Protocol Audit Report

Version 1.0

Cyfrin.io

July 10, 2025

Protocol Audit Report

Vlad SDME

July 10, 2025

Prepared by: Vlad_SDME Lead Security Researches: - Vlad_SDME

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

Vlad_SDME 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
	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 corresponded to the following commit hash:

```
7d55682ddc4301a7b13ae9413095feffd9924566
```

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

We spent 12 hours with 1 auditor(s) using at least 1 tool

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 everyone, and no longer private

Description: All data stored on-chain is public and visible to everyone 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: Lets create a local chain running. The `Foundry` framework has the tool named as `anvil`. In terminal, launch the `anvil` by the matching command: `anvil` Next we deploy our contract. In terminal we launch the deployment script by `make deploy` command The `Foundry` framework allows us to check the storage of a deployed contract with a very simple tool called `cast`. For this we'll need to recall to which storage slot the s_password variable is assigned - `Password::s_password` has slot 1 in the contract's storage. With this consideration we can run the command

```
1 `cast storage <address> <storageSlotNumber>`
```

Run the command in terminal:

```
1 `cast storage 0x5FbDB2315678afecb367f032d93F642f64180aa3 1`
```

We should receive an output similar to this:

```
1 `0x6d7950617373776f726400000000000000000000000000000000000000000014`
```

We earn the storage`s slot1` bytes form. By using another convinient Foundry command we can now decode this data:

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

As the result becomes:

```
1 `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 controls, meaning a non-owner could change the password

Description: The `PasswordStore::setPassword` fucntion is marked as `external` function, however it`s NatSpec indicate that`This function allows only the owner to set a new password.

```
1     function setPassword(string memory newPassword) external {
```

```
>@      // @Audit - There are no Access Controls.
        s_password = newPassword;
        emit SetNewPassword();
    }
```

Impact:: `PasswordStore::setPassword` function has no access control, so anyone who has access to this contract feel free to change the value of the `PasswordStore::s_password` variable. That severely breaking the contracts intended functionality.

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

```
function test_anyone_can_set_password(address randomAddress) public {
    vm.assume(randomAddress != owner);
    vm.startPrank(randomAddress);
    string memory expectedPassword = "myNewPassword";
    passwordStore.setPassword(expectedPassword);
    vm.startPrank(owner);
    string memory actualPassword = passwordStore.getPassword();
    assertEq(actualPassword, expectedPassword);
}
```

Recommended Mitigation: Add an access control conditional to `PasswordStore::setPassword`

```
    if(msg.sender != 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.
    */

   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
```
```

```
11
12 **Recommended Mitigation:**
13 Remove the incorrect NatSpec line:
14 ```diff
15     /*
16      * @notice This allows only the owner to retrieve the password.
17 -     * @param newPassword The new password to set.
18      */
```