Protocol Audit Report

Version 1.0

zuruOnyx

September 26, 2025

Protocol Audit Report

Sunday, Justice

25th Sept, 2025

Prepared by: Justice Lead Auditors:

- xxxxxxx

Table of Contents

Protocol Summary

PasswordStore is a protocol dedicated to storage and retieval of a user's passwords. The protocol is designed to be used by a single user, and is not designed to be used by mutiple users. Only the owner should be able to set and access this password.

Disclaimer

The ZuruOnyx 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
Likelihood	High	H	H/M	M
	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 has:

```
2e8f81e263b3a9d18fab4fb5c46805ffc10a9990
```

Scope

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

Roles

- Owner: The user who can set the password and read the password.
- Outsides: No one else should be able to set or read the password.

Executive Summary

Issues found

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

Findings

High

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

Description: All data stored on-chain is visible to anyone, and can be read directly from the blockchain. The `PasswordStore::s_Passowrd` 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 what 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 Concepts: (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 of `s_password` in the contract.

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

   you'll get an output that looks like this: `0x6d7950617373776f72640000000000000000000000000000`

   you can parse that hex to a string with

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

   and get an output of:

   ```
   myPassword
   ```

   Note You can check for storage slot by

   ```
   forge inspect <Contract_Name> storage-layout
   ```

   it will display a table showing you the name, type and slot position.

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 woudn't want the user to accidentally send traction with the password that decrypts your password.

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

Informational

[I-1] `PasswordStore::getPassword` natspec indicates a parameter that doesn't exist', causing the natspec to be incorrect.

Gas