Protocol Audit Report

Version 1.0

Cyfrin.io

September 2, 2025

Protocol Audit Report

Reidner

September 02, 2025

Prepared by: Reidner GitHub

Table of Contents

Protocol Summary

This protocol allows the owner of Smart Contract Storage his passwords.

Disclaimer

The Reidner 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 correspond the following commit hash::

```
be47a4de6d0ccb6621e075caee60409c3f7eac28
```

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.

Issues found

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

Findings

High

[H-1] Storing the password on-chain make 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 cakked by the owner of the contract.

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 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 get 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 the ouput

```
myPassword
```

Recommended Mitigation: The contract's architecture needs to be redesigned to ensure password privacy. Instead of storing the password on-chain, users should first **hash** their password off-chain using a secure cryptographic function. This hash can then be safely stored on-chain. To verify the password, the user would provide it again, the contract would hash it, and then compare the new hash with the stored hash. This one-way process ensures the original password is never exposed. The `getPassword` function is a critical vulnerability and should be removed entirely, as it violates the principle of password privacy by attempting to retrieve a value that should not be visible.

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

Description: the `PasswordSoter::setPassword` function is set to be an `external` function, however, the natspec of the funtion 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
        s_password = newPassword;
        emit SetNewPassword();
    }
```

Impact: The lack of access control allows anyone to set/change the password of the contract.

Proof of Concept: Add the following to the `Password.t.sol`

Code

```
    function test_non_owner_can_set_password(address randomAddress)
        public {
        vm.assume(owner != randomAddress);
        vm.prank(randomAddress);
        string memory expectedPassword = "myPassword";
        passwordStore.setPassword(expectedPassword);

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

Recommended Mitigation: Add an access control dondidional to the `setPassword` function

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

Informational

[I-1] The `PasswordStore::getPssword` natSpac indicates a parameter that doesn´t exist, causing 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;
    }
```

The `PasswordStore::getPssword` function signature is `getPassword()` while the natspex say it should be `getPassword(string)`.

Impact:

Recommended Mitigation: remove the incorrect natspec line

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