Protocol Audit Report

Version 1.0

Gokalp

October 24, 2025

Protocol Audit Report

Gokalp

October 24, 2025

Prepared by: Gokalp

Table of Contents

Protocol Summary

PasswordStore is a smart contract application for storing a password. Users should be able to store a password and then retrieve it later. Others should not be able to access the password.

Disclaimer

The Gokalp T. 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

commit hash:

```
2e8f81e263b3a9d18fab4fb5c46805ffc10a9990
```

Scope

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

Roles

- Owner: Only the owner may set and retrieve their password.
- Outsides: No one else should be able to set or read the password.

Executive Summary

Used Forge and Anvil as tools. Wrote missing test cases.

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, no longer private.

Description: All data stored on-chain is visible to anyone, and can be directly read from the blockchain. The `PasswordStore::s_password` variable is intended to be a private variable and only accessed via the `PasswordStore::getPassword` function, which is intended to be only called by the owner.

Impact: Anyone can read the private password, severly breaking the functionality of the protocol.

Proof of Concept: 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

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

We gave `1`, because `PasswordStore::s_password` is the storage slot on the contract.

you will get output like this; `0x6d7950617373776f7264000000000000000000000000000000000000000`

You can parse that hex to a string wtih:

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

and you will get the output: `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 you password.

[H-2] `PasswordStore::setPassword` has no access control, anyone can change owners password.

Description: The `PasswordStore::setPassword` is set to be an external password. But this function should allow only the owner to set a new password.

Impact: Anyone can change the password of the owner, severly breaking the contract intended functionality.

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

Code

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

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

Recommended Mitigation: Add an access control conditional to the `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 natspec to be incorrect.

Description: The `PasswordStore::getPassword` function signature is `getPassword()` while the natspec says it should be `getPassword(string)`.

```
    /*
     * @notice This allows only the owner to retrieve the password.
>>   * @param newPassword The new password to set.  <<
     */
    function getPassword() external view returns (string memory) {
```

Impact: NatSpec is incorrect.

Recommended Mitigation: Remove the incorrect natspec line.

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