Protocol Audit Report

Version 1.0

0xStriker

March 28, 2024

Protocol Audit Report

0xStriker

March 26, 2024

Prepared by: 0xStriker Lead Auditor: - 0xStriker

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

The 0xStriker 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

commit hash

```
2e8f81e263b3a9d18fab4fb5c46805ffc10a9990
```

Scope

In scope

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

Roles

Owner: Is the only one who should be able to set and access the password. # Executive Summary The contrants base archcitcture is wrong and should be altered entirely

Issues found

Severity	Number of ssues found
High	2
Medium	0
Low	0
Info	1
Gas	3

Findings

High

[H-1] Any variable that is stored on chain is not private and can be accessed by anyone

Description: All the data stored on chain is public and can be accessed by anyone. The `PasswordStore::s_password` variable is inteded to be a private variable and can only be accessed through the `PasswordStore::getPassword` function which can only be called by the owner. However, anyone can direclty read this using any number of off chain methodologies

Impact: Anyone can read the private variables, breaking the inteded functionality of the contract.

Proof of Concept: The below test case shows how anyone could read the password directly from the blockchain. We use foundry's cast tool to read directly from the storage of the contract, without being the owner.

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 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:

```
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` can be called by anyone

Description: The `PasswordStore::setPassword` function is set to be an external function, however the natspec and the overall behaviour of the contract is that this function should only be called by the owner of the contract.

```
function setPassword(string memory newPassword) external {
@>      @audit - there are no access control here.
        s_password = newPassword;
        emit SetNetPassword();
    }
```

Impact: Anyone can the set the password while it should only be the owner who do so.

Proof of Concept: The below test could be added to the PasswordStore.t.sol to show the behaviour.

Code

```
        function test_Is_setPassword_callable_by_anyone(address
            randomAdd) public {
            vm.assume(randomAdd != owner);
            vm.prank(randomAdd);
            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(msg.sender != s_owner){
   revert PasswordStor__NotOwner();
}
```

Informational

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

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)
```

The natspec for the function PasswordStore::getPassword indicates it should have a parameter with the signature getPassword(string). However, the actual function signature is getPassword().

Impact: The natspec is incorrect.

Recommended Mitigation: Remove the incorrect natspec line.

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