PasswordStore Audit Report

Version 0.1

Prepared by: Kzlandx

Date: March 10, 2025

Table of contents

About Kzlandx

I am a beginner smart contract security researcher. Getting better everyday!

Disclaimer

The Kzlandx team makes all effort to find as many vulnerabilities in the code in the given time period, but holds no responsibilities for the 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

Audit Details

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

```
2e8f81e263b3a9d18fab4fb5c46805ffc10a9990
```

Scope

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

Protocol Summary

PasswordStore is a protocol dedicated to storage and retrieval of a user's passwords. The protocol is designed to be used by a single user, and is not designed to be used by multiple users. Only the owner should be able to set and access this password.

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.

Issues found

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

Findings

High

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

Description: Data stored on-chain is visible to anyone, no matter its visibility specifier (public, private or internal). The `PasswordStore::s_password` variable though intended to be visible only to the owner through the use of `PasswordStore::getPassword` function, can actually be read by anyone.

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

Impact: Anyone can read the private password, severely 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
```

2. Deploy the contract

```
make deploy
```

3. Run the storage tool

```
cast storage <CONTRACT_ADDRESS> 0x1 --rpc-url http://localhost:8545
```

You'll get an output that looks like this:
`0x6d7950617373776f726400000000000000000000000000000000000000000014`

You can the parse that hex to a string with:

```
cast --to-ascii 0x6d7950617373776f726400000000000000000000000000000000000000000014
```

OR

```
cast parse-bytes32-string 0x6d7950617373776f72640000000000000000000000000000000000000
```

And you will 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, then store the encrypted password on-chain. But again, this would require the user to remember another password off-chain to decrypt the encrypted password. You would also likely want to remove the view function as you would'nt want the user to user to accidentally send a transaction with the password that decrypts the encrypted password.

[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 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, severely breaking the intended contract functionality.

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

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

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

Recommended Mitigation: Add an access control conditional to the `setPassword()` function.

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

Low

[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()` which the natspec say, it should be `getPassword(string)`.

Impact: The natspec is incorrect.

Recommended Mitigation: Remove the incorrect natspec line.

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