# PasswordStore Audit Report

Prepared by: Jason

# Table of Contents



# Executive Summary

Twilly conducted a security review for `PasswordStore` from March 6th 2025 to March 7th 2025. During the engagement, Twilly reviewed the source code for security vulnerabilities, design issues and general flaws in security posture. As a result, 2 high severity findings and one informational finding were discovered.

# Protocol Summary

The `PasswordStore` protocol 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.

# Audit Details

## Roles

Lead Security Researcher: Jason

## Scope of Assessment

- **Repository:** https://github.com/Cyfrin/3-passwordstore-audit
- **Version/Commit Hash:** 7d55682ddc4301a7b13ae9413095feffd9924566
- **Programming Language(s):** Solidity
- **Platform:** EVM

## 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.

## Issues found

| **Severity** | **Count** |
|---|---|
| High | 2 |
| Medium | 0 |
| Low | 0 |
| Informational | 1 |

# Detailed Findings

## [H-1] Storing the password on-chain makes it visible to everyone.

### Description

The `PasswordStore::s_password` variable is intended to only be retrieved from the `PasswordStore::getPassword` function. However, it is stored on-chain and can be read by anyone. The solidity keyword `private` does not properly hide the storage variable from outside users.

### Impact

Anyone can read the password, which severly breaks the confidentiality 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
```

1. Deploy the contract

```
make deploy
```

1. Read the storage slot data

The `s_password` variable is located at storage slot 1.

```
cast storage <CONTRACT_ADDRESS> 1
```

You will get an output similar to this:

`0x6d7950617373776f726400000000000000000000000000000000000000000014`

1. Parse the password

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

### Recommended Mitigation

Recommend encrypting the password and storing it off-chain so it can not be read by every user.

## [H-2] The setPassword function lacks access control, meaning anyone can change the password

### Description

The `PasswordStore::setPassword` function does not have any access control protections in place. This allows any user to be able to change the password and break the security of the project.

```
    function setPassword(string memory newPassword) external {
        #>      // @audit - there are no access control checks
        s_password = newPassword;
        emit SetNetPassword();
    }
```

### Impact

Any user can change the contract password, which can lead to breaking the security of the entire project.

### Proof of Concept

We have created the below test cast to confirm that any arbitrary user can call the `setPassword` function and change the password. Add this to the `PasswordStore.t.sol` file.

▶ Details

```
    function test_non_owner_set_password() public {
        vm.startPrank(address(69));
        vm.expectRevert(PasswordStore.PasswordStore__NotOwner.selector);

        passwordStore.setPassword("thisShouldFail");
    }
```

### Recommended Mitigation

Add an access modifier to only allow the owner to call the function.

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

## [I-1] Incorrect NatSpec Details for the `getPassword` Function

### Description

The NatSpec comment for the `PasswordStore::getPassword` function contains an error. It incorrectly states: `@param newPassword The new password to set.`

However, the `getPassword` function does not take any parameters. This discrepancy may mislead developers or auditors when reviewing the contract.

### Impact

The incorrect NatSpec comment can lead to confusion during code reviews, potentially affecting the accuracy of documentation and audits.

### Recommended Mitigation

Remove the incorrect line from the NatSpec comment.

```
    /*
     * @notice This allows only the owner to retrieve the password.
```

```
-     * @param newPassword The new password to set.
      */
     function getPassword() external view returns (string memory) {
```

# Conclusion

This security assessment provided a detailed analysis of PasswordStore's current security posture. We identified 3 issues of varying severity and provided actionable recommendations to address these concerns. By addressing the findings outlined in this report and implementing the recommended best practices, PasswordStore can significantly strengthen the security and reliability of their platform.

We appreciate the opportunity to work with PasswordStore and remain available to provide additional guidance as needed.

## Disclaimer

This assessment does not guarantee the identification of all vulnerabilities in the reviewed codebase, nor does it assure the absence of issues in future code changes. We recommend supplementing this assessment with additional reviews and continuous security practices.

All recommendations provided in this report are intended as starting points for addressing identified issues. Code samples are for illustrative purposes and may not be fully functional or tested.

The findings and recommendations contained in this report are provided for informational purposes only and should not be construed as legal, tax, investment, or financial advice. This report does not constitute an endorsement or solicitation for PasswordStore's project.