Protocol Audit Report

Version 1.0

Satya

January 9, 2025

Protocol Audit Report

satyasai

jan 9, 2025

Prepared by: satyasai Lead Auditors:

- xxxxxxx

Table of Contents

Protocol Summary

Protocol does PasswordStore

Disclaimer

The YOUR_NAME_HERE team makes all effort to find as many vulnerabilities in the code in the given time period, but holds no responsibilities for the findings provided in this document. A security audit by the team is not an endorsement of the underlying business or product. The audit was time-boxed and the review of the code was solely on the security aspects of the Solidity implementation of the contracts.

Risk Classification

		Impact		
		High	Medium	Low
	High	H	H/M	M
Likelihood	Medium	H/M	M	M/L
	Low	M	M/L	L

We use the CodeHawks severity matrix to determine severity. See the documentation for more details.

Audit Details

Commit Hash

The finding described in this document correspond the following commit hash:

```
2e8f81e263b3a9d18fab4fb5c46805ffc10a9990
```

Scope

```
.src/
    passwordStore.sol
```

Roles

- Owner: The user who can set the password and read the password.
- OutSides: No one else should to able to set or read the password.

Executive Summary

We use foundry

Issues found

severtity	Number of issue found
High	2
Medium	
Low	
Info	1
Total	3

Findings

High

[H-1] Storing the password on-chain makes it visible to anyone and no longer `private`

Description:

All data stored on chain `is public` and visible to anyone. The `PasswordStore:s_password` variable is intended to be hidden and only accessible by the owner through the `PasswordStore:getPassword` function.

I show one such method of reading any data off chain below. **Impact:** Anyone is able to read the `Private` password , severly breaking the functionality of the protocol.

Proof of Concept: (Proof of Code)

The below test case shows how anyone can read the password directly from the blockchain.

1.create a locally running chain

```
make anvil
```

2.Deploy the contract to the chain

```
make deploy
```

3.Run the storage tool

We use 1 because that's the storage slot of `s_password` in the contract

```
   cast storage <CONTRACT_ADDRESS_HERE> 1 --rpc-url http
      ://127.0.0.1:8545
```

You'll get an output that look like this:

`0x6d7950617373776f726400000000000000000000000000000000000000000014`

You can then parse that hex to a string with:

`cast parse-bytes32-string 0x6d7950617373776f726400000000000000000000000000000000000000000`

And get an output of:

```
myPassword
```

Recommended Mitigation: Due to this, the overall architecture of the contract should be rethought. One could encrypt the password off-chain, and then store the encrypted password on-chain. This would require the user to remember another password off-chain to decrypt the stored password. However, you're also likely want to remove the view function as you wouldn't want the user to accidentally send a transaction with this decryption key.

[H-2] `PasswordStore:getPassword` is callable by anyone

Description: The `PasswordStore:getPassword` function is set to be an `external` function, however the natspec of the function and overall purpose of the smart contract is that `This function allows only the owner to set a new password.`

```
1       function setPassword(string memory newPassword) external {
2   @>      //@audit - There are no access controls here
3        s_password = newPassword;
4        emit SetNetPassword();
5    }
```

Impact: Anyone can set/change the password of the contract

Proof of Concept: Add the following the `PasswordStore.t.sol` test suite.

```
1   function test_anyone_can_set_password() public {
2           vm.startPrank(address(1));
3           string memory expectedPassword = "myNewPassword";
4           passwordStore.setPassword(expectedPassword);
5           vm.stopPrank();
6           vm.prank(owner);
7           string memory actualPassword = passwordStore.getPassword();
8           assertEq(actualPassword, expectedPassword);
9       }
```

Recommended Mitigation: Add an access control modifier to the `PasswordStore.sol: setPassword` function

```
1    if(msg.sender =! s_owner) {
2     revert PasswordStore__NotOwner();
3    }
```

Informational

[L-1] The `PasswordStore:getPassword` natspec indicates a parameter that doesn't exist, causing the natspec to be incorrect

Description:

```
1       /*
2        * @notice This allows only the owner to retrieve the password.
3    @>  * @param newPassword The new password to set.
4        */
5       function getPassword() external view returns (string memory) {
```

The natspec for the function `PasswordStore::getPassword` indicates it should have a parameter with the signature `getPassword(string)`. However, the actual function signature is `getPassword()`.

Impact: The natspec is incorrect.

Proof of Concept:

Recommended Mitigation:

```
-     * @param newPassword The new password to set.
```

Gas