Protocol Audit Report

Version 1.0

Hamidov Siyovush

January 17, 2024

Protocol Audit Report

Hamidov Siyovush

Jan 17, 2024

Prepared by: Hamidov Siyovush Lead Auditors: - Hamidov Siyovush

Table of Contents

Protocol Summary

PasswordStore is a protocol dedicated to storage and retrieval of a user's passwords. The protocol is designed to be used 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.

Disclaimer

The YOUR_NAME_HERE 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
	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

Scope

```
./srs/PasswordStore.sol
```

Roles

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

Executive Summary

We have learned a lot of usefool techniques to handle smart-contract audit issues, spending on it 3 days in a row.

Issues found

Severity	Number of issues found
High	2
Medium	0
Low	0
Info	1
Total	3

Findings

High

[High-1] Storing the Password On-Chain Makes It Visible to Anyone

Description All data stored on-chain is public and visible to anyone. The `PasswordStore::s_password` variable is intended to be hidden and only accessible by the owner through the `PasswordStore::getPassword` function.

Below, I will demonstrate a method of reading any data off-chain.

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

Proof of Concept The following test case shows how anyone could read the password directly from the blockchain. We use Foundry's `cast` tool to read directly from the storage of the contract, without being the owner.

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 receive an output that looks like this:

   ```
   0x6d7950617373776f726400000000000000000000000000000000000000000014
   ```

   You can then parse that hex to a string with:

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

   And get an output of:

   ```
   myPassword
   ```

Recommended Mitigation Due to this issue, 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 would also likely want to remove the view function as you wouldn't want the user to accidentally send a transaction with this decryption key.

[High-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 the 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 SetNewPassword();
}
```

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

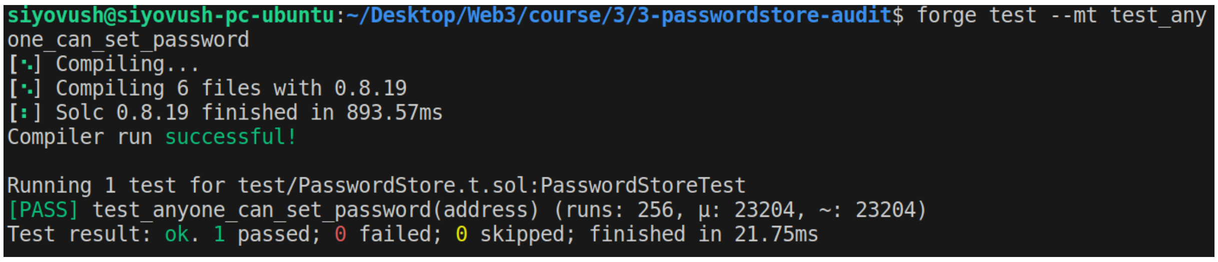
Proof of Concept Add the following to the `PasswordStore.t.sol` test file:

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

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

Recommended Mitigation Add an access control conditional to `PasswordStore::setPassword`.

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



```
siyovush@siyovush-pc-ubuntu:~/Desktop/Web3/course/3/3-passwordstore-audit$ forge test --mt test_any
one_can_set_password
[⠊] Compiling...
[⠊] Compiling 6 files with 0.8.19
[⠒] Solc 0.8.19 finished in 893.57ms
Compiler run successful!

Running 1 test for test/PasswordStore.t.sol:PasswordStoreTest
[PASS] test_anyone_can_set_password(address) (runs: 256, μ: 23204, ~: 23204)
Test result: ok. 1 passed; 0 failed; 0 skipped; finished in 21.75ms
```

Figure 1: Alt text

Informational

[Informational-1] The `PasswordStore::getPassword` NatSpec Indicates a Parameter That Doesn't Exist, causing the natspec tp be incorrect.

Description The `PasswordStore::getPassword` function signature is `getPassword()`, while the NatSpec suggests it should be `getPassword(string)`.

```
/**
 * @notice This allows only the owner to retrieve the password.
 * @param newPassword The new password to set.
 */
function getPassword() external view returns (string memory) {}
```

Impact The NatSpec is incorrect.

Recommended Mitigation Remove the incorrect NatSpec line.

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

- Impact: NONE
- Likelyhood: NONE
- Severity Informational/Gas/Non-crits