Protocol Audit Report

Version 1.0

Alqasem

August 14, 2025

Protocol Audit Report

Alqasem Hasan

August 14, 2025

Prepared by: Alqasem Lead Auditors: - Alqasem Hasan

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, Alqasem Hasan, make all the 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
	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 described in this document correspond the following commit hash:

```
2e8f81e263b3a9d18fab4fb5c46805ffc10a9990
```

Scope

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

Roles

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

Executive Summary

We spent 2 hours with 1 auditor, and we used Foundry Fuzz tests.

Issues found

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

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 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. Any password's stored on this contract are compromised.

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: (Proof of Code)

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

1. Create a locally running anvil 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 0
    x6d7950617373776f726400000000000000000000000000000000000000000014
```

And you will get an output of:

```
myPassword
```

Recommended Mitigation: Due to the fact that any password stored on this contract can be viewed by anyone, the entire architecture of the contract needs to be rethought. You could encrypt the password off-chain, and then store the encrypted password on-chain. However, this would require the user to remmember another passowrd 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 a non-owner could change the password

Description: The `PasswordStore::setPassword` is set to be an external function that should only the owner of the contract to call it according to the natspec. However, there is no check in the function to only allow the owner to call it, therefore anyone can call the function.

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

```
    }
```

Impact: Anyone is able to set the password of the contract, severly breaking the intended purpose of the contract.

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

Code

```
function test_anyone_can_set_password(address randomAddress) public {
        vm.assume(randomAddress != owner); // Ensure the random address
            is not the owner
        vm.startPrank(randomAddress);

        string memory expectedPassword = "myPassword";
        string memory newPassword = "newPassword";
        passwordStore.setPassword(newPassword);
        vm.stopPrank();
        vm.prank(owner);
        string memory actualPassword = passwordStore.getPassword();
        assertEq(actualPassword, newPassword);
    }
```

Recommended Mitigation: Add an access control conditional to the `PasswordStore::setPassword` function. It can look like this:

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

Informational

[I-1] The `PasswordStore::getPassword` natspec indicates that there should be parameter that does exist, causing the natspec to be wrong.

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) {
```

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.

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