Protocol Audit Report

Version 1.0

Cyfe45

November 28, 2024

Protocol Audit Report

Cyfe45

28 November, 2024

Prepared by: Cyfe45

Lead Security Researcher:

- Cyfe45

Table of Contents

Protocol Summary

PasswordStore is a protocol dedicated to storage and retreival of a user's passwords. The protocol is designed to be used by a single user, and is not designed to be used by multiple users. Only the owner should be able to set and access the password.

Disclaimer

Cyfe45 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
Likelihood	High	H	H/M	M
	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

The findings described in this document correspond to the following commit hash: Commit Hash:

```
2e8f81e263b3a9d18fab4fb5c46805ffc10a9990
```

Scope

```
./src/
#-- PasswordStore.sol
```

Roles

- Owner: The owner of the contract is the only address that can set and read the password.
- Outsiders: No one else should be able to set or read the password.

Executive Summary

This is a simple protocol that allows only the owner to set and read the password. The owner should be able to set the password, and anyone else should not be able to read the password. However, we were able to find a few issues with the protocol.

- Outsiders are able to read and set the password, which is a serious issue.
- Documentation of the code can be improved further.

Issues found

Severity	Number of Issues Found
High	2
Medium	0
Low	0
Info	1
Total	3

Findings

High

[H-1] Storing the password on-chain makes it visable to anyone, and longer private

Description: All data stored on-chain is visible to anyone, and can be read directly from the blockchain. The `PasswordSTore::s_password` variable is intended to be a private variable and only accessed through the `PasswordSTore::getPassword` function, which is intended to be only called by the owner of the contract.

We show one such method of reading any data off chain below.

Impact: Anyone can 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 using another terminal

```
make deploy
```

3. Run the storage tool

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

```
cast storage <ADDRESS_HERE> ` --rapc-url http://localhost:8545
```

You';; get an output that looks like this:

`0x6d7950617373776f726400000000000000000000000000000000000000000014`

You can then parse the hex to a string with:

```
cast parse-bytes32-string 0
    x6d7950617373776f726400000000000000000000000000000000000000000014
```

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 password. Hopwever, you's also likely want to rtemove the view function as you wouldn't want the user to accidentally send a transaction with the password that decrypts the password.

[H-2] `PasswordStore::setPassword` has no access control, meaning a non-owner could change the password

Description: The `PasswordStore::setPassword` function is set to be an `external` function, however, the natspec of the function and overall purpose of the smart contract is `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 in this function,
       anyone can call this function and set the password.
3          s_password = newPassword;
4          emit SetNetPassword();
5      }
```

Impact: Anyone can set/change the password of the contract, severly breaking the contract intended functionality.

Proof of Concept: Add the following to the `PasswortdStore.t.sol` test file:

Code

```
1      function test_anyone_can_set_password(address randomAddress) public
           {
2          vm.assume(randomAddress != owner);
3          vm.prank(randomAddress);
4          string memory expectedPassword = "myNewPassword";
5          passwordStore.setPassword(expectedPassword);
6
7          vm.prank(owner);
8          string memory actualPassword = passwordStore.getPassword();
9          assertEq(actualPassword, expectedPassword);
10     }
```

Recommended Mitigation: Add an access control conditional to the `setPassword` function.

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

Informational

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

Description:

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

The `PasswordStore::getPassword` function has an incorrect natspec comment. The function signature is `getPassword()`, but the natspec suggests it should be `getPassword(string)`.

Impact: The natspec is incorrect, which may lead to confusion for developers or users of the contract.

Recommended Mitigation: Remove the incorrect natspec line.

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