Protocol Audit Report

Version 1.0

DVYN Security

November 24, 2025

Protocol Audit Report

Koz Mik

November 23, 2025

Prepared by: [DVYN SECURITY] Lead Security Researcher: - K05m1k

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 DVYN SECURITY 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

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 two hours with one auditor using security tools, we found two high and one informational bug. We included recommendations for mitigations of each and provided proof of concept so that you can recreate.

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 (Root Cause + Impact)

Description: All data stored on-chain is visible to anyone, and can be read directly from the blockchain. The `PasswordStore::s_password` variable is intended to be a private variable and only accessed through the `PasswordStore::getPassword` function, which is intended to be only called by the owner of the contract.

We show one such method of reading any data off chain below.

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

Proof of Concept: (Proof of Code)

The below test case shows how anyone can read the password directly from the blockchain.

1. Create a locally running chain

```
make anvil
```

2. Deploy the contract to the chain

```
make deploy
```

3. Run the storage tool

We use `1` because that's the storage slot of `s_password` in the contract

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

You'll get an output that looks like this: `0x6d7950617373776f726400000000000000000000000000000000`

You can then parse that hex to a string with:

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

And get an output of:

```
myPassword
```

Recommended Mitigation: Due to this, the overall architecture of the contract should be rethought. One could encrypt the password off-chain, and then store the encrypted password on-chain. This would require the user to remember another password off-chain to decrypt the password. However, you

[H-2] `PasswordStore::setPassword` has no access controls, meaning a non-owner can change the password (Root Cause + Impact)

Description: The `PasswordStore::setPassword` function is set to be an `external` function, however, the natspec of the function and overall purpose of the smart contract is that

```
This function allows only the owner to set a new password.
```

```
    function setPassword(string memory newPassword) external {
        // @audit - There are no access controls
        s_password = newPassword;
        emit SetNetPassword();
    }
```

Impact: Anyone can set/chainge the password of the contract, severely breaking the contract's functionality

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

Code

```
     function test_non_owner_can_set_password(address randomAddress) public{
        vm.assume(owner!=randomAddress);
        vm.prank(randomAddress);
        string memory expectedPassword = "myPassword";
        passwordStore.setPassword(expectedPassword);

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

    }
```

Recommended Mitigation: Add an access control conditional to the setPassword function.

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

Informational

[I-1] `PasswordStore::getPassword` natspec indicates a parameter that doesn't exist, causing the natspec to be incorrect (Root Cause + Impact)

Description:

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

The `PasswordStore::getPassword` function signature is `getPassword()` which the natspec say it should be `getPassword(string)`.

Impact: The natspec is incorrect.

Recommended Mitigation: Remove the incorrect natspec line.

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