Protocol Audit Report

Version 1.0

Cyfrin.io

August 12, 2025

Protocol Audit Report

Elena

August 12, 2025

Prepared by: Elena Lead security researcher: - xxxxxxx

Table of Contents

Protocol Summary

PasswordStore is protocol dedicated to storage and retrieval of a user's password. 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 YOUR_NAME_HERE 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 currespond the following commit hash:

```
e8f81e263b3a9d18fab4fb5c46805ffc10a9
```

Scope

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

Roles

- Owner: the user who can set the password and read the password.
- Outsider: no one else should be able to set or read the password. # Executive Summary ## Issues found # Findings ## High ### [H-1] TITLE Storing the password on-chain makes it visiable to anyone, and no longer private

Description: all data 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, severly 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. create a locally running chain `bash make anvil` 2. deploy the contract to the chain `make deploy` 3. get contract address `0x5FbDB2315678afecb367f032d93F642f64180aa3` 4. cast storage address 1(slot, which is s_password) –rpc-url http://localhost:8545 5. then get the output like `0x6d7950617373776f726400000000000000000000000000000000000000000014` 6. cast parse-bytes32-string output then get output `my password`

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 accidently send a transaction with the password that decrypts your password

likehood & impact

-impact :HIGHp -Likelihood: HIGH -Severity :HIGH ### [H-2] TITLE `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 control
        s_password = newPassword;
        emit SetNetPassword();
    }
```

Impact: anyone can set/change the password of the contract, severly 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{
        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 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] TITLE the `passwordStore::getPassword` natspec indicates a parameter that doesn't exist, causing the natspec to be incorrct

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 line.

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

Likelihood & impact :

-impact : HIGH -likelihood: NONE -severity: Informational /Gas /Non-crits

Informational: this is not a bug, but you should know.. ## Gas