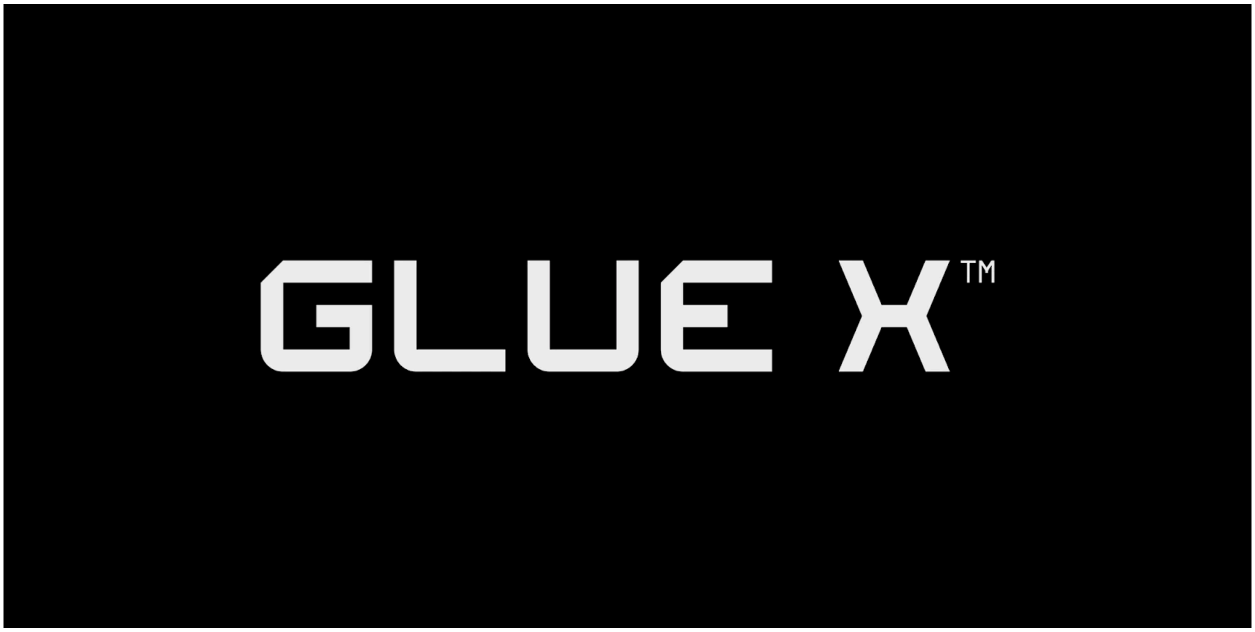
Symbiotic

Assignment for DeFi SE Internship

Arnav Suman

GITHUB LINK -

https://github.com/arnavsuman/GlueXProtocolAssignment



Introduction

Symbiotic is a shared security protocol designed to create a marketplace for economic security. It enables networks that need security to access it from those who have assets to stake, creating an efficient ecosystem where stake can be shared and utilized acr multiple networks. Through its flexible architecture, stake providers can maximize the returns while networks can obtain the security guarantees they need

Collateral

Collateral is a concept introduced by Symbiotic that brings capital efficiency and scale enabling assets used to secure Symbiotic networks to be held outside of the Symbiotic protocol - e.g. in DeFi positions on networks other than Ethereum.

Symbiotic achieves this by separating the ability to slash assets from the underlying asset itself, similar to how liquid staking tokens create tokenized representations of underlying staked positions. Technically, collateral positions in Symbiotic are ERC-20 tokens with extended functionality to handle slashing incidents if applicable. In other words, if the collateral token aims to support slashing, it should be possible to create a Burner responsible for properly burning the asset.

Deposit Function

The deposit function allows users to deposit their tokens into the contract, effective transferring ownership of those tokens to the contract while updating the user's balance within the contract. Once the vault is deployed, it becomes possible to deposit funds. In general, vaults can be configured to accept any ERC-20 compatible token, includi currently deployed DefaultCollateral tokens. However, we expect that vaults will be set up to accept the tokens directly without wrapping through DefaultCollateral. The DefaultCollateral will continue to be used as “pre-staking” collateral with caps.

Deposit to the vault.

```
python3 symb.py deposit VAULT_ADDRESS AMOUNT ON_BEHALF_OF
```

VAULT_ADDRESS - an address of the vault to deposit to

AMOUNT - an amount of tokens to deposit (in the token value, e.g., 1000 for 1000 ETH)

ON_BEHALF_OF - an address to make a deposit on behalf of (default: address of the signer)

Input Token Required for Deposit:

The input token required for the deposit is typically an ERC-20 token. The specific tok depends on the implementation of the DefaultCollateral contract, but it generally adheres to the ERC-20 standard.

Output Token Received After a Successful Deposit:

After a successful deposit, the user receives a representation of the deposited tokens within the DefaultCollateral contract. This representation is often in the form of a balance update in the contract, reflecting the amount of tokens deposited. The user does n receive a new token but rather an updated balance that signifies their deposited asse within the contract.

Key Logic

The **deposit function** in the DefaultCollateral contract involves key backend logic to handle balance calculations, deductions, and state updates securely and efficiently. Here's a detailed explanation:

Approval Check

Before a deposit can occur, the contract verifies that the user has approved the DefaultCollateral contract to spend their ERC-20 tokens. This is done through the approve function of the ERC-20 token contract. Once approved, the transferFrom function deducts the specified amount from the user's wallet. If the approval is insufficient or missing, the transaction fails, ensuring no tokens are transferred unintentionally. This approval mechanism serves as a security layer, allowing users to control how their tokens are accessed.

Balance Update

After successfully transferring the ERC-20 tokens to the contract’s balance, the DefaultCollateral contract mints an equivalent amount of its own tokens and updates the user’s balance. This is done using internal mappings that track each user’s holdings of DefaultCollateral tokens. Simultaneously, the contract's balance of the ERC-20 tokens is reflected in the token’s own balanceOf function, ensuring that every minted token is backed by an equal amount of the deposited asset.

Event Logging

For transparency, the contract emits a Deposit event after every successful deposit. This event records critical information, such as the depositor's address and the amount deposited. It allows external tools, such as blockchain explorers or analytics platforms, to monitor activity within the protocol. Event logging ensures that all operations are traceable, enhancing user trust and accountability within the system.

WITHDRAW FUNCTION

The withdraw function in the DefaultCollateral contract allows users to redeem their DefaultCollateral tokens in exchange for the underlying ERC-20 tokens they previously deposited. When a user initiates a withdrawal, the contract first verifies that the user has a sufficient balance of DefaultCollateral tokens to cover the withdrawal amount. It then burns the specified amount of DefaultCollateral tokens, reducing the total supply and updating the user’s token balance in the internal ledger. Following this, the contract transfers an equivalent amount of the underlying ERC-20 tokens from its balance to the user's wallet using the transfer function. If any condition fails, such as an insufficient balance of tokens or gas, the transaction reverts, ensuring atomicity. Additionally, the contract emits a Withdraw event, logging details of the withdrawal, such as the user’s address and the amount withdrawn, ensuring transparency and auditability within the protocol.

Withdraw from the vault

python3 symb.py withdraw VAULT_ADDRESS AMOUNT CLAIMER

VAULT_ADDRESS - an address of the vault to withdraw from

AMOUNT - an amount of tokens to withdraw (in the token value, e.g., 1000 for 1000 ETH)

CLAIMER - an address that needs to claim the withdrawal (default: address of the signer)

Input Token Needed for Withdrawal

The input token required for the withdrawal process is the DefaultCollateral token, which the user must hold in their wallet. These tokens represent the user's claim to an equivalent amount of the underlying ERC-20 tokens stored in the contract. The user specifies the amount of DefaultCollateral tokens they wish to redeem, which the contract then burns as part of the withdrawal process.

Output Token Received After Withdrawing

The output token received after a successful withdrawal is the underlying ERC-20 token originally deposited into the DefaultCollateral contract. The amount of ERC-20 tokens transferred to the user is exactly equal to the amount of DefaultCollateral tokens burned, ensuring a 1:1 exchange rate. These ERC-20 tokens are sent directly to the user's wallet from the contract’s balance, restoring the user’s ownership of the original asset.

Restrictions or Conditions Affecting the Withdrawal Process

1. **Sufficient DefaultCollateral Balance:**
    - The user must hold enough DefaultCollateral tokens to cover the requested withdrawal amount. If the balance is insufficient, the withdrawal process fails.
2. **Contract's ERC-20 Token Balance:**
    - The contract must have an adequate balance of the underlying ERC-20 tokens to fulfill the withdrawal request. If the contract’s balance is insufficient

(e.g., due to other withdrawals or operational issues), the transaction cannot proceed.

3. **Burning of Tokens:**
   - The DefaultCollateral tokens being redeemed are burned during the withdrawal process. This means the tokens are permanently removed from circulation, reducing the user's balance and the total token supply.
4. **Gas Requirements:**
   - The user must provide sufficient gas to cover the computational costs of the withdrawal process. If gas is insufficient, the transaction fails and reverts.
5. **Transaction Atomicity:**
   - The withdrawal operation is atomic. Any failure in checks (e.g., balance verification or transfer failure) will cause the entire transaction to revert, ensuring no partial state updates occur.
6. **Compliance with Contract Rules:**
   - The withdrawal must comply with any additional rules or limitations coded into the contract, such as time locks, withdrawal fees, or specific withdrawal caps, if applicable.

For Part 2 and Part 3 refer to google colab notebook

https://colab.research.google.com/drive/1AyL4IfZzg7NFVmfcXD1-TKybZKzQsRZx?usp=sharing