Protocol Audit Report

Version 1.0

Daniel Stamler

January 26, 2024

Password Store Audit Report

Daniel Stamler

January 26, 2024

Prepared by: Daniel Stamler - Lead Security Researcher

Table of Contents

Protocol Summary

The “PasswordStore” protocol provides a secure way to store and update a private password, ensuring that only the owner can access and modify the password, thus maintaining its confidentiality.

Disclaimer

I made an effort to find as many vulnerabilities in the code in the given time period, but hold no responsibilities for the findings provided in this document. A security audit 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
Likelihood	High	H	H/M	M
	Medium	H/M	M	M/L
	Low	M	M/L	L

I 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 comit hash:

```
2e8f81e263b3a9d18fab4fb5c46805ffc10a9990
```

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 ## Issues found

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

Findings

High

[H-1] On-chain stored variables are visible to anyone, regardless of solidity visibility, making the password not 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 on 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.

We use foundry's cast tool to read directly from the storage of the contract, without being the owner.

1. Create a locally running chain

```
make anvil
```

2. Deploy the contract to the chain

```
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 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 0
    x6d7950617373776f726400000000000000000000000000000000000000000014
```

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. 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 control, meaning a non-owner can set the paswword

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.`

```
1      function setPassword(string memory newPassword) external {
2  @>      // @audit - There are no access controls
3          s_password = newPassword;
4          emit SetNetPassword();
5      }
```

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.

Test Code

```
    function test_non_owner_tries_to_set_password(address randomAddress
        ) public {
        vm.assume(randomAddress != owner);
        vm.startPrank(randomAddress);
        string memory blackHatPassword = "imABadActor";
        passwordStore.setPassword(blackHatPassword);
        vm.startPrank(address(owner));
        string memory currentPassword = passwordStore.getPassword();
        assertEq(blackHatPassword, currentPassword);
    }
```

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

Impact: The natspec is incorrect.

Recommended Mitigation: Remove the incorrect natspec line.

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