Protocol Audit Report

Version 1.0

Fawarano

September 12, 2025

Protocol Audit Report

fawarano

September 12, 2025

Prepared by: Fawarano Lead Auditors:

- Fawarano

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 pawword on-chain makes it visible to anyone, and no longer private
    * [H-2] `Passwordstore::setPassword` has no access control, meaning a non - owner could change the password
  - Informational
    * [I-1] Incorrect NatSpec for `PasswordStore::getPassword`

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 Fawarano 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 correspond the following commit hash:

```
2e8f81e263b3a9d18fab4fb5c46805ffc10a9990
```

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

Issues found

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

Findings

High

[H-1] Storing the pawword 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 inteded 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 bellow.

Impact: Anyone can read the private password, severly breaking the protocol functionality.

Proof of Concept: (or Proof of Code)

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

1. Create a locally running 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 get an output of:

```
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 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 to decrypts your password.

[H-2] `Passwordstore::setPassword` has no access control, meaning a non -owner could change the password

Description: The function `Passwordstore::setPassword` have `external` visibility and no access control whereas it's written on the natspec that `This function allows only the owner to set a new password`.

Impact: Anyone can set/change the password of the contract, severly breaking the protocol functionality.

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

Code

```
function test_anyone_can_set_password(address randomUser) public {
        vm.assume(randomUser != owner);
        string memory hackedPassword = "hackedPassword";
        vm.prank(randomUser);
        passwordStore.setPassword(hackedPassword);
        vm.prank(owner);
        string memory actualPassword = passwordStore.getPassword();
        assertEq(actualPassword, hackedPassword);
    }
```

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

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

Informational

[I-1] Incorrect NatSpec for `PasswordStore::getPassword`

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 NatSpec for `PasswordStore::getPassword` includes a parameter (`newPassword`) that does not exist in the function signature.
The actual function signature is `getPassword()` with no parameters.

Impact:
The NatSpec documentation is inaccurate and may cause confusion for developers and auditors.

Recommended Mitigation:
Remove the incorrect `@param` line.

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