Protocol Audit Report

Prepared by: Caducus

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

Caducus 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.
- Outsides: No one else should be able to set or read the password.

Executive Summary

Add some notes about how the audit went, types of things you found, etc.

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 visable 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 accesed 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, severly breaking the functionality of the protocol.

Proof of Concept: (Proof of Code) The below shows how anyone can read the password from the blockchain.

1. Create a local chain and deploy the contract.

```
make anvil
```

```
make deploy
```

2. Run the storage tool. We use storage slot 1, because that's where `s_password` is stored.

```
cast storage 0x5FbDB2315678afecb367f032d93F642f64180aa3 1 --rpc-url
http://127.0.0.1:8545
```

```
The output will look like this:
`0x6d7950617373776f726400000000000000000000000000000000000000000014`
```

3. You can then parse that hex to a string with :

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

4. The output will reveal the `s_password` variable. `myPassword`

Recommended Mitigation: Unfortunately the whole functionality of the contract is compromised to this, so the overall architecture needs to be rethought. The password could be encrypted off-chain and then the encrypted password could be stored on-chain. This would require the user to remember the key to decrypt it.

[H-2] `PasswordStore::setPassword` has no Access Control, meaning a non-owner could change it.

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) {

    }
```

Impact: Incorrect natspec.

Recommended Mitigation: Remove the incorrect natspec line.

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