Protocol Audit Report

Version 1.0

equious.eth

March 8, 2025

Protocol Audit Report

Afolabi Marvelous

March 3, 2025

Prepared by: Marvelous Afolabi Lead Security Researcher

- Marvelous Afolabi

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 isnot designed to be used by multiple users. Only the owner should be able to set and access the passowrd.

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 correspond the following commit hash:

```
7d55682ddc4301a7b13ae9413095feffd9924566
```

Scope

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

Roles

```
- Owner: The user who can set the password and read the password.
- Outsides: No one else should be able to set or read the password.
```

Executive Summary

- I spent about 3 hours roughly
- Add some notes about how the audit went, types of things and tools used. etx

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` 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.

Impact: Any one can read the password stored on-chain, hence giving people access to the owners password, severly breaking the protocol.

Proof of Concept: The below test case shows how anyone can read the password directly from the blockchain.

1. Create locally running chain

```
make anvil
```

2. Deploy the contract to the chain

```
make deploy
```

3. Run the storage tool. We use `1` because that is the storage slot for `PasswordStore::s_password` variable.

```
cast storage <contract address> 1 --rpc-url http:127.0.0.1:8545
```

We get an output that looks like

```
0x6d7950617373776f726400000000000000000000000000000000000000000014
```

4. Run the parse-bytes32-string tool

```
cast parse-bytes32-string 0
    x6d7950617373776f726400000000000000000000000000000000000000000014
```

we will get an out of

```
myPassword
```

Recommended Mitigation: Due to this, the overall architecture of the contract should be rethrought. One could encrypt the password off-chain, and then store 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.

```
## Likelihood & Impact:

- Impact: HIGH
- Likelihood: HIGH
- Severity: HIGH
```

[H-2] `PasswordStore::setPassword` function has no access control. This basically means non owners can set the password

Description: Any person can change the password as the `PasswordStore::setPassword` function does not have any access control. The `PasswordStore::setPassword` is meant to be only called by the owner but right now, anyone can call this function, hereby changing the 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 change the password, hence defeating the purpose of this function and the protocol is not safe.

Proof of Concept: The below fuzz test shows the error. It shows that anybody can call the `PasswordStore::setPassword` function. Paste the below in your test file and run it.

Code

```
1  function test_fuzz_anyone_can_set_password(address randomAddress)
       public {
2      vm.assume(randomAddress != address(0));
3      vm.startPrank(randomAddress);
4      string memory newPassword = "Marvelous";
5      passwordStore.setPassword(newPassword);
6      vm.stopPrank();
7
8      vm.startPrank(owner);
9      string memory actualPassword = passwordStore.getPassword();
10     assertEq(actualPassword, newPassword);
11     vm.stopPrank();
12 }
```

Recommended Mitigation: Add an access control condition to the `PasswordStore::setPassword` function, so that this runs whwen the function is being called.

```
1  if (msg.sender != s_owner) {
2      revert PasswordStore__NotOwner();
3  }
```

```
1  ## Likelihood & Impact:
2
3  - Impact: HIGH
4  - Likelihood: HIGH
5  - Severity: HIGH
```

Informational

[I-1] TITLE (Root Cause + Impact) The `PasswordStore::getPassword` natspec indicates a parameter that doesn't exists, causing the natspect to be incorrect.

Description:

```
/*
    * @notice This allows only the owner to retrieve the password.
 @=>   * @param newPassword The new password to set.
    */
```

The getPassword does not take any parameter. The natspec would have been correct if the function was to be `getPassword(string)`

Impact: The natspect is incorrect

Recommended Mitigation: Remove the incorrect natspec line

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

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