



Protocol Audit Report

Version 1.0

Aditya Mohan

December 28, 2024

Protocol Audit Report

Aditya Mohan Jha

December 28, 2024

Prepared by: Aditya Lead Security Researcher: - Aditya

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 make visible to anyone.
 - * [H-2] `PasswordStore::setPassword` has no access controls,meaning a non-owner could change the password
 - Informational
 - * [I-1] The `PasswordStore::getPassword` natspec indicate a parameter that doesn't exist,causing the natspec to be incorrect.

Protocol Summary

PasswordStore is a protocol designed to store and retrieve password. This protocol is designed to be used by a single user.Only owner can set and access the 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

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 set and read the password.

Executive Summary

Issues found

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

Findings

High

[H-1] Storing the password on-chain make visible to anyone.

Description: All the 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.

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 from 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

```
1 forge inspect PasswordStore storage
```

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

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

You'll get the output which will look like this:

[illegible]

You can then parse that hex to string with:

```
1 cast parse-bytes32-string 0  
   x6d7950617373776f72644000000000000000000000000000000000000000000014
```

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 decrypts 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 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 randomUser) public {
2      vm.assume(randomUser != owner);
3      vm.prank(randomUser);
4      string memory expectedPassword = 'newPassword';
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.

Code

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

Informational

[I-1] The PasswordStore::getPassword natspec indicate a parameter that doesn't exist, causing the natspec to be incorrect.

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 `PasswordStore::getPassword` function signature is `getPassword()` which the natspec say it should be `getPassword(string)`

Impact: The natspec is incorrect

Recommended Mitigation: Remove the incorrect natspec line

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