



Protocol Audit Report

Prepared by: Ya-Sin

Table of Contents

- [Table of Contents](#)
- [Protocol Summary](#)
- [Risk Classification](#)
- [Audit Details](#)
 - [Scope](#)
 - [Roles](#)
- [Executive Summary](#)
 - [Issues found](#)
- [Findings](#)
- [High](#)
 - [\[H-1\] Storing the password on-chain makes it visible to anyone and no longer private.](#)
 - [\[H-2\] PasswordStore::setPassword has no access controls, meaning a non-owner could change the password.](#)
- [Informational](#)
 - [\[I-1\] The PasswordStore::getPassword natspec indicates a parameter that doesn't exist, causing the natspec to be incorrect](#)

Protocol Summary

A smart contract applicatoin 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.

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 :

2e8f81e263b3a9d18fab4fb5c46805ffc10a9990

Scope

```
./src/  
└─ PasswordStore.sol
```

Roles

Users should be able to store a password and then retrieve it later. Others should not be able to access the password.

Executive Summary

Issues found

Severity	Numbers 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` 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.

Whe 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:

The below test case proves it.

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 would 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` has no access controls, meaning a non-owner could change the password.

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 {
    @>    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);
}
```

The below test case proves it.

Recommended Mitigation: Add an access control conditional to the `setPassword` function.

Informational

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

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

Impact: The natspec is incorrect.

Recommended Mitigation: Remove the incorrect natspec line

+ * @params