Protocol Audit Report

Version 1.0

Tim Sigl

June 12, 2024

Protocol Audit Report

Tim Sigl

June 12, 2024

Prepared by: Tim Sigl Lead Security Researcher:

- Tim Sigl

Table of Contents

Protocol Summary

A smart contract application for storing a password. Users should be able to store a password and then retrieve it later. Others should not be able to access the password.

Disclaimer

The Tim-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
Likelihood	High	H	H/M	M
	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 with the following commit hash:

```
2e8f81e263b3a9d18fab4fb5c46805ffc10a9990
```

Scope

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

Roles

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

Executive Summary

This audit took 2 hours and was conducted by a single security researcher. The code was first scanned by using the static analysis tool Slither. Addition tests were executed to show the presence of vulnerabilities. Last, the code was then manually reviewed to identify any additional vulnerabilities.

Issues found

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

Findings

High

[H-1] Storage variables are visible to everyone resulting in the password not being private

Description: All data stored on-chain is visible to everyone. This means that the password stored in the storage variable `PasswordStore::s_password` is not private and get be retrieved without using the `PasswordStore::getPassword()` function.

A PoC is provided in the section below.

Impact: Anyone can read the password, severely compromising the security of the protocol.

Proof of Concept: Proof of Code The following steps allow anyone to read the password from storage:

1. Deploy the contract to a local anvil chain

```
anvil
make deploy
```

2. Inspect the storage location where the password is stored. Its the second storage slot. The first one stores the owner.

```
cast storage <contract-address> 1
```

3. Cast the encoded hex value into a ascii string

```
cast --to-ascii 0
    x6d7950617373776f726400000000000000000000000000000000000000000014
```

4. See the logged string matches the password

```
myPassword
```

Recommended Mitigation: The overall architecture of the protocol builds on the assumption that the password can be saved in storage without being visible to everyone, which is flawed by design. The password could be encrypted off-chain before being stored in storage, and again decrypted off-chain when needed. Although this is possible, it does not make much sense since the user would need to remember a password to encrypt and decrypt the password stored in the contract.

[H-2] `PasswordStore::setPassword` has no access control, allowing anyone to set the password

Description: The function to set the stored password `PasswordStore::setPassword` has no access control to restrict invocation. The intended functionality is to allow the deployer to set the password and deny anyone else from setting it.

Quote from the NatSpec:

> This function allows only the owner to set a new password.

Impact: Anyone can set the password, compromising the intended functionality of the protocol.

Proof of Concept: The following tests proofs that anyone can set the password:

Code Snippet

```
function test_anyone_can_set_password(address randomAddress) public {
        vm.assume(randomAddress != owner);
```

```
3           vm.prank(randomAddress);
4           string memory passwordFromAStranger = "youGotPwnd";
5           passwordStore.setPassword(passwordFromAStranger);
6           vm.prank(owner);
7           string memory actualPassword = passwordStore.getPassword();
8           assertEq(actualPassword, passwordFromAStranger);
9       }
```

Recommended Mitigation: Add an access control conditional or modifier to the `PasswordStore::setPassword` function to restrict invocation to the owner.

Code Snippet

```
1     function setPassword(string memory newPassword) external {
2           if (msg.sender != owner) {
3               revert PasswordStore__NotOwner();
4           }
5           s_password = newPassword;
6           emit SetNetPassword();
7       }
```

Informational

[I-1] The NatSpec in `PasswordStore::getPassword` suggests a parameter that is not present in the function signature

Description: The NatSpec in `PasswordStore::getPassword` suggests that a newPassword should be passed to the function, but the function signature does not include a parameter for newPassword.

Impact: Misleading documentation can lead to confusion and misunderstanding of the intended functionality.

Recommended Mitigation: Remove the parameter from the NatSpec in `PasswordStore::getPassword` to align with the function signature.

```
1       /*
2        * @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) {
```