AUDIT SECURE

Trusted Audit Solutions

Protocol Audit Report

Version 1.0

auditSecure

January 12, 2026

Protocol Audit Report

audit_secure

january 12, 2026

Prepared by: auditSecure

Lead Security Reseacher: - ma_buba

Table of Contents

    - [I-#] The `PasswordStore::getPassword` NatSpec indicate a non-existent parameter, causing the natspec to be incorrect.
  - Gas

Protocol Summary

PasswordStore is a protocal dedicated to storage and retreival of the user's password. 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 audit_secure 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

The findings described in this document correspond to the following commit Hash:

```
7d55682ddc4301a7b13ae9413095feffd9924566
```

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

I spent X hours with Y auditors using Z tools.

Issues found

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

Findings

High

[H-#] storing the password on-chain makes it visible to anyone, and no longer 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 variable and only access 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 password, severily 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 anvil

```
make anvil
```

2. deploy the contract to the chain

```
make deploy
```

3. Run the storage tool

we use 1 because that the storage slot for `s_password` in the contract

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

you'll get an output that look 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'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 decrypt your password.

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

Description: The `PasswordStore::setPassword` 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 the owner to set a new password.`

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

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

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

code

```
    function testAnyoneCanSetPassword(address randomAddress) public {
        //means ignore cases where "randomAddress is equals owner"
        vm.assume(randomAddress != owner);
        vm.prank(randomAddress);
        string memory expectedPassword = "myNewPassword";
        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();
}
```

Medium

Low

Informational

[I-#] The `PasswordStore::getPassword` NatSpec indicate a non-existent parameter, causing the natspec to be incorrect.

Description:

```
    /*
     * @notice This allows only the owner to retrieve the password.
@>   * @param newPassword The new password to set.
     */
    //@audit there's no newPassword parameter(documentation error)
    function getPassword() external view returns (string memory) {
```

The `PasswordStore::getPassword` function signature is `getPassword()`, but the NatSpec documentation suggests it should be `getPassword(string)`. This mismatch makes the NatSpec incorrect.

Impact: The NatSpec documentation is inaccurate, which may mislead developers and auditors.

Recommended Mitigation: Remove the incorrect NatSpec line.

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

Gas