Protocol Audit Report

Version 1.0

Ranir

April 24, 2025

Protocol Audit Report

Ranir

April 23,2025

Prepared by: Ranir Lead Auditor: - Ranir

Table of Contents

Protocol Summary

PasswordStore is a protocol specifically designed for the secure storage and retrieval of a user's password. It is intended for single-user usage, ensuring that only the designated owner of the contract has the authority to set or retrieve the stored password. The design assumes a trusted owner context, and any interaction with the protocol should enforce strict access control mechanisms to prevent unauthorized access or leakage of sensitive data.

Disclaimer

We 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 describe in this document correspond in the following commit hash

```
2e8f81e263b3a9d18fab4fb5c46805ffc10a9990
```

Scope

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

Roles

- Owner:The user who can set the password and read it.
- Outsides:No one else should be able to do what the owner does. ## Issues found

Severity	Number of issues found
High	2
Medium	0
Low	0
Infos	1
Totals	3

Findings

High

[H-1] Exposure of sensive data - The varible `PasswordStore::s_password` is visible.

Description: The intended design is for `PassordStore::s_password` to be accessible exclusively by `PasswordStore::s_owner`.However, due to the transparent nature of blockchain storage, any data stored on-chain is publicly readable, regardless of visibility modifiers or access control mechanisms defined in the contract.The `PassordStore::s_password` variable is meant to be retrieved solely via the `PassordStore::getPassword` function which enforces access restrictions such that only the contract owner can invoke it. Despite this, the raw storage slot where`PasswordStore::s_password` resides can still be directly queried by anyone with access to a full node or RPC endpoint.

In the Proof of Concept section, we demonstrate a low-level approach for extracting such on-chain secrets by reading the storage layout directly, bypassing any Solidity-level access control.

Impact: Everybody can read `PasswordStore::s_password` seriously undermining the protocol.

Proof of Concept: The following section demonstrates how an attacker can access a user's password, which is supposed to remain private. We used the Foundry framework and deployed the contract on a local blockchain to read the contract's storage section without being the owner.

1. Create a local chain

```
make anvil
```

2. Deploy the contract on a local chain

```
make deploy
```

3. Engage the Foundry tool

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

4. Copy the Hex data at the bottom of your command 0x6d79506[...].
5. You can next analyze that hex data with the command below. cast parse-bytes32-string 0x6d79506[...].
6. After the execution of teh last command you will get the password [...].

Recommended Mitigation: As a result, the overall architecture of the contract should be reconsidered. One option would be to encrypt the password off-chain and then store the encrypted password on-chain. Alternatively, you could store a hash instead of the actual value. In this setup, the user would need to remember a separate off-chain password to decrypt the stored password. However, it would also be advisable to remove the view function, as you wouldn't want the user to accidentally send a transaction containing the decryption key.

[H-2] Everybody can call the function `PasswordStore::setPassword` and setPassword

Description: Given that the `PasswordStore::setPassword` function has external visibility, any externally owned account or contract can invoke it. This design choice implies that anyone—not just the intended owner—can modify the password stored by your protocol. Since the protocol lacks any form of access control on this critical function, it effectively allows unauthorized entities to overwrite sensitive data.This directly contradicts the intended security goal of the protocol, which presumably aims to restrict password updates to a trusted party `PasswordStore::s_owner` . The absence of proper authorization checks renders the protocol vulnerable to arbitrary state manipulation.

```
function setPassword(string memory newPassword) external {
        /* There is no access control */
        s_password = newPassword;
        emit SetNetPassword();
    }
```

impact: Anyone can change the implemented password, severely undermining the intended function of your protocol.

Proof of Concept: Add the test bellow to your PasswordStore.t.sol file .

Code

```
    function test_every_user_can_setpassword(address ramdomAddress)
        public{
          vm.assume(ramdomAddress !=owner);
          vm.startPrank(ramdomAddress);
          string memory expectedPassword="votremotdepasse";
          passwordStore.setPassword(expectedPassword);

          vm.startPrank(owner);
          string memory owneractualPasswordStore= passwordStore.
              getPassword();

          assertEq(expectedPassword,owneractualPasswordStore);
    }
```

Recommended Mitigation: Add a conditionnal access control to your `PasswordStore::setPassword`

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

Informational

[I-1] The function `Password::getPassword` requires a parameter in its natspec that does not exist in the function

Description: the new password parameter is missing.

code

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

```
5         function getPassword() external view returns (string memory){}
6
7  the function signature `Password::getPassword` is `getPassword()` while
       its natspec indicates that it should be `getPassword(string)`.
```

Impact: The napsec of the function `Password::getPassword` is incorrect.

Recommended Mitigation: Delete the non-existent documented setting

```
1   /*
2  -     * @param newPassword The new password to set.
3
4       */
```