Protocol Audit Report

Bikalpa Regmi

April 13, 2025

Prepared by: Bikalpa Regmi
Lead Security Researcher: Bikalpa Regmi

…

Table of Contents

Protocol Summary

PasswordStore is a protocol that is dedicated to storage and retrieval of a user's passwords. The Protocol is designed to be used by a single user, and is not designed to be used by multiple users. Only the owner should be able to set and retrieve the password.

Disclaimer

The Bikalpa Regmi 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

The findings described in this document corresponds to the following commit hash :

`7d55682ddc4301a7b13ae9413095feffd9924566`

Scope

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

Roles

- Owner : The user who can read the password and set the pasword.
- Outsiders : Who can not read or set the password.

Executive Summary

We spent 12 hours with one auditor using foundry test tool.

Issues found

Severity	Number of Issue Found
High	2
Medium	0
Low	0
Info	1
Total	3

Findings

High

[H-1] The variables stored in a blockchain are visible to anyone even if the visibility keywords are private.

Description : Every data on a blockchain is visible to anyone, and can be read directly from blockchain. The `PasswordStore::s_password` is intended to be a private variable and can only be accesed through `PasswordStore::getPassword` function which is intended to be the owner.

We show one such method of reading any offchain data below.

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

Proof of Concept : (Proof of code)

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 thats the storage slot for `s_password` in the contract.

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

you will get an hash output

you can then parse that hex to a string with :

```
cast parse-bytes32-string _Above_hash_string
```

And get an output of:

```
myPassword
```

Recommended Mitigation : Due to this, the overall architecture of a contract should be rethought. One can encrypt the password offchain and then encrypt the password onchain. This would require the user to remember the password offchain to decrypt the password. However, you would also likely want to remove the view function as you would not want the user to accidently send a transaction with the password that decrypt your password.

[H-2] `PasswordStore::setPassword` has no access control, meaning the non owner can easily change the password.

Description : The `PasswordStore::setPassword` function is set to be an `external` function, However, the natspecs of the function and overall function of a smart contract is that `This function allows only the owner to set the new password`.

```
function setPassword(string memory newPassword) external {
@>//@audit there are no access controls
        s_password = newPassword;
        emit SetNetPassword();
    }
```

Impact : Anyone can change the password of a contract, severly breaking the contract intended functionality.

Proof of Concept : Add the following to the PasswordStore.t.sol 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 condition to `PasswordStore::setPassword` function.

```
if(msg.sender!=owner){
    revert PasswordStore_NotOwner();
}
```

Informational

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

Description:

```
/*
     * @notice This allows only the owner to retrieve the password.
@>     * @param newPassword The new password to set.
     * @audit This function lacks params and it is not required.
     */
    function getPassword() external view returns (string memory) {}
```

The `PasswordStore::getPassword` function signature is `getPassword()` which the natspec says it should be `getPassword(string)`.

Impact: The natspec is incorrect.

Recommended Mitigation: Remove the incorrect natspec line.

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