Protocol Audit Report

Version 1.0

Bricks

January 15, 2026

Protocol Audit Report

Bricks

January 15, 2026

Prepared by: Bricks Lead Auditors: - Bricks

Table of Contents

- Table of Contents
- Protocol Summary
- Disclaimer
- Risk Classification
- Audit Details
  - Scope
  - Roles
- Executive Summary
  - Issues found
- Findings
  - High
    * [H-1] Passwords stored on-chain is public data no matter the storage visibility type
    * [H-2] `PasswordStore::setPassword` has no access control, a non-owner can set the password.
  - Informational
    * [I-1] `PasswordStore::getPassword` has a `newPassword` nonexistent parameter in its natspec, causing the natspec to be incorrect

Protocol Summary

The password store protocol is a simple contract that stores a password and allows users to retrieve it. Only the owner of the contract should be able to set the password and retrieve the password.

Disclaimer

The Bricks 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

The findings described in this document corresponds with the following commit hash:

```
7d55682ddc4301a7b13ae9413095feffd9924566
```

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

The PasswordStore audit commenced with a review of the available documentation to establish an understanding of the protocol's intended behavior and design assumptions. The Bricks team subsequently defined the audit scope by identifying the contracts relevant to the protocol's core functionality. A manual, line-by-line review of the in-scope contracts was then conducted to assess correctness, security properties, and adherence to best practices.

The audit identified the following issues: 1. High:- Sensitive data is stored on-chain, making it publicly accessible to any external actor. 2. High:- The `setPassword` function lacks access control, allowing unauthorized parties to modify the stored password. 3. Informational:- The NatSpec documentation for `getPassword` function does not accurately reflect the function signature.

These findings indicate that the protocol's current design relies on incorrect assumptions regarding data confidentiality and access control on public blockchains. Addressing the identified issues is strongly recommended prior to deployment.

Issues found

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

Findings

High

[H-1] Passwords stored on-chain is public data no matter the storage visibility type

Description: All data stored on-chain is public data and can be accessed by anyone on the blockchain. The `PasswordStor::s_password` variable is only private in the context of the contract code environment and can only be accessed by the owner using the `PasswordStore::getPassword` function. We show one of such method of reading public data off chain from the blockchain below

Impact: Anyone can read the password severly breaking the core functionality of the protocol

Proof of Concept: [Proof of code] The below test case shows how anyone can read storage directly from the blockchain and retrieve the password. This is a serious security issue as it is possible for anyone to retrieve the password.

1. Create a locally running blockchain

```
anvil
```

2. Deploy the contract

```
make deploy
```

3. Read the password from the local blockchain using cast comand

```
cast storage <CONTRACT_ADDRESS> <s_password_sorage_slot> --rpc-url http
    ://127.0.0.1:8545
```

You will get an output like this `0x6d7950617373776f726400000000000000000000000000000000000000`

4. Convert the output to a string

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

You will get an output of:

```
myPassword
```

Recommended Mitigation: Due to this the whole achitectural design of the protocol needs to be re-evaluated. one could consider encrypting the password off-chain and store the encrypted password on-chain. This would require the user to remember another password off-chain in order to decrypt the password. However, You would also want to remove the view function as you wouldn't the user to accidentally send a transaction with the password that decrypt your password.

[H-2] `PasswordStore::setPassword` has no access control, a non-owner can set the password.

Description: The `PasswordStore::setPassword` function is set tobe an `external` function, However in the function's natspec it explicitly states that `This function allows only the owner to set a new password.` But this is not the case.

```
function setPassword(string memory newPassword) external {
@>    // @audit this function is not restricted to being called by the
    owner
        s_password = newPassword;
        emit SetNetPassword();
    }
```

Impact: Anyone can set the password and overide the current password to a new password.

Proof of Concept: Add the folloeing test in the `PasswordStore.t.sol` file

code

```
function test_anyone_can_set_password(address randomAddress) public {
        vm.assume(randomAddress != owner);
        vm.startPrank(randomAddress);
        string memory newPassword = "hackedPassword";
        passwordStore.setPassword(newPassword);
        vm.stopPrank();

        vm.startPrank(owner);
        string memory actualPassword = passwordStore.getPassword();
        assertEq(actualPassword, newPassword);
    }
```

the above test passes and proofs that anyone can set the password.

Recommended Mitigation: Add an access control condition to the `PasswordStore::setPassword` function to ensure that only the owner can set the password.

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

Informational

[I-1] `PasswordStore::getPassword` has a `newPassword` nonexistent parameter in its natspec, 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) {}
```

the `PasswordStore::getPassword` function signature is `getPassword()` but the natspec says otherwise `getPassword(string)`

Impact: the natspec is incorrect

Recommended Mitigation:

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