Protocol Audit Report

Version 1.0

Cyfrin.io

November 23, 2025

Protocol Audit Report

Cyfrin.io

March 7, 2023

Prepared by: Cyfrin Lead Auditors: - xxxxxxx

Table of Contents

Protocol Summary

Protocol does X, Y, Z

Disclaimer

The YOUR_NAME_HERE 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

The findings described in this document correspond to the following hash:

```
2e8f81e263b3a9d18fab4fb5c46805ffc10a9990
```

Scope

```
./src/
#-- 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

Add some notes about how the audit went, types of things you found, etc.

We spent X hours with Z auditors using Y tools, etc

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, and no longer private

Description: All data stored on chain is public and visible 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 functionality of the protocol.

Proof of Concept: (Proof of Code)

The below test case shows anyone can read the password directly from the blockchain

1. Create a locally running chain "bash 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 1 –rpc-url http://127.0.0.1:8545

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

You can then parse then hex string with:
```

cast parse-bytes32-string 0x6d7950617373776f726400000000000000000000000000000000000000000014

```

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 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.`

'''
function setPassword(string memory newPassword) external {
    // @Audit - There are no Access Controls.
    s_password = newPassword;
    emit SetNewPassword();
}
'''

**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:

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

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

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

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


## Informational

**Title:** [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 natspec line.

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