Protocol Audit Report

Version 1.0

io10

December 18, 2024

Protocol Audit Report

io10

17/12/2024

Solo Auditor:

- io10

Table of Contents

Protocol Summary

PasswordStore is 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

io10 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 in this report correspond to the following commit hash:

```

7d55682ddc4301a7b13ae9413095feffd9924566
```

Repo URL

https://github.com/Cyfrin/3-passwordstore-audit

In scope vs out of scope contracts

```
./src/
PasswordStore.sol
```

Compatibilities

- Solc Version: 0.8.18

- Chain(s) to deploy contract to:
  - ETH
- Tokens: None

  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 smart contract was audited to evaluate its design, implementation, and security posture. The contract is designed to allow a user (the owner) to securely store and retrieve a password, ensuring no unauthorized access. This audit focused on identifying security vulnerabilities, adherence to Solidity best practices, and the overall functionality of the contract. The audit revealed several critical, high, and low-severity issues, along with opportunities for improvement in code structure and clarity. Key issues include insufficient access control, exposure of private state variables, and misleading documentation. The PasswordStore smart contract demonstrates fundamental functionality but has critical security flaws that must be addressed to meet its intended design goals. By implementing the recommended mitigations, the contract can significantly improve its security and reliability.

Issues found

Severity	Number Of Issues Found
High	2
Medium	0
Low	0
Informational	1
Total	3

Findings

High

[H-1] Access Control not set for `PasswordStore::setPassword` which allows anyone to call

`PasswordStore::setPassword` function

Description: `PasswordStore::setPassword` allows any user to call it due to the external Solidity tag attached to the function. External calls give functions this ability in solidity. As a result, any user can call `PasswordStore::setPassword` and pass in any string which will change the password for `PasswordStore::s_owner` to whatever password the arbitrary user has set.

Code

```
    function setPassword(string memory newPassword) external {
            //bug attacker can call this function and set a new
               password
            s_password = newPassword;
            emit SetNetPassword();
        }
```

Impact: The designated password for `PasswordStore::s_owner` can be changed by any user at any time.

Proof of Concept: (Proof of code)

1. Spin up a local anvil node:

```
anvil
```

2. Deploy `PasswordStore` contract on anvil node:

```
make deploy
```

3. Call `PasswordStore::setPassword` with a different account to the one that created the contract:

```
cast send PASSWORDSTORECONTRACTADDRESS "setPassword(string memory)
   " "enterpasswordstring" --rpc-url 127.0.0.1:8545 --private-key
   ENTERPRIVATEKEYOFALTERNATEANVILACCOUNT
```

4. Call `PasswordStore::getPassword` with the account that created the contract:

```
cast call PASSWORDSTORECONTRACTADDRESS "getPassword()"  --rpc-url
   127.0.0.1:8545 --private-key
   ENTERPRIVATEKEYOFCREATORANVILACCOUNT
```

5. Convert returned hex data to ascii to see string version of password:

```
cast to-ascii HEXDATARETURNEDFROMSTEP4
```

Recommended Mitigation: To remedy the access control bug, it is advised to import and inherit the `Ownable` contract from the openzeppelin contracts repository. This contract comes equipped with an onlyOwner modifier which can be applied to both `PasswordStore::setPassword` and `PasswordStore::getPassword` to prevent users who aren't owners of the contract from calling any of the functions in the contract.

[H-2] `PasswordStore::s_password` is visible to anyone on the blockchain which allows any user to retrieve the password for `PasswordStore::s_owner`

Description: `PasswordStore::s_password` is a state variable stored in storage slot 1 of the `PasswordStore` contract. Although the visibility of the variable is set to private, this means that it is not visible to other contracts but it can be retrieved directly from the blockchain by any user.

Impact: This exposes the password for `PasswordStore::s_owner` which is not intended functionality and breaks a major protocols invariants.

Proof of Concept: (Proof of Code)

1. Spin up a local anvil node:

```
anvil
```

2. Deploy `PasswordStore` contract on anvil node:

```
make deploy
```

3. Retrieve the hex data stored in storage slot 1 of `PasswordStore`:

```
cast storage PASSWORDSTORECONTRACTADDRESS 1 --rpc-url
    127.0.0.1:8545
```

4. Convert returned hex data to ascii to see string version of password:

```
cast to-ascii HEXDATARETURNEDFROMSTEP3
```

Recommended Mitigation: There are a lot of considerations that could remedy the issue. One such remedy would include hashing the password using a keccak256 hashing algorithm and redefining `PasswordStore::s_password` to a bytes data type. This will make it more difficult for users to view the passeord as only the hash will be visible in the storage slots. As a result of this, `PasswordStore::getPassword` will have to be refactored as when `PasswordStore::s_owner` calls the function, it would only return the hashed password which the user will not be able to decode. The architecture of the protocol will have to be redesigned to factor the security of the `PasswordStore::s_password` variable.

Informational

[I-1] Natspec contains non-existent parameter in `PasswordStore::getPassword` which can be misleading to developers

Description: Natspec for `PasswordStore::getPassword` contains the following:

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

This signals to developers that a parameter is to be passed to `PasswordStore::getPassword`. This proposed parameter is not necessary and also not included in the function in the in scope contract.

Impact: This produces code and documentation inconsistency which can lead to incorrect implementation.

Proof of Concept: See relevant natspec for `PasswordStore::getPassword`

Recommended Mitigation: Remove inconsistent natspec parameter related to `PasswordStore::getPassword`

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