PasswordShare Protocol Audit Report

Version 1.0

FalseGenius

April 19, 2024

PasswordShare Protocol Audit Report

FalseGenius

April 19, 2024

Prepared by: FalseGenius Lead Auditors: - xxxxxxx

Table of Contents

Protocol Summary

PasswordStore is a protocol dedicated to storage and retrival of a user's passwords. The protocol is designed to be used by a single user. Only the owner should be able to set and access this password.

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:

```
2e8f81e263b3a9d18fab4fb5c46805ffc10a9990
```

Scope

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

Roles

- Owner: The user who can set the password and retrieve it.
- Outsides: No one else should be able to read/set passwords.

Executive Summary

Add Notes about how the audit went, types of things you found etc. We spent X hours with Z auditors using Y tools etc

Issues found

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

Findings

High

[H-01] Storing password on chain makes it visible to anyone, regardless of the variable visibility.

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

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

Impact: Attackers can steal the password by reading it directly from the blockchain, severely breaking functionality of the contract.

Proof of Concept:

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

1. Create a locally running chain

```
make anvil
```

2. Deploy the contract to the chain.

```
forge script script/DeployPasswordStore.s.sol:DeployPasswordStore --rpc
    -url http://127.0.0.1:8545 --private-key <ANVIL_PRIVATE_KEY> --
    broadcast -vvv
```

3. Run the storage tool We use 1 because that's the storage slot of `s_password`

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

You'll get an output like this, `0x6d7950617373776f726400000000000000000000000000000000000000000000`

4. Decode the output You can parse that hex to string using,

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

and get output of,

```
myPassword
```

Recommended Mitigation: Due to this, the overall architecture of the contract should be rethought. One could encrypt the password before storing it on the chain. This would require user to remember another password to decrypt the password. However, you'd also likely want to remove the view function as you wouldn't want user to accidently trigger a transaction with the password that decrypts it.

[H-02] `PasswordStore::setPassword()` has no access control. Non-owner can set the password.

Description: The function `PasswordStore::setPassword()` can be used by non-owner to set the password, breaking the functionality of the contract, as opposed to the natspec of the function that states, `This function allows only the owner to set a new password.`

```
    function setPassword(string memory newPassword) external {
@>      // @audit No access control checks here
        s_password = newPassword;
        emit SetNetPassword();
    }
```

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

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

Code

```
    function testNonOwnerCanChangePassword(address randomAddress)
        public {
        vm.startPrank(randomAddress);
        string memory newPassword = "password2";
        passwordStore.setPassword(newPassword);
        vm.stopPrank();

        vm.prank(owner);
        string memory actualPassword = passwordStore.getPassword();

        // @audit Assert passes
        assertEq(actualPassword, newPassword);
    }
```

Recommended Mitigation: Add Access control conditional to that `PasswordStore::setPassword()` function.

```
    if (msg.sender != s_owner) revert PasswordStore__NotOwner();
```

Informational

[I-01] `PasswordStore::getPassword()` natspec indicates a newPassword parameter, causing the natspec to be incorrect.

Description: The function `PasswordStore::getPassword()` signature is `getPassword()` while natspec states that it should be `getPassword(string)`.

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

Impact: The natspec is incorrect.

Recommended Mitigation: Remove the incorrect natspec line.

```
    /*
     * @notice This allows only the owner to retrieve the password.
-    * @param newPassword The new password to set.
     * @audit-low getPassword doesn't have a newPassword parameter
```

```
5       */
6      function getPassword() external view returns (string memory) {}
```