PasswordStore Audit Report

Version 1.0

r00tSid.github.io

September 7, 2025

PasswordStore Audit Report

r00tSid

September 6, 2025

Prepared by: r00tSid Lead Auditors: - r00tSid

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 r00tSid 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
Likelihood	High	H	H/M	M
	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

- Github Link: https://github.com/Cyfrin/3-passwordstore-audit/tree/onboarded
- Commit Hash: `7d55682ddc4301a7b13ae9413095feffd992456`
- Solc Version: 0.8.18
- Chain(s) to deploy contract to: Ethereum

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

There were 20 lines of code to audit in `PasswordStore` and we spent more than 2 hours and found 2 high vulnerabilities and 1 informative vulnerability.

Issues found

Severity	Number Of issues found
High	2
Medium	0
Low	0
Info	1
Total	3

Findings

High

[H-1] Storing the password onchain makes it visible to anyone, no longer to private

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 variableand only accessed through the `PasswordStore::getpassword` function, which is intended to be only called by the owner of the contract.

Impact: Anyone cn read the password, severly breaking the functionallity of the protocol.

Proof of Concept:

1. Create a locally running chain

```
make anvil
```

2. Deploy the contract to chain

```
make deploy
```

3. Run storage tool

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

You will get an ouptut like this:

```
0x6d7950617373776f726400000000000000000000000000000000000000000014
```

4. Parse that hex to string

```
cast parse-bytes32-string 0x6d7950617373776f72640000000000000
0000000000000000000000000000014
```

You will get an otput of password:

```
myPassword
```

Recommended Mitigation: 1. Avoid Storing Secrets On-Chain

```
- Redesign the contract to not require passwords or sensitive data on-
chain.
```

```
- Instead, use off-chain storage (e.g., secure database, secret manager, or external
```

2. Use Cryptographic Hashing (if unavoidable)
    - If a password-like mechanism is absolutely required for validation, store only a hashed value (e.g., using keccak256(password)), not the raw password.
    - During verification, compare the hash of the user-provided input with the stored hash. This prevents direct recovery of the plaintext password from blockchain storage.
3. Consider Alternative Authentication Models
    - Instead of a password-based model, rely on Ethereum's native authentication via msg.sender (i.e., account ownership with private keys).
    - Implement role-based access control with onlyOwner or AccessControl from OpenZeppelin

[H-2] `PasswordStore::setPassword` function has no access controls, meaning a non-owner could change the password.

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 only allows only the owner to set a new password.` But the function is not checking if the password was changed by the owner or an non-owner.

```
    function setPassword(string memory newPassword) external {
        // missing checks of an owner, no means access control
        s_password = newPassword;
        emit SetNewPassword();
    }
```

Impact: Anyone can set/change password of the contract, severly breaking the contract intended functionality.

Proof of Concept: Add the code to test this:

```
function test_anyone_can_set_password(address randomAddress) public{
        vm.prank(randomAddress != owner);
        string memory expectedpassword= 'hackedPassword';
        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] The `PasswordStore::getPassword` natspec indicates a parameter that doesn't exist, causing the natspec to be incorrect.

Description:

```
 /*
     * @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;
    }
```

The `PasswordStore::getPassword` funtion 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.
```