Protocol Audit Report

Version 1.0

Cyfrin.io

December 27, 2023

PasswordStore Audit Report

0xd403

Dec 27, 2023

Password Store Audit Report

Prepared by: 0xd403

Lead Auditors:

- 0xd403

Assisting Auditors:

- None

Table of Contents

- Findings
- High
  - [HIGH-1] Passwords stored on-chain are visable to anyone, not matter solidity variable visibility
  - [HIGH-2] Missing access control on `PasswordStore::setPassword()`, meaning a non-owner could change the password
- Informational
  - [Informational-1] Incorrect natspet for `PasswordStore::getPassword()`, it indicates a parameter that doesn't exists

Disclaimer

The 0xd403 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.

Protocol Summary

PasswordStore is a protocol designed for a single user to store his password privately on-chain. Only the owner is able to store and later read the password from the contract. This protocol isn´t designed

to be multi-user.

Audit Details

Repository audited: https://github.com/Cyfrin/3-passwordstore-audit/tree/onboarded

All the findings described in this document correspond to the following commit hash:

```
2e8f81e263b3a9d18fab4fb5c46805ffc10a9990
```

Scope

```
src/
--- PasswordStore.sol    SOLC: 20
```

Roles

- Owner: is the only one who should be able to set/read the password

Only the owner should be able to interact with this contract.

Executive Summary

Issues found

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

Findings

High

[HIGH-1] Passwords stored on-chain are visable to anyone, not matter solidity variable visibility

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

However, anyone can direclty read this using any number of off-chain methodologies

Impact: The password is not private, severely breaking the contract's main purpose.

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.

[HIGH-2] Missing access control on `PasswordStore::setPassword()`, meaning a non-owner could change the password

Description: By reading the natspect, `PasswordStore::setPassword()` should be callable only be the owner, but this isn't checked at all in the code.

```
1 function setPassword(string memory newPassword) external {
2     //@ audit missing access control, anyone can set the password
3     s_password = newPassword;
4     emit SetNetPassword();
5 }
```

Impact: Anyone can set/change the password, severely breaking the contract's purpose.

Proof of Concept: Add this fuzz test to `test/PasswordStore.t.sol`.

```
1  function test_non_owner_can_set_passwordFuzz(address randomAddress)
       public {
2      vm.assume(randomAddress != owner);
3
4      vm.startPrank(randomAddress);
5      string memory newPwd = "I'm not the owner!";
6      passwordStore.setPassword(newPwd);
7
8      vm.startPrank(owner);
9      string memory actualPwd = passwordStore.getPassword();
10     assertEq(newPwd, actualPwd);
11 }
```

Reccomended Mitigations: Add an access control condition to `PasswordStore::setPassword()`.

```
1 function setPassword(string memory newPassword) external {
2 +     if (msg.sender != s_owner) {
3 +         revert PasswordStore__NotOwner();
4 +     }
5     s_password = newPassword;
6     emit SetNetPassword();
7 }
```

Informational

[Informational-1] Incorrect natspet for `PasswordStore::getPassword()`, it indicates a parameter that doesn't exists

Reccomended Mitigations: Remove the incorrect natspec line showed below

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