Protocol Audit Report

Version 1.0

github.com/Zyrrow

November 3, 2024

Protocol Audit Report

Zyrrow

October 26 , 2024

Prepared by: [Zyrrow] Lead Auditors:

- Zyrrow

Table of Contents

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

- Gas

Protocol Summary

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

Zyrrow 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 me 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: 7d55682ddc4301a7b13ae9413095feffd9924566

Scope

./src/

- PasswordStore.sol

Roles

-Owner: The user who can set the password and read the password. -Outsides: No one else should be able to set or read the password.

Executive Summary

This report details security vulnerabilities identified in the PasswordStore contract. The discovered issues compromise the confidentiality of sensitive information and the integrity of specific contract functions. Security flaws include insufficient access control, improper handling of sensitive data visibility, and documentation errors, all of which impact the contract's security and reliability.

Issues found

Severity	Issue
High	Storing the password in plain text on the blockchain
High	Lack of access control on the `setPassword` function
Informational	Incorrect natspec documentation

Findings

Actually 2 vulnerabilities found and 1 informationnal issue.

High

[H-1] Storing the password on-chain makes it visable to anyone, and no longer private (Root Cause + Impact)

Description: All data on the blockchain is visible to anyone and can be directly read from the blockchain. The `PasswordStore::s_password` variable is intended to be a private variable and should only be accessed through the `PasswordStore::getPassword` function, which is intended to be called solely by the contract owner. Below, we demonstrate a method to read any on-chain data.

Impact: Everyone 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 to the chain

```
make deploy
```

3. Run the storage tool we use 1 because that's the storage slot of `s_password` in the contract.

```
cast storage <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 hex to a string with:

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

And get an output of : myPsassword

Recommended Mitigation: As a result, the contract's overall architecture may need to be reconsidered. One approach could be to encrypt the password off-chain and store only the encrypted version on-chain. This would require the user to remember an additional off-chain password for decryption. Additionally, it may be wise to remove the view function to avoid the risk of users inadvertently sending a transaction containing 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.`

Impact: Anyone can set/change the stored password, severly breaking the contract's intended 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 = "mynewpassword";
    passwordStore.setPassword(expectedPassword);

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

Recommended Mitigation: Add an access control conditionnal to the `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:**

Recommended Mitigation: Remove the incorrect natspec line

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

Gas