Protocol Audit Report

Version 1.0

sharthak18

November 8, 2025

Protocol Audit Report

sharthak18

November 7, 2025

Prepared by: sharthak18 Lead Security Researcher: - sharthak18

Table of Contents

Protocol Summary

PasswordStore is a simple smart contract that allows the owner to store and retrieve a password on the Ethereum blockchain. The contract uses OpenZeppelin's Ownable contract to manage ownership and restrict access to certain functions.

Disclaimer

The sharthak18 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 the following commit hash:

```
7d55682ddc4301a7b13ae9413095feffd9924566
```

Scope

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

Roles

- Owner: The user who can set the password and read the password.
- Outsides: No one else should be able to set or read the password. # Executive Summary
  - The audit identified several security issues in the PasswordStore contract.
  - The most critical issue is that the password is stored on-chain in plain text, making it accessible to anyone who can read the blockchain data. This undermines the purpose of the contract, which is to keep the password private.
  - Another significant issue is that the setPassword function lacks access control, allowing anyone to change the password. This could lead to unauthorized changes and compromise the integrity of the stored password.
  - Additionally, the getPassword function's natspec documentation is misleading, as it indicates a parameter that does not exist. This could confuse users and lead to incorrect usage of the function.
  - Overall, the audit highlights the need for improved security measures in the PasswordStore contract to ensure that passwords are stored and managed securely. ## Issues found

```
| Severity      | Number of Issues |
| ------------- | ---------------- |
| High          | 2                |
| Medium        | 0                |
| Low           | 0                |
| Informational | 1                |
| Gas           | 0                |
| Total         | 3                |
```

Findings

High

[H-1] Storing the password on-chain makes it visible to everyone, and no longer private.

Description: All data stored on-chain is visible to anyone, and can read directly from the blockchain. The `PasswordStore::s_password` variable is intended to be private variable, and only accessed throug 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 data off chain below.

Impact: Anyone can read the private password stored on-chain, severely breaking the functionality of the protocol.

Proof of Concept: (Proof of Code)

The below test case shows anyone can read the password directly from the blockchain, without needing to be the owner of the contract.

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 the storage slot of `s_password` is 1 (0 is for owner) `bash cast storage 0x5FbDB2315678afecb367f032d93F642f64180aa3 1 --rpc-url http://127.0.0.1:8545` you will get output like this : `0x6d7950617373776f72640000000000`

   you can parse that hex to string like this:

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

   which will give you the output: `myPassword`

Recommended Mitigation: Due to this, the overall architecture 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 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 actully send a transaction with the password that decrypts your password.

[H-2] `passwordStore::setPassword` has no access control, meaning a non-owner can 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 the password.`

```
function setPassword(string memory newPassword) external {
@>       // @audit - there is no access control here
        s_password = newPassword;
        emit SetNetPassword();
    }
```

Impact: Anyone can set/change the password of the contract, severely breaking the functionality of the contract.

Proof of Concept: Add the following test case to `test/PasswordStore.t.sol`

Code

```
    function test_anyone_can_set_password() 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 != owner) {
    revert PasswordStore__NotOwner();
}
```

Informational

[I-1] The `passwordStore::getPassword` natspec indicates a parameter that does not exist, Causing the natspec to be misleading.

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()` which the natspec say it should be `getPassword(string)`.

Impact: The natspec is incorrect.

Recommended Mitigation: Remove the incorrect natspec.

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