PasswordStore Initial Audit Report

Version 1.0

Auditryx

September 21, 2025

PasswordStore Initial Audit Report

0xrafi-kaji

September 20, 2025

Prepared by: 0xrafi-kaji

Lead Auditors: - 0xrafi-kaji

Assisting Auditors: - None

Table of Contents

Protocol Summary

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 Auditryx 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.
- Outsides: No one else should be able to set or read the password.

Executive Summary

The PasswordStore smart contract was reviewed to assess its design, implementation, and overall security posture. The audit was conducted following a structured methodology consisting of scoping, reconnaissance, vulnerability identification, and reporting.

The review highlighted that the security readiness of a protocol depends heavily on documentation quality, onboarding details, and code maturity. Through iterative versions (V1–V3), the audit demonstrated how incomplete documentation, limited testing, and insufficient client onboarding can significantly hinder the effectiveness of a review.

Key observations included potential risks in storage patterns, lack of clear specifications, and weak design decisions.

Summary of Findings

- Clear documentation and onboarding are essential for effective audits.
- Tooling support (cloc, Solidity Metrics) helps scope complexity and focus review efforts.
- Code maturity and testing are prerequisites for audit readiness.
- Even simple contracts may introduce security risks if design patterns are not carefully considered.

Issues found

Severity	Number of issues found
High	2
Medium	0
Low	0
Informational	1
Total	3

Findings

High

[H-01] Missing Access Control in `PasswordStore::setPassword`(Anyone can change stored password)

Description: The function setPassword(string memory newPassword) has no access control. Anyone can overwrite the stored password. NatSpec incorrectly claims “only the owner” can call it, but the check is missing.

```
    function setPassword(string memory newPassword) external {
@>      // @audir - There are no access controls here, anyone can call this function.
        s_password = newPassword;
        emit SetNetPassword();
    }
```

Impact: Any random address can change the password by calling `PasswordStore::setPassword`, breaking contract `integrity` and rendering the `system unusable`.

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

Click to see the code

```
    function test_anyone_can_set_password(address randomAddress) public {
        vm.assume(randomAddress != owner);

        // Simulate an attacker changing the password
        vm.prank(randomAddress);
        string memory newPassword = "hackedPassword";
        passwordStore.setPassword(newPassword);

        vm.prank(owner);
        string memory acctualPassword = passwordStore.getPassword();

        // Verify that the password has been changed
        assertEq(acctualPassword, newPassword);
        }
```

Recommended Mitigation: Add explicit ownership check in `PasswordStore::setPassword` function.

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

[H-02] Sensitive Data Exposed On-Chain (Storing plaintext password makes it publicly readable)

Description: The contract stores `PasswordStore::s_password` directly in storage. Ethereum storage is public, it's doesn't matter storage variable is private or not! So anyone can read the password directly without calling `PasswordStore::getPassword()`.

Impact: Exposes confidential information. Anyone can recover the plaintext password using on-chain data. Severly brakes 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

```
1    make anvil
```

2. Deploy the contract

```
1    make deploy
```

-> get contract address from the deploy output

3. Run the storage tool

We use 1 because that's the storage slot of `s_password` variable. (State variables are stored sequentially starting from slot 0)

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

-> returns <0x6d7950617373776f726400000000000000000000000000000000000000000014>

We can decode this hex value to a string:

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

-> returns `myPassword` (Our original password!)

Recommended Mitigation:

- Do not store sensitive plaintext data on-chain.
- Store hashed/encrypted values only.
- Consider client-side encryption and on-chain ciphertext storage.

Informational

[I-01] Incorrect NatSpec Documentation in `getPassword` (Misleading documentation)

Description: The NatSpec for `PasswordStore::getPassword()` includes a `@param` tag for `newPassword`, but the function takes no parameters.

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

Impact: Causes confusion for developers and auditors. Documentation mismatch makes audits harder and may mislead integrators.

Recommended Mitigation: Remove the incorrect natspec line.

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