Protocol Audit Report

Version 1.0

ZenoraSec

July 14, 2024

Protocol Audit Report

ZenoraSec

July 14, 2024

Prepared by: ZenoraSec Lead Security Researcher: Jeremy Zenora

Table of Contents

    * [I-1] The Natspec of function `PasswordStore::getPassword` wrongly implies a parameter that does not exist
  - Gas

Protocol Summary

Protocol handles storage and retrieval of user's password. Protocol is meant for one single user. Only the owner of the contract (single user) can change and access the password.

Disclaimer

The ZenoraSec 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

Following github commit hash was used during the security review:

```
7d55682ddc4301a7b13ae9413095feffd9924566
```

Scope

```
./src/
#-- 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 found a few major issues that contradicted the logic and original use of the protocol. We recommend to rethink the way the whole protocol works as suggested in the findings below. We used foundry framework with tools such as fuzzing, symbolic/formal verification, manual review.

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 is visible for everyone to see, and thus no longer private

Description:

All data stored on-chain is visible for everyone to see, it can be read easily from the blockchain. The "private" keyword for the variable `PasswordStore::s_password` does not hide it.

Impact:

Anyone can read the private password, breaking the meaning of the function `PasswordStore::getPassword()` which was meant to be the only way of reading the password.

Proof of Concept:

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 is the storage slot of `s_password` in the contract.

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

You will get output like this `0x6d7950617373776f72640000000000000000000000000000000000`

You can parse that hex to a string with:

```
cast parse-bytes32-string 0
    x6d7950617373776f7264000000000000000000000000000000000000000000000014
```

You can then parse that hex to a string with:

You then get an output of:

```
myPassword
```

Recommended Mitigation: Due to this, you should consider encrypting the password off-chain and then store the encrypted password hash on-chain. User would then need to remember another password off-chain to decrypt the password. Also you should remove the function to show the password entirely, as you do not want your user to expose the password on othe blockchain.

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

Description: `PasswordStore::setPassword` does not have any way to prevent non-owners to change the password such as a modifier onlyOwner or logic in the function to check if the msg.sender equals to the owner.

```
function setPassword(string memory newPassword) external {
 @>     //@audit-high acces control exploit, any user can set a
    password
        s_password = newPassword;
        emit SetNetPassword();
    }
```

Impact: Password can be changed at any time by any user, thereby anyone can know the password of the owner.

Proof of Concept:

type this into the terminal: `forge test --mt test_anyone_can_change_password` this test proves that random address could change password even though it was not the owner.

here's the function code block:

Code

```
function test_anyone_can_change_password(address randomAddress) public
    {
        vm.assume(randomAddress != owner);
        vm.prank(randomAddress);
        string memory newPassword = "thisWasntOwnerPassword";
        passwordStore.setPassword(newPassword);

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

Recommended Mitigation: To restrict usage of the function specificly to the owner, you can put this small code logic at the beginning of the function:

```
if (msg.sender != s_owner) {
    revert PasswordStore__NotOwner(); //this makes sure only the owner
       can set password.
}
```

Medium

Low

Informational

[I-1] The Natspec of function `PasswordStore::getPassword` wrongly implies a parameter that does not exist

Description:

The comment indicates that there should be a parameter in the function, but there is none.

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

Function signature is currently `getPassword()` but natspec says `getPassword(string)`.

Impact: Wrong natspec of the function.

Recommended Mitigation: Remove the incorrect natspec line.

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

Gas