SMART CONTRACT SECURITY AUDIT REPORT

Project Name: PasswordStore Audit Report

Date: 22-02-2025

Auditor: Ashiq Ahamed

Audit Version: 1.0

Table of Contents

---

1. Introduction

This security audit was conducted on the `PasswordStore` smart contract to identify security vulnerabilities and ensure the reliability of its implementation. The primary objective was to detect critical, high, and informational issues that could impact contract security and functionality.

2. Scope of Audit

The audit focused on reviewing the Solidity smart contract implementation, analyzing: - **Access Control Vulnerabilities** - **Storage and Privacy Risks** - **Logical and Functional Bugs**

3. Findings Summary

ID	Title	Severity
H-1	Storing the password on-chain makes it visible	High/Critical
H-2	The `setPassword` function lacks access control	High/Critical
I-1	Incorrect natspec documentation in `getPassword`	Informational

4. Detailed Findings

[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` is intended to be a private variable and can be only accessed through the `PasswordStore::`

`getPassword` function, which is intended to only called by the owner of the contract.

We show one such method of reading data off-chain below

Impact:

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

Proof of Concept:

(Proof of Code)

The below test case shows how anyone can read the password directly from the blockchain.

1. Create a locally running chain

```
make anvil
```

1. Deploy the contract to the chain

```
make deploy
```

3.Run the local tool

```
cast storage <YOUR_ADDRESS> 1 --rpc-url 127.0.0.1:8545
0x6d7950617373776f726400000000000000000000000000000000000000000014
```

`1` is the storage slot fo the `PasswordStore::s_password` variable and `0x6d7950617373776f726400000000000000000000000000000000000000000014` is the password in bytes32

You can then parse the bytes32 into a string by:

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

And get an output of `myPassword`

Recommended Mitigation:

Due to this the overall contract has to be rethought.

Likelihood and Impact:

- Impact: HIGH
- Likelihood: HIGH
- Severity: HIGH || CRITICAL

[H-2] The `PasswordStore::setPassword` has no access control, meaning anyone can change the password.

Description:

The `PasswordStore::setPassword` function can only be called by the owner as per the natspec `* @notice This function allows only the owner to set a new password` , but as the function is external and there is no access control non-owners can also call this function.

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

Impact:

Anyone can set/update the password of the contract, severly breaking the contract intended functionality

Proof of Concept:

Add the following the `PasswordStore.t.sol` test file.

```
function test_anyone_can_set_password(address randomAddress)
        public {
    vm.assume(randomAddress != owner);
    string memory expectedPassword = "MyPassword123";

    //Prank as some random address
    vm.prank(randomAddress);

    // Set the password
    ///@notice this should not be called by any random
        address except the owner
->  passwordStore.setPassword(expectedPassword);

    // Prank as the owner of this contract
    vm.prank(owner);
    // Call the function getPassword as the owner, this can
        be called only by the owner.
    string memory actualPassword =
        passwordStore.getPassword();

    assertEq(actualPassword, expectedPassword, "Different
        password");
    }
```

Recommended Mitigation:

Add an access control condition to the `setPassword` function (or) add an onlyOwner modifier to the `setPassword` function

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

Or use the onlyOwner modifier

```
// OnlyOwner modifier
modifier onlyOwner() {
    require(s_owner == msg.sender);
    _;
}

// use the onlyOwner modifier in the setPassword function
function setPassword(string memory newPassword) external
        onlyOwner {}
```

Likelihood and Impact:

- Impact: HIGH
- Likelihood: HIGH

- ## Severity: HIGH || CRITICAL

[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.
*/
//@audit there is no new password param
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.
```

Likelihood and Impact:

- Impact: NONE
- Likelihood: HIGH
- Severity: Informational/Gas/Non-Crits

Conclusion

This audit highlights critical vulnerabilities in the PasswordStore contract, including password exposure and lack of access control. The contract requires fundamental design changes to ensure security.

Key Recommendations:

```
•   Never store private data on-chain.
•   Implement strict access control mechanisms.
•   Ensure accurate documentation (natspec comments).
```

By fixing these issues, the contract can be made more secure and robust.