



Protocol Audit Report

Version 1.0

Codepeaks

September 19, 2024

PasswordStore Protocol Audit Report

Codepeaks

September 19, 2024

Prepared by: Codepeaks Lead Security Researcher: - codepeaks

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 control, meaning non-owned could change the password.
 - Informational
 - * [I-1] The `PasswordStore::getPassword` natspec indicates a parameter that doesn't exist, causing the natspec to be incorrect.
 - 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 CODEPEAKS 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 to the following commit hash:

```
1 7d55682ddc4301a7b13ae9413095feffd9924566
```

Scope

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

Solc Version: 0.8.18 Chain(s) to deploy contract to: Ethereum ## Roles 1. Owner: The user who can set the password and read the password. 2. Outsiders: No one else should be able to set or read the password. # Executive Summary Spent 1 hour to audit this contract. found 2 high severity and 1 informational. ## 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.

Description: All data store 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 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 password stored on-chain, 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

```
1 make deploy
```

3. Run the storage tool

we use 1 because that is the storage slot of `s_password` in the contract.

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

You'll get the output of the storage tool:

[illegible]

you can parse the output hex to a string:

[illegible]

You will get the output:

```
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 control, meaning non-owned 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.

View test case

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

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

```
1 if(!msg.sender == 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:

```
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 says it should be `getPassword(string)`.

Impact: Natspec is incorrect.

Recommended Mitigation: Remove the incorrect natspec line.

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

Gas