Protocol Audit Report

Sodiq Adewole

August 9, 2024

Protocol Audit Report

Version 1.0

SodiqAdewole

August 11, 2024

Protocol Audit Report

Sodiq Adewole

August 9, 2024

Prepared by: [Sodiq Adewole] Lead Security Researcher: - Sodiq Adewole

Table of Contents

Protocol Summary

PasswordStore is a protocol dedicated to storage and retrieval of user's password. The protocol is designed to be used by only one user and not to be used by multiple users. Only the user should be able to set and change the password.

Disclaimer

The Sodiq Adewole 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
Likelihood	High	H	H/M	M
	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

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

`7d55682ddc4301a7b13ae9413095feffd9924566`

Scope

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

Roles

- Owners: The user can set and read the password.
- Outsiders: No one should be able to set or read the password.

Executive Summary

Issues found

Severity	Number Of Issues Found
High	2
Medium	0
Low	0
Info	1
Total	3

Findings

High

[H-1] Storing the password on-chain makes it visible to anyone, and the password 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` variable is intended to be private and can only be accessed to through the `PasswordStore::getPassword` function, which is intended to be only called by the owner of the contract.

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

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

Proof of Concept: (Proof Of Code) The below test shows anyone can read the password from the blockchain

1. Create a locally running chain

```
make anvil
```

2. Deploy the smart 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 1 –rpc-url http://127.0.0.1:8545

You will get an ouput that looks like this

```
0x6d7950617373776f726400000000000000000000000000000000000000000014
```

You can parse that hex with a string with:

cast parse-bytes32-string `0x6d7950617373776f726400000000000000000000000000000000000000000014`

And 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 and then store the encrypted password on-chain. This would require the user to remember another password to decrypt the password. However, you would also likely to remove the view function as you shouldn't want the user to send a transaction that decrypts your password.

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

Description: The `Password::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
       s_password = newPassword;
       emit SetNetPassword();
   }
```

Impact: Anyone can set/change the password of the contract, severly breaking the contract intended functioinality

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

Code

```
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 conditional to the `setPassword` function

```
if(msgs.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 original natspec incorrectly indicated 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: Information/Gas/Non-crits

Whenever you have the impact being none and Likelihood being high or interchangeably, the severity will be Information/Gas/Non-crits