Protocol Audit Report

Version 1.0

Prepared by: yappy-yum

July 8, 2025

Contents

Protocol Summary

PasswordStore is a protocol dedicated to store and retrive user's password. 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 this password

Disclaimer

yappy-yum 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 yappy-yum 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 the following commit hash:

```
2e8f81e263b3a9d18fab4fb5c46805ffc10a9990
```

Scope

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

Roles

- Owner: Only the owner may set and retrieve their password
- Outsiders: No one else should be able to set or read the password

Executive Summary

During the security review, I've discovered multiple vulnerabilities. We identified 2 high-severity vulnerabilities and 1 informational vulnerabiltity. One of the high-severity vulnerabilities stems from a conditional access control issue, while the other relates to the storage variable `s_password`. Below, we have provided in-dept details of all the code reviews

2 hour 28 minutes is spent only with 1 auditor myself during this audit

Issues found

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

Findings

High

[H-1] password stored on-chain is visible to anyone

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::getPasword` function, which is instended 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:

The test case below 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

```
cast storage <contract-address> 1
```

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

You'll get an output that looks like this:

```
0x6d7950617373776f726400000000000000000000000000000000000000000014
```

You can then parse that 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 rethough. This 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 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] missing access control in `PasswordStore::setPassword`

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 that `This function allows only the owner to set a new password`.

```
    /*
     * @notice This function allows only the owner to set a new password.
     * @param newPassword The new password to set.
     */
    function setPassword(string memory newPassword) external {
@>      // @audit high - There's no access control
        s_password = newPassword;
        emit SetNetPassword();
    }
```

Impact:

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

Proof of Concept:

```
    function test_anyone_can_setPassword(
        address _user,
        string calldata _password
    ) public {
        vm.assume(_user != address(0));

        // anyone can set password
        vm.prank(_user);
        passwordStore.setPassword(_password);
        vm.stopPrank();

        // check that password is stored
        vm.prank(owner);
        assertEq(passwordStore.getPassword(), _password);
    }
```

Recommended Mitigation:

Add an access control to the `setPassword` function

```
    /*
     * @notice This function allows only the owner to set a new password.
     * @param newPassword The new password to set.
     */
    function setPassword(string memory newPassword) external {
+       if (msg.sender != s_owner) {
+           revert PasswordStore__NotOwner();
+       }
        s_password = newPassword;
        emit SetNetPassword();
    }
```

Informational

[I-1] Incorrect natspec in `PasswordStore::getPassword`

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 `PasswordStore::getPassword` function signature is `getPassword()`, while the natspec say it should be `getPassword(string)`

Impact:

The natspec is incorrect

Recommended Mitigtion:

Remove the incorrect natspec line

```
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) {
```