Protocol Audit Report

Version 1.0

Josh Regnart

April 1, 2025

Protocol Audit Report

Josh Regnart

March 7, 2023

Prepared by: Josh Regnart

Lead Security Researcher: - Josh Regnart

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 the password.

Disclaimer

I make all effort to find as many vulnerabilities in the code in the given time period, but hold no responsibilities for the findings provided in this document. A security audit 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 corresspond to the following commit hash:

```
7d55682ddc4301a7b13ae9413095feffd9924566
```

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 set or read the password.

Executive Summary

The security audit of the PasswordStore protocol revealed critical vulnerabilities that fundamentally undermine its core functionality as a secure password storage solution. Given these findings, particularly the first high-severity issue, the protocol requires a complete architectural redesign to achieve its security objectives. A potential solution could involve implementing off-chain encryption before storing passwords, though this would introduce additional complexity and user requirements.

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 private and only accessed through the `PasswordStore::getPassword` function, which is intended to only be 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, severely breaking the functionality of the protocol.

Proof of Concept:

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

Start an anvil chain:

```
1 anvil
```

Deploy the contract:

```
1 forge script script/DeployPasswordStore.s.sol:DeployPasswordStore $(
    NETWORK_ARGS)
```

Read storage slot one (s_password) of the deployed contract: `bash cast storage 0 x5FbDB2315678afecb367f032d93F642f64180aa3 1 --rpc-url http://127.0.0.1:8545` Parse the bytes output to a string:

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

Returning the password:

```
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 password. However, you'd also likely want to remove the view function as you wouldn't want the user to accdently send a transaction with the password that decrypts your password. ### Likelihood & Impact: - Impact: HIGH - Likelihood: HIGH - Severity: HIGH

[H-2] `PasswordStore::setPassword` has no access controls, meaning anyone can set the password

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.`

```
1      function setPassword(string memory newPassword) external {
2  @>      // @audit - There are no access controls
3          s_password = newPassword;
4          emit SetNetPassword();
5      }
```

Impact:

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

Proof of Concept:

Add the following to the `PasswordStore.t.sol`:

Code

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

Recommended Mitigation:

Add an access control conditional to the `setPassword` function.

```
1  if(msg.sender != s_owner) {
2      revert PasswordStore__NotOwner();
3  }
```

Likelihood & Impact:

- Impact: HIGH
- Likelihood: HIGH
- Severity: HIGH

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

Recommended Mitigation: Remove the incorrect natspec line

```
    /*
     * @notice This allows only the owner to retrieve the password.
-    * @param newPassword The new password to set.
     */
```

Likelihood & Impact:

- Impact: NONE
- Likelihood: NONE
- Severity: Informational