PasswordStore Audit Report

Version 1.0

0xNTN

April 6, 2025

PasswordStore Audit Report

0xNTN

April 6, 2025

Prepared by: 0xNTN

Table of Contents

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.

Disclaimer

The 0xNTN 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:

```
2e8f81e263b3a9d18fab4fb5c46805ffc10a9990
```

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

Issues found

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

Findings

High

[H-1] Storing password on-chain makes it visible to everyone, and no longer private

Description: Variables stored on-chain are visible for everyone, no matter of the Solidity visibility keyword meaning that the password is not realy private password. 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 anly called by the owner of the contract.

We show one such method of reading any data off chain below.

Impact: Anyone can read the private password, severly breaking the functionality of the protocol.

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

1. Create a locally running chain

```
make anvil
```

2. Deploy the contract on the chain

```
make deploy
```

3. Run the storage tools

We use 1 as a storage slot because that is the storage slot of `PasswordStore::s_password` in the contract.

```
cast storage <CONTRACT_ADDRESS_HERE> 1 --rpc-url http://127.0.0.1:8545
```

You'll get an output that looks like this: `0x6d7950617373776f726400000000000000000000000000000000`

You can then parse that hax 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()` has missing access controll, meaning everyone can change the password

Description: The `PasswordStore::setPassword()` function has missing check for the contract owner before set/change the password, however, the matspec of the function says `This function allows only the owner to set a new password.`

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

Impact: Anyone can set or change the password of the contract, severly breaking the contract intended functionality.

Proof of Concept: Add the following to `PasswordStore.t.sol` test file

Code

```
function test_anyone_can_set_password(address randomAccount) public {
    vm.assume(randomAccount != owner);
    string memory expectedPassword = "myNewPassword";

    vm.prank(randomAccount);
    passwordStore.setPassword(expectedPassword);

    vm.prank(owner);
    string memory actualPassword = passwordStore.getPassword();
    assertEq(actualPassword, expectedPassword);
}
```

Recommended Mitigation: Add access control check to the `PasswordStore::setPassword()` method.

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

Informational

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

Description:

```
    /*
    * @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 `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.

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