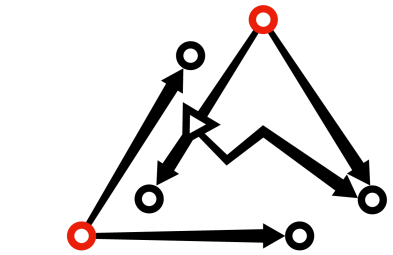

Least Authority

PRIVACY MATTERS

Silent Protocol Smart Contracts

Security Audit Report

Silent Research Labs

Final Audit Report: 2 September 2024

Table of Contents

Overview

Background

Silent Research Labs has requested that Least Authority perform a security audit of their Silent Protocol smart contracts.

Project Dates

- **June 3, 2024 - July 8, 2024:** Initial Code Review *(Completed)*
- **July 10, 2024:** Delivery of Initial Audit Report *(Completed)*
- **September 2, 2024:** Verification Review *(Completed)*
- **September 2, 2024:** Delivery of Final Audit Report *(Completed)*

Review Team

- Nikos Iliakis, Security Researcher and Engineer
- Will Sklenars, Security Researcher and Engineer

Coverage

Target Code and Revision

For this audit, we performed research, investigation, and review of the Silent Protocol smart contracts followed by issue reporting, along with mitigation and remediation instructions as outlined in this report.

The following code repositories are considered in scope for the review:

- Repository:
  https://github.com/Silent-Protocol/smasp-compliance
- Audit scope information:
  https://hackmd.io/m-mHuIFiRamUEp0nrEtR2A

Specifically, we examined the Git revision for our initial review:

- `9fb2b091cdc9c2ebee4a7012063e2c6c83eaca42`

For the verification, we examined the Git revision:

- `0de0db7b269b59dde7a99a234f2fc743e07680ab`

For the review, this repository was cloned for use during the audit and for reference in this report:

- Silent Protocol:
  https://github.com/LeastAuthority/silent-protocol-smasp-compliance/tree/9fb2b091cdc9c2ebee4a7012063e2c6c83eaca42

All file references in this document use Unix-style paths relative to the project's root directory.

In addition, any dependency and third-party code, unless specifically mentioned as in scope, were considered out of scope for this review.

Supporting Documentation

The following documentation was available to the review team:

- Website:
  https://www.silentresearchlabs.org
- Previous audits:
  - https://www.zksecurity.xyz/blog/2023-silent-smart-contracts.pdf
  - https://www.zksecurity.xyz/blog/2023-silent-circuits.pdf
- Yieldfarm_docs.zip *(shared with Least Authority via Discord on 25 June 2024)*
- SilentYieldFarm contract documentation.html *(shared with Least Authority via Discord on 25 June 2024)*

In addition, this audit report references the following documents:

- `Ownable2StepUpgradeable.sol`:
  https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/blob/v4.8.1/contracts/access/Ownable2StepUpgradeable.sol
- `EnumerableMap`:
  https://docs.openzeppelin.com/contracts/3.x/api/utils#EnumerableMap
- Published Notes | Developing Silent Protocol:
  https://hackmd.io/@SilentDAOOfficial

Areas of Concern

Our investigation focused on the following areas:

- Correctness of the implementation;
- Adversarial actions and other attacks on the network;
- Potential misuse and gaming of the smart contracts;
- Attacks that impact funds, such as the draining or manipulation of funds;
- Mismanagement of funds via transactions;
- Denial of Service (DoS) and other security exploits that would impact the intended use of the smart contracts or disrupt their execution;
- Vulnerabilities in the smart contracts' code;
- Protection against malicious attacks and other ways to exploit the smart contracts;
- Inappropriate permissions and excess authority;
- Data privacy, data leaking, and information integrity; and
- Anything else as identified during the initial analysis phase.

Findings

General Comments

The Silent Protocol is a multi-asset shielded pool, with cross-chain capabilities through integration with the Layer Zero Protocol. Silent Protocol facilitates shielded asset transfer of ERC-20 and native tokens.

The protocol features yield-farming capabilities, referred to as `Silent Yield Farm` (SYF), which is designed to generate yield from user funds held in the pool by deploying them to DeFi protocols, such as Athena. The protocol also includes a referral scheme for its users. The yield generated by SYF is not directly paid to users, but rather collected into the `Silent Vault` (`SVault`). Our team noted that SYF could have the potential to expose user funds to risk uncertainty. In the event that SYF positions make a loss, funds can be drawn from `SVault` to provide liquidity for user withdrawals. We recommend that the

Silent Research Labs team seek appropriate legal guidance to ensure their obligations to their users are met with regards to the yield farming aspects of their protocol, along with communicating this risk clearly to their users.

System Design

The `SilentYieldFarm` contract deploys the funds that are trusted to it by Gateway contract. The percentage of the funds that are meant to be invested is calculated in the Gateway contract. For this reason, it uses different instances of the Strategy contract, which is responsible for handling one type of `SYFAsset`. Each strategy may use one or more yield farming protocols in order to generate profit on the funds allocated to it.

Our team examined the design of the Silent Protocol smart contracts, the cross-chain communication, the yield farming strategies, as well as the adapters implemented for the decentralized finance (DeFi) protocols. We found that the implementation relies on a bridge (`layerZero`) for its cross-chain communication.
The smart contracts rely heavily on access control, depending on the roles of Silent protocol, which delegates the security of the system to the safety of these accounts. The users interact only with the Gateway contract (and the `smasp`), which are responsible for calling the corresponding forwarders that use the `layerzero` bridge for the cross-chain communication.

In our review, we examined the smart contracts and the design of the system and identified issues primarily in the `Gateway` contract, and its integration with `Silent Yield Farm`, which may lead to accounting inconsistencies.

We also identified several issues in the withdrawal flow that can result in users losing access to their funds (Issue A, Issue B, Issue C, Issue D, Issue H).

In addition, Silent Protocol utilizes the industry-standard `OpenZeppelin` contracts, such as `Ownable` and `Ownable2Step` to facilitate the transfer of contract ownership. We recommend standardizing on two-step ownership transfer to mitigate the risk of transferring contract ownership to an invalid address (Suggestion 9), and using an upgradability pattern, such as `Ownable2StepUpgradeable` or the Eternal Storage (Suggestion 5).

Code Quality

Our team found that code quality varied throughout the codebase. Parts of the codebase were well-refined, generally organized, and in adherence to Solidity best practices, particularly the core functionality of the Silent Protocol involving zero-knowledge proofs, and the cross-chain integrations with Layer Zero. However, the contracts related to the `Silent Yield Farm` were still under active development at the start of this audit, and our team proceeded with reviewing the target commit hash as bugs were being fixed by the Silent Research Labs team. Due to this, the majority of the findings that our team reported were identified in the `Silent Yield Farm` and the `Gateway` contract that integrates with it. We recommend that the Silent Research Labs team continue to invest developer resources to refactor and simplify the `Gateway` and `Silent Yield Farm` contracts. Continuing to work on these contracts will help the Silent Research Labs Team firm up assumptions about the expected behavior of crucial functions and ensure that the implementation matches the protocol design (Suggestion 20).

Moreover, our team identified several opportunities for gas cost optimizations in the implementation of the smart contracts (Suggestion 10, Suggestion 16, Suggestion 17, Suggestion 22) and documented additional findings that would make functions more gas-efficient, in Appendix A of this report. Additionally, we found opportunities for refactoring redundant code (Suggestion 3, Suggestion 13), as well as

suggestions relating to adherence to best practices (Suggestion 1, Suggestion 4, Suggestion 5, Suggestion 7, Suggestion 8, Suggestion 9, Suggestion 15).

Tests

The repositories in scope include tests that cover the basic functionality of the protocol.

Documentation and Code Comments

Although there was no public documentation for the contracts in scope, the Silent Research Labs team shared different resources with our team, such as flow diagrams and descriptions of functions and variables. The team also shared some of their concerns as notes in the documentation, which helped guide our investigation of the areas of concern outlined in this report. However, our team noted that the documentation consists of brief descriptions and lacks detailed explanations that would help facilitate reasoning about the security properties of the system. We recommend that project documentation be improved to include comprehensive technical and end user documentation (Suggestion 19). Additionally, while the codebase includes some code comments that describe the intended behavior of security-critical components and functions, our team found numerous instances of misleading comments and variable naming, which we recommend be corrected (Suggestion 20). We also further recommend improving code comments (Suggestion 18).

Scope

The scope of this review was sufficient and included all security-critical components. However, due to the number of findings identified, especially in the `Gateway` and `Silent Yield Farm` contracts, we strongly recommend performing a comprehensive, follow-up security audit once the Issues and suggestions in this report have been adequately addressed. We also additionally recommend that the Silent Research Labs team commission a comprehensive security audit of the Silent Tokens contracts (`packages/contracts/contracts/token`).

Dependencies

Our team did not identify any security issues in the use of dependencies. The Silent Research Labs team uses the well-audited `OpenZeppelin` and `layerzero` libraries.

Specific Issues & Suggestions

We list the issues and suggestions found during the review, in the order we reported them. In most cases, remediation of an issue is preferable, but mitigation is suggested as another option for cases where a trade-off could be required.

ISSUE / SUGGESTION	STATUS
Issue A: User Withdrawals Could Be Reverted if Asset idlePercentage Is Zero	Resolved
Issue B: User Withdrawals Could Fail to Access sVault	Resolved
Issue C: User Withdrawal Could Revert Due to Integer Underflow	Resolved
Issue D: Incomplete Availability of Funds in sVault Could Result in Full Revert of User Withdrawals	Unresolved
Issue E: Potential totalLoss Accounting Error	Resolved

Issue F: _withdrawAssetFromSYF May Withdraw Less or More Than the Requested Amount	Resolved
Issue G: _computeReqIdleBalance Can Return Incorrect Value	Resolved
Issue H: Incorrect Comparison Results in Reverted Withdrawal Despite Funds Being Available	Resolved
Issue I: Potential Accounting Error Due To Incorrect require Check	Resolved
Issue J: Funds May Become Locked in Contracts Indefinitely	Resolved
Issue K: Incorrect Function Visibility	Resolved
Issue L: Calling updateUserDepositInfo Followed by revertUpdateUserDepositInfo Results in Accounting Discrepancies	Resolved
Suggestion 1: Replace Deprecated safeApprove Function With approve Function	Resolved
Suggestion 2: Consider Validating idlePercentage Parameter	Resolved
Suggestion 3: Remove Redundant Code	Resolved
Suggestion 4: Declare Storage Variables as Immutable Where Possible	Resolved
Suggestion 5: Consider Implementing Contract Upgradability Pattern	Resolved
Suggestion 6: Consider Extending whitelistToken Function To Allow a Token To Be Removed From the Whitelist	Resolved
Suggestion 7: Add Zero Address Check	Resolved
Suggestion 8: Use an Explicit Address for Ethereum	Resolved
Suggestion 9: Implement Two-Step Ownership Transfer for Ownable Contracts	Resolved
Suggestion 10: Consider Using enumerableMap for finalStrategies	Resolved
Suggestion 11: Consider Extending setDisallowedFunctionCall	Resolved
Suggestion 12: Remove Redundant Code in _ensureBalanceInvariants	Resolved
Suggestion 13: Remove Duplicate Logic	Resolved
Suggestion 14: Set Slippage by Swap	Unresolved
Suggestion 15: Follow Checks-Effects-Interactions Pattern	Resolved
Suggestion 16: Refactor _rebalance To Save Gas	Resolved

Suggestion 17: Utilize Storage Pointers To Save Gas	Resolved
Suggestion 18: Improve Code Comments	Resolved
Suggestion 19: Improve Project Documentation	Resolved
Suggestion 20: Update Function / Variable Naming and Code Comments	Resolved
Suggestion 21: Consider Using call Function Instead of transfer Function	Resolved
Suggestion 22: Declare Constants To Save Gas	Resolved

Issue A: User Withdrawals Could Be Reverted if Asset idlePercentage Is Zero

Location

`packages/contracts/contracts/sp/Gateway.sol#L884`

Synopsis

The function `_ensureAssetBalance` checks if `yfAssets[_token].idlePercentage` is positive before attempting to move funds. However, `idlePercentage` is not the best way to check if there is idle balance in the `Gateway` contract. Considering it is possible to add an asset with `idlePercentage = 0`, there could conceivably be idle balance if the asset has a positive `THRESHOLD` value set.

Impact

A user attempting to make a withdrawal could be denied access to their funds.

Preconditions

`Gateway` should have `yfAsset.idlePercentage == 0 && yfAsset.threshold > 0`.

Feasibility

Misconfiguration by the silent administrator may result in these preconditions being met.

Technical Details

`yfAsset.idlePercentage` is used as a proxy to determine whether the asset has idle liquidity.

`if (yfAssets[_token].idlePercentage > 0) {}`

`yfAsset.idlePercentage` may not be the most effective marker of `idle` liquidity.

Remediation

We recommend directly referencing `idleAssetBalance[_token]` instead.

Status

The Silent Research Labs development team has added a check to ensure that when a new asset is added, it will have a positive `idlePercentage` value.

Verification

Resolved.

Issue B: User Withdrawals Could Fail to Access sVault

Location

packages/contracts/contracts/sp/Gateway.sol#L884

packages/contracts/contracts/sp/Gateway.sol#L901

Synopsis

If `yfAssets[_token].idlePercentage == 0`, the `_ensureAssetBalance` function will not be able to withdraw from `sVault`. This means that users could be blocked from withdrawing their funds, despite there being sufficient funds available in `sVault`.

Impact

A user attempting to make a withdrawal could be denied access to their funds.

Preconditions

`Gateway` should have `yfAsset.idlePercentage = 0`, and `sVault` should have a positive token balance.

Feasibility

Misconfiguration by the silent administrator could result in these preconditions being met.

Technical Details

When users make a withdrawal, they should be able to draw on funds from `Silent Yield Farm`, and from `sVault`, if required. However, access to `sVault` is currently dependent on the condition `yfAssets[_token].idlePercentage > 0`.

Remediation

We recommend that funds from `sVault` be made available for user withdrawals, regardless of the `Silent Yield Farm` configuration. We additionally recommend considering removing the `if` statement:
`if (yfAssets[_token].idlePercentage > 0) {}`.

Status

The Silent Research Labs development team has added a check to ensure that when a new asset is added, it will have a positive `idlePercentage` value. Note that the remediation of Issue A also solves this Issue.

Verification

Resolved.

Issue C: User Withdrawal Could Revert Due to Integer Underflow

Location

packages/contracts/contracts/sp/Gateway.sol#L914

Synopsis

There are potentially situations where the `finalFee` calculation could result in a negative value. As `finalFee` is an unsigned integer, a negative value would cause the transaction to revert.

Impact

A user attempting to make a withdrawal could be denied access to their funds.

Feasibility

Based on the scenario outlined below, the event seems feasible.

Technical Details

`finalFee` is calculated as follows:

```
finalFee = total - _userWithdrawAmount - _computeReqIdleBalance(_token,
assetTotalAUM[_token] -_userWithdrawAmount);
```

`Total` is synonymous with `gatewayAssetBalance` (as noted Appendix A, Item 1). So the calculation could be rewritten as:

```
finalFee = gatewayAssetBalance - _userWithdrawAmount -
_computeReqIdleBalance(_token, assetTotalAUM[_token] -_userWithdrawAmount);
```

Supposing the following preconditions:

```
gatewayAssetBalance = 1000

threshold = 800

idlePercentage = 50%

_userWithdrawAmount = 100
```

Then:

```
_computeReqIdleBalance(_token, assetTotalAUM[_token] -_userWithdrawAmount) =
850
```

- Given `threshold = 800`, and `idlePercentage = 50%`

```
finalFee = 1000 - 100 - 950

      = -50
```

Remediation

We recommend that the Silent Research Labs team further inspect their accounting mechanisms to ensure that users are able to access their funds. If `finalFee` can be negative, that could be indicative of some incorrect assumptions about the underlying behavior of the accounting system.

Status

The Silent Research Labs development team has refactored the calculations and eliminated the underflow case.

Verification

Resolved.

Issue D: Incomplete Availability of Funds in sVault Could Result in Full Revert of User Withdrawals

Location

packages/contracts/contracts/sp/Gateway.sol#L900

Synopsis

When carrying out a user withdrawal, and in the event that the `Gateway` contract renders a liquidity shortfall, the Silent Protocol is designed to draw on funds held in `sVault`. However, if the funds held in `sVault` only cover part of the shortfall, the user withdrawal will revert completely.

Impact

In such a scenario, the user withdrawal transaction will be reverted, despite some funds being made available.

Preconditions

`Gateway` and `sVault` would need to have a combined liquidity amounting to less than the amount the user wants to withdraw.

Feasibility

While this situation should ideally not arise, should Silent protocol face liquidity issues due to economic or other circumstances, the preconditions could be possible.

Technical Details

```
if (_token == address(0x0)) {

  sVault.transferETH(payable(address(this)), lossIncurred);

} else {

  sVault.transferERC20(

      IERC20(_token),

      address(this),

      lossIncurred

  );

}
```

In the aforementioned lines of code, there is no check on the funds available, and should there be a shortfall, the transaction will revert.

Remediation

We recommend checking available funds before attempting the transfer. Should there be insufficient liquidity, we recommend sending a descriptive error message to the user stating how much liquidity is available so that the user can attempt to make a smaller withdrawal.

Status

The Silent Research Labs development team did not yet find a solution and noted that if this Issue arises, they will address it through manual intervention.

Verification

Unresolved.

Issue E: Potential totalLoss Accounting Error

Location

packages/contracts/contracts/sp/Gateway.sol#L984

Synopsis

In the function `_withdrawAssetFromSYF`, `yfAssets[_token].totalLoss` has the potential to be overwritten each time the function is called. However, it appears that `yfAssets[_token].totalLoss` is intended to be a value that accumulates over time.

Impact

If the storage variable is being overwritten when it should actually be getting updated, then the Silent Protocol accounts could end up in an inconsistent state, leading to unexpected behavior.

Remediation

We recommend that the Silent Research Labs team inspect the function to determine whether the current implementation behaves correctly, and if it does not, we recommend updating the code to make the storage variable `yfAssets[_token].totalLoss` accumulative, as follows:

```
yfAssets[_token].totalLoss += totalSyfLoss
```

Status

The Silent Research Labs development team has updated the `totalLoss` calculation to make it accumulative instead of replacing the value on consecutive function calls.

Verification

Resolved.

Issue F: _withdrawAssetFromSYF May Withdraw Less or More Than the Requested Amount

Location

packages/contracts/contracts/sp/Gateway.sol#L968

Synopsis

The `_withdrawAssetFromSYF` function is described as follows:

```
  /// @dev Withdraws funds from SilentYieldFarm contract for a target YFAsset

   /// @param _token token address of the target YFAsset that needs to be
withdrawn

   /// @param _amount amount of _token to be withdrawn

   function _withdrawAssetFromSYF(

       address _token,

       uint256 _amount

   )
```

Our team identified scenarios where the amount withdrawn by the function deviates from the amount supplied, despite funds being available.

Impact

If the function `_withdrawAssetFromSYF` is incorrectly implemented, it could result in accounting errors.

Technical Details

The first scenario concerns the code:

```
if (_amount == 0) {

   totalWithdrawAmount = reqIdleBalance - presentIdleBalance;

} else {

   totalWithdrawAmount = _amount - presentIdleBalance + reqIdleBalance;

}
```

Suppose the following scenario:

```
assetTotalAUM[_token] = 1500

withdraw _amount = 500

newTotalAUM = assetTotalAUM[_token] - 500;

                  = 1000

threshold = 1000

idlePercentage = 50%

presentIdleBalance = 1250

reqIdleBalance = _computeReqIdleBalance(_token, newTotalAUM)

                   = 1000

totalWithdrawAmount = _amount - presentIdleBalance + reqIdleBalance;
```

```
                        = 500 - 1250 + 1000

                        = 250
```

`totalWithdrawAmount = _amount - presentIdleBalance + reqIdleBalance;`
`totalWithdrawAmount = 500 - 1250 + 1000 = 250`

In the scenario presented above, the amount withdrawn in the end is `250` and not the `500` that was requested, despite the funds being available.

In the second scenario, we note that if `_amount = 0`, further logic is carried out that can result in a non-zero amount being withdrawn:

```
if (_amount == 0) {

  totalWithdrawAmount = reqIdleBalance - presentIdleBalance;

}
```

Remediation

The scenarios our team identified show a deviation from the functionality described by the function name and comments. We recommend that the Silent Research Labs team review this function to determine what its expected behavior is. If the scenarios described do in fact conform to the function specification, we recommend updating the function name and comments to more clearly describe the function. If the scenarios described invalidate the function specification, we recommend updating the code accordingly.

Status

The Silent Research Labs development team has significantly changed the `_withdrawAssetFromSYF` function by reducing it from 41 lines to 22 lines. In the new version, the two scenarios raised are no longer an Issue.

Verification

Resolved.

Issue G: _computeReqIdleBalance Can Return Incorrect Value

Location

[packages/contracts/contracts/sp/Gateway.sol#L924]

Synopsis

if `newTotalAUM` is below the threshold, the function `_computeReqIdleBalance` returns `idleBalance = 0`. However, in this scenario, the `idleBalance` returned should be equal to `newTotalAUM`, or `100%` of the assets under management.

Impact

If the function `_computeReqIdleBalance` returns `0` instead of the correct non-zero value, the Silent Protocol could potentially end up over-investing user funds, violating the protocol's idle liquidity targets.

Preconditions

`newTotalAUM <= yfAsset.threshold`

Feasibility

In the event of a series of user withdrawals, `newTotalAUM` could conceivably drop below `yfAsset.threshold`, resulting in the incorrect calculation.

Technical Details

The Issue concerns the following code block. In this case, should `newTotalAUM` be less than `yfAsset.threshold`, `reqIdleBalance` would be set to `0`. `reqIdleBalance` is then returned by the function, as follows:

```
if (newTotalAUM >= yfAsset.threshold) {

  difference = newTotalAUM - yfAsset.threshold;

  reqIdleBalance =

      yfAsset.threshold +

      (yfAsset.idlePercentage * difference) /

      DENOMINATOR;

} else {

  difference = 0;

  reqIdleBalance = 0;

}
```

Remediation

We recommend updating the function as follows:

```
function _computeReqIdleBalance(

  address _token,

  uint256 newTotalAUM

) internal view returns (uint256) {

  YFAsset memory yfAsset = yfAssets[_token];


  uint256 difference;

  uint256 reqIdleBalance;

  // if newTotalAUM < threshold :: we can't maintain the threshold amount in
Gateway

  if (newTotalAUM >= yfAsset.threshold) {

      difference = newTotalAUM - yfAsset.threshold;
```

```
        reqIdleBalance =
        yfAsset.threshold +
        (yfAsset.idlePercentage * difference) /
        DENOMINATOR;
        return reqIdleBalance
    }

    return newTotalAUM;
}
```

Status

The Silent Research Labs development team has implemented the proposed remediation.

Verification

Resolved.

Issue H: Incorrect Comparison Results in Reverted Withdrawal Despite Funds Being Available

Location

packages/contracts/contracts/sp/Gateway.sol#L955

Synopsis

In the function `_withdrawAssetFromSYF`, there is a conditional checking that the total assets under management is greater than the amount required by the withdrawal.

Impact

The function `_withdrawAssetFromSYF` could revert, despite funds being available. This could result in user withdrawals being unnecessarily reverted.

Preconditions

The amount being withdrawn would have to be equal to the total assets under management:

```
assetTotalAUM[_token] == _amount
```

Feasibility

This exact scenario seems unlikely to occur.

Technical Details

The following comparison resolves to `false` if there is exactly `_amount` or assets under management, even though there are sufficient funds to complete the withdrawal:

```
if (assetTotalAUM[_token] > _amount) {
```

Remediation

We suggest using the >= (greater than or equal to) operator, as follows:

```
if (assetTotalAUM[_token] >= _amount) {
```

Status

The Silent Research Labs development team has implemented the proposed remediation.

Verification

Resolved.

Issue I: Potential Accounting Error Due To Incorrect require Check

Location

packages/contracts/contracts/sp/Gateway.sol#L971-981

Synopsis

An incorrect comparison in a `require` statement could lead to accounting errors. Our team also identified a potential opportunity for refactoring and simplifying the `_withdrawAssetFromSYF` function.

Technical Details

```
uint256 balanceBefore = _checkAssetBalance(_token, address(this));

totalAmountWithdrawn = ISYF(syfContract).withdrawAssetToGateway(

  _token,

  totalWithdrawAmount

);

uint256 balanceAfter = _checkAssetBalance(_token, address(this));


require(

  totalAmountWithdrawn >= balanceAfter - balanceBefore,

  "Balance change mismatch"

);
```

The above code suggests that `totalWithdrawAmount` could be different than `totalAmountWithdrawn`, both of which could be different to what actually ends up being withdrawn, as calculated by `balanceAfter - balanceBefore` (hereinafter referred to as the variable `actualAmountWithdrawn`).

Function execution is allowed to continue if the `actualAmountWithdrawn` is smaller than the `totalAmountWithdrawn` variable, as dictated by the `require`.

Additionally, the contract accounts are updated based on the `totalAmountWithdrawn` variable, rather than the `actualAmountWithdrawn` variable:

`idleAssetBalance[_token] += totalAmountWithdrawn;`

This could potentially lead to accounting errors if

`actualAmountWithdrawn < totalAmountWithdrawn`.

Remediation

We recommend updating `idleAssetBalance[_token] += totalAmountWithdrawn` to `idleAssetBalance[_token] += actualAmountWithdrawn`.

However, generally speaking, `_withdrawAssetFromSYF` has the potential to be refactored and simplified to reduce potential accounting errors and attack vectors. For example, if `totalAmountWithdrawn` being different to `actualAmountWithdrawn` is an incorrect assumption, then the `balanceAfter - balanceBefore` logic is redundant.

Status

The Silent Research Labs development team has significantly changed the `_withdrawAssetFromSYF` function by reducing it to 22 lines, from the original 41 lines. In the new version, this Issue is not present.

Verification

Resolved.

Issue J: Funds May Become Locked in Contracts Indefinitely

Location

All contracts.

Synopsis

For non-upgradable contracts, we recommend implementing a way to easily retrieve funds held within the contract. Funds can end up being held by a contract if a user mistakenly sends funds to payable functions in the contract. Additionally, accounting errors can result in a contract holding funds that were intended to be passed on to other accounts. This is less of a concern with upgradable contracts, as arbitrary fund recovery logic can be pushed at a later date, if required.

Impact

Without a recovery mechanism, funds held by a contract account can be difficult or impossible to retrieve, and could be locked in the contract indefinitely.

Remediation

We recommend that each smart contract implement a simple withdrawal mechanism, protected by access control. For example:

```
function recoverFunds(address token, uint256 amount) external onlyOwner {
  if (token == address(0)) {
      // Recover Ether
      uint256 balance = address(this).balance;
      require(amount == 0 || amount <= balance, "Insufficient Ether balance");
```

```
        uint256 amountToSend = amount == 0 ? balance : amount;
        (bool success, ) = owner().call{value: amountToSend}("");
        require(success, "Ether transfer failed");
        emit FundsRecovered(address(0), amountToSend);
    } else {
        // Recover ERC20 tokens
        IERC20 tokenContract = IERC20(token);
        uint256 balance = tokenContract.balanceOf(address(this));
        require(amount == 0 || amount <= balance, "Insufficient token balance");
        uint256 amountToSend = amount == 0 ? balance : amount;
        require(tokenContract.transfer(owner(), amountToSend), "Token transfer
failed");
        emit FundsRecovered(token, amountToSend);
    }
}
```

Status

The Silent Research Labs development team has implemented the functionality, which enables the recovery of funds from all contracts.

Verification

Resolved.

Issue K: Incorrect Function Visibility

Location

packages/contracts/contracts/sp/Gateway.sol

Synopsis

The `_ensureAssetBalance` function is meant to be used only as an internal function in the protocol. However, it is currently set as `public`. This deviates from recommended best practices, as it allows users to trigger it.

Impact

A malicious user will not be able to manipulate the function for their own benefit; however, they could prevent the protocol from functioning as intended.

Preconditions

No preconditions are necessary.

Remediation

We recommend updating the function to make it `internal`, as it was intended to be.

Status

The Silent Research Labs development team has changed the function visibility to `internal`.

Verification

Resolved.

Issue L: Calling updateUserDepositInfo Followed by revertUpdateUserDepositInfo Results in Accounting Discrepancies

Location

contracts/contracts/referral/ReferralController.sol#L214

contracts/contracts/referral/ReferralController.sol#L240

Synopsis

Calling `updateUserDepositInfo` and then `revertUpdateUserDepositInfo` in a following transaction will cause storage variables to be incorrectly reverted, due to fluctuating exchange rates.

Impact

Calling the `revert` function does not perform a true `revert`.

Preconditions

For the inconsistency to arise, the exchange rate between `eth` and the asset in question must have changed.

Feasibility

The scenario is likely to occur every time `revertUpdateUserDepositInfo` is called.

Technical Details

The function `updateUserDepositInfo` has the following signature:

```
function updateUserDepositInfo(
  address user,
  address tokenAddress,
  uint256 amount
)
```

However, accounting updates are based on the `eth` value of the supplied asset and amount at the time the transaction executes:

```
uint256 ethAmount = _getAmountInEth(tokenAddress, amount);
deposits[referrer][epochNumber] += ethAmount;
```

```
totalDeposits[epochNumber] += ethAmount;
```

`revertUpdateUserDepositInfo()` has the same function signature.

```
function revertUpdateUserDepositInfo(
  address user,
  address tokenAddress,
  uint256 amount
)
```

Additionally, the function performs the same exchange rate query:

```
uint256 ethAmount = _getAmountInEth(tokenAddress, amount);
```

And then performs the account updates in reverse :

```
deposits[referrer][epochNumber] -= ethAmount;
totalDeposits[epochNumber] -= ethAmount;
```

This Issue occurs when the function `revertUpdateUserDepositInfo` is called some time after `updateUserDepositInfo`, when the exchange rate has changed.

Remediation

For the `revert` to affect the accounts in a consistent manner, we recommend that the contract keep a state object mapping the value in `eth` associated with each call to `updateUserDepositInfo`, such that the correct value in `eth` can be referenced by `revertUpdateUserDepositInfo`.

Status

The Silent Research Labs development team has updated the function `updateUserDepositInfo`. It currently returns `ethAmount`, which is passed to SMASP so that the value can be used when calling `revertUpdateUserDepositInfo`.

Verification

Resolved.

Suggestions

Suggestion 1: Replace Deprecated safeApprove Function With approve Function

Location

packages/contracts/contracts/sp/Gateway.sol#L859

packages/contracts/contracts/yieldFarm/SilentYieldFarm.sol#L533

packages/contracts/contracts/yieldFarm/mocks/MockGateway.sol#L347

Synopsis

The `safeApprove` function was added to `OpenZeppelin` as a solution to a front running vulnerability associated with the `approve` function. However, the `safeApprove` function has issues of its own, which are similar to the issues relating to the `approve` function. Consequently, it has the potential to convey a false sense of security. Because of this, it was deprecated by `OpenZeppelin` that recommended the use of the `approve` function instead.

Mitigation

We recommend using the `approve` function instead of the deprecated `safeApprove` function.

Status

The Silent Research Labs development team has replaced instances of `safeApprove` with `approve`.

Verification

Resolved.

Suggestion 2: Consider Validating idlePercentage Parameter

Location

packages/contracts/contracts/sp/Gateway.sol#L779

Synopsis

When adding an asset with the function `addYFAsset`, the code checks to see if the `idle` percentage is zero, reporting the error `yfAsset` already set if it is non-zero:

```
require(yfAsset.idlePercentage == 0, "yfAsset already set");
```

This suggests that `idlePercentage` is potentially regarded as a proxy for the asset `yfAsset` being added to the system. However, the function `addYFAsset` does not check whether the argument `_yfAsset.idlePercentage` is non-zero before setting the value. This could result in a situation where an asset has already been added (`idlePercentage == 0`), and is then able to be added again, overwriting the original asset.

Mitigation

If appropriate, we recommend considering adding the following input validation:

```
require(_yfAsset.idlePercentage > 0, "Invalid idle percentage");
```

Status

The Silent Research Labs development team has added validation to ensure that `idlePercentage` is non-zero.

Verification

Resolved.

Suggestion 3: Remove Redundant Code

Location

packages/contracts/contracts/sp/Gateway.sol#L940

packages/contracts/contracts/yieldFarm/strategies/ethenaStrategy/EthenaStrategy.sol#L269

packages/contracts/contracts/yieldFarm/SilentYieldFarm.sol#L661

Synopsis

Some lines of code do not have any effect on program execution and can be deleted.

Mitigation

We recommend deleting the aforementioned lines of code.

Status

The Silent Research Labs development team has removed the redundant lines of code.

Verification

Resolved.

Suggestion 4: Declare Storage Variables as Immutable Where Possible

Location

packages/contracts/contracts/referral/ReferralController.sol#L69

packages/contracts/contracts/referral/ReferralRewardPool.sol#L27

packages/contracts/contracts/sp/Gateway.sol#L20

packages/contracts/contracts/sp/Gateway.sol#L79

packages/contracts/contracts/yieldFarm/SilentYieldFarm.sol#L32

Synopsis

`Immutable` variables are more gas-efficient to read than regular `storage` variables. Any variable that is set only once during contract initialization can be defined as `immutable`.

Mitigation

We recommend declaring these variables as immutable. For example:
`uint256 public immutable someVariable;`

Status

Where appropriate (considering the new updates), the Silent Research Labs development team has marked the aforementioned variables as `immutable`.

Verification

Resolved.

Suggestion 5: Consider Implementing Contract Upgradability Pattern

Synopsis

Some of the Silent Protocol contracts have considerable complexity. For complex smart contracts, having upgradability can be useful in the event that a vulnerability is found in production code, or if the protocol needs to be updated to suit a new usage pattern.

Mitigation

We recommend that the Silent Research Labs team consider implementing contract upgradability. This would provide a mitigation path in the event that a bug is found once the system has been deployed. Some options include the proxy pattern, for which `OpenZepplin`'s upgradable contracts could be used, or the Eternal Storage pattern.

Status

The Silent Research Labs development team is currently using OpenZeppelin's `UUPSUpgradeable`.

Verification

Resolved.

Suggestion 6: Consider Extending whitelistToken Function To Allow a Token To Be Removed From the Whitelist

Location

`contracts/contracts/referral/ReferralController.sol#L401`

Synopsis

Currently, a token can be whitelisted but cannot be removed from the whitelist.

Mitigation

We recommend that the Silent Research Labs team consider whether it could potentially be useful to be able to remove a token from the whitelist, and if so, we recommend extending the function. More specifically, the function `whitelistToken` could be extended to receive a Boolean as the second parameter:

```
function whitelistToken(address tokenAddress, bool whitelisted) external
onlyOwner {

  whitelistedToken[tokenAddress] = whitelisted;

  emit TokenWhitelisted(tokenAddress);

}
```

Status

The Silent Research Labs development team has extended `whitelistToken` to accept a Boolean parameter to allow delisting.

Verification

Resolved.

Suggestion 7: Add Zero Address Check

Location

For a complete list of locations, see Appendix B.

Synopsis

We recommend checking for the zero address for all mutating external-facing functions. Checks should be performed on address arguments that are not meant to be the zero address. For example, addresses that are to receive a payment should be checked for the zero address, while token addresses should not be checked, as they can legitimately be the zero address in the case of native tokens. Checking for the zero address can help prevent unexpected outcomes, such as tokens mistakenly being burned, or the protocol being incorrectly configured.

Mitigation

We recommend checking all mutating external functions, such as the ones listed in Appendix B, and implementing zero address checks where appropriate.

Status

The Silent Research Labs development team has added extensive zero address checks.

Verification

Resolved.

Suggestion 8: Use an Explicit Address for Ethereum

Location

Multiple locations throughout the codebase.

Synopsis

The protocol supports different ERC20 tokens and Ether. The protocol checks the token address passed to its functions for verification (whitelisted etc.) and assumes that if the zero address is passed, then it has to handle a transaction related to Ether. However, every empty variable falls to the zero value in the EVM ecosystem (uninitialized variables etc.). Although our team did not identify any issues in the codebase related to this finding, this practice is prone to mistakes.

Mitigation

We recommend setting a dedicated address and using it when interacting with the protocol. For example, the following could be an option:

```
public constant ETH = 0xEeeeeEeeeEeEeeEeEeEeeEEEeeeeEeeeeeeeEEeE
```

Status

The Silent Research Labs development team has updated the contracts, such that they currently use `0xEeeeeEeeeEeEeeEeEeEeeEEEeeeeEeeeeeeeEEeE` for the ETH address.

Verification

Resolved.

Suggestion 9: Implement Two-Step Ownership Transfer for Ownable Contracts

Location

packages/contracts/contracts/referral/ReferralController.sol#L29

contracts/contracts/referral/ReferralRegistry.sol#L15

contracts/contracts/referral/ReferralRewardPool.sol#L13

packages/contracts/contracts/sp/Gateway.sol#L14

packages/contracts/contracts/sp/Points.sol#L7

contracts/contracts/sp/SMASPForwarder.sol#L10

packages/contracts/contracts/sp/SmaspGatewayForwarder.sol#L10

packages/contracts/contracts/yieldFarm/SilentYieldFarm.sol#L13

contracts/contracts/yieldFarm/adapters/SilentAdapterBase.sol#L11

contracts/yieldFarm/strategies/SilentBaseStrategyV1.sol#L30

Synopsis

Having a two-step claimable ownership reduces the risk of transferring ownership to an invalid address.

Mitigation

We recommend using `OpenZepplin`'s `Ownable2Step.sol` instead of `Ownable.sol`.

Status

The Silent Research Labs development team has implemented two-step ownership transfer for all contracts.

Verification

Resolved.

Suggestion 10: Consider Using enumerableMap for finalStrategies

Location

contracts/yieldFarm/SilentYieldFarm.sol

Synopsis

The Silent Research Labs team uses an array for storing final strategies. The team performs some costly operations on this array, such as the `contains` function, which uses a `for loop` to iterate the array, or some re-arrangement. These kinds of operations consume a considerable amount of gas.

Mitigation

We recommend considering using EnumerableMap, which is better suited for the use case of the Silent Research Labs team, as entries can be added, removed, and checked for existence in O(1). Note that, as

stated in the documentation, no guarantees are made on the ordering of the objects in the `EnumerableMap`.

Status

The Silent Research Labs development team chose not to implement the suggestion and stated that they need the ordering of strategies for withdrawals and deposit priorities, and `enumerableMapping` does not facilitate this requirement. Our team agrees with the development team's response and thus considers this suggestion resolved.

Verification

Resolved.

Suggestion 11: Consider Extending setDisallowedFunctionCall

Location

contracts/contracts/yieldFarm/adapters/SilentAdapterBase.sol#L34

Synopsis

Currently, although a function call can be disallowed using `setDisallowedFunctionCall`, it can be reinstated.

Mitigation

We recommend that the Silent Research Labs team consider whether it could potentially be useful to be able to reenable a function call, and if so, we recommend extending the function. More specifically, the function `setDisallowedFunctionCall` could be extended to receive a Boolean as the second parameter, and also renamed.

```
function setFunctionCallAllowed(
  bytes4 _functionSelector,
  bool allowed
) external onlyOwner {
  require(isAllowedFunction[_functionSelector] != allowed, "No state change");
  isAllowedFunction[_functionSelector] = allowed;
}
```

Status

The Silent Research Labs development team has renamed the aforementioned function to `setFunctionCallAllowed` and extended it to accept a Boolean parameter.

Verification

Resolved.

Suggestion 12: Remove Redundant Code in _ensureBalanceInvariants

Location

packages/contracts/contracts/yieldFarm/SilentYieldFarm.sol#L871

Synopsis

```
uint256 syfAssetTotalStrategyDebt_cumulative;

uint256 syfAssetRealStrategyDebt_cumulative;

…

require(

      strategyContractDebt == strategySyfDebt,

      "strategyContractDebt != strategySyfDebt"

);

syfAssetTotalStrategyDebt_cumulative += strategySyfDebt;

syfAssetRealStrategyDebt_cumulative += strategyContractDebt;


require(syfAssetRealStrategyDebt_cumulative ==
syfAssetTotalStrategyDebt_cumulative, "Real strategy debt cumulative != SYF
strategy debt cumulative");
```

In the aforementioned lines of code, `syfAssetTotalStrategyDebt_cumulative` and `syfAssetRealStrategyDebt_cumulative` appear to move in lockstep, as enforced by `require("strategyContractDebt != strategySyfDebt")`. Therefore, it is redundant to have both variables, and the last `require` statement is also redundant.

Mitigation

The function can be simplified by removing a redundant variable and redundant `require`.

Status

The Silent Research Labs development team has simplified the function to remove the redundant code.

Verification

Resolved.

Suggestion 13: Remove Duplicate Logic

Location

packages/contracts/contracts/yieldFarm/SilentYieldFarm.sol#L400

packages/contracts/contracts/yieldFarm/SilentYieldFarm.sol#L486

Synopsis

The functions `rebalance` and `depositAssetToStrategies` are functionally the same.

Mitigation

As only one function is required, we recommend refactoring to remove duplicate logic.

Status

The Silent Research Labs development team has removed the `rebalance` function.

Verification

Resolved.

Suggestion 14: Set Slippage by Swap

Location

`contracts/contracts/yieldFarm/strategies/SilentBaseStrategyV1.sol`

Synopsis

In the `strategy` contracts, the Silent Research Labs team sets a universal slippage tolerance threshold for all swaps under the strategy.

Mitigation

We recommend allowing slippage to be passed as a parameter in each swap, as it would be better to be able to parametrize it to each specific case.

Status

The Silent Research Labs development team has acknowledged the finding but stated that they want to ensure a maximum swap threshold and cannot depend upon external information only. As a result, at the time of the verification, the suggested mitigation has not been resolved.

Verification

Unresolved.

Suggestion 15: Follow Checks-Effects-Interactions Pattern

Location

`packages/contracts/contracts/yieldFarm/SilentYieldFarm.sol#L518`

`packages/contracts/contracts/yieldFarm/SilentYieldFarm.sol#L431`

`packages/contracts/contracts/yieldFarm/SilentYieldFarm.sol#L304`

`packages/contracts/contracts/sp/SMASP.sol#L694`

- call `_setBalances` before `_commitWithdraw`

`packages/contracts/contracts/sp/Gateway.sol#L322`

- move `_sendFunds` to bottom of the function

Synopsis

Where possible, best practice recommends following the checks-effects-interactions pattern to minimize potential unknown attack vectors.

Mitigation

We recommend updating the internal contract state before calling external contracts, as follows:

```
_depositAssetToStrategy():

strategy.totalDebt += _amount;

syfAsset.totalStrategyDebt += _amount;

strategy.lastHarvest = block.timestamp;

// IERC20 calls bellow
```

Status

The Silent Research Labs development team has implemented the mitigation, as recommended.

Verification

Resolved.

Suggestion 16: Refactor _rebalance To Save Gas

Location

packages/contracts/contracts/yieldFarm/SilentYieldFarm.sol#L682

Synopsis

There are some minor modifications that can be made to the function `_rebalance` to save some gas.

Mitigation

We recommend moving the first `require` up to line 686 for an early `revert`.

Additionally, the memory pointer `syfAsset` is set on line 683. However, parts of the code still reference the storage variable. We recommend using the memory pointer wherever possible (e.g., for lines 685 and 693).

Status

The Silent Research Labs development team has refactored the function and made it more gas efficient.

Verification

Resolved.

Suggestion 17: Utilize Storage Pointers To Save Gas

Location

packages/contracts/contracts/yieldFarm/SilentYieldFarm.sol#L169

- Pointer for `syfAssets[_asset]`

packages/contracts/contracts/yieldFarm/SilentYieldFarm.sol#L644

- Pointer for `assetStrategies[_asset][_strategy]`

packages/contracts/contracts/sp/SMASP.sol#L255

- Pointer for `userBalance[_assetId]`

packages/contracts/contracts/sp/Points.sol#L276

packages/contracts/contracts/sp/Points.sol#L291

packages/contracts/contracts/sp/Points.sol#L245

packages/contracts/contracts/sp/Gateway.sol#L307

- Pointers for `assets[asset]`

Synopsis

`Storage` pointers can be used to avoid repeated reads of `storage` variables and cut down on gas costs.

Mitigation

We recommend declaring `Storage` pointers whenever repeated `storage` variable reads occur.

Status

The Silent Research Labs development team has added several storage pointers.

Verification

Resolved.

Suggestion 18: Improve Code Comments

Synopsis

Our team generally found code commenting to be somewhat limited. Some of the function comments describe the function name and parameters, but do not explain the function intent or greater context.

Mitigation

We recommend that the Silent Research Labs team continue to improve their code commenting. For complex functions, we recommend adding comments that explain the reasoning behind calculations and state updates.

Status

The Silent Research Labs development team has added several new code comments and also updated existing ones.

Verification

Resolved.

Suggestion 19: Improve Project Documentation

Synopsis

Robust and comprehensive documentation allows a security team to assess the in-scope components and understand the expected behavior of the system being audited. In addition, clear and concise user documentation provides users with a guide to utilize the application according to security best practices.

Mitigation

We recommend that the Silent Research Labs team improve the project's general documentation by creating a high-level description of the system, each of the components, and the interactions between those components. This can include developer documentation and architectural diagrams.

In addition, we recommend that comprehensive user documentation be created to help users interact with the system correctly and as intended, which encourages secure and correct usage. We also advise the Silent Research Labs team to disclose the risk associated with using the protocol to the users.

Status

The Silent Research Labs development team has added several new sections of project documentation.

Verification

Resolved.

Suggestion 20: Update Function / Variable Naming and Code Comments

Location

Throughout the codebase, as detailed in Appendix A.

Synopsis

Clear naming conventions and descriptive comments help make the code easier to read. They can also make logical errors easier to spot.

Mitigation

We recommend updating some comments as well as the naming of some functions and variables, according to the suggestions outlined in Appendix A.

Status

The Silent Research Labs development team has implemented the mitigation, as recommended.

Verification

Resolved.

Suggestion 21: Consider Using call Function Instead of transfer Function

Location

`packages/contracts/contracts/yieldFarm/SilentYieldFarm.sol`

Synopsis

The Solidity `transfer` function allows only `2300` gas. In the event that the receiving account is a smart contract, the transaction may run out of gas and revert. Failure could occur if the receiving contract is an

upgradable contract, or if the receiving contract's fallback function has custom logic. By default, the `call` function allows the maximum gas available, but the `call` function can also be passed a gas allowance as an argument.

Mitigation

We recommend checking all occurrences of the `transfer` function and investigating the likelihood that the recipient may be a smart contract. If it is possible that the recipient could be a smart contract, we recommend using the `call` function instead. Note that if the `call` function is used, extra caution should be taken to protect against re-entrancy.

Status

The Silent Research Labs development team is currently using the internal function `_transferFunds` for most cases where funds are transferred to external entities. The `_transferFunds` function utilizes `call`.

Verification

Resolved.

Suggestion 22: Declare Constants To Save Gas

Location

packages/contracts/contracts/sp/SMASP.sol#L374

Synopsis

Variations on the following code appear multiple times throughout `SMASP.sol`:

```
// set the offset

uint256 offset = ANONPOOLSIZE * 4 + 2;
```

Mitigation

Since offset can be known at compile time, instead of performing calculations at runtime, we recommend declaring contract-level constants and referencing them in the functions, as follows:

```
uint256 private constant OFFSET_SUBSCRIBE = ANONPOOLSIZE * 4;

uint256 private constant OFFSET_DEPOSIT_WITHDRAW = ANONPOOLSIZE * 4 + 2;

uint256 private constant OFFSET_TRANSFER = ANONPOOLSIZE * 4 + 3;

uint256 private constant OFFSET_TRANSFER_NON_SILENT = ANONPOOLSIZE * 4 + 4;
```

Status

The Silent Research Labs development team has modified the code to utilize constants.

Verification

Resolved.

Appendix

Appendix A: Comprehensive List of Variable/Function Names and Comments That Should Be Updated

The following is a comprehensive list of all the variable names that should be updated to increase code legibility:

Item 1

packages/contracts/contracts/sp/Gateway.sol#L891

```
uint256 total = gatewayAssetBalance + amountWithdrawn;
```

Here, the name `total` does not accurately convey what the total represents.

Based on the inner logic of `_withdrawAssetFromSYF`, it appears that `total` should be equal to the new value of `idleAssetBalance[_token]`) (i.e., `gatewayAssetBalance`).

To better convey the semantics of the code, we recommend updating the already defined variable `gatewayAssetBalance`:

```
gatewayAssetBalance = gatewayAssetBalance + amountWithdrawn;
```

Item 2

packages/contracts/contracts/sp/Gateway.sol#L924

The function `_computeReqIdleBalance` could be more clearly named `_computeReqiredIdleBalance`, as `req` could refer to multiple, different words (e.g., request).

Item 3

contracts/yieldFarm/strategies/ethenaStrategy/EthenaStrategy.sol#L461

```
uint256 estTotalUnderlyingInvested =
_estimatedTotalUnderlyingInvested(_adapter, _swapParams);

if (estTotalUnderlyingInvested > totalUnderlyingInvested) {}
```

These lines of code could be made more clear by renaming `estTotalUnderlyingInvested` and `_estimatedTotalUnderlyingInvested` to `estimatedStragegyEquity` and `estimateStragegyEquity` to differentiate them from the value `totalUnderlyingInvested`.

Item 4

packages/contracts/contracts/yieldFarm/interfaces/IStrategy.sol#L22

The term 'Dynamic' has a different meaning elsewhere in the codebase. We recommend that the Silent Research Labs team consider whether the use of 'Dynamic' adds value here, and if not, we recommend renaming `DynamicAdapterSwapParams` to `AdapterSwapParams`.

Item 5

packages/contracts/contracts/yieldFarm/SilentYieldFarm.sol

In `SilentYieldFarm`, naming conventions differ between functions. Having a consistent nomenclature improves legibility and helps with identifying logical errors.

- In [SilentYieldFarm.sol#L601]:

  `Empty Withdrawal Stack` could be renamed to `Empty Strategy Stack`.

- In [SilentYieldFarm.sol#L44]:

  ```
  mapping(address => address[MAX_STRATEGIES]) public strategyStack;
  ```

  The mapping is from `address` to `strategyStack`, so the mapping could be pluralized:

  ```
  mapping(address => address[MAX_STRATEGIES]) public strategyStacks
  ```

- In [SilentYieldFarm.sol#L811]:

  `_inStrategyArray` could be called from `_inStrategyStack`

  `_removeFromStrategyArray` could be called from `_removeFromStrategyStack`

- In [SilentYieldFarm.sol#L438] (and other locations within `SilentYieldFarm.sol`):

  `strategyInfo` could be renamed to `strategy`, as it refers to a `Strategy` data type.

- In [SilentYieldFarm.sol#L418] (and other locations within `SilentYieldFarm.sol`):

  Anywhere that a `strategy` address is referenced, it could be called `strategyAddress`, rather than `strategy`, to differentiate it from the `Strategy` data type.

Item 6

[packages/contracts/contracts/yieldFarm/SilentYieldFarm.sol#L246]

We recommend refactoring `setStrategyStack`. In the aforementioned location, some of the variable names and comments may be misleading (e.g., `strategies` is referred to as assets). From conversations with the Silent Research Labs team, it became apparent that there may be some uncertain assumptions regarding `setStrategyStack`, and it may need to be updated to match changes in other parts of the codebase.

As a starting point, the Silent Research Labs Team can utilize the refactored function our team provided in the shared Discord channel. It features updated nomenclature, comments, and some simplified logic. The Silent Research Labs Team can reference the Discord thread for other points to consider, such as the loop being potentially redundant.

Suggested variable renaming for the refactored version:

- rename `strategies` to `currentStrategyStack`
- rename `strMap` to `strategies`
- rename `finalStrategies` to `finalStrategyStack`
- rename `currentAsset` to `strategyAddress`
- rename `oldAsset` to `currentStrategyAddress`
- rename `_stack` to `_newStrategyStack`
- rename `allocPointsSum` to `totalAllocPoints`

Item 7

Throughout the codebase, we recommend renaming `totalAllocPoint` to the plural `totalAllocPoints`.

Item 8

contracts/yieldFarm/strategies/ethenaStrategy/EthenaStrategy.sol#L298

The function `/// @dev` is triggered when the `prepareWithdrawToSYF` function is called with the `withdrawAll` parameter set to `true`.

The `prepareWithdrawToSYF` function does not have a `withdrawAll` parameter, as it is a computed property. We recommend updating the comment.

Item 9

packages/contracts/contracts/yieldFarm/SilentYieldFarm.sol#L336

Some of the comments in `withdrawAssetToGateway` are not correct and should be removed or updated:

1. The status of the function `/// @dev Asset` does not need to be active since all of the `gatewayDebt` might need to be withdrawn for the asset, once it is inactive. Hence, we recommend removing this comment, as it is incorrect.
2. `Regarding the comment:`
   `/// @param _amount The amount of the asset to be withdrawn`

   The full amount cannot always be withdrawn, so the comment should be updated to:

   `/// @param _amount The maximum amount of the asset to be withdrawn`

Item 10

packages/contracts/contracts/yieldFarm/SilentYieldFarm.sol#L839

The following function could be renamed as follows to more clearly express the intent of the function:

`_reorganizeStrategyArray => _compactStrategyArray`

Item 11

packages/contracts/contracts/yieldFarm/SilentYieldFarm.sol#L35

```
uint256 public minReportInterval = 1 days;
```

could be declared as a constant as follows:

```
uint256 public constant MIN_REPORT_INTERVAL = 1 days;
```

Item 12

packages/contracts/contracts/sp/SVault.sol#L37

`lastDappId` should be renamed to `nextDappId`

Item 13

packages/contracts/contracts/sp/SVault.sol#L125

`uint256[] memory balance`

should be renamed to the plural:

`uint256[] memory balances`

Item 14

packages/contracts/contracts/sp/SMASP.sol

`EncryptedBalance[ANONPOOLSIZE] memory encryptedBalance;`

should be renamed to the plural:

`EncryptedBalance[ANONPOOLSIZE] memory encryptedBalances;`

Item 15

packages/contracts/contracts/sp/SMASP.sol#L209

The function `_checkProofTime` could be refactored to be simpler and more gas efficient, with early exit for valid proof:

```
function _checkProofTime(uint256 _proofBlockNumber) internal view {
  if (_proofBlockNumber > block.number) {
    return; // Proof is valid (from the future)
  }
  if (block.number - _proofBlockNumber >= subscriptionDelta) {
     // proof is too old
    revert ProofExpired({
      currentBlockNumber: block.number,
      userBlockNumber: _proofBlockNumber
    });
  }
}
```

Item 16

packages/contracts/contracts/sp/SMASP.sol#L186

packages/contracts/contracts/sp/Gateway.sol#L269

We recommend standardizing on either `_checkRange` or `_rangeCheck`.

We additionally recommend abstracting `2 ** 30` into a constant to save gas:

`uint256 private constant MAX_AMOUNT = 2 ** 30;`

Item 17

In multiple places throughout the codebase (e.g., packages/contracts/contracts/sp/SMASP.sol#L144), Phase and Stage are used interchangeably. We recommend standardizing to use only one of them:

```
/// @dev checks if the phase 1 is ended for withdraw type transactions

modifier stage1Ended() {

    require(stage1ended, "phase 1 not ended");

    _;

}
```

Item 18

packages/contracts/contracts/sp/SMASP.sol#L56

The following comment:

```
/// @dev stores if an asset is allowed to be deposited and withdrawn
```

should be updated to:

```
/// @dev stores whether an asset may be deposited and withdrawn, or only withdrawn.
```

Item 19

contracts/contracts/sp/SMASP.sol#L1210

Since `allowedAsset` dictates `deposit` access, not `withdraw` access, a more clear naming could be:

Changing `setIsAllowedAsset` to `setAssetDepositsAllowed`

Changing `isAllowedAsset` to `assetDepositsAllowed`

Item 20

contracts/contracts/sp/SilentSanctionsList.sol#L64

The function `sanctionsList` returns a Bool, not a list. We recommend renaming it to `isSanctioned`.

contracts/contracts/sp/SilentSanctionsList.sol#L73

The function `silentSanctionsList` returns a Bool, not a list. We recommend renaming it to `isSanctionedSilent`.

Item 21

contracts/contracts/sp/Gateway.sol#L554

```
/// @dev See {IGateway-fromEZEEID}
```

should be updated to:

```
/// @dev See {IGateway-claimAndRegister}
```

Item 22

packages/contracts/contracts/sp/Gateway.sol#L367

`handleBurnSubscription` could be renamed to `burnSubscriptionFee`.

Item 23

```
uint256 public minYFDeployTimestamp = 86400; // 1 day
```

We recommend declaring the above as constant to save gas:

```
uint256 PUBLIC constant MIN_YF_DEPLOY_TIMESTAMP = 86400; // 1 day
```

Item 24

packages/contracts/contracts/sp/Gateway.sol#L315

packages/contracts/contracts/sp/SMASP.sol#L201

We recommend renaming `_checkAsset` to `_checkDepositsAllowed`.

Item 25

packages/contracts/contracts/referral/ReferralRewardPool.sol#L111

```
* @dev Allows owner get funds back to his address.
```

The owner is referenced in the comment, but the `onlyController` modifier is applied. We recommend checking if the correct access role is implemented, and verifying whether the comment is correct.

Appendix B: Comprehensive List of Locations Where a Zero Address Check Should Be Added

The following is a comprehensive list of all the locations where a zero address check needs to be added:

- packages/contracts/contracts/referral/ReferralController.sol#L132
- packages/contracts/contracts/referral/ReferralController.sol#L205
- packages/contracts/contracts/referral/ReferralController.sol#L234
- packages/contracts/contracts/referral/ReferralController.sol#L381
- packages/contracts/contracts/referral/ReferralController.sol#L391
- packages/contracts/contracts/referral/ReferralRegistry.sol#L114
- packages/contracts/contracts/referral/ReferralRegistry.sol#L126
- packages/contracts/contracts/referral/ReferralRewardPool.sol#L39
- packages/contracts/contracts/referral/ReferralRewardPool.sol#L98
- packages/contracts/contracts/referral/ReferralRewardPool.sol#L115
- packages/contracts/contracts/sp/Gateway.sol#L342

- packages/contracts/contracts/sp/Gateway.sol#L375
- packages/contracts/contracts/sp/Gateway.sol#L384
- packages/contracts/contracts/sp/Gateway.sol#L628
- packages/contracts/contracts/sp/Gateway.sol#L633
- packages/contracts/contracts/sp/Gateway.sol#L643
- packages/contracts/contracts/sp/Gateway.sol#L696
- packages/contracts/contracts/sp/Gateway.sol#L703
- packages/contracts/contracts/sp/Points.sol#L89
- packages/contracts/contracts/sp/Points.sol#L302
- packages/contracts/contracts/sp/SilentSanctionsList.sol#L27
- packages/contracts/contracts/sp/SilentSanctionsList.sol#L41
- packages/contracts/contracts/sp/SilentSanctionsList.sol#L53
- packages/contracts/contracts/sp/SMASP.sol#L109
- packages/contracts/contracts/sp/SMASP.sol#L324
- packages/contracts/contracts/sp/SMASP.sol#L369
- packages/contracts/contracts/sp/SMASP.sol#L428
- packages/contracts/contracts/sp/SMASP.sol#L968
- packages/contracts/contracts/sp/SMASP.sol#L986
- packages/contracts/contracts/sp/SMASP.sol#L1007
- packages/contracts/contracts/sp/SMASP.sol#L1025
- packages/contracts/contracts/sp/SMASP.sol#L1046
- packages/contracts/contracts/sp/SMASP.sol#L1119
- packages/contracts/contracts/sp/SMASP.sol#L1167
- packages/contracts/contracts/sp/SMASP.sol#L1179
- packages/contracts/contracts/sp/SMASP.sol#L1205
- packages/contracts/contracts/sp/SMASP.sol#L1207
- contracts/contracts/sp/SMASPForwarder.sol#L29
- contracts/contracts/sp/SMASPForwarder.sol#L128
- contracts/contracts/sp/SMASPForwarder.sol#L156

- contracts/contracts/sp/SmaspGatewayForwarder.sol#L96
- contracts/contracts/sp/SVault.sol#L239
- contracts/contracts/yieldFarm/SilentYieldFarm.sol#L166
- contracts/contracts/yieldFarm/SilentYieldFarm.sol#L234
- contracts/contracts/yieldFarm/SilentYieldFarm.sol#L418
- contracts/contracts/yieldFarm/SilentYieldFarm.sol#L435
- contracts/contracts/yieldFarm/SilentYieldFarm.sol#L520
- contracts/contracts/yieldFarm/adapters/SilentAdapterRegistry.sol#L22
- yieldFarm/strategies/ethenaStrategy/EthenaStrategy.sol#L193

About Least Authority

We believe that people have a fundamental right to privacy and that the use of secure solutions enables people to more freely use the Internet and other connected technologies. We provide security consulting services to help others make their solutions more resistant to unauthorized access to data and unintended manipulation of the system. We support teams from the design phase through the production launch and after.

The Least Authority team has skills for reviewing code in multiple Languages, such as C, C++, Python, Haskell, Rust, Node.js, Solidity, Go, JavaScript, ZoKrates, and circom, for common security vulnerabilities and specific attack vectors. The team has reviewed implementations of cryptographic protocols and distributed system architecture in cryptocurrency, blockchains, payments, smart contracts, zero-knowledge protocols, and consensus protocols. Additionally, the team can utilize various tools to scan code and networks and build custom tools as necessary.

Least Authority was formed in 2011 to create and further empower freedom-compatible technologies. We moved the company to Berlin in 2016 and continue to expand our efforts. We are an international team that believes we can have a significant impact on the world by being transparent and open about the work we do.

For more information about our security consulting, please visit https://leastauthority.com/security-consulting/.

Our Methodology

We like to work with a transparent process and make our reviews a collaborative effort. The goals of our security audits are to improve the quality of systems we review and aim for sufficient remediation to help protect users. The following is the methodology we use in our security audit process.

Manual Code Review

In manually reviewing all of the code, we look for any potential issues with code logic, error handling, protocol and header parsing, cryptographic errors, and random number generators. We also watch for areas where more defensive programming could reduce the risk of future mistakes and speed up future audits. Although our primary focus is on the in-scope code, we examine dependency code and behavior when it is relevant to a particular line of investigation.

Vulnerability Analysis

Our audit techniques include manual code analysis, user interface interaction, and whitebox penetration testing. We look at the project's website to get a high level understanding of what functionality the software under review provides. We then meet with the developers to gain an appreciation of their vision of the software. We install and use the relevant software, exploring the user interactions and roles. As we do this, we brainstorm threat models and attack surfaces. We read design documentation, review other audit results, search for similar projects, examine source code dependencies, skim open issue tickets, and generally investigate details other than the implementation. We hypothesize what vulnerabilities may be present and possibly resulting in Issue entries, then for each, we follow the following Issue Investigation and Remediation process.

Documenting Results

We follow a conservative and transparent process for analyzing potential security vulnerabilities and seeing them through successful remediation. Whenever a potential issue is discovered, we immediately create an Issue entry for it in this document, even before having verified the feasibility and impact of the issue. This process is conservative because we document our suspicions early even if they are later shown to not represent exploitable vulnerabilities. We generally follow a process of first documenting the suspicion with unresolved questions, then confirming the issue through code analysis, live experimentation, or automated tests. Code analysis is the most tentative, and we strive to provide test code, log captures, or screenshots demonstrating our confirmation. After this, we analyze the feasibility of an attack in a live system.

Suggested Solutions

We search for immediate and comprehensive mitigations that live deployments can take, and finally, we suggest the requirements for remediation engineering for future releases. The mitigation and remediation recommendations should be scrutinized by the developers and deployment engineers, and successful mitigation and remediation is an ongoing collaborative process after we deliver our Initial Audit Report, and before we perform a verification review.

Before our report, including any details about our findings and the solutions are shared, we like to work with your team to find reasonable outcomes that can be addressed as soon as possible without an overly negative impact on pre-existing plans. Although the handling of issues must be done on a case-by-case basis, we always like to agree on a timeline for a resolution that balances the impact on the users and the needs of your project team.

Resolutions & Publishing

Once the findings are comprehensively addressed, we complete a verification review to assess that the issues and suggestions are sufficiently addressed. When this analysis is completed, we update the report and provide a Final Audit Report that can be published in whole. If there are critical unaddressed issues, we suggest the report not be published and the users and other stakeholders be alerted of the impact. We encourage that all findings be dealt with and the Final Audit Report be shared publicly for the transparency of efforts and the advancement of security learnings within the industry.