PasswordStore Audit Report

Lead Auditors:

- Benas Volkovas

Table of Contents

▶ See table

Disclaimer

The 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 correspond the following commit hash:

```
2e8f81e263b3a9d18fab4fb5c46805ffc10a9990
```

Scope

src/ --- PasswordStore.sol

Protocol Summary

PasswordStore is a protocol dedicated to storage and retrieval of a user's passwords. 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.

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

Add some notes about how the audit went, types of things you found, etc. We spent X hours with Z auditors using Y tools. etc

Issues found

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

Findings

High

[H-1] Storing private variables on-chain makes them visable to anyone and not private

- Impact: HIGH
- Likelihood: HIGH
- Severity: HIGH

Description: All data stored on-chain is visable 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 password stored in the contrac, severly breaking the functionality of the protocol.

Proof of Concept: The below test case shows how anyone can read the password stored in the contract.

1. Create a locally running chain

```
make anvil
```

2. Deploy the contract

```
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://localhost:8545
```

You'll get an output that looks like this:
`0x6d7950617373776f726400000000000000000000000000000000000000000014`

4. Parse the hex to string

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

5. Get an output Expected output: `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::getPassword` has no access control, meaning a non-owner can change password

- Impact: HIGH
- Likelihood: HIGH
- Severity: HIGH

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

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

Impact: Anyone can change the password stored in the contract, severly breaking the functionality of the protocol.

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

▶ Code

Recommended Mitigation: Add an access control condition to the `setPassword()` function

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

Medium

- None

Low

- None

Informational

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

- Impact: NONE
- Likelihood: NONE
- Severity: Informational

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) {
    // code
}
```

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

Impact: The natspec is incorrect and could cause confusion for developers.

Recommended Mitigation: Remove the incorrect natspec line.

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

Gas

- None