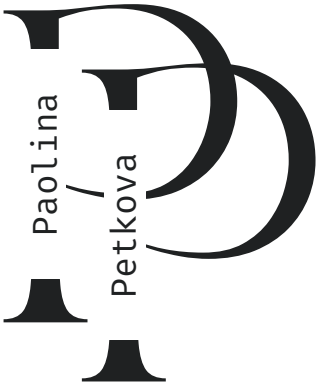

Protocol Audit Report

Version 1.0

Paolina Petkova

December 31, 2023

Protocol Audit Report

Paolina Petkova

Dec 31, 2023

Prepared by: [Paolina Petkova]

Lead Auditors:

- Paolina Petkova

Table of Contents

Protocol Summary

PasswordStore is a protocol dedicated to storage and retrival of user's passwords. The protocol is designed to be used by a single user, and is not designed to be used my multiple users. Only the owner should be able to set and access the password.

Disclaimer

Paolina 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 her 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 corespond with the following commit hash

```
7d55682ddc4301a7b13ae9413095feffd9924566
```

Scope

```
./src/
|__ PasswordStore.sol
```

Roles

- Owner: the user who can set the password and read the password
- Outsiders: no one else should be able to set or read the password

Executive Summary

The audit went great, I spend aroud 2 hours on reading, finding and documenting all the findings.

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.

root cause -> Storing the password on chain makes it visible to anyone impact -> no longer private

Description:

All data stored onchain is vidsible to anyone and can be read directly from the blockchain. The `PasswordStore::s_password` variable is intended to be a private and only accessed through

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

We show one such method of reading any data off chain below.

Impact:

Anyone can read the private password, severly breaking the functionality of the protocol.s

Proof of Concept: (Prove of Code)

The below test case shows how anyone can read teh 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 it is the storage slot of `s_password` in the contract.

   ```
   cast storage <ADDRES_HERE>_http://127.0.0.1:8545
   ```

   You will get an output that looks like this:

   ```
   0x6d7950617373776f726400000000000000000000000000000000000000000014
   ```

   Then you need to parse this bytes to string to see the human readable value:

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

   And you will get an output:

   ```
   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 stored password. However, you're also likely want to remove the view function as you wouldn't want the user to accidentally send a transaction with this decryption key.

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

Description:

`PasswordStore::setPassword` is an `external` function and could be accessed outside the contract, according to the documentation `This function allows only the owner to set a new password.` but nowhere is checked if the user is the owner.

```
function setPassword(string memory newPassword) external {
@>  // @audit - there is no access control
    s_password = newPassword;
    emit SetNetPassword();
}
```

Impact:

Anyone can set/change the password of the contact, severly breaking the contract intended functionality.

Proof of Concept:

Added the following test to `PasswordStore.t.sol` file which proves that any user can set the password.

Code

```
function test_anyone_can_set_the_password(address randomAddress) public {
    vm.assume(randomAddress != owner);
    vm.prank(randomAddress);
    string memory expectedPassword = "myNewPasswoword";
    passwordStore.setPassword(expectedPassword);

    vm.prank(owner);
    string memory actualPassword = passwordStore.getPassword();

    assertEq(actualPassword, expectedPassword);
}
```

Recommended Mitigation:

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

```
function setPassword(string memory newPassword) external {
    if (msg.sender != s_owner) {
        revert PasswordStore__NotOwner();
    }
    s_password = newPassword;
    emit SetNetPassword();
```

```
7 }
```

Informational

[I-1] `PasswordStore::getPassword` natspec indicates a parameter that does not exist, causing the natspec to be incorect

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) {
```

`PasswordStore::getPassword` function signiture is `getPassword()` while natspec says it should be `getPassword(string)`.

Impact:

The natspec is incorrect.

Proof of Concept:

None.

Recommended Mitigation:

Remove the incorrect natspec line.

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