# Password Store Initial Audit Report


**m1nd0v3rfl0w.io**

Prepared by: m1nd0v3rfl0w
Lead Auditors:

- Abhinav Shukla

# Table of Contents


- Table of Contents
- Protocol Summary
- Disclaimer
- Risk Classification
- Audit Details
  - Scope
  - Roles
- Executive Summary
  - Issues found

- Findings
- High
- Medium
- Low
- Informational
- Gas


# Protocol Summary

Protocol allows the owner to set a password, and retrieve it. Other users should not be able to see the stored password.

# Disclaimer

m1nd0v3rfl0w has made 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 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.

## Issues found

| Severity | Number of issues found |
| --- | --- |
| High | 2 |
| Medium | 0 |
| Low | 0 |
| Info | 1 |
| Total | 3 |

# Findings

# High

### [H-1] Storing the passoword 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 the PasswordStore::getPassword function, which is intended to be only called by the owner of the contract.

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

**Impact:**
Anyone can read the private password, severly breaking the functionality of the protocol.

**Proof of Concept:**
The below test case shows how anyone can read the password directly from the blockchain.

1. Deploy the contract on the Anvil chain using the make deploy command. The password set using the default script is 'myPassword'.
2. Access the storage layout of the contract using the `forge inspect PasswordStore storage-layout --pretty` command. The password is stored in slot 1.
3. Access the storage slot 1 using `cast storage 0x5fbdb2315678afecb367f032d93f642f64180aa3 1`. The output obtained is `0x6d7950617373776f726400000000000000000000000000000000000000000014`.
4. Decode this using `cast to-ascii 0x6d7950617373776f726400000000000000000000000000000000000000000014`. The password 'myPassword' is clearly visible.

**Recommended Mitigation:**
Due to this, the overall architecture of the protocol needs to 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 password. However, you'd also likely want to remove the view function as you wouldn't want the user to accidently send a transaction with the password that decrypts the 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 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-issue - There are no access controls
        s_password = newPassword;
        emit SetNetPassword();
    }
```

**Impact:** Anyone can set/change the password of the contract, severly breaking the contract's intended functionality.

**Proof of Concept:**

▼ Add the following to the test suite:

```
function test_anyone_can_set_password_fuzz(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 Password_NotOwner();
}
```

# 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.
```