Protocol Audit Report

Version 1.0

Bobby's Investigations

April 5, 2024

Protocol Audit Report

Bobby William Major

2 April, 2024

Prepared by: Bobby William Major Lead Secrurity Researcher:

- Bobby William Major

Table of Contents

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 Bobby's Investigations 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:

```
1  2e8f81e263b3a9d18fab4fb5c46805ffc10a9990
```

Scope

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

Roles

- Owner: Is the only one who should be able to set and access the password.

For this contract, only the owner should be able to interact with the contract.

Executive Summary

Issues found

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

Findings

High

[H-1] TITLE Storing the password on-chain makes it visible to anyone, and no longer private

Description: All data stored onchain 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 `Password::getPassword` function, which is intended to be only calledby the owner of the contract.

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

Proof of Concept: (Proof of Code)

The below test case shows how anyone can read the password directly from 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 <ADDRESS_HERE> 1 --rpc-url http://127.0.0.8545
```

You'll get an output that looks like this:

0x6d7950617373776f726400000000000000000000000000000000000000000014

You can then parse that hex to a string with:

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

And the output is:

```
1  myPassword
```

Recommended Mitigation: Due to this, the overrall architecture of the contract should be rethought. One could encrypt the paasword offchain, 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 a non-owner could

change password

Description: The `PasswordStore::setPassword` function is set to be an `external` function, however, the natspec of the function and overrall 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 any user can set the password
3          s_password = newPassword;
4          emit SetNetPassword();
5      }
```

Impact: Anyone can set/change the password of the contract, severly 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          vm.assume(randomAddress != owner);
3          vm.prank(randomAddress);
4          string memory expectedPassword = "myNewPassword";
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.

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

Informational

[I-1] The `PasswordStore::getPassword` natspec indicates a parameter that doesn't exit,

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.
```