Password Store Audit Report

Version 1.0

Aayush Gupta

December 23, 2023

Password Store Audit Report

Aayush Gupta

December 17, 2023

Prepared by: Aayush Gupta Lead Security Researcher: - Aayush Gupta

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

I (Aayush Gupta) make every effort to identify as many vulnerabilities in the code within the given time period but bear no responsibility for the findings presented in this document. A security audit by the team does not constitute an endorsement of the underlying business or product. The audit was time-boxed, and the code review focused 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:

```
2e8f81e263b3a9d18fab4fb5c46805ffc10a9990
```

Scope

```
./src/──
# PasswordStore.sol
```

Roles

- Owner: Is the only one who should be able to set and access the password.
- Outsiders: No one else should be able to set or read the password

Executive Summary

Add some notes about how the audit want, types of things you found, etc.

We spent x hours with z auditors using Y tools etc |Severtity|Number of issues found| |———|————————-| | High | 2 |
| Medium | 0 | | Low | 0 | | Info | 1 | | Total | 3 |

Issues found

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 `PaswordStore::s_password` variable is intended to be a private variable and only accessed through the `Password::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 private password, severly 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 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 beacuse that's the storage slot of `s_password` in the contract

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

You'll get something like this

`0x6d7950617373776f726400000000000000000000000000000000000000000014`

You can then parse that hex to a string with:

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

and get an output of:

```
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 an additional off-chain password for decryption. However, you'd also likely want to remove the view functions, 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 netspec of the function and overall purpose of the smart contract is that `This function allows only the owner to set a new password.`

```
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 contract, severly breaking the contract intended functionality.

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

Code

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

        vm.prank(owner);
        string memory originalPassword = passwordStore.getPassword();
        assertEq(expectedPassword, originalPassword);
    }
```

Recommended Mitigation: Add Access control like

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

Medium

Low

Informational

[I-1] Incorrect Parameter in `PasswordStore::getPassword` Netspec

Description:

```
    /*
     * @notice This function allows only the owner to retrieve the password.
     */
    function getPassword() external view returns (string memory) {}
```

The function signature for `PasswordStore::getPassword` is `getPassword()`, but the netspec incorrectly suggests it should be `getPassword(string)`.

Impact: This discrepancy in the netspec leads to confusion.

Proof Of Concept: None.

Recommended Mitigation: Correct the netspec by removing the inaccurate parameter description.

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

Gas