PasswordStore Audit Report

Version 1.0

Stunspotx

Sat Jan 24 2026

Prepared by: Stunspotx

Lead Security Researcher:

- Sunshine

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 Stunspotx 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
	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

- Owner: The user who can set the password and read the password.
- Outsiders: 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

Severity	Number of issues found
Informational	1
Total	3

Findings

High

[H-1] password storage variable declared as private can be read by anyone

Description: The `PasswordStore::s_password` variable was declared as private but is not actually private since it can be read publicly from the storage in the blockchain. Once the password is set through the `PasswordStore::setPassword` function, the data is stored in a storage slot and will be visible to anyone.

The POC shows how this intended private variable can be publicly accessed via the blockchain.

Impact: Anyone can read the private password, severely breaking the core functionality of the blockchain.

Proof of Concept: The below scenario shows how anyone can publicly read data from the blockchain:

1. Create a local running chain:

```
make anvil
```

2. Deploy the script:

```
make deploy
```

3. Run the cast storage tool:

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

You will get an output like this:
`0x6d7950617373776f726400000000000000000000000000000000000000000014`

4. Convert hex output to string:

```
cast to-utf8
0x6d7950617373776f726400000000000000000000000000000000000000000014
```

This will give you the password in plain readable text

Recommended Mitigation: Due to this, the overall architecture of the contract should be rethought. One could encrypt the password offchain and then store the encrypted password on-chain. This would require the user to remember another password offchain 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 his encrypted password.

[H-2] `PasswordStore::setPassword` has no access control, meaning a non-owner could change the password.

Description: According to the natspec of the `PasswordStore::setPassword` function, `the function allows only the owner to set a new password`, but the function does otherwise, allowing anyone to call it and putting the protocol in severe danger.

```
    function setPassword(string memory newPassword) external {
@>    // No access control
        s_password = newPassword;
        emit SetNetPassword();
    }
```

Impact: Anyone can set/change password, severely breaking the intended functionality of the contract.

Proof of Concept: Below is the proof of code that anyone can set password. Add the following to the `PasswordStore.t.sol` test file:

▶ code

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

        string memory newPassword = "HackedPassword";

        vm.prank(_randomAddr);
        passwordStore.setPassword(newPassword);

        vm.prank(owner);
        string memory getPassword = passwordStore.getPassword();

        assertEq(getPassword, newPassword);
    }
```

Recommended Mitigation: Add an access control mechanism/condition to the `PasswordStore::setPassword` function:

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

Informational

[I-1] `newPassword` is not a parameter in `PasswordStore::getPassword` function

Description: According to the natspec of the `PasswordStore::getPassword` function, `newPassword` is supposed to be a parameter of the function, but no parameter was provided in the function.

```
    /*
     * @notice This allows only the owner to retrieve the password.
     // @audit getPassword function does not take in any parameters.
->     * @param newPassword The new password to set.
    */
```

Impact: This results in a faulty documentation.

Recommended Mitigation: Simply take out the wrong natspec or specify a parameter in the `PasswordStore::getPassword` function, if it takes in any.

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