Protocol Audit Report

Version 1.0

Squilliam

April 5, 2024

Protocol Audit Report

Squilliam

April 5, 2024

Prepared by: [Squilliam] Lead Auditors:

- Squilliam

Table of Contents

- Table of Contents
- Protocol Summary
- Disclaimer
- Risk Classification
- Audit Details
  - Scope
  - Roles
- Executive Summary
  - Issues found
- Findings
  - High
    - [H-1] Storing the password on-chain makes it viable to anyone and no longer private
    - [H-2] `PasswordStore::setPassword` has no access controls, meaning a non-owner could change the password
  - Informational
    - [I-1] `PasswordStore::getPassword` natspec indicates a paramter that doesnt exist, causing the natspec to be incorrect

Protocol Summary

Password store 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 mulitple users. Only the owner should be able to set and access this password.

Disclaimer

Squilliam 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 Squilliam 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

```
1234567CommitHash89
```

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

Add some notes about how the audit went, types of things you found, etc.

We spent 5 hours with 1 auditor using foundry as tools.

Issues found

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

Findings

High

[H-1] Storing the password on-chain makes it viable 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 `PaswordStore::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 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

```
make anvil
```

2. Deploy the contract to the chain

```
make deploy
```

3. Run the storage tool We use 1 because thats the storage slot of `s_password` in the contract

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

```
youll get an output that looks like this:
`0
   x6d7950617373776f726400000000000000000000000000000000000000000014
   `

you can then parse that hex into a string with this:
`cast parse-bytes32-string 0
   x6d7950617373776f726400000000000000000000000000000000000000000014
   `

and youll 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, youd 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.

Likelihood & Impact: -Impact: HIGH -Likelihood: HIGH -Severity: HIGH

[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 (so anyone can call it), 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, severly breaking the intention of the contract.

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

```
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 `setPassword` function.

```
if(msg.sender != s_owner){
    revert PasswordStore_notOwner();
}
```

Likelihood & Impact: -Impact: HIGH -Likelihood: HIGH -Severity: HIGH

Informational

[I-1] `PasswordStore::getPassword` natspec indicates a paramter that doesnt exist, causing the natspec to be incorrect

Description:

```
    /*
     * @notice This allows only the owner to retrieve the password.
```

```
3 @>     * @param newPassword The new password to set.
4        */
5       function getPassword() external view returns (string memory) {
```

The `PasswordStore::getPassword` function signature is `getPassword` which the natspec says it should be `getPassword(string)`.

Impact: The natspec is incorrect

Recommended Mitigation: Remove the incorrect natspec line.

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

Likelihood & Impact: -Impact: NONE -Likelihood: HIGH -Severity: INFORMATIONAL/GAS/Non-Crits