Sphinx Security Review

Auditors

Desmond Ho, Lead Security Researcher

Saw-Mon and Natalie, Lead Security Researcher

M4rio.eth, Security Researcher

David Chaparro, Junior Security Researcher

Sabnock, Junior Security Researcher

Report prepared by: Lucas Goiriz

December 11, 2023

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

Sphinx automates the smart contract deployment process by funding, executing, and verifying deployments on your behalf. Its key features are gasless deployments, one-click multichain deployments, deployments in CI, automatic etherscan verification and Forge Scripts compatibility.

Disclaimer: This security review does not guarantee against a hack. It is a snapshot in time of sphinx 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 3 days in total, Sphinx-Labs engaged with Spearbit to review the sphinx protocol. In this period of time a total of **26** issues were found.

Summary

Project Name	Sphinx-Labs
Repository	sphinx
Commit	fdf467...e5a332
Type of Project	Infrastructure, Merkle Trie
Audit Timeline	Nov 23 to Nov 26
Two week fix period	Nov 26 - Nov 28

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	0	0	0
High Risk	0	0	0
Medium Risk	0	0	0
Low Risk	6	5	1
Gas Optimizations	9	8	1
Informational	11	10	1
Total	**26**	**23**	**3**

5 Findings

5.1 Low Risk

5.1.1 When `SphinxModule` is initialised the integrity of the used safe wallet is not checked

Severity: Low Risk

Context: SphinxModule.sol#L72-L75

Description: When `SphinxModule` is initialised the integrity of the used safe wallet is not checked. Using an older or a newer safe wallet or a customised one can potentially cause funds to be locked/lost.

Recommendation: Perform the following checks upon initialisation of the `SphinxModule` to make sure the safe wallet is one of the official ones:

1. Check the codehash of the safe wallet proxy and make sure it belongs to the correct set of code hashes.
2. Call the `_safeProxy`'s `masterCopy()` endpoints and make sure it returns one of the official implementation `singleton` addresses (note this might be chain specific).

Sphinx-Labs:

We decided to use the same strategy as Gnosis Safe to mitigate the threat of malicious Gnosis Safe singletons: we'll maintain a list of trusted singleton addresses off-chain.

To prevent users from accidentally using a non-malicious yet incompatible Gnosis Safe singleton with Sphinx, the `SphinxModuleProxy`'s `initializer` function checks that the user's Gnosis Safe proxy and Gnosis Safe singleton have compatible code hashes.

See commit 192ca7e7 for the relevant contracts and specs updates.

Spearbit: Fixed in PR 1241 by adding checks for codehashes for the proxy and singleton contracts and only verifying addresses of the trusted singletons off-chain.

Although the PR allows mix and matching the proxy and singleton between the allowed version of the `safe-contracts` (`1.3.0` and `1.4.1`), it should not cause an issue since the difference between the proxy byte codes are only `ipfs` hashes:

```
0x608060405273ffffffffffffffffffffffffffffffffffffffff600054167fa619486e0000000000000000000000000000000
↪ 0000000000000000000000006000351415605057806000526020600f35b3660008037600080366000845af43d6000803e
↪ 6000811415607057d6000fd5b3d6000f3 fe a2 64 69706673 58 22 1220
↪ 03d1488ee65e08fa41e58e888a9865554c535f2c77126a82cb4c0f917f314413 64 736f6c63 43 000706 0033
0x608060405273ffffffffffffffffffffffffffffffffffffffff600054167fa619486e0000000000000000000000000000000
↪ 0000000000000000000000006000351415605057806000526020600f35b3660008037600080366000845af43d6000803e
↪ 6000811415607057d6000fd5b3d6000f3 fe a2 64 69706673 58 22 1220
↪ d1429297349653a4918076d650332de1a1068c5f3e07c5c82360c277770b9552 64 736f6c63 43 000706 0033
```

5.1.2 Check that `networkDeploymentData.txs` conforms to `SphinxTransaction`

Severity: Low Risk

Context: merkle-tree.ts#L258

Description: The properties of `networkDeploymentData` are not guaranteed to conform to `SphinxTransaction`.

Recommendation: The form of `networkDeploymentData` should be checked.

Sphinx-Labs: Fixed in commit b2f9e1fc.

Spearbit: Fixed.

5.1.3 3rd merkle tree invariant isn't checked when generating merkle tree

Severity: Low Risk

Context: merkle-tree.ts#L149-L232

Description: The 3rd invariant stated in the merkle tree specification:

```
If `arbitraryChain` is true in *any* `APPROVE` leaf, then there must be exactly one `APPROVE` leaf in
↪  the entire tree, and no `CANCEL` leaves
```

isn't checked when generating the merkle tree, thus leading to possible violations of this invariant.

Recommendation: An error should be thrown in the cancel case if there is an arbitrary approval leaf.

```
// If there has already been an arbitrary approval leaf, then throw an error
if (arbitraryApprovalIncluded) {
  throw new Error(
    'Detected arbitraryChain = true in multiple DeploymentData entries'
  )
}
```

The converse should also be checked by introducing a `cancellationIncluded` variable and reverting in the approval / execution code block.

```
// If there has already been an arbitrary cancellation leaf, then throw an error
if (cancellationIncluded) {
  throw new Error(
    'Detected cancellation while trying to set arbitraryChain = true in multiple DeploymentData entries'
  )
}
```

Sphinx-Labs: Fixed in commit 58cf99cd.

Spearbit: Fixed.

5.1.4 The `execute` function does not handle the return data

Severity: Low Risk

Context: SphinxModule.sol#L367

Description: When an `EXECUTE` leaf is executed, the Gnosis Safe function `execTransactionFromModule` is called. This function handles only success/revert state of a function which means, if an action fails silently (meaning that it returns a `false` as a bool for example) the action will be marked as success within the `SphinxModule`. The Gnosis Safe module has also a function called `execTransactionFromModuleReturnData` which also returns the data that the transaction returns. This can be used to expand the functionality of the current `SphinxModule` to handle also the silent reverts.

Recommendation: Consider to adopt the logic to handle the silent revert within the `SphinxModule`. If this requires too many changes to the current version, consider carefully document it for the users to be aware of this.

Sphinx-Labs: Acknowledged. We don't think it's worth adding this complexity to the `SphinxModule`. Instead, if this is something that's requested by users, we think it'd make sense to include a Gnosis Safe library that contains this logic.

Spearbit: Acknowledged.

5.1.5 The EIP-712 domain separator is missing the version field

Severity: Low Risk

Context: SphinxModule.sol#L41

Description: The EIP-712 domain separator is used to uniquely for hashing and signing of typed structured data as opposed to just bytestrings.

In the EIP there are various attributes that should be incorporated in the signature. The implementation in the Sphinx Module is missing to encode some of these attributes like:

- Version.
- Verifying contract.
- Chain id.

We do understand why chain id and verifying contract where not implemented due to one signature being used for multiple deployments on multiple chains, we do consider that the version is critical to be implemented in case a new versions of the SphinxModule is developed, the signatures meant for one version can be reused for other versions as well.

Recommendation: Consider incorporating the `VERSION` field in the EIP712 domain separator.

Sphinx-Labs: Fixed in commit f537a778.

Spearbit: Fixed.

5.1.6 OpenZeppelin library leaves sorting differs from 9th merkle tree invariant

Severity: Low Risk

Context: merkle-tree.ts#L1

Description: `@openzeppelin/merkle-tree` is used to assemble the Merkle tree. Its implementation sorts the leaves according to their hashes then stored starting from the last slot in the tree array.

```
export function makeMerkleTree(leaves: Bytes[]): Bytes[] {
  leaves.forEach(checkValidMerkleNode);

  if (leaves.length === 0) {
    throw new Error('Expected non-zero number of leaves');
  }

  const tree = new Array<Bytes>(2 * leaves.length - 1);

  for (const [i, leaf] of leaves.entries()) {
    tree[tree.length - 1 - i] = leaf;
  }
  for (let i = tree.length - 1 - leaves.length; i >= 0; i--) {
    tree[i] = hashPair(
      tree[leftChildIndex(i)]!,
      tree[rightChildIndex(i)]!,
    );
  }

  return tree;
}
```

This likely runs contrary to the 9th high-level merkle tree invariant: Merkle tree leaves must be ordered in the tree by the leaf's `index` and `chainId` fields ascending.

Recommendation: Update the invariant to be consistent with the library implementation.

Sphinx-Labs: Addressed in commit 82fbfda7. We resolved this by updating the specification to reflect the actual ordering of the leaves within the tree as determined by `@openzeppelin/merkle-tree`. Note that instead of testing this invariant ourselves, we specifically called out an assumption that the hashing and sorting logic implemented in @openzeppelin/merkle-tree is robust and bug-free. We chose not to implement our own tests for this as they would have used a very similar sorting mechanism as the underlying dependency does. We think it's more transparent to cover it in an assumption.

Spearbit: Fixed. It is also acknowledged that the invariant is no longer explicitly tested in merkle-tree.spec.ts, but is clearly stated as an assumption in the merkle tree specification.

5.2 Gas Optimization

5.2.1 Removing unnecessary reads of known variables will save gas

Severity: Gas Optimization

Context: SphinxModule.sol#L183, SphinxModule.sol#L267.

Description: The variable `leafSafeProxy` is decoded from `leaf.data` and is subsequently checked to ensure it matches `safeProxy`. Later, within the same function, `safeProxy` is used leading to an unnecessary storage read operation, as the value of `safeProxy` is already accesible at `leafSafeProxy`.

Recommendation: Use `leafSafeProxy` rather than reading the storage again to obtain `safeProxy` address.

Sphinx-Labs: Fixed in commit 7b31fa20.

Spearbit: Fixed.

5.2.2 Caching variables accessed multiple times will save gas

Severity: Gas Optimization

Context: SphinxModule.sol#L273-L403, SphinxModule.sol#L306-L312, SphinxModule.sol#L285-L399

Description: Caching variables that are accessed multiple times within the same function (i.e. `activeMerkleRoot`) will save gas, as each access to a state variable like `activeMerkleRoot` requires a storage read (`SLOAD`).

- `activeMerkleRoot` at SphinxModule.sol#L273-L403 can be stored in a local variable at the start of `execute` function.
- `state.arbitraryChain` at SphinxModule.sol#L306-L312 can be stored in a local variable at L305.
- `state.leavesExecuted` can be stored in a local variable at the very beginning of `execute` and just update it at the very end SphinxModule.sol#L285-L399.

Recommendation: Store state variables into a local variable and access it for all subsequent operations within the function that would access the state variable. This change will reduce the number of `SLOAD` operations, therefore saving gas.

Sphinx-Labs: Fixed in commits b5385431, 71b67448 and 826ae8a0.

Spearbit: Fixed.

5.2.3 `msg.sender` is cheaper gas-wise and provides more clarity than the `executor` variable

Severity: Gas Optimization

Context: SphinxModule.sol#L140, SphinxModule.sol#L150, SphinxModule.sol#L234, SphinxModule.sol#L242, SphinxModule.sol#L253

Description: The `executor` local variable is used in several places after verifying that `executor` is equal to `msg.sender`.

```
require(executor == msg.sender, "SphinxModule: caller isn't executor");
```

Using a local variable for the `executor` when it equals `msg.sender` is inefficient. Additionally, using `msg.sender` provides more clarity on what is going on in the contract logic.

Recommendation: Replace the use of the `executor` local variable with `msg.sender` directly in the contract functions. Since it's already verified that `executor == msg.sender`, using `msg.sender` directly will optimize the contract for gas usage without affecting the logic or security of the contract.

Sphinx-Labs: Fixed in commit 41a13acc.

Spearbit: Fixed.

5.2.4 `SphinxMerkleRootApproved` event can avoid reading from the state

Severity: Gas Optimization

Context: SphinxModule.sol#L136-L143

Description: The `SphinxMerkleRootApproved` event uses a couple of state parameters, namely `merkleRootNonce` and `activeMerkleRoot`, that can be replaced with its user specified counterparts. This is due to having a couple of previous require checks:

```
require(activeMerkleRoot == bytes32(0), "SphinxModule: active merkle root"); // @audit activeMerkleRoot
↪  SLOAD
// ...
require(leafMerkleRootNonce == merkleRootNonce, "SphinxModule: invalid nonce"); //@audit
↪  merkleRootNonce SLOAD
```

This leads to a couple of optimizations

1. **Avoid using `merkleRootNonce`:** Reading `leafMerkleRootNonce` will avoid reading from state and provide the same value.
2. **Avoid using `activeMerkleRoot`:** Given the check `require(activeMerkleRoot == bytes32(0), "SphinxModule: active merkle root");` earlier in the code, `activeMerkleRoot` can be replaced with `bytes32(0)` or removed entirely from the event parameters.

Recommendation: Implement the following changes for gas optimization:

1. Use `leafMerkleRootNonce` to reduce gas costs.
2. Simplify the `SphinxMerkleRootApproved` event by either replacing `activeMerkleRoot` with `bytes32(0)` or removing it if it always equals `bytes32(0)`.

Sphinx-Labs: Fixed in commits 0e91b77c and 1622963f.

Spearbit: Fixed.

5.2.5 Reordering `require` statements can save gas in case of revert

Severity: Gas Optimization

Context: SphinxModule.sol#L92-L94, SphinxModule.sol#L201-L202.

Description: The order of condition checks can be optimized for gas usage. Specifically, checks for input parameters should precede checks that require state variable loading (`SLOAD`). Loading state variables is a more expensive operation compared to checking input parameters.

Recommendation: Swap the order of the condition checks to check first input parameters. This will result in reduced gas costs in scenarios where the function execution reverts due to an invalid input parameter.

```
- require(activeMerkleRoot != bytes32(0), "SphinxModule: no root to cancel");
  require(_root != bytes32(0), "SphinxModule: invalid root");
+ require(activeMerkleRoot != bytes32(0), "SphinxModule: no root to cancel");
```

Sphinx-Labs: Fixed in commit c723fb87.

Spearbit: Fixed.

5.2.6 `merkleRootNonce` increment can be gas optimized and avoid a `SLOAD`

Severity: Gas Optimization

Context: SphinxModule.sol#L153, SphinxModule.sol#L256

Description: `merkleRootNonce` is incremented using `+= 1` operation in `approve` and `cancel` functions. Using an `unchecked { ++merkleRootNonce; }` can offer gas savings as it not expectable to ever overflow.

Additionally, `leafMerkleRootNonce` is equivalent to `merkleRootNonce` as per the previous require statement, therefore an extra `SLOAD` can be avoided by using the local variable `leafMerkleRootNonce` to calculate the final value.

Recommendation: Consider the following change for gas optimizations:

```
- merkleRootNonce += 1;
+ unchecked { merkleRootNonce = ++leafMerkleRootNonce; }
```

Sphinx-Labs: Fixed in commit c6ec37d6. We do this instead:

```
unchecked {
    merkleRootNonce = leafMerkleRootNonce + 1;
}
```

This seems to be slightly cheaper than doing `++leafMerkleRootNonce`, and a little easier to read in my opinion

Spearbit: Fixed.

5.2.7 Custom errors can be used for gas savings

Severity: Gas Optimization

Context: ManagedService.sol#L33, ManagedService.sol#L51, ManagedService.sol#L52, ManagedService.sol#L61, SphinxModuleProxyFactory.sol#L55, SphinxModuleProxyFactory.sol#L73, SphinxModule.sol#L73, SphinxModule.sol#L92, SphinxModule.sol#L94, SphinxModule.sol#L98, SphinxModule.sol#L102, SphinxModule.sol#L107, SphinxModule.sol#L109, SphinxModule.sol#L122, SphinxModule.sol#L123, SphinxModule.sol#L124, SphinxModule.sol#L126, SphinxModule.sol#L127, SphinxModule.sol#L130, SphinxModule.sol#L133, SphinxModule.sol#L201, SphinxModule.sol#L203, SphinxModule.sol#L207, SphinxModule.sol#L211, SphinxModule.sol#L216, SphinxModule.sol#L218, SphinxModule.sol#L230, SphinxModule.sol#L231, SphinxModule.sol#L232, SphinxModule.sol#L233, SphinxModule.sol#L234, SphinxModule.sol#L238, SphinxModule.sol#L275, SphinxModule.sol#L276, SphinxModule.sol#L280, SphinxModule.sol#L284, SphinxModule.sol#L296, SphinxModule.sol#L300, SphinxModule.sol#L302, SphinxModule.sol#L305, SphinxModule.sol#L311, SphinxModule.sol#L352,

Description: The current implementation of error handling in the smart contracts utilizes traditional `require/revert` statements with string messages. Since Solidity 0.8.4, the use of custom errors has been encouraged for gas optimization. Custom errors consume less gas compared to traditional error messages, especially when the same error message is repeated across various functions.

Recommendation: Refactor the existing error handling approach to use custom errors instead of revert statements with string messages. This change will not only reduce the gas costs associated with contract deployment and execution but also enhance the readability and maintainability of the code.

Sphinx-Labs: We aren't going to use custom errors in our contracts. We prefer to use revert messages since they're easier to debug.

Spearbit: Acknowledged.

5.2.8 Duplicate `_safeProxy` zero address check

Severity: Gas Optimization

Context: SphinxModuleProxyFactory.sol#L55

Description: The zero address check on `_safeProxy` is redundant because it will be checked when initializing `SphinxModule`.

```
require(_safeProxy != address(0), "SphinxModule: invalid Safe address");
```

Recommendation: The check in `SphinxModuleProxyFactory` can be removed.

Sphinx-Labs: Fixed in commit 72d3e541.

Spearbit: Fixed.

5.2.9 Redundant variable setters

Severity: Gas Optimization

Context: SphinxModule.sol#L167, SphinxModule.sol#L380

Description: There is no difference between the old and new values set in the referenced lines. Therefore, they are redundant and can be removed.

Recommendation: The referenced lines can be removed, though it would also be best to leave a reason of what their expected values are.

```
- activeMerkleRoot = bytes32(0);

- success = false;
```

Sphinx-Labs: Fixed in commits d71af766 and 62fe62d6.

Spearbit: Fixed.

5.3 Informational

5.3.1 OpenZeppelin's `StandardMerkleTree` allows creating trees with leaves that carry the same information

Severity: Informational

Context: merkle-tree.ts#L289

Description: It is possible to create a tree with multiple leaves that would carry the same raw information when one is using OpenZeppelin's `StandardMerkleTree`. This does not create any issues when one queries the `proof`, but the queried proof picks the path for the last leaf in the leaf array that carries the same information.

```
leafHash(leaf: T): string {
  return hex(standardLeafHash(leaf, this.leafEncoding));
}

leafLookup(leaf: T): number {
  return this.hashLookup[this.leafHash(leaf)] ?? throwError('Leaf is not in tree');
}

getProof(leaf: number | T): string[] {
  // input validity
  const valueIndex = typeof leaf === 'number' ? leaf : this.leafLookup(leaf);
  this.validateValue(valueIndex);

  // rebuild tree index and generate proof
  const { treeIndex } = this.values[valueIndex]!;
  const proof = getProof(this.tree, treeIndex);

  // sanity check proof
  if (!this._verify(this.tree[treeIndex]!, proof)) {
    throw new Error('Unable to prove value');
  }

  // return proof in hex format
  return proof.map(hex);
}
```

This is due to the fact that `this.hashLookup` only saves the last `valueIndex` for the same `hex(standardLeafHash(value, leafEncoding))`.

Recommendation: While this does not affect Sphinx since every leaf has a unique `index` and `chainId` making each leaf's hash different, this should be documented.

Sphinx-Labs: Fixed in commit e0422d82.

Spearbit: Fixed.

5.3.2 Mixed usage of `uint256` and `uint`

Severity: Informational

Context: mekle-tree.ts#L163, mekle-tree.ts#L88

Description: There is mixed usage of `uint` and `uint256` within the file.

Recommendation: This mixed usage of `uint` and `uint256` should be fixed in favor of using only `uint256`.

Sphinx-Labs: Fixed in commit f97e45e8.

Spearbit: Fixed.

5.3.3 Use common base type check function

Severity: Informational

Context: merkle-tree.ts#L252-256

Description: The type check logic is reused twice unnecessarily.

Recommendation: Use a common `isBaseNetworkData` function to deduplicate the type check logic in these lines.

Sphinx-Labs: Fixed in commit 99879f2b.

Spearbit: Fixed.

5.3.4 Use `onlyRole` modifier in `MangedService`

Severity: Informational

Context: ManagedService.sol#L51

Description: The following check could be simplified with the `onlyRole` modifier provided in `AccessControl.sol`. Additionally, this would allow meta transactions to be used, as it uses `_msgSender()` rather than `msg.sender`:

```
require(hasRole(RELAYER_ROLE, msg.sender), "ManagedService: invalid caller");
```

Recommendation: The check should be removed and `onlyRole(RELAYER_ROLE)` should be added as a modifier to the `exec` function.

Sphinx-Labs: Fixed in commit 48b1621e.

Spearbit: Fixed.

5.3.5 Redundant explicit boolean comparisons

Severity: Informational

Context: merkle-tree.ts#L153, merkle-tree.ts#L157

Description: Since `arbitraryApprovalIncluded` and `data.arbitraryChain` are booleans, their explicit comparisons are redundant.

Recommendation: Remove the boolean comparisons.

```
- if (arbitraryApprovalIncluded === true) {
+ if (arbitraryApprovalIncluded) {

- } else if (data.arbitraryChain === true) {
+ } else if (data.arbitraryChain) {
```

Sphinx-Labs: Fixed in commit c1532c40.

Spearbit: Fixed.

5.3.6 Consider adding more documentation to `exec` function and its arbitrary external call

Severity: Informational

Context: ManagedService.sol#L57

Description: The `exec` function includes an arbitrary external call. This raises many questions and thus, requires extra documentation for the sake of clarity. Especially, regarding one of the following questions:

> Is expected to call only deployed smart contracts or also EOA? We expect only to call contracts. - Sphinx-Labs

- The function not only allows to make arbitrary external calls to contracts, but to EOAs. Consider documenting this or adding a check for `<address>.code.length > 0` to avoid EOAs.
- In order to explicitly indicate that `_to` is intended to receive Ether, consider marking the `_to` input parameter as `payable`. It won't change the current behavior, however, is enhances readability as it clearly highlights that the address can handle Ether transactions.

Recommendation: Consider documenting the expected inputs and behaviors as much as possible.

Sphinx-Labs: Fixed in commits 3b1bc452 and b338a733.

Spearbit: Fixed.

5.3.7 Use the already defined `DEFAULT_ADMIN_ROLE` rather than `bytes32(0)` for consistency

Severity: Informational

Context: ManagedService.sol#L34

Description: The contract currently uses a direct `bytes32(0)` value to represent the default admin role. However, OpenZeppelin's Access Control provides a named constant `DEFAULT_ADMIN_ROLE` for this purpose. Using this constant enhances code readability and maintainability.

Recommendation: Consider implementing the following change to improve code readability and maintainability.

```
- _grantRole(bytes32(0), _owner);
+ _grantRole(DEFAULT_ADMIN_ROLE, _owner);
```

Sphinx-Labs: Fixed in commit c8766ffe.

Spearbit: Fixed.

5.3.8 Unnecesary castings can be removed for consistency and clarity

Severity: Informational

Context: SphinxModuleProxyFactory.sol#L95, SphinxModuleProxyFactory.sol#L57.

Description: At `SphinxModuleProxyFactory.sol`, `SPHINX_MODULE_IMPL` is `address` type, however, it is casted to `address` at:

```
Clones.predictDeterministicAddress(addres(SPHINX_MODULE_IMPL), salt, MODULE_FACTORY);
```

This issue is also found in a similar instance of `Clones.cloneDeterministic`.

Recommendation: To improve code consistency and clarity, remove unnecessary `address` casting of `SPHINX_MODULE_IMPL` in the calls to `Clones.predictDeterministicAddress` and `Clones.cloneDeterministic`. While this has no impact on gas usage, it enhances code readability and maintains consistent coding practices.

Sphinx-Labs: Fixed in commit b9b79c71.

Spearbit: Fixed.

5.3.9 Safe owners only sign a typed data hash for the root of a merkle tree and might not know the leaf information

Severity: Informational

Context: SphinxModule.sol#L47, SphinxModule.sol#L178-L183, SphinxModule.sol#L262-L267

Description: When a set of leaves are used to construct a merkle tree for the Sphinx protocol, the users would only sign the `root` and only the `root` of this tree is used in the `EIP-712` typed data signing, i.e., the wallet UIs would only show the `root` information. The safe owners might get phished into signing a malicious root allowing malicious transactions being executed through the Sphinx proxy module.

Creating a different typed structured data for the signing mechanism such as:

```
SphinxMerkleTree {
   SphinxLeaf[] leaves;
}
```

and signing and verifying above on-chain might be expensive for the `approve` endpoint.

Recommendation: The above limitation/issue should be documented as an advisory warning for the users.

Sphinx-Labs:

We plan to add off-chain tooling that lets users view the Merkle leaves that correspond to a Merkle root. Adding the Merkle leaves to the EIP-712 data structure would be prohibitively expensive because the Merkle leaves can exist on many chains and may contain lots of data (e.g. contract creation code).

Spearbit: Acknowledged.

5.3.10 Typographical issues

Severity: Informational

Context: SphinxModule.sol#L125, SphinxModule.sol#L134

Description: Various typos have been identified throughout the SphinxModule contract.

- SphinxModule.sol#L125:

```
- ...because there must always at least be an `APPROVE` leaf.
+ ...because there must always be an `APPROVE` leaf.
```

- SphinxModule.sol#L134:

```
- // We don't validate the `uri` because it we allow it to be empty.
+ // We don't validate the `uri` because we allow it to be empty.
```

Recommendation: Consider resolving the typographical issues.

Sphinx-Labs: Fixed in commits ab9a6849 and 43fd15c3.

Spearbit: Fixed.

5.3.11 Open pragma version is inconsistent and can lead to unexpected behaviors

Severity: Informational

Context: SphinxModuleProxyFactory.sol#L2, ManagedService.sol#L2, SphinxModule.sol#L2

Description: The pragma solidity versions specified in various contract files are inconsistent, using a range (`>=0.7.0 <0.9.0`) rather than a locked version or using an open `^0.8.0` or `^0.8.2` version. This can lead to unexpected behaviors if contracts are deployed with different Solidity versions than the specified while testing. `foundry.toml` file specifies the use of Solidity version 0.8.4, suggesting this as the intended version for contract deployment.

Recommendation: Standardize the pragma solidity directive across all contract files to exactly `0.8.4` (not `^0.8.2` or `>=0.7.0 <0.9.0`). This practice ensures that all contracts are compiled with the intended version, minimizing the risk of unexpected behaviors due to known bugs or behaviors that were changed or fixed as new compiler versions were released.

Sphinx-Labs: Fixed in commit b667ce70.

Spearbit: Fixed.

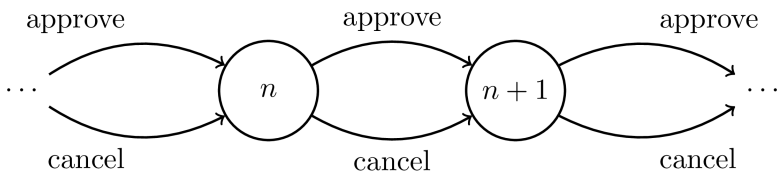
6 Appendix

6.1 State Transitions

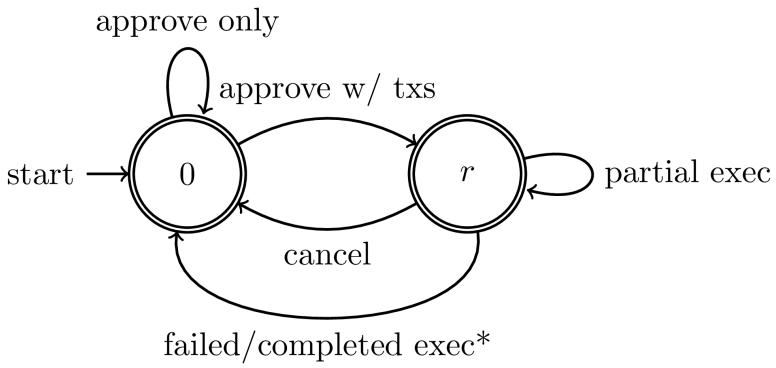
Context: SphinxModule.sol#L57, SphinxModule.sol#L62, SphinxModule.sol#L52

Description:

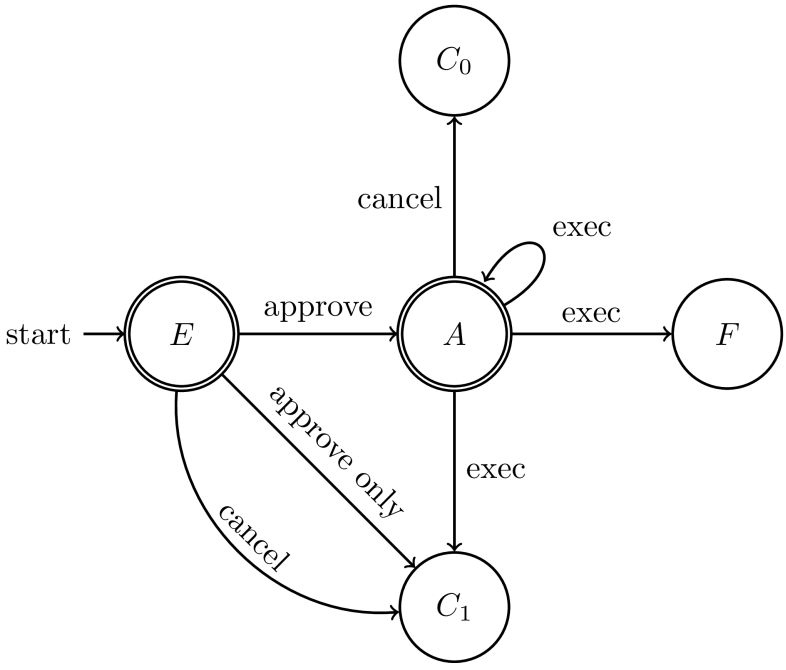
- `merkleRootNonce`:



- `activeMerkleRoot`:



- `merkleRootStates[roo].status`:



symbol	description
E	`EMPTY`
A	`APPROVED`
C_0	`CANCELED`
C_1	`COMPLETED`
F	`FAILED`

Note: It is not possible to do certain status transitions due to the way `activeMerkleRoot` is used. A non-zero `activeMerkleRoot` can only be set when the `status` is `EMPTY` and all the other transitions would reset the `activeMerkleRoot` to `bytes32(0)`.