Protocol Audit Report

Version 1.0

Tanu Gupta

July 16, 2025

Protocol Audit Report

Tanu Gupta

June 27, 2025

Prepared by: Tanu Gupta

Lead Security Researcher:

- Tanu Gupta

Table of Contents

Protocol Summary

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

Disclaimer

The Tanu 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: The user who can set the password and read the password.

Outsiders: No one else should be able to set or read the password.

Executive Summary

Found the bugs using a tool called foundry.

Issues found

Severity	Number of issues found
High	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_password` is intended to be a private variable and only accessed through `PasswordStore::getPassword` function, which is intended to be called by the owner of the contract.

We show one such method of reding any data off-chain below.

Impact: Anyone can read the private password, severely 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 `PasswordStore::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 the hex to a string with

```
cast parse-bytes32-string 0
    x6d7950617373776f726400000000000000000000000000000000000000000014
```

    And get an output of: `myPassword`

4. **Recommended Mitigation:** Due to this, the overall architecture of the contract should be rethought. One could encrypt the password offchain and then store the encrypted password on-chain. This would require user to remember the key offchain used for encryption. However, you'd likely also want to remove the view function as you wouldn't want the user to accidently send a transaction with the key that decrypts the password.

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

Description: The `PasswordStore::setPassword` is set to be external function, however, the natspec of the function and overall purpose of the smart contract is `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 SetNetPassword();
    }
```

Impact: Anyone can set/change the password of the contract, severely breaking the intended functionality.

Proof of Concept: Add following to the `test/PasswordStore.t.sol` test file

Code

```
function test_any_one_can_set_password(address randomAddress) public {
        vm.assume(randomAddress != owner);
        vm.prank(randomAddress);
        string memory newPassword = "hey there";
        passwordStore.setPassword(newPassword);

        vm.prank(owner);
        string memory actualPassword = passwordStore.getPassword();
        assert(keccak256(abi.encodePacked(actualPassword)) == keccak256
            (abi.encodePacked(newPassword)));
    }
```

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

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

Informational

[I-1] The `PasswordStore::getPassword` natspec indicates a parameter that does not exist, causing the netspec 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();
        }
```

```
9           return s_password;
10      }
```

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 netspec line:

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