PasswordStore Audit Report

SulfurPT

February 28, 2025

PasswordStore Audit Report

SulfurPT

February 28, 2025

Document Version Control

Report	
Version	1.0
State	Final
Date	28/02/2025
Auditors	SulfurPT
Approved by	SulfurPT

Version	Date	Description	Pages
1.0	28/02/2025	Document Creation	All

Table of Contents

Protocol Summary

PasswordStore is 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

I, SulfurPT, make all effort to find as many vulnerabilities in the code in the given time period, but hold 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

		Impact		
	High	H	H/M	M
Likelihood	Medium	H/M	M	M/L
	Low	M	M/L	L

Audit Details

Commit Hash: 2e8f81e263b3a9d18fab4fb5c46805ffc10a9990

Scope

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

Roles

Owner - Only the owner may set and retrieve their password

Executive Summary

This report presents the findings from the security audit conducted by SulfurPT on the PasswordStore smart contract. The primary objective of the audit was to assess the security, reliability, and efficiency of the contract's implementation. The audit follows industry best practices and security methodologies to identify potential vulnerabilities that could compromise the integrity and confidentiality of user data.

Issues found

Severaty	Number of issues found
High	2
Medium	0

Severaty	Number of issues found
Low	0
info	1
Total	3

Findings

High

[H-1] Variable password is visable to anyone on-chain

Description: All data store on-chain is visible to anyone and it 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, but, it is visible to everyone on-chain.

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

Proof of Concept: Above are the steps, to create the PoC:

Steps

Create a locally running chain

```
make anvil
```

Deploy the contract to the chain

```
make deploy
```

Run the storage tool on the storage slot for `PasswordStore::s_password`

```
cast storage <ADDRESS_HERE> 1 --rpc-url http://LocalAnvilIP:Port
```

You should get the output:

```
0x6d7950617373776f726400000000000000000000000000000000000000000014
```

You can then parse that hex to a string with:

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

And get an output of the password:

```
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 stored password. However, you're also likely want to remove the view function as you wouldn't want the user to accidentally send a transaction with this decryption key.

[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 purpose of the smart contract and function's natspec indicate that `This function allows only the owner to set a new password.`

```
    function setPassword(string memory newPassword) external {
->      // @Audit - There are no Access Controls.
        s_password = newPassword;
        emit SetNetPassword();
    }
```

Impact: Anyone can set/change the stored password, severely breaking the contract's intended functionality

Proof of Concept:

Add the following to the PasswordStore.t.sol:

Code

```
    function test_notOwner_can_set_password(address randomAddress)
        public {
        vm.assume(randomAddress != owner);
        vm.startPrank(randomAddress);
        string memory expectedPassword = "notOwnerPassword";
        passwordStore.setPassword(expectedPassword);

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

Recommended Mitigation:

Add an access control in the `PasswordStore::setPassword` function

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

Informational

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

Proof of Concept:

N/A

Recommended Mitigation:

Remove the incorrect natspec line

```
+ line you want to add (shown in green)
- line you want to remove (shown in red)
```