Protocol Audit Report

Version 1.0

khal45

September 1, 2025

Protocol Audit Report

khal45

September 1st, 2025

Prepared by: khal45 Lead Security Researcher:

- khal45

Table of Contents

Protocol Summary

Passwordstore is a protocol that allows users to store their passwords and retrieve it later. The protocol is designed in such a way that only the owner should be able to set and access the password

Disclaimer

Khal45 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 me 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:

The findings described in this document correspond the following commit hash:

```
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

Over the course of the security review, khal45 engaged with the **passwordstore** protocol to review it. In this period of time, a total of 3 issues were found

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 publicly visible and can be read directly from the blockchain. The `PasswordStore::s_password` variable is intended to be private and only accessible through

the `PasswordStore::getPassword` function, which is meant to be called exclusively by the contract owner.

However, the password can be read directly from storage. We demonstrate one method of retrieving it below.

Impact:
Anyone can read the private password, severely compromising the intended functionality of the protocol.

Proof of Concept:

The following test case shows how anyone can read the password directly from the blockchain:

1. Start a local Anvil chain:

```
make anvil
```

2. Deploy the `PasswordStore` contract:

```
make deploy
```

3. Retrieve the storage slot of `s_password`:

```
cast storage <contract_address> 1 --rpc-url <rpc_url>
```

We use 1 because `s_password` is stored at slot 1 in the contract.

Example output:

```
0x6d7950617373776f726400000000000000000000000000000000000000000014
```

4. Decode the stored value into a string:

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

Output:

```
myPassword
```

Recommended Mitigation:
The architecture of the contract should be rethought. One option is to encrypt the password off-chain and only store the encrypted value on-chain. This would require users to keep an off-chain decryption key. Additionally, consider removing the view function to prevent users from accidentally exposing the decryption key on-chain.

[H-2] `PasswordStore::setPassword` lacks access control, allowing anyone to change the password

Description:
The `PasswordStore::setPassword` function is declared `external`. According to its NatSpec and the intended purpose of the contract, only the owner should be able to set a new password. However, there are no access control checks in place.

```
function setPassword(string memory newPassword) external {
    // @audit - No access control
    s_password = newPassword;
    emit SetNewPassword();
}
```

Impact:
Any account can set or change the contract's password, completely breaking the intended functionality.

Proof of Concept:
Add the following test to `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 check to ensure only the owner can call `setPassword`:

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

Informational

[I-1] Incorrect NatSpec for `PasswordStore::getPassword`

Description:
The NatSpec for `PasswordStore::getPassword` incorrectly includes a parameter that does not exist:

```
/*
 * @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:
The NatSpec documentation is incorrect and misleading.

Recommended Mitigation:
Remove the invalid `@param` line:

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