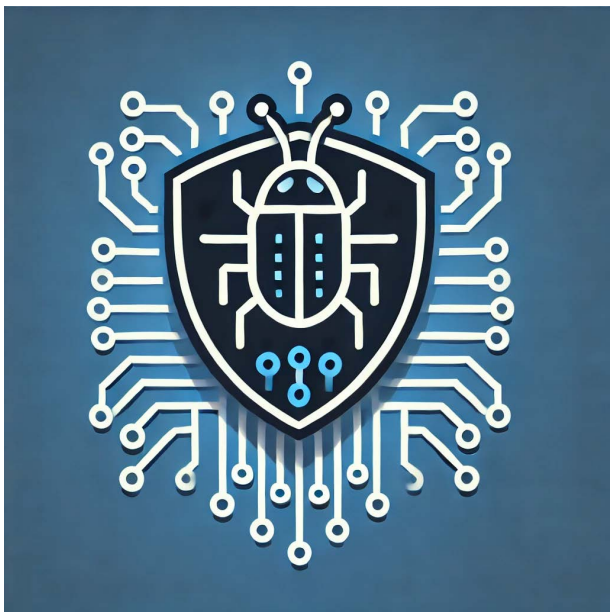

Protocol Audit Report

Version 1.0

SnakeSec

October 14, 2024

Protocol Audit Report

Cyfrin.io

March 7, 2023

Prepared by: Ivan Lead Researchers: - Ivan Kartunov

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
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

Scope

```
./src/PasswordStore.sol
```

Roles

Executive Summary

Issues found

Findings

High

[H-1] Storing the password on-chain makes it visible to anyone, and no more private

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.

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

Proof of Concept: (Proof of Code)

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

1. Create locally running chain
2. Deploy contract on the chain
3. Run the storage tool

Recommended Mitigation:

[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 natspec of the function and overall purpose of the smart contract is 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 SetNetPassword();
5      }
```

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

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

Code

```
    function test_anyone_can_set_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: Add an access control conditional to the `setPassword` function.

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

Informational

[I-1] `PasswordStore::getPassword` natspec indicates a parameter that does not 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()` which naspec say it should be `getPassword(string)`.

Impact: The natspec is incorrect

Recommended Mitigation: Remove the incorrect natspec line.

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

Gas