Protocol Audit Report

Version 1.0

EricSullivan

July 14, 2025

Protocol Audit Report

Eric Sullivan and Patrick Collins

July 14, 2025

Prepared by: Eric Lead Auditors: - Eric Sullivan

Table of Contents

Protocol Summary

A smart contract application for storing a password. Users should be able to store a password and then retrieve it later. Others should not be able to access the password.

Disclaimer

The Eric Sullivan 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 Scope Details

- Commit Hash: 7d55682ddc4301a7b13ae9413095feffd9924566
- In 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 6 hours with Updraft Cyfrin team doing this audit. We found some interesting flaws on the protocol design.

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 (Root Cause + Impact)

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: Steps

1- Run an anvil Chain

2- Deploy our script with the command “make deploy”

3- cast storage contract number and storage slot where the password is store. Use the following command: cast storage 0x5FbDB2315678afecb367f032d93F642f64180aa3 1 –rpc-url http:// wahtever anvil chain config

4- Get the returned value from step 3 and “cast parse-bytes32-string 0x….” and you will get your password in string

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 function is set to be an external function, however the purpose of the smart contract and function’s natspec indicate that This function allows only the owner to set a new password. Therefore there should be a check in place for this.

```
1
2 function setPassword(string memory newPassword) external {
3     // @Audit - There are no Access Controls.
4     s_password = newPassword;
5     emit SetNewPassword();
6 }
```

Impact: Anyone can set/change the stored password, severely breaking the contract’s intended functionality

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

```
1
2 function test_anyone_can_set_password(address randomAddress) public {
3         vm.assume(randomAddress != owner);
4         vm.startPrank(randomAddress);
5         string memory expectedPassword = "myNewPassword";
6         passwordStore.setPassword(expectedPassword);
7
8         vm.startPrank(owner);
```

```
9           string memory actualPassword = passwordStore.getPassword();
10          assertEq(actualPassword, expectedPassword);
11      }
```

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

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

Medium

Low

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

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

Gas