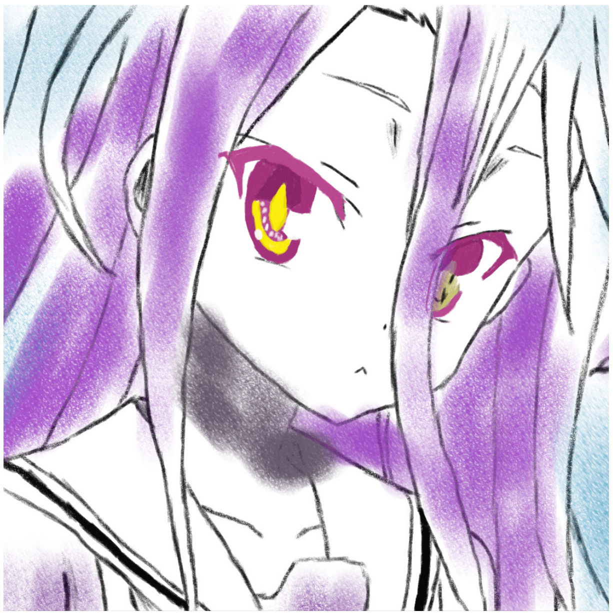

Protocol Audit Report

Version 1.0

Natalie Abigail

June 28, 2025

Protocol Audit Report

Natalie Abigail

June 28, 2025

Prepared by: Natalie “Nataly” Abigail Lead Auditors: - Nataly

Table of Contents

Protocol Summary

PasswordStore is a protocol dedicated to storage and retrieval 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 this password.

Disclaimer

I, Natalie "Nataly" Abigail, make 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 either an invidual or a 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

The findings in this document correspond to the following commit hash: `2e8f81e263b3a9d18fab4fb5c4680`

Scope

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

Roles

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

Executive Summary

This audit was done in roughly 1 hour by myself *(well, of course, with huge support from Patrick's course)*.

Issues found

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

Findings

Critical

[C-1] Storing the password on-chain makes it visible to everyone, thus not "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 purpose of the protocol.

Proof of Concept: 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 here since this is the storage slot of `s_password` in the deployed contract.

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

This will give you an output like this:

`0x6d7950617373776f726400000000000000000000000000000000000000000014`

that could be parsed to a string with this command:

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

and get an output of:

```
myPassword
```

Recommended Mitigation: Due to this vulnerability, the overall architecture of this contract is invalidated and should be rebuilt. One possible option is to encrypt the password off-chain, then store the encrypted value on-chain. However, should we carry on with this idea, the view function `getPassword()` might need to be removed, in order to not leak the passcode/key used for decrypting the on-chain password by mistake.

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

Description: The `PasswordStore::setPassword` is intended so that, as the documentation notated, `This function allows only the owner to set a new password`. However, the function is set to be `external`, and there is no check for owner's address in the code.

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

Impact: Anyone, regardless of owning the contract or not, could call the function, change the password, and sabotage the stored data of the contract owner. In other words, this breaks the contract's functionality.

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

Code

```
    function test_anyone_can_set_password(address randomAddress) public
        {
        vm.assume(randomAddress != owner);

        vm.prank(randomAddress);
        string memory expectedPassword = "myNewPassword";
        passwordStore.setPassword(expectedPassword);

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

Then run

```
forge test
```

We should expect this test to fail, confirming the vulnerability.

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

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

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.
     */
```

Impact: The natspec is incorrect.

Recommended Mitigation: Remove the incorrect natspec line.

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