Protocol Audit Report

Zitife.O

April 9, 2025

{

Protocol Audit Report

Version 1.0

Zitife.O

April 10, 2025

}

Protocol Audit Report

Zitife.O

April 9, 2025

Prepared by: Zitife Security Researcher: - Zitife

Table of Contents

Protocol Summary

PasswordStore is a protocol dedicated to storing and retrieving passwords for users. It is designed for single users and not designed for multiple users. Only the owner can set and access passwords.

Disclaimer

The security researcher(zitife) 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

Impact

We use the CodeHawks severity matrix to determine severity. See the documentation for more details.

Audit Details

The Findings described in this document corresponds with the following commit hash:

```
2e8f81e263b3a9d18fab4fb5c46805ffc10a9990
```

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 and read the password

Executive Summary

I took my time to check the code extensively. I saw a couple of things that needed rectifying. Solidity metrics was used to check for the length of the code.

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 stored on-chain can be visible to anyone, and can be read directly from the blockchain. The `PasswordStore::s_password` 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, severely breaking the functionality of the protocol

Proof of Concept: (Proof code)

The below test can show that anyone can read the password from the blockchain 1. Create a locally running chain

```
make anvil
```

2. Deploy the contract on chain

```
make deploy
```

3. Run the storage tool We use `1` because 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 an output that looks like this:

```
0x6d7950617373776f726400000000000000000000000000000000000000000014
```

You can then parse that hex to a string with:

```
cast parse-bytes32-string
```

```
0x6d7950617373776f726400000000000000000000000000000000000000000014
```

And get an output of:

```
myPassword
```

Recommended Mitigation: Due to this, the contract's overall architecture should be reconsidered. A better approach would be to encrypt the password off-chain and store only the encrypted version on-chain. However, this would require users to remember an additional password for decryption. Additionally, it may be advisable to remove the view function to prevent users from unintentionally submitting a transaction that could expose their decrypted 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 that shows the overall purpose of the smart contract as well as the purpose of the function states that `This function allows only the owner to set a new password`

```
     function setPassword(string memory newPassword) external {
<@       // @audit there are no access controls
         s_password = newPassword;
         emit SetNetPassword();
    }
```

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

```
    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 actualPassword = passwordStore.getPassword();
            assertEq(actualPassword, expectedPassword);
        }
```

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

```
    if(msg.sender != s_owner) {
        revert PasswordStore_NotOwner();
    }
```

Informational

[I-1] The `PaswordStore::getPassword` natspec indicates a parameter that doesn't exist thus causing the natspec to be incorrect

Description:

```
     /*
     * @notice This allows only the owner to retrieve the password.
     * @audit: no newPassword parameter was set
  @>   * @param newPassword The new password to set.
     */
    function getPassword() external view returns (string memory)
```

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

Impact: The natspec is incorrect

Recommended Mitigation: Remove the incorrect natspec line

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