Protocol Audit Report

Version 1.0

Danail Vasilev

February 21, 2024

Protocol Audit Report

Danail Vasilev

February 21, 2024

Prepared by: Danail Vasilev Lead Security Researcher: - Danail Vasilev

Table of Contents

Protocol Summary

PasswordStore is a 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 access and modify the password.

Disclaimer

The Danail Vasilev 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

Commit Hash: **The findings described in this document correspond to the following commit hash:**

```
be77629eed6461ed483725fb73491373c57e06e6
```

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 spent X hours with Y auditors, using Z tools, etc.

Issues found

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

Findings

High

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

Description: All data stored on-chain is visible to anyone and can be read directly by the blockchain. The `PassowrdStore::s_password` variable is intended to be private variable and only accessed through the `PasswordStore::getPassword` function which is intended to be called only 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 shows how to directly read from the blockchain

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 0x5FbDB2315678afecb367f032d93F642f64180aa3 1 --rpc-url
    http://127.0.0.1:8545
```

You will get an output that looks like that:

`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 protocol should be rethought. One could encrypt the password off-chain and then store the encrypted password offchain. This would require the user to remember another password off-chain to decrypt the password. Howerver, you'd also want to remove the view function as you wouldn't want the user to send a transaction with the password that decrypts your password.

[H-2] `PasswordStore::setPassword` has no access control, meaning a non-woner can change the password

Description: The `PasswordStore::setPassword` function is set to be an `external` function, however, the natspec of the function and the 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 is no access control here
        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
        {
        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 `PasswordStore::setPassword` function.

```
if (msg.sender != owner) {
    revert PasswordStore__NotOwner();
}
```

Informational

[I-1] / [NC-1] The `PassswordStore::getPassword`function natspec indicates a parameter that doesn't exist, causing the natspec to be incorrect

Description:

```
    /*
     * @notice This function allows only the owner to set a new
        password.
@>   * @param newPassword The new password to set.
     */
    function setPassword(string memory newPassword) external {
        s_password = newPassword;
        emit SetNetPassword();
    }
```

The `PasswordStore:getPassword` function signature is `getPassword()` while 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.
```