51.

Spearbit: Fixed.

5.2 Low Risk

5.2.1 Reentrancy risks

Severity: Low Risk

Context: SonicStaking.sol#L318, SonicStaking.sol#L387, SonicStaking.sol#L669, SonicStaking.sol#L690

Description: There is a reentrancy risk in `claimRewards()`:

- Assume the `treasury` wants to pro-active and immediately `deposit()` the fee back into the `SonicStaking` protocol;
- The `treasury` gets control at the `treasury.call{...}("")` of `claimRewards()` and can then initiate actions;
- The `treasury` can then directly call `deposit()`, which doesn't have a `nonReentrant` modifier (see the proof of concept below);
- The `totalPool` is adjusted right after the external call, so within the external call the shares are relatively cheap;
- So by calling `deposit()` the treasury will get more shares than it would have gotten, when it had done the `deposit()` at a later moment.

Although the `treasury` is trusted, it could still do this in a well intended way.

`undelegateFromPool()` could also be called and there are read only reentrancy risks, because the view functions `getRate()`, `totalAssets()`, `convertToShares()` and `convertToAssets()` could also be called and would give an inaccurate result.

Proof of Concept:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;
import "hardhat/console.sol";

contract Testcontract {
    uint totalPool = 10e18;
    uint totalSupply = 10e18;
    uint treasuryShares;

    function totalAssets() public view returns (uint256) {
        return totalPool;
    }
    function convertToShares(uint256 assetAmount) public view returns (uint256) {
        uint256 assetsTotal = totalAssets();
        uint256 totalShares = totalSupply;
        return (assetAmount * totalShares) / assetsTotal;
    }
    function convertToAssets(uint256 sharesAmount) public view returns (uint256) {
        uint256 assetsTotal = totalAssets();
        uint256 totalShares = totalSupply;
        return (sharesAmount * assetsTotal) / totalShares;
    }
    function deposit(uint amount) public payable returns (uint){
        uint256 sharesAmount = convertToShares(amount);
        totalPool += amount;
        totalSupply += sharesAmount; // equivalent of mint
        return sharesAmount;
    }
    function claimRewards(uint totalRewardsClaimed) public {
        treasuryShares += deposit(totalRewardsClaimed); // simulate the treasury directly deposits
        totalPool += totalRewardsClaimed; // this is done after the external call
    }
    constructor() {
```

```
        uint totalRewardsClaimed = 1e18;
        claimRewards(totalRewardsClaimed);
        console.log("TotalRewardsClaimed %s * 1e16",totalRewardsClaimed / 1e16);
        console.log("Converted to shares and back to assets %s * 1e16",convertToAssets(treasuryShares)
↪  / 1e16);
        // TotalRewardsClaimed 100 * 1e16
        // Converted to shares and back to assets 109 * 1e16
    }
}
```

Recommendation: Consider doing the following:

- Adjust `totalPool` before the external call.
- Additionally consider adding the `nonReentrant` modifier to `deposit()` and `undelegateFromPool()`.

Beethoven:

- `undelegateFromPool` `nonReentrant` in PR 42.
- `deposit` `nonReentrant` in PR 55.
- Moved accounting for `totalPool` in PR 58.

Spearbit: Fixed.

5.2.2 Out-of-bounds access errors in deployment script

Severity: Low Risk

Context: DeploySonicStaking.sol#L34-L35, DeploySonicStaking.sol#L39-L40

Description: In the deployment script defined in `script/DeploySonicStaking.sol`, there are two instances of an out-of-bounds access error that will prevent the script from executing. The `ownerProposers` and `adminProposers` arrays are initialized with a length of zero, but they should be initialized with a length of 1.

Recommendation: Update the script to correctly define the arrays' initial size.

```
- address[] memory ownerProposers = new address[](0);
+ address[] memory ownerProposers = new address[](1);
  ownerProposers[0] = sonicStakingOwner;
```

```
- address[] memory adminProposers = new address[](0);
+ address[] memory adminProposers = new address[](1);
  adminProposers[0] = sonicStakingAdmin;
```

Beethoven: Fixed in PR 56.

Spearbit: Fixed.

5.2.3 Potential race condition for `undelegateFromPool()`

Severity: Low Risk

Context: SonicStaking.sol#L387

Description: After a slashing event, `undelegateFromPool()` will result in more funds than `undelegate()` from a validator. This might result in a race condition to do `undelegateFromPool()`, because that prevents others from doing the same, depending on the amount in the pool.

Recommendation: To reduce the impact of this issue, make sure the pool is relatively small. This can be done be regularly calling `delegate()`, especially directly after `operatorExecuteClawBack()`.

Also see "Assymmetry between deposit()/delegate() and undelegate()".

Beethoven: Acknowledged. We actively monitor the free pool and `delegate()` regularly.

Spearbit: Acknowledged.

5.2.4 Funds directly send to the `SonicStaking` contract get stuck

Severity: Low Risk

Context: SonicStaking.sol#L765

Description: Function `receive()` allows receiving of funds, but it does not register this in the `totalPool`. Combined with `claimRewards()` and `withdraw()` this works fine, because they do calculate the added funds. However if funds are send in a different way they stay stuck in the contract.

Note: they can be retrieved after upgrading the contract.

Recommendation: Consider reverting if `msg.sender != SFC`, so only from the SFC contract are accepted. *Note: this does cost some extra gas.*

Beethoven: Fixed in PR 53.

Spearbit: Fixed.

5.2.5 Missing input validation in `SFC` and `treasury` initialization

Severity: Low Risk

Context: SonicStaking.sol#L1, SonicStaking.sol#L189, SonicStaking.sol#L190

Description: Missing input validation in `SFC` and `treasury` initialization. In fact, it is intended for such variables to be validated, as can be seen in the setter function from the treasury, where it is checked that the address is not set to 0:

```
function setTreasury(address newTreasury) external onlyRole(DEFAULT_ADMIN_ROLE) {
    require(newTreasury != address(0), TreasuryAddressCannotBeZero());

    treasury = newTreasury;
}
```

Recommendation:

```
  function initialize(ISFC _sfc, address _treasury) public initializer {
      __ERC20_init("Beets Staked Sonic", "stS"); //ok
      __ERC20Burnable_init();
      __ERC20Permit_init("Beets Staked Sonic");

    __Ownable_init(msg.sender);
    __UUPSUpgradeable_init();
    __ReentrancyGuard_init();
    _grantRole(DEFAULT_ADMIN_ROLE, msg.sender);
+   require(_treasury != address(0), TreasuryAddressCannotBeZero());
+   require(address(_sfc) != address(0), SFCAddressCannotBeZero());

    SFC = _sfc;
    treasury = _treasury;
```

Beethoven: Fixed in PR 45.

Spearbit: Fixed.

5.2.6 `undelegation` can be partially DoSed for some users

Severity: Low Risk

Context: SonicStaking.sol#L343

Description: Some users will not be able to undelegate their staked assets given that assets and shares are not always 1:1 , specially because of there is a permissioned donate function to inflate the `totalPool` amount. There is a requirement in the `deposit` function where the minimum amount to deposit is 1 ether:

```
function deposit() external payable {
    uint256 amount = msg.value;
    require(amount >= MIN_DEPOSIT, DepositTooSmall()); // <<<
```

The same minimum amount is required for undelegation:

```
function undelegate(uint256 validatorId, uint256 amountShares) public nonReentrant returns (uint256
↪  withdrawId) {
    require(!undelegatePaused, UndelegatePaused());
    require(amountShares >= MIN_UNDELEGATE_AMOUNT_SHARES, UndelegateAmountTooSmall()); // <<<
```

So if you deposit the minimum: `1e18`, you will most likely mint less than `1e18` shares if there has been any donation at some point, which can affect a users undelegation, needing them to deposit more to be able to undelegate all their shares. Simple example to demonstrate how the assets and shares could differ easily:

```
* - 1: first user deposits 2e18 (2 ETHER)
*      - assetTotal = 2e18 + 0 + 0 = 2e18
*      - totalShares = 2e18 shares minted
* - 2: second user deposits 1e18 (1 ETHER)
*      - `convertToShares` (1e18 * 2e18) / 2e18 = 1e18 shares minted
*      - assetTotal = 3e18 + 0 + 0 = 3e18
*      - totalShares = 3e18
* - 3: Operator Role donates 1e18
*      - assetTotal = 4e18 + 0 + 0 = 4e18
*      - totalShares = 3e18
* - 4: third user deposits 1e18 (1 ETHER)
*      - `convertToShares` (1e18 * 3e18) / 4e18 = less than 1e18, which is less than
↪  `MIN_UNDELEGATE_AMOUNT_SHARES`, it would revert
```

Recommendation: The minimum amount of shares to withdraw does not need to be that big. The best decision might be to monitor the difference between assets and shares and change the minimum amount required to be undelegated to not be constant, and update it accordingly with the asset to share ratio.

Beethoven: Fixed in PR 52 lowering the `MIN_UNDELEGATE_AMOUNT_SHARES`.

Spearbit: Fixed.

5.2.7 Delegations of the entire totalPool can be griefed

Severity: Low Risk

Context: SonicStaking.sol#L490, SonicStaking.sol#L506

Description: Delegations are permissioned only to the operator role. An intended action would be for the operator role to fetch the total pool value and call delegate with the entire amount:

```
function delegate(uint256 validatorId, uint256 amount) external nonReentrant onlyRole(OPERATOR_ROLE) {
    require(amount > 0, DelegateAmountCannotBeZero());
    require(amount <= totalPool, DelegateAmountLargerThanPool()); // <<<

    totalPool -= amount;
    totalDelegated += amount;

    SFC.delegate{value: amount}(validatorId);

    emit Delegated(validatorId, amount);
}
```

This can be prevented by a just-in-time undelegation from a user, maliciously or not, where the `totalPool` would decrease:

```
function undelegateFromPool(uint256 amountShares) external returns (uint256 withdrawId) {
    require(amountShares >= MIN_UNDELEGATE_AMOUNT_SHARES, UndelegateAmountTooSmall());
    //  uint256 public constant MIN_UNDELEGATE_AMOUNT_SHARES = 1 ether;
    uint256 amountToUndelegate = convertToAssets(amountShares);

    require(amountToUndelegate <= totalPool, UndelegateAmountExceedsPool());

    _burn(msg.sender, amountShares);

    // The validatorId is ignored for pool withdrawals
    withdrawId = _createAndPersistWithdrawRequest(WithdrawKind.POOL, 0, amountToUndelegate);

    // The amount is subtracted from the pool, but the assets stay in this contract.
    // The user is able to `withdraw` their assets after the `withdrawDelay` has passed.
    totalPool -= amountToUndelegate; // <<<

    emit Undelegated(msg.sender, withdrawId, 0, amountToUndelegate, WithdrawKind.POOL);
}
```

Therefore the initial delegation by the operator would revert given that the following require statement would not be true:

```
require(amount <= totalPool, DelegateAmountLargerThanPool());
```

Recommendation: Change the following line:

```
  function delegate(uint256 validatorId, uint256 amount) external nonReentrant onlyRole(OPERATOR_ROLE) {
      require(amount > 0, DelegateAmountCannotBeZero());
-     require(amount <= totalPool, DelegateAmountLargerThanPool());
+     if (amount > totalPool) amount = totalPool;

      totalPool -= amount;
      totalDelegated += amount;

  if (amount > totalPool) amount = totalPool;
```

Beethoven: Fixed in PR 46 and PR 62.

Spearbit: Fixed.

5.2.8 Missing event emission for key state changes

Severity: Low Risk

Context: SonicStaking.sol#L652

Description: The functions `setTreasury` and `setProtocolFeeBIPS` do update key state variables but do not emit events:

```
function setTreasury(address newTreasury) external onlyRole(DEFAULT_ADMIN_ROLE) {
    require(newTreasury != address(0), TreasuryAddressCannotBeZero());

    treasury = newTreasury;
}
```

```
function setProtocolFeeBIPS(uint256 newFeeBIPS) external onlyRole(DEFAULT_ADMIN_ROLE) {
    require(newFeeBIPS <= MAX_PROTOCOL_FEE_BIPS, ProtocolFeeTooHigh());

    protocolFeeBIPS = newFeeBIPS;
}
```

Recommendation: Emit an event for each of these functions:.

Beethoven: Fixed in PR 39.

Spearbit: Fixed.

5.2.9 `operatorInitiateClawBack()` can be griefed

Severity: Low Risk

Context: SonicStaking.sol#L341, SonicStaking.sol#L505

Description: When calling `operatorInitiateClawBack()`, a user can front-run the call by calling `undelegate()` for the same `validatorId` and prevent `operatorInitiateClawBack()` from executing.

Recommendation: Consider lowering the `amountAssets` to the amount that is available, for example in the following way:

```
  function operatorInitiateClawBack(/*...*/, uint256 amountAssets) /*...*/ {
      // ...
      uint256 amountDelegated = SFC.getStake(address(this), validatorId);
+     if (amountAssets> amountDelegated) {
+         amountAssets = amountDelegated;
+     }
      // ...
-     require(amountAssets <= amountDelegated, UndelegateAmountExceedsDelegated(validatorId));
      // ...
  }
```

Beethoven: Addressed in PR 61 and PR 62.

Spearbit: Fixed.

5.3 Gas Optimization

5.3.1 `modifier withValidWithdrawId` can be optimized

Severity: Gas Optimization

Context: SonicStaking.sol#L205-L209

Description: The `modifier withValidWithdrawId` retrieves the storage variable `request.requestTimestamp` twice which costs extra gas.

Recommendation: Consider storing the value of `request.requestTimestamp` in a temporary variable.

Beethoven: Acknowledged, we'll leave this as is though.

Spearbit: Acknowledged.

5.3.2 `nonReentrant` used inside a loop

Severity: Gas Optimization

Context: SonicStaking.sol#L471-L475

Description: The functions `withdrawMany()` and `undelegateMany()` repeatedly call an underlying function that has the `nonReentrant` modifier. This means the `nonReentrant` flag is repeatedly turned on and off, which cost unnecessary gas

Recommendation: Consider making `external` and `internal` functions for `withdraw()` and `undelegate()`. Call the `internal` functions from `withdrawMany()` and `undelegateMany()` and move the `nonReentrant` modifier to the external functions.

Beethoven: Fixed in PR 41 and PR 57.

Spearbit: Fixed.

5.3.3 Role declaration is cheaper when hardcoded

Severity: Gas Optimization

Context: SonicStaking.sol#L34

Description: When computing roles such as `bytes32 public constant OPERATOR_ROLE = keccak256("OPERATOR_ROLE");` it is cheaper to pre-compute them and hardcode them before deployment

Recommendation: Hardcode the result of the `keccak256("OPERATOR_ROLE")`.

Beethoven: Acknowledged, but as gas is not an issue on `Sonic`, we've decided to leave it as is.

Spearbit: Acknowledged.

5.3.4 Redundant check in `claimRewards()`

Severity: Gas Optimization

Context: SonicStaking.sol#L669, SonicStaking.sol#L683-L687

Description: Function `claimRewards()` first enforces `totalRewardsClaimed > MIN_CLAIM_REWARDS_AMOUNT` and then checks `totalRewardsClaimed > 0`. The second check isn't necessary because it can never be `true`.

Recommendation: Consider changing the code to:

```
- if (totalRewardsClaimed > 0 && protocolFeeBIPS > 0) {
+ if (protocolFeeBIPS > 0) {
```

Beethoven: Fixed in PR 48.

Spearbit: Fixed.

5.3.5 `SFC` could be `immutable`

Severity: Gas Optimization

Context: SonicStaking.sol#L73, SonicStaking.sol#L172, SonicStaking.sol#L179-L189

Description: `SFC` doesn't change, so it could also be made `immutable` and set in the `contructor`.

Recommendation: Consider setting `SFC` in the constructor and making it immutable.

Beethoven: Acknowledged, but we'll leave as is.

Spearbit: Acknowledged.

5.4 Informational

5.4.1 Different parameter order in event `Withdrawn` and `OperatorClawBackExecuted`

Severity: Informational

Context: SonicStaking.sol#L136, SonicStaking.sol#L140

Description: In event `Withdrawn`, the `emergency` is the last parameter, while in event `OperatorClawBackExecuted` it is in the middle. Using the same order will make the code more readable.

Recommendation: Consider moving `emergency` to the end of `OperatorClawBackExecuted`.

Beethoven: Fixed in PR 44.

Spearbit: Fixed.

5.4.2 Constructor had modifier `initializer`

Severity: Informational

Context: SonicStaking.sol#L172

Description: The `constructor` of `SonicStaking.sol` has the modifier `initializer`, to prevent executing `initialize()` on the implementation contract.

The documented way by OpenZeppelin is to use `_disableInitializers()`.

Recommendation: Consider using `_disableInitializers()`:

```
- constructor() initializer {}
+ constructor() {
+     _disableInitializers();
+ }
```

Beethoven: Fixed in PR 60.

Spearbit: Fixed.

5.4.3 Timing-dependent user rewards

Severity: Informational

Context: SonicStaking.sol#L341

Description: In the SonicStaking contract, users risk leaving behind a portion of their accrued rewards if they do not promptly `undelegate` after a "claimer" calls the `claimRewards` function for all the validators. The amount of rewards "lost" grows as time passes since the last rewards claim. When claims are infrequent the unclaimed rewards can become more significant. Users need to be aware that their yield is partially dependent on prompt action following reward claims.

Recommendation: Merely informative. Ensure that:

- Rewards are claimed frequently.
- Users are informed of the described functionality.

Beethoven: Acknowledged. We run claims once an hour on `fantom opera` and intend to continue this on `Sonic`. This way the amount of pending rewards should be negligible at any given time.

Spearbit: Acknowledged.

5.4.4 `withdrawDelay` difference between `SonicStaking` and `SFC`

Severity: Informational

Context: SonicStaking.sol#L191

Description: The `withdrawDelay` of `SonicStaking.sol` is 14 days. However the SFC `withdrawalPeriodTime` is 7 days. This leads to unnecessary delays on the `SonicStaking.sol` contract.

Recommendation: Consider using the same period as the `SFC` contract and have a procedure to monitor changes of `withdrawalPeriodTime` in the SFC contract.

Alternatively consider querying directly `SFC.constsAddress().withdrawalPeriodTime()`.

Beethoven: Will not be solved in a technical way.

Spearbit: Acknowledged.

5.4.5 Lack of a two-step transfer ownership pattern

Severity: Informational

Context: SonicStaking.sol#L29

Description: The `SonicStaking` contract uses the `OwnableUpgradeable` library which involves the current owner calling the `transferOwnership()` function. If the nominated EOA account is not a valid account, it is possible that the owner may accidentally transfer ownership to an uncontrolled account thereby losing access to all the functions with the `onlyOwner` modifier. This contract relies on the current owner to perform contract upgrades.

Recommendation: It is recommended to implement a two-step ownership transfer where the owner nominates a new owner and the nominated account explicitly accepts ownership. This ensures the nominated EOA account is a valid and active account. This can be achieved by using an implementation similar to the OpenZeppelin's Ownable2Step library.

Beethoven: Acknowledged. The owner of the contract will be a timelock which we assume won't change, or at least not frequently.

Spearbit: Acknowledged.

5.4.6 Functions `deposit()` and `operatorExecuteClawBack()` don't return a value

Severity: Informational

Context: SonicStaking.sol#L318-L333, SonicStaking.sol#L411-L415

Description: `deposit()` doesn't return a value, while the inverse value `withdraw()` returns `amountWithdrawn`. `operatorExecuteClawBack()`, that is similar to `withdraw()` also doesn't return a value.

Recommendation: Consider doing the following:

- Return `sharesAmount` from function `deposit()`;
- Return `actualWithdrawnAmount` from function `operatorExecuteClawBack()`.

Beethoven: Implemented on the `operatorExecuteClawBack()` but not for `deposit()` in PR 51. Deposit does return a value now. It looks like it was done as a commit instead of part of a (see 4f543b6e).

5.4.7 Assymmetry between `deposit()`/`delegate()` and `undelegate()`

Severity: Informational

Context: *(No context files were provided by the reviewer)*

Description: There is an assymmetry between `deposit()`/`delegate()` and `undelegate()`:

- With `deposit()` the user can't select a validator;
- With `undelegate()` the user has to select a validator.

Related to this there are a number of practical issues:

- It is difficult for a user to select the validator to `undelegate()` from:
  - It is difficult to see if a validator is `slashed`. This is also important to know because its unfavourable to `undelegate()` from a slashed validator for the user: it is better to wait for the operator to do `operatorInitiateClawBack()`;
  - It is difficult to see if the penalty for the `slashing` has been set;
  - The used validators can only be retrieved via `events`, which can only be done offchain and limits composability;
- The user can still influence the used validators by selectively `undelegate()`;
- An operator has to do the `delegate()`, which is an extra action and also has some time delays.

Recommendation: Consider doing one or more of the following:

- Track what amount is delegated to which validator. This is an expansion of the already tracked `totalDelegated`. Then make this information accessible via view functions;
- Have a `view` function to query if a validator is `slashed`;
- Have a `view` function to query if the penalty for the `slashing` has been set.

Beethoven: Appreciate the great input. We've opted to not include these changes and defer to the SFC as the source of truth.

Spearbit: Acknowledged.

5.4.8 Function `_burn()` doesn't require an allowance

Severity: Informational

Context: SonicStaking.sol#L350, SonicStaking.sol#L394

Description: The function `_burn()` doesn't require an allowance. This might be unexpected for the user calling `undelegate()` / `undelegateFromPool()`.

Recommendation: Double check if this is the desired behaviour. If not, consider using `burnFrom()`, which does check the allowance.

Beethoven: This is the desired behavior.

Spearbit: Acknowledged.

5.4.9 When to use `emergency == true`?

Severity: Informational

Context: SonicStaking.sol#L411

Description: For a user it is difficult to know when to use `emergency == true` when calling `withdraw()`. The same is true for an `operator` when calling `operatorExecuteClawBack()`.

Recommendation: Consider adding the following:

- Tave a view function to query if a validator is slashed;
- Tave a view function to query if the penalty for the slashing has been set;
- For extra safety in functions `withdraw()` and `operatorExecuteClawBack()`, when `emergency == true` and the validator is slashed, check the penalty for the slashing has been set.

See also the related issue: "Unhandled error in emergency clawback execution may prevent accounting lost funds after slashing".

Beethoven: Acknowledged, we've decided to not include this since it can be queried on the `SFC` and `SonicStaking` exposes a reference to the `SFC`. `Withdraw()` now checks for slashing more thoroughly see finding "Unhandled error in emergency clawback execution may prevent accounting lost funds after slashing".

Spearbit: Acknowledged.

5.4.10 Incorrect comment for admin functions

Severity: Informational

Context: SonicStaking.sol#L608, SonicStaking.sol#L760

Description: Before the functions that only accounts with the `DEFAULT_ADMIN_ROLE` role can call, there's an incorrect comment reading "`OWNER functions`". Additionally there is one OWNER function that doesn't have a prefix comment: `_authorizeUpgrade()`.

Recommendation: Update the comment so that it reads "`DEFAULT_ADMIN_ROLE functions`". Add a comment `OWNER functions` before `_authorizeUpgrade()`.

Beethoven: Fixed in PR 40.

Spearbit: Fixed.

5.4.11 Floating pragma

Severity: Informational

Context: SonicStaking.sol#L2

Description: Using a strict pragma is widely considered a best practice. Employing a floating pragma can lead to the inadvertent deployment of contracts with outdated or problematic compiler versions. Currently, the `SonicStaking` contract has a floating pragma of `^0.8.27`.

Recommendation: Consider setting the Solidity compiler version to a specific stable release when deploying production contracts. In this concrete case, choose whether you want to perform the deployment using the 0.8.27 or the latest 0.8.28 Solidity compiler version.

Beethoven: Acknowledged. We'll leave as is since there will be a single deployment of this contract.

Spearbit: Acknowledged.