PasswordStore Audit Report

Prepared by: Abdul Azeez V

Table of Contents

Protocol Summary

PasswordStore is a protocol dedicated to storage and retrieval of passwords safely. It is designed for a single user. Only the owner should be allowded to set and access password.

Disclaimer

I 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

	Impact			
	Low	M	M/L	L

Audit Details

The findings described in this document correspond the following commit hash:

```
2e8f81e263b3a9d18fab4fb5c46805ffc10a9990
```

Scope

```
src/
--- PasswordStore.sol
```

Roles

- Owner: Is the only person to access and change paasword.

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] Passwords stored on-chain are visable to anyone, not private anymore

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 the

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

However, anyone can direclty read this using any number of off chain methodologies

Impact: The password is not private.

Proof of Concept: The below test case shows how anyone could read the password directly from the blockchain. We use foundry's cast tool to read directly from the storage of the contract, without being the owner.

1. Deploy the contract in localchain

```
make anvil    # creates localchain
make deploy   # deploys contract
```

2. 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 overall architecture of the contract should be rethought. One could encrypt the password off-chain, and then store the encrypted password on-chain.

[H-2] `PasswordStore::setPassword` is callable by anyone, So Anyone can change password

Description: The `PasswordStore::setPassword` function is set to be an `external` function, however the natspec 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 are no access controls here
        s_password = newPassword;
        emit SetNetPassword();
    }
```

Impact: Anyone can set/change the password of the contract.

Proof of Concept:

Add the following to the `PasswordStore.t.sol` test suite.

```
function test_anyone_can_set_password(address randomAddress) public {
    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();
}
```

Low

[L-1] Initialization Timeframe Vulnerability

Description: There is a period between contract deployment and the explicit call to setPassword during which the password remains in its default state. During this initialization timeframe, the contract's password is effectively empty and can be considered a security gap.

Impact: During the initialization timeframe, the contract's password is left empty, potentially exposing the contract to unauthorized access or unintended behavior.

Recommended Mitigation: Add a default password during deployment.

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 natspec for the function `PasswordStore::getPassword` indicates it should have a parameter with the signature `getPassword(string)`. However, the actual function signature is `getPassword()`.

Impact: The natspec is incorrect.

Recommended Mitigation: Remove the incorrect natspec line.

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