Protocol Audit Report

Version 1.0

https://github.com/Aditya-alchemist

January 1, 2026

Protocol Audit Report

Aditya kumar Mishra

Jan 1, 2026

Prepared by: Alchemist

Lead Auditors: - Aditya kumar Mishra

Table of Contents

Protocol Summary

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

Disclaimer

The Alchemist 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:

2e8f81e263b3a9d18fab4fb5c46805ffc10a9990

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

We spent 1 hours with 1 auditors using foundry.

Issues found

Servertity	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 publicly readable and no longer private

Description

All data stored on-chain is publicly accessible and can be read directly from the blockchain. The variable `PasswordStore::s_password` is intended to be private and only accessible through the `PasswordStore::getPassword` function, which itself is intended to be callable only by the contract owner.

However, despite this intention, any user can read the password directly from contract storage without interacting with the contract logic. One such off-chain method is demonstrated below.

Impact

Any user can read the supposedly private password, completely breaking the confidentiality guarantees of the protocol and rendering the password mechanism ineffective.

Proof of Concept

The following steps demonstrate how private on-chain data can be read directly:

1. Start a local blockchain:

```
make anvil
```

2. Deploy the contract:

```
make deploy
```

3. Read the storage slot containing the password:

```
cast storage 0x5FbDB2315678afecb367f032d93F642f64180aa3 1 --rpc-url
    http://127.0.0.1:8545
```

Example output:

```
0x6d7950617373776f726400000000000000000000000000000000000000000014
```

4. Decode the `bytes32` value into a string:

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

5. Output:

```
myPassword
```

This demonstrates that the password can be retrieved directly from on-chain storage without any authorization.

Recommended Mitigation

The contract architecture should be redesigned to avoid storing sensitive information in plaintext on-chain. One possible approach is to encrypt the password off-chain and store only the encrypted value on-chain. The user would then be responsible for securely managing the decryption key off-chain.

Additionally, the `getPassword` view function should likely be removed, as returning the decrypted password on-chain defeats the purpose of encryption and risks accidental disclosure via transactions.

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

Description

The `PasswordStore::setPassword` function is declared as `external`, and according to its natspec documentation, it is intended to allow **only the owner** to set a new password. However, no access control is implemented.

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

Impact

Any user can call `setPassword` and overwrite the stored password, completely bypassing the intended access control and compromising the protocol.

Proof of Concept

```
function test_Anyone_can_set_password(address random) public {
    vm.assume(random != owner);
    vm.prank(random);

    string memory newPassword = "hackedPassword";
    passwordStore.setPassword(newPassword);

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

    assertEq(actualPassword, newPassword);
}
```

This test demonstrates that a non-owner can successfully overwrite the password.

Recommended Mitigation

Restrict access to `setPassword` by adding an `onlyOwner` modifier (or equivalent access control mechanism).

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

Informational

[I-1] Incorrect natspec documentation in `PasswordStore::getPassword`

Description

The natspec documentation for `PasswordStore::getPassword` references a parameter that does not exist in the function signature.

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

The function does not accept any parameters, making the natspec misleading.

Impact

Incorrect documentation can confuse developers, auditors, and integrators, potentially leading to incorrect assumptions or usage of the contract.

Recommended Mitigation

Remove the invalid natspec parameter annotation.

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