Protocol Audit Report

Version 1.0

Radcipher

January 27, 2025

Protocol Audit Report

Radcipher

January 27, 2025

Prepared by: Radcipher Lead Security Researcher: - Radcipher

Table of Contents

Protocol Summary

PasswordStore is a protocol specifically designed for securely storing and retrieving a user's passwords. It is intended for single-user usage and is not built to support multiple users. Only the owner of the passwords has the ability to manage and control access to them.

Disclaimer

The Radcipher team makes every effort to identify as many vulnerabilities as possible within the given time frame. However, the team accepts no responsibil-

ity for the findings presented in this document. A security audit conducted by the team does not imply an endorsement of the underlying business or product. The audit was time-limited and focused exclusively 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

```
./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

Executive Summary

- We dedicated approximately 2.5 hours, working with a lead security researcher, to identify all vulnerabilities and prepare the report using foundry testing on a local testnet.

Issues found

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

Severity	Number of issues found
Total	3

Findings

High

[H-1] Storing the password on-chaing makes it visible to anyone and no longer private.

Description: All data on-chain is visible to anyone, and can be read 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 only called by the owner of the contract. We show one such method of reading any data off-chain below.

Impact: Anyone 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 chaing

```
make anvvil
```

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 will get an output that looks like this:

```
0x6d7950617373776f726400000000000000000000000000000000000000000014
```

You can then parse that hex to 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 off-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 controls, meaning a non-owner could change the password

Description: The `PasswordStore::setPassword` function is set to be an `external` function,however, the natspec of the function and overall purpose of the smart contract is that `The 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, severely breaking the contract 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.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(msg.sender != s_owner){
        revert Password
    }
```

Informational

[I-1] The `PasswordStore::getPassword` natspec indicated 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 The `getPassword()` which the natspect say it should be The `getPassword(string)`.

Impact: The natspec is incorrect.

Recommended Mitigation: Remove the incorrect natspec line

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