Protocol Audit Report

Version 1.0

Cyfrin.io

April 26, 2025

Protocol Audit Report

EmptySet

25 April 2025

Prepared by: [EmptySet] Lead Auditors: - EmptySet

Table of Contents

Protocol Summary

The protocol stores password for the contract owner and the owner can retrieve the password anytime. Others cannot read or write the password.

Disclaimer

The EmptySet 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: 7d55682ddc4301a7b13ae9413095feffd9924566

Scope

```
1  ./src/
2  #-- 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 is from a cyfrin updraft course and I spent 2 days finishing this material. We found vulnerabilities using cast and tests.

Issues found

Severity	Number of Issues Found
H	2
M	0
L	0
Informational	1
Total	3

Findings

High

[H-1] Storing a data on-chain makes it visible to anyone

Description: All data stored on-chain is public to anyone. The `PasswordStore::s_password` variable is intended to be hidden and only accessible by the owner through the `PasswordStore::getPassword` function.

Impact: Anyone is able to read the private password, severely breaking the protocol functionality.

Proof of Concept: The below test shows how anyone could read the password directly from the blockchain. We used foundry's cast tool to read directly from the storage of the contract, without being the owner. Create a locally running chain

```
1  make anvil
```

Deploy the contract to the chain

```
1  make deploy
```

Run the storage tool

We use 1 because that's the storage slot of s_password in the contract.

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

You'll get an output that looks like this:

```
1  0x6d7950617373776f726400000000000000000000000000000000000000000014
```

You can then parse that hex to a string with:

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

And get an output of:

```
1  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 stored password. However, you're also likely want to remove the view function as you wouldn't want the user to accidentally send a transaction with this decryption key.

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

Description: The `PasswordStore::setPassword` function is set to be an external function, however the purpose of the smart contract and function's natspec indicate that `This function allows only the owner to set a new password`.

```
1  function setPassword(string memory newPassword) external {
2  @>   // @Audit - There are no Access Controls.
3      s_password = newPassword;
4      emit SetNewPassword();
5  }
```

Impact: Anyone can set/change the stored password, severely breaking the contract's intended functionality

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

```
1  function test_anyone_can_set_password(address randomAddress) public {
2      vm.assume(randomAddress != owner);
3      vm.startPrank(randomAddress);
4      string memory expectedPassword = "myNewPassword";
5      passwordStore.setPassword(expectedPassword);
6
7      vm.startPrank(owner);
8      string memory actualPassword = passwordStore.getPassword();
9      assertEq(actualPassword, expectedPassword);
10     }
```

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

```
1  if(msg.sender != s_owner){
2      revert PasswordStore__NotOwner();
3  }
```

Medium

Low

Informational

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

Description:

```
1      /*
2       * @notice This allows only the owner to retrieve the password.
3  @>   * @param newPassword The new password to set.
4       */
5      function getPassword() external view returns (string memory) {}
```

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

Impact: The natspec is incorrect

Recommended Mitigation: Remove the incorrect natspec line.

```
1      /*
2       * @notice This allows only the owner to retrieve the password.
3  -    * @param newPassword The new password to set.
```

```
4        */
```

Gas