Protocol Audit Report

Version 1.0

equious.eth

August 31, 2025

Protocol Audit Report

ykgneh

August 31, 2025

Prepared by: ykgneh Lead Auditors:

- ykgneh

Table of Contents

Protocol Summary

PasswordStore is a protocol 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 access this password.

Disclaimer

The Yykgneh 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:

```
7d55682ddc4301a7b13ae9413095feffd9924566
```

Scope

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

Roles

- Owner: Is the only one who should be able to set and access the password.

For this contract, only the owner should be able to interact with the contract.

Executive Summary

Issues found

severity	number of findings
high	2
medium	0
low	0
info	1
total	3

Findings

High

[H-1] Storing the password on-chain is visible to anyone, and no longer private

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

we show such method of reading any data off chain below.

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

Proof of Concept: The below test case shows how anyone could read the password directly from the blockchain. We use foundry's cast tool to read directly from the storage of the contract, without being the owner.

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 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 control, anyone can set/change the pasword

Description: The `PasswordStore::setPassword` is intended to be `external` function callable by anyone. However the intended Natspec 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: no access control! anyone can store and change the
            password
        s_password = newPassword;
```

```
4           emit SetNetPassword();
5       }
```

Impact: Anyone can change or set the password, severly breaking the contract functionality

Proof of Concept: add this following to `PasswordStore.t.sol` file

code

```
1  function test_non_owner_can_set_password(address randomAddress) public
       {
2           //random address calling setPassword
3           vm.prank(randomAddress);
4           string memory notOwnerPassword = "notOwnerPassword";
5           passwordStore.setPassword(notOwnerPassword);
6
7           //owner reading the password
8           vm.prank(owner);
9           string memory actualPassword = passwordStore.getPassword();
10
11          //asserting that password was changed by non-owner
12          assertEq(notOwnerPassword, actualPassword);
13      }
```

Recommended Mitigation: Add an access control conditional to `PasswordStore::setPassword`.

```
1  if(msg.sender != s_owner){
2      revert PasswordStore__NotOwner();
3  }
```

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){}` The `PasswordStore::getPassword` function signature is `getPassword()` while the natspec says it should be `getPassword(string)`. **Impact:** The natspec is incorrect **Recommended Mitigation:** Remove the incorrect natspec line.

```
1       /*
2        * @notice This allows only the owner to retrieve the password.
3  -     * @param newPassword The new password to set.
```

```
4         */
```