Smart Contract Security

Protocol Audit Report

Version 1.0

Phylax

August 13, 2024

Protocol Audit Report

Phylax

August 13, 2024

Prepared by: Phylax

Table of Contents

Protocol Summary

PasswordStore protocol is dedicated for storage and retrieval of the user's password. The protocol is designed to be used by a single user where only the owner can store and retrieve password.

Disclaimer

Phylax 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 Phylax 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: 7d55682ddc4301a7b13ae9413095feffd9924566

Scope

- In Scope:

```
src
#-- PasswordStore.sol
```

Roles

- Owner: The user that can set the password and read the password.
- Outsiders: No one else should be able to set or read password.

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

We show one such method of reading any data off chain below.

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

Proof of Concept: (Proof of code)

The below test case shows how anybody can read the password directly from the blockchain.

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 the storage slot for `s_password` is the second storage slot.

```
cast storage <ADDRESS_HERE> 1
```

You will get an output like this:

`0x6d7950617373776f726400000000000000000000000000000000000000000014`

You can parse the hex to a string like this:

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

And the output will be:

`myPassword`

Recommended Mitigation: Due to this, the overall architecture of the contract should be rethought. One could encrypt the password off-chain before storing it on-chain, but this requires that you would need to remember the password for decryption off-chain. The view function should likely be removed because the password could be exposed by making a transaction by mistake.

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

Description: The `PasswordStore::setPassword` function is set to be `external`, however, the natspec of the function and the 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 are no access controls
        s_password = newPassword;
        emit SetNetPassword();
    }
```

Impact: Anyone can set/change the password of the contract, severly breaking the contract intended functionality.

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

Code

```
function test_anyone_can_set_password(address randomAddress) public {
        vm.assume(randomAddress != owner);
        vm.prank(randomAddress);
        string memory expectedPassword = "myNewPassword";
        passwordStore.setPassword(expectedPassword);

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

Recommended Mitigation: Add an access control conditional to the `setPassword` function.

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

Informational

[I-1] The `PasswordStore::getPassword` natspec indicates a parameter that does not exist, cousing natspec to be incorrect

Description: In the natspec for the `getPassword` it says `@param newPassword The new password to set.`, but the function has no parameter.

```
/*
     * @notice This allows only the owner to retrieve the password.
     * @param newPassword The new password to set.
     */
@>   function getPassword() external view returns (string memory) {
```

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

Impact: This makes the natspec incorrect.

Recommended Mitigation: Remove the incorrect natspec line:

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