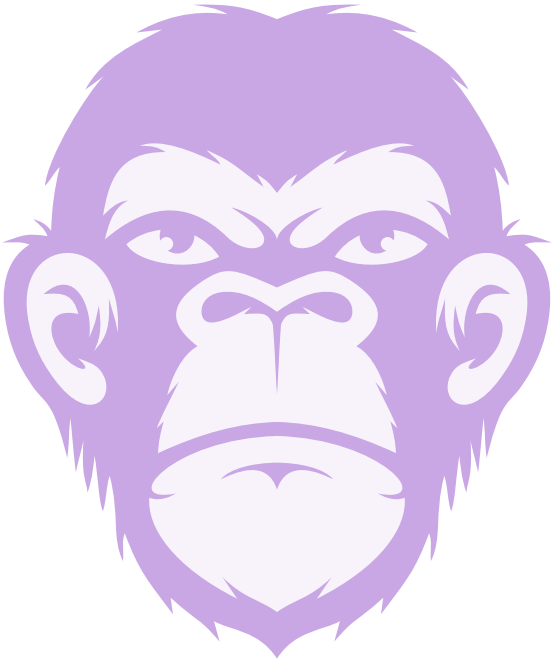

PasswordStore Audit Report

Version 1.0

Sebi de la Mata

February 18, 2026

PasswordStore Audit Report

Sebi de la Mata

February 17th, 2026

Prepared by: Sebi de la Mata Lead Auditors: - Sebi de la Mata

Table of Contents

Protocol Summary

PasswordStore protocol is dedicated to storage and retrieval of a user's password. The protocol is designed to only be used by a single user. Only the contract owner should be allowed to set and view this password.

Disclaimer

The Sebi de la Mata 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 in this document correspond with the following commit hash:##

```
7d55682ddc4301a7b13ae9413095feffd9924566
```

Scope

```
./src/
#-- PasswordStore.sol
```

Roles

```
- Owner: User that can set and view password.
- Outsiders: No one else should be able to view or set the password.
```

Executive Summary

We spent 2 hours with one auditor using Foundry to review the scoped code. Two high level findings and one informational finding were observed. One finding may require a rethink of the protocol design due to its impact.

Issues found

Severity	Number of issues found
High	2
Medium	0
Low	0
Informational	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 a private variable and only accessed

through the `PasswordStore::getPassword` function, which is only intended to be called by the owner of the contract.

We show one such method of reading any data offchain below:

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

Proof of Concept:

The below tests 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 is the storage slot for `PasswordStore::s_password`.

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

You'll get an output that looks like this: `0x6d7950617373776f726400000000000000000000000000000000`

You can then parse that hex to a string with:

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

And you get an output of:

```
myPassword
```

Recommended Mitigation: Due to this, the overall architecture of the project should be rethought. One could encrypt the password off-chain, then store the encrypted password on-chain. This would require the user to remember another password off-chain to decrypt the onchain stored password. However, you would also likely want to remove the view function as you wouldn't want the user to accidentally send a transaction with the off-chain password used to decrypt the onchain password.

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

Description: The `PasswordStore::setPassword` function is external, but only intended to allow the contract owner to call this function. The `PasswordStore::setPassword` function lacks an 'onlyOwner' function modifier that would be used to limit function calls based on the owner role.

```
1 /*
2 * @notice This function allows only the owner to set a new password.
3 * @param newPassword The new password to set.
4 */
5 // @audit any user can set a password
6 function setPassword(string memory newPassword) external {
7     s_password = newPassword;
8     emit SetNetPassword();
9 }
```

Impact: Anyone can set the password, severely breaking the functionality of the protocol.

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

Code

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

Recommended Mitigation: Add access control so that `PasswordStore::setPassword` can only be called by the owner:

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

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.
*/
// @audit no newPassword parameter
function getPassword() external view returns (string memory) {
    if (msg.sender != s_owner) {
        revert PasswordStore__NotOwner();
    }
    return s_password;
}
```

The `PasswordStore::getPassword` function signature is `getPassword()`, while the natspec indicates that it is `getPassword(string)`.

Impact: The natspec is incorrectly documenting the function.

Recommended Mitigation: Remove the incorrect natspec line.

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