PasswordStore Protocol Audit Report

Prepared by: Dominick Luviano

. . .

Table of Contents

Protocol Summary

PasswordStore is a protocol dedicated to storage and retrieval of a users protocol. This protocol is intented to be used by a single user and no other users should be able to retrieve your own password.

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

		Impact		
	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

** The findings in this document correspond to the following git hash:**

```
2e8f81e263b3a9d18fab4fb5c46805ffc10a9990
```

Scope

```
./src/
└── PasswordStore.sol
```

Roles

- Owner: The user who can set the password and retrieve the password.
- Outsiders: No one else should set or retrieve the password.

Executive Summary

Issues found

Severity	Number of issues found
High	2
Medium	0
Low	0
Info	1
Total	3

Findings

High

[H-1] Storing password on-chain makes it visible to anyone, and no 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 intented to be a private veriable and accessed through the `PasswordStore::getapaswword` function, which is intended to be only called by the owner of the contract.

Impact: Anyone can read the private password, severely breaking the functionality of the protocol.

Proof of Concept: The below test proves that anyone can read 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 <ADDRESS_HERE> 1 --rpc-url http://127.0.0.1:8545
```

You'll get an output that looks like this:

`0x6d7950617373776f726400000000000000000000000000000000000000000014`

You can then parse that hex to a string with:

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

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. However, you'd also likely want to remove the view function as you wouldn't want the user to accidentally send a transaction with the password that decrypts your password.

[H-2] `PasswordStore::setPassword` has no access controls, meaning non-owner can set password.

Description: The `PasswordStore::setPassword` function is set to be an external function, however the natspec of the function is that `This allows only the owner to retrieve the password.`

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

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

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

▶

```
        vm.prank(owner);
        string memory actualPassword = passwordStore.getPassword();
        assertEq(actualPassword, expectedPassword);
    }
```

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

```
if(msg.sender != owner) {
    revert PasswordStore__NotOwner();
}
```

Informational

[I-1] The `PasswordStore::getPassword` natspec indicates a function that does not exists, 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) {S
```

The `PasswordStore::getPassword` function selector is `getPassword()`, while the natspec says it should be `getPassword(string)`.

Impact: The natspec is incorrect

Recommended Mitigation:

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