Password Store Protocol Audit Report

Version 1.0

Mario Danish

May 27, 2025

Password Store Protocol Audit Report

Mario Danish

May 28, 2025

Prepared by: Mario Danish

Lead Auditors:

- Mario Danish

Table of Contents

- Table of Contents
- Protocol Summary
- Disclaimer
- Risk Classification
- Audit Details
  - The findings described in this document correspond the following commit hash:
  - Scope
  - Roles
- Executive Summary
  - Issues found
- Findings
  - High
    - [H-1] Storing the password on-chain makes it visible to anyone, and no longer private.
    - [H-2] `PasswordStore::setPassword` has no access control, meaning a non-owner could change the password
  - Informational
    - [I-1] The `PasswordStore::getPassword` natspec indicates a parameter that doesn't exist, causing the natspec to be incorrect.

Protocol Summary

PasswordStore is a protocol dedicated to store and retrieve user's password. Protocol only be access by the owner. Protocol is for single-user not for multiple-user.

Disclaimer

The Mario Danish 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
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: Is the only one who should be able to set and access the password.
- For this contract, only the owner should be able to interact with the contract. # Executive Summary

Issues found

Severity	Number of issues
HIGH	2
MEDIUM	0
LOW	0
INFORMATIONAL	1

Findings

High

[H-1] Storing the password on-chain makes it visible to anyone, and no longer private.

Description: All data stored on chain is public and visible to anyone. The `PasswordStore::s_password` variable is intended to be hidden and only accessible by the owner through the `PasswordStore::getPassword` function.

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

Impact: Anyone is able to read the private password, severely breaking the functionality of the protocol.

Proof of Concept: The below test case shows how anyone could read the password directly from the blockchain. We use foundry's cast tool to read directly from the storage of the contract, without being the owner.

1. Create a locally running chain

```
1    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 <PasswordStore_CONTRACT_ADDRESS_HERE> 1 --rpc-url http://
    127.0.0.1:8545
```

4. You'll get an output that looks like this:

```
0x6d7950617373776f726400000000000000000000000000000000000000000014
```

5. You can then parse that hex to a string with:

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

6. 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 stored password. However, you're also likely want to remove the view function as you wouldn't want the user to accidentally send a transaction with this decryption key.

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

Description: The `PasswordStore::setPassword` function is said to be `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.`

```
function setPassword(string memory newPassword) external {
@>        // @audit this function should be call by the owner but it
    has no access control modifier for the owner.
        s_password = newPassword;
        emit SetNetPassword();
    }
```

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

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

Code

```
1
2   function testAnyoneCanSetPassword(address randomAddress) public {
3           vm.assume(randomAddress != owner);
4           vm.prank(randomAddress);
5           string memory expectedPassword = "mynewPassword";
6           passwordStore.setPassword(expectedPassword);
7
8           vm.prank(owner);
9           string memory actualPassword = passwordStore.getPassword();
10          assertEq(actualPassword, expectedPassword);
11      }
```

Recommended Mitigation: Add the 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: The `PasswordStore::getPassword` function signature is `getPassword()` while the natspec says it should be `getPassword(string)`.

Impact: The natspec is incorrect

Recommended Mitigation: Remove the incorrect natspec line.

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