Protocol Audit Report

Version 1.0

mPTYmem

March 6, 2024

Protocol Audit Report

mPTYmem

March 5, 2024

Prepared by: mPTYmem

Lead Auditors: mPTYmem

Table of Contents

Protocol Summary

A smart contract applicatoin 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

mPTYmem 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
Likelihood	High	H	H/M	M
	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 wiht the following commit hash

Commit Hash:

```
53ca9cb1808e58d3f14d5853aada6364177f6e53
```

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

This audit took mPTYmem 10 hours to complete. Foundry, tintinweb.solidity-metrics, tintinweb.solidity-visual-auditor, pandoc, and LaTeX were used to create this report.

Issues Found

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

Findings

High

[H-1] Variables stored on-chain are visible to anyone so non-owners can check password.

Description:

All data to 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 viaraible and only accessed through the `PasswordStore::getPassword` function, which is intended to only be called by the owner of the contract.

There is one such method of reading any data off chain shown below.

Impact: Anyone can read the private password, severely breaking the functionality of the protocol.

Proof of Concept:

The below test case can show how anyone can read data off chain.

Test case

1. Create a locally running chain

```
make anvil
```

2. Deploy the contract to the chain

```
make deploy
```

3. Run the storage tool.

We use 1 because this is the storage slot that correlates to `PasswordStore::s_password`.

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

You'll get an output of `0x6d7950617373776f726400000000000000000000000000000000000000000014`

.

4. You can parse that hex to a string with

```
cast
parse-bytes32-string 0x6d7950617373776f72640000000000
0000000000000000000000000000000000000014
```

and you'll 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 likely want to remove the view function as you wouldn't want the user to accientally send a transaction with the password that decrypts your password.

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

Description:

The `PasswordStore::setPassword` method should check that caller is owner, so that non-owners cannot set password. This check is missing, so anyone can set password. This violates the expectation that

> Others should not be able to access the password

https://github.com/Cyfrin/3-passwordstore-audit/blob/53ca9cb1808e58d3f14d5853aada6364177f6e53/src/PasswordS
L29C6

```
function setPassword(string memory newPassword) external {
    // @audit no access controls
    s_password = newPassword;
    emit SetNetPassword();
}
```

There is one such method shown below.

Impact:

The resource at s_owner can lose their recorded password and in turn could lose access to the resource it protects. This severely breaks the contracts intended functionality

Proof of Concept:

The below test case can show how anyone can have their password overwriten by an attacker. This test case should be added to `PasswordStore.t.sol`.

Code

```
function test_review_non_owner_can_set_password(address attacker)
    public {
    vm.assume(attacker != owner);
    vm.prank(attacker);
    string memory expectedPassword = "attackPassword";
    passwordStore.setPassword(expectedPassword);

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

Recommended Mitigation:

The following code should be added to `PasswordStore.sol`.

Code

```
function setPassword(string memory newPassword) external {
+   if (msg.sender != s_owner) {
+       revert PasswordStore__NotOwner();
+   }
    s_password = newPassword;
    emit SetNetPassword();
}
```

Informational

[I-1] The natspec for `PasswordStore::getPassword` mentions a missing parameter, causing confusion

Description:

The natspec for `PasswordStore::getPassword` function mentions a `newPassword` parameter, but it does not exist in the function's implementation.

Code

```
/*
  * @notice This allows only the owner to retrieve the password.
  * @param newPassword The new password to set.
  */
function getPassword() external view returns (string memory) {
    if (msg.sender != s_owner) {
        revert PasswordStore__NotOwner();
    }
    return s_password;
}
```

Impact:

The natspec is incorrect.

Recommended Mitigation:

Remove the following line from `PasswordStore::getPassword`.

Code

```
/*
  * @notice This allows only the owner to retrieve the password.
- * @param newPassword The new password to set.
  */
function getPassword() external view returns (string memory) {
    if (msg.sender != s_owner) {
        revert PasswordStore__NotOwner();
    }
    return s_password;
}
```