Protocol Audit Report

Version 1.0

Cyfrin.io

January 12, 2025

PasswordStore Audit Report

Shoaib Khan

January 12, 2025

PasswordStore Audit Report

Prepared by: Shoaib Khan

Table of contents

See table

About Shoaib

Hi, I'm Shoaib—a passionate blockchain and smart contract developer who loves diving deep into the world of decentralized technology. This report marks a significant milestone in my journey as it is my very first audit report.

My goal with this audit is not just to ensure code quality and security but also to set a benchmark for thoroughness and precision. This report reflects my commitment to learning, growing, and delivering value to the blockchain community. Every finding and observation in this report is a step toward building a safer and more robust decentralized ecosystem.

As I continue this journey, I'm excited to refine my skills further, embrace challenges, and contribute meaningfully to the tech world. Thank you for trusting me with this task—I'm just getting started!

Protocol Summary

The `PasswordStore` protocol aims to provide a way for users to store sensitive information (like passwords) on-chain. The protocol offers password retrieval and update functionality, designed for owner-only access. However, critical flaws exist in its implementation that undermine its core purpose of ensuring security and privacy.

Disclaimer

The SHOAIB KHAN team made every effort to find as many vulnerabilities as possible within the given time frame. However, this report does not guarantee complete coverage or an endorsement of the underlying business or product. The audit focused solely on the security aspects of the Solidity implementation of the contracts and was time-boxed.

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 levels. Refer to the documentation for further details.

Audit Details

The findings described in this document correspond the following commit hash:

```
dd4ba666741997f5c5346392f4e5b5516e64168d
```

Scope

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

Roles

- Owner: Is the only one who should be able to set and access the password.

For this contract, only the owner should be able to interact with the contract.

Executive Summary

The audit of the `PasswordStore` protocol revealed critical vulnerabilities that compromise the security and privacy of the stored passwords. Two high-severity issues were identified: the visibility of passwords stored on-chain and the lack of access control in the `setPassword` function. Additionally, an informational issue was found regarding incorrect NatSpec documentation. No medium or low-severity issues were discovered. The audit highlights the need for significant improvements to ensure the protocol's intended security guarantees.

Issues found

Severity	Number of issues found
High	2
Medium	0
Low	1
Info	1
Gas Optimizations	0
Total	0

Findings

High

[H-1] Passwords stored on-chain are visable to anyone, not matter 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 direclty 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:

`0x6d7950617373776f726400000000000000000000000000000000000000000014`

You can then parse that hex to a string with:

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

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);
```

```
8 }
```

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

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

Informational

[I-1] The `PasswordStore::getPassword` Natspec indicates a parameter that doesn't exist

Description: The NatSpec for `getPassword` mentions a `@param newPassword` that does not exist in the function's signature. This inconsistency can cause confusion.

```
    /*
     * @notice This allows only the owner to retrieve the password.
@>   * @param newPassword The new password to set.
     */
    function getPassword() external view returns (string memory) {
```

The natspec for the function `PasswordStore::getPassword` indicates it should have a parameter with the signature `getPassword(string)`. However, the actual function signature is `getPassword()`.

Impact: The natspec is incorrect.

Recommended Mitigation: Remove the incorrect natspec line.

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