



Protocol Audit Report

Prepared by: Ricardo Mazuera

Table of Contents

- [Table of Contents](#)
- [Protocol Summary](#)
- [Disclaimer](#)
- [Risk Classification](#)
- [Audit Details](#)
 - [Scope](#)
 - [Roles](#)
- [Executive Summary](#)
 - [Issues found](#)
- [Findings](#)
 - [High](#)
 - [\[H-1\] On-chain passwords are visible to everyone, regardless of Solidity variable visibility.](#)
 - [\[H-2\] PasswordStore::setPassword is callable by anyone](#)

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 Ricardo Mazuera 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 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

Issues found

Severity	Number of issues found
High	2
Medium	0
Low	0
Info	0
Gas Optimizations	0
Total	2

Findings

High

[H-1] On-chain passwords are visible to everyone, regardless of Solidity variable visibility.

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.

However, anyone can directly read this using any number of off chain methodologies

Impact: The password is not private.

Proof of Concept: The below test case shows how anyone could 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:

[illegible]

You can then parse that hex to a string with:

[illegible]

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 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.

[H-2] PasswordStore::setPassword is callable by anyone

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.`

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

Impact: Anyone can set/change the password of the contract.

Proof of Concept:

Add the following to the `PasswordStore.t.sol` test suite.

```
function test_anyone_can_set_password(address randomAddress) public {  
    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 modifier to the `setPassword` function.

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