PasswordStore Audit Report

Version 1.0

https://github.com/Heavens01

June 15, 2025

PasswordStore Audit Report

Heavens01

June 15, 2025

Prepared by: Heavens01 Lead Security Researcher(s): - Heavens01

Table of Contents

Protocol Summary

This Protocol allows you to store a private password that others won't be able to see. You can update your password at any time but only if you are the owner of the contract/protocol. This protocol is not designed to be used by multiple users at once rather it is One CONTRACT for One USER.

Disclaimer

The HEAVENS01 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

Commit Hash:

```
2e8f81e263b3a9d18fab4fb5c46805ffc10a9990
```

Scope

```
./src/
#-- PasswordStore.sol
```

Roles

- Owner: The user who can store the password and retrieve/read it alone
- Outsiders: Other persons (excluding the owner) who are not allowed to retrieve or set password.

Executive Summary

*We spent 0.5 hours (30 minutes) with one auditor using tools like solidity metrics, fuzz testing and manual reading of codebase.

Issues found

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

Findings

High

[H-1] Password onchain is visible and accessible to anyone through storage slots regardless of visibility specifiers. So therefore, the password will be public knowledge

Description: All data stored on-chain is visible to anyone and can be read directly from the blockchain through storage slots. The `PasswordStore::s_password` variable, although it's visibility set to private, it actually has a storage slot which anybody can access through the blockchain which defies

the main purpose of this protocol (the owner having sole access to set and retrieve password at will) and only through `PasswordStore::getPassword` function.

We show such method of reading such data off chain below

Impact: Anyone can read the private password, automatically defiling the security of the protocol being able to safely guide the password from unauthorized persons.

Proof of Concept: (Proof Of Code)

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

1. Spin a local chain

```
anvil
```

2. Deploy contract with script and call `PasswordStore::setPassword` function with desired password. (In this case: `"SuperSecretPassword"`)

```
make deploy
```

Then call setPassword with desired password.

3. Run the storage tool in foundry

```
cast storage 0xe7f1725E7734CE288F8367e1Bb143E90bb3F0512 1 --rpc-url
    http://localhost:8545
```

You'll get an output that looks like this: `0x537570657253656372657450617373776f726400000000000000`

Then run this code and substitute your output for mine

```
cast parse-bytes32-string 0
    x537570657253656372657450617373776f726400000000000000000000000026
```

You'll inevitably get the password you used (and for this case): `SuperSecretPassword`

Recommended Mitigation: Due to this, the overall architecture of this 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'd also likely want to remove the view function as you wouldn't want the user to accidentally send a transaction with the password that decrypts your password.

[H-2] `PasswordStore:setPassword` has no access controls therefore anyone can change password

Description: The function `PasswordStore::setPassword` intends to allow only the owner to set password but no access control was specified in/for it. This now makes password to be overridable by just anybody.

This is it below:

```
1      function setPassword(string memory newPassword) external {
2  @>      // No access control modifier / access control logic
3          s_password = newPassword;
4          emit SetNetPassword();
5      }
```

Impact: Anyone can change or set the password which is not the intended functionality.

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

Code

```
1      function test_anyone_can_set_password(address randomCaller) public
           {
2          vm.assume(randomCaller != owner);
3          vm.prank(randomCaller);
4          string memory expectedPassword = "randomUserPassword";
5          passwordStore.setPassword(expectedPassword);
6
7          vm.prank(owner);
8          string memory actualPassword = passwordStore.getPassword();
9          assertEq(actualPassword, expectedPassword);
10     }
```

Now run test with:

```
1  forge test --mt test_anyone_can_set_password -vvvv
```

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

```
1  if (msg.sender != s_owner) {
2      revert PasswordStore__NotOwner();
3  }
```

Informational

[I-1] The `PaswordStore::getPassword` natspec indicates there is a parameter but the function has no parameter causing the natspec to be wrong/incorrect

Description:

```
1      /*
2       * @notice This allows only the owner to retrieve the password.
3   @>  * @param newPassword The new password to set.
4       */
5      function getPassword() external view returns (string memory) {}
```

The function signature is `getPassword()` but the natspec is saying it should be `getPassword(string)`.

Impact: The natspec is incorrect

Proof of Concept: Proofs Itself

Recommended Mitigation: Remove the incorrect natspec line.

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