Two-Factor Authentication Smart Contract

Index

Smart contract Link :
https://sepolia.etherscan.io/address/0x81c2e414c54eef4240f2e079e8ba4b91450a962c#code

1. Introduction

This report provides a detailed analysis of the Solidity smart contract `TwoFactorAuth`, developed for implementing a two-factor authentication (2FA) system on the Ethereum blockchain. The contract is designed to enhance security by using a time-based one-time password (OTP) system.

2. Contract Overview

The `TwoFactorAuth` contract is a decentralized 2FA system. It allows users to register with a username and a public key (Ethereum address), after which the user can generate OTPs based on a shared seed and the current time. These OTPs can then be used for authentication within a specific time window, ensuring that only authorized users are able to authenticate.

Key Features:

- **Time-based OTP (TOTP)**: OTP is generated based on the current timestamp and a shared secret.
- **Security**: Mitigates replay attacks by ensuring that OTPs are time-limited.
- **User Management**: Users can register with a username and public key.
- **Authentication Events**: Authentication attempts are logged with events.

3. Functionalities

3.1 Register User

The `registerUser` function allows a new user to be registered with a username, public key, and OTP seed. A user cannot be registered more than once.

- **Parameters**:
  - `_username`: The user's chosen name.
  - `_publicKey`: The user's public key (Ethereum address).
  - `_otpSeed`: The seed used for OTP generation.
- **Logic**:
  - Verifies that the user is not already registered.
  - Creates a new `User` struct and stores it in the `users` mapping.
  - Emits a `UserRegistered` event.

3.2 Generate OTP

The `generateOTP` function generates a one-time password based on the OTP seed and the current time window. The OTP is valid for 30 seconds, and only registered users can request OTPs.

- **Logic**:
  - Retrieves the user's OTP seed and combines it with the current time (divided into 30-second windows).
  - Hashes the seed and time factor using `keccak256`.
  - Converts the hash into a 6-digit OTP by restricting its range.

3.3 Authenticate User

The `authenticate` function checks if the provided OTP is valid for the user. It prevents replay attacks by ensuring that each OTP can only be used once within the 30-second window.

- **Parameters**:
  - `_user`: The user's public key (Ethereum address).
  - `_otp`: The OTP provided by the user.
- **Logic**:
  - Ensures that the OTP is not being replayed within the same time window.
  - Verifies the OTP by comparing it to the one generated by `generateOTP`.
  - Logs successful authentications using the `Authenticated` event.

4. Technical Development

4.1 Structs

- **User**:
  - Holds the user's username, public key, OTP seed, the last OTP they used, and their last login time.

4.2 Mappings

- **users**:
  - Maps each Ethereum address (public key) to a `User` struct, managing all registered users.

4.3 Events

- **UserRegistered**:
  - Emitted when a new user registers.
- **Authenticated**:
  - Emitted when a user successfully authenticates with a valid OTP.

4.4 Modifiers

- **onlyRegistered**:
  - Ensures that only registered users can perform certain actions (such as generating OTPs and authenticating).

4.5 Constants

- **TIME_WINDOW**:
  - The duration (30 seconds) for which an OTP is valid.
- **OTP_LENGTH**:
  - Defines the length of the OTP, which is set to 6 digits.

5. Security Considerations

Replay Attack Prevention

The `authenticate` function includes logic to prevent OTP replay attacks by checking if the time since the user's last login is greater than the OTP validity window (30 seconds). This ensures that OTPs cannot be reused within the same time window.

Time-Based OTP

The contract implements a TOTP mechanism, where OTPs are based on the current time divided into 30-second intervals. The use of `keccak256` to hash the OTP seed and time factor ensures that OTPs are cryptographically secure.

OTP Size

To minimize brute force attacks, the OTP is restricted to 6 digits, providing 1,000,000 possible combinations for each 30-second interval.

6. Test Scenarios

- **Register a New User**:
    - Call `registerUser` with a unique username, public key, and OTP seed.
    - Ensure that the user is stored in the `users` mapping and the `UserRegistered` event is emitted.
- **Generate OTP**:
    - Call `generateOTP` for a registered user.
    - Verify that the generated OTP is within the expected range (6 digits).
- **Authenticate User**:
    - Call `authenticate` with a valid OTP.
    - Ensure the function returns `true` for valid OTPs and emits the `Authenticated` event.
    - Test the prevention of OTP replay attacks by attempting to authenticate within the same time window.

7. Conclusion

The `TwoFactorAuth` contract effectively implements a decentralized 2FA system using TOTP. The contract provides secure OTP generation and authentication mechanisms, with proper safeguards against replay attacks. The use of Ethereum's blockchain infrastructure allows for secure and transparent management of user identities and authentication events.