PasswordStore Security Review

Version 1.0

Kelbels

January 27, 2025

PasswordStore Security Review

Kelbels

January 27, 2025

Prepared by: Kelbels

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

I, Kelbels, make all effort to find as many vulnerabilities in the code in the given time period, but hold no responsibilities for the findings provided in this document. A security audit by me 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

I use the CodeHawks severity matrix to determine severity. See the documentation for more details.

Audit Details

The findings described in this document correspond with 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

Example security audit for educational purposes.

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 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 accessed 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, severely breaking the functionality of the protocol.

Proof of Concept: The below test case shows how anyone can read a password directly from the blockchain.

1. Create a locally running chain

```
anvil
```

2. Deploy the contract to the chain

```
make deploy
```

3. Run the storage tool

We use 1 because that's the storage slot of `s_password` in the contract.

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

You'll get an output that looks like this:

`0x6d7950617373776f726400000000000000000000000000000000000000000014`

You can then parse that hex to a string with:

```
```
run 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 decrypts your password.

[H-2] `PasswordStore::setPassword` has no access controls, meaning a non-owner could change the password.

Informational

[I-1] `PasswordStore::getPassword` NatSpec indicates a parameter that doesn't exist, causing NatSpec to be incorrect.

Description:

```
    /*
```

```
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 `PasswordStore::getPassword` function signature is `getPassword()` while the NatSpec says it should be `getPassword(string)`.

Impact: The NatSpec is wrong.

Recommended Mitigation: Remove the incorrect NatSpec line.

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