Protocol Audit Report

Version 1.0

Predator

October 24, 2024

Protocol Audit Report

Predator

October 24, 2024

Prepared by: [Predator] Lead Security Researcher: - Predator

Table of Contents

Protocol Summary

PasswordStore is a protocol dedicated to storage and retrieval of a user's password. The protocol is designed to to be used by a single user, and it is not desinged to be used by multiple users.Only owner should be able store a password and then access it later.

Disclaimer

The PREDATOR 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
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 are described in this document corresponf the following commit:

```
- Commit Hash:  2e8f81e263b3a9d18fab4fb5c46805ffc10a9990
```

Scope

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

Roles

Owner: The user who can set the password and read the password. Outsides: No one else should be able to set or read the password.

Executive Summary

We spent around 1 hour with team to identify all vulnerabilities.

Issues found

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 a password on-chain makes it visible to anyone, and no longer private

Description: All data stored on-chain is vivible to anyone, can be read directly from blockchain. The `PasswordStore::s_password` in intended to be private variable, and to get accessed only through the `PasswordStore::getPassword` function, which is intended to be called only by owner of the contract.

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

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

Proof of Concepts (Proof of Code)

The below test how anyone can read the password directly from the blockchain.

1. Start a local node

```
make anvil
```

2. Deploy the contract to chain

This will default to your local node. You need to have it running in another terminal in order for it to deploy.

```
make deploy
```

3. Run the storage tool

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

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

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

You can parse that hex to a string with

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

And you get an output

```
myPassword
```

Recommended mitigation Due to this, the overall architecture of the contract should be rethought. One could encrypt password off-chain and store encrypted password on-chain. This would require user to remember another password off-chain to inder to decrypt the password. However, you'd also likely want to remove view function, cause u don't want that user accidently send a transaction with a password that will decrypt your password.

Impact: High Likelyhood: High Severity: High

[H-2] `PasswordStore::setPassword` has no access control, it means anyone can set a password

Description The `PasswordStore::setPassword` function is set to be `external`, 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 controls
        s_password = newPassword;
        emit SetNetPassword(); // i name convention, better to use
            class name __ b4 event name
    }
```

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

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

Code

```
    function test_anyone_can_change_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 You need to add access control to the function `PasswordStore::setPassword`

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

Impact: High Likelyhood: High Severity: High

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

Impact The natspec is incorrect

Recommended mitigation Remove the incorrect naspec line

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