Protocol Audit Report

Version 1.0

0xwhisperingwoods

January 6, 2024

Protocol Audit Report

0xwhisperingwoods

January 6, 2024

Prepared by: Frankline Omondi Lead Security Researcher: - Frankline Omondi

Table of Contents

Protocol Summary

PasswordStore is a protocol dedicated to storage and retrieval of 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 0xwhisperingwoods 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 to 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 or read the password

Executive Summary

The audit went for ten hours. Using tools such as Fuzz testing, manual reviews, two high severity vulnerabilities and one informational was found. The audit proposes how the developers should go about solving the issues encoundered.

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 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 `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 private password, which severely breaks the functionality of the protocol.

Proof of Concept: (Proof of Code)

The test case below 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` because that would be the storage slot of `s_password` in the contract.

   ```
   cast storage <ADDRESS_HERE> 1 --rpc-url https://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 0
       x6d7950617373776f726400000000000000000000000000000000000000000000
   ```

   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 offchain, and then store the encrypted password on-chain. This would need 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 would not want the user to accidentally send a transaction with the password that decrypts your password.

Likelihood & Impact

-Impact: HIGH -Likelihood: HIGH -Severity: HIGH

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

Description: The `PasswordStore::setPassword` is set to be an external function, however, the natspec of the function and overall purpose of the Smart Contract is that `This function only allows 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, severly breaking the contract intended functionality.

Proof of Concept: (Proof of Code) 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 conditional to the `setPassword` function.

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

Likelihood & Impact

-Impact: HIGH -Likelihood: HIGH -Severity: HIGH

Informational

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

Description:

```
/*
     * @notice This allows only the owner to retrieve the password.
@>   * @param newPassword The new password to set.
     */
    function getPassword() external view returns (string memory)
```

The `PasswordStore::getPassword` function signature is `getPassword()` while 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.
```

Likelihood & Impact

-Impact: NONE -Likelihood: HIGH? -Severity: Informational/Gas/Non-crits