Protocol Audit Report

Prepared by: Rusty Metalhead

Date: 16th June 2024

Table of Contents

- Table of Contents
- Protocol Summary
- Disclaimer
- Risk Classification
- Audit Details
  - Commit Hash
  - Scope
  - Roles
  - Issues found
- Findings
- High
- Medium
- Low
- Informational
- Gas

Protocol Summary

Protocol does X, Y, Z

Disclaimer

The Rusty Researchers 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

Commit Hash

7d55682ddc4301a7b13ae9413095feffd9924566

Scope

```
./src/
└── PasswordStore.sol
```

Roles

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

Issues found

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

Findings

High

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

Description: All data stored on-chain is visible to anyone, and can be ready directly from the blockchain. The `PasswordStore::s_password` variable is intended to be a private variable and only accessed through the `PasswordStore::getPassword` function which is intended to be called only by the owner of the contract. The private visibility states that only the current contract and no other contracts can access the variable.

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

Proof of Concept: The below steps show how anyone can read the password directly from blockchain

1. Create a locally running chain

```
anvil
```

2. Deploy the contract to the chain

```
make deploy
```

3. Run the storage tool

```
cast storage <CONTRACT_ADDRESS> 1 --rpc-url http://127.0.0.1:8545
```

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

You can then parse this hex to a string with

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

Following would be the output:

```
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 on-chain to decrypt the password. However, you'd also likely want to remove the view function as you wouldn't want the user to accidentally send a transaction with the password that decrypts your password.

[H-2] `PasswordStore::setPassword` has no access control, meaning anyone could change the password of the contract owner.

Description: Since the `PasswordStore::setPassword` is an external function, anyone/ any contract can call the function. We want only the contract owner be able to change the password. 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/ contract owner.

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

▶ Code

```
function test_anyone_can_set_password(address randomAddress) public {
    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 modifier 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) {
        if (msg.sender != s_owner) {
            revert PasswordStore__NotOwner();
        }
        return s_password;
    }
```

Impact: The natspec is incorrect.

Recommended Mitigation:

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