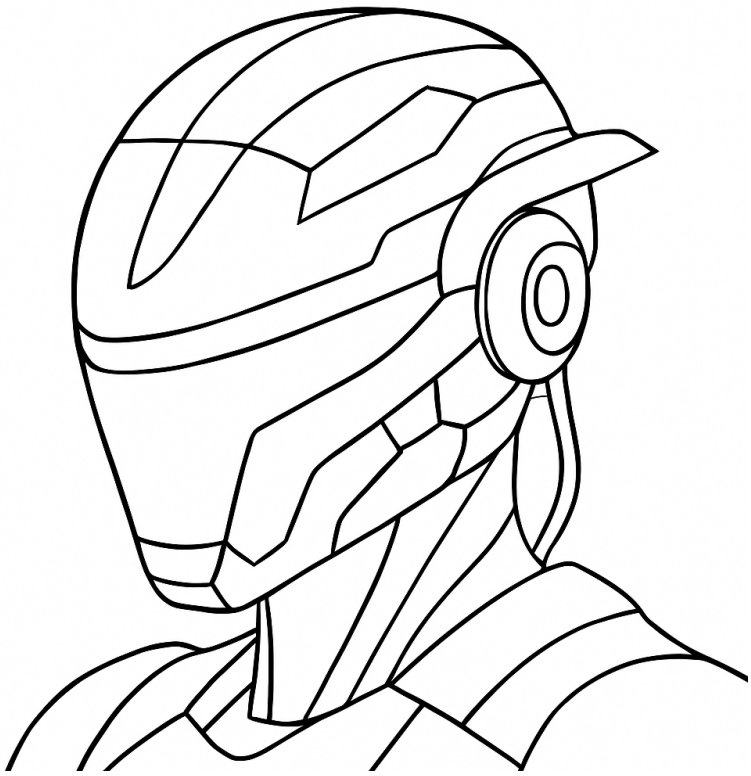

PasswordStore Audit Report

Prepared by: AlexScherbayuk

Table of Contents

Protocol Summary

PasswordStore is a protocol dedicated to storage and retrieval of a 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 the password.

Disclaimer

The AlexScherbatyuk 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:

```
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.

Executive Summary

We found several 2 high issues and 1 documentation issue We spent 3 days with 1 auditor using Aderyn, Slither, Foundry tools

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 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, severely breaking the functionality of the protocol. **Proof of Concept:** (Proof of Code) The below test case shows how anyone can read the password directly from the blockchain.

1. Start a local node

```
make anvil
```

2. Deploy This will default to your local node. You need to have it running in another terminal in order for it to deploy.

```
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 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
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. 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 accidentally send a transaction with the password that decrypts your 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 - 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 {
        // make sure randomAddress is not owner
        vm.assume(randomAddress != owner);
        // prank randomAddress
        vm.prank(randomAddress);
        // set password
        string memory expectedPassword = "myNewPassword";
        passwordStore.setPassword(expectedPassword);
        // prank owner
        vm.prank(owner);
        // retrieve stored password
        string memory actualPassword = passwordStore.getPassword();
        // assert actualPassword is equal to expectedPassword
        assertEq(actualPassword, expectedPassword);
    }
```

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

```
    if (msg.sender != s_owner) {
        revert PasswordStore__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 `PasswordStore::getPassword` function signature is `getPassword()` which the natSpec says it should be `getPassword(string)`. **Impact: Recommended Mitigation:** Remove the incorrect natSpec line.

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