



Protocol Audit Report

Version 1.0

Frank Shwan

July 31, 2025

Protocol Audit Report

Frank Shawn

July 31, 2025

Prepared by: yellowtree85 Lead Security Researcher: - Frank Shawn

Table of Contents

- Table of Contents
- Protocol Summary
- Disclaimer
- Risk Classification
- Audit Details
 - 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 controls, meaning a non-owner could change the password.
 - Medium
 - Low
 - Informational
 - * [I-1] `PasswordStore::getPassword` NatSpec references a nonexistent parameter
 - Gas

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

The YOUR_NAME_HERE 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:

```
1 2e8f81e263b3a9d18fab4fb5c46805ffc10a9990
```

Scope

```
1 src/  
2 --- 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

Add some notes about how the audit went, types of things you found, etc. We spent X hours with Z auditors using Y tools. etc.

Issues found

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

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 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.

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 how anyone can read the password directly form the blockchain.

- ### 1. Create a locally running chain

```
1 make anvil
```

- ## 2. Deploy the contract to the chain

```
1 make deploy
```

- ### 3. Run the storage tool

We use 1 because that's the storage slot of `s_password` in the contract.

```
1 cast storage <CONTRACT_ADDRESS> 1 --rpc-url http://127.0.0.1:8545
```

You'll get an output that looks like this:

[illegible]

You can then parse that hex to a string with:

[illegible]

And get an output of:

```
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 accidentally send a transaction with the password that decrypt your password.

[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 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` test file.

Code

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

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

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

Medium

Low

Informational

[I-1] PasswordStore::getPassword NatSpec references a nonexistent parameter

Description:

```
1      /*
2      * @notice This allows only the owner to retrieve the password.
3      * @param newPassword The new password to set.
4      */
5      function getPassword() external view returns (string memory) {
```

The `getPassword()` function does not accept any parameters, but the NatSpec comment incorrectly includes a `@param` tag for `newPassword`. This makes the NatSpec misleading and invalid.

Impact:

Incorrect NatSpec annotations can confuse developers and automated documentation tools, reducing code readability and maintainability.

Recommended Mitigation:

Remove the invalid `@param` line from the NatSpec comment to accurately reflect the function signature.

```
1      /*
2      * @notice This allows only the owner to retrieve the password.
3      -   * @param newPassword The new password to set.
4      */
5      function getPassword() external view returns (string memory) {
```

Gas