Protocol Audit Report

Version 1.0

Jack

July 25, 2025

Protocol Audit Report

Jack

July 22, 2025

Prepared by: Jack Lead Security Researcher: - Jack

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 Jack 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 corespond to the following commit:

```
7d55682ddc4301a7b13ae9413095feffd9924566
```

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

Add some notes about how the audit went, types of things you found, etc. We spent X hours with Z auditors using Y tools. etc

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

Description
All data stored on-chain is publicly readable. The `PasswordStore::s_password` variable—though only exposed via the `getPassword` view—is still stored in a public slot. An attacker can query it directly:

Impact
Anyone can read the “private” password, completely breaking confidentiality.

Proof of Concept

1. Spin up a local Anvil node:

```
bash make anvil
```

2. Deploy the contract:

```
make deploy
```

3. Read slot 1 (where `s_password` lives):

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

You'll get something like:

```
0x6d7950617373776f726400000000000000000000000000000000000000000014
```

4. Decode it to a string:

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

Output:

```
myPassword
```

Recommended Mitigation

1. Re-architect so that secrets are never stored in plaintext on-chain.
2. For example, store only an off-chain-encrypted blob; require the user to provide a decryption key in a transaction (or better yet, keep decryption fully off-chain).
3. Remove or restrict the `getPassword` view to avoid accidental on-chain exposure.

[H-2] `setPassword` has no access control

Description

The `setPassword` function is `external` without any `onlyOwner` check:

```
// Incorrect: anyone can call
function setPassword(string memory newPassword) external {
    // @audit: missing access control
    s_password = newPassword;
    emit SetNewPassword();
}
```

Impact

Anyone can overwrite the stored password, breaking protocol integrity.

Proof of Concept

Solidity test snippet

```
function test_anyone_can_set_password(address randomAddr) public {
    vm.assume(owner != randomAddr);
    vm.prank(randomAddr);
    passwordStore.setPassword("hackedPassword");

    vm.prank(owner);
    string memory actual = passwordStore.getPassword();
    assertEq(actual, "hackedPassword");
}
```

Recommended Mitigation

Add an owner-only modifier:

```
function setPassword(string memory newPassword) external onlyOwner {
    s_password = newPassword;
    emit SetNewPassword();
}
```

Informational

[I-1] Incorrect natspec `@param` in `getPassword`

Description

The `getPassword` function takes no parameters, yet the natspec includes an irrelevant `@param` tag:

```
/**
 * @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();
    return s_password;
}
```

Impact

Documentation generators and auditors will be misled into thinking `getPassword` requires an argument.

Recommended Mitigation

1. Remove the incorrect `@param` line.
2. Add a proper `@return` tag.

```
/**
 * @notice This allows only the owner to retrieve the password.
- * @param newPassword The new password to set.
+ * @return password The stored password string.
 */
function getPassword() external view returns (string memory) {
    if (msg.sender != s_owner) revert PasswordStore__NotOwner();
    return s_password;
}
```

Corrected Version

```
/**
 * @notice This allows only the owner to retrieve the password.
 * @return password The stored password string.
 */
function getPassword() external view onlyOwner returns (string memory)
    {
    return s_password;
}
```

(Optionally combine with an `onlyOwner` modifier for extra safety.)