Sharpe Base v2

Sharpe Labs

Auditor:
Sudeep, Independent Review

November 22, 2023

1 Security Document: Sharpe Base V2

1.1 Base Account

Definition: A *Base Account* in Sharpe Base V2 is a smart account (smart contract) created by the user to manage positions on the DApp. The creation of a Base Account is a prerequisite for any earnings-related interactions with Sharpe Base. It facilitates the consolidation of multiple transactions into a single operation, thereby enabling the creation and management of leveraged positions effectively.

1.1.1 Usage

The user's address is the sole proprietor of the Base Account. In the future stages of the project, users will have the option to authorize certain automated contracts. These contracts will have the capability to manage the user's positions autonomously. This feature ensures that users retain complete control over their individual positions, offering an advantage over traditional vault-based systems.

Note: According to the design framework, each address (Externally Owned Account, or EOA) is permitted to manage one instance of an account for each ETH Correlated position. For instance, a user must create two distinct accounts for managing individual wstETH-ETH and rETH-ETH positions. This approach significantly enhances the management of individual leveraged positions and improves the overall usability of the application. Importantly, only the user will have access to manage the assets or positions within their account.

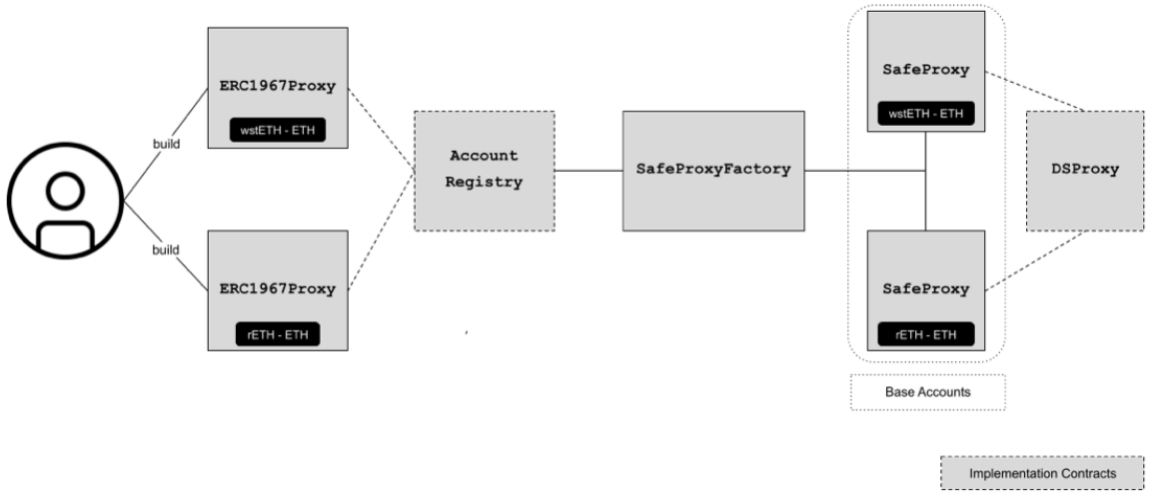

Figure 1: User build a base account instance for self

2 Smart Contract Descriptions

DSProxy.sol
It is a simple upgradeable contract that shall allow users to bundle tasks together like flashloan, swap, lend and borrow using Sharpe Base. This is the implementation of the smart account.

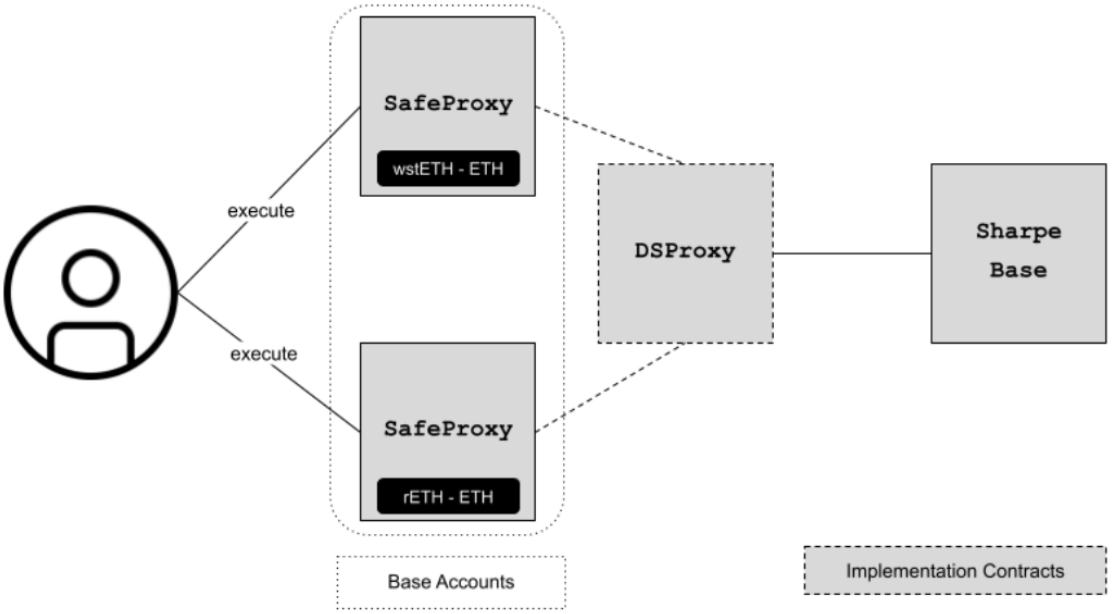

Figure 2: User sends bundled transaction data to Sharpe Base

SafeProxy.sol
It is a proxy to the DSProxy contract. This shall enable the owner of a smart account (user) to upgrade to a new implementation of Base Account without affecting their position. This is the contract where the leveraged position is owned.

SafeProxyFactory.sol
It is the contract responsible for creating an instance of SafeProxy.

AccountRegistry.sol
This contract uses SafeProxyFactory to create an account and stores the handle to account instance for each user only to retrieve it on demand. This is an upgradeable implementation.

ERC1967.sol
Proxy to the Account registry. This shall enable the Sharpe Base team to manage the smart accounts better in the future.

3 Sharpe Base

Sharpe Base is a DeFi Platform that allows users to create, customize, and manage yield positions.

Security By Architecture

There are a couple of aspects of how security is being addressed by architecture and design:

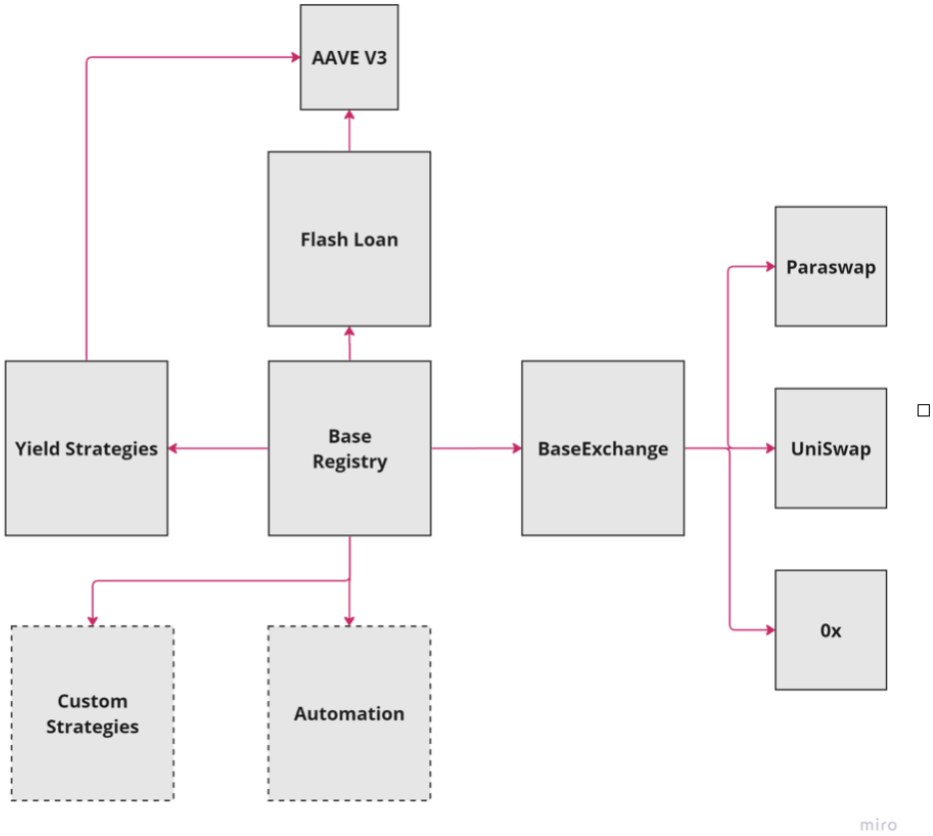

Figure 3: Figure 3: Base Registry and other components

Absence of an ERC4626 Vault

It is a common design in the Ethereum world to create an ERC4626 Vault to earn yields. However, while an ERC4626 is an optimized and standard way to earn yields, it is also a honey pot for attackers.

Sharpe Base's architecture allows users to enter leverage positions and earn yields on popular protocols like AAVE via individual positions, instead of a vault. The positions are held and managed by individual wallet addresses and their proxy accounts, i.e., smart contracts (Base Account: a proxy account that lets users program and automate or manually manage positions). Thus, an individual is able to create and manage leveraged wstETH - ETH positions without the need for an ERC4626 Vault.

Sole Ownership and Position Management

An individual can now create and manage leverage positions on protocols like AAVE without the dependency of a Vault. The Base Account is a proxy smart contract solely managed by the user. It is designed to bundle multiple transactions like taking a flashloan, swap, buy/sell assets, lend, borrow, withdraw into a single transaction, allowing even a novice DeFi user to earn higher yields via leverage.

The Base Account is further designed to allow users to integrate automation to manage one's yield positions. It can be programmed by an individual to boost current leverage, deleverage, or even close one's positions based on preconfigured triggers.

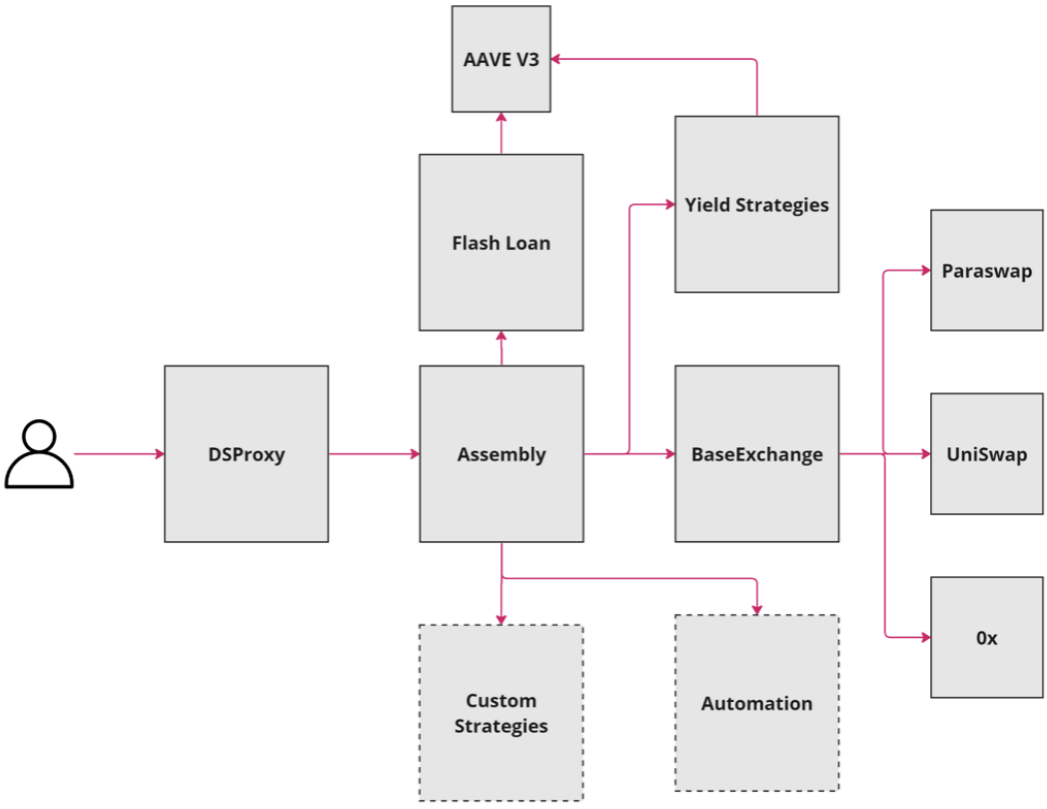

Figure 4: Figure 4: User interacts with DSProxy and Assembly to bundle and execute DeFi transactions

Using Registry Instead of Proxies

Proxy Contracts (not to be confused with the above-mentioned proxy) is a design principle where the contract implementation and the contract state are split. The implementation contracts hold the business logic, and the Proxy Contract owns the state. This allows the contract deployer to change the business logic while keeping the state intact. However, individual users should always direct the contracts via the Proxy Contract.

In contrast, a Registry is a design principle where the contract maintains the current implementation logic and keeps track of it via a Registry contract. The registry contract does not handle the state, unlike the Proxy. We opted for the Registry design since most of our contracts do not maintain any state as a design choice and only facilitate users' state management via the Base Account.

Security By Best Practices: Static Analysis with Slither

Slither is a static analysis framework specifically designed for analyzing Solidity smart contracts on the Ethereum blockchain. It is useful for developers, auditors, and security researchers to identify potential security vulnerabilities, design flaws, and other issues in Solidity code before deployment.

Benefits of Slither:

- *Comprehensive Analysis:* Slither performs a wide range of checks on Solidity contracts, detecting vulnerabilities like reentrancy bugs, integer overflow/underflow, and more.
- *Reporting and Visualization:* The tool provides detailed reports and visualizations to help users understand the issues found in their contracts.
- *Up-to-date:* Slither is actively maintained, keeping up with the latest security best practices and vulnerabilities in the Solidity ecosystem.

Using Slither for analysis is beneficial for identifying potential security flaws in Solidity smart contracts, a crucial step given their immutability post-deployment.

Links to Reports

Initial Report: `https://github.com/Sharpelabs/base-wallet/blob/forReview/slither_report_v2.json`

Final Report: `https://github.com/Sharpelabs/base-wallet/blob/forReview/slither_report_v2_resolved.json`

Code Coverage - Solidity Coverage

Code coverage tools, such as solidity-coverage, play a crucial role in ensuring the quality and reliability of smart contracts in Solidity projects. They are instrumental in measuring the effectiveness of test suites and identifying inadequately tested areas of the code.

Test Suite Effectiveness: These tools measure how much of the codebase is exercised during testing, helping developers improve the test suite by covering missed areas.

Bug Detection: Higher code coverage means more parts of the code have been tested, reducing the likelihood of unnoticed bugs or vulnerabilities and enhancing the contract's robustness.

Security Assurance: Since Solidity code often deals with financial transactions and assets, high code coverage helps ensure that the contract behaves as intended, minimizing security risks.

Documentation and Code Understanding: Coverage reports can also serve as documentation, aiding in understanding how different parts of the contract interact and highlighting potential areas of complexity or risk.

Compliance: In some cases, regulatory requirements or security standards may mandate a certain level of code coverage, and these tools help demonstrate compliance.

Solidity-coverage instruments the contract's bytecode to determine which parts of the code are executed during test runs, generating detailed reports on coverage percentage and highlighting the executed and non-executed lines. This tool is essential for improving the quality, security, and reliability of Solidity smart contracts.

Base Account

```
  67 passing (34s)

-----------------------------|----------|----------|----------|----------|----------------|
File                         |  % Stmts | % Branch |  % Funcs |  % Lines |Uncovered Lines |
-----------------------------|----------|----------|----------|----------|----------------|
 contracts/                  |      100 |    80.77 |      100 |      100 |                |
  Initializable.sol          |      100 |      100 |      100 |      100 |                |
  Proxy.sol                  |      100 |    68.75 |      100 |      100 |                |
  Variable.sol               |      100 |      100 |      100 |      100 |                |
 contracts/ds-auth/          |      100 |    93.75 |      100 |      100 |                |
  auth.sol                   |      100 |    93.75 |      100 |      100 |                |
 contracts/ds-note/          |      100 |      100 |      100 |      100 |                |
  note.sol                   |      100 |      100 |      100 |      100 |                |
 contracts/factory/          |       70 |       50 |       70 |    75.86 |                |
  IProxyCreationCallback.sol |      100 |      100 |      100 |      100 |                |
  SafeProxy.sol              |      100 |       50 |      100 |      100 |                |
  SafeProxyFactory.sol       |    68.42 |       50 |     62.5 |    73.08 |... 102,103,104 |
 contracts/registry/         |      100 |     87.5 |      100 |    94.12 |                |
  AccountRegistry.sol        |      100 |     87.5 |      100 |    94.12 |             79 |
  ProxyToAccountRegistry.sol |      100 |      100 |      100 |      100 |                |
-----------------------------|----------|----------|----------|----------|----------------|
All files                    |    90.48 |    80.88 |    90.63 |    91.75 |                |
-----------------------------|----------|----------|----------|----------|----------------|
```

Figure 5: Solidity Coverage report for Base Account

Peer Code Reviews

Peer code review is a critical software development practice where developers review each other's code to find defects, improve quality, and ensure adherence to standards and best practices.

Purpose: The main aim is to ensure high-quality code, identifying and fixing defects, and ensuring alignment with the project's architecture and design.

Process: Involves submitting code changes for review by other developers, who then provide feedback, leading to necessary revisions by the author.

Code Review Tool: Tools like GitHub pull requests are used to facilitate code reviews.

Benefits:

- *Bug Detection:* Identifies defects or logic errors.
- *Knowledge Sharing:* Offers learning opportunities and a deeper understanding of the codebase.
- *Code Consistency:* Ensures adherence to coding standards.
- *Continuous Improvement:* Promotes skill development and a culture of improvement.

Upcoming Enhancements with Respect to Security

- Invariant and Fuzzing Tests.
- Improved Code Coverage.
- CI/CD integration for tasks like static analysis and code coverage.
- Third Party Audits.
- Enhancements in Core Contract and functionality.

Conclusion

Sharpe Base v2 represents a significant advancement in decentralized finance (DeFi). It allows users to create, customize, and manage yield positions with enhanced security and usability. The platform's architecture offers individual user control and smart contract efficiency. Tools like Slither and solidity-coverage underline the platform's commitment to security. The reduction of issues in the Base Account and the strategic use of a Registry design over Proxy Contracts showcase a dedication to continuous improvement. Peer code reviews, future enhancements, and third-party audits ensure Sharpe Base v2 remains a leading solution in DeFi.

References

1. "Sharpe Base v2: Revolutionizing DeFi," Sharpe Labs Official Website, [Accessed Nov 22, 2023].
2. "Understanding Sharpe Base v2: A Non-Custodial DeFi Management Terminal," Sharpe Labs Medium Blog, [Accessed Nov 22, 2023].
3. "Sharpe Base v2 Initial Report," GitHub Repository, Sharpe Labs, [Accessed Nov 22, 2023], available at `https://github.com/Sharpelabs/base-wallet/blob/forReview/slither_report_v2.json`.
4. "Sharpe Base v2 Final Report," GitHub Repository, Sharpe Labs, [Accessed Nov 22, 2023], available at `https://github.com/Sharpelabs/base-wallet/blob/forReview/slither_report_v2_resolved.json`.
5. "Solidity Coverage Tool - Solidity-Coverage," GitHub Repository, [Accessed Nov 22, 2023].